

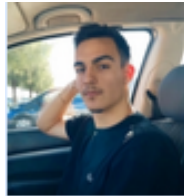
Simulador de Campeonatos de Automobilismo

Desenvolvimento de Sistemas de Software

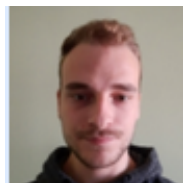
Miguel Neiva
A92945



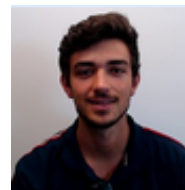
Maurício Pereira
A95338



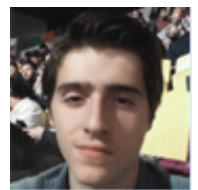
Luís Araújo
A96351



Pedro Pinto
A87983



João Cardoso
A94595



Grupo 33

<https://github.com/luisAraujo11/DSS-22-23>



Departamento de Informática
Universidade do Minho
Novembro 2022

Índice

1	Introdução	3
2	Modelo de Dominio	4
3	Modelos de Use Case	7
3.1	Atores e os seus use cases	7
4	Use Cases	8
4.1	Administrador	8
4.1.1	Piloto	8
4.1.2	Campeonato	9
4.1.3	Carro	10
4.1.4	Circuito	12
4.2	Jogador	13
4.2.1	Simular Campeonato	13
4.2.2	Consultar Pontuação/Classificação	13
4.2.3	Escolher Campeonato	14
5	Diagrama de Use Case	15
6	Identificação dos Subsistemas	16
7	Diagrama de Componentes	17
8	Diagrama de Classes e Diagrama de Package	18
9	Diagrama de Sequência	22
10	Diagrama de Atividades	25
11	Interface	27
11.1	Menu	27

12 Conclusão	29
12.1 Conclusão 1ª Fase	29
12.2 Conclusão 2ª Fase	29
12.3 Conclusão 3ª Fase	30

1 Introdução

Com este trabalho prático pretende-se conceber um sistema que permita simular campeonatos de automobilismo, que segundo o enunciado deverá assemelhar-se a algo como o F1 Manager (famoso jogo de gestão de corridas).

Na prática este simulador/jogo permitirá que utilizadores registados como jogadores consigam competir em provas automobilísticas, ao mesmo tempo permitirá também, a utilizadores registados como administradores, criar/customizar um número diverso de variáveis, tais como, a classe do carro, o seu modelo, o piloto e as suas respectivas habilidades/perícias próprias, inclusive o próprio circuito que futuramente poderá jogar.

Esta primeira fase tem como objetivo modelar este sistema em modelos UML. Serão para isso apresentados o modelo de domínio e de use cases. Para tentar perceber melhor como funciona este tipo de simulador, observamos online, parte da forma como é estruturado o jogo anteriormente referido, F1 Manager. Por fim, após analisar todos os cenários providenciados no enunciado do trabalho, procedemos à realização dos modelos de domínio e use cases.

Na segunda parte, realizamos a divisão em fluxos de sequências de transações, a identificação de responsabilidades da LN, definição das APIs e identificação dos subsistemas, na parte dos use cases. Criamos um diagrama de componentes, com vários subsistemas, que por sua vez estão definidos em diagramas de classes e diagramas de package. Criamos ainda diagramas de sequência e por fim, diagramas de atividades.

2 Modelo de Dominio

Através de uma análise do enunciado proposto, detectamos as seguintes entidades principais:

- Administrador
- Jogador
- Campeonato
- Carro
- Circuito
- Piloto

Existem ainda as entidades, que podemos considerar “secundárias”, ou seja, que não são tão importantes para a estrutura principal do modelo, tais como:

- | | | |
|-----------------|---------------|------------------|
| • Resultados | • Marca | • ModoMotor |
| • Acontecimento | • CTS | • Acontecimento |
| • Volta | • SVA | • C1Hibrido |
| • Corrida | • Informações | • C2Hibrido |
| • Registo | • Resultados | • GTHibrido |
| • Cilindrada | • Pneus | • Sitio |
| • Distância | • C1 | • Sujeito |
| • Curva | • C2 | • Descrição |
| • Reta | • GT | • MotorEletrico |
| • Chicane | • SC | • MotorCombustao |
| • GDU | • Modelo | • Fiabilidade |
| • Potência | • PAC | |
| • Afinação | | |

De todas as entidades que observamos, temos duas principais, que irão interagir com o sistema proposto, o Administrador e o Jogador. No caso do Administrador, este faz login e tem como papel gerir um conjunto de entidades contidas no sistema, particularmente adicionar, remover ou alterar um ou mais Carros, Pilotos, Campeonatos e Circuitos.

No caso do Jogador, após fazer login, este pode escolher, ou não, Pilotos, Campeonatos, Pneus e Carros. A entidade Campeonato contém Corridas e determina os Resultados finais. Temos também a entidade Carro que possui Pneus, uma dada Cilindrada associada, um PAC, uma Fiabilidade, um Modelo, um motorCombustao (que é definido por um ModoMotor) e uma Marca.

Inclui também as classes C1, C2, GT e SC. Por sua vez, estas classes estão associadas às suas subclasses, C1Hibrido, C2Hibrido e GTHibrido, das quais também está associada uma Potência (relacionada juntamente com o motorCombustao). Ainda sobre as classes, temos a Afinação (exceto o para os GT e SC).

Na entidade Piloto, este apenas é identificado pelas suas perícias, CTS e SVA. Abrangemos também a entidade Corrida, que providencia Informações e é composta por Voltas e Circuitos. A Volta é por sua vez composta por Acontecimentos, dos quais, estão descritos por um Sítio, um Sujeito e uma Descrição. Quanto ao Circuito, este mede uma Distância e é composto por Curvas, Chicanes e Retas, que são caracterizadas pela GDU.

Por último, podemos falar da entidade Registo, que vai apontar o PAC, o Jogador, o Campeonato, o Circuito, o Carro e o Piloto. Esta entidade terá como objetivo informar, a qualquer momento da simulação, após o início da mesma, os registos feitos sobre as entidades referidas anteriormente.

Nota: A entidade Potência deveria estar ligada por um relacionamento à entidade MotorEletrico, apesar de estar corrigido no modelo do github, a imagem acima apresenta essa falha. A Fiabilidade também deveria estar ligada diretamente ao carro.

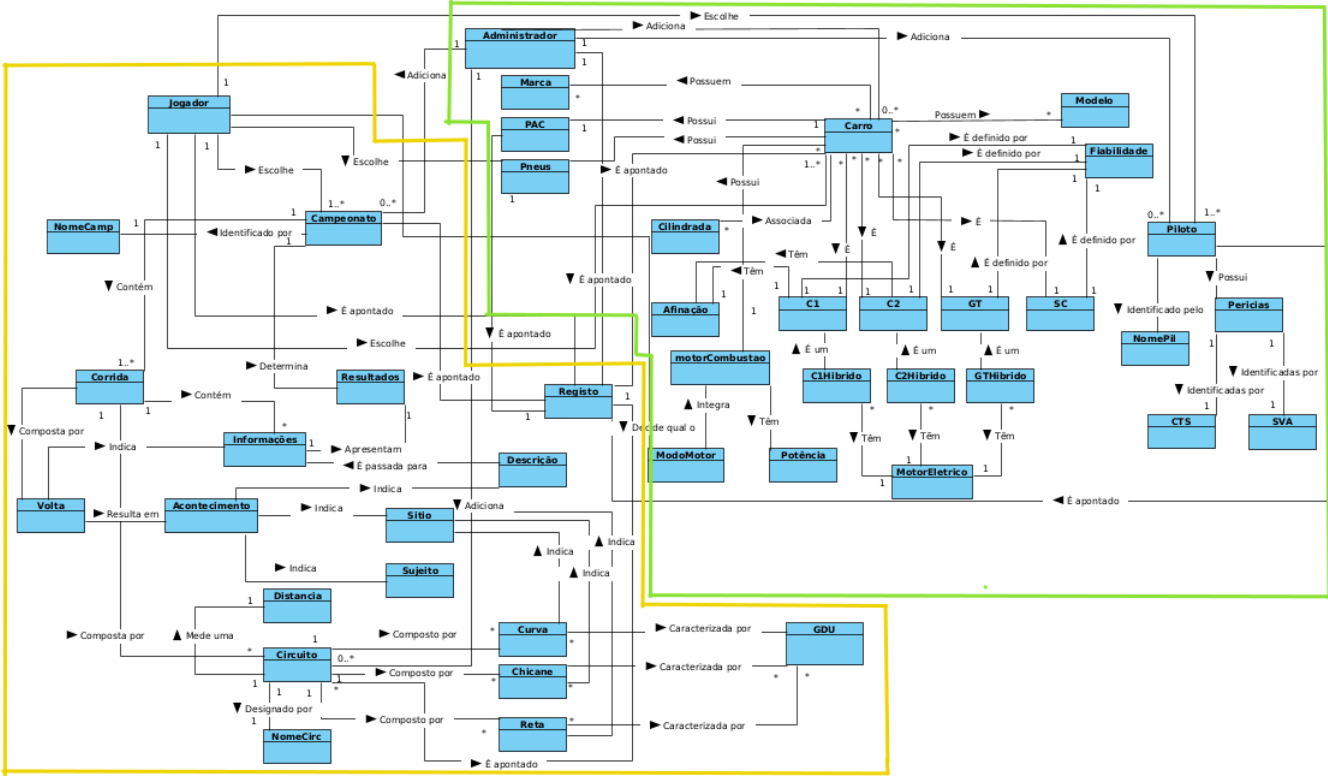


Figure 1: Diagrama de Classe

3 Modelos de Use Case

3.1 Atores e os seus use cases

Jogador:

1. Simular Campeonato
2. Consultar Resultados
3. Escolher Campeonatos

Administrador:

1. Gerir Piloto
 - Adicionar Piloto
 - Remover Piloto
 - Alterar Piloto
2. Gerir Campeonato
 - Adicionar Campeonato
 - Remover Campeonato
 - Alterar Campeonato
3. Gerir Carro
 - Adicionar Carro
 - Remover Carro
 - Alterar Carro
4. Gerir Circuito
 - Adicionar Circuito
 - Remover Circuito
 - Alterar Circuito

4 Use Cases

4.1 Administrador

4.1.1 Piloto

4.1.1.1 Adicionar Piloto

USE CASE:	Adicionar Piloto			
DESCRIÇÃO:	O administrador regista um novo Piloto			
CENÁRIOS:	4			
PRÉ-CONDIÇÃO:	Ator está autenticado			
PÓS-CONDIÇÃO:	Piloto fica disponível para jogar			
FLUXO NORMAL:		1. Dividir os fluxos em sequências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)
	1. Ator introduz nome, CTS e SVA	UI		
	2. Sistema regista o nome do piloto, CTS e SVA		Registrar nome do piloto e suas pericias	regPiloto(nomePil: String, cts: Float, sva: Float): String
	3. Sistema termina o processo	-		SubPiloto
FLUXO DE EXCEÇÃO				
	(1) [Nome já existe] (passo 1)			
	1.1 Sistema verifica que nome do piloto já existe		Verificar se o nome do piloto já existe	existePiloto(nomePil: String): boolean
	1.2 Regressa ao passo 1	-		SubPiloto

Figure 2: Use Case Adicionar Piloto

4.1.1.2 Alterar Piloto

USE CASE:	Alterar Piloto			
DESCRIÇÃO:	O administrador altera um Piloto			
CENÁRIOS:	4			
PRÉ-CONDIÇÃO:	Piloto já existe			
PÓS-CONDIÇÃO:	Piloto fica alterado			
FLUXO NORMAL:		1. Dividir os fluxos em sequências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)
	1. Ator introduz nome e alterar	UI		
	2. Sistema guarda alterações do piloto		Registrar alterações feitas	regPiloto(nomePil: String, cts: Float, sva: Float): String
	3. Sistema termina o processo	-		SubPiloto
FLUXO ALTERNATIVO				
	(1) [altera CTS] (passo 1)			
	1.1 Ator introduz um novo CTS	UI		
	1.2 Volta para o passo 2	-		
FLUXO ALTERNATIVO				
	(2) [altera SVA] (passo 1)			
	2.1 Ator introduz um novo SVA	UI		
	2.2 Volta para o passo 2	-		

Figure 3: Use Case Alterar Piloto

4.1.1.3 Remover Piloto

USE CASE:	Eliminar Piloto	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subelementos (agrupar métodos)
DESCRIÇÃO:	O administrador elimina um Piloto				
CENÁRIOS:	4				
PRÉ-CONDIÇÃO:	Piloto existe				
PÓS-CONDIÇÃO:	Piloto já não existe				
FLUXO NORMAL:					
	1. Ator escolhe nome a remover	UI			
	2. Sistema elimina o Piloto		Remover o nome de um piloto e seus atributos	eliminaPiloto(nomePil: String): void	SubPiloto
	3. Sistema termina o processo	-			
FLUXO DE EXCEÇÃO	(1) [Nome não existe] (passo 1)				
	1.1 Sistema verifica que nome não existe		Verificar se o nome do piloto não existe	existePiloto(nomePil: String): boolean	SubPiloto
	1.2 Volta para o passo 2	-			

Figure 4: Use Case Remover Piloto

4.1.2 Campeonato

4.1.2.1 Adicionar Campeonato

USE CASE:	Adicionar Campeonato	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subelementos (agrupar métodos)
DESCRIÇÃO:	O administrador registra um novo campeonato				
CENÁRIOS:	1				
PRÉ-CONDIÇÃO:	Ator está autenticado				
PÓS-CONDIÇÃO:	campeonatos				
FLUXO NORMAL:					
	1. Ator escolhe um nome para o campeonato	UI			
	2. Sistema registra o campeonato		Registrar nome do campeonato	regCamp(nomeCamp: String): String	SubCampeonato
	3. Sistema termina o processo	-			
FLUXO DE EXCEÇÃO	(1) [Nome já existe] (passo 1)				
	1.1 Sistema verifica que nome do campeonato já existe		Verificar se o nome do piloto já existe	existeCamp(nomeCamp: String): boolean	SubCampeonato
	1.2 Regressa ao passo 1	-			

Figure 5: Use Case Adicionar Campeonato

4.1.2.2 Alterar Campeonato

USE CASE:	Alterar Campeonato	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar subelementos (agrupar métodos)
DESCRIÇÃO:	O administrador altera um campeonato já existente				
CENÁRIOS:	1				
PRÉ-CONDIÇÃO:	O campeonato tem de existir				
PÓS-CONDIÇÃO:	O campeonato é alterado				
FLUXO NORMAL:					
	1. Ator escolhe o nome do campeonato que deseja alterar	UI			
	2. Sistema altera o campeonato selecionado		Registrar alterações feitas	regCamp(nomeCamp: String): String	SubCampeonato
	3. Sistema termina o processo	-			
FLUXO ALTERNATIVO	(1) [Nome não existe] (passo 1)				
	1.1 Sistema verifica que nome não existe		Verificar se o nome do campeonato já existe	existeCamp(nomeCamp: String): boolean	SubCampeonato
	1.2 Regressa ao passo 1	-			

Figure 6: Use Case Alterar Campeonato

4.1.2.3 Remover Campeonato

USE CASE:	Eliminar Campeonato	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar sub-sistemas (agrupar métodos)
DESCRIÇÃO:	O administrador remove um campeonato já existente				
CENÁRIOS:	1				
PRÉ-CONDIÇÃO:	O campeonato tem de existir				
PÓS-CONDIÇÃO:	O campeonato deixa de existir				
FLUXO NORMAL:					
	1. Ator escolhe o nome do campeonato que deseja excluir	UI			
	2. Sistema elimina o campeonato escolhido		Remover o nome de um campeonato	eliminaCamp(nomeCamp: String): void	SubCampeonato
	3. Sistema termina o processo	-			
FLUXO DE EXCEÇÃO	(1) [Nome não existe] (passo 1)				
	1.1 Sistema verifica que nome não existe		Verificar se o nome do campeonato já existe	existeCamp(nomeCamp: String): boolean	SubCampeonato
	1.2 Volta para o passo 2	-			

Figure 7: Use Case Remover Campeonato

4.1.3 Carro

4.1.3.1 Adicionar Carro

USE CASE:	Adicionar Carro	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar sub-sistemas (agrupar métodos)
DESCRIÇÃO:	O administrador registra um novo carro				
CENÁRIOS:	3				
PRÉ-CONDIÇÃO:	Ator está autenticado				
PÓS-CONDIÇÃO:	O sistema fica com mais um carro disponível para jogar				
FLUXO NORMAL:					
	1. Sistema apresenta categorias disponíveis	UI	Apresentar as categorias disponíveis	apresentaCat():String	SubRegistro
	2. Ator escolhe classe, marca, modelo, cilindrada e potência				
	3. Sistema verifica que o carro é da Classe C1 (e que pode ser híbrido)				
	4. Sistema atribui fiabilidade aleatoriamente ao carro		Apresentar valor aleatório	atribuiFiabilidade():int	SubRegistro
	5. Ator indica que o carro é não híbrido	UI			
	6. Ator indica a afinação do Carro	UI			
	7. Ator indica PAC	UI			
	8. Sistema regista carro		Registar o novo carro com as suas características	registraCarro(classe:String, marca:String)	SubRegistro
FLUXO ALTERNATIVO	(1) [carro é SC] (passo 3)				
	3.1 Sistema verifica que carro é da Classe SC	-			
	3.2 Sistema atribui fiabilidade aleatoriamente ao carro		Apresentar valor aleatório	atribuiFiabilidade():int	SubRegistro
	3.3 Ator indica PAC	UI			
	3.4 Regressa a 8	-			
FLUXO ALTERNATIVO	(2) [carro é C2] (passo 3)				
	3.1 Sistema verifica que carro é da Classe C2	-			
	3.2 Sistema atribui fiabilidade aleatoriamente ao carro		Apresentar valor aleatório	atribuiFiabilidade():int	SubRegistro
	3.3 Ator indica que o carro é não híbrido	UI			
	3.4 Ator indica a afinação do Carro	UI			
	3.5 Ator indica PAC	UI			
	3.6 Regressa a 8	-			
FLUXO ALTERNATIVO	(3) [carro é GT] (passo 3)				
	3.1 Sistema verifica que carro é da Classe GT	-			
	3.2 Sistema atribui fiabilidade aleatoriamente ao carro		Apresentar valor aleatório	atribuiFiabilidade():int	SubRegistro
	3.3 Ator indica que o carro é não híbrido	UI			
	3.4 Ator indica PAC	UI			
	3.5 Regressa a 8	-			
FLUXO ALTERNATIVO	(4) [carro é híbrido] (passo 5)				
	5.1 Ator indica que é híbrido e indica potência do motor elétrico	UI			
	5.2 Ator indica a afinação do Carro	UI			
	5.3 Ator indica PAC	UI			
	5.4 Regressa a 8	-			

Figure 8: Use Case Adicionar Carro

4.1.3.2 Alterar Carro

USE CASE:	Alterar Carro	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
DESCRIÇÃO:	O administrador altera um carro				
CENÁRIOS:	3				
PRÉ-CONDIÇÃO:	Ator está autenticado				
POS-CONDIÇÃO:	Carro fica com características alteradas				
FLUXO NORMAL:					
	1. Ator escolhe marca e modelo	UI			
	2. Sistema apresenta lista de características que podem ser mudadas		Apresentar lista de Características	apresentaCaracteristicas():String	SubRegisto
	3. Ator seleciona a desejada	UI			
	4. Ator escolhe novos valores para as características selecionadas	UI			
	5. Sistema verifica se valor introduzido está dentro dos possíveis		Verificar se o valor é possível	verificaValCarro(classe:String, marca:String)	SubVerifica
	6. Sistema atualiza características do carro		Atualizar características do carro	registraCarro(classe:String, marca:String)	SubRegisto
FLUXO ALTERNATIVO					
	(1) [Sistema verifica que valor inserido não é permitido] (passo 5)				
	5.1 Sistema pede que ator insira outro valor	-			
	5.2 Ator insere valor novo	UI			
	5.3 Sistema verifica que valor introduzido está dentro dos possíveis		Verificar se o valor é possível	verificaValCarro(classe:String, marca:String)	SubVerifica
	5.4 Volta a 6				

Figure 9: Use Case Alterar Carro

4.1.3.3 Remover Carro

USE CASE:	Remover Carro	1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
DESCRIÇÃO:	O administrador remove um carro				
CENÁRIOS:	3				
PRÉ-CONDIÇÃO:	Ator está autenticado				
POS-CONDIÇÃO:	Carro deixa de estar disponível para jogar				
FLUXO NORMAL:					
	1. Ator escolhe marca e modelo	UI			
	2. Sistema apaga carro		Remover o carro com a marca e modelo selecionado	eliminaCarro(marca: String, modelo:String)	SubElimina
FLUXO DE EXCEÇÃO					
	(1) [Nome não existe] (passo 1)				
	1.1 Sistema verifica que nome não existe		Verificar se o nome do campeonato já existe	existeCarro(marca: String, modelo:String)	SubVerifica
	1.2 Sistema termina processo	-			

Figure 10: Use Case Remover Carro

4.1.4 Circuito

4.1.4.1 Adicionar Circuito

USE CASE:	Adicionar Circuito				
DESCRIÇÃO:	O administrador regista um novo circuito				
CENÁRIOS:	2				
PRÉ-CONDIÇÃO:	Administrador está autenticado				
PÓS-CONDIÇÃO:	Circuito fica disponível para ser escolhido				
FLUXO NORMAL:		1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar sub-sistemas (agrupar métodos)
	1. Actor cria um circuito e dá-lhe um nome	UI			
	2. de curvas e chicanes	UI			
	3. Sistema com essa informação calcula o número de retas		Calcular o número de retas	calculaRetas(comprimentoCirc:Float, numCurvas:Int, numChicane:Int)	SubRegisto
	4. Sistema apresenta a lista de curvas, rectas e GDUs disponíveis (impossível, possível e difícil)				
	5. Actor escolhe um GDU para cada uma das curvas e retas	UI			
	6. Sistema define GDU da chicane como difícil		Escolher o GDU para cada curva, reta e chicane	defineGDU(curva:Int, reta:Int, chicane:Int): curva, reta(Int,String), chicane	SubRegisto
	7. Administrador indica o número de voltas e regista o circuito	UI			
	8. Sistema adiciona o circuito à lista de circuitos disponíveis		Adicionar um Circuito aos circuitos disponíveis	adicionaCircuito(nomeCircuito:String, numCurvas:Int, compCirc:Float)	SubRegisto
FLUXO DE EXCEÇÃO	(1) [Circuito já existente](passo 1)				
	1.1 Sistema verifica que um circuito com o mesmo nome já existe		Verificar se o circuito já existe	verificaCircuito(nomeCircuito:String): Boolean	SubVerifica
	1.2 Sistema termina o processo	-			

Figure 11: Use Case Adicionar Circuito

4.1.4.2 Alterar Circuito

USE CASE:	Alterar Circuito				
DESCRIÇÃO:	O administrador altera um circuito já existente				
CENÁRIOS:	2				
PRÉ-CONDIÇÃO:	Administrador está autenticado e o circuito já existe				
PÓS-CONDIÇÃO:	Os atributos do circuito são alterados				
FLUXO NORMAL:		1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar sub-sistemas (agrupar métodos)
	1. Ator escolhe o nome do circuito que quer alterar	UI			
	2. Administrador insere os novos dados	UI			
	3. Sistema grava o circuito com as alterações		Adicionar um Circuito aos circuitos disponíveis	adicionaCircuito(nomeCircuito:String, numCurvas:Int, compCirc:Float)	SubAlterar
FLUXO ALTERNATIVO	(1) [Circuito não existe] (passo 1)				
	Sistema verifica que o circuito a ser retirado não existe				
	1.1		Verificar se o circuito já existe	verificaCircuito(nomeCircuito:String): Boolean	SubVerifica
	1.2 Sistema termina o processo	-			

Figure 12: Use Case Alterar Circuito

4.1.4.3 Remover Circuito

USE CASE:	Eliminar Circuito				
DESCRIÇÃO:	O administrador remove um circuito já existente				
CENÁRIOS:	2				
PRÉ-CONDIÇÃO:	Administrador está autenticado e o circuito já existe				
PÓS-CONDIÇÃO:	Circuito não está na lista de circuitos disponíveis				
FLUXO NORMAL:		1. Dividir os fluxos em seqüências de transações	2. Identificar responsabilidades da LN	3. definir API (identificar métodos)	4. identificar sub-sistemas (agrupar métodos)
	1. Administrador escolhe o nome do circuito que quer eliminar	UI			
	2. Sistema remove o circuito da lista de circuitos disponíveis	-	Remover um circuito da lista	removeCircuito(nomeCircuito:String):	SubElimina
FLUXO DE EXCEÇÃO	(1) [Circuito não existente] (passo 1)				
	1.1 Sistema verifica que o circuito a ser retirado não existe		Verificar se o circuito já existe	verificaCircuito(nomeCircuito:String): Boolean	SubVerifica
	1.2 Sistema termina o processo	-			

Figure 13: Use Case Remover Circuito

4.2 Jogador

4.2.1 Simular Campeonato

USE CASE:	Simular Campeonato				
DESCRIÇÃO:	Racing Manager simula o Campeonato				
CENÁRIOS:	Cenário 5				
PRÉ-CONDIÇÃO:	Jogador está pronto				
PÓS-CONDIÇÃO:	Campeonato é simulado				
FLUXO NORMAL:		1. Dividir os fluxos em sequências de transações	2. Identificar responsabilidades de LN	3. Definir API (identificar métodos)	4. Identificar subtemas (agrupar métodos)
1.	Sistema valida as características do campeonato, do circuito, dos carros e dos pilotos		Validar o campeonato e as suas características	validaCampeonato(campeonato : Campeonato) : boolean	SubCorrida
2.	Jogador inicia a corrida	UI	Escolher aleatoriamente a situação meteorológica	escolheMeteo() : meteorologia	SubCorrida
3.	Sistema escolhe aleatoriamente a situação meteorológica		Simular Corrida	simularCorrida(corrida : Corrida, pilotos : Piloto, meteorologia : String) : void	SubCorrida
4.	Sistema simula a corrida				
5.	Sistema verifica acontecimentos que foram identificados através de um Sítio, um Sujeito e uma Descrição		Verificar acontecimentos associados a um sítio, um sujeito e uma descrição	verificarAcontecimentos(corrida : Corrida) : boolean	SubCorrida
6.	Sistema indica acontecimentos (despiste, ultrapassagens e avarias) por volta		Indicar os acontecimentos por volta	IndicaAcontecimentos() : List<Acontecimentos>	SubCorrida
7.	Sistema atualiza e indica a ordem dos carros no final de cada volta		Indicar a ordem dos carros em cada volta	indicaOrdemVolta(corrida : Corrida) : ArrayList<Carro>	SubCorrida
8.	Sistema verifica se a corrida acabou		Verificar se a corrida Acabou	finalCorrida(corrida : Corrida) : boolean	SubCorrida
9.	Sistema regista a pontuação de corrida na pontuação geral		Atualizar a Pontuação Geral	atualizaPontuacaoGeral(carros : Map<idCarro, Carro>, pontuacoes : Map<idPiloto, Pontuacao>) : pontAtualizada : Map<idPiloto, Pontuacao>	SubJogador
10.	Regressa a 2	-			
FLUXO ALTERNATIVO					
(1)	[Sistema não indica acontecimentos] (passo 5)		Verificar acontecimentos associados a um sítio, um sujeito e uma descrição	verificarAcontecimentos(corrida : Corrida) : boolean	SubCorrida
5.1.	Sistema verifica que durante a volta, não foram registados acontecimentos				
5.2.	Regressa a 7	-			
FLUXO ALTERNATIVO					
(2)	[Já foram realizadas todas as corridas] (passo 8)		Atualizar a Pontuação Geral	atualizaPontuacaoGeral(carros : Map<idCarro, Carro>, pontuacoes : Map<idPiloto, Pontuacao>) : pontAtualizada : Map<String, Pontuacao>	SubCorrida
8.1	Sistema regista a pontuação de corrida na pontuação geral				
8.2	Sistema termina o processo	-			

Figure 14: Use Case Simular Campeonato

4.2.2 Consultar Pontuação/Classificação

USE CASE:	Consultar pontuação/classificação				
DESCRIÇÃO:	O jogador consulta pontuação geral (resultados conjugados das corridas realizadas) / consulta a classificação da corrida				
CENÁRIOS:	5				
PRÉ-CONDIÇÃO:	Jogador está logado				
PÓS-CONDIÇÃO:	Pontuação geral/classificação da corrida é apresentada ao utilizador				
FLUXO NORMAL:		1. Dividir os fluxos em sequências de transações	2. Identificar responsabilidades de LN	3. Definir API (identificar métodos)	4. Identificar subtemas (agrupar métodos)
1.	Ator consulta a pontuacao geral	UI	Listar pontuações gerais	listaPontuacao(idCorrida : String, pilotos : Map<idPiloto, Piloto>) : pontuacaoGeral : Map<idPiloto, Pontuacao>	SubJogador
2.	Sistema lista a pontuação geral		Verificar se existe próxima corrida no campeonato	existeCorrida() : boolean	SubJogador
3.	Sistema verifica se existe próxima corrida		Simula a próxima corrida	simularCorrida(corrida : Corrida, pilotos : Map<idPiloto, Piloto>, meteorologia : Meteorologia)	SubCorrida
4.	Sistema avança para simulação da próxima corrida				
5.	Regressa a 1	-			
FLUXO ALTERNATIVO					
(1)	[Pretende consultar a classificação da corrida] (passo 1)				
1.1	Ator consulta a classificação da corrida	UI	Listar classificação da corrida	listaClassificacoes(corrida : Corrida) : classificacoes : Map<posicao, idPiloto> (?)	SubJogador
1.2	O sistema lista a classificação da corrida				
1.3	Regressa a 1	-			
FLUXO ALTERNATIVO					
(2)	[Não consulta a pontuação geral] (passo 1)				
1.1	Regressa a 3	-			
FLUXO DE EXCEÇÃO					
(3)	[As corridas já foram todas simuladas] (passo 3)		Listar resultados finais	listaResultados(nomeCamp : String) : resultadosFinais : Map<idPiloto, Pontuacao>	SubJogador
3.1	Sistema apresenta pontuação final do campeonato				
3.2	Sistema termina processo	-			

Figure 15: Use Case Consultar Pontuação/Classificação

4.2.3 Escolher Campeonato

USE CASE:	Escolher Campeonato				
DESCRIÇÃO:	O jogador escolhe um Campeonato				
CENÁRIOS:	5				
PRÉ-CONDIÇÃO:	O Jogador está autenticado				
POS-CONDIÇÃO:	Jogador está pronto para a Corrida				
FLUXO NORMAL:					
	1. Jogador escolhe um campeonato na lista de campeonatos existentes	UI			
	2. Jogador escolhe um piloto e carro	UI			
	3. Sistema verifica a classe do carro		Verificar classe	verificaClasse(carro: Carro): boolean	SubJogador
	4. Jogador altera a afinação do carro (modoMotor e PAC)	UI			
	5. Sistema aceita afinação e registra o modo do motor e o PAC		Registrar Afinação	regAfinacao(PAC: Float, tipoMotor: Motor) : boolean	SubCampeonato
	6. Jogador escolhe o tipo de pneus (macio, duro ou chuva)	UI			
	7. Sistema registra os pneus		Registrar Pneus	regPneu(tipoPneu: Pneu): boolean	SubCampeonato
	Sistema registra as opções escolhidas		Registrar opções feitas	regJogo(nomeCamp: String, nomePili: String, carro: Carro): boolean	SubCampeonato
FLUXO ALTERNATIVO	(1)				
	3.1 [Caso os carros sejam Classe GT ou SC] (passo 3)				
	3.2 Jogador não altera a afinação do carro	UI			
	Regressa a 6	-			

Figure 16: Use Case Escolher Campeonato

5 Diagrama de Use Case

Com base nos atores e nos use cases identificados procedemos à realização do diagrama de use cases. O Administrador do Sistema tem como tarefa a gerência dos pilotos, dos campeonatos, dos circuitos e dos carros, podendo adicionar, alterar e remover cada uma destas entidades. Ao jogador compete-lhe escolher o campeonato que quer jogar, assim como o carro e o piloto que quer utilizar para as suas corridas. Pode ainda iniciar a simulação das corridas, consultar as informações (voltas, posição relativa de cada carro e descrição dos acontecimentos), escolher o tipo de pneus (macio, duro ou chuva) e o modo do motor (conservador, normal ou agressivo) e definir a afinação do carro. Por último tem a capacidade de consultar os resultados finais do campeonato.

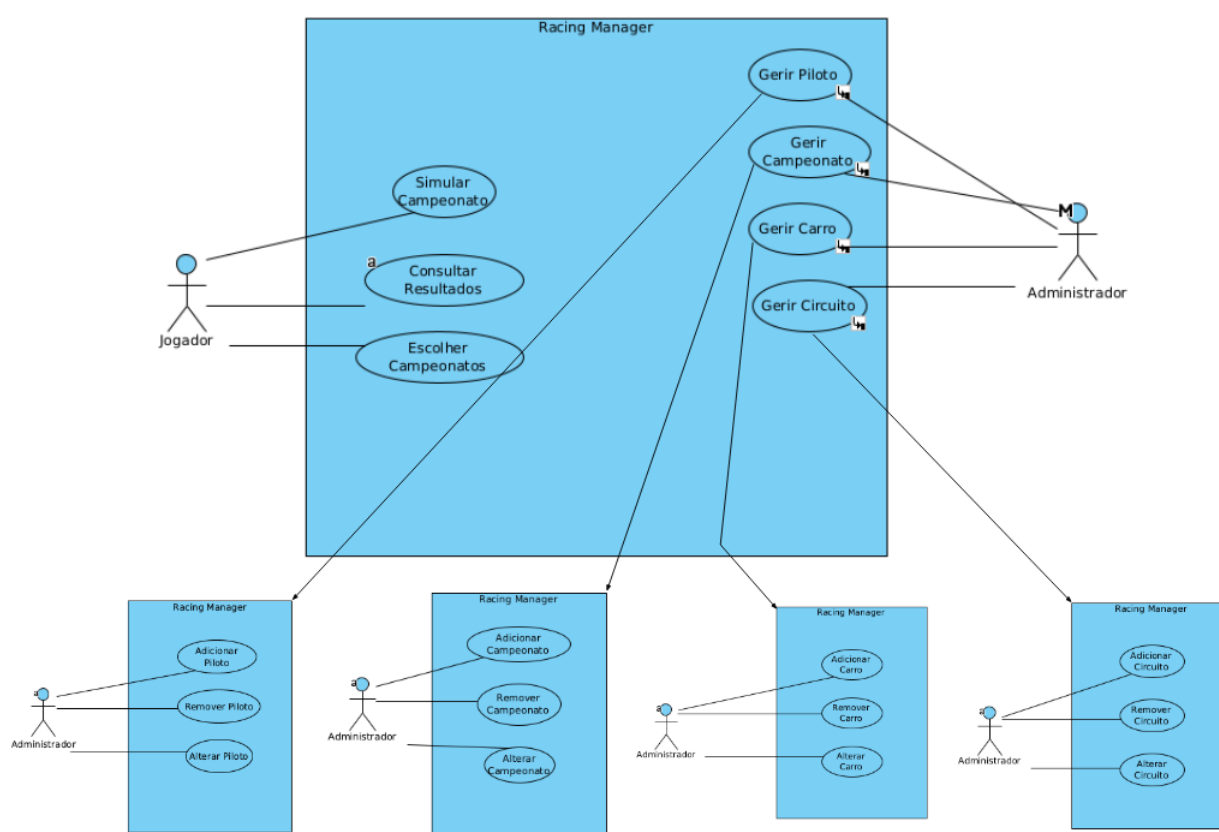


Figure 17: Diagrama de Use Cases

¹Nota: O use case "Consultar Resultados" passou a chamar-se "Consultar Pontuação/Classificação"

6 Identificação dos Subsistemas

Neste projeto foram escolhidos três subsistemas indispensáveis para uma melhor organização e modelação da implementação final.

- SubCorrida - Subsistema que comporta funcionalidades correspondentes à simulação da corrida, identificação de acontecimentos e atualização da pontuação geral.
- SubJogador - Subsistema que engloba maioritariamente decisões do Jogador como escolher campeonato, escolher carro e piloto, alterar afinação, escolher tipo de pneus e consultar classificação e pontuação.
- SubCampeonato - Subsistema que envolve o registo do campeonato com os respectivos circuitos associados, o registo do carro e as suas características, o registo do circuito com os respetivos sítios, GDUs e número de voltas e o registo do piloto e os seus atributos.

7 Diagrama de Componentes

Após a definição dos subsistemas é possível obter o diagrama de componentes. A delineação de subsistemas permite uma melhor organização, esta prática é fundamental na questão do encapsulamento uma vez que cada subsistema implementa uma interface com os métodos que lhes foram atribuídos. Deste modo existe controlo no acesso aos dados e uma estruturação mais vantajosa do código. Assim, foram definidos os Subsistemas: SubCorrida, SubJogador e SubCampeonato.

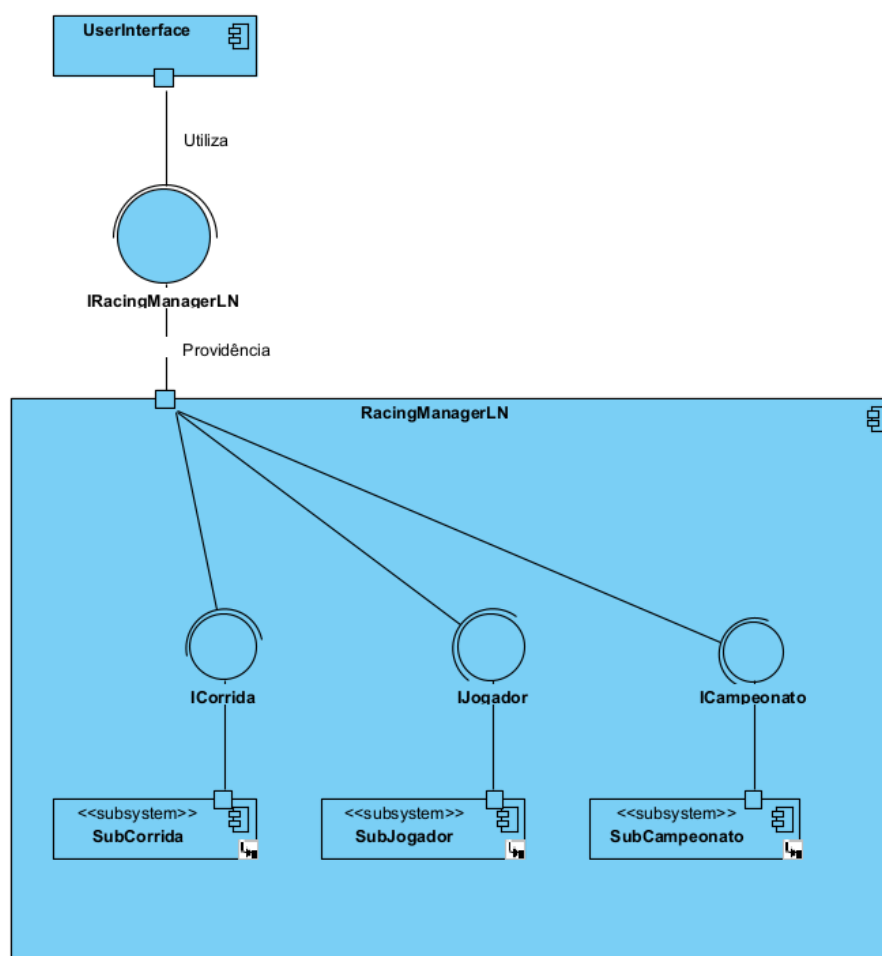


Figure 18: Diagrama de Componentes

8 Diagrama de Classes e Diagrama de Package

O diagrama de classes tem o propósito de definir as classes e os respectivos atributos e métodos. É usado também para representar as relações que existem entre as diferentes classes e as relações com a facade e a interface. O diagrama de package serve para descrever os pacotes do sistema e as dependências entre si. Cada pacote é composto por classes, por isso neste projeto escolhemos representar os dois tipos de diagramas num só diagrama geral. Foram criados os packages e dentro de cada um são representadas as classes correspondentes com a definição dos métodos e atributos. A lógica de negócio contém os métodos que representam a API e que serão divididos pelos respetivos subsistemas.

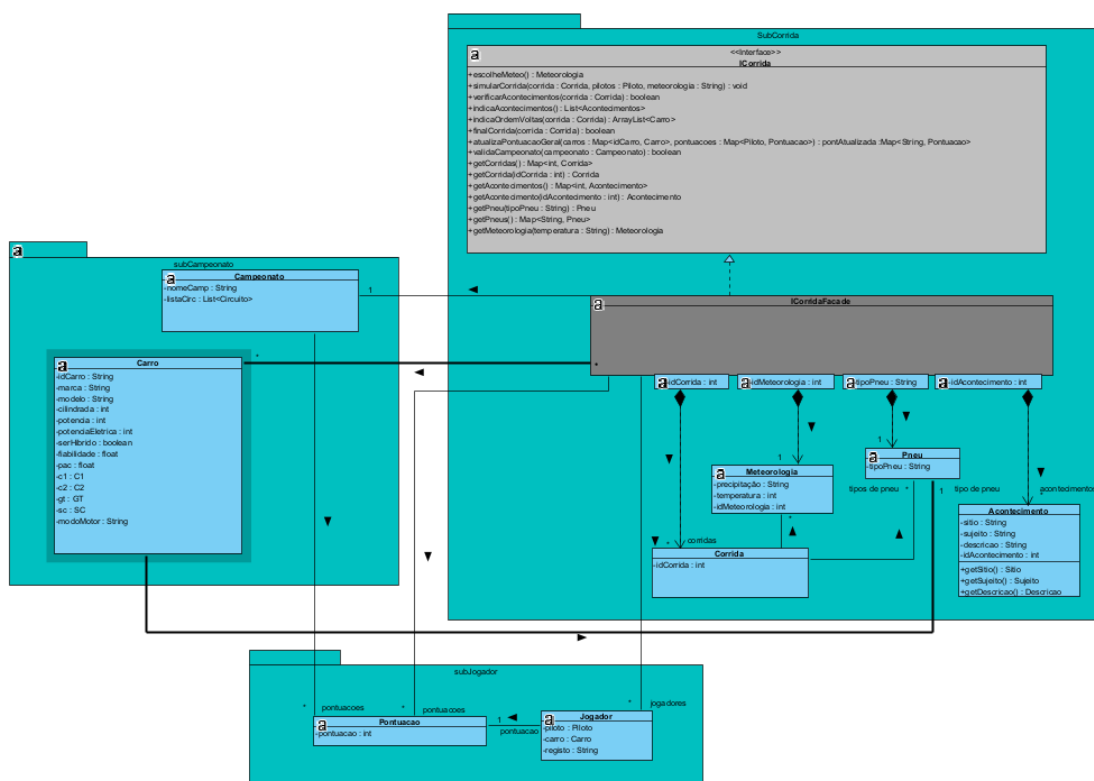


Figure 19: Diagrama de Classes e Package do SubCorrida

No diagrama da Figura 19, criamos a interface `ICorrida` e a respetiva classe `ICorridaFacade` que implementa os seus métodos. Adicionalmente existem também as classes `Meteorologia`, `Corrida`, `Piloto` e `Acontecimento` que se encontram inseridos no `SubCorrida` e são relacionadas por composição à facade. Existem também neste diagrama associações a classes inseridas noutros Subsistemas, como as classes `Carro` e `Campeonato` do `SubCampeonato`.

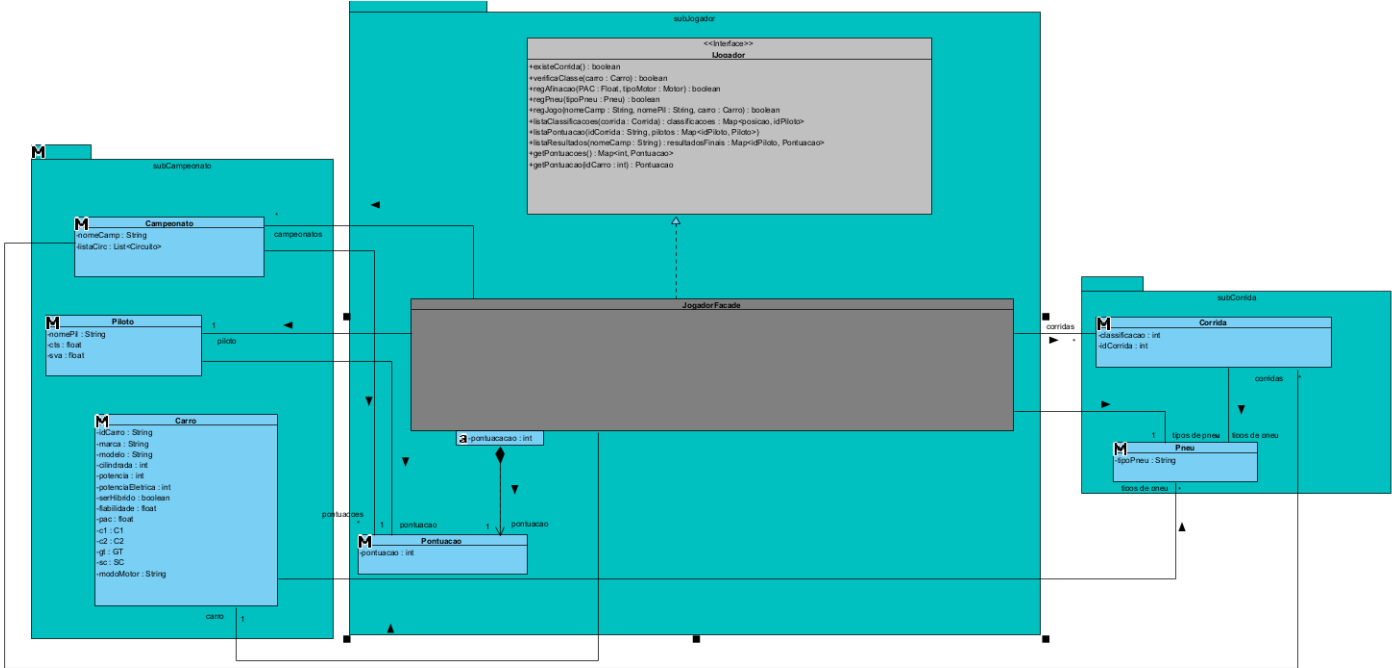


Figure 20: Diagrama de Classes e Package do SubJogador

Na figura 20 criamos a interface IJogador e a respectiva classe IJogadorFacade que implementa os seus métodos. Além disso, foi criada a classe Pontuacao que se encontra relacionado por composição à facade. Que por sua vez se encontra associado à classe Campeonato e Piloto que pertencem ao SubCampeonato. Associado á JogadorFacade encontram-se ainda as classes Corrida e Pneu que pertencem ao SubCorrida.

Com a realização fase 3 deste projeto, a implementação do jogo, deparamo-nos com alguns problemas e erros e para os resolver tomamos escolhas diferentes das tomadas na fase de especificação. Por esta razão, alguns métodos presentes nos diagramas acima não foram implementados, enquanto que outros métodos novos foram adicionados. Assim foram criados novos diagramas de classes para representar os métodos presentes na implementação

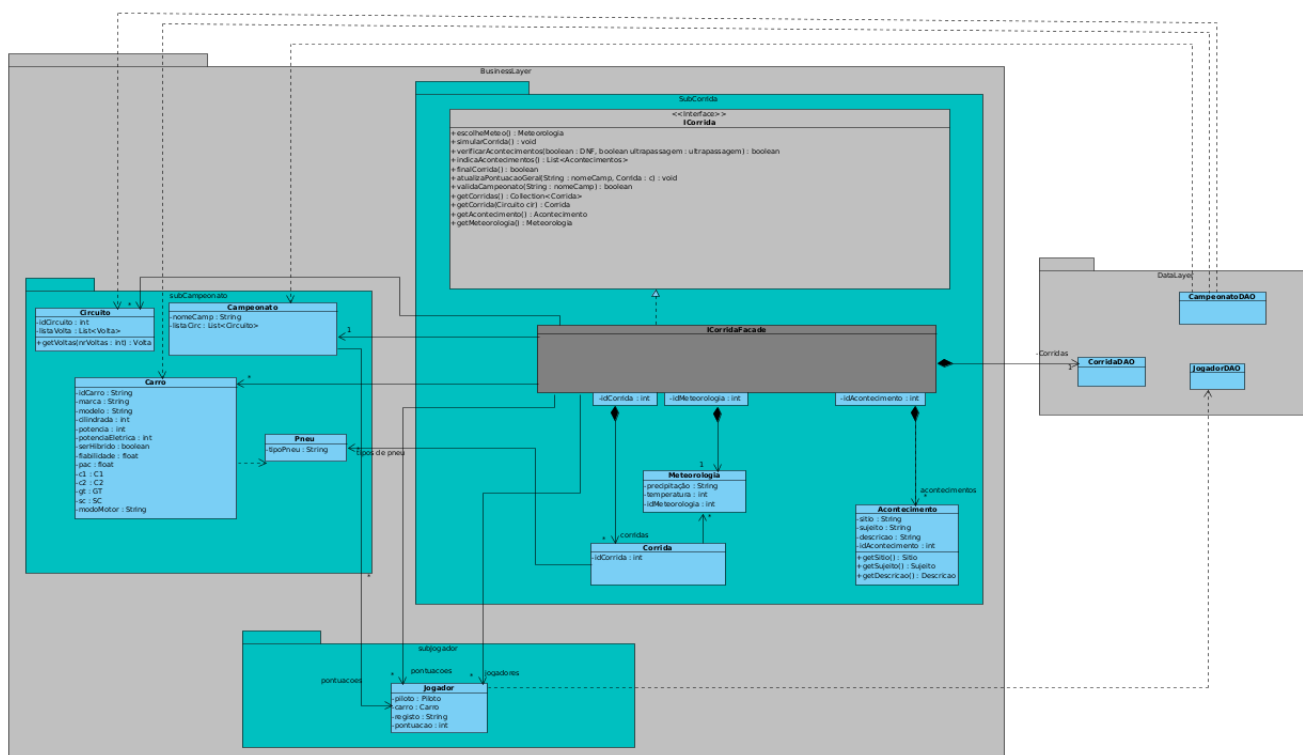


Figure 21: Diagrama de Classes

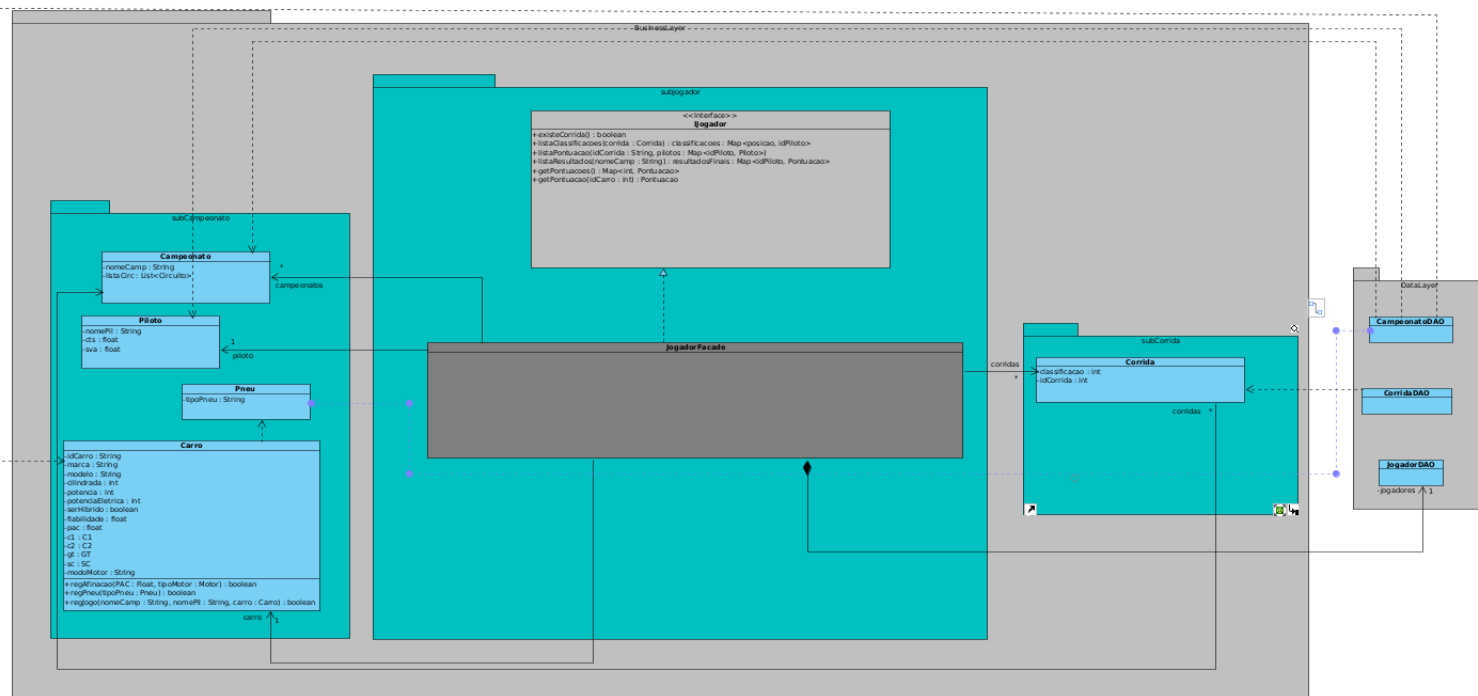


Figure 22: Diagrama de Classes

9 Diagrama de Sequência

Os diagramas de sequência têm como objetivo focar-se no ordenamento temporal da troca de mensagens, permitindo observar como é que os objetos comunicam entre si. No sentido de contemplar os seguintes cenários fornecidos pelos docentes, realizamos os diagramas de sequência que se sucedem.

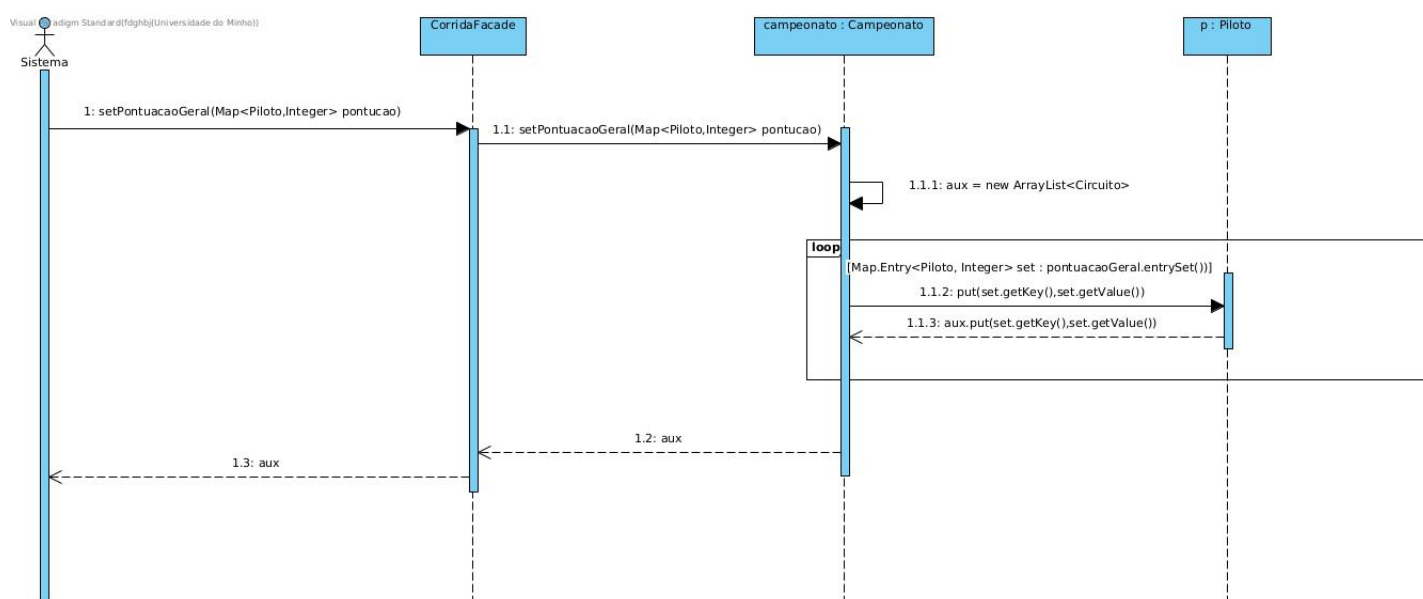


Figure 24: Diagrama de Sequência do método atualizaPontuacaoGeral()

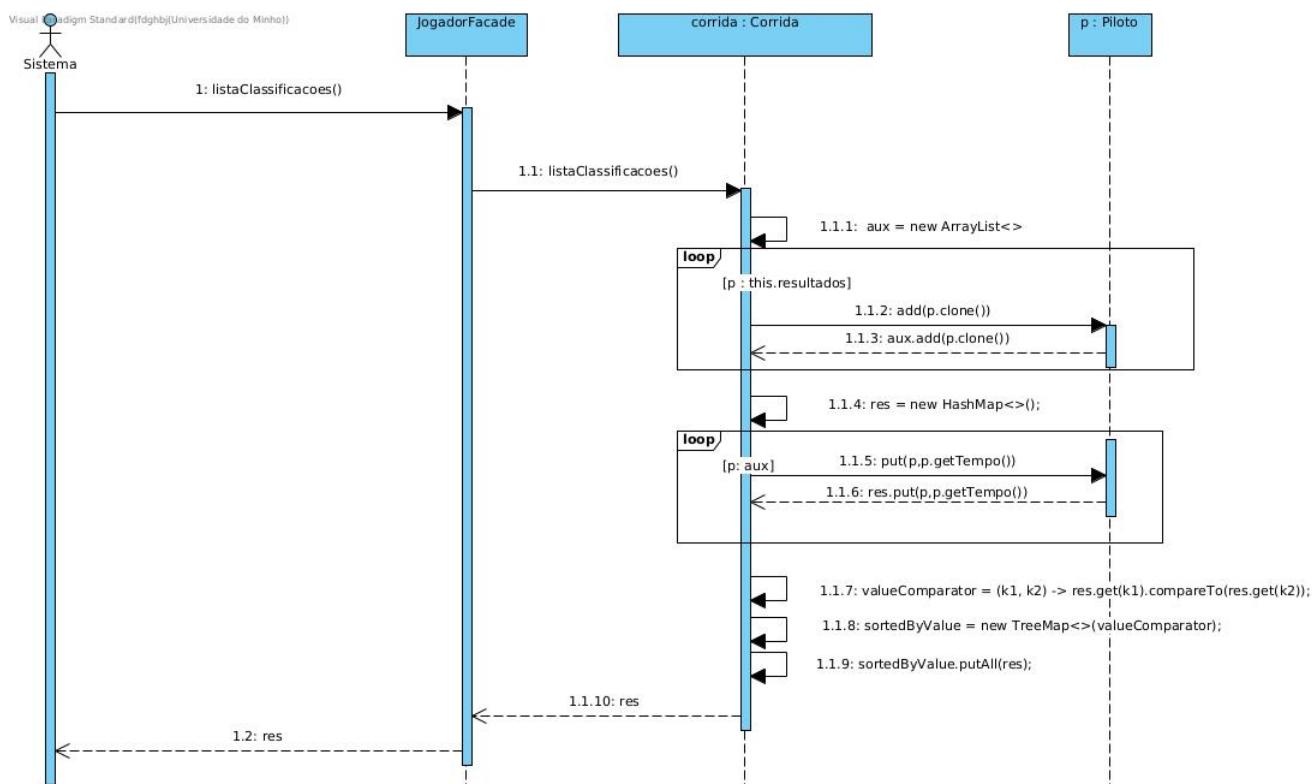


Figure 25: Diagrama de Sequência do método listaClassificacoes()

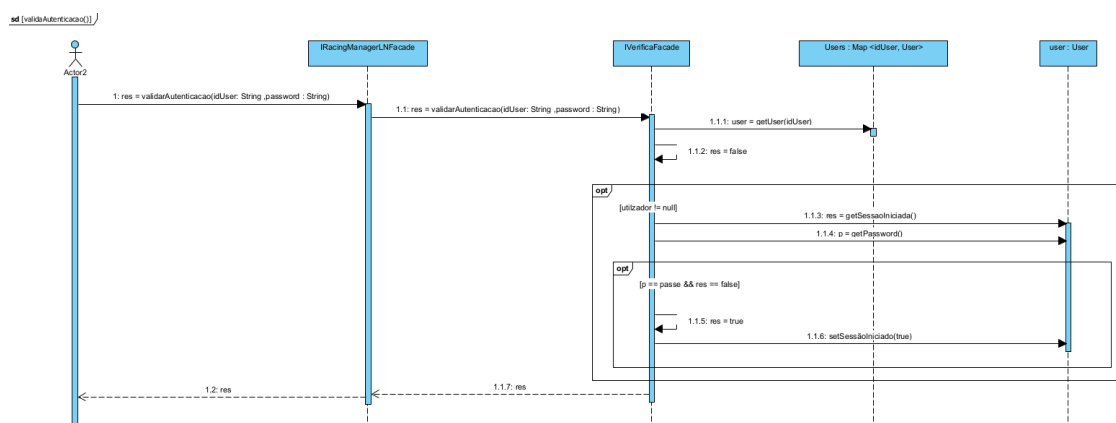


Figure 26: Diagrama de Sequência do método validaAutenticacao()

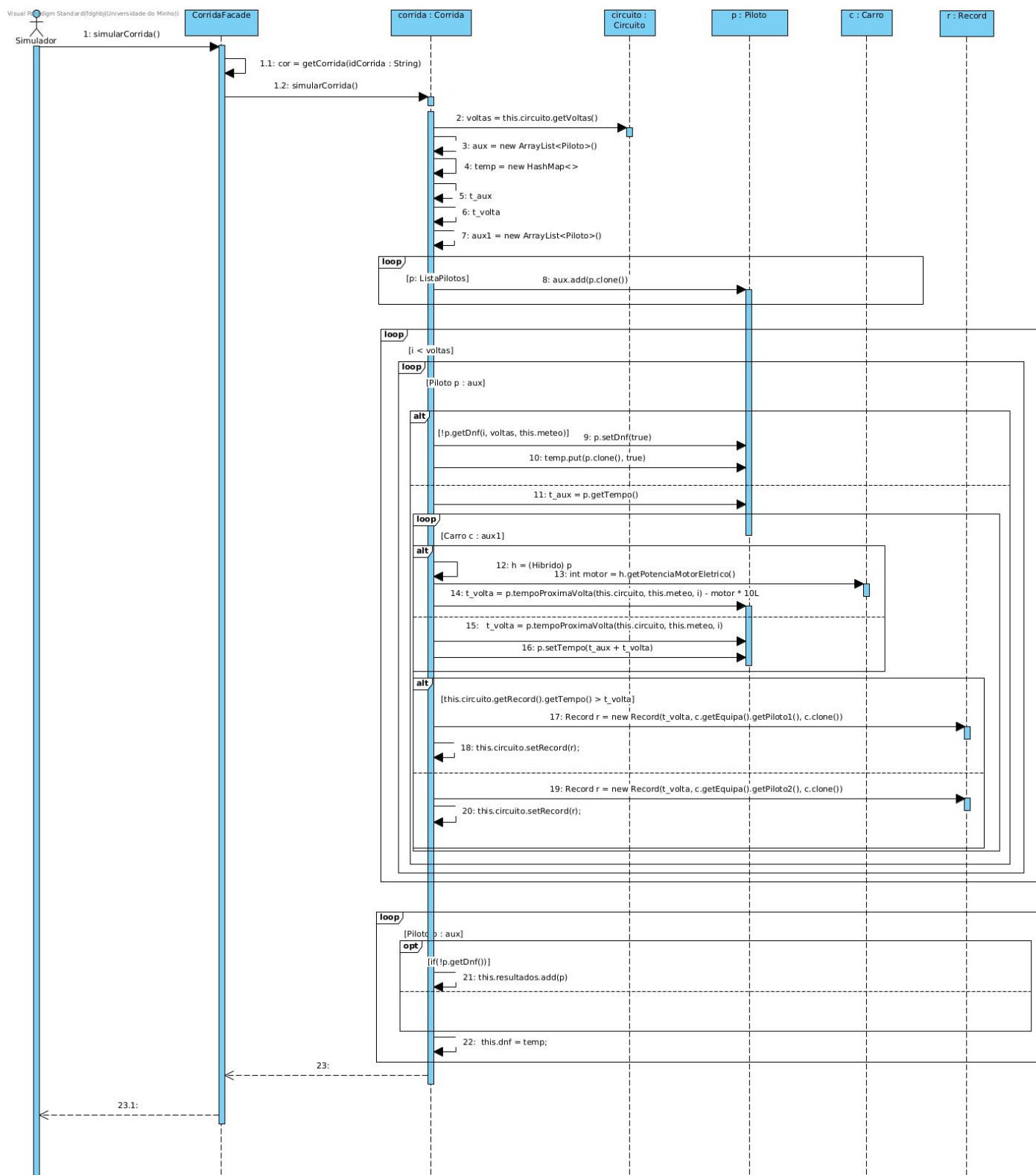


Figure 27: Diagrama de Sequência do método simulCorrida()

10 Diagrama de Atividades

Para representar o funcionamento do RaceSimulator e o desenvolvimento passo a passo da simulação das corridas e da interface do utilizador, foram desenvolvidos dois diagramas de atividades um para o manual do utilizador, onde estão representadas as escolhas e interações do utilizador e outro para o sistema.

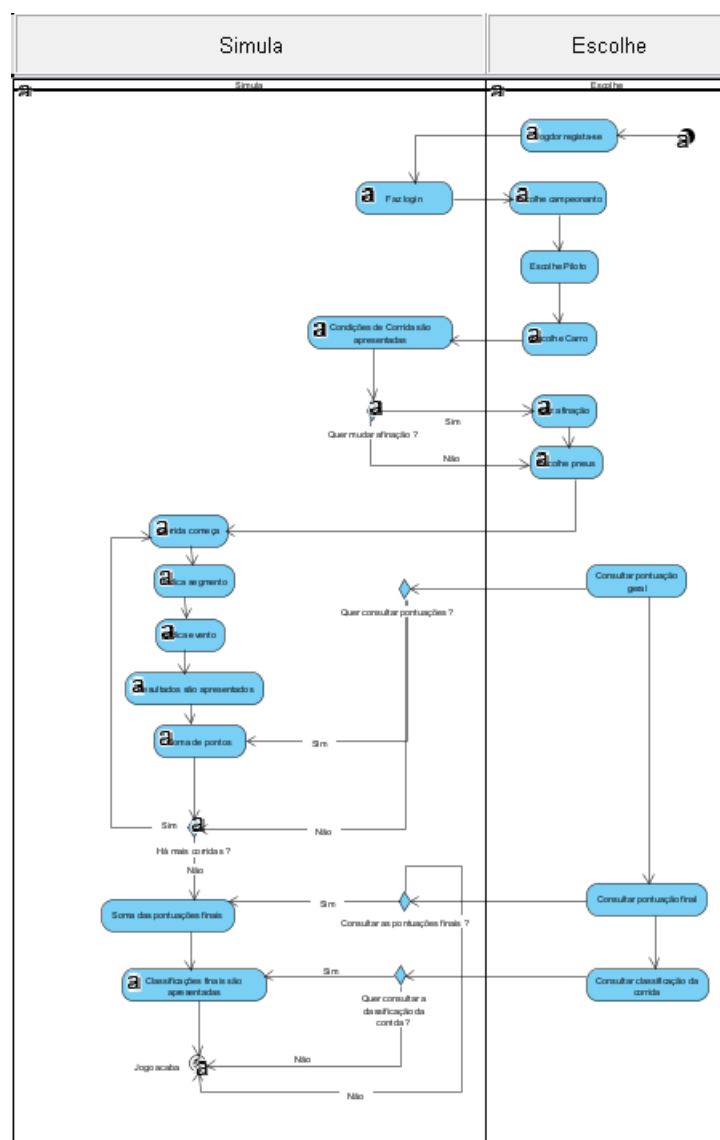


Figure 28: Diagrama de Atividades do Manual do Utilizador

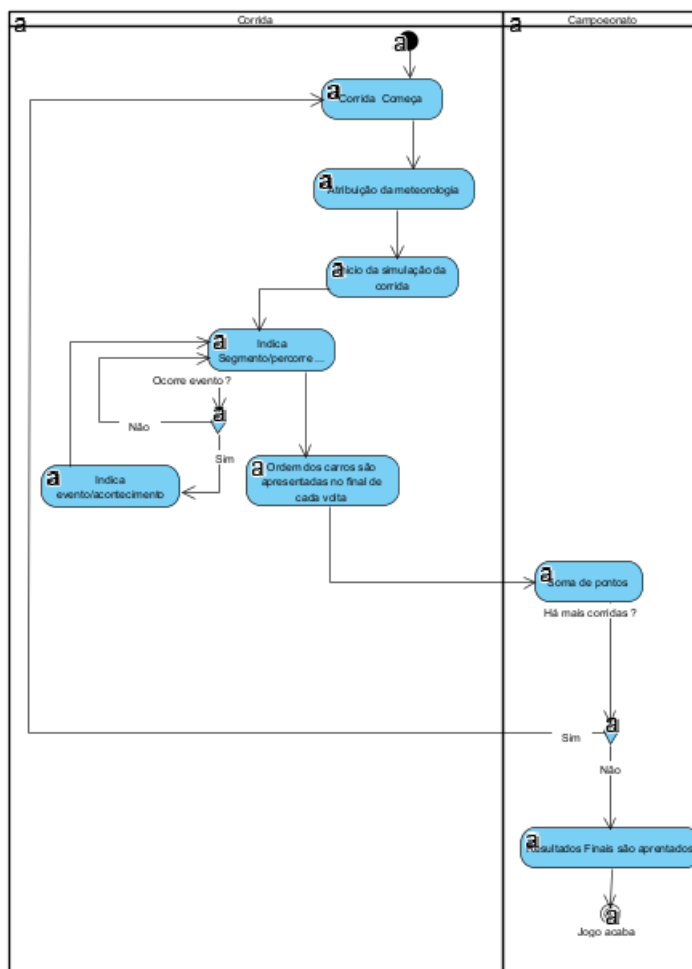


Figure 29: Diagrama de Atividades do Sistema

11 Interface

11.1 Menu

```
*** Menu ***  
1 - Adicionar um campeonato  
2 - Começar um campeonato  
3 - Listar Campeonatos  
4 - Mostrar Jogadores  
0 - Sair  
Opção:
```

Figure 30: Menu Principal

O menu principal é a primeira interface que será apresentada e o utilizador poderá interagir com ele. Através deste será possível adicionar um campeonato na opção (1), onde o utilizador pode escrever o nome do campeonato, é guardado e é apresentado de novo o Menu. Será também possível começar um campeonato previamente criado na opção (2), listar os campeonatos existentes na opção (3) e mostrar os jogadores atuais no jogo na opção (4). A opção (0) Sair, termina o programa.

```
*** Menu ***  
1 - Simular Próxima Corrida  
2 - Listar Classificação Geral  
3 - Simular todas as corridas  
0 - Sair  
Opção:
```

Figure 31: Menu da Opção 2 - Começar um campeonato

Caso o utilizador escolha a Opção 2 do Menu Principal (Começar um campeonato) é apresentado um subMenu onde o utilizador poderá simular a próxima corrida na opção (1), nesta opção os carros percorrem o circuito no número de voltas definido, no final é listada a classificação da corrida e é atualizada a classificação geral do campeonato, depois disto é apresentado novamente o subMenu de Começar um campeonato. A opção (2) serve para listar a classificação geral atualizada, se nenhuma corrida tiver sido simulada são apresentadas as classificações de cada jogador com o valor de 0. Finalmente a opção (3) tem a função de simular todas as corridas, seguidamente surge a classificação geral do campeonato e no final volta a ser apresentado o Menu Principal.

Nota : A implementação dos métodos de Simular Corrida não correu como esperávamos e por isso estes métodos não estão incluído na execução do programa. Estão implementados mas devido a razões de calendário não foi possível concluir a testagem destes.

12 Conclusão

12.1 Conclusão 1ª Fase

Na fase inicial do trabalho prático de Desenvolvimento de Sistemas de Software tratamos pela primeira vez da realização de diagramas de domínio e diagramas de use cases.

Após esta fase do trabalho, além de ganharmos experiência a modelar, estamos preparados para fazer a implementação do sistema de uma forma mais estruturada e correta. O modelo de domínio ajuda a esquematizar uma possível organização das classes e do código da fase final do projeto.

Dá-nos ainda uma visão geral sobre as relações entre as entidades, e a sua multiplicidade, o que nos permite facilmente compreender o problema. Já os use cases e o respetivo diagrama de use cases dão-nos discernimento sobre as funcionalidades da simulação que vamos implementar.

Permite, além disso, identificar como os atores vão interagir com o sistema, assim como as modificações que estes vão realizar. Acreditamos ter criado um modelo firme e bem estruturado para assim a próxima fase do projeto poder ser viável.

12.2 Conclusão 2ª Fase

Nesta segunda fase, no caso da API da lógica de negócio, consideramos que os métodos obtidos são suficientes para suportar as funcionalidades do sistema.

Além disso, esses métodos foram divididos em diferentes subsistemas de forma lógica, originando os diagramas de componentes. Através do diagrama de classes e diagrama de package implementamos as classes que consideramos necessárias para o sistema.

Com a construção dos diagramas de sequência, concedemos uma noção temporal e sequencial dos métodos, permitindo assim um fundamento para a terceira fase do projeto, que se irá traduzir na implementação do sistema. Este ultimo diagrama gerou alguma dificuldade pois exigiu nos pensar na futura implementação de cada método, sem os ter definido.

12.3 Conclusão 3ª Fase

Durante a fase de implementação entendemos realmente a importância da existência de diagramas de classe e de sequência bem desenvolvidos. O apoio dos diagramas de classes, previamente desenvolvidos tornou a implementação mais metódica e coerente.

Porém, os diagramas de sequência, como não estavam totalmente claros e corretos, tiveram que ser desenvolvidos de novo. Desta forma conseguimos tirar partido destes e implementar determinados métodos com mais facilidade e organização.

Apercebemos-nos também na implementação, que alguns métodos presentes eram prescindíveis e outros que não estavam presentes nos diagramas, foram adicionados para o programa poder executar corretamente.

Deste modo podemos dizer que o resultado da aprendizagem que resultou da elaboração deste projeto foi perceber a importância que o UML e os respetivos diagramas estudados nesta unidade curricular têm.

Um dos principais benefícios da modelação UML é que ajuda a identificar e resolver problemas potenciais antes da construção do sistema, economizando tempo e recursos a longo prazo. Ao criar um modelo visual do sistema, é mais fácil identificar problemas ou conflitos potenciais e fazer alterações necessárias no design antes de implementar o sistema.

No geral, o modelamento UML é uma habilidade valiosa para qualquer desenvolvedor de software ter em sua caixa de ferramentas. Permite uma comunicação mais eficiente e eficaz com membros da equipe e stakeholders, e ajuda a garantir que o produto final atenda às necessidades e exigências dos usuários. Ao dominar o uso de diagramas UML, os desenvolvedores podem criar sistemas de software mais robustos e confiáveis.