Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia informática

Unidade Curricular de Desenvolvimento de Sistemas de Software

Ano Letivo de 2022/2023

Grupo 31

Inês Ferreira (A97372)

José Ferreira (A97642)

Joana Branco (A96584)

Marta Sá (A97158)

Robert Szabo (A91682)

URL do repositório: https://github.com/venicexbish/DSS

Braga, 22 de novembro de 2022

Índice

1. Introdução

Este relatório foi desenvolvido no âmbito da Unidade Curricular de Desenvolvimento de Sistemas de Software onde foi-nos proposta a criação de um sistema, semelhante ao *F1 Manager,* capaz de simular campeonatos de automobilismo. Assim, a aplicação tem como objetivo criar um ambiente de jogo onde os seus participantes podem competir entre si.

Na primeira fase foi feito o levantamento dos requisitos apoiados pelos cenários fornecidos no enunciado. De seguida, os requisitos passaram por uma análise e, posteriormente, foi criado um modelo com base neles. Para a conceção e desenvolvimento da aplicação foram criados modelos em UML utilizados para especificar o Modelo de Domínio e o Modelo de Use Cases.

Esta segunda fase, destina-se à implementação de uma arquitetura conceptual do projeto capaz de suportar os requisitos analisados na primeira fase. Para isso, desenvolveu-se os modelos necessários para descrever o comportamento do sistema, entre eles, um Diagrama de Pacotes, um Diagrama de Componentes, um Diagrama de Classes e Diagramas de Sequências.

2. Objetivos

A segunda fase do projeto tem como objetivo efetuar a análise dos requisitos que foram, anteriormente, identificados a partir de uma arquitetura conceptual baseada em diferentes tipos de diagramas sendo possível descrever o comportamento que o sistema deve adotar.

O Diagrama de Pacotes tem por definição a redução na complexidade da organização das classes desde o ponto inicial. Desta forma, é possível descrever desde logo a informação importante para a evolução do sistema. Posteriormente, o Diagrama de Classes será incorporado neste último representando assim *packages* de classes.

Já o Diagrama de Componente explora uma parte modular do sistema e os comportamentos das interfaces propostas. São as interfaces que indicam os serviços requeridos ou fornecidos pelo componente e, as portas indicam os pontos de interação com esses mesmo componentes.

Quanto aos Diagramas de Sequência têm como objetivo retratar as interações entre os objetos através das mensagens que são trocadas entre eles. A ordem temporal dos acontecimentos é crucial neste diagrama, sendo possível a distribuição de tarefas pelas diferentes entidades.

3. Descrição

No enunciado do trabalho é nos proposto criarmos um sistema que permita a criação de campeonatos automobilísticos, com vários componentes. O sistema que vai ser construído deverá assegurar o processo desde a criação de corridas, carros, circuitos, utilizadores, etc, até termos toda a constituição de campeonatos e podermos realizar vários desses mesmos.

Partindo do ponto da fase anterior, os Use Cases do sistema foram divididos numa folha de Excel sendo assim possível descrevê-los em diferentes métodos capazes de executar o comportamento pretendido do sistema. Agrupando estes mesmos métodos em subsistemas deu-se origem a um Diagrama de Pacotes.

Tendo em conta a informação obtida até este ponto, todos os métodos foram agrupados à classe que lhes diz respeito obtendo-se assim um Diagrama de Classes. Este último refere todos os atributos, operações e relações necessárias para que o sistema execute do modo pretendido. Neste contexto, decidiu-se agrupar este último diagrama referido e o Diagrama de Pacotes num só por termos de simplicidade e para melhor compreensão.

O Diagrama de Componentes é gerado com base nos diagramas produzidos até ao momento e tem como objetivo tornar observável todo o sistema, incluindo todas as interfaces com o utilizador, o ambiente de simulação e a base de dados.

Por último, foram desenvolvidos os Diagramas de Sequência, focados principalmente no cenário 5 do enunciado, onde é explicado todo o ato de jogar. Nestes diagramas é dado o foco no ordenamento temporal das trocas de mensagens.

4. Identificação de subsistemas e APIs

Ao analisar os Use Cases, definidos na primeira fase do projeto, identificaram-se as seguintes funções que são responsáveis pela lógica de negócio a implementar.

Use Case	**Funcionalidades**
Criar Conta	Registar nome de utilizador, palavra-passe; Escolher modo de utilizador; Verificar nome de utilizador da conta;
Fazer login	Verificar existência de username; Verificar se Password corresponde à correta;
Criar Campeonato	Inserir nome para campeonato; Escolher os circuitos;
Criar Circuito	Inserir nome do circuito, valor de distância, nº de chicanes e nº de voltas à corrida; Calcular o número de retas; Atribuir GDU a cada elemento;
Criar Carro	Inserir categoria, marca, modelo, cilindrada e potência; Verificar categoria do carro; Atribuir a fiabilidade ao carro; Inserir se o carro será híbrido ou não; Definir o perfil aerodinâmico do carro;
Criar Piloto	Inserir nome do piloto, valor de CTS, valor de SVA;
Configurar Campeonato	Apresentar e escolher um campeonato lista de campeonatos; Apresentar informação do campeonato; Apresentar e escolher um carro da lista de carros; Apresentar e escolher um piloto da lista de pilotos; Registar participante no campeonato;
Configurar Corrida	Apresentar a possibilidade de afinar; Escolher pneus; Verificar condições para afinar; Definir downforce; Escolher modo de motor;
Simular Corrida	Confirmar a versão do jogo; Calcular ultrapassagens, despistes e avarias; Apresentar resultado da volta; Verificar se há mais voltas; Apresentar resultado da corrida; Apresentar resultado do campeonato Verificar se há mais corridas;

	Calcular ultrapassagens, despistes e avarias; Retirar o participante; Modificar a classificação do participante;
Consultar Resultado final	Apresentar a possibilidade de login; Adicionar pontos à classificação total do campeonato; Atualizar posições no campeonato e mostrar ranking final.

Em anexo são apresentados os Use Cases descritos e organizados numa folha de Excel tendo como foco o cenário 5.

Uma vez identificadas as funcionalidades, prosseguimos para o próximo passo e transformamos essas mesmas para métodos capazes de ser implementados em linguagem Java, exprimido no Diagrama de Classes (secção 6). Dividindo-os pelos subsistemas mais importantes para o nosso sistema, como apresentado na secção 5.

5. Diagrama de Componentes

Após efetuar na primeira fase a análise dos requisitos do sistema e do domínio do problema, deu-se início ao desenvolvimento do Diagrama de Componentes. Assim, definiu-se os subsistemas: Circuito, Campeonato, Conta e Carro que devem implementar a respetiva interface com os métodos associados.

Desta forma, permite-se a interação entre subsistemas sem que seja necessário conhecer a organização interna dos mesmos.

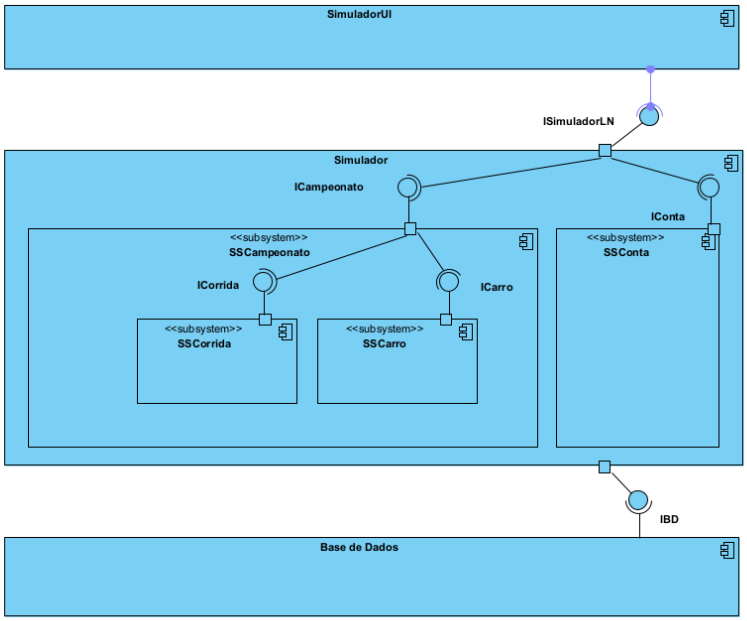

Figura 1 – Diagrama de Componentes

6. Diagrama de Classes

Após a identificação das API e subsistemas, identificaram-se, também, os componentes do sistema, contudo falta definir as classes compostas por variáveis de instância e métodos, existentes no sistema, bem como a forma que elas se interligam entre si. Para resolver essa questão, procedeu-se à realização de diagramas de classes para cada componente identificado.

Neste tópico é também incluído o Diagrama de Pacotes estando este incorporado no Diagrama de Classes.

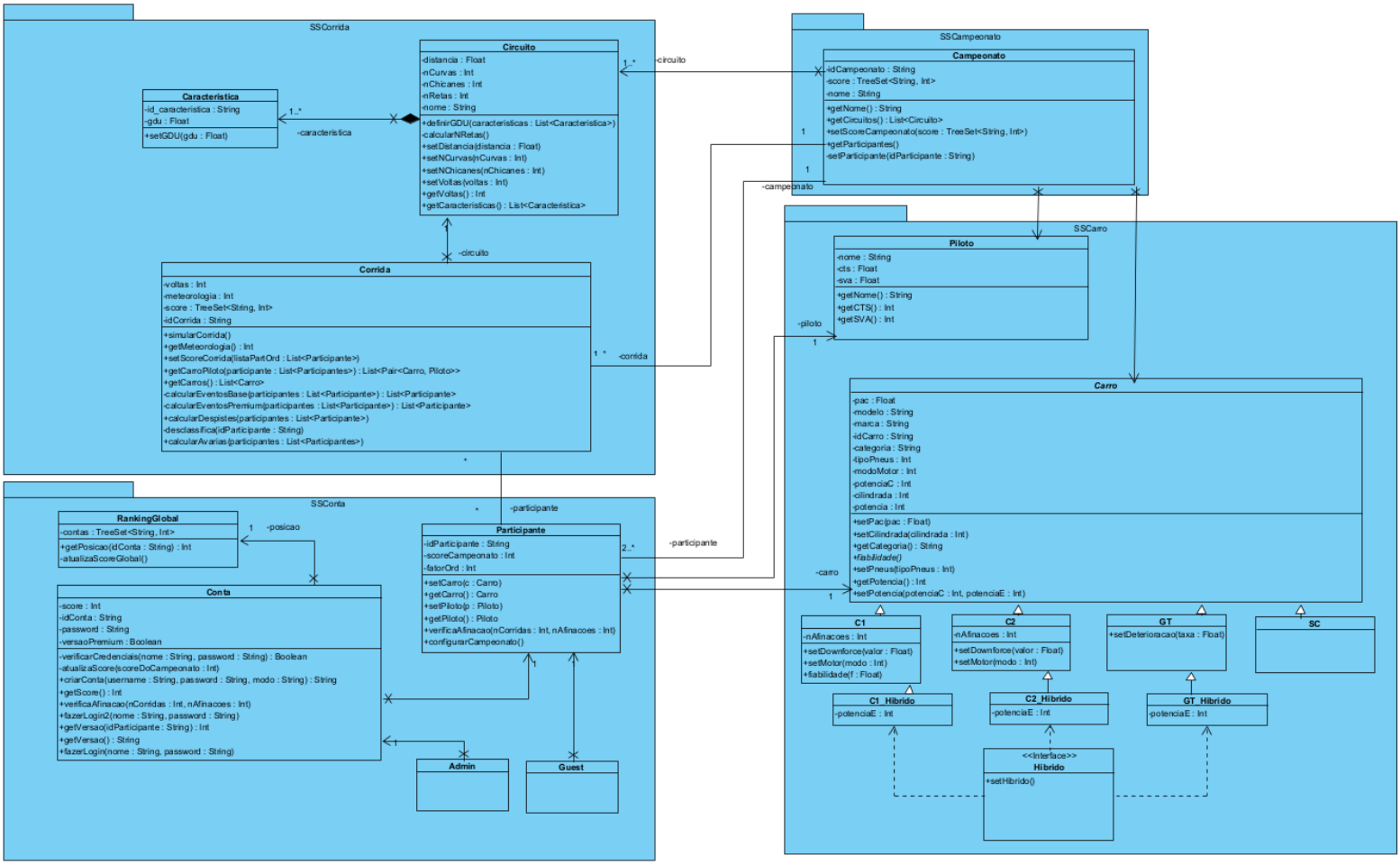

Figura 2 – Diagrama de classes

7. Diagrama de Sequências

Com base nos Use Cases e nas respetivas especificações, na análise de requisitos e de uma boa conceção da arquitetura e do comportamento do sistema, construímos Diagrama de Sequências.

Estes diagramas permitem-nos compreender com maior pormenor o fluxo de eventos necessário para o bom funcionamento do nosso sistema. Assim permitir, futuramente, programar o software com uma maior facilidade visto que foi construída uma ideia consistente e simplificada das interações dos seus componentes.

7.1. Registar participante

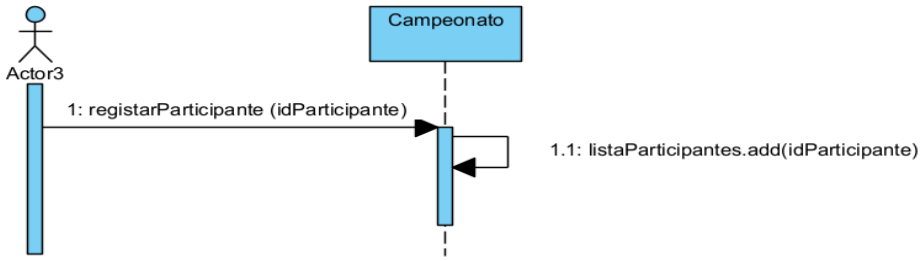

Figura 3 - registarParticipante

7.2. Verificar credenciais

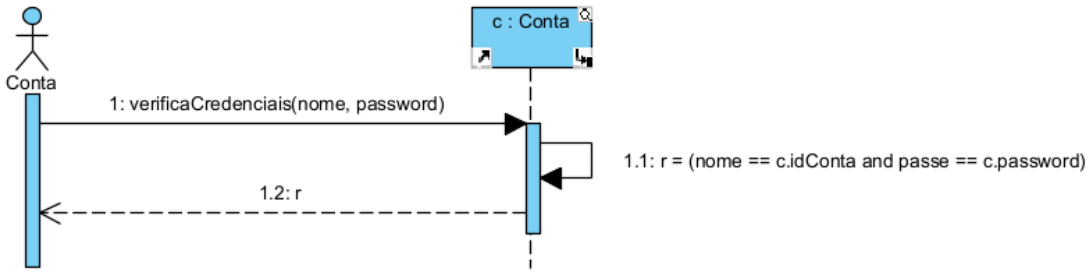

Figura 4- verificarCredenciais

7.3. Fazer login

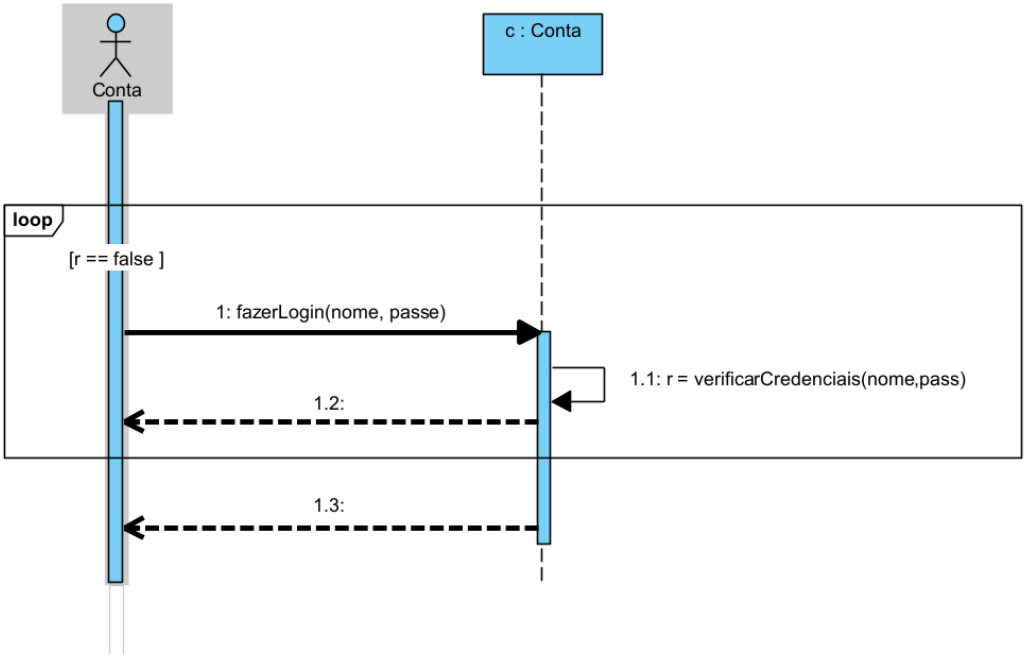

Figura 5- fazerLogin

7.4. Verificar afinação

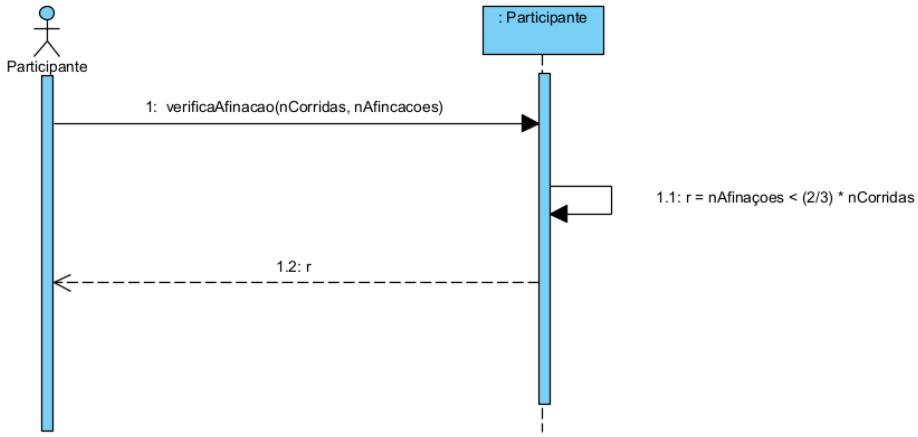

Figura 6 – verificarAfinação

7.5. Simular corrida

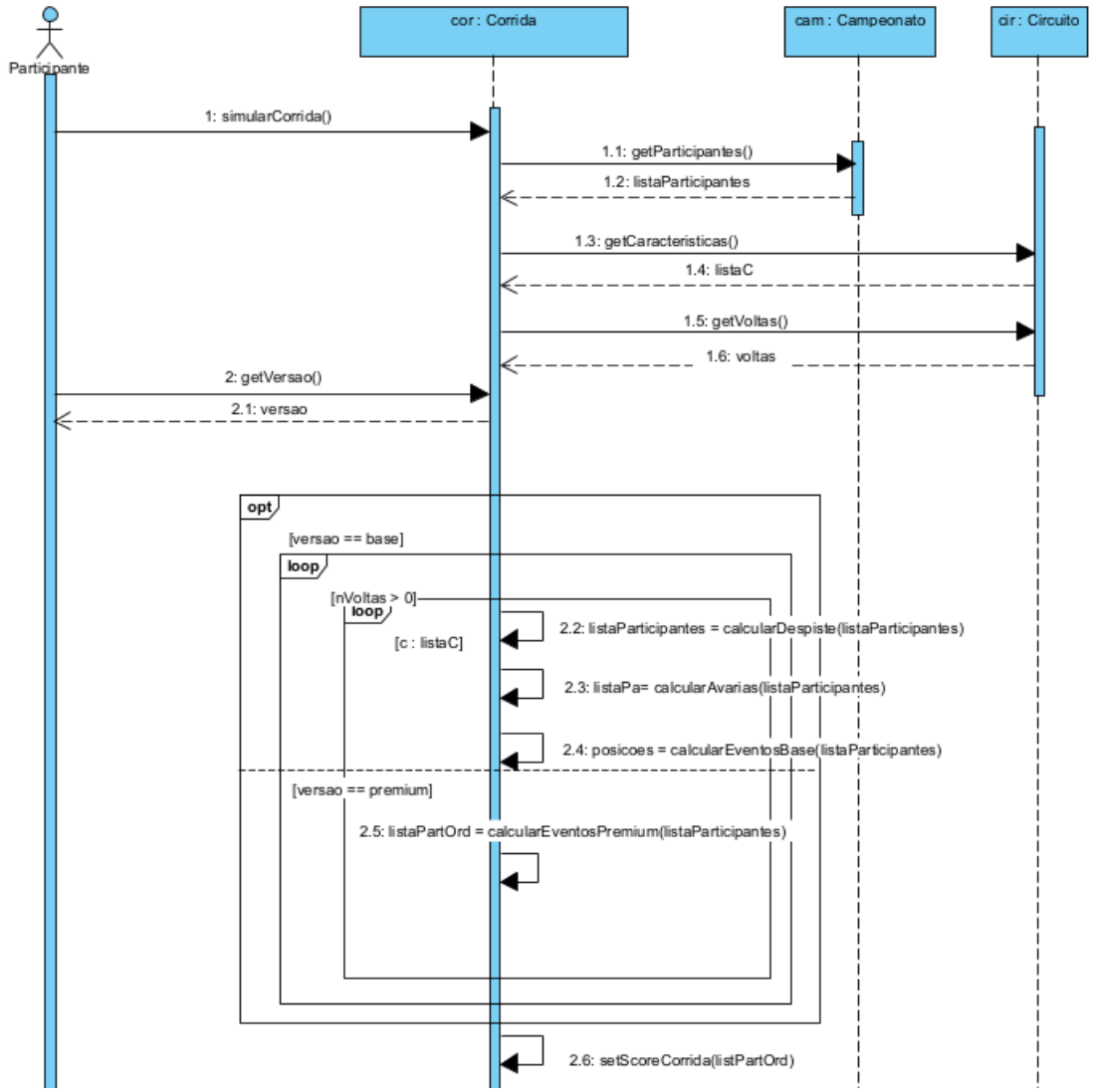

Figura 7- simularCorrida

7.6. Atualizar score

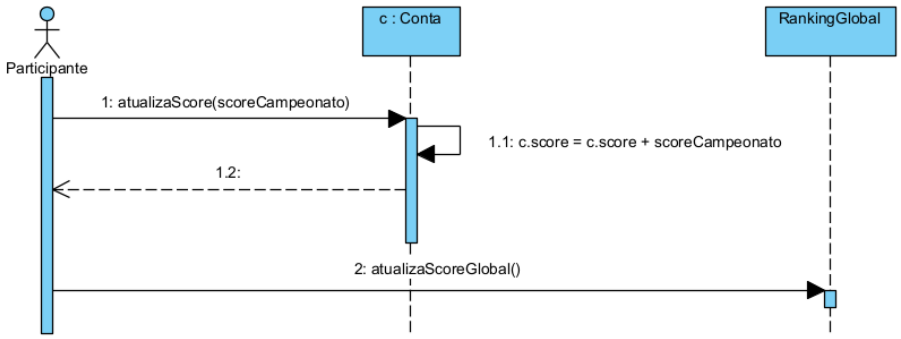

Figura 8 – atualizarScore

7.7. Desclassificar participante

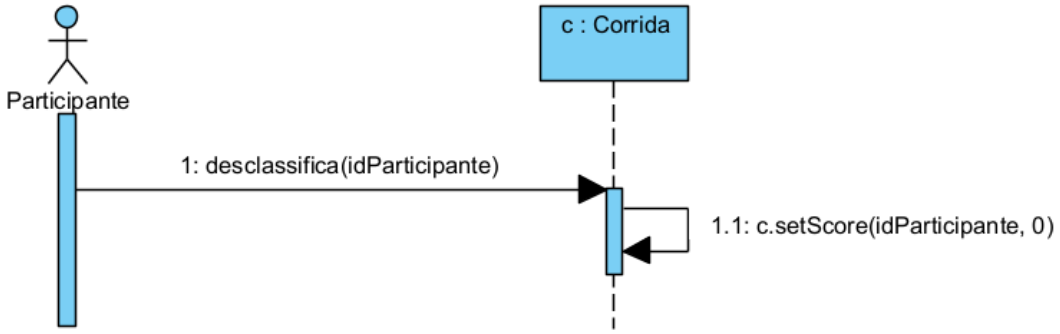

Figura 9 – desclassificarPartipante

7.8. Calcular número de retas

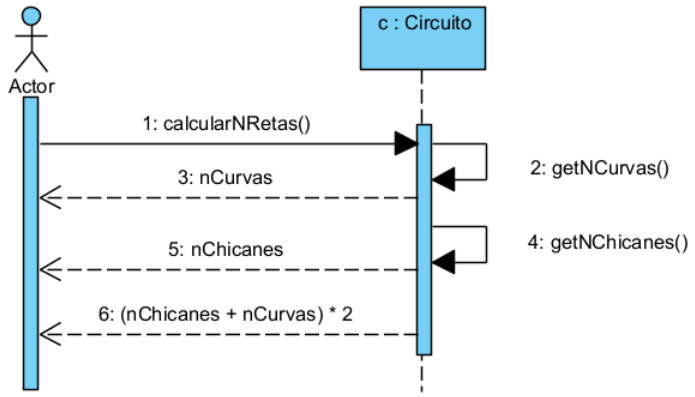

Figura 10- calcularNRetas

7.9. Calcular eventos base

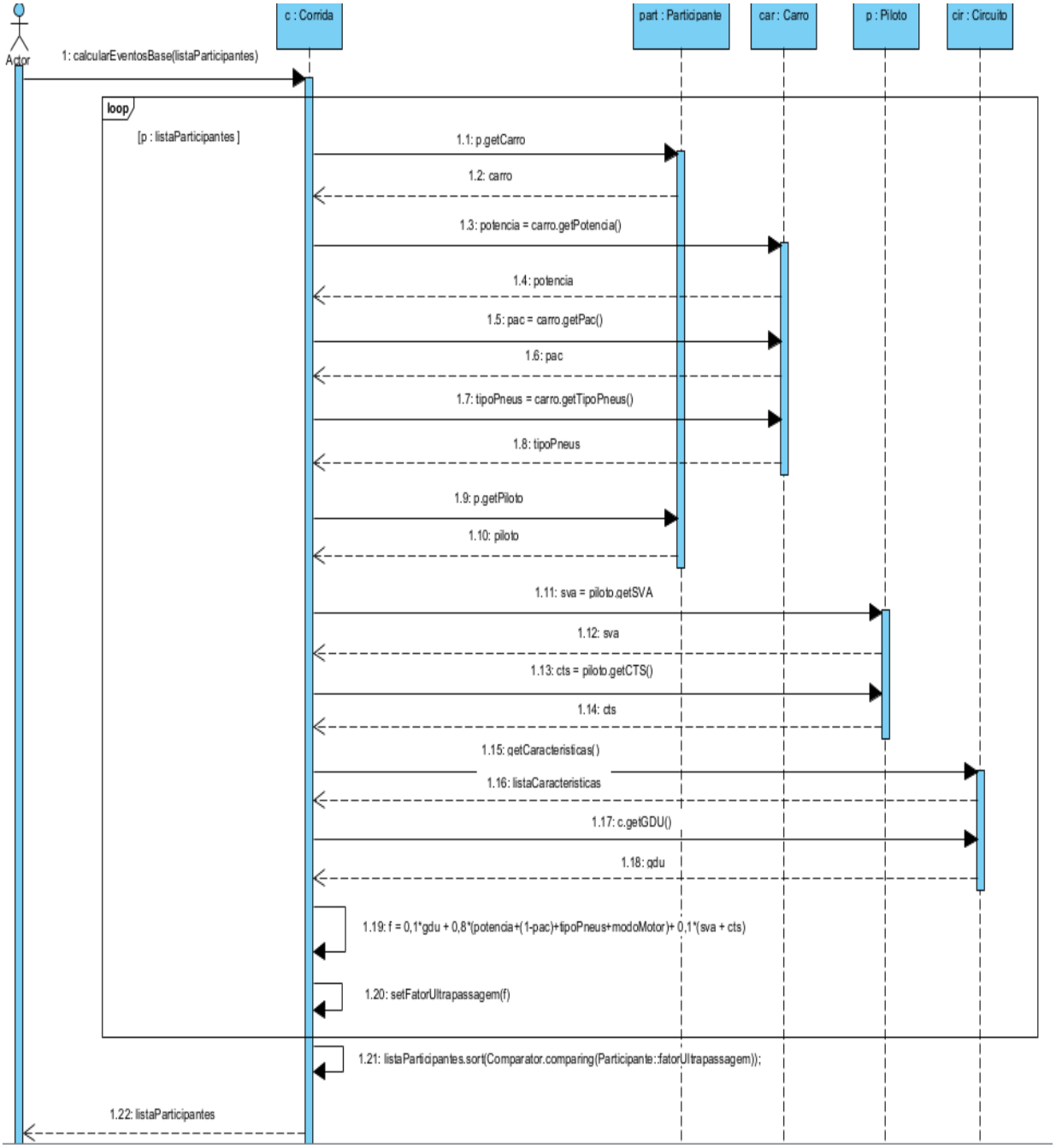

Figura 11 - calcularEventosBase

7.10. Calcular despistes

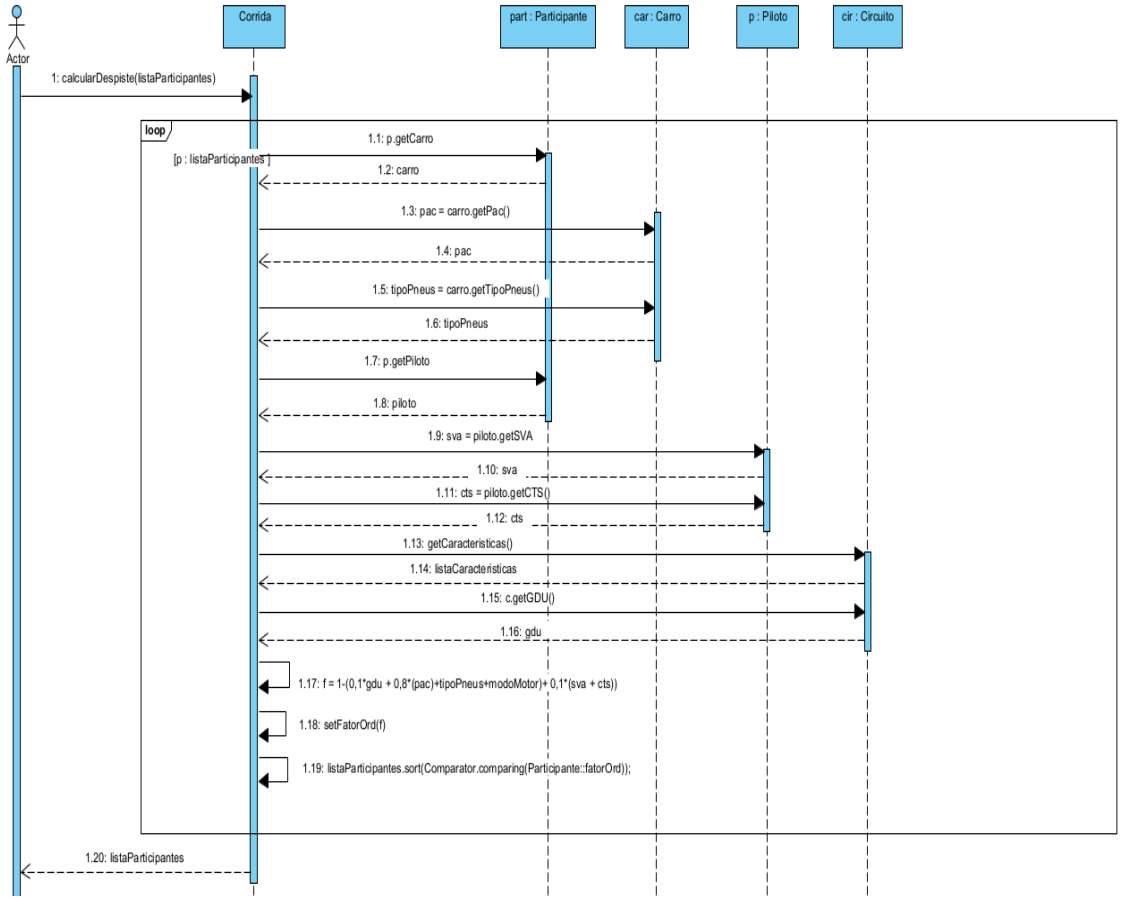

Figura 12 - calcularDespistes

7.11. Atualizar score global

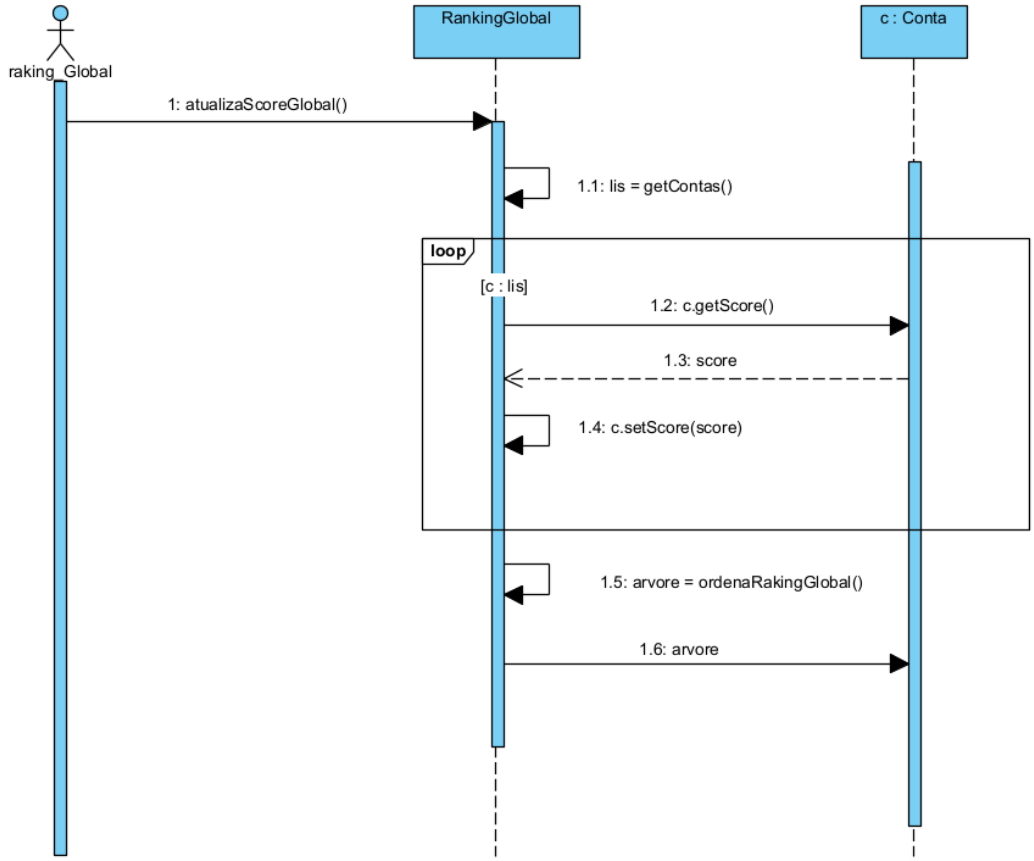

Figura 13 - atualizarScoreGlobal

7.12. Configurar campeonato

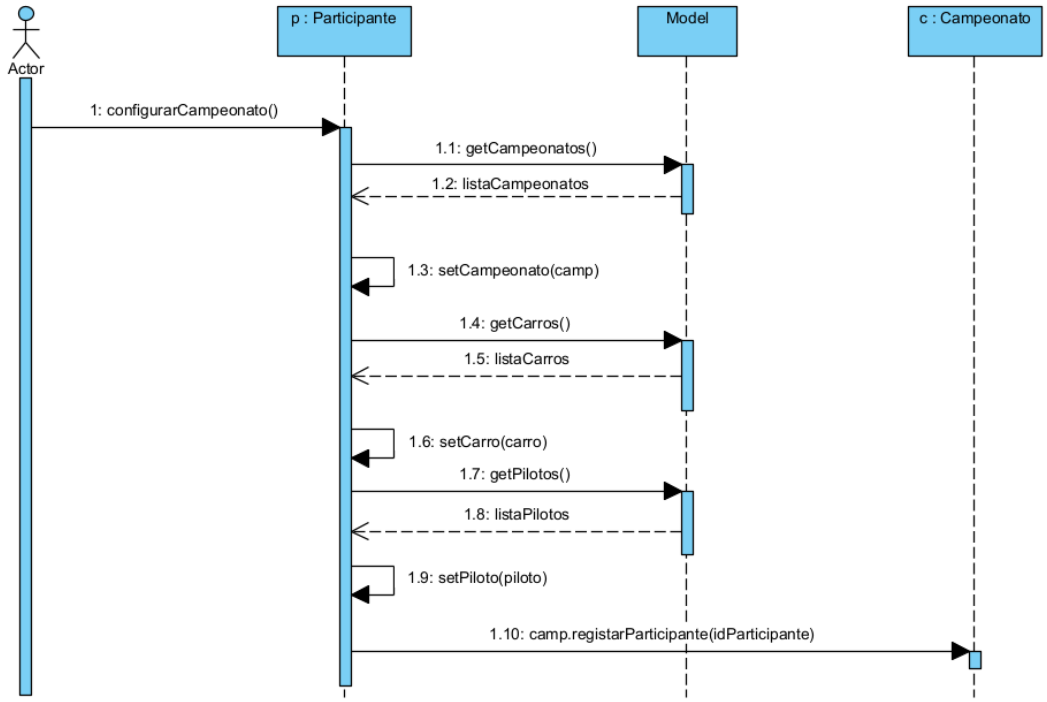

Figura 14 – configurarCampeonato

7.13. Calcular avarias

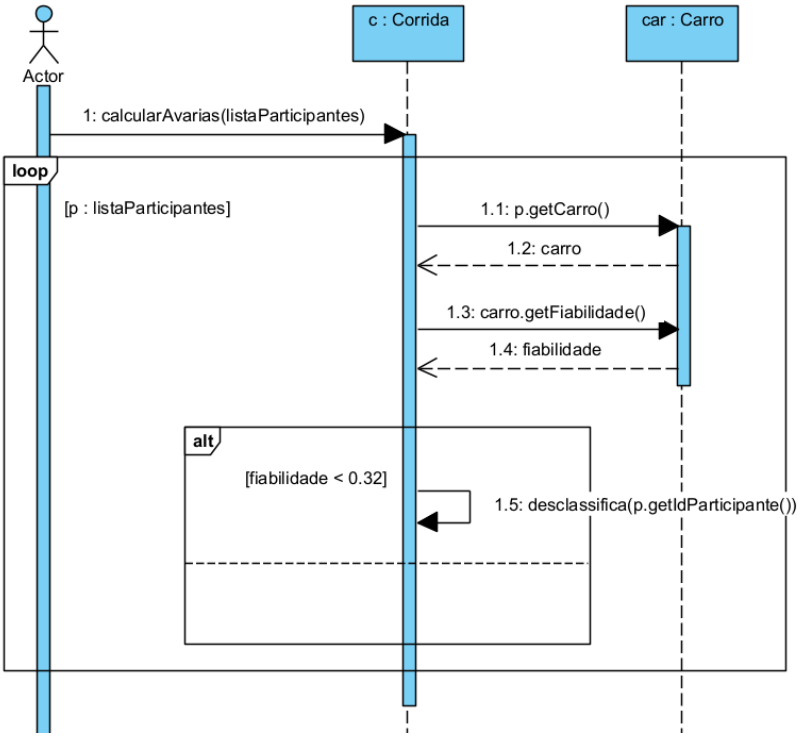

Figura 15 – calcularAvarias

8. Conclusão

Esta segunda fase do projeto exigiu mais organização e permitiu que fosse desenvolvida toda a preparação para a implementação e desenvolvimento da terceira fase. Como já tinham sido selecionados os requisitos apoiados pelos cenários fornecidos, tornou-se clara a divisão dos Use Case em diversas funcionalidades com os métodos praticar.

Desta forma já é possível idealizar um panorama geral de como deve ser desenvolvido todo este projeto. E é de acordo coletivo de que conseguimos atingir os objetivos propostos e que, de certa forma, já está pensado e analisado todo o processamento da aplicação previstos para a terceira fase.

9. Anexos

9.1. Use Cases (Cenário 5)

			1. Dividir os fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
USE CASE:		**Configurar Campeonato**				
DESCRIÇÃO:		Num determinado campeonato, cada participante escolhe um carro e um pilo				
CENÁRIOS:		ver Cenário 5 - Jogar (Configurar Campeonato)				
PRÉ-CONDIÇÃO:		Há carros e pilotos disponíveis nas respetivas listas e, pelo menos, dois participantes.				
PÓS-CONDIÇÃO:		O campeonato é assumido pelo sistema.				
FLUXO NORMAL:						
	1.	O sistema apresenta a lista de campeonatos;		Apresentar lista de campeonatos	listaCampeonatos ( ) : String	Subsistema Campeonato
	2.	O participante seleciona um campeonato;		Escolher um campeonato da lista de campe		
	3.	O sistema apresenta as especificações do campeonato selecionado;		Apresentar informação do campeonato	infoCampeonato (idCampeonato : String) : String	
	4.	O sistema apresenta a lista de carros;		Apresentar lista de carros	listaCarros ( ) : String	Subsistema Conta
	5.	Cada participante escolhe um carro;		Escolher um carro da lista de carros	setCarro(c : Carro)	
	6.	O sistema apresenta a lista de pilotos;		Apresentar lista de pilotos	listaPilotos ( ) : Piloto	
	7	Cada participante escolhe um piloto;		Escolher piloto da lista de pilotos	setPiloto(p : piloto)	
	8.	O sistema regista os participantes.		Registar participante no campeonato	registarParticipante (idParticipante : String)	Subsistema Campeonato

Figura 16 – Use Case “Configurar Campeonato”

			1. Dividir os fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
USE CASE:		**Configurar Corrida**				
DESCRIÇÃO:		Cada participante faz as preparações finais do carro.				
CENÁRIOS:		ver Cenário 5 - Jogar (Configurar Corridas)				
PRÉ-CONDIÇÃO:		Todos os participantes estão registados.				
PÓS-CONDIÇÃO:		A corrida é configurada de acordo com as escolhas de cada participante.				
FLUXO NORMAL:						
	1.	O circuito e a situação meteorológica são apresentados;		Apresentar meteorologia	getMeteorologia( ): Int	Subsistema Corrida
	2.	O sistema pergunta se quer fazer alguma afinação;		Apresentar a possibilidade de afinar	verificaAfinacao(nCorridas : Int, nAfinacoes : Int)	Subsistema Carro
	3.	O participante decide não afinar o carro;				
	4.	Cada participante escolhe o tipo de pneus;		Escolher pneus	setPneus(tipoPneus: Int)	
FLUXO ALTERNATIVO	(1)	**[o participante decide afinar o carro] (passo 3):**				
	3.1.	O sistema verifica se é possível fazer uma afinação;		Verificar condições para afinar	verificaAfinacao(nCorridas : Int, nAfinacoes : Int) : Boolean	Subsistema Conta
	3.2.	O participante define o novo valor de downforce;		Definir downforce	setDownforce(valor : Float)	Subsistema Carro
	3.3.	Cada participante escolhe o modo do motor.		Escolher modo de motor	setMotor(modoMotor : Int)	
	3.4.	Regressa a 4.				
FLUXO ALTERNATIVO	(2)	**[não é possível fazer uma afinação] (passo 3.1):**				
	3.1.1.	O sistema informa que não é possível fazer uma afinação;				Subsistema Carro
	3.1.2.	Regressa a 4.				

Figura 17 – Use Case “Configurar Corrida”

			1. Dividir os fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
USE CASE:		**Simular Corrida**				
DESCRIÇÃO:		Cada corrida de um determinado campeonato é simulada.				
CENÁRIOS:		ver Cenário 5 - Simular Corridas				
PRÉ-CONDIÇÃO:		Cada corrida do campeonato está configurada.				
PÓS-CONDIÇÃO:		O campeonato é disputado e as pontuações do mesmo são atualizadas.				
FLUXO NORMAL:						
	1.	O sistema confirma que pelo menos um jogador tem a versão base do jogo;		Confirmar a versão do jogo	getVersao(idParticipante : String) : Int	Subsistema Conta
	2.	O sistema calcula para cada curva, reta e chicane eventuais ultrapassagens, despistes e avarias a partir da posição;		Calcular ultrapassagens, despistes e avaria	calcularEventosBase(participantes : List<Participante>) : List<Participante> calcularEventosPremium(participantes : List<Participante>) :	
	3.	O sistema apresenta o resultado da volta;		Apresentar resultado da volta		Subsistema Corrida
	4.	O sistema verifica se há mais voltas;		Verificar se há mais voltas		
	5.	O sistema apresenta o resultado da corrida, por categoria;		Apresentar resultado da corrida	setScore(score : TreeSet<String, Int>)	
	6.	O sistema atualiza e apresenta os pontos da classificação do campeonato de cada participante;		Apresentar resultado do campeonato	setScoreCampeonato(score : HashSet<String, Int>)	Subsistema Campeonato
	7.	O sistema verifica se há mais corridas a serem disputadas;		Verificar se há mais corridas		
	8.	O campeonato termina.				
FLUXO ALTERNATIVO	(1)	**[todos os jogadores têm a versão premium do jogo] (passo 1):**				
	1.1.	O sistema calcula para cada curva, reta e chicane eventuais ultrapassagens, despistes e avarias a partir das diferenças de tempo entre		Calcular ultrapassagens, despistes e avaria	calcularEventosBase(participantes : List<Participante>) : List<Participante> calcularEventosPremium(participantes : List<Participante>) :	Subsistema Corrida
	1.2.	Regressa a 3.				
FLUXO ALTERNATIVO	(2)	**[um participante despista-se ou tem uma avaria] (passo 2):**				
	2.1.	O sistema indica que o participante teve um despiste ou avaria;				Subsistema Corrida
	2.2.	O participante é "retirado" da corrida;		Retirar o participante	desclassifica(idParticipante : String)	
	2.3.	O sistema atribui uma classificação "nula" ao participante;		Modificar a classificação do participante		
	2.4.	Regressa a 3.				
FLUXO ALTERNATIVO	(3)	**[nº de voltas realizadas é < nº total de voltas] (passo 4):**				
	4.1.	Regressa a 1.				
FLUXO ALTERNATIVO	(4)	**[nº de corridas realizadas é < nº total de corridas] (passo 7):**				
	7.1.	Regressa a 1.				

Figura 18 – Use Case “Simular Corrida”

			1. Dividir os fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
USE CASE:		**Consultar Resultado Final**				
DESCRIÇÃO:		O sistema atualiza e apresenta a pontuação global do *ranking* do *Racing Man*				
CENÁRIOS:		ver Cenário 5 - Resultado Final				
PRÉ-CONDIÇÃO:		Concluir um campeonato.				
PÓS-CONDIÇÃO:		A classificação global do *ranking* do *Racing Manager* é apresentada.				
FLUXO NORMAL:						
	1.	O sistema pergunta a cada participante se quer fazer o login;		Apresentar a possibilidade de login		Subsistema Conta
	2.	O participante decide fazer o login;			fazerLogin(nome : String, passe : String)	
	3.	O Sistema soma os pontos do campeonato de cada participante à classificação global do *Racing Manager*.		Adicionar pontos à classificaçao total	atualizaScore (scoreDoCampeonato : Int)	
	4.	O sistema atualiza as posições da classificação do *Racing Manager*.		Atualizar posições	atualizaScoreGlobal( )	
	5.	O sistema apresenta a classificação global do ranking do *Racing Manager*.		Mostrar ranking		
FLUXO ALTERNATIVO	(1)	**[o participante não faz login] (passo 2):**				
	2.1.	O sistema informa que a pontuação feita durante o campeonato não será considerada na classificação global do *Racing Manager*;				Subsistema Conta
	2.2.	Regressa a 3.				

Figura 19 – Use Case “Consultar Resultado Final”