Escola de Engenharia

Universidade do Minho – Departamento de Informática
Licenciatura em Engenharia Informática (LEI)

3º ano – 1º semestre
Ano Letivo 2022/2023

RELATÓRIO – Trabalho Prático
Entrega intermédia 2

Desenvolvimento de Sistemas de Software (DSS)

F1 Manager: Simulador de campeonatos de automobilismo

Grupo 27

A97540, Ana Rita Moreira Vaz
A92847, Guilherme Sousa Silva Martins
A97777, Millena de Freitas Santos
A96794, Ricardo Alves Oliveira

Link repositório GitHub: https://github.com/GuiSSMartins/DSS_Grupo27_2022_23




Conteúdo

1 Introdução **3**

2 Objetivos **4**

3 Terminologia específica **5**
3.1 Cilindrada . . . 5
3.2 Chicane . . . 5
3.3 Classe 1 . . . 5
3.4 Classe 2 . . . 5
3.5 Classe GT . . . 5
3.6 Classe SC . . . 5
3.7 CTS . . . 5
3.8 GDU . . . 5
3.9 PAC . . . 5
3.10 SVA . . . 5

4 Revisões efetuadas **6**

5 Processamento dos Use Cases **8**

6 Reaproveitamento de código **13**
6.1 Aposta e Equipa . . . 13
6.2 JogadorComparatorNome e TimeConverter . . . 13
6.3 CarteiradeJogadores . . . 13
6.4 Campeonato . . . 13
6.5 Piloto . . . 13
6.6 Record . . . 14
6.7 Circuito . . . 14
6.8 Carro . . . 14

7 Diagrama de Componentes **15**

8 Diagrama de Classes e de *Packages* **16**
8.1 *Package* Utilizadores . . . 16
8.2 *Package* Catálogo . . . 17
8.3 *Package* Partidas . . . 17

9 Diagramas de Atividade **18**
9.1 Simulação . . . 18
9.2 Calcular Novo Evento . . . 19
9.2.1 Versão Base . . . 20

10 Diagramas de Sequência **21**
10.1 verificaDisponibilidadeAfinações . . . 21
10.2 entrarNaPartida . . . 22
10.3 aplicarNovaAfinacao . . . 22
10.4 registaConfiguracaoCorrida . . . 23
10.5 registarConfiguracao . . . 23
10.6 calculaEventos . . . 24
10.7 registarCampeonato . . . 24
10.8 registarCircuito . . . 25
10.9 registarCarro . . . 25
10.10registarPiloto . . . 26

11 Conclusão e Análise dos resultados obtidos **27**

Lista de Figuras

1 Revisão efetuada: Diagrama de Use Cases . . . 6
2 Revisão efetuada: Registar Utilizador . . . 6
3 Revisão efetuada: Mudar versão de jogo do Jogador . . . 7
4 Revisão efetuada: Mudar versão de jogo do Jogador . . . 7
5 Modelo de Domínio . . . 8
6 Processamento do Use Case: Adicionar Utilizador . . . 9
7 Processamento do Use Case: Registar Jogador . . . 9
8 Processamento do Use Case: Recuperar Jogador . . . 9
9 Processamento do Use Case: Registar Administrador . . . 9
10 Processamento do Use Case: Adicionar Campeonato . . . 9
11 Processamento do Use Case: Adicionar Circuito . . . 10
12 Processamento do Use Case: Adicionar C1 . . . 10
13 Processamento do Use Case: Adicionar C2 . . . 10
14 Processamento do Use Case: Adicionar GT . . . 10
15 Processamento do Use Case: Adicionar SC . . . 11
16 Processamento do Use Case: Adicionar Piloto . . . 11
17 Processamento do Use Case: Configurar Campeonato . . . 11
18 Processamento do Use Case: Configurar Campeonato . . . 11
19 Processamento do Use Case: Entrar em Campeonato . . . 11
20 Processamento do Use Case: Configurar Corridas . . . 12
21 Processamento do Use Case: Simular Corrida . . . 12
22 Processamento do Use Case: Finalizar Campeonato . . . 12
23 Processamento do Use Case: Mudar Versão do Jogo . . . 12
24 Modelo proposto para os carros . . . 14
25 1ª Versão do Diagrama de Componentes . . . 15
26 Diagrama de Componentes . . . 15
27 Diagrama de Packages geral . . . 16
28 Diagrama de Classe dos Utilizadores . . . 16
29 Diagrama de Classe do Catálogo . . . 17
30 Diagrama de Classe das Partidas . . . 17
31 Diagrama de Atividade do processo de simulação . . . 18
32 Diagrama de Atividade do processo de simulação (Versão Premium) . . . 19
33 Diagrama de Atividade do processo de simulação (Versão Base) . . . 20
34 Diagrama de Sequência do método "verificarDisponibilidadeAfinacoes" . . . 21
35 Diagrama de Sequência do método "entrarNaPartida" . . . 22
36 Diagrama de Sequência do método "aplicarNovaAfinacao.png" . . . 22
37 Diagrama de Sequência do método "registaConfiguracaoCorrida.png" . . . 23
38 Diagrama de Sequência do método "registarConfiguracao" . . . 23
39 Diagrama de Sequência do método "calculaEventos" . . . 24
40 Diagrama de Sequência do método "registarCampeonato" . . . 24
41 Diagrama de Sequência do método "registarCircuito" . . . 25
42 Diagrama de Sequência do método "registarCarro" . . . 25
43 Diagrama de Sequência do método "registarPiloto" . . . 26

1 Introdução

Este relatório tem como objetivo apresentar uma visão aprofundada do trabalho prático e do processo de desenvolvimento seguido, a fim de alcançar um resultado final robusto, coerente, válido e confiável. Nele está descrita a terminologia específica do sistema, bem como, os diagramas que compõem a 2ª fase do projeto: Conceção da solução.

Utilizou-se a UML, *Linguagem de Modelagem Unificada*, a fim de elaborar os diagramas que compõem este projeto, nomeadamente: diagrama de componentes, de sequência, de atividade e de packages. Outra ferramenta utilizada foi o Microsoft Excel para especificar os `Use Cases` e os seus respetivos Quadros de Processamento de Use Cases.

2 Objetivos

Para a segunda fase do trabalho prático, foi proposto a conceção da solução do sistema, a partir dos requisitos criados na primeira fase do projeto. Assim, neste relatório estão definidos:

- A arquitetura conceptual do sistema, capaz de suportar os requisitos identificados na primeira fase do projeto;
- Construção e descrição dos modelos comportamentais necessários para descrever o comportamento pretendido para o sistema.

3 Terminologia específica

Os termos seguintes são típicos do "mundo"do Automobilismo. Entender-los pode facilitar na interpretação dos modelos e do sistema em si.

3.1 Cilindrada

Capacidade de um motor de combustão de queimar a soma de combustível e ar nos seus pistões.

3.2 Chicane

Desvio artificial de um circuito, a fim de diminuir a velocidade de quem por ele passa, como medida de segurança.

3.3 Classe 1

Os carros de Classe 1 são protótipos feitos especialmente para o campeonato e apresentam uma cilindrada de 6000cm3.

3.4 Classe 2

Os carros de Classe 2 são veículos de alta performance que podem apresentar cilindradas entre 3000cm3 e 5000cm3.

3.5 Classe GT

Os carros de Classe GT (Grande Turismo) são carros desportivos produzidos em massa que podem apresentar uma cilindrada que varie entre 2000cm3 e 4000cm3.

3.6 Classe SC

Os carros da Classe SC (Stock Cars), são derivados de automóveis quotidianos apresentando, assim, uma cilindrada de 2500 cm3.

3.7 CTS

O CTS, "Chuva vs. Tempo Seco", é um critério que varia de 0 a 1 e indica o nível de perícia de um piloto em relação às condições atmosféricas. Um valor mais baixo indica um melhor desempenho em tempo chuvoso, já o contrário demonstra um melhor desempenho em tempo seco.

3.8 GDU

O GDU, ou Grau de Dificuldade de Ultrapassagem, indica, para cada segmento de um circuito, o quão difícil seria realizar uma ultrapassagem (não tendo em conta diferenças entre carros). Este pode tomar três valores: Possível, Difícil e Impossível.

3.9 PAC

O PAC, ou Perfil Aerodinâmico do Carro, é uma métrica que varia entre 0 e 1, onde um valor próximo de 0 indica um carro mais rápido, enquanto um valor mais próximo de 1 indica um carro mais estável.

3.10 SVA

O SVA, "Segurança vs. Agressividade", é um critério que varia de 0 a 1 e indica a tendência de um piloto em cometer decisões mais arriscadas ao longo das corridas. Isto significando que um piloto com um maior nível de SVA é mais propicio a arriscar uma ultrapassagem perigosa que poderá resultar em despiste.

4 Revisões efetuadas

Nesta secção serão abordadas as alterações efetuadas durante a segunda parte do trabalho, após a revisão da entrega anterior.

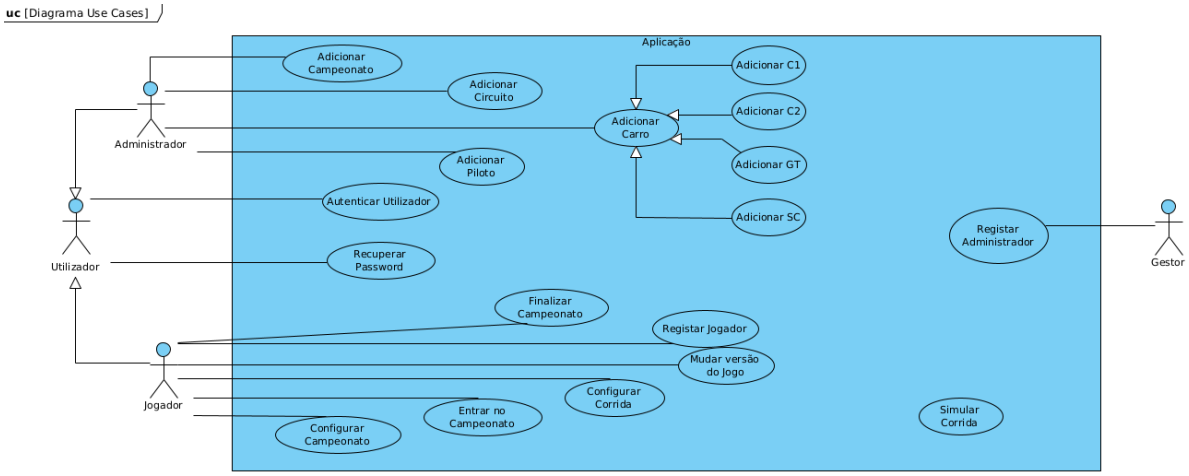

Figura 1: Revisão efetuada: Diagrama de Use Cases

No **Use Case - Registar Jogador**, implementado na primeira fase, não foi considerada a versão de simulação que o jogador gostaria de obter ao criar a conta. Portanto, visto ser um ponto de grande importância para a realização da aplicação, este use case foi revisto e esta questão passou a ser considerada e devidamente abordada, como se pode observar na Figura 1.

Use Case: Registar Jogador		
Descrição: Jogador efetua seu registo na Aplicação		
Ator: Jogador		
Pré-Condição: Nenhum Jogador autenticado		
Pós-Condição: Jogador fica registado		
	Ator	**Aplicação**
Fluxo Normal	1. Jogador indica e-mail	
		2. Aplicação valida o e-mail
	3. Jogador indica password e nome	
	4. Jogador indica a versão de jogo desejada	
		5. Aplicação regista o Utilizador
Fluxo Excepção 1 [e-mail já existe] (passo 2)		2.1 Aplicação informa que o e-mail já existe
		2.2 Aplicação cancela a operação
Fluxo Excepção 2 [username já existe] (passo 4)		4.1 Aplicação informa que o username já existe
		4.2 Aplicação cancela a operação

Figura 2: Revisão efetuada: Registar Utilizador

Da mesma forma que o Jogador pode escolher a versão que adquire ao registar a sua conta, também deve ser possível mudar a sua versão para a outra disponível. Desta forma, foi criado um novo Use Case designado Mudar versão do jogo, no qual o Jogador solicita a mudança e a aplicação muda para a versão contrária a que o Jogador possuir. As alterações podem ser observadas no ficheiro **useCases.vpp**. Isto é possível pois há apenas duas versões disponíveis: base e premium. Portanto não achamos necessário abordar as duas separadamente no use case, e considerar que, por exemplo, a base seja executada no fluxo normal e a premium no fluxo alternativo. Consideramos que essa opção seria de certa forma redundante, e é perfeitamente percetível a lógica com este use case mais simples e conciso apresentado na Figura 2.

Use Case: Mudar versão do jogo		
Descrição: Jogador muda sua versão de jogo		
Ator: Jogador		
Pré-Condição: Jogador está autenticado		
Pós-Condição: Jogador tem sua versão de jogo modificada		
	Ator	**Aplicação**
Fluxo Normal	1. Jogador indica que quer mudar de versão de jogo	
		2. Aplicação modifica a versão de jogo do Jogador

Figura 3: Revisão efetuada: Mudar versão de jogo do Jogador

Use Case: Mudar versão do jogo		
Descrição: Jogador muda sua versão de jogo		
Ator: Jogador		
Pré-Condição: Jogador está autenticado		
Pós-Condição: Jogador tem sua versão de jogo modificada		
	Ator	**Aplicação**
Fluxo Normal	1. Jogador indica que quer mudar de versão de jogo	
		2. Aplicação modifica a versão de jogo do Jogador

Figura 4: Revisão efetuada: Mudar versão de jogo do Jogador

5 Processamento dos Use Cases

Para facilitar a leitura e pensamento, o Modelo de Domínio foi separado por cores, cada a representar um conjunto de entidades que consideramos que fazem sentido pertencerem ao mesmo subsistema.

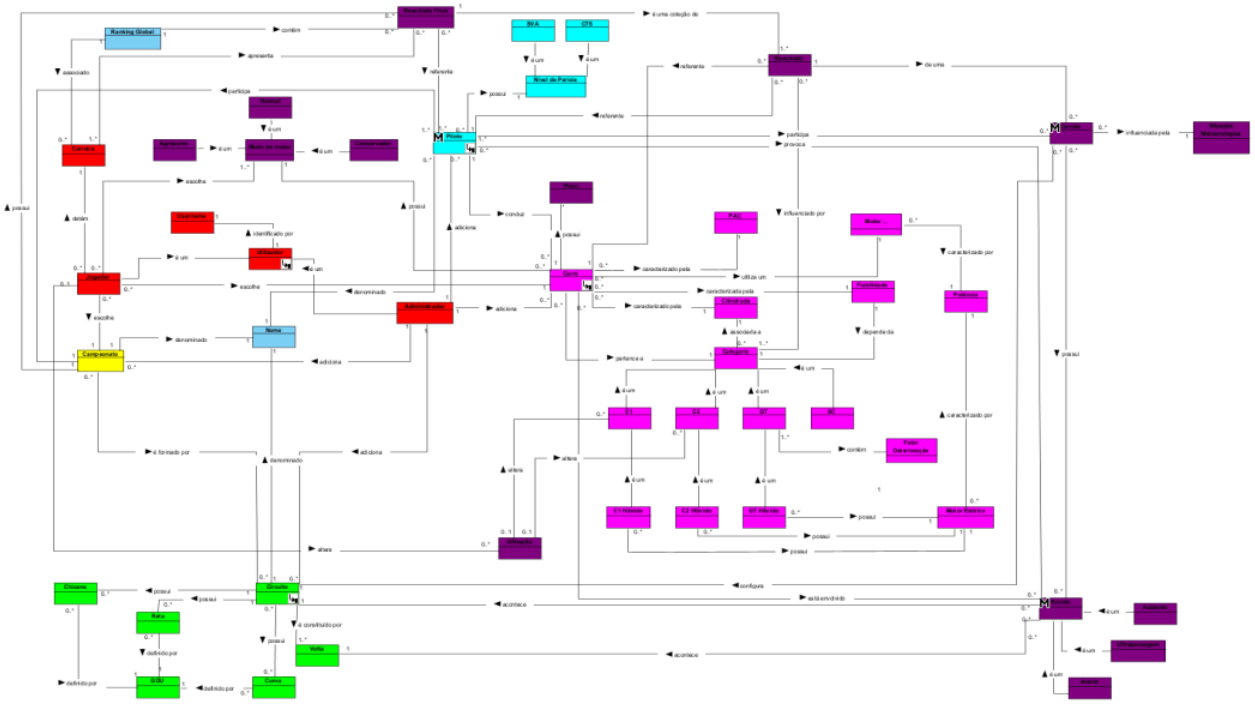

Figura 5: Modelo de Domínio

Para a 2ª fase do projeto, começamos por identificar, a partir dos Use Cases definidos na primeiro relatório do trabalho prático, as responsabilidades, as APIs, bem como os Subsistemas onde estas mesmas APIs pertencem. As alterações podem ser observadas no ficheiro **FaseII_DSS_Parte1.vpp** no **Modelo_Dominio_FaseII**.

Chegamos à conclusão que seriam necessários, no total, 3 SubSistemas: subUtilizadores, subCatálogo e subPartidas, que serão abordados em pormenor mais à frente neste relatório.

A seguir é apresentado um conjunto de tabelas com os campos indicados, que abordam todos os Use Cases mencionados anteriormente.

Autenticar Utilizador	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Utilizador indica login e password			
2. Aplicação valida o acesso	Verificar que utilizador existe e que a password está correta	validaAcesso(login: String, password: String) : boolean	subUtilizadores
Fluxo Excepção 1 [credenciais inválidas] (passo 2)			
2.1 Aplicação avisa sobre credenciais erradas			
2.2 Aplicação cancela operação			

Figura 6: Processamento do Use Case: Adicionar Utilizador

Registar Jogador	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Jogador indica e-mail			
2. Aplicação valida o e-mail	Verificar que o e-mail é válido	validaEmail(email: String) : boolean	subUtilizadores
3. Jogador indica password e nome			
4. Jogador indica a versão de jogo desejada			
5. Aplicação regista o Utilizador	Registar o novo utilizador como jogador	registarJogador(email: String, nome: String, password: String, versaoJogo: String): boolean	subUtilizadores
Fluxo Excepção 1 [e-mail já existe] (passo 2)			
2.1 Aplicação informa que o e-mail já existe			
2.2 Aplicação cancela a operação			

Figura 7: Processamento do Use Case: Registar Jogador

Recuperar Password	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Utilizador indica e-mail			
2. Aplicação valida o e-mail	Verificar que o e-mail é válido	validaEmail(email: String) : boolean	subUtilizadores
3. Utilizador indica código de recuperação			
4. Aplicação valida o código de recuperação	Verificar que o código de recuperação é válido	validaCodigo(email: String, codRecuperacao: String) : boolean	subUtilizadores
5. Utilizador fornece a nova password			
6. Aplicação regista a nova password	Registar a nova password	registaPassword(email: String, password: String): boolean	subUtilizadores
Fluxo Excepção 1 [e-mail não existe] (passo 2)			
2.1 Aplicação informa que o e-mail não existe			
2.2 Aplicação cancela a operação			
Fluxo Excepção 2 [código de recuperação inválido)] (passo 4)			
4.1 Aplicação informa que código de recuperação é inválido			
4.2 Aplicação cancela a operação			

Figura 8: Processamento do Use Case: Recuperar Jogador

Registar Administrador	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Gestor indica e-mail e password			
2. Aplicação verifica que o e-mail é válido	Verificar que o e-mail é válido	validaEmail(email: String) : boolean	subUtilizadores
3. Aplicação regista o Administrador	Registar o novo utilizador como administrador	registarAdministrador(email: String, password: String): boolean	subUtilizadores
Fluxo Excepção 1 [e-mail já existe] (passo 2)			
2.1 Aplicação informa que o e-mail já existe			
2.2 Aplicação cancela a operação			

Figura 9: Processamento do Use Case: Registar Administrador

Adicionar Campeonato	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece o nome do Campeonato			
2. Aplicação verifica que o nome é válido	Verificar que o nome do Campeonato é válido	validaNomeCampeonato(nome: String): boolean	subCatálogos
3. Aplicação apresenta lista de Circuitos disponíveis	Recolher Circuitos disponíveis	recolherCircuitos(): List<Circuito>	subCatálogos
4. Administrador seleciona os Circuitos			
5. Administrador regista o Campeonato			
6. Aplicação adiciona o Campeonato à lista de campeonatos disponíveis	Registar Campeonato à lista de campeonatos disponíveis	registarCampeonato(nome: String, nomesCircuitos: List<String>): boolean	subCatálogos
Fluxo Excepção 1 [nome do Campeonato já existe] (passo 2)			
2.1 Aplicação informa que o nome já existe			
2.2 Aplicação cancela a operação			
Fluxo Excepção 2 [administrador não conclui registo do Campeonato] (passo 5)			
5.1 Administrador não regista o Campeonato			
5.2 Aplicação cancela a operação			

Figura 10: Processamento do Use Case: Adicionar Campeonato

Adicionar Circuito	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece o nome do Circuito			
2. Aplicação verifica que o nome é válido	Verificar que o nome do Circuito é válido	validaNomeCircuito(nome: String): boolean	subCatálogos
3. Administrador fornece o comprimento do Circuito, número de curvas e de chicanes			
4. Aplicação calcula o número total de retas do Circuito	Calcular o número total de retas	calcularQuantasRetas(numeroCurvas: int, numeroChicanes: int): int	subCatálogos
5. Aplicação apresenta lista de retas e curvas do Circuito			
6. Administrador fornece o GDU (grau de dificuldade) de cada curva e reta			
7. Administrador indica o número de voltas do Circuito			
8. Administrador regista o Circuito com os dados inseridos			
9. Aplicação regista o Circuito na lista de circuitos disponíveis	Registar Circuito na lista de circuitos disponíveis	registarCircuito(nome: String, comprimento: double, numeroCurvas: int, numeroChicanes: int, numeroRetas: int, numeroVoltas: int, secções: List<Secção>): boolean	subCatálogos
Fluxo Excepção 1 [nome do Circuito já existe] (passo 2)			
2.1 Aplicação informa que o nome já existe			
2.2 Aplicação cancela a operação			
Fluxo Excepção 2 [administrador não conclui registo do Circuito] (passo 8)			
8.1 Administrador não regista o Circuito			
8.2 Aplicação cancela a operação			

Figura 11: Processamento do Use Case: Adicionar Circuito

Adicionar C1	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece marca, modelo e potência			
2. Administrador indica fiabilidade			
3. Aplicação verifica que fiabilidade é aproximadamente 95%	Verificar a fiabilidade	verificaFiabilidade(fiabilidade: double): boolean	subCatálogos
4. Administrador indica que carro não é híbrido			
5. Administrador indica perfil aerodinâmico do Carro			
6. Administrador conclui o registo do Carro com os dados inseridos			
7. Aplicação regista o Carro na lista de carros disponíveis	Registar Carro na lista de carros disponíveis	registarC1(nome: String, fiabilidade: double, potencia: int, marca: String, modelo: String, hibrido: boolean, potenciaEletrico: int, perfil: double)	subCatálogos
Fluxo Alternativo 1 [administrador indica que carro é híbrido] (passo 4)			
4.1 Administrador indica potência do Motor Elétrico			
4.2 Regressa a 5			
Fluxo Excepção 2 [aplicação verifica que fiabilidade não é aproximadamente 95%] (passo 3)			
3.1 Aplicação verifica que fiabilidade não é aproximadamente 95% e é inválida para a categoria C1			
3.2 Aplicação cancela a operação			
Fluxo Excepção 3 [administrador não conclui o registo do Carro] (passo 6)			
6.1 Administrador não conclui o registo do Carro			
6.2 Aplicação cancela a operação			

Figura 12: Processamento do Use Case: Adicionar C1

Adicionar C2	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece marca, modelo, cilindrada e potência			
2. Aplicação verifica que cilindrada está entre 3000cm3 e 5000cm3	Verificar a cilindrada	verificaCilindrada(cilindrada: int): boolean	subCatálogos
3. Administrador indica que carro não é híbrido			
4. Administrador indica perfil aerodinâmico do Carro			
5. Administrador conclui o registo do Carro com os dados inseridos			
6. Aplicação regista o Carro na lista de carros disponíveis	Registar Carro na lista de carros disponíveis	registarC2(nome: String, fiabilidade: int, potencia: int, marca: String , modelo: String, hibrido: bool, potenciaEletrico: int, perfil: double)	subCatálogos
Fluxo Alternativo 1 [administrador indica que carro é híbrido] (passo 3)			
3.1 Administrador indica potência do Motor Elétrico			
3.2 Regressa a 4			
Fluxo Excepção 2 [aplicação verifica que cilindrada não está entre 3000cm3 e 5000cm3] (passo 2)			
2.1 Aplicação informa que cilindrada não está entre 3000cm3 e 5000cm3			
2.2 Aplicação cancela a operação			
Fluxo Excepção 3 [administrador não conclui o registo do Carro] (passo 5)			
5.1 Administrador não conclui o registo do Carro			
5.2 Aplicação cancela a operação			

Figura 13: Processamento do Use Case: Adicionar C2

Adicionar GT	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece marca, modelo, cilindrada e potência			
2. Aplicação verifica que cilindrada está entre 2000cm3 e 4000cm3	Verificar a cilindrada	verificaCilindrada(cilindrada: int): boolean	subCatálogos
3. Administrador indica que carro não é híbrido			
4. Administrador indica perfil aerodinâmico do Carro			
5. Administrador indica taxa de deterioração do Carro			
6. Administrador conclui o registo do Carro com os dados inseridos			
7. Aplicação regista o Carro na lista de carros disponíveis	Registar Carro na lista de carros disponíveis	registarGT(nome: String, fiabilidade: int, potencia: int, marca: String , modelo: String, hibrido: bool, potenciaEletrico: int, perfil: double, taxaDeterioracao: double)	subCatálogos
Fluxo Alternativo 1 [administrador indica que carro é híbrido] (passo 3)			
3.1 Administrador indica potência do Motor Elétrico			
3.2 Regressa a 4			
Fluxo Excepção 2 [aplicação verifica que cilindrada não está entre 2000cm3 e 4000cm3] (passo 2)			
2.1 Aplicação informa que cilindrada não está entre 2000cm3 e 4000cm3			
2.2 Aplicação cancela a operação			
Fluxo Excepção 3 [administrador não conclui o registo do Carro] (passo 6)			
6.1 Administrador não conclui o registo do Carro			
6.2 Aplicação cancela a operação			

Figura 14: Processamento do Use Case: Adicionar GT

Adicionar SC	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece marca, modelo e potência			
2. Administrador indica perfil aerodinâmico do Carro			
3. Administrador conclui o registo do Carro com os dados inseridos			
4. Aplicação regista o Carro na lista de carros disponíveis	Registar Carro na lista de carros disponíveis	registarSC(nome: String, potencia: int, marca: String, modelo: String, perfil: int)	subCatálogos
Fluxo Excepção 1 [administrador não conclui o registo do Carro] (passo 3)			
3.1 Administrador não conclui o registo do Carro			
3.2 Aplicação cancela a operação			

Figura 15: Processamento do Use Case: Adicionar SC

Adicionar Piloto	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Administrador fornece o nome do Piloto			
2. Administrador indica os níveis de perícia do Piloto			
3. Administrador regista o Piloto com os dados inseridos			
4. Aplicação regista o Piloto na lista de pilotos disponíveis	Registar Piloto na lista de pilotos disponíveis	registarPiloto(nome:String, CTS : double, SVA : double)	subCatálogos
Fluxo Excepção 1 [administrador não conclui o registo do Piloto] (passo 3)			
3.1 Administrador não conclui o registo do Piloto			
3.2 Aplicação cancela a operação			

Figura 16: Processamento do Use Case: Adicionar Piloto

Configurar Campeonato	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação apresenta lista de campeonatos disponíveis	Recolher lista de campeonatos disponíveis	getCampeonatos(): List<Campeonato>	subCatálogos
2. Jogador escolhe Campeonato			
3. Aplicação apresenta lista de carros disponíveis	Recolher lista de carros disponíveis	getCarros(): List<Carro>	subCatálogos
4. Jogador escolhe Carro			
5. Aplicação apresenta lista de pilotos disponíveis	Recolher lista de pilotos disponíveis	getPilotos(): List<Piloto>	subCatálogos
6. Jogador escolhe Piloto			
7. Jogador conclui a configuração do Campeonato			
8. Aplicação regista Campeonato como pronto para jogar	Registar Campeonato como pronto pra jogar	registarConfiguracao(campeonato: Campeonato, idJogador: String): void	SubPartidas
Fluxo Excepção 1 [jogador não conclui a configuração do Campeonato] (passo 7)			
7.1 Jogador não conclui a configuração do Campeonato			
7.2 Aplicação cancela a operação			

Figura 17: Processamento do Use Case: Configurar Campeonato

Entrar em Campeonato	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação apresenta lista de Pilotos disponíveis	Recolher lista de pilotos disponíveis	getPilotos(): List<Piloto>	subCatálogos
2. Utilizador escolhe Piloto			
3. Aplicação apresenta lista de Carros disponíveis	Recolher lista de carros disponíveis	getCarros(): List<Carro>	subCatálogos
4. Utilizador escolhe Carro			
5. Aplicação verifica que a categoria do carro é SC e calcula a fiabilidade	Verificar que a categoria é SC Calcular fiabilidade	verificaCarroSC(carro: String): boolean calculaFiabilidadeSC(carro: String, piloto: String): double	subCatálogos
6. Utilizador conclui o seu registo no Campeonato			
7. Aplicação regista Utilizador ao Campeonato	Registar Utilizador ao Campeonato	entrarNaPartida(idPartida: String, user: String, piloto: Piloto, carro: Carro): boolean	subPartidas
Fluxo Alternativo 1 [aplicação verifica que carro não pertence à categoria SC] (passo 5)			
5.1 Aplicação verifica que o carro não pertence à categoria SC			
5.2 Regressa a 6			
Fluxo Excepção 2 [utilizador não conclui o seu registo no Campeonato] (passo 6)			
6.1 Utilizador não conclui o seu registo no Campeonato			
6.2 Aplicação cancela a operação			

Figura 18: Processamento do Use Case: Configurar Campeonato

Entrar em Campeonato	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação apresenta lista de Pilotos disponíveis	Recolher lista de pilotos disponíveis	getPilotos(): List<Piloto>	subCatálogos
2. Utilizador escolhe Piloto			
3. Aplicação apresenta lista de Carros disponíveis	Recolher lista de carros disponíveis	getCarros(): List<Carro>	subCatálogos
4. Utilizador escolhe Carro			
5. Aplicação verifica que a categoria do carro é SC e calcula a fiabilidade	Verificar que a categoria é SC Calcular fiabilidade	verificaCarroSC(carro: String): boolean calculaFiabilidadeSC(carro: String, piloto: String): double	subCatálogos
6. Utilizador conclui o seu registo no Campeonato			
7. Aplicação regista Utilizador ao Campeonato	Registar Utilizador ao Campeonato	entrarNaPartida(idPartida: String, user: String, piloto: Piloto, carro: Carro): boolean	subPartidas
Fluxo Alternativo 1 [aplicação verifica que carro não pertence à categoria SC] (passo 5)			
5.1 Aplicação verifica que o carro não pertence à categoria SC			
5.2 Regressa a 6			
Fluxo Excepção 2 [utilizador não conclui o seu registo no Campeonato] (passo 6)			
6.1 Utilizador não conclui o seu registo no Campeonato			
6.2 Aplicação cancela a operação			

Figura 19: Processamento do Use Case: Entrar em Campeonato

Configurar Corridas	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação avalia que categoria do Carro permite alterações de afinação	Verificar que a categoria do Carro permite alterações	verificaPossibilidadeAlteracoes(carro: String): boolean	subCatálogos
2. Aplicação verifica que há alterações de afinação disponíveis	Verificar que há alterações disponíveis	verificaDisponibilidadeAfinacoes(user: String): boolean	subPartidas
3. Utilizador decide alterar afinação do Carro			
4. Utilizador indica as alterações de afinação do Carro			
5. Aplicação regista alteração de afinação realizada	Registar alteração de afinação	aplicarNovaAfinacao(afinacao: double, user: String): boolean	subPartidas
6. Utilizador escolhe os pneus e e modo do motor			
7. Utilizador conclui a configuração na Corrida			
8. Aplicação regista o Utilizador como pronto a iniciar a Corrida	Registar configurações da Corrida feitas pelo Utilizador e que este encontra-se pronto a iniciar a Corrida	registarConfiguracaoCorrida(tipoPneus: String, modoMotor: String ,user: String): boolean	subPartidas
Fluxo Alternativo 1 [aplicação avalia que categoria do Carro não permite alterações de afinação] (passo 1)			
1.1 Aplicação verifica que categoria do Carro não permite alterações de afinação			
1.2 Regressa a 6			
Fluxo alternativo 2 [aplicação verifica que não há alterações de afinação disponíveis] (passo 2)			
2.1 Aplicação verifica que não há alterações de afinação disponíveis			
2.2 Regressa a 6			
Fluxo alternativo 3 [utilizador não altera afinação] (passo 3)			
3.1 Utilizador não altera a afinação do Carro			
3.2 Regressa a 6			

Figura 20: Processamento do Use Case: Configurar Corridas

Simular Corridas	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação inicia a Corrida	Iniciar a Corrida	iniciarCorrida()	subPartidas
2. Aplicação confirma que a Simulação está na versão base	Verificar qual o tipo de versão de jogo do Jogador	verificarVersao(username: String): boolean	subUtilizador
3. Aplicação calcula os eventos (despistes, avarias e ultrapassagens) a cada reta, curva e chicane	Calcular os eventos a cada reta, curva e chicane	calcularEventos(corrida : Corrida): List<Evento>	subPartidas
4. Aplicação calcula as posições dos Carros ao final de cada volta	Calcular as posições dos Carros	calcularPosicao() : List<String>	subPartidas
5. Aplicação calcula e atualiza a fiabilidade dos carros pertencentes à categoria GT ao final de cada volta	Calcular a fiabilidade dos Carros	calcularFiabilidades()	subPartidas
6. Aplicação calcula e regista resultado final da Corrida	Calcular resultado final da Corrida	finalizarCorrida() : Resultado	subPartidas
Fluxo Alternativo 1 [simulação está na versão premium] (passo 2)			
2.1 Aplicação calcula o tempo entre os Carros	Calcular o tempo entre os Carros	calcularTempo()	subPartidas
2.2 Aplicação calcula os eventos e posições a cada reta, curva e chicane	Calcular os eventos a cada reta, curva e chicane	calcularEventos(corrida: Corrida): List<Evento>	subPartidas
2.3 Regressa a 5			

Figura 21: Processamento do Use Case: Simular Corrida

Finalizar Campeonato	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Aplicação verifica os resultados referentes a cada Corrida do Campeonato e calcula o resultado fina	Calcular resultado final	calculaResultadoFinal() : List<String>	subPartidas
2. Aplicação verifica que o Jogador está autenticado	Verificar que o Jogador está autenticado	validaEmail(user: String) : boolean	subUtilizadores
3. Aplicação regista pontuação do Jogador no ranking global	Registar pontuação do Jogador	registarPontuacao(username: String, nomeCampeonato: String, pontos: int)	subUtilizadores
4. Aplicação finaliza Campeonato	Finalizar Campeonato	finalizarCampeonato()	subPartidas
Fluxo Alternativo 1 [jogador autentica-se] (passo 2)			
2.1 Aplicação verifica que jogador não está autenticado			
2.2 Jogador autentica-se			
2.3 Regressa a 3			
Fluxo Alternativo 2 [jogador não deseja se autenticar] (passo 2.2)			
2.2.1 Jogador não se autentica			
2.2.2 Regressa 4			

Figura 22: Processamento do Use Case: Finalizar Campeonato

Mudar versão do Jogo	Responsabilidades	API	Subsistemas
Fluxo Normal			
1. Jogador indica que quer mudar a versão de jogo			
2. Aplicação modifica a versão de jogo do Jogador	Alterar versão de jogo do Jogador	alteraVersão(idJogador : String)	subUtilizadores

Figura 23: Processamento do Use Case: Mudar Versão do Jogo

6 Reaproveitamento de código

Foi fornecido um código *legacy* de uma aplicação Racing Manager similar à que está a ser desenvolvida. O objetivo seria ajudar na identificação de componentes e facilitar o processo de implementação, ao possibilitar o reaproveitamento de código ou da lógica envolvida.

A seguir será indicado explicitamente o que se espera reaproveitar conjuntamente com as respetivas explicações.

6.1 Aposta e Equipa

As classes Aposta e Equipa foram desconsideradas por não se encaixarem nos requisitos e funcionamento proposto da aplicação a ser desenvolvida, visto que não possui funcionalidades de apostas e também não possui o conceito de equipa. Há apenas Jogadores, cada um com sua escolha de Carro e Piloto entre outros fatores.

6.2 JogadorComparatorNome e TimeConverter

Foi decidido que estas classes não são necessárias na implementação deste projeto. Por exemplo, a classe TimeConverter pode ser reformulada como um método. Estas questões podem ser reavaliadas e alteradas aquando a implementação do código.

6.3 CarteiradeJogadores

É necessário possuir de alguma forma todos os jogadores registados na aplicação. De uma forma mais geral, todos os utilizadores, visto que neste projeto há o conceito de Administrador que não há na versão legacy fornecida.

Portanto, a lógica e o conceito desta classe será reaproveitada, porém com a visão de uma coleção de utilizadores, estes podendo ser jogadores ou administradores.

Nesta classe fornecida há também um Set que representa a classificação dos Jogadores. Foi decido não aproveitar e, ao invés, cada Jogador possuir uma Carreira, na qual irá constar as partidas em que esteve presente e sua devida pontuação.

6.4 Campeonato

A classe Campeonato não será reaproveita pois, nesta versão legacy o Campeonato refere-se à simulação da partida. Foi decidido manter uma classe Campeonato mas que reflete apenas os atributos físicos do Campeonato que são imutáveis, tal como o nome e a lista de Circuitos.

Portanto, para o tratamento da simulação do jogo em si, há uma classe Partida que será composta pelo Campeonato a ser realizado, as Corridas realizadas e todos os Eventos que acontecem durante a simulação. Ou seja, tudo que é mutável e possivelmente único àquela simulação.

6.5 Piloto

Esta classe será reaproveitada com umas pequenas alterações nas variáveis de instância, visto que neste projeto será apenas considerado o nome e os níveis de perícia.

6.6 Record

A lógica em si desta classe será reaproveitada, visto que é essencial guardar o Carro e o Piloto escolhido pelo Jogador. Porém, a classe terá um nome diferente e informação adicional.

Neste projeto será a classe Estado que irá abordar as escolhas do Jogador associadas a cada Corrida que compõe a simulação da Partida no Campeonato. Estas classe englobará a informação do Carro, Piloto, tipo de pneus, modo do motor, quantas afinações ainda possui e a afinação em si (downforce).

6.7 Circuito

Esta classe será reaproveitada, porém com a mesma lógica do reaproveitamento da classe Campeonato mencionada anteriormente. Apenas será levado em conta tudo que corresponda ao facto físico e imutável do Circuito em si.

Esta classe contém, por exemplo, os Records que já caracterizam escolhas relativas a uma corrida num circuito já numa simulação. Portanto, tudo que diz respeito à Partida será descartado da classe Circuito, mantendo apenas o que foi passado aquando a construção do mesmo pelo Administrador.

6.8 Carro

A classe Carro assim como a sua abstração e a lógica de herança será reaproveitada. Para além disso, também será reaproveitada o código referente à diferença entre os carros que possuem a opção do motor Híbrido. Isto foi decidido pelo facto de se encaixar com perfeição no Modelo de Domínio previamente proposto como é possível analisar na Figura 24.

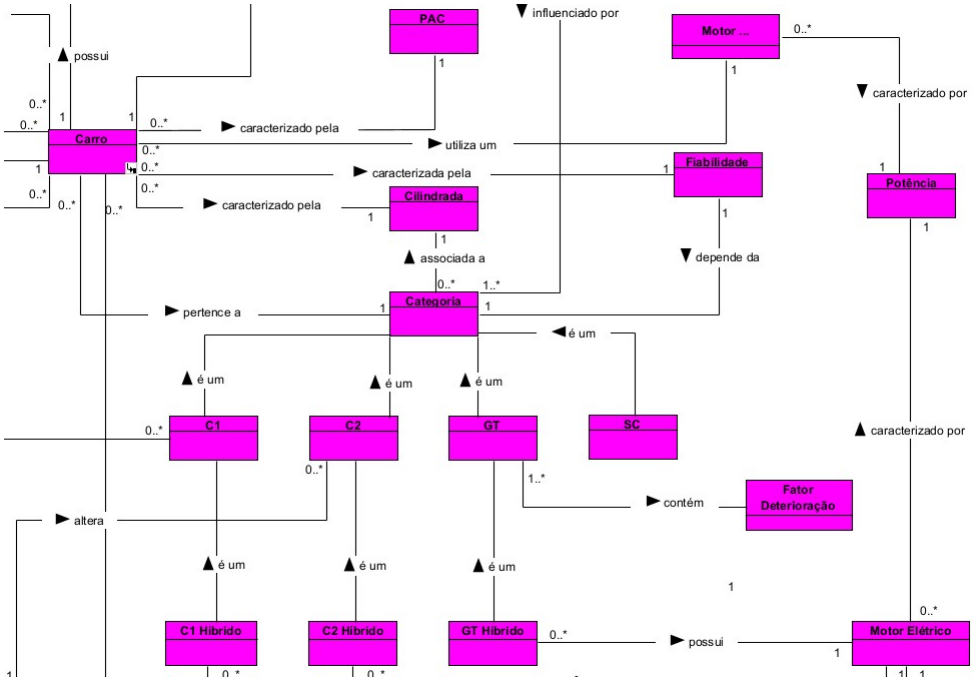

Figura 24: Modelo proposto para os carros

7 Diagrama de Componentes

Durante a modelação de grandes sistemas de software, é importante dividir o software em diferentes subsistemas, quer para facilitar a gestão do mesmo, como também contribui para a modularidade do código.

Assim, foi criada a primeira versão do Diagrama de Componentes, tal como é possível observar na Figura 25.

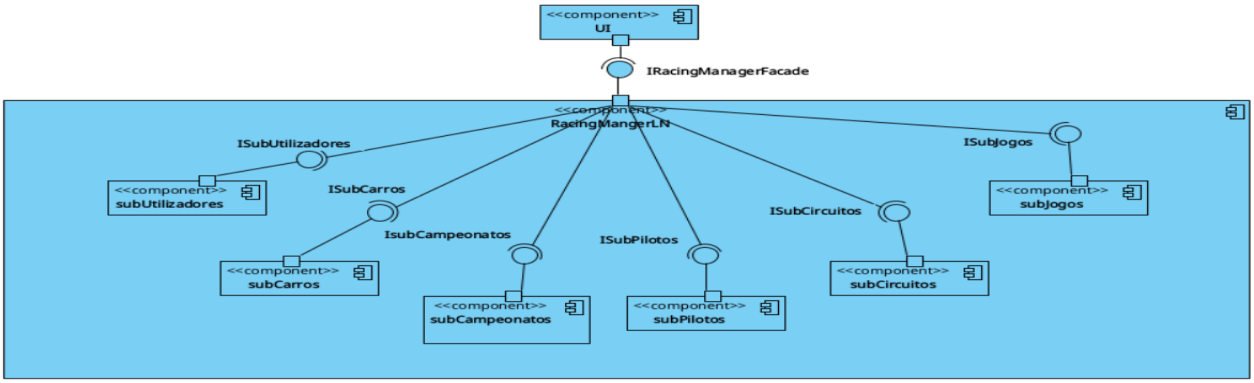

Figura 25: 1ª Versão do Diagrama de Componentes

Inicialmente, a divisão prevista dividiria o sistema em: Partidas, Utilizadores, Campeonatos, Carros e Pilotos. No entanto, estes 3 últimos limitam-se ao acesso a informação armazenada previamente. Dado isso, decidiu-se que a melhor estratégia seria a criação de um subsistema de catálogos que, por sua vez, seria responsável pelo acesso aos dados do sistema.

Por sua vez, o subsistema de utilizadores será responsável por operações sobre os mesmos, como por exemplo, criação de conta, verificação da versão, entre outras.

Já o subsistema de partidas é responsável pelo próprio "jogo", isto é, a realização das partidas entre os jogadores, a simulação dos campeonatos, bem como de cada uma das corridas do mesmo.

Tendo em conta tudo isto, foi, finalmente, construido o diagrama de componentes final, constituído pelos três subsistemas, bem como as suas interfaces, a facade do próprio sistema e o componente de UI que será utilizado para interagir com o mesmo.

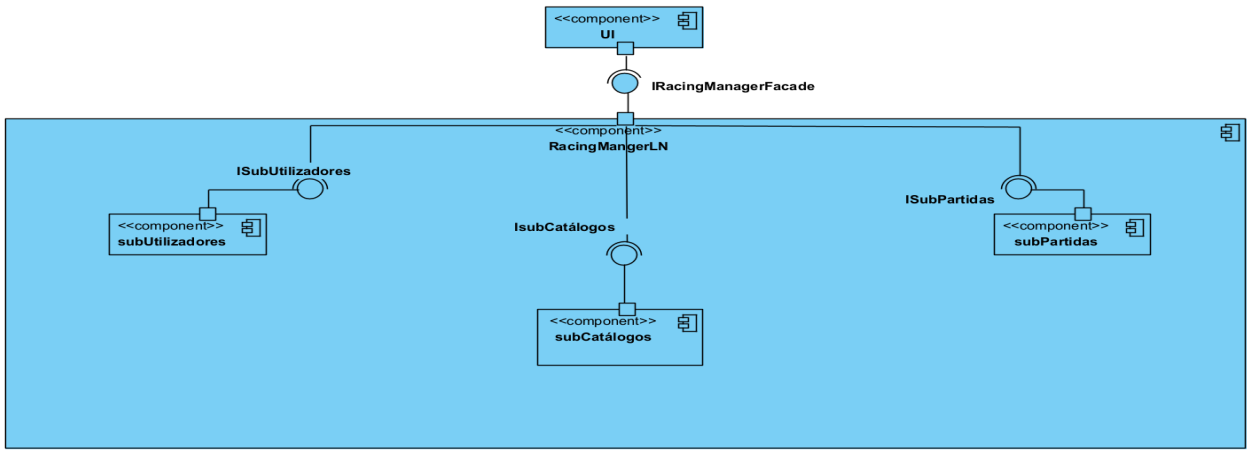

Figura 26: Diagrama de Componentes

8 Diagrama de Classes e de *Packages*

Como no programa *Visual Paradigm (UML)* é possível misturar os diagramas de classes com os de packages, apresentamos apenas uma figura que mostra os diagramas de classes e de packages construídos para a modelação do sistema em causa.

Cada package é composto por uma interface, uma classe Subsistema e um conjunto de classes respetivas a esse package. Foram adicionados os métodos especificados anteriormente na fase de processamento de use cases.

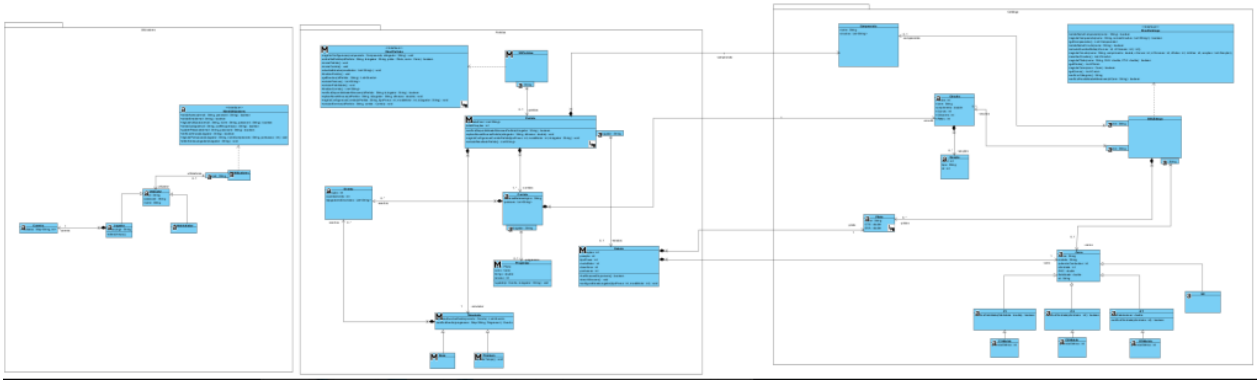

Figura 27: Diagrama de Packages geral

8.1 *Package* Utilizadores

O *package* Utilizadores contém as classes que identificam as várias informações dos Utilizadores que estão registados no sistema. Este *package* tem como interface IGestUtilizadores, que fornece os métodos visíveis a classes externas.

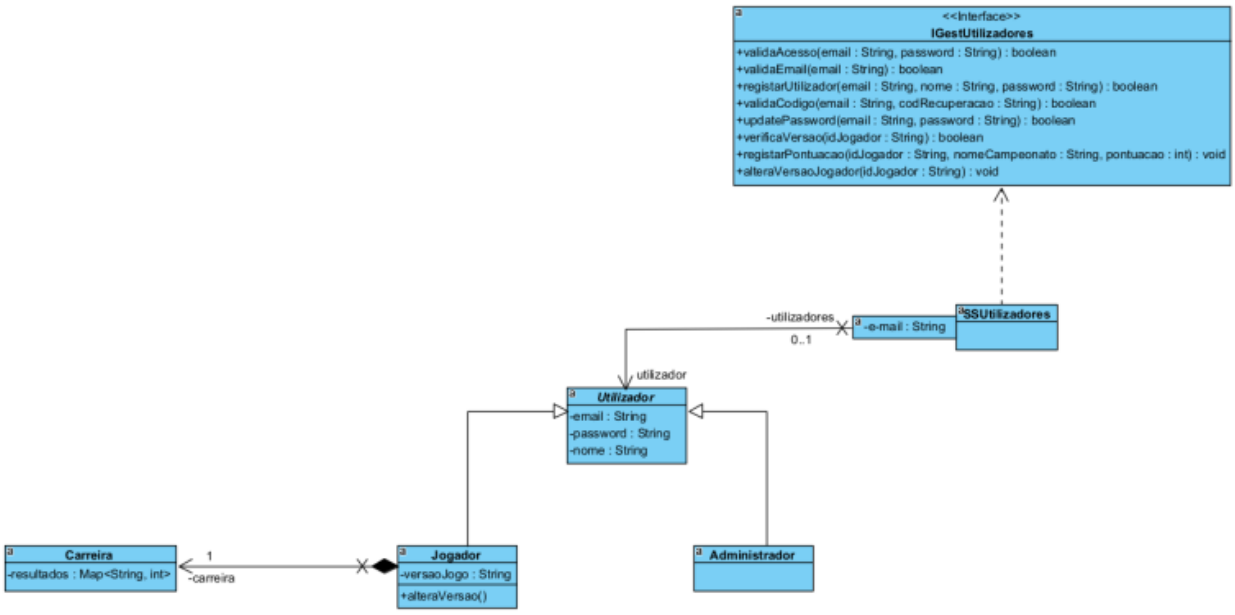

Figura 28: Diagrama de Classe dos Utilizadores

8.2 *Package* Catálogo

Este *package* contém todas as classes e métodos que envolvem o acesso e gestão dos dados de todos os pilotos, veículos, circuitos e campeonatos disponíveis no sistema. É através da interface IGestCatálogo que são fornecidos os métodos visíveis às classes externas.

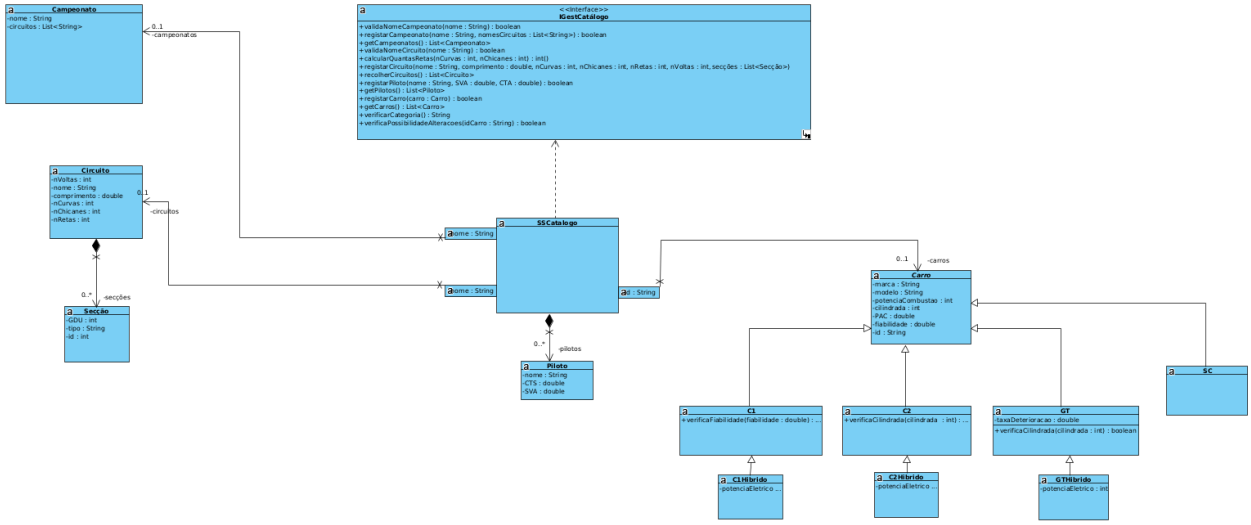

Figura 29: Diagrama de Classe do Catálogo

8.3 *Package* Partidas

O *package* Partidas contém as várias classes que geram todas as partidas, incluindo os estados e eventos que ocorreram em cada uma delas. Tem como interface IGestPartidas, que fornece os métodos visíveis a classes externas. Para além disso, tem acesso à classe Carro através do SubSistemaCatálogo.

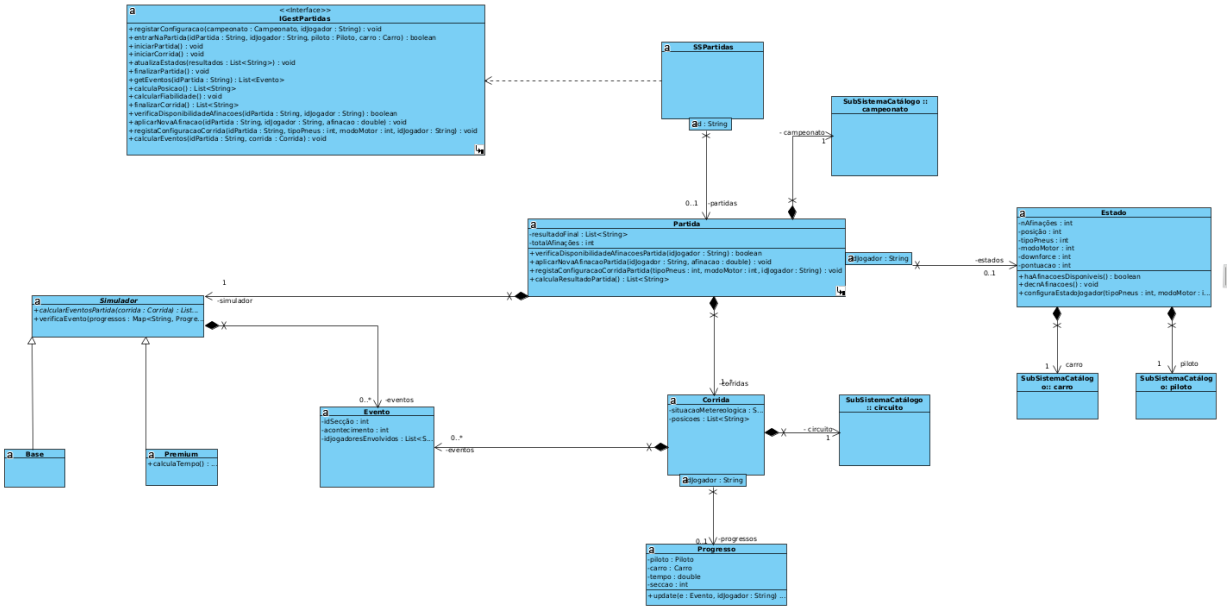

Figura 30: Diagrama de Classe das Partidas

9 Diagramas de Atividade

Para uma melhor e mais clara especificação do funcionamento do programa foram criados diagramas de atividades relativos quer ao funcionamento global das partidas, quer relativo à simulação de uma partida.

9.1 Simulação

O seguinte diagrama de atividades demonstra a sequência de eventos ao longo do processo de simulação de uma partida. Decidimos utilizar este diagrama para diminuir a complexidade do problema e tornar o funcionamento do programa mais percetível .

Este diagrama inicia-se quando um novo jogador decide participar numa partida. Assim, nesse momento, o mesmo pode decidir entrar numa partida já criada(*Figura 36:"entrarNaPartida"*) ou criar uma nova, passando a configurá-la (*Figura 37:"registarConfiguracao"*).

Após todos os jogadores estarem registados e preparados para jogar, dá-se início à partida. As partidas são compostas por campeonatos, que, por sua vez, são compostos por circuitos. Assim, no decorrer da partida, todas as corridas do campeonato escolhido são realizadas. Para além disso, no final de cada uma, é verificada a existência de corridas por realizar até que se esgotem. A cada uma das corridas o jogador deve registar a configuração para esta (*Figura 34:"registarConfiguracaoCorrida"*). Mais uma vez, quando todos os jogadores concluírem o registo, dar-se-à o início à corrida.

A corrida, dependendo da versão de jogo do criador da partida, irá, então, realizar a simulação dos eventos, tendo em conta os carros e pilotos que nela participam. Tudo isto é um processo complexo. Assim, para uma melhor descrição do mesmo foi desenvolvido o diagrama de sequência "calculaEvento"(*Figura 36:"calculaEvento"*), responsável por melhor caracterizar o processo inicial da criação do mapa inicial de jogadores e o seu respetivo progresso. Para complementar o mesmo, decidimos que o mais adequado seria o desenvolvimento de um diagrama de atividades (*Figura 31:"Diagrama de Atividade do processo de simulação"*), uma vez que consideramos que é capaz de representar melhor os momentos de cálculo de eventos e fluxo da própria simulação. Ainda assim, os detalhes mais específicos, isto é, as condições para o ocorrência dos próprios eventos nas corridas, tornariam, quer este último diagrama, como um diagrama de sequência, menos compreensível, para além de complexo. Após tal observação, concluímos que seria mais adequada uma descrição mais detalhada, que, não sendo em UML, é ainda clara e descritiva do problema a analisar, especificando-o devidamente para a futura implementação (*Capitulo 8*).

Tendo terminado a simulação de todas as voltas e todos os eventos, o programa deve finalizar a corrida e atualizar os estados atuais de todos os jogadores de acordo com o resultado da mesma.

Finalmente, quando todas as corridas terminam, deve-se calcular o resultado final da partida, obtendo a tabela de classificações finais para a mesma. O diagrama termina com a atividade de finalizar partida que vai definir a mesma como concluída, não existindo mais ações a realizar sobre a mesma.

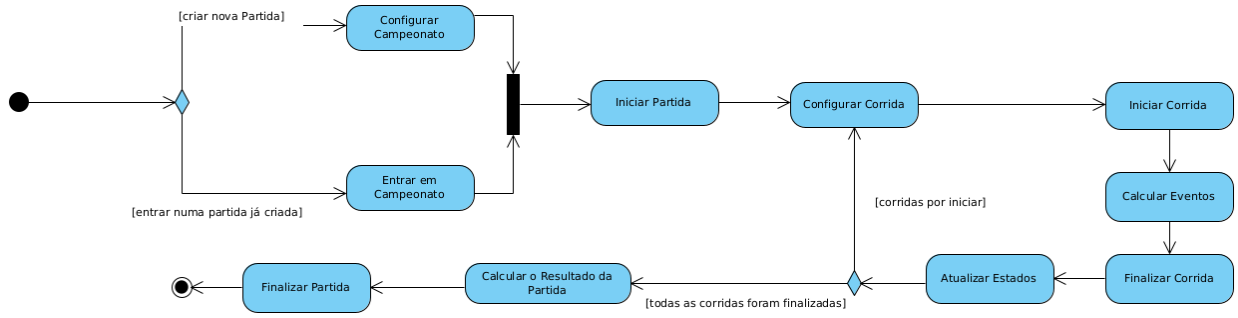

Figura 31: Diagrama de Atividade do processo de simulação

9.2 Calcular Novo Evento

O diagrama de atividades abaixo representa a criação da lista de eventos que decorrem ao longo de uma corrida.

Para iniciar o processo o sistema recolhe o número de voltas da corrida atual(N). Este é utilizado de forma a garantir que todas as secções da pista são percorridas pela simulação N vezes. Por sua vez, também as secções são recolhidas pelo sistema para que, por cada volta, todas sejam percorridas ordenadamente pelos pilotos.

A cada par de piloto-carro, a cada secção, é realizada a verificação se estiveram envolvidos em algum evento. Isto é, se o par realizou uma manobra de ultrapassagem ou se esteve envolvido num acidente com zero, um ou vários outros carros.

Caso nenhuma das situações anteriores se verifique o progresso do jogador atual é atualizado de modo a simbolizar que passou a secção e registar o novo tempo. Por outro lado, tendo-se realizado algum dos eventos descritos a cima, este é inserido na lista de eventos da corrida. Adicionalmente, caso vários jogadores estejam envolvidos num evento, todos devem ter o seu progresso atualizado. Para isso devemos obter a lista de envolvidos, retirando da lista de jogadores a verificar os participantes que não seja o jogador a ser verificado atualmente. Ainda relativo aos jogadores envolvidos deve-se atualizar o seu estado, individualmente. Em caso de incidente, o mesmo deve ser "retirado da pista", isto é, não pode realizar mais voltas ou estar envolvido em novos eventos. Nos casos de ultrapassagem, apesar de envolver vários carros, apenas afetarão, a nível de progresso, o tempo dos jogadores, uma vez que o carro que ultrapassa ficará com um tempo menor e, consequentemente, à frente do seu adversário. Neste caso, a secção é também atualizada uma vez que todos os envolvidos continuam a progredir na corrida.

Finalmente, após todas as voltas e secções serem verificadas dá-se o final da simulação, retornando a lista de eventos que ocorreram ao longo da corrida.

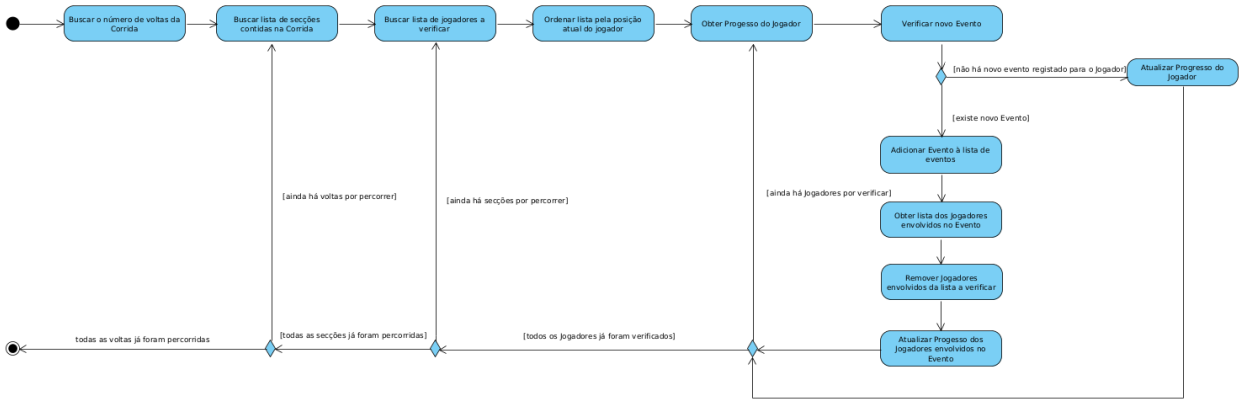

Figura 32: Diagrama de Atividade do processo de simulação (Versão Premium)

9.2.1 Versão Base

É importante referir que a versão base do jogo não realiza uma simulação tão complexa. Assim sendo, ao invés de os eventos serem verificados a cada secção do circuito, estes são realizados volta a volta. Isto resulta numa simulação menos precisa e consequentemente com menos detalhe.

O diagrama de atividades, apesar de muito semelhante ao da simulação premium, difere no sentido de apenas realizar as verificações de eventos volta a volta, em contraste com a verificação secção a secção anteriormente analisada.

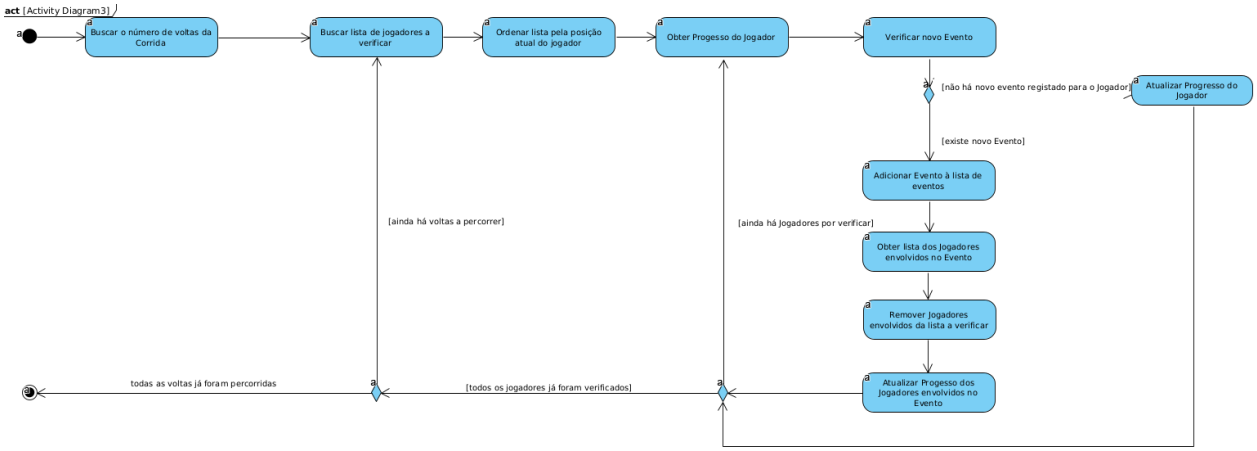

Figura 33: Diagrama de Atividade do processo de simulação (Versão Base)

10 Diagramas de Sequência

Com o objetivo de exemplificar o funcionamento das várias operações, a forma de como estas interagem com os subsistemas, e as entidades envolvidas, foram desenvolvidos os seguintes diagramas, organizados segundo os *packages* onde estão inseridos.

Apenas foram desenvolvidos os diagramas de sequências que consideramos as mais importantes para os *Packages* "Utilizadores", "Partidas"e "Catálogo", uma vez que consideramos que os seus outros métodos não têm tanta influência direta na lógica de negócio do sistema.

Assim, os diagramas de sequência desenvolvidos seguintes pertencem a operações presentes nos *packages* Partidas e Utilizadores.

De realçar que todos os diagramas desenvolvidos têm como primeira linha de vida o *facade* **RacingManagerFacade**. Esta será a classe que servirá como uma interface frontal que "esconde"o código subjacente ou estrutural mais complexo.

A seguir são demonstrados e explicados os diagramas de sequência desenvolvidos.

10.1 verificaDisponibilidadeAfinações

No caso do método verificaDisponibilidadeAfinações, que, tal como o nome indica, se pretende verificar se existem afinações disponíveis, existem 5 linhas de vida, excluindo a *facade*. Este método recebe dois argumentos, o id da Partida e o id do Jogador.

Primeiro, é verificado num Map designado partidas e que pertence ao SubSistema Partidas, se a partida em questão, identificada pelo id recebido como argumento, existe. Caso exista, é recebido o objeto partida, à qual é aplicada o método principal, no entanto, neste caso, apenas com o argumento id do Jogador. Após a obtenção do estado do jogador, existente num Map de estados, é verificado, nesse estado, se há afinações disponíveis. A *flag* final é colocada a `True` se tal se verificar. Senão é retornado `False`.

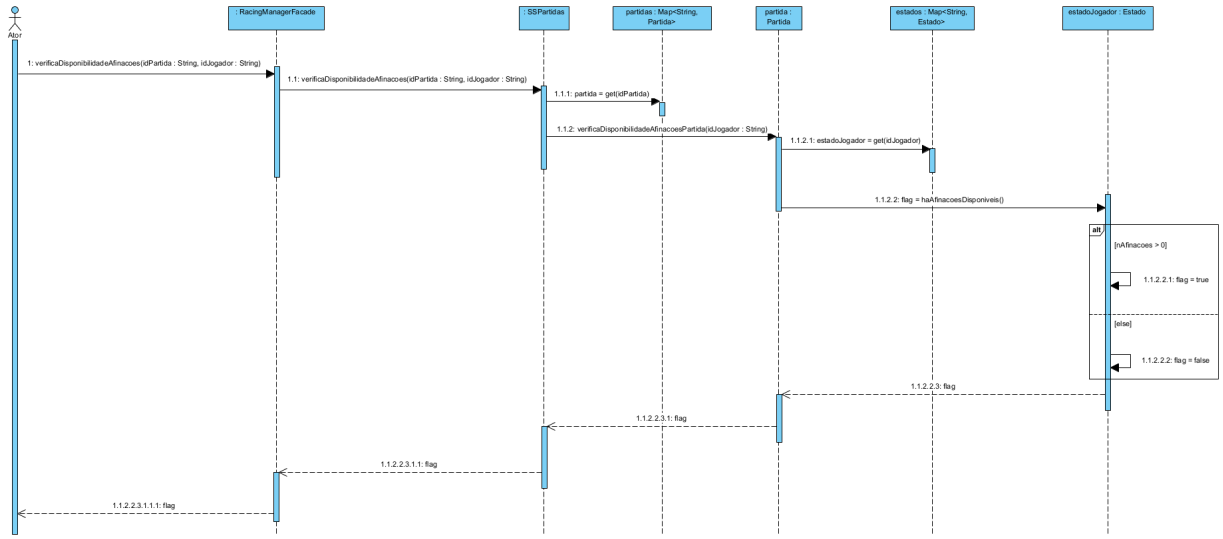

Figura 34: Diagrama de Sequência do método "verificarDisponibilidadeAfinacoes"

10.2 entrarNaPartida

Este método tem como objetivo inserir um Jogador (com um Piloto e Carros já definidos) numa Partida específica, para que este possa jogar.

Sabendo que o método recebe o id da Partida, a partir do Map de partidas existentes (pertencente ao SubSistema Partidas) obtemos a Partida desejada. Antes mesmo de o Jogador ser inserido no Map jogadores, o sistema calcula o número de afinações que o Jogador terá disponíveis durante a Partida. Com esses dados mais o id do Jogador, o piloto e o carro que o respetivo Jogador escolheu, cria-se o Estado do Jogador. Por fim, o id do Jogador e o seu respetivo Estado são inseridos no Map de jogadores.

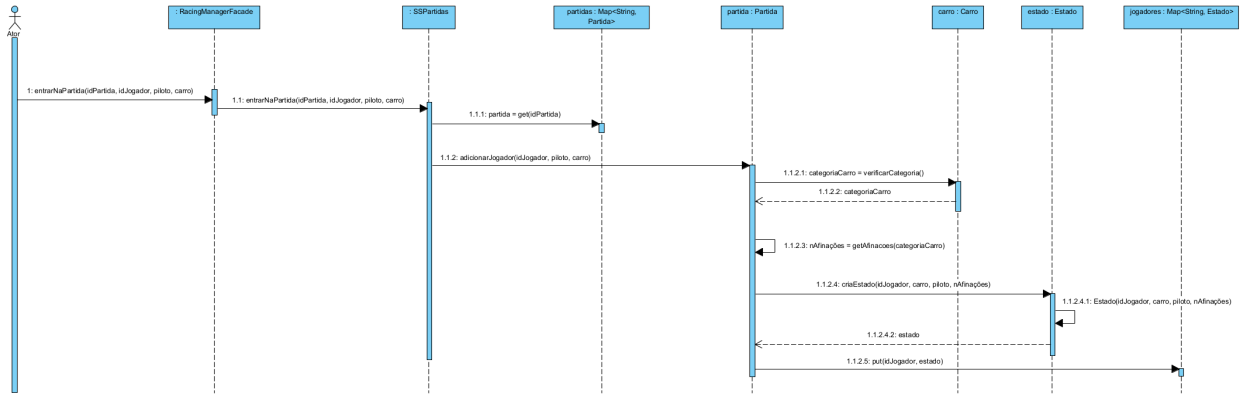

Figura 35: Diagrama de Sequência do método "entrarNaPartida"

10.3 aplicarNovaAfinacao

O método **aplicarNovaAfinacao** diminui um valor unitário ao número de afinações que estão disponíveis para o jogador utilizar na respetiva Partida. Para além disso, tem de alterar o valor da *Downforce* que um Jogador possui.

O número de afinações só é reduzido se o número de afinações disponíveis for superior a zero. O método **aplicarNovaAfinacao** vai ao SubSistema Partidas procurar pela Partida e, consecutivamente, pelo Jogador para aplicar as alterações ao seu número de afinações e *Downforce*.

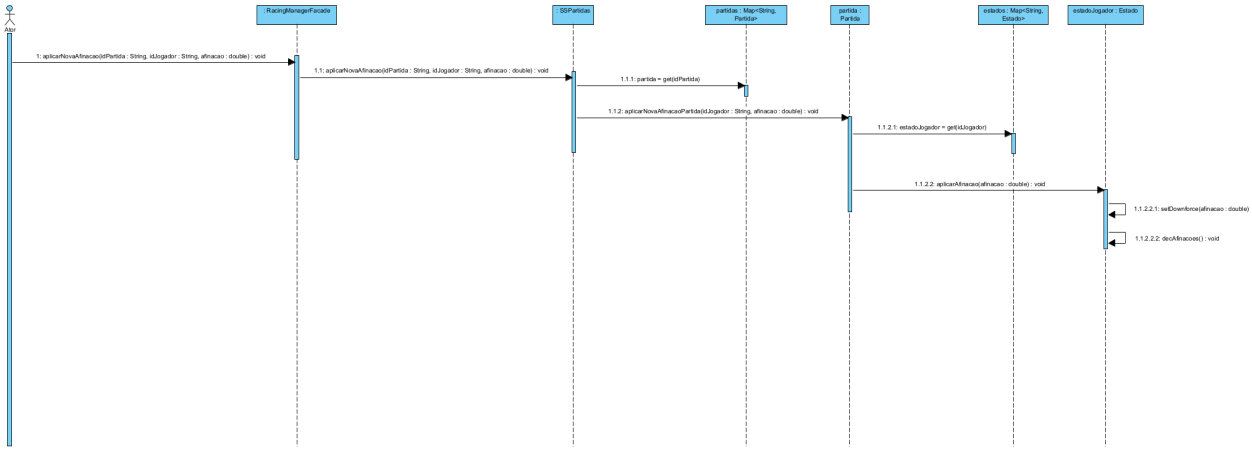

Figura 36: Diagrama de Sequência do método "aplicarNovaAfinacao.png"

10.4 registaConfiguracaoCorrida

Este método acede primeiramente à partida especificada pelo idPartida recebida como argumento. Nessa partida, efetua o método registarConfiguracaoCorridaPartida com os restante argumentos recebidos. Para além disso, através do idJogador também é obtido o objeto estado associado ao Jogador. Nesse estado é adicionado o tipo de Pneu e o Modo do Motor.

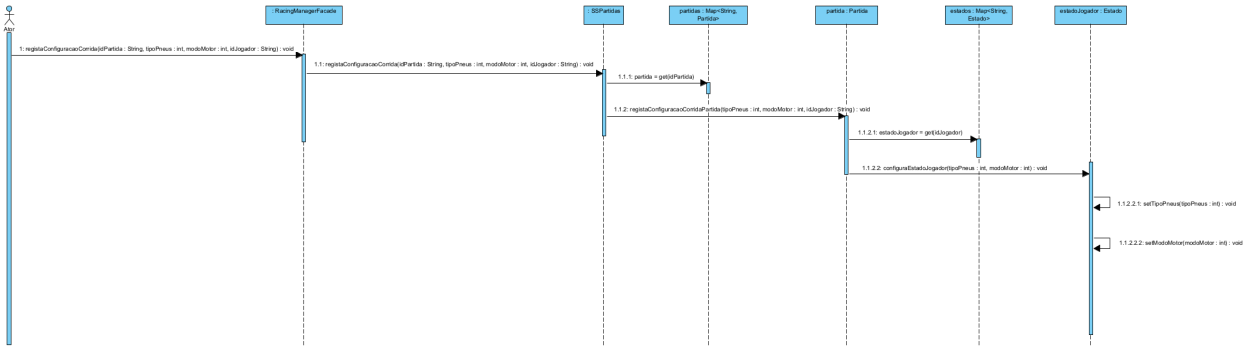

Figura 37: Diagrama de Sequência do método "registaConfiguracaoCorrida.png"

10.5 registarConfiguracao

Este método acede primeiramente ao subsistema de partidas de modo a, recebendo um campeonato e o id de um jogador, criar um novo objeto do tipo Partida. Recorrendo ao id do utilizador, este método acede ao subsistema do mesmo de modo a verificar a versão de jogo. Após obter a versão a utilizar, deve ser criado o simulador desejado, guardando-o na própria partida. Por fim, deve-se, a partir do campeonato, obter o total de afinações disponíveis para os carros que participarem desta partida. Quando a partida acaba de ser especificada, é colocada no mapa de partidas atuais, presente no SSPartidas.

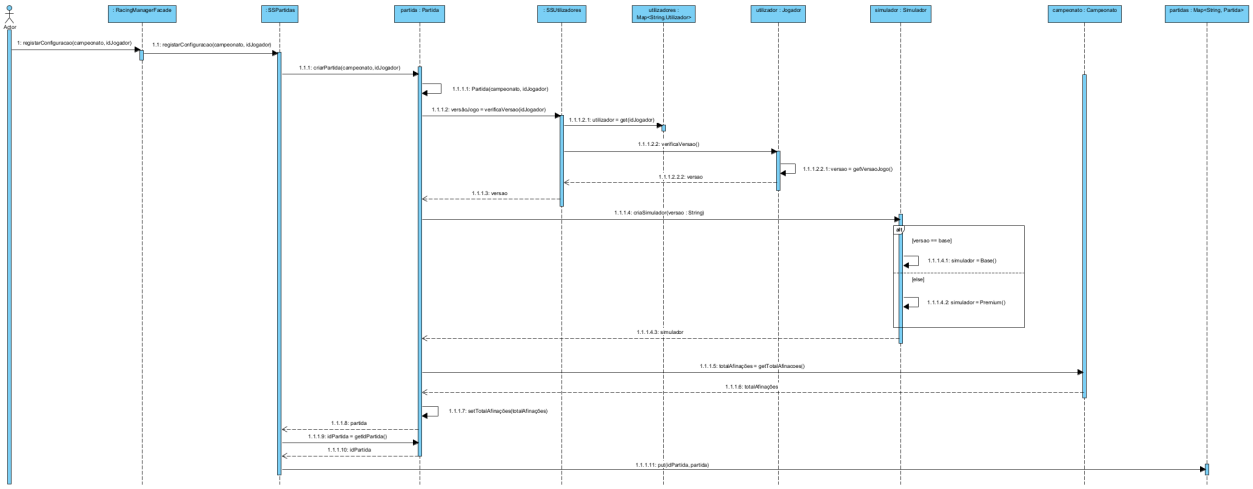

Figura 38: Diagrama de Sequência do método "registarConfiguracao"

10.6 calculaEventos

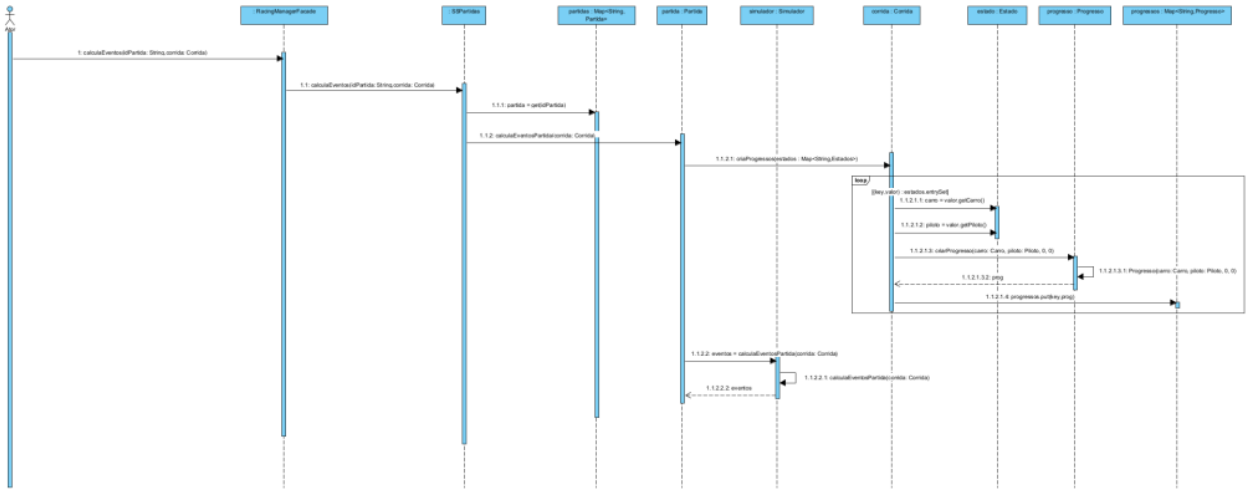

Figura 39: Diagrama de Sequência do método "calculaEventos"

10.7 registarCampeonato

O método registarCampeonato é responsável por inserir no SubSistema Catálogo um Campeonato com todos os dados para a sua criação. Caso já exista um Campeonato com o mesmo nome este não é inserido novamente, permanecendo o original. Se o nome desejado ainda não existir no mapa de campeonatos vai ser criado um novo objeto com as informações fornecidas. Finalmente este é inserido na lista de campeonatos. A função retorna verdadeiro caso tenha sido criado o novo campeonato.

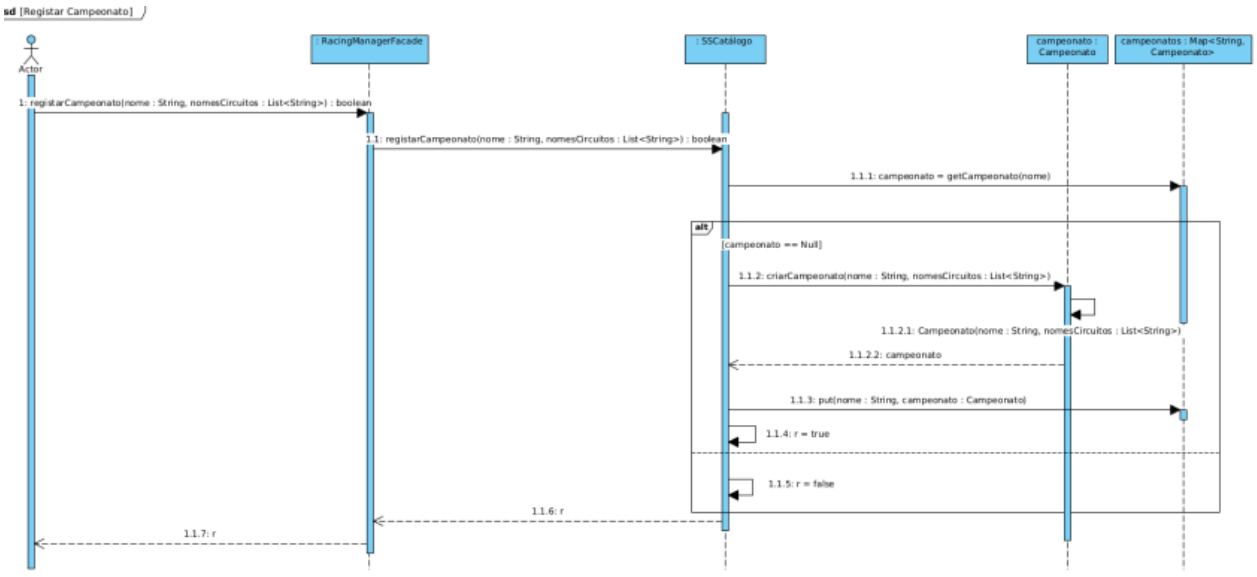

Figura 40: Diagrama de Sequência do método "registarCampeonato"

10.8 registarCircuito

O método registarCircuito é responsável por inserir no SubSistema Catálogo um Circuito com todos os dados para a sua criação. Caso já exista um Circuito com o mesmo nome, o circuito que se queira inicialmente inserir não é inserido no Map com os circuitos disponíveis no sistema.

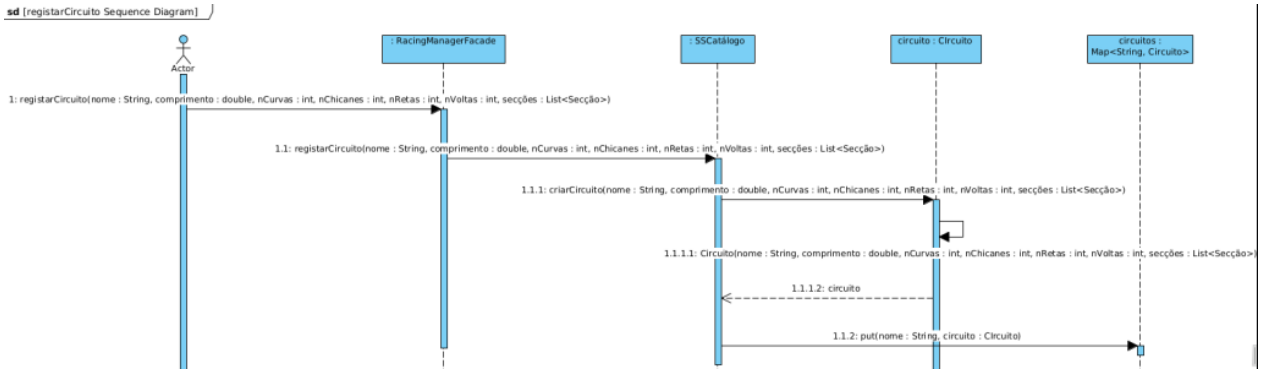

Figura 41: Diagrama de Sequência do método "registarCircuito"

10.9 registarCarro

O método registarCarro consiste em guardar no SubSistema Catálogo um Carro (já construído) no Map de carros disponíveis no sistema.

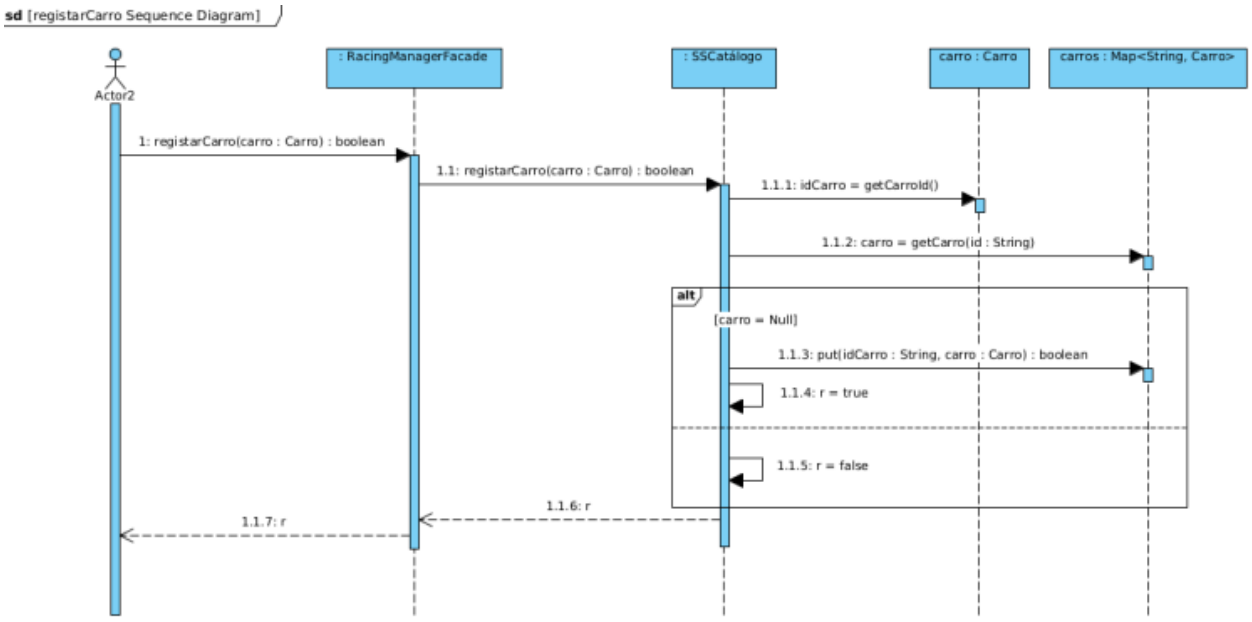

Figura 42: Diagrama de Sequência do método "registarCarro"

10.10 registarPiloto

O método registarPiloto consiste em inserir um novo Piloto no SubSistema Catálogo, desde que este não exista na lista de pilotos disponíveis no sistema.

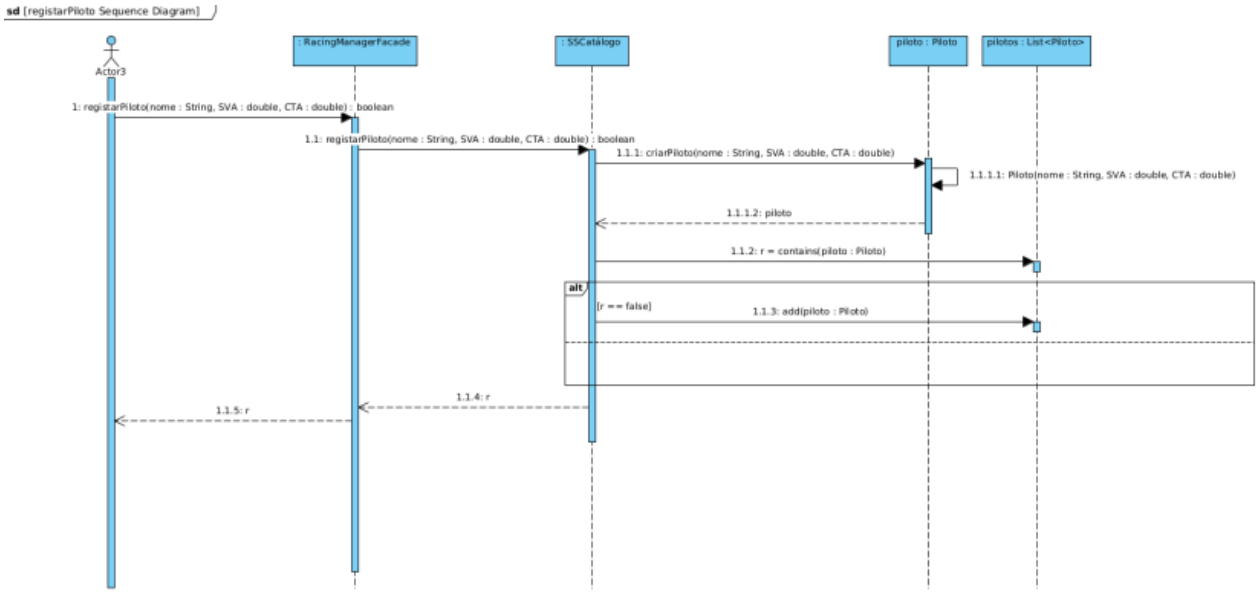

Figura 43: Diagrama de Sequência do método "registarPiloto"

11 Conclusão e Análise dos resultados obtidos

Com o projeto realizado, foi possível aprofundar conhecimentos no que toca à análise conceptual de sistemas a partir de requisitos bem definidos, bem como na construção de modelos comportamentais que definam o sistema, nomeadamente, diagramas de componentes, de atividade, de sequência e de *packages*.

Em suma, o projeto está a percorrer um bom caminho e considera-se que o mesmo se encontra bem estruturado de forma a permitir o início da próxima e última fase de forma organizada e concreta: Modelação conceptual e implementação da solução.