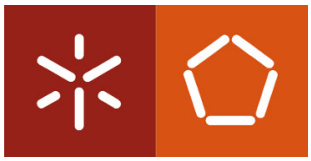
Universidade do Minho

Escola de Engenharia

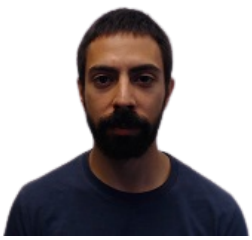


Desenvolvimento de Sistemas de Software

Licenciatura em Engenharia Informática

Grupo 17

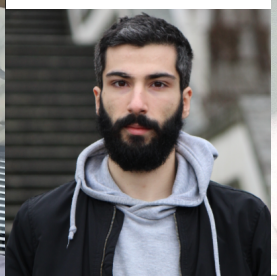
GIT: https://github.com/MiguelJacinto99/DSS



(a) A78778 - Adélio Fernandes

(b) A96854 - João Ferreira

(c) A84518 - Miguel Carvalho



(d) A80960 - Rúben Rodrigues

(e) A76650 - Rui Morais

Índice

Lista de Figuras

1 Introdução

No âmbito da unidade curricular de Desenvolvimento de Sistemas de Software do 1º semestre do 3º ano do curso de Engenharia Informática da Universidade do Minho, com o principal objetivo a consolidação da matéria lecionada.

Foi nos proposto o desenvolvimento de um simulador de campeonatos onde os utilizadores competem em provas automobilísticas que o software vai simular, chamado com a génese da aplicação similar à do F1 Manager. Com uma implementação faseada, sendo este relatório referente ao desenvolvimento da segunda fase do trabalho prático. Fizemos também pequenas alterações relativamente à primeira fase que serão descritas ao longo do relatório.

Todas as decisões importantes foram tomadas em conjunto, devido à complexidade e subjetividade do projeto. É também importante mencionar que durante o período de realização da segunda fase alteramos várias vezes a estrutura do projeto até à solução final, apresentada neste relatório.

2 Objetivos da Segunda Fase

O objetivo da segunda fase do projeto consiste no desenvolvimento do Modelo Conceptual da solução. Os principais objetivos, definidos pelos docentes, são:

→ **Desenho da Arquitetura Conceptual do Sistema, capaz de suportar os requisitos identificados**

→ **Os modelos comportamentais necessários para descrever o comportamento pretendido para o sistema**

Depois de analisarmos e termos uma ideia de que funcionalidades o nosso jogo poderá ter, iniciamos um conjunto inicial de métodos que deverão suportar o jogo final. Começamos então, por definir as responsabilidades da lógica de negócio que nos vai permitir identificar melhor a API global da lógica de negócio. De seguida, de modo a permitir uma organização da lógica de negócio, identificamos subsistemas de maneira a permitir uma divisão maior das responsabilidades, impedindo que todo o código se destine a uma só classe.

Construimos o **Diagrama de Componentes** no qual iremos apresentar os sistemas e subsistemas da arquitetura, bem como as relações entre eles.

O **Diagrama de Classes** onde identificamos todas as classes, atributos e métodos utilizados, assim como as relações entre eles.

O **Diagrama de Sequência** que permitem uma maior clareza da das mensagens trocadas.

O **Diagrama de Packages** Para dar uma visão geral dos packages utilizados e as relações entre eles.

3 Alteração dos Requisitos da Primeira Fase

Com a implementação dos diagramas da segunda fase do projeto, e também tendo em atenção as dicas fornecidas pelos docentes, resolvemos fazer umas pequenas alterações nos requisitos definidos na primeira fase.
Sendo o projeto divido em várias fases vitais á implementação final, acreditamos que venham ainda a ser necessárias futuras alterações geradas pela informação referente à implementação de uma terceira fase.

3.1 Modelo de Domínios

Primeiro resolvemos fazer uma alteração simples, mas importante para facilitar a leitura do modelo, alterar as cores. Começamos por identificar as identidades principais e associar uma cor especifica a cada uma, de seguida alteramos a cor de qualquer outra identidade que esteja associada as que consideramos como principais.

De seguida, fizemos uma alteração nas entidades relacionadas com as classes relacionadas ao carro, sendo que é importante no projeto diferenciar as classificações de um carro com motor de combustão e um carro com motor híbrido. Assim, resolvemos acrescentar novas entidades, híbridas, que se relacionam com as de combustão.

Por fim, alteramos também a afinação, pois após mais uma analise do problema, apercebemos-nos nesta segunda fase, que a afinação engloba o PAC e o modo de motor, e um jogador pode fazer essa alteração no fim de cada partida.

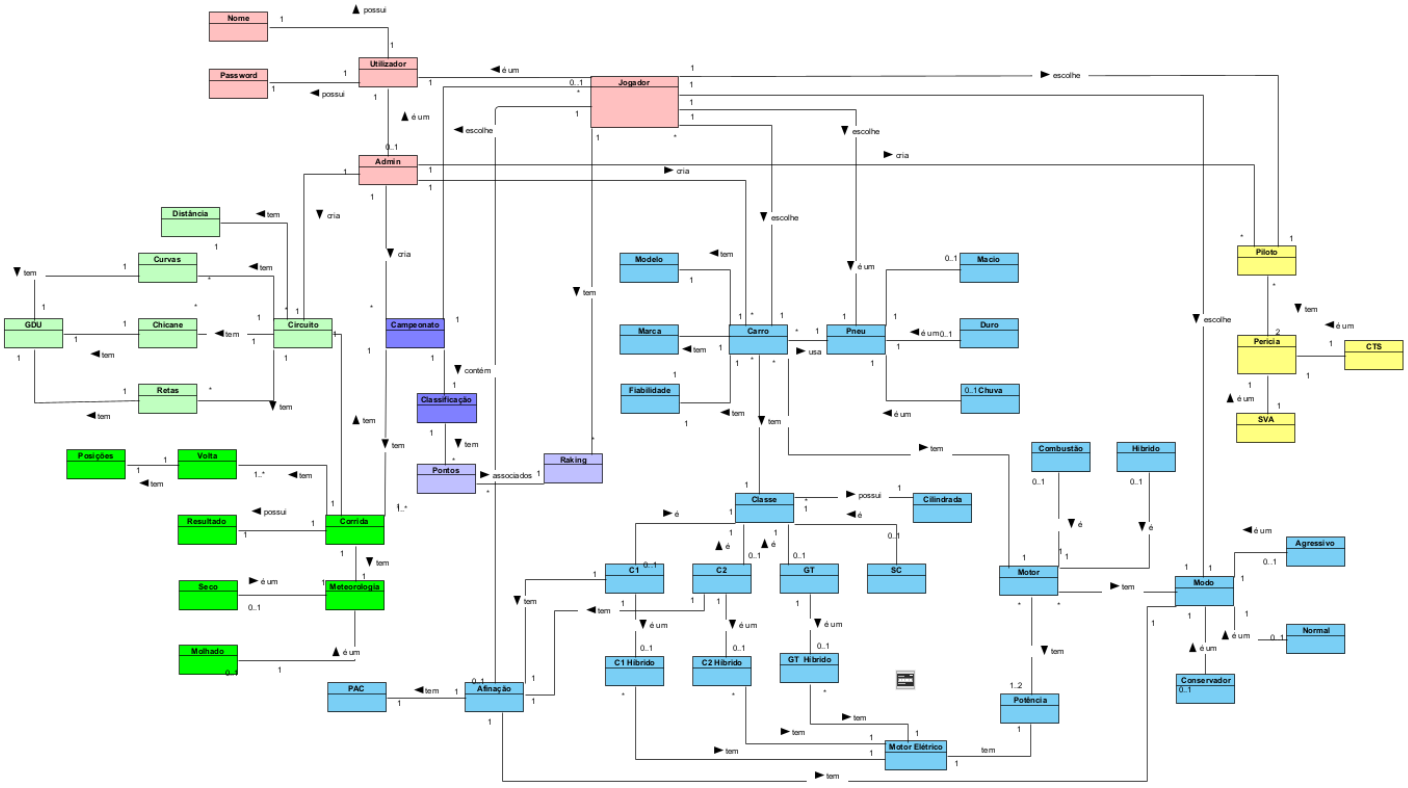


Figura 2: Modelo de domínios atualizado.

3.2 Diagrama de use cases

No diagrama de use cases, apenas removemos o use case **simula**, sendo que não tem qualquer interação direta com um utilizador.

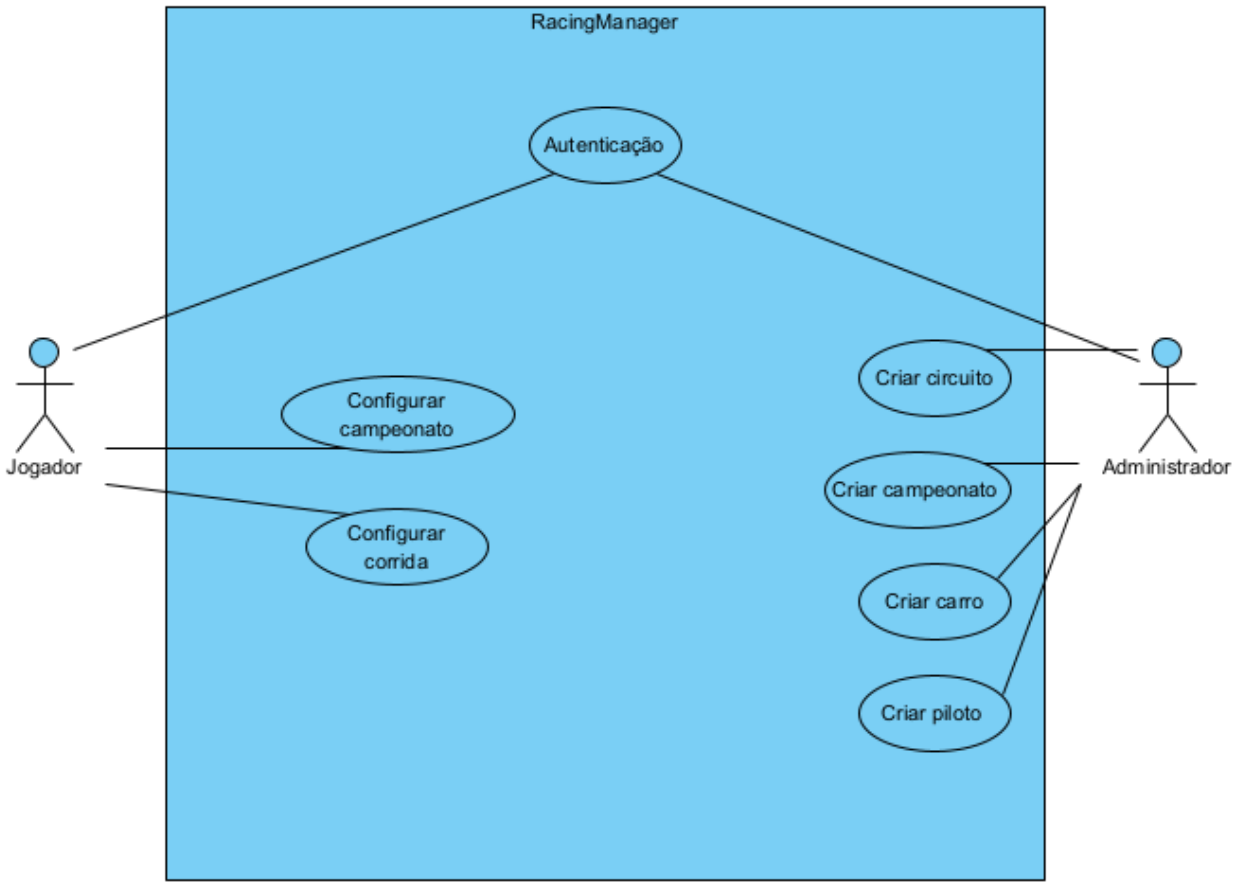


Figura 3: Diagrama de use cases atualizado.

4 Identificação das responsabilidades e desenvolvimento da API da Lógica de Negócios

Analisando a descrição de cada um dos Use Cases, fomos capazes de definir um conjunto de responsabilidades que a Lógica de Negócios do sistema deverá cumprir para os suportar. Apresentamos, em seguida estas responsabilidades, divididas de acordo com o Use Case em que foram identificadas, associando a cada uma das responsabilidades um método a implementar, definindo assim a API global da Lógica de Negócios.

Use Case	Fluxo	Responsabilidade	API	Subsistema
O administrador seleciona a opção de criar campeonato	UI			
O administrador indica o nome do campeonato	UI			
O sistema verifica que o nome é válido		Verificar validade do nome	nomeValidoC(String nome) : boolean	SSCampeonatos
O sistema apresenta a lista de circuitos disponíveis		Apresentar a lista de circuitos disponíveis	getListCircuitos() : List<Circuito>	SSCorridas
O administrador seleciona os circuitos pretendidos dessa lista	UI			
O administrador indica que quer adicionar o novo campeonato à lista de disponíveis para jogar	UI			
O sistema adiciona o novo campeonato à lista de disponíveis para jogar		Adicionar o campeonato à lista de campeonatos disponíveis	AddListCampeonato(campeonato Campeonato) : void	SSCampeonatos
Exceção	1	Nome inválido		
O sistema verifica que o nome é inválido		Verificar validade do nome	nomeValidoC(String nome) : boolean	SSCampeonatos
O sistema indica que a criação do campeonato foi cancelada	UI			
Exceção	2	Administrador não quer adicionar à lista		
O administrador não quer adicionar	UI			
O sistema indica que a criação do campeonato foi cancelada	UI			

Figura 4: Representação tabela Use Case cenário 1 - Adicionar Campeonato.

Use Case	Fluxo	Responsabilidade	API	Subsistema
O administrador seleciona a opção de criar circuito	UI			
O administrador indica o nome do circuito	UI			
O sistema verifica que o nome é válido		Verifica se o nome é válido	nomeValidoCir(String nome) : boolean	SSCorridas
O administrador fornece os valores da distância, nº de curvas e nº de chicanes	UI			
O sistema verifica que os valores fornecidos são válidos		Verificar validade dos valores fornecidos		SSCorridas
O sistema calcula o nº de retas do circuito e apresenta a lista de curvas e retas		Calcular no nº de retas do circuito e apresentar a lista de curvas e retas	setCaminho(int curvas, int chicane) : List<String>	SSCorridas
O administrador indica o GDU em cada uma das retas e curvas	UI			
O administrador fornece o nº de voltas	UI			
O administrador indica que quer adicionar o novo circuito à lista de disponíveis para integrar em campeonatos	UI			
O sistema adiciona o novo circuito à lista de disponíveis para jogar		Adicionar circuito à lista de circuitos	addListCircuitos(circuito Circuito) : void	SSCorridas
Fluxo Exceção	1	Nome inválido		
O sistema verifica que o nome é inválido		Verificar o nome inserido	nomeValidoCir(String nome): boolean	SSCorridas
O sistema cancela a criação do circuito	UI			
Fluxo Exceção	2	Valor inválido		
O sistema verifica que existem valores inválidos		Verificar os valores inseridos		SSCorridas
O sistema indica que a criação do circuito foi cancelada	UI			
Fluxo Exceção	3	Administrador não quer adicionar à lista		
Administrador não quer adicionar	UI			
O sistema indica que a criação do circuito foi cancelada	UI			

Figura 5: Representação tabela Use Case cenário 2 - Adicionar Circuito.

Use Case	Fluxo	Responsabilidade	API	Subsistema
O administrador seleciona a opção de criar um novo carro	UI			
O sistema fornece as classes disponíveis nesta versão do jogo		Fornecer as lista de classes disponíveis	getListClasses() : List<String>	SSCampeonatos
O administrador escolhe a classe	UI			
O sistema verifica que a classe selecionada é C1 e que necessita de fiabilidade		Verificar a classe selecionada		SSCampeonatos
O administrador fornece o valor da fiabilidade	UI			
O sistema verifica que a fiabilidade é aproximadamente 95%		Verificar a fiabilidade		SSCampeonatos
O administrador indica o modelo, marca, cilindrada e potência do motor de combustão	UI			
O sistema verifica que os valores fornecidos são válidos		Verificar validade dos valores fornecidos		SSCampeonatos
O administrador indica que não é híbrido	UI			
O administrador indica o PAC	UI			
O sistema verifica que o valor do PAC é válido		Verificar validade do PAC		SSCampeonatos
O administrador indica que quer adicionar o novo carro à lista de disponíveis para ser utilizado em jogo	UI			
O sistema adiciona o novo carro à lista de disponíveis para ser utilizado em jogos		Adicionar o carro à lista de carros disponíveis	addListCarro(carro Carro) : void	SSCampeonatos
Exceção	1	Fiabilidade muito distante de 95%		

O sistema verifica que a fiabilidade é demasiado distante de 95%		Verificar valor da fiabilidade inserido		SSCampeonatos
O sistema indica que a criação do carro foi cancelada	UI			
Exceção	2	Valor de modelo, marca, cilindrada ou potência do motor de combustão inválido		
O sistema verifica que um dos valores fornecidos é inválido		Verifica que um dos valores é inválido		SSCampeonatos
O sistema indica que a criação do carro foi cancelada	UI			
Alternativo	3	Carro é híbrido		
O administrador indica que é híbrido	UI			
O sistema verifica que a informação fornecida é válida em conta a classe anteriormente fornecida		Verifica validade da informação fornecida		SSCampeonatos
O administrador indica a potência do motor elétrico	UI			
O sistema verifica que o valor fornecido é válido		Verifica validade da informação fornecida		SSCampeonatos
Regressa a 10				
Exceção	4	carro não pode ser híbrido devido à classe selecionada		
O sistema verifica que a informação fornecida é inválida tendo em conta a classe anteriormente fornecida		Verificar validade da informação fornecida		SSCampeonatos
O sistema indica que a criação do carro foi cancelada	UI			
Exceção	5	Valor do PAC inválido		

O sistema verifica que o valor do PAC é inválido		Verificar a validade do PAC fornecido		SSCampeonatos
O sistema indica que a criação do carro foi cancelada	UI			
alternativo	7	A classe selecionada é C2		
O sistema verifica que a classe selecionada é C2 e que necessita de fiabilidade		Verificar se a classe selecionada necessita de fiabilidade		SSCampeonatos
O administrador fornece o valor da fiabilidade	UI			
O sistema verifica que a fiabilidade é aproximadamente 80		Verificar o valor da fiabilidade inserido		
Regressa a 7				
Exceção	8	Fiabilidade muito distante de 80%		
O sistema verifica que a fiabilidade é demasiado distante de 80%		Verificar o valor da fiabilidade inserido		SSCampeonatos
O sistema indica que a criação do carro foi cancelada				
Alternativo	9	A classe selecionada é o GT		
o sistema verifica que a classe selecionada é GT		Verificar se a classe selecionada		SSCampeonatos
regressa a 7				
Alternativo	10	A classe selecionada é SC		
o sistema verifica que a classe selecionada é SC		Verificar se a classe selecionada		SSCampeonatos
regressa a 7				

Figura 6: Representação tabela Use Case cenário 3 - Adicionar Carro.

Use Case	Fluxo	Responsabilidade	API	Subsistema
O administrador seleciona a opção de criar piloto	UI			
O administrador indica o nome do piloto	UI			
O sistema verifica que o nome é válido		Verifica se o nome é válido	nomeValidoP(String nome) : boolean	SSCampeonatos
O administrador indica os níveis de perícia	UI			
O sistema verifica que os valores fornecidos são válidos		Verifica validade dos valores fornecidos		SSCampeonatos
O administrador indica que quer adicionar o novo piloto à lista de disponíveis para ser utilizado em jogos	UI			
Exceção	1	Nome inválido		
O sistema verifica que o nome do piloto é inválido		Verifica se o nome é válido	NomeValidoP(String nome) : boolean	SSCampeonatos
O sistema indica que a criação do piloto foi cancelada	UI			
Exceção	2	Níveis de perícia inválidos		
O sistema verifica que os níveis de perícia fornecidos são inválidos		Verifica validade dos valores fornecidos		SSCampeonatos
O sistema indica que a criação do piloto foi cancelada	UI			
Exceção	3	Administrador não quer adicionar		
O administrador não quer adicionar à lista de disponíveis para ser utilizado em jogos	UI			
O sistema indica que a criação do piloto foi cancelada	UI			

Figura 7: Representação tabela Use Case cenário 4 - Adicionar Piloto .

Use Case	Fluxo	Responsabilidade	API	Subsistema
O sistema fornece a lista de campeonatos disponíveis para jogar		Fornecer lista de campeonatos disponíveis	getListCampeonatos() : List<Campeonato>	SSCampeonatos
O jogador escolhe um campeonato	UI			
O sistema fornece a lista de carros disponíveis		Fornecer a lista de carros disponíveis	getListCarros() : List<Carro>	SSCampeonatos
O jogador escolhe o carro	UI			
O sistema fornece a lista de pilotos disponíveis		Fornecer lista de pilotos disponíveis	GetListPilotos(): List<Piloto>	SSCampeonatos
O jogador escolhe o piloto	UI			
O jogador indica que quer continuar	UI			
O sistema indica que a configuração do campeonato está completa	UI			
Alternativo	1	Não quer continuar		
O jogador indica que não quer continuar	UI			
O sistema indica que a configuração do campeonato foi cancelada	UI			

Figura 8: Representação tabela Use Case cenário 5 - Configurar Campeonato.

Use Case	Fluxo	Responsabilidade	API	Subsistema
O sistema indica a corrida e fornece as condições em que se vai realizar	UI			
O jogador não quer alterar a afinação	UI			
O jogador escolhe o tipo de pneus e o modo de funcionamento do motor	UI			
O sistema indica que a configuração da corrida está completa	UI			
Alternativo	1	Quer alterar afinação		
o jogador quer alterar afinação	UI			
o sistema verifica que o jogador ainda não atingiu o limite de alterações possíveis e que a classe do carro utilizado o permite		Verificar se o jogador atingiu limite de alterações	verificaNrAlteracoes() : boolean	SSCorridas
o jogador fornece o valor da downforce	UI			
regressa a 4				
Alternativo	2	Não é possível alterar afinação		
o sistema verifica que o jogador já atingiu o limite de alterações possíveis ou que a classe do carro utilizado não o permite		Verificar se o jogador atingiu limite de alterações	verificaNrAlteracoes() : boolean	SSCorridas
regressa a 4				

Figura 9: Representação tabela Use Case cenário 5 - Configurar Corrida .

Use Case	Fluxo	Responsabilidade	API	Subsistema
O sistema dá início à corrida.	UI			
O sistema simula a volta	UI			
O sistema apresenta as posições dos jogadores no final da volta	UI			
O sistema verifica que não há mais voltas a dar		Verifica número de voltas restantes	getProva() : int	SSCampeonatos
O sistema fornece os resultados da corrida	UI			
O sistema verifica que não existem circuitos por finalizar no campeonato		Verificar circuitos por finalizar	simularProximaCorrida() : String	SSCampeonatos
O sistema fornece as classificações do campeonato	UI			
Alternativo	1	Existem voltas a dar		
o sistema verifica que ainda existem voltas efetuar no circuito		Verifica número de voltas restantes	getProva() : int	SSCampeonatos
regressa a 2				
Alternativo	2	Existem circuitos por finalizar no campeonato		
o sistema verifica que existem circuitos por finalizar no campeonato		Verificar circuitos por finalizar	simularProximaCorrida() : String	SScampeonatos
regressa a 1				

Figura 10: Representação tabela Use Case cenário 5 - Simular .

O utilizador introduz nome e password	UI			
O sistema verifica que nome e password são válidos		Verificar dados inseridos	nomeValidoU(String nome) : boolean pwValida(String password) : boolean	SSUtilizadores
O sistema informa que autenticação foi bem-sucedida	UI			
Exceção	1	Dados inválidos		
O sistema verifica que nome ou password estão inválidos		Verificar dados inseridos	nomeValidoU(String nome) : boolean pwValida(String password) : boolean	SSUtilizadores
O sistema indica que a autenticação foi cancelada	UI			

Figura 11: Representação tabela Use Case - Autenticar Utilizador .

5 Diagrama de Componentes

A partir da análise dos **Use Cases** relativos ao **Cenário 5**, criamos um Diagrama de Componentes, com os seguintes sistemas:

- **RacingManagerUI**: Camada que trata da interatividade entre utilizador e aplicação.

- **RacingManagerLN**: Lógica de negócio da aplicação, com todos os subsistemas necessários. Esta camada trata dos dados da aplicação.

RacingManagerUI

Este sistema trata da interface de utilizador, onde o utilizador irá interagir com a aplicação. Para obter as informações a apresentar ao utilizador, usa a interface IRacingManagerLN.

RacingManagerLN

O sistema de lógica de negócio, expõe a interface IRacingManagerLN, para garantir o encapsulamento de dados e abstração da implementação. Através desta interface, o sistema RacingManagerUI, irá comunicar e obter a informação a apresentar para o utilizador.

- SSUtilizadores: Subsistema relativo aos dados de utilizadores. A ligação com este sistema é através da interface ISSUtilizadores.

- SSCorridas: Subsistema relativo aos dados das corridas. A ligação com este sistema é através da interface ISSCorridas.

- SSCampeonatos: Substistema relativo ao campeonatos. A ligação com este sistema é através da interface ISSCampeonatos.

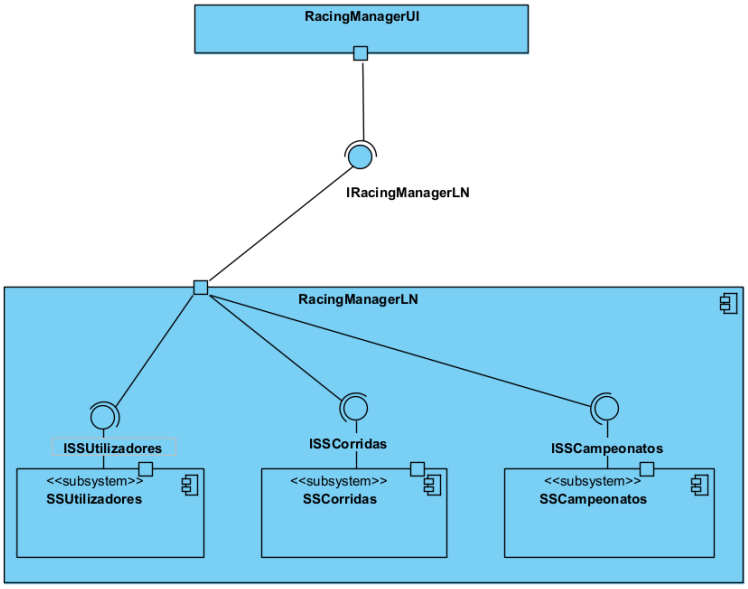


Figura 12: Representação do Diagrama de Componentes

6 Diagrama de Classes

Utilizando a estrutura apresentada no Diagrama de Componentes desenvolvemos alguns Diagramas de Classe para representar as classes que compõem cada um dos sistemas definindo de forma concreta essas classes a implementar e os relacionamentos estruturais entre as várias entidades.

De seguida apresentamos os vários diagramas de classes construídos no VisualParadigm:

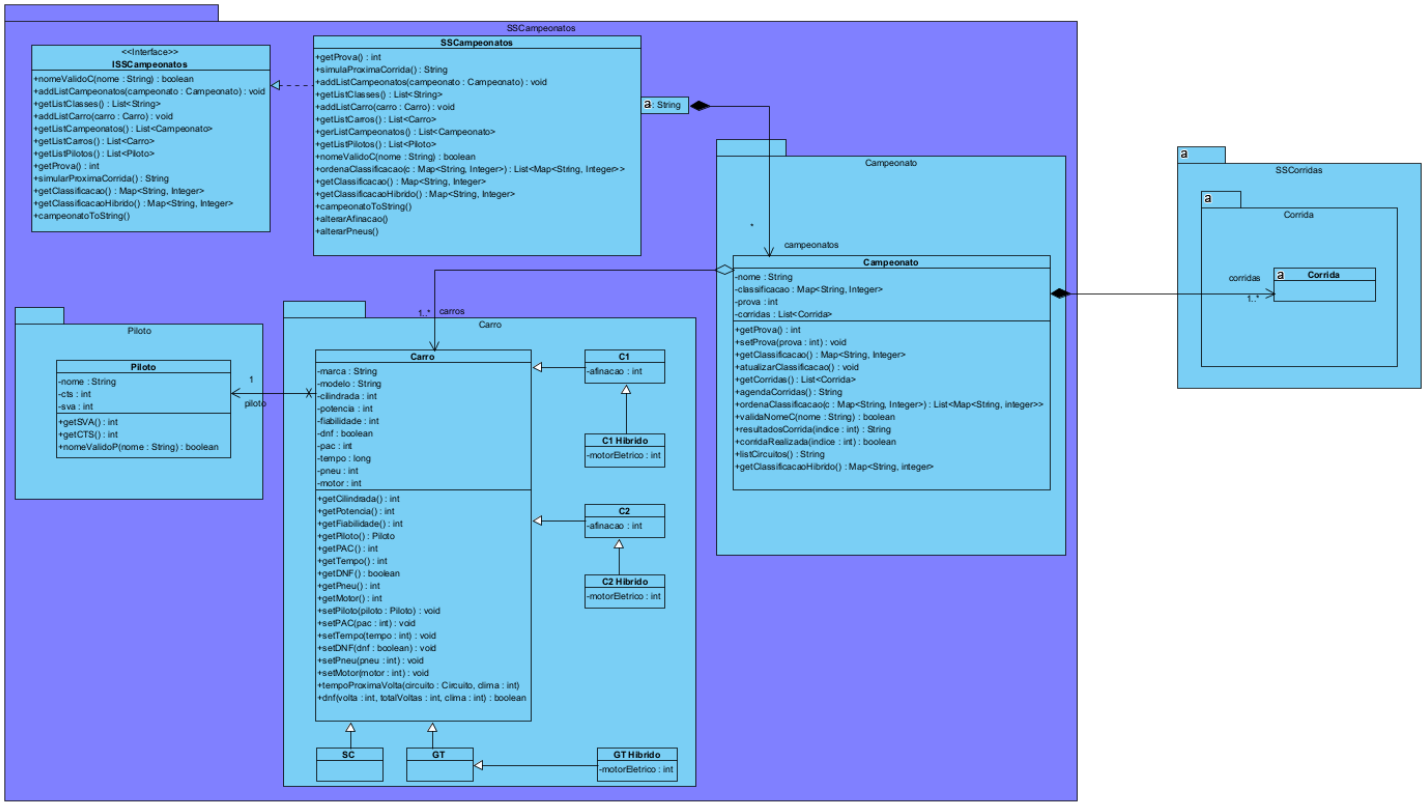


Figura 13: Representação do Diagrama de classes - Subsistema de Campeonatos.

O subsistema de Campeonatos divide-se em três classes fundamentais. A classe Campeonato, a classe Carro e a classe Piloto. Faz sentido que tanto o Carro como o Piloto estejam dentro do subsistema visto que ambos são escolhidos para um campeonato em específico. É também importante mencionar que o subsistema Campeonato relaciona-se diretamente com o subsistema Corrida.

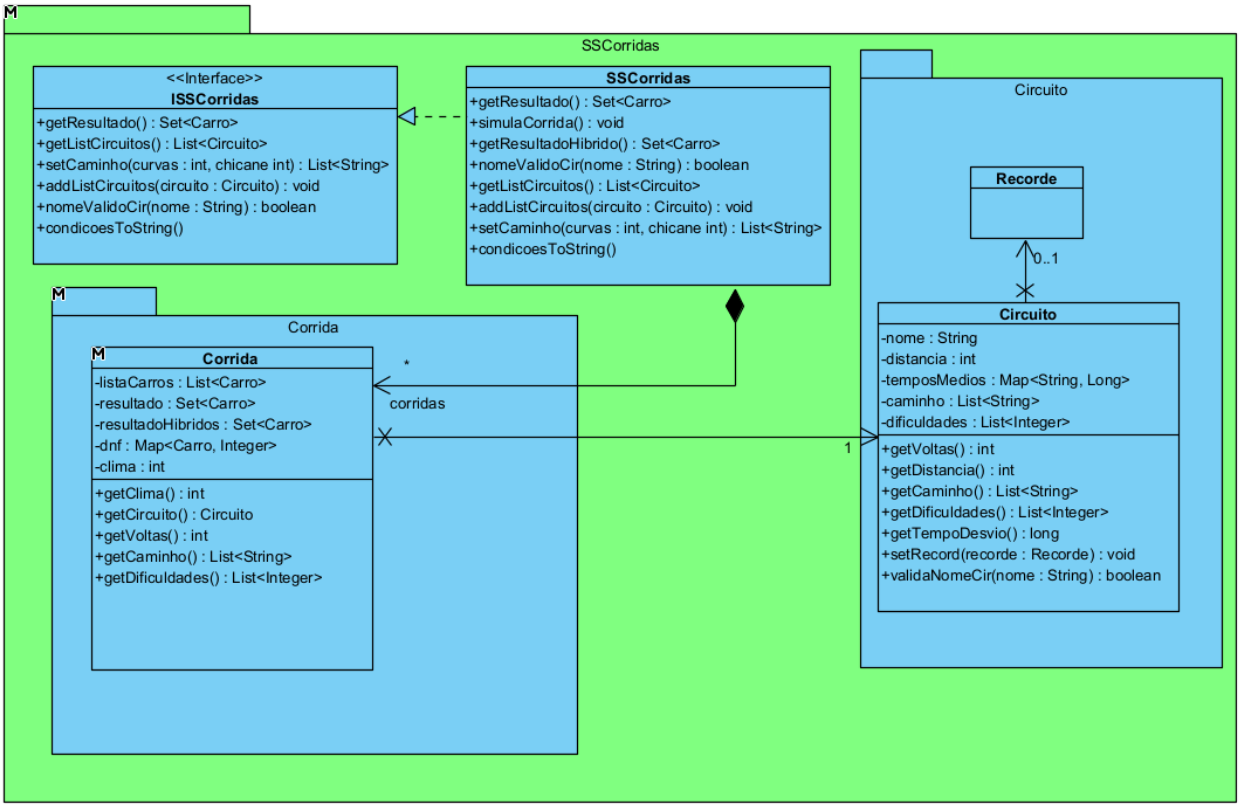


Figura 14: Representação do Diagrama de classes - Subsistema de Corridas.

O subsistema de Corridas divide-se em duas classes fundamentais. A classe Corrida e a classe Circuito. Seria possível colocar a classe Circuito dentro do subsistema Campeonato mas não faria tanto sentido porque a Corrida pode ser considerada um subsistema independente apenas com o Circuito.

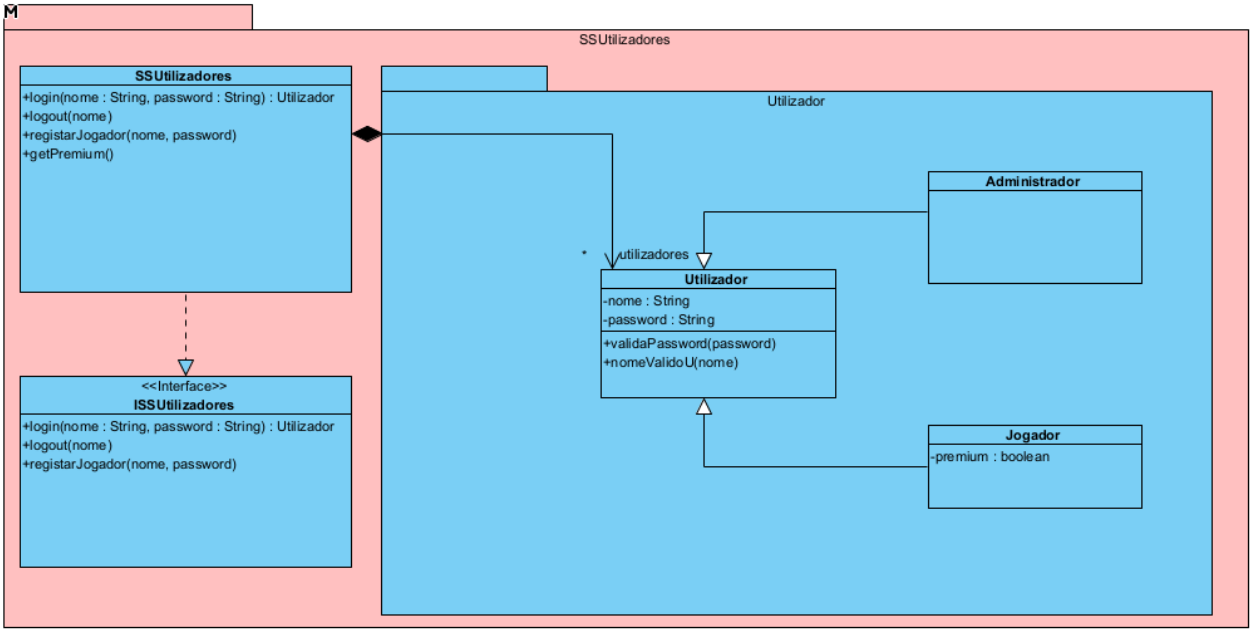


Figura 15: Representação do Diagrama de classes - Subsistema de Utilizadores.

O subsistema de Utilizadores divide-se em três classes fundamentais. A classe Administrador e a classe Jogador. O Admistrador é o responsável por criar Circuitos, Pilotos, Carros e Campeonatos e o Jogador é o utilizador padrão do RacingManager.

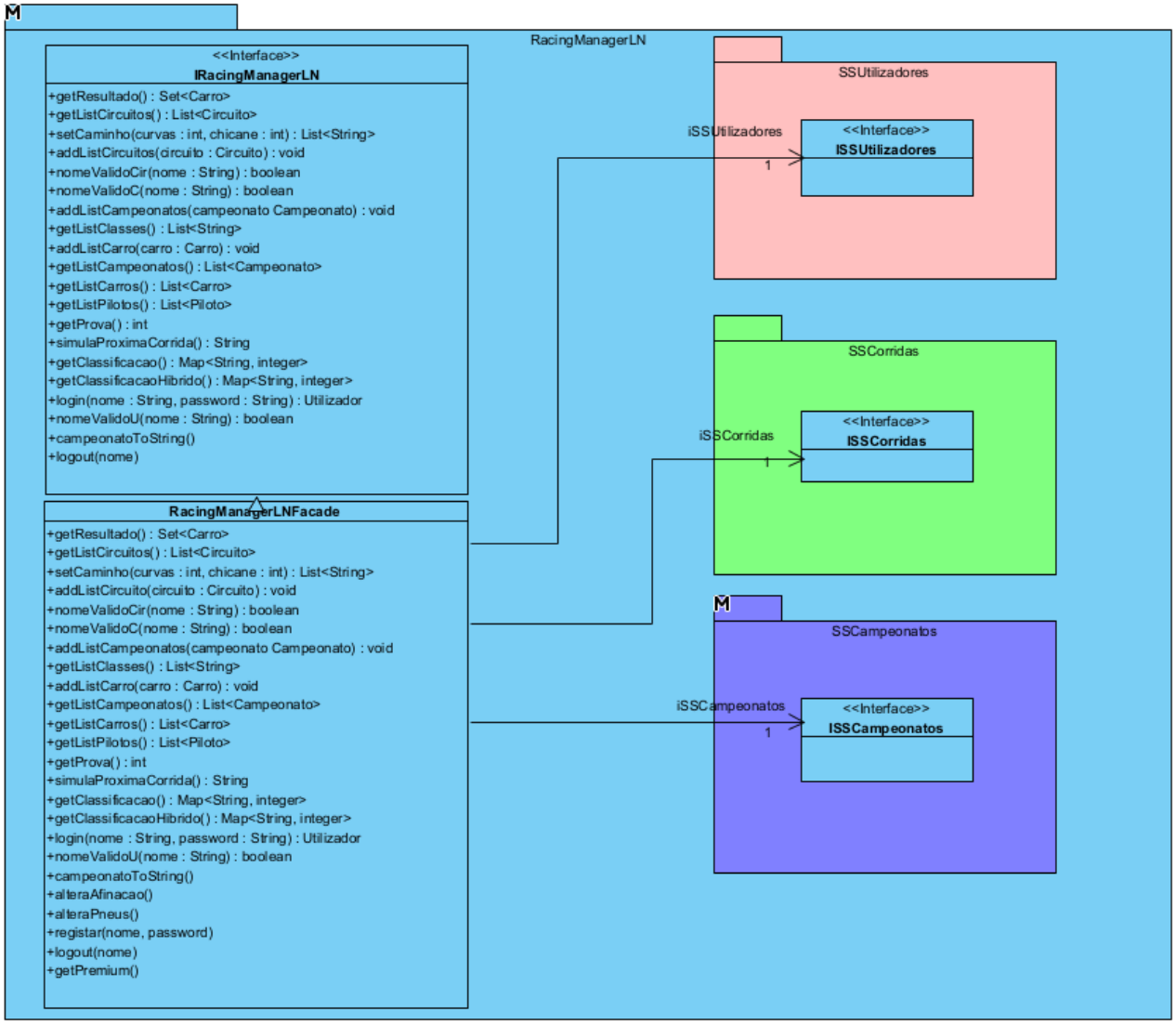


Figura 16: Representação do Diagrama de classes - Geral.

O subsistema RacingManager é o subsistema geral da aplicação. É constituído pelo RacingManagerFacade e pelas relações com os outros três subsistemas, Utilizador, Corrida e Campeonato.

7 Diagrama de Sequência

Atendendo aos Use Case do cenário 5 já apresentados na primeira fase do projeto, criamos os Diagramas de Sequência apresentados de seguida, de maneira a representarmos as iterações entre objetos através das mensagens que são trocadas entre eles, de forma ordenada ao longo do tempo, permitindo analisar assim a distribuição de responsabilidades pelas diferentes entidades (onde está a ser efetuado o processamento)

Apresentamos, de seguida, os diagramas de sequência dos métodos definidos anteriormente:

7.1 Autenticar Utilizador

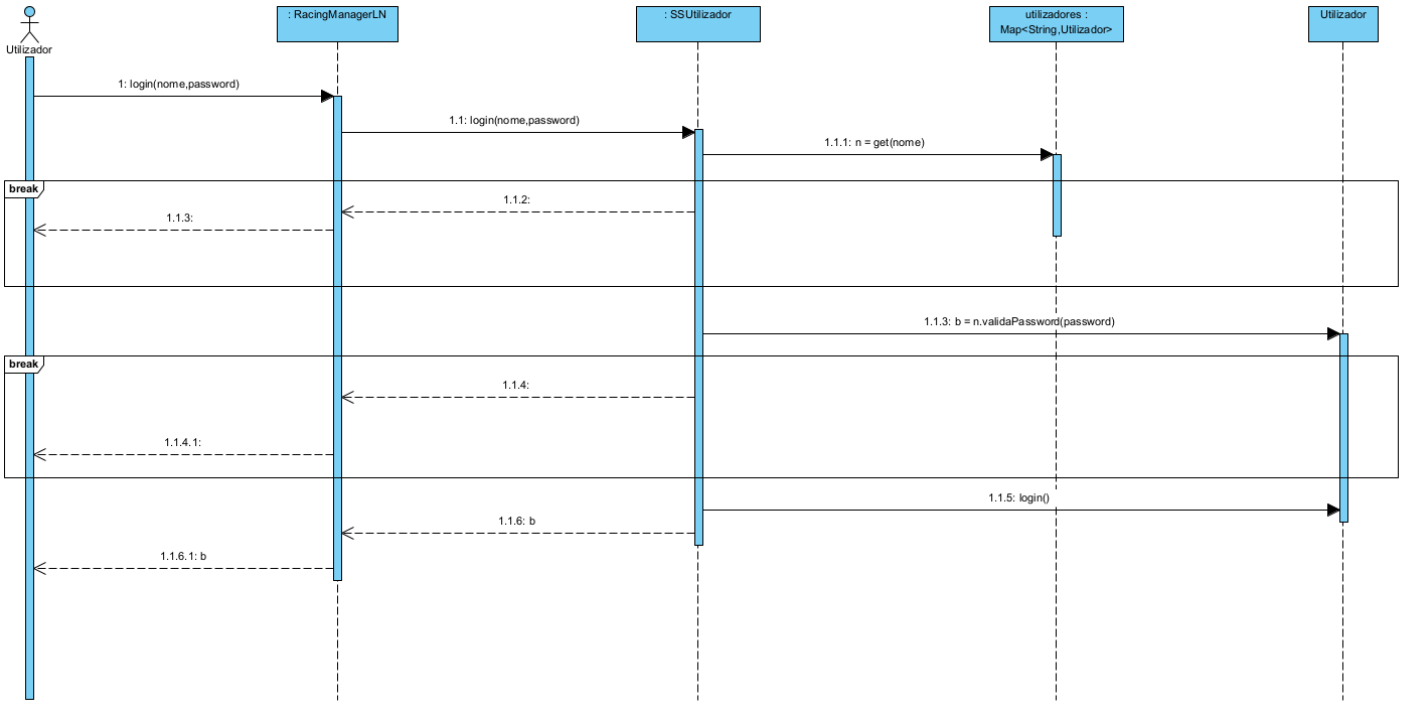


Figura 17: Representação do Diagrama de sequência - Autenticar Utilizadores.

As responsabilidades de autenticar um utilizador pertencente ao subsistema utilizadores. Primeiramente verificamos se o utilizador em questão existe no sistema, terminando o processo se este não existir. Seguidamente verificamos a password usando o método validaPassword(password), que se estiver correta é feito o login devolvendo true, se não estiver correta, o processo é terminado.

7.2 Terminar Sessão

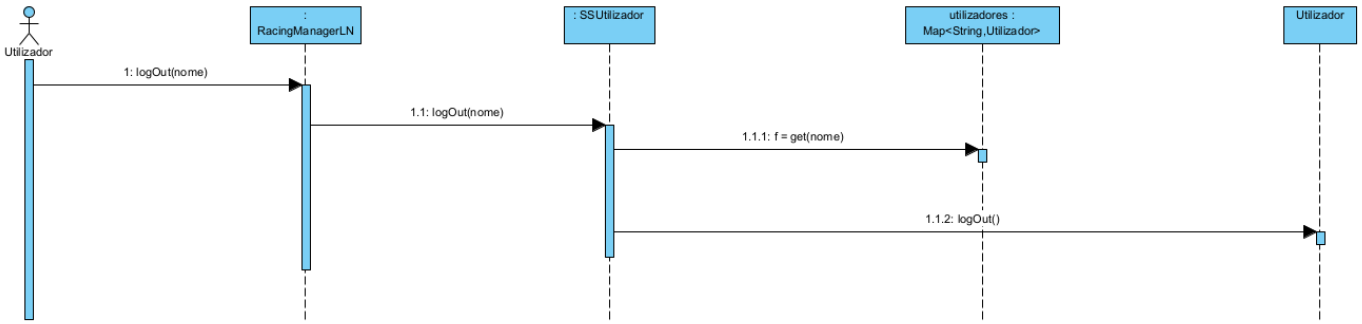


Figura 18: Representação do Diagrama de sequência - Terminar sessão.

Terminar a sessão é uma responsabilidade do subsistema de utilizadores, que deverá atualizar o registo do utilizador correspondente, de forma a que este fique marcado como sessão terminada.

7.3 Registar Utilizador

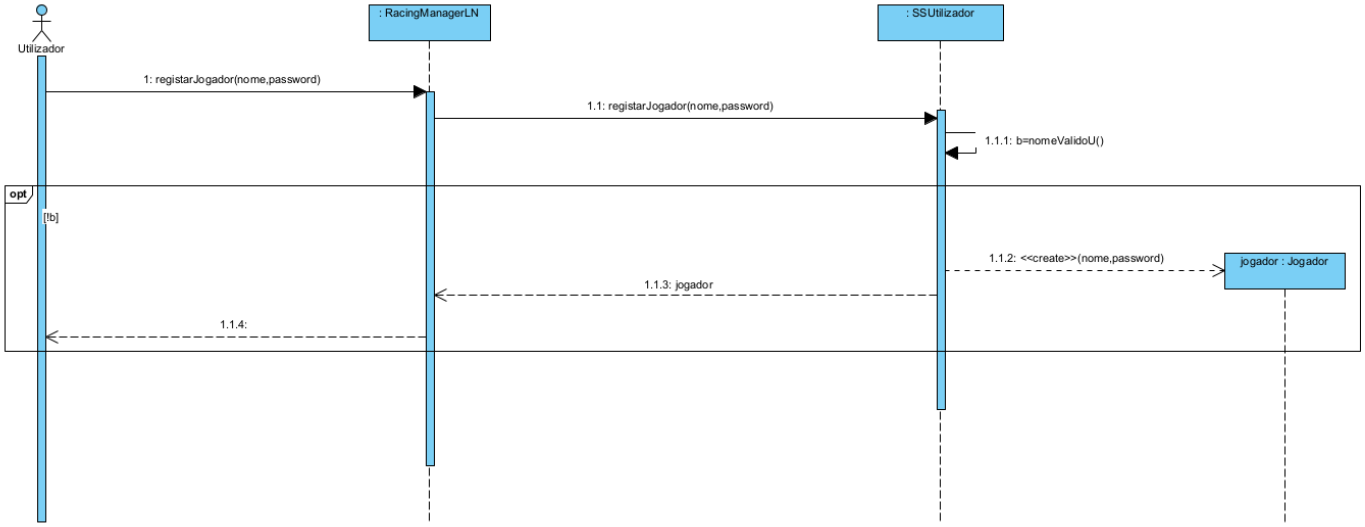


Figura 19: Representação do Diagrama de sequência - Registar Utilizador

A responsabilidade de registar um utilizador pertence ao subsistema de utilizadores, sempre que é feito um pedido de registo, primeiro é validado o nome, para confirmar se não existe nenhum igual, de seguida o sistema cria um jogador novo, sendo que as contas de administrador não podem ser criadas desta forma.

7.4 Apresentar lista de campeonatos disponíveis

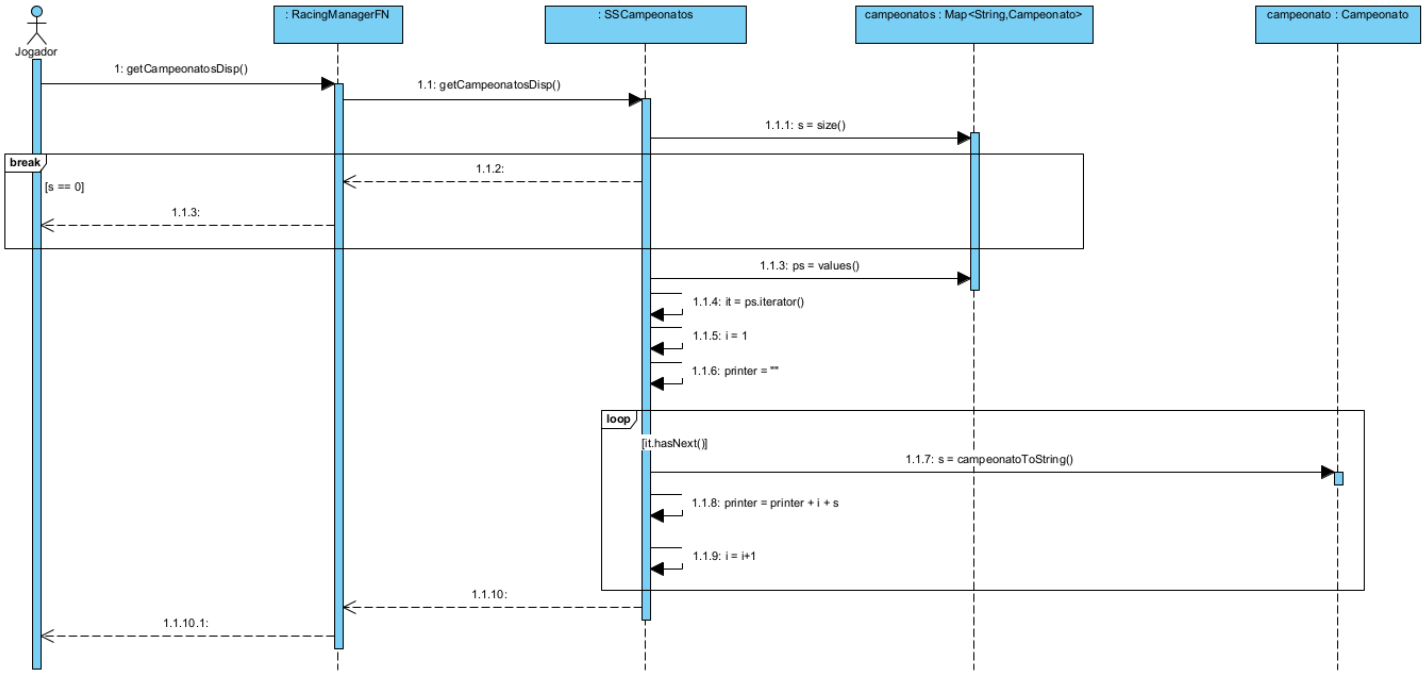


Figura 20: Representação do Diagrama de sequência - Apresentar lista de campeonatos disponíveis.

A apresentação da lista de campeonatos a um jogador, é responsabilidade do subsistema de campeonatos, assim que o pedido é feito, verifica que existe um ou mais campeonatos no sistema, caso contrário termina o processo, de seguida cria uma *String* que guarda a informação de cada campeonato, com um identificador no inicio de cada um, para facilitar a identificação da escolha do jogador.

7.5 Apresentar lista de corridas de um campeonato

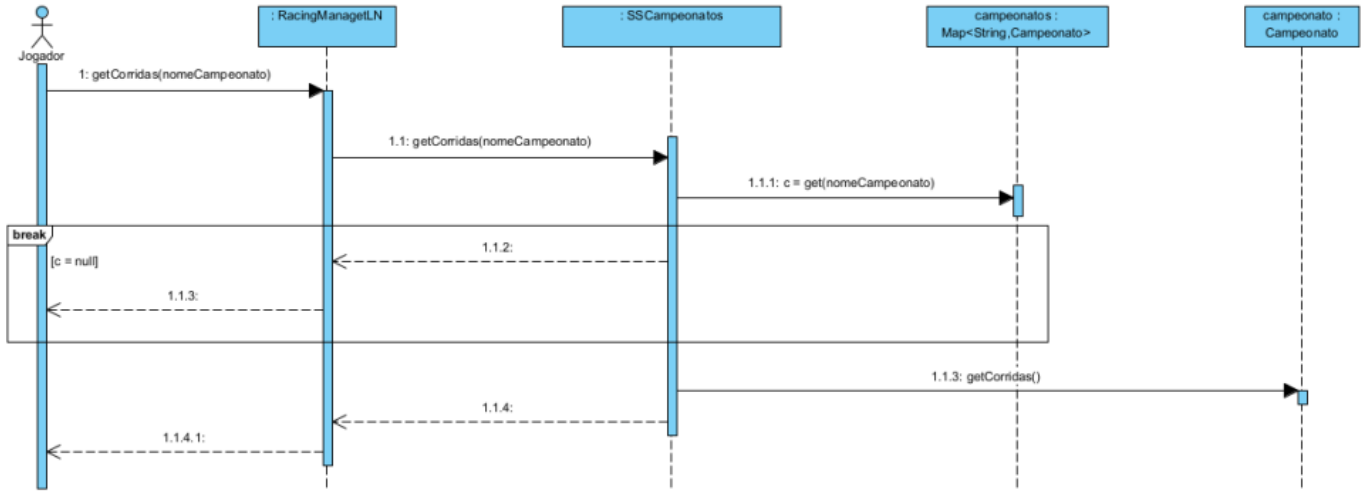


Figura 21: Representação do Diagrama de sequência - Apresentar lista de corridas de um campeonato.

A apresentação da lista de corridas de um campeonato a um jogador, é responsabilidade do subsistema de campeonatos. Quando um jogador escolhe um campeonato, é apresentada uma lista de corridas que o constituem, para isso, acedemos a um campeonato especifico, e se esse existir, retornamos a lista em questão, caso não exista, terminamos o processo.

7.6 Apresentar lista de carros disponíveis

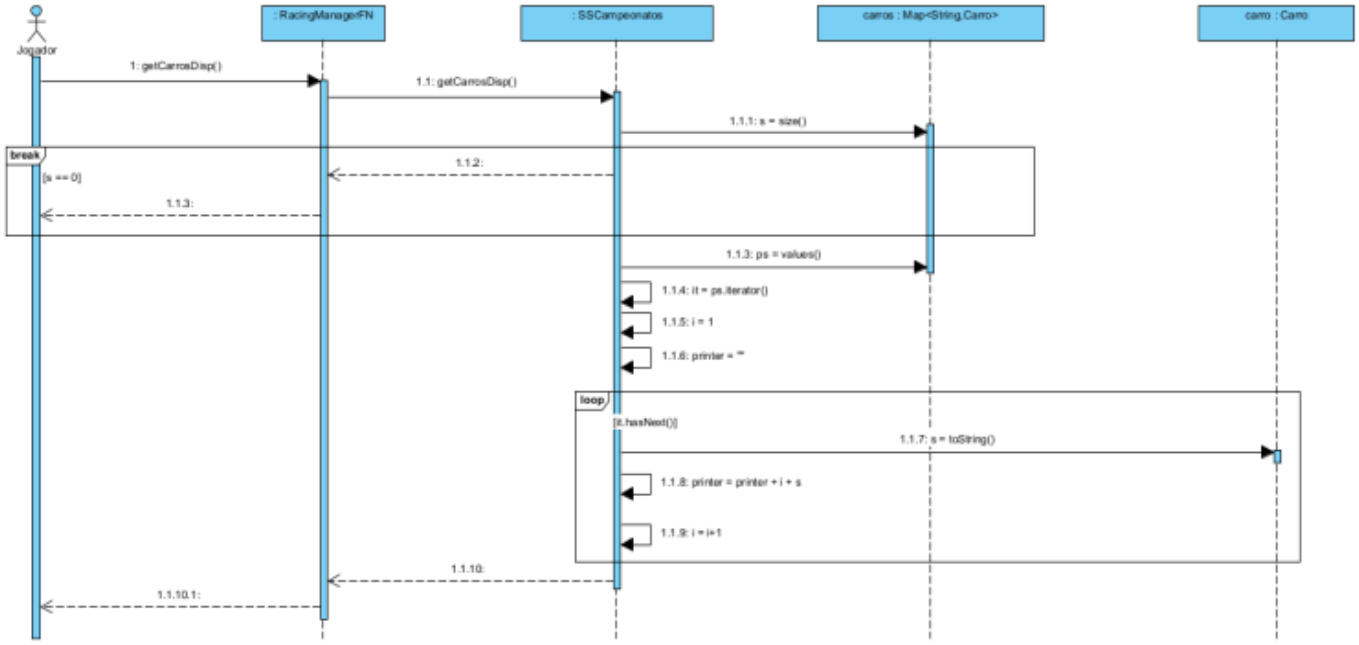


Figura 22: Representação do Diagrama de sequência - Apresentar lista de carros disponíveis.

A apresentação da lista de carros disponíveis a um jogador, é responsabilidade do subsistema de campeonatos. Da mesma forma que é feito ao apresentar campeonatos, é verificado se o numero de carros disponíveis é maior ou igual a 1, terminando o processo caso contrário, e de seguida são percorridos todos os carros, um a um, acrescentando a uma *String* todas as características que o definem, antecipado por um identificador. Essa string é devolvida ao jogador.

7.7 Apresentar lista de pilotos disponíveis

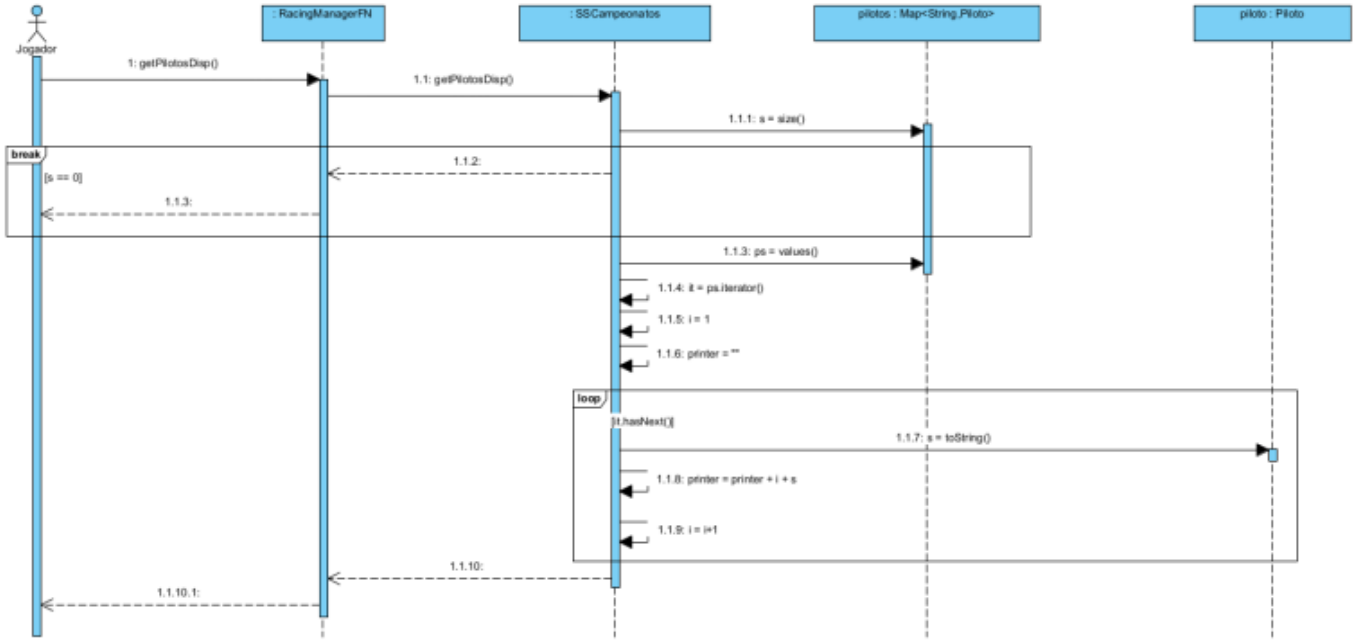


Figura 23: Representação do Diagrama de sequência - Apresentar lista de pilotos disponíveis.

A apresentação da lista de carros disponíveis a um jogador, é responsabilidade do subsistema de campeonatos. Com a mesma ideia seguida na descrição do diagrama anterior, começamos por verificar se o numero de pilotos disponíveis é maior ou igual a 1, terminando o processo caso contrário, e de seguida são percorridos todos os pilotos, um a um, acrescentando a uma *String* todas as características que o definem, antecipado por um identificador. Essa string é devolvida ao jogador.

7.8 Registar num campeonato

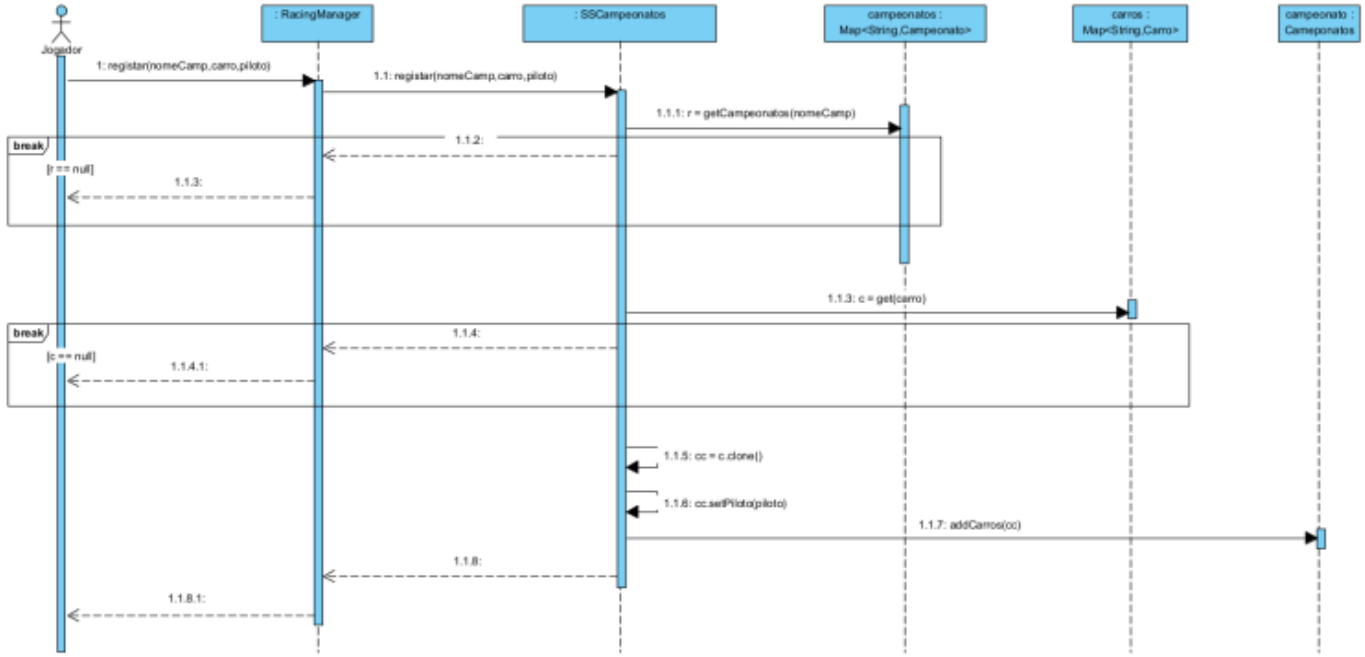


Figura 24: Representação do Diagrama de sequência - Registar num campeonato.

O registo num determinado campeonato, é responsabilidade do subsistema de campeonatos. Após ser apresentada a informação relativa ao campeonato, e depois de terem sido escolhidos ambos o carro e o piloto, é chamado o método **registar(nomeCamp,carro,piloto)**. De seguida, confirmamos que este campeonato existe, terminando o processo se não existir, caso contrário, vamos aceder ao carro escolhido, confirmando também a sua validade, e criamos um clone, ao qual associamos um piloto. Por fim, adicionamos esse carro com um piloto associado à lista de carros participantes do campeonato em questão.

7.9 Simula campeonato

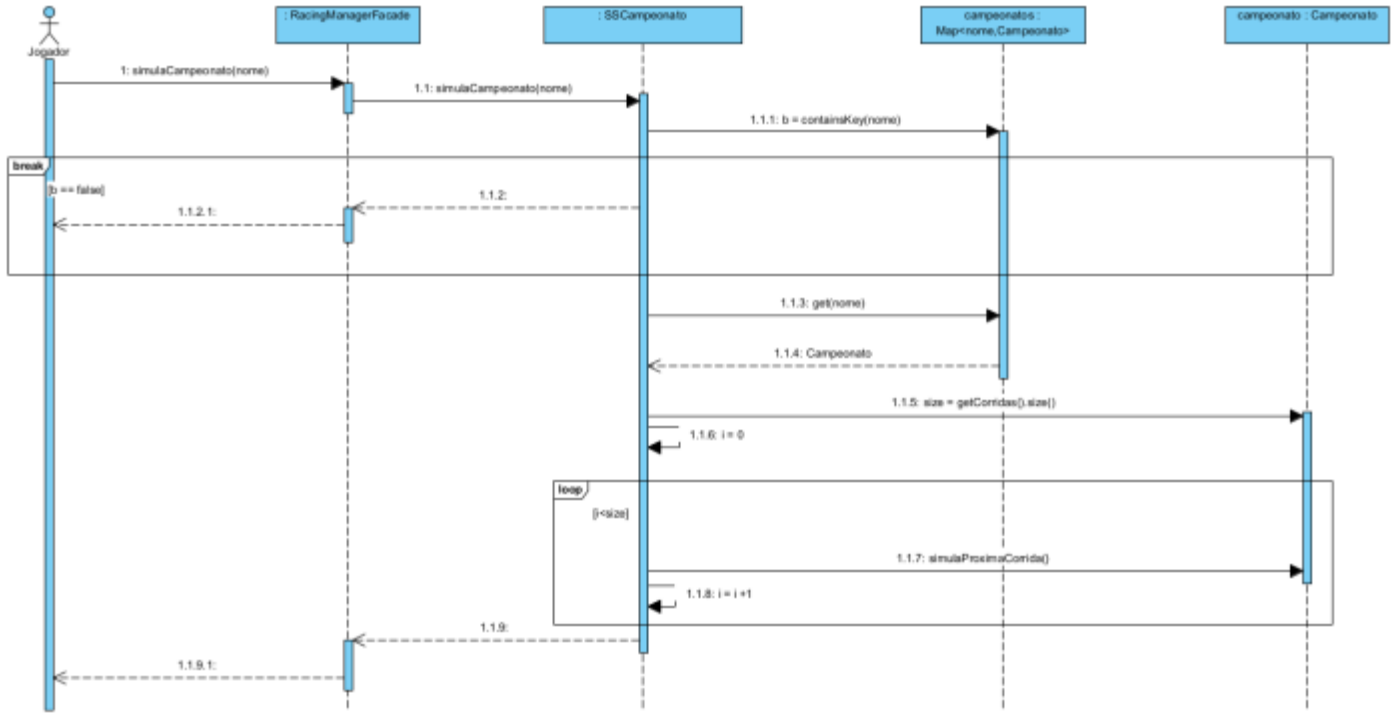


Figura 25: Representação do Diagrama de sequência - Simula Campeonato.

A fase de simulação de um campeonato, é responsabilidade do subsistema de campeonatos. Depois de todos os registos terem sido efetuados, damos inicio á sua simulação, com o auxilio do método **simulaCampeonato(nome)**, que confirma a existência desse campeonato, terminando o processo caso não exista, de seguida entra em ciclo, e executa um numero de vezes igual ao numero de corridas que constituem esse campeonato, e por cada iteração, usa o método **simulaProximaCorrida()** para garantir que todas as corridas são finalizadas antes de terminar um campeonato.

7.10 Apresenta condições

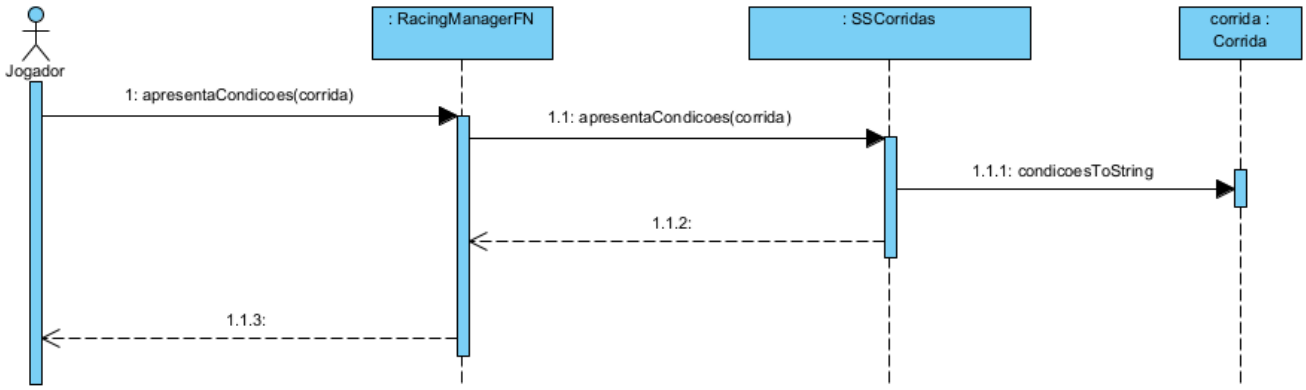


Figura 26: Representação do Diagrama de sequência - Apresenta condições.

A apresentação das condições de uma corrida, faz parte do subsistema de corridas. Antes de fazer qualquer alteração no carro, é preciso apresentar as condições da próxima corrida, para isso usamos o método **condicoesToString()** que devolve um string com a informação necessária.

7.11 Alterar afinação

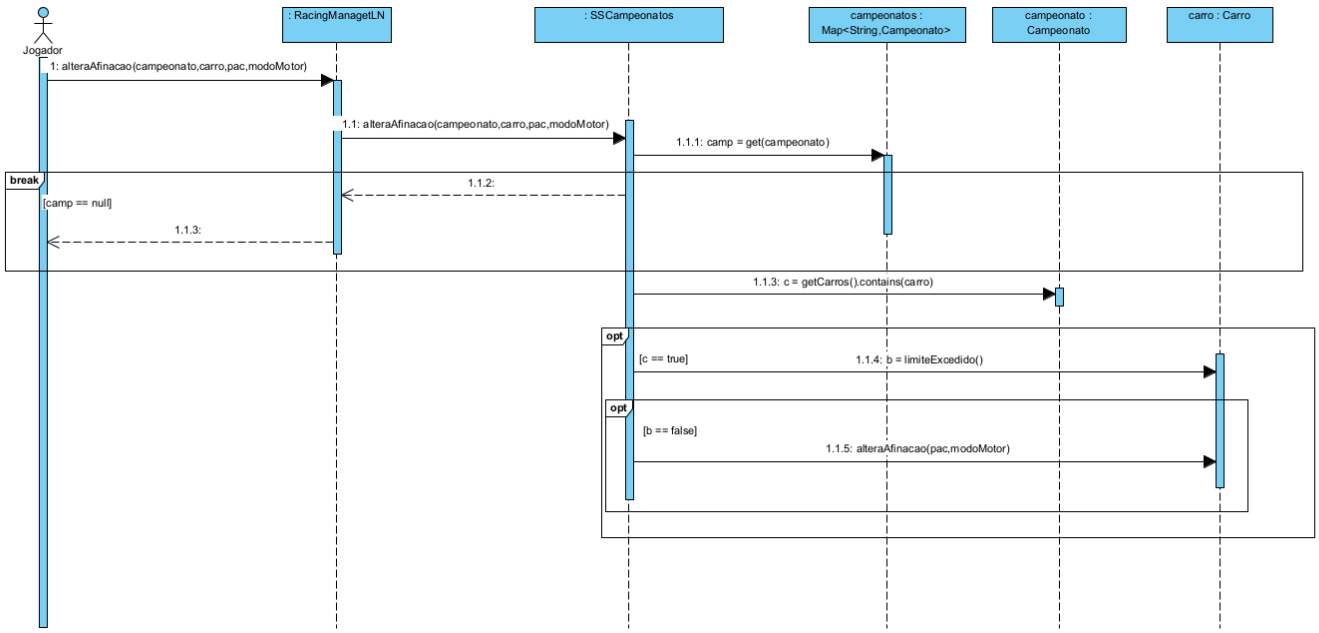


Figura 27: Representação do Diagrama de sequência - Alterar afinação.

A responsabilidade de alterar a afinação do carro pertence ao subsistema de campeonatos. É executado o método **alteraAfinacao(campeonato, carro, pac, modoMotor)**. O sistema busca o campeonato e carro inseridos. Após ter o campeonato e o carro selecionados, o sistema verifica se o jogador pode alterar a afinação do carro consoante o número de alterações já efetuadas pelo mesmo. Caso o jogador já tenha atingido o limite de alterações, o sistema cancela a alteração da afinação, caso contrário, altera a afinação do carro consoante o input do utilizador.

7.12 Alterar pneus

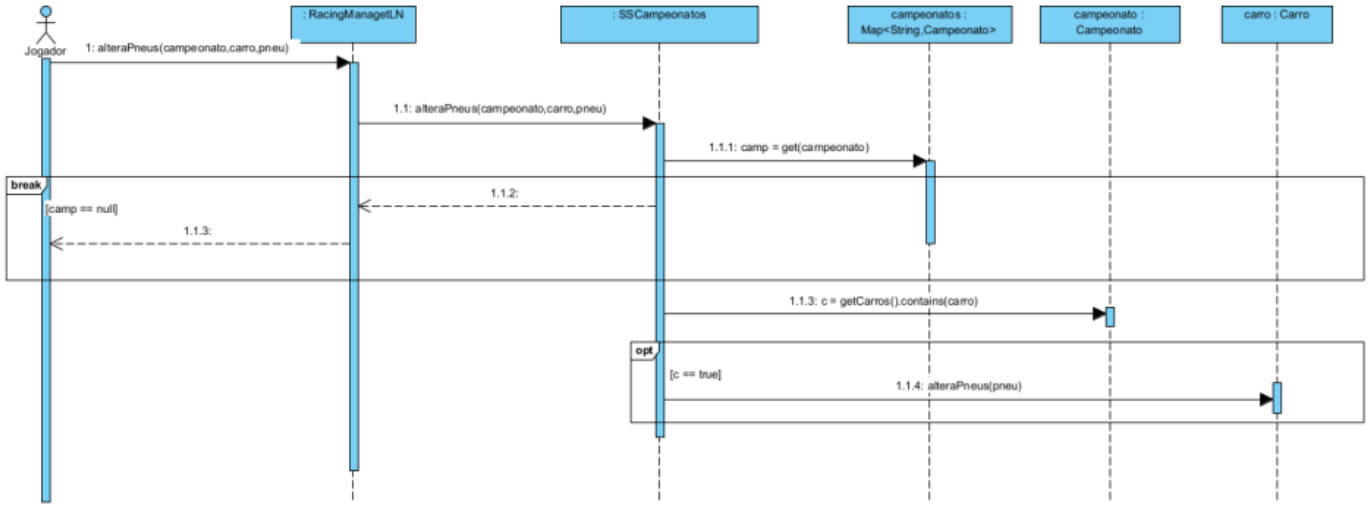


Figura 28: Representação do Diagrama de sequência - Alterar pneus.

A responsabilidade de alterar os pneus pertence ao subsistema de campeonatos. Começa por ser chamado o método **alteraPneus(campeonato,carro,pneu)**, e o sistema começa por ir buscar o campeonato em questão, terminando o processo caso não exista, de seguida faz a alteração no carro selecionado.

7.13 Simular Corrida

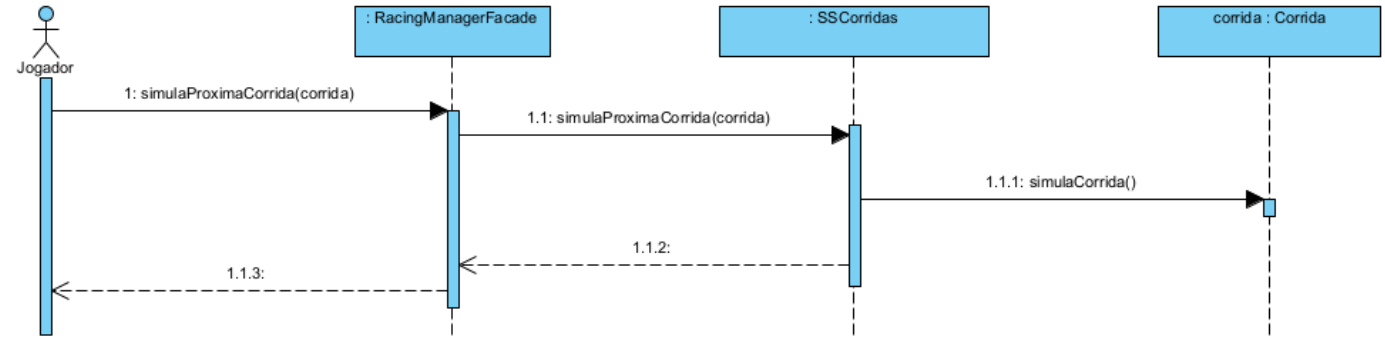


Figura 29: Representação do Diagrama de sequência - Simular corrida.

A responsabilidade de simular uma corrida pertence ao subsistema de corridas. Sempre que um campeonato simula a próxima corrida, é feito uma chamada ao método simulaCorrida(), que simula uma corrida independentemente do campeonato, e apenas retorna os resultados.

8 Diagrama de Packages

De seguida apresentamos o nosso Diagrama de Packages onde agrupamos as classes em pacotes distintos de maneira a termos uma visão mais geral e facilitada do sistemas e dos packages utilizados e as relações entre eles.

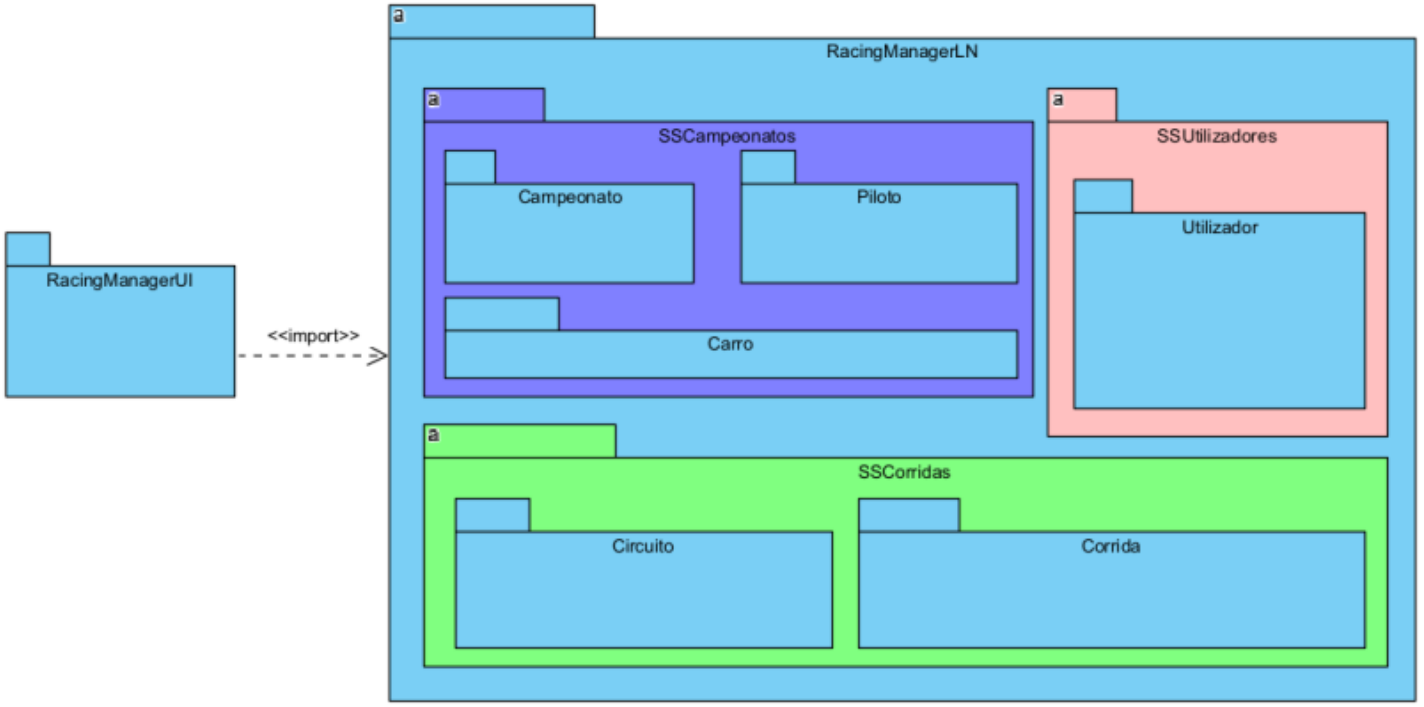


Figura 30: Representação do Diagrama de Packages

9 Conclusão

Com o processo de desenvolvimento da segunda fase deste projeto, foi necessária a alteração de alguns aspetos em falta na análise de requisitos. Falhas que, se tornaram bastante evidentes à medida que fomos construindo a modelação conceptual da solução.

Com a introdução desta segunda fase, foi também bastante útil analisar, perceber e alterar o código inicial de uma solução para o projeto disponibilizado pelo docente. Mesmo não concordando com muitos métodos dessa solução, com auxilio à alteração da mesma, foi-nos possível imaginar uma futura implementação que facilitou o desenvolvimento dos diagramas necessários.

Relativamente a modelação, foi desenvolvida uma segunda fase, usando a primeira como um ponto de partida, ao qual tentamos dar continuidade à solução visionada, alterando apenas o que consideramos apropriado com a nova informação que obtemos. Para o desenvolvimento de ambas as fases, aproveitou-se a linguagem UML que mune o grupo com as ferramentas necessárias para representar os conceitos com o nível de abstração suficiente, como recomendado pelos docentes da unidade curricular.

Por fim, estamos ainda cientes das lacunas presentes na forma como iremos implementar uma solução final, mas assim como estas duas primeiras fases foram bastante esclarecedoras, acreditamos que numa próxima fase seremos capazes de perceber os aspetos que estão atualmente em falta.