



Relatório Trabalho Prático LI1

”Micro Machines”

Grupo 77

Francisco Peixoto e Maria Miguel Regueiras

Braga, 20 de Dezembro 2017

Conteúdo

1	Introdução	3
1.1	Contextualização	3
1.2	Motivação	3
1.3	Objectivos	3
2	Análise de Requisitos	4
2.1	Fase 1	4
2.2	Fase 2	5
3	A Nossa Solução	6
3.1	Tarefa 1 - Construir Mapas	6
3.2	Tarefa 2 - Validar Mapas	8
3.3	Tarefa 3 - Movimentar o Carro	11
3.4	Tarefa 4 - Atualizar Estado	12
3.5	Tarefa 5 - Implementação do Jogo em <i>Gloss</i>	14
3.6	Tarefa 6 - Implementar uma Estratégia de Corrida	17
4	Validação da Solução	18
5	Conclusão	19

Lista de Figuras

3.1	Naves	14
3.2	Menu Inicial	14
3.3	Peças do mundo de gelo	15
3.4	Exemplo de um Tabuleiro	15

Capítulo 1

Introdução

1.1 Contextualização

Este trabalho prático, realizado no âmbito da unidade curricular de Laboratórios de Informática 1, teve como objetivo o desenvolvimento do jogo *Micro Machines* na linguagem de programação *Haskell*.

1.2 Motivação

O trabalho em si teve grande importância para os membros do grupo a vários níveis, já que permitiu o desenvolvimento do trabalho em equipa e, individualmente, contribuiu para a melhor compreensão da linguagem em questão.

1.3 Objectivos

Sumariamente, este projeto teve como objetivos a aprendizagem da linguagem em causa, assim como o desenvolvimento do pensamento lógico e crítico aplicado à programação e aos os seus paradigmas. Também visou a cooperação entre os membros do grupo e o a utilização do *SVN*.

Capítulo 2

Análise de Requisitos

O trabalho encontra-se dividido em duas partes fulcrais : a *Fase 1* e a *Fase 2*. Estas, por sua vez, estão também divididas, cada uma em três *Tarefas*. Uma *Tarefa* apresenta diferentes problemas com o objetivo de simplificar o desenvolvimento do jogo.

2.1 Fase 1

Na primeira fase encontramos, tal como referido em cima, três Tarefas:

Tarefa 1 - Construir Mapas

- Construir *Mapas* a partir de *Caminhos*, isto é, a partir de um conjunto de *Passos* como "Avança", "Curva Dir", "Sobe", entre outros.

Tarefa 2 - Validar Mapas

- O nome da Tarefa é bastante objetivo, resume-se a verificar se um *Mapa* é válido ou não tendo em consideração um conjunto de restrições/regras.

Tarefa 3 - Movimentar o Carro

- Focando agora no carro em si, esta Tarefa visa movimentar o *Carro* e definir as possíveis interações com o *Mapa*, como por exemplo, colisões.

2.2 Fase 2

Dando continuação à Fase 1, na Fase 2 estão presentes mais três Tarefas:

Tarefa 4 - Atualizar o Estado

- Introduzindo dois novos tipos de dados, *Jogo* e *Ação*, o objetivo desta Tarefa é atualizar o *Estado* interno do *Jogo* dado uma *Ação* realizada pelo jogador.

Tarefa 5 - Implementação do Jogo em *Gloss*

- Usando a biblioteca *Gloss* que permite ao programador desenvolver animações, simulações e desenho gráfico, implementar a componente gráfica do jogo complementando-a com a junção de todas as Tarefas previamente resolvidas.

Tarefa 6 - Implementar uma Estratégia de Corrida

- Por fim, é proposto a criação de um *bot* com o intuito de implementar uma estratégia de corrida para competir com *bots* desenvolvidos pelos docentes da UC e pelos colegas num torneio.

Capítulo 3

A Nossa Solução

Com os objetivos de cada Fase em consideração, o grupo chegou a soluções plausíveis e executáveis para cada Tarefa. O processo foi longo e com obstáculos mas, em suma, aqui se apresentam as respetivas resoluções desenvolvidas para cada Tarefa (dadas as informações da biblioteca "LI11718.hs", fornecida pelos docentes da UC).

3.1 Tarefa 1 - Construir Mapas

Função Objetivo:


```
constroi :: Caminho -> Mapa
```

Tendo em conta que um *Caminho* é um conjunto de *Passos* e um *Mapa* assume a seguinte forma :

```
(Mapa (Posicao Inicial, Orientacao Inicial) Tabuleiro)
```

Ora, com isto, foram necessárias duas funções essenciais para:

 Obter a Posição Inicial;

 Obter o Tabuleiro.

A primeira função (posição inicial) já nos era fornecida na biblioteca (a função **partida**). Para além disso, como referido no enunciado, a orientação inicial seria sempre **Este**.

Assim, tentou-se converter um *Passo* numa *Peça* através da função **iPeca** de forma a começar a construir o *Tabuleiro*.

Seguidamente, essa *Peça* teria de ser inserida num *Tabuleiro* vazio, isto é, um *Tabuleiro* apenas constituído por peças do tipo "Lava".

Para isso foi necessário criar uma função que originasse um *Tabuleiro* deste carácter, a função **tabVazio**.

Concluindo a primeira parte, a função **iTabuleiro** tem como propósito inserir uma só *Peça* no *Tabuleiro* vazio (com ajuda de funções auxiliares):

```
iTabuleiro :: Passo -> Altura -> Orientacao -> Posicao -> Tabuleiro -> Tabuleiro
iTabuleiro p a o (x,y) (h:r)
  | y==0 = subx x h (iPeca p a o) :r
  | otherwise = h : iTabuleiro p a o (x,y-1) r
```

Contudo, um *Caminho* é constituído por vários passos e, por conseguinte, várias *Peças* se originam. Então, foram desenvolvidas funções que calculassem a posição, a orientação e a altura seguintes da *Peça* ou do *Passo* em questão para conseguir converter o *Caminho* completo. Estas são a **iPosicao** e a **auxPeca**.

Por fim, criou-se a função **iTabuleiros** que é a junção de todas as funções direccionadas para a criação do *Tabuleiro* com todas as *Peças* inseridas conforme o *Caminho* dado:

```
iTabuleiros :: Caminho -> Altura -> Orientacao -> Posicao -> Tabuleiro -> Tabuleiro
iTabuleiros [] a o p t = t
iTabuleiros (x:xs) a o p t = iTabuleiros xs a' o' p' (iTabuleiro x a o p t)
  where
    a' = fst(auxPeca x a o)
    o' = snd(auxPeca x a o)
    p' = iPosicao x o p
```

Função Objetivo sugerida:

```
constroi :: Caminho -> Mapa
constroi [] = Mapa (partida [], Este) (tabVazio(dimensao []))
constroi c =
  Mapa (partida c, Este) (iTabuleiros c 0 Este (partida c) (tabVazio(dimensao c)))
```


3.2 Tarefa 2 - Validar Mapas

Função Objetivo:





```
valida :: Mapa -> Bool
```

Para validar um *Mapa* há um conjunto de regras que devemos ter em consideração, segundo o enunciado fornecido pelos docentes da UC. Assim, o grupo tomou a iniciativa de dividir este problema em três partes essenciais:

1. A validação de aspetos relacionados apenas com o *Tabuleiro*;
2. A validação de aspetos relacionados apenas com a *Peça* de Partida;
3. A validação de aspetos relacionados apenas com o *Percurso*.

Validação do Tabuleiro

Existem quatro regras principais para um *Tabuleiro* ser válido:

-  Apresentar um formato regular;
-  A moldura deve ser constituída apenas por *Peças* do tipo "Lava";
-  Não pode ser constituído apenas por *Peças* do tipo "Lava";
-  Todas as *Peças* do tipo "Lava" têm de possuir altura igual a zero.

Para verificar o primeiro aspeto, o grupo desenvolveu a função **vFormatoTabuleiro** que, resumidamente, verifica se existem o mesmo número de colunas e linhas.

Em relação ao segundo, terceiro e quarto pontos, foram criadas as funções **vMoldura**, **nValidaTabLava**, **vAltLava**.

A função final que junta todas as funções referidas, **vBases**, é dada por:

```
vBases :: Mapa -> Bool
vBases ( Mapa (p,o) tab) = vFT && vM && nVTL && vAL
  where
    vFT = vFormatoTabuleiro tab
    vM  = vMoldura tab
    nVTL = nValidaTabLava tab
    vAL  = vAltLava tab
```

Validação da Peça de Partida

Para validar a *Peça* de Partida é preciso ter em mente que:

- ❏ A orientação inicial tem de ser compatível com *Peça*;
- ❏ Não pode ser uma *Peça* do tipo "Lava".

Assim, desenvolveram-se duas funções, primeiro uma função simples que só verifica se a *Peça* não é do tipo "Lava", a função **vPosicaoInicial**. De seguida, a função **vOrientacaoPPartida** que verifica todas as opções possíveis e determina se a orientação é compatível com a *Peça* de Partida.

A função final que junta todas as funções referidas, **vInicio**, é dada por:

```
vInicio :: Mapa -> Bool
vInicio m = vOrientacaoPPartida m && vPosicaoInicial m
```

Validação do Percurso

Ao abordar este ponto, o grupo decidiu criar um novo tipo, o tipo *Percurso* de forma a facilitar a compreensão de quem lê o código e o desenvolvimento da *Tarefa* por parte do grupo.

Este novo tipo consiste num par de listas de *Posições* e *Peças* correspondentes a uma trajetória.

```
type Percurso = ([Posicao], [Peca])
```

Para validar o *Percurso* existem três restrições principais:

- ❏ A trajetória tem de ser válida;
- ❏ As alturas do percurso têm de ser compatíveis entre si, assim como as *Peças*;
- ❏ Todas as *Peças* fora do *Percurso* têm de ser do tipo "Lava".

De forma resumida, antes de tudo, foi essencial desenvolver uma forma de arranjar o *Percurso* a partir do *Mapa* em si, a função **iPercurso** resolveu a questão (recorrendo a funções auxiliares):

```
iPercurso :: Mapa -> Percurso
iPercurso (Mapa (p,o) t) = if notlava p t
    then auxPercurso p o t ([],[])
    else ([p], [Peca Lava 0])
```

De maneira a resolver o primeiro ponto, foi criada a função **vTrajetoria** que verifica se uma Trajetória de um Percurso é válida, ou seja, se o primeiro e o último par Posição/Peça são iguais e diferentes de Lava.

Seguidamente, em relação ao segundo ponto, criou-se a função **vAltura** que não só verifica as alturas das *Peças* como também verifica se estas são compatíveis entre si.

Por fim, a função **vResto** certifica-se que tudo que não pertença ao *Percurso* é do tipo "Lava".

A função que junta todas as funções referidas, **vPercurso**, é dada por:

```
vPercurso :: Mapa -> Bool
vPercurso m = vTrajetoria m && vResto m && vAltura m
```

Função Objetivo sugerida:

```
valida :: Mapa -> Bool
valida m = vBases m && vInicio m && vPercurso m
```

3.3 Tarefa 3 - Movimentar o Carro

Função Objetivo:

```
movimenta :: Tabuleiro -> Tempo -> Carro -> Maybe Carro
```

Para mover o *Carro*, o grupo encontrou alguns contratempos durante a Fase 1. Contudo, chegou a uma solução executável embora simplista.

Dadas as informações do *Carro*, foi necessário compreender como este se comportava e foi claro que este iria sofrer *colisões* com as paredes de alturas diferentes, com *Peças* do tipo "Curva" e que quando se encontrava numa do tipo "Lava" era prejudicado.

Primeiramente, foi essencial definir uma função que movesse efetivamente o *Carro*, a função **move**.

Assim, foi importante saber onde este estaria a entrar numa *Peça* do tipo "Curva", dado que existem duas formas possíveis de entrar e, por conseguinte, se estaria a virar à direita ou à esquerda. Criaram-se, então, as funções **vEsquerda** e **vDireita** que verificam para que lado está o *Carro* a virar.

Sabendo agora para que lado está a virar, definiu-se uma função que dita o que acontece em cada *Peça* do Tabuleiro e devolve o novo *Estado* do *Carro*, a função **iPartida** (que devolve *Nothing* quando este cai à "Lava" e *Just Carro* com as devidas alterações para as outras *Peças*).

Por fim, foi necessário definir o que acontece durante todo o tempo dado, assim definiu-se a função **i** que funciona como função auxiliar para a função final.

Função Objetivo sugerida:

```
movimenta :: Tabuleiro -> Tempo -> Carro -> Maybe Carro
movimenta tab t (Carro (x,y) d vl) = i tab (Just (Carro (x,y) d vl)) t
```

3.4 Tarefa 4 - Atualizar Estado

Função Objetivo:

`atualiza :: Tempo -> Jogo -> Int -> Acao -> Jogo`

De modo a atualizar o *Estado* do *Jogo* dadas as *Ações* do jogador estão implícitas várias componentes.

Num *Jogo* há 3 partes que necessitam atualização:

- O *Histórico*.
- A lista de tempos dos *Nitros*;
- A lista de *Carros*;

Atualizar o Histórico

De forma a atualizar o *Histórico*, foi necessário verificar se a *Posição* onde o *Carro* se encontra já existe na lista, se a posição já fizer parte da lista, esta mantém-se igual, caso não se encontre adiciona-se a respetiva posição à cabeça da lista. Isto através da função **ihistorico**.

Atualizar os Nitros

Já para atualizar a lista de tempos dos *Nitros*, apenas foi preciso identificar o jogador que está a utilizar o *Nitro* e retirar o tempo que foi usado ao jogador respetivo, através da função **iNitro**.

Atualizar o Carro

Por fim, de modo a atualizar os *Carros* existem alguns pontos a ter em consideração, como a *Direção* e a *Velocidade*. No *Jogo* são fornecidas as *Propriedades* da pista, ou seja, valores que se devem aplicar ao *Carro*. Todas são constantes exceto o atrito que é uma percentagem. Assim, através de fórmulas, cálculos de vetores, entre outros processos se desenvolveram as seguintes funções:

1. Direção

Componente que se atualiza tendo em conta para que lado o *Carro* está a virar e a constante *k_roda* através da função **virar** que aplica a constante e confirma se este está a virar ou não.

2. Velocidade

Para esta componente, primeiramente, aplicaram-se as constantes `k_atrito` e `k_pneus` à velocidade inicial pela função **fIniciais**. Depois foram desenvolvidas funções que aplicassem o resto das constantes dependendo das *Ações* realizadas pelo jogador: **fGrav** que aplica a constante `k_peso` apenas nas *Peças* do tipo "Rampa"; **fAcel** que aplica a constante `k_acel` sempre que o *Carro* acelera; **fNitro** que aplica nitro no jogador indicado na *Ação*.

A função que realiza primeiro a **fIniciais** e só depois todas as outras mencionadas é a função **forças**.

Função Objetivo sugerida:

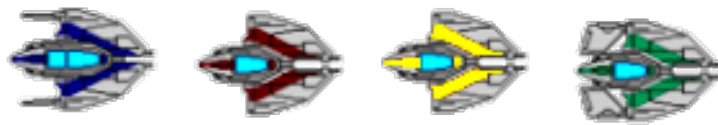
```
atualiza :: Tempo -> Jogo -> Int -> Acao -> Jogo
atualiza t (Jogo m p c lN lH) jogador acao = Jogo m p carros nitros historico
  where
    carros      = (forças jogador (Jogo m p c lN lH) t acao)
    nitros      = (initro t (Jogo m p c lN lH) acao jogador )
    historico   = (ihistorico (Jogo m p c lN lH) jogador)
```

3.5 Tarefa 5 - Implementação do Jogo em *Gloss*

Na *Tarefa* que se segue o grupo teve a liberdade de explorar a biblioteca *Gloss* e desenvolver a componente gráfica do jogo. O modelo final tem alguns erros que o grupo não conseguiu resolver, contudo implementaram-se algumas funcionalidades interessantes que tornam o jogo mais apelativo.

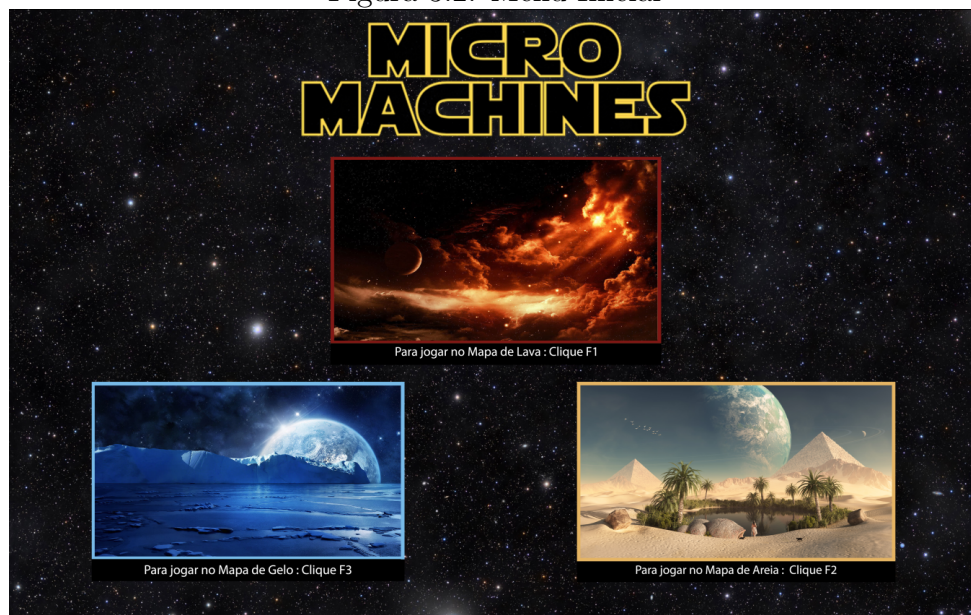
O tema escolhido foi baseado nos filmes da saga "Star Wars" e, em vez de um jogo de carros, criou-se um jogo de corridas de *naves* espaciais:

Figura 3.1: Naves



Ao abrir o executável da *Tarefa*, a janela abre em *Full Screen* um menu que permite ao jogador escolher de entre três *Mapas* um onde quer jogar (através das teclas F1, F2 e F3): um de lava, um de areia e um de gelo.

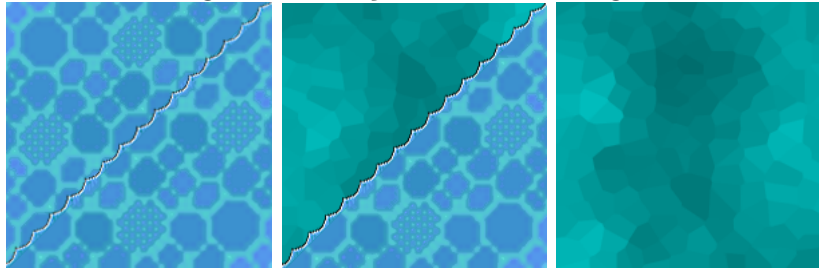
Figura 3.2: Menu Inicial



É de notar que é possível mudar de *Mapa* sem voltar ao menu inicial e que os *Mapas* têm *Propriedades* diferentes.

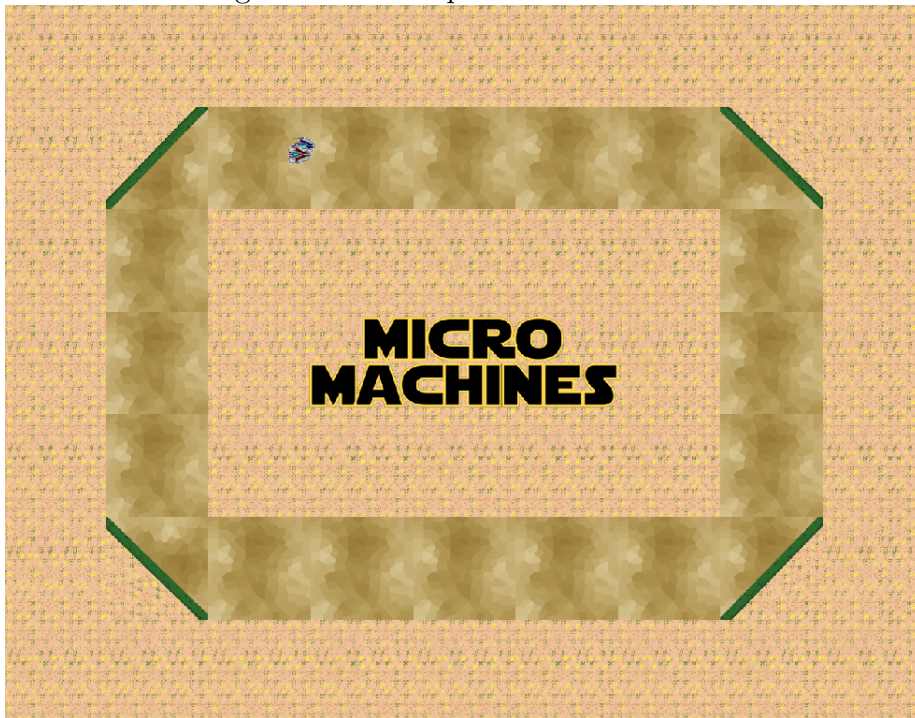
Para cada *Mapa* foram criadas diferentes imagens para as *Peças*, por exemplo, no *Mapa* de gelo encontramos as imagens seguintes:

Figura 3.3: Peças do mundo de gelo



De forma a transformar a informação contida no *Tabuleiro* numa imagem no ecrã, criou-se a função **tabpic** que resulta de uma combinação de funções auxiliares, que têm como intuito moldar os vários tipos existentes, como por exemplo as "linhas" de *Peças*.

Figura 3.4: Exemplo de um Tabuleiro



Para desenvolver a mecânica do jogo, começou por se definir o tipo *Estado*:

```
type Estado = (Int, Jogo, Acao, Picture, Picture, [Picture], [Picture], [Picture])
```

O *Estado* constitui a base da *Tarefa*, já que permite o funcionamento da função do Gloss **play**, que trabalha segundo os seguintes componentes:

```
play = Display -> Color -> Int  
      -> estadoInicial -> desenhaEstado -> reageEvento -> reageTempo -> IO ()
```

Primeiramente, foi necessário criar um estado inicial. Por isso, a função **estadoInicial** contém as informações necessárias para começar um jogo imprimindo no ecrã uma imagem composta.

De seguida, foi essencial converter o *Estado* atualizado em outputs gráficos, para isso criou-se a função **desenhaEstado**, que permite obter o jogo pretendido consoante o *Mapa* desejado.

Já para receber inputs do Utilizador, o grupo desenvolveu a função **reageEvento** que gera *Estados* do jogo atualizados respetivamente.

Por fim, de modo a atualizar o *Estado* em si, desenvolveu-se a função **reageTempo** que altera o *Jogo* segundo a função **atualizaMovimenta** (junção da função "atualiza" e da função "movimenta" correspondentes à *Tarefa* 4, ver 3.4, e à *Tarefa* 3, ver 3.3, respetivamente)

É de notar que o **Jogo** permite apenas um *Jogador* que compete contra 3 *bots*, criados pelo grupo para a *Tarefa* 6.

3.6 Tarefa 6 - Implementar uma Estratégia de Corrida

Função Objetivo:

```
bot :: Tempo -> Jogo -> Int -> Acao
```

A *Tarefa 6* consistiu na criação de um bot, que percorre um dado *Mapa* segundo as *Ações* ponderadas pelo Grupo.

Primeiramente, com recurso ao tipo *Percurso* (ver 3.2) o *bot* identifica por onde se vai movimentar, tendo em atenção a *Peça* atual e a *Peça* futura. De modo a analisar as *peças* relevantes, o grupo elaborou as funções **pecaAtual** e a **pecaFutura**, que identificam as *Peças* respetivamente.

Mais tarde, foi necessária as funções **muda** e **identifica** para identificar no *Percurso* a *Posição/Peça* onde se encontra o *Carro*.

Por fim, as funções **curvaNO**, **retaNITRO** e **curvaYES** atribuem *Ações* consoante a *velocidade* do *Carro*, de forma a garantir uma boa performance no percurso, com base nas *Peças*.


Função Objetivo sugerida:


```
bot :: Tempo -> Jogo -> Int -> Acao
bot tick (Jogo mapa p c n h) i
  | vel0 c i = curvaYES i (pecaAtual percurso) (pecaFutura percurso) c
  | vel1 c i = retaNITRO i (pecaAtual percurso) (pecaFutura percurso) c
  | otherwise = curvaNO i (pecaAtual percurso) (pecaFutura percurso) c
                                where percurso = muda i (iPercurso mapa) c
```


Capítulo 4


Validação da Solução


Para validar a solução do grupo, em cada Tarefa foram realizadas diferentes formas de testar:

 Para a **Tarefa 1** e para a **Tarefa 2** (ver 3.1 e 3.2) foram realizados testes que iriam ser verificados no oráculo criado pelos docentes.

 Na **Tarefa 3** (ver 3.3), para além dos testes que se criaram para a posterior verificação do oráculo, desenvolveu-se uma função auxiliar, a função **m**, que facilitou a realização e a leitura dos testes por parte do grupo.

 **Tarefa 4** (ver 3.4) o grupo também escolheu criar funções que facilitassem a sua validação, as funções **a0**, **a1**, **a2** e **a3**, cada uma referindo-se ao jogador em causa.

 Para validar a **Tarefa 5** (ver 3.5) o grupo testou o executável criado e foi ajustando os eventuais problemas.

 Finalmente, na **Tarefa 6** (ver 3.6) usou-se o torneio e tendo em conta as corridas que se iam realizando fez-se ajustes às funções conforme o comportamento do *Carro*.

Capítulo 5

Conclusão

Em suma, elaborar este projeto contribuiu não só para a aprendizagem da Linguagem *Haskell*, mas também alargou os nossos conhecimentos no que toca à cooperação em grupo e ao uso do *SVN*.

Para além disso, sentimos que este trabalho contribuiu de forma positiva para nos mostrar outras vertentes de uma linguagem funcional (neste caso *Haskell*), como é o caso do *Gloss*, que nos ajudou na elaboração da interface gráfica.