

# Tanks

1ª fase do projeto de Laboratórios de Informática 1 2018/19

## Introdução

Neste enunciado apresentam-se as tarefas referentes à primeira fase do projecto da unidade curricular de Laboratórios de Informática I. O projecto será desenvolvido por grupos de 2 elementos, e consiste em pequenas aplicações Haskell que deverão responder a diferentes tarefas (apresentadas adiante).

O objetivo do projeto deste ano é implementar um jogo 2D de combates entre pequenos tanques. A ideia geral do jogo é colocar disparar tiros de forma a atingir os tanques adversários, destruindo inimigos e obstáculos no mapa. Os tanques e os tiros deslocam-se numa grelha, existindo diferentes tipos de tiros.



## Tarefas

**Importante:** Cada tarefa deverá ser desenvolvida num módulo Haskell independente, nomeado `Tarefan_20171i1gxxx.hs`, em que `n` é o número da tarefa e `xxx` o número do grupo, que estará associado ao repositório SVN de cada grupo. Os grupos **não devem alterar** os nomes, tipos e assinaturas das funções previamente definidas, sob pena de não serem corretamente avaliados.

Para aceder pela primeira vez ao repositório SVN do seu grupo, pode executar o seguinte comando (substituindo g999 pelo número do seu grupo e a1 pelo nome do seu utilizador):

```
svn checkout svn://svn.alunos.di.uminho.pt/2018li1g999 --username 2018li1g999a1
```

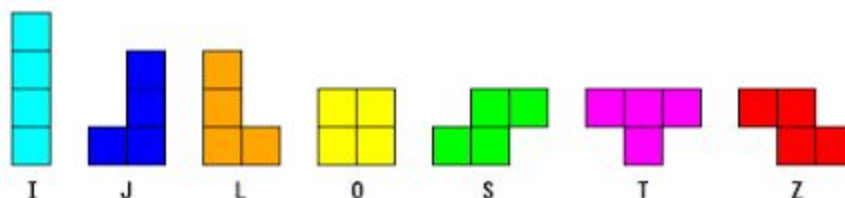
## Tarefa 1 - Construir mapas


A primeira tarefa consiste em desenvolver a funcionalidade de um editor de mapas baseado em peças de *Tétris*. Desta forma, construir um mapa consiste em ir *colocando peças de Tétris* numa grelha, substituindo o conteúdo anteriormente presente nas casas onde a peça é colocada. O processo de colocação de peças é descrito como uma sequência de instruções. Seguem-se as descrições das **peças de Tétris (Tetrominós)** e das **instruções** para construir um mapa.

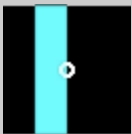

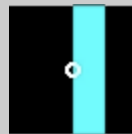

### Tetrominós

























Considerem 7 tipos de peças de Tétris, denominadas *Tetrominós* (<https://en.wikipedia.org/wiki/Tetromino>), identificados pelas letras I, J, L, O, S, T, Z, representadas abaixo pela respetiva ordem.

data Tetromino = I | J | L | O | S | T | Z



Estes 7 Tetrominós são classificados de uniláteros por permitirem criar todas as combinações de peças conhecidas do jogo Tétris utilizando apenas rotações. Na tabela abaixo, as direções são dadas pelas letras “C” (cima), “D” (direita), “B” (baixo), e “E” (esquerda). Por exemplo, a peça “L” com orientação para a esquerda é dada pelo desenho na linha “L” e coluna “E” da tabela. (  )

Peça \ Direção	C	D	B	E
I				

J				
L				
O				
S				
T				
Z				

## Mapas

Um mapa é uma grelha ou matriz de peças, em que cada peça é um bloco de parede ou vazia. Um bloco de parede pode ter diferentes propriedades, e ser nomeadamente destrutível ou indestrutível.

```
type Mapa = [[Peca]]
data Peca = Bloco Parede | Vazia
data Parede = Indestrutivel | Destrutivel
```

Um exemplo de um mapa simples de dimensão 10x10 pode ser codificado como:

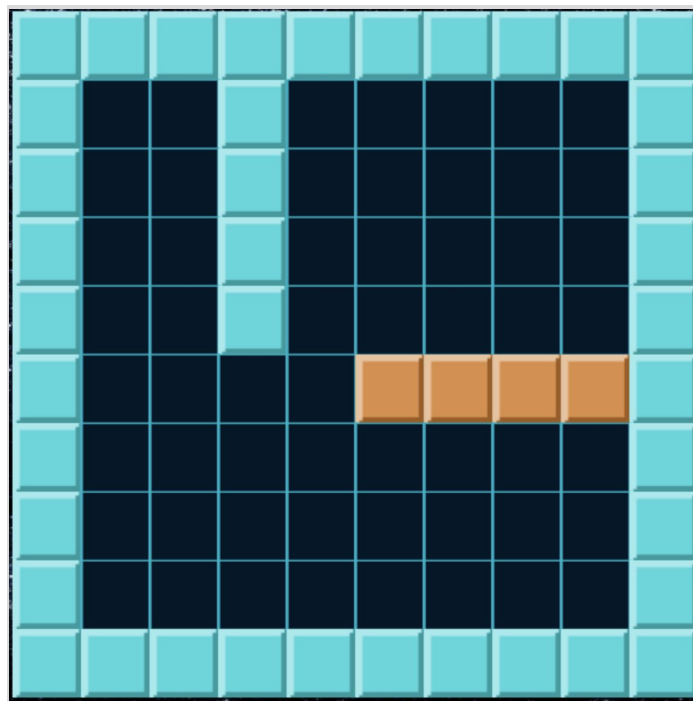
```
m =
  [[Bloco Indestrutivel,Bloco Indestrutivel,Bloco Indestrutivel
    ,Bloco Indestrutivel,Bloco Indestrutivel,Bloco Indestrutivel
    ,Bloco Indestrutivel,Bloco Indestrutivel,Bloco Indestrutivel
    ,Bloco Indestrutivel]
   ,[Bloco Indestrutivel,Vazia,Vazia,Bloco Indestrutivel,Vazia
    ,Vazia,Vazia,Vazia,Vazia,Bloco Indestrutivel]
```

```

, [Bloco Indestrutivel, Vazia, Vazia, Bloco Indestrutivel, Vazia
, Vazia, Vazia, Vazia, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Bloco Indestrutivel, Vazia
, Vazia, Vazia, Vazia, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Bloco Indestrutivel, Vazia
, Vazia, Vazia, Vazia, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Vazia, Vazia, Bloco Destrutivel
, Bloco Destrutivel, Bloco Destrutivel, Bloco Destrutivel, Bloco
Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia
, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia
, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia, Vazia
, Vazia, Bloco Indestrutivel]
, [Bloco Indestrutivel, Bloco Indestrutivel, Bloco Indestrutivel
, Bloco Indestrutivel, Bloco Indestrutivel, Bloco Indestrutivel
, Bloco Indestrutivel, Bloco Indestrutivel, Bloco Indestrutivel
, Bloco Indestrutivel]]

```

Visualmente, o mapa m corresponde à seguinte figura:



Pode visualizar graficamente este e outros mapas que defina em <https://li1.lsd.di.uminho.pt/mapviewer/MapView.jsexe/run.html>.

## Instruções

Um editor de mapas permite construir mapas de forma interactiva e mais compacta, começando com um mapa vazio, e aplicando uma sequência de instruções. Uma instrução dada a um editor é então codificada no seguinte tipo:

```
data Instrucao =  
    Move Direcao | Roda | MudaTetromino | MudaParede | Desenha  
data Direcao = C | D | B | E
```

Por ordem, uma instrução permite mover o cursor numa dada direção (cima **C**, direita **D**, baixo **B**, esquerda **E**), rodar, mudar o tipo de peça ou parede da próxima peça a colocar, ou desenhar a próxima peça com a posição, rotação e tipos atuais. Para compreender melhor o comportamento esperado de cada instrução pode experimentar um editor de mapas interativo em <https://li1.lsd.di.uminho.pt/mapeditor/MapEditor.jsexe/run.html>.

Como exemplo, o mapa **m** acima pode ser construído com a seguinte sequência de instruções:

```
is = [Move D,Desenha,Move B,Move B,Move B,Roda,MudaParede  
      ,Move D,Move D,Move D,Desenha,Move B]
```

Note que um mapa contém sempre uma borda de parede indestrutível.

## Funções a implementar

O objectivo desta tarefa é definir a função

```
constroi :: Instrucoes -> Mapa
```

que constrói um mapa que corresponde à sequência de instruções recebida. Por exemplo, deverá ser verdade que `constroi is == m`, usando as instrução `is` e o mapa `m` definidos acima. Pode consultar o resultado de `constroi is` no visualizador de mapas. Um detalhe **muito importante** é que a dimensão do mapa resultante deve ser **sempre** a dada pela função `dimensaoInicial` - em particular isto garante que ao processar cada `instrucao` o cursor ou tetrominós a ser desenhados cabem sempre dentro das bordas do mapa.

Como auxílio à resolução desta tarefa, são providenciadas também algumas assinaturas de funções auxiliares que deve completar e utilizar. Consulte a documentação online da Tarefa 1 ([https://li1.lsd.di.uminho.pt/doc/src/Tarefa1\\_2018li1g000.html](https://li1.lsd.di.uminho.pt/doc/src/Tarefa1_2018li1g000.html)) para mais detalhes.

## Tarefa 2 - Efetuar jogadas

O objectivo desta tarefa é, dada uma descrição do estado do jogo e uma jogada de um dos jogadores, determinar o efeito dessa jogada no estado do jogo.

### Jogadas

Uma jogada consiste numa movimentação numa dada direção ou o disparo de uma arma, de acordo com o seguinte tipo:

```
data Jogada = Movimenta Direcao | Dispara Arma
```

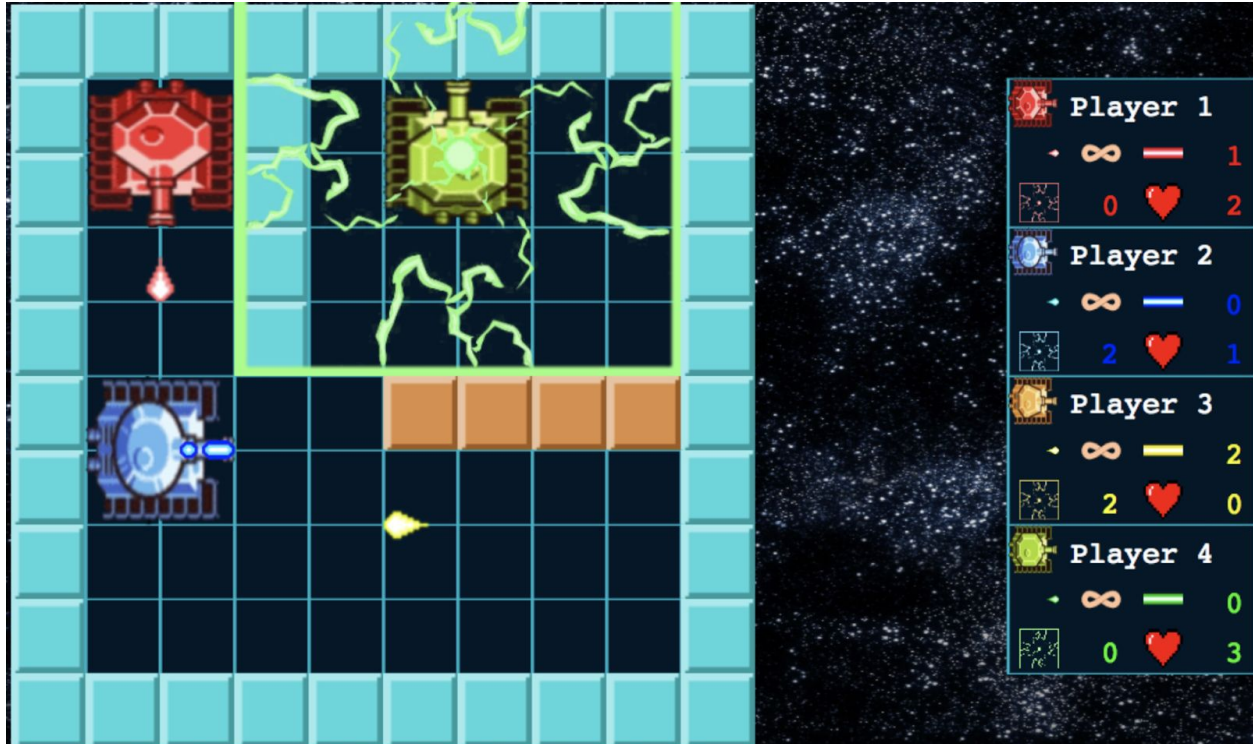
### Armas

Cada jogador possui 3 armas: um canhão que dispara tiros que se deslocam faseadamente ao longo do eixo de disparo, um laser que dispara raios que afetam instantaneamente qualquer obstáculo no eixo de disparo e uma arma de choques que bloqueia os sistemas de transmissão de outros tanques numa área circundante.

```
data Arma = Canhao | Laser | Choque
```

### Estado do jogo

Considere a seguinte imagem que representa graficamente um estado do jogo:



Mais concretamente, o estado do jogo consiste num mapa e em informação adicional sobre jogadores e disparos de armas em curso, e é representado pelo seguinte formato:

```
data Estado = Estado { mapaEstado      :: Mapa
                      , jogadoresEstado :: [Jogador]
                      , disparosEstado  :: [Disparo] }

```

O estado de exemplo pode ser portanto representado pelo valor:

e = Estado m js ds

O estado  $e$  contém 4 jogadores (cuja cores correspondem às cores na figura):

```
js = [jogador1, jogador2, jogador3, jogador4]
jogador1 = Jogador (1,1) B 2 1 0
jogador2 = Jogador (5,1) D 1 0 2
jogador3 = Jogador (6,5) E 0 2 2
jogador4 = Jogador (1,5) C 3 0 0
```



## Jogadores

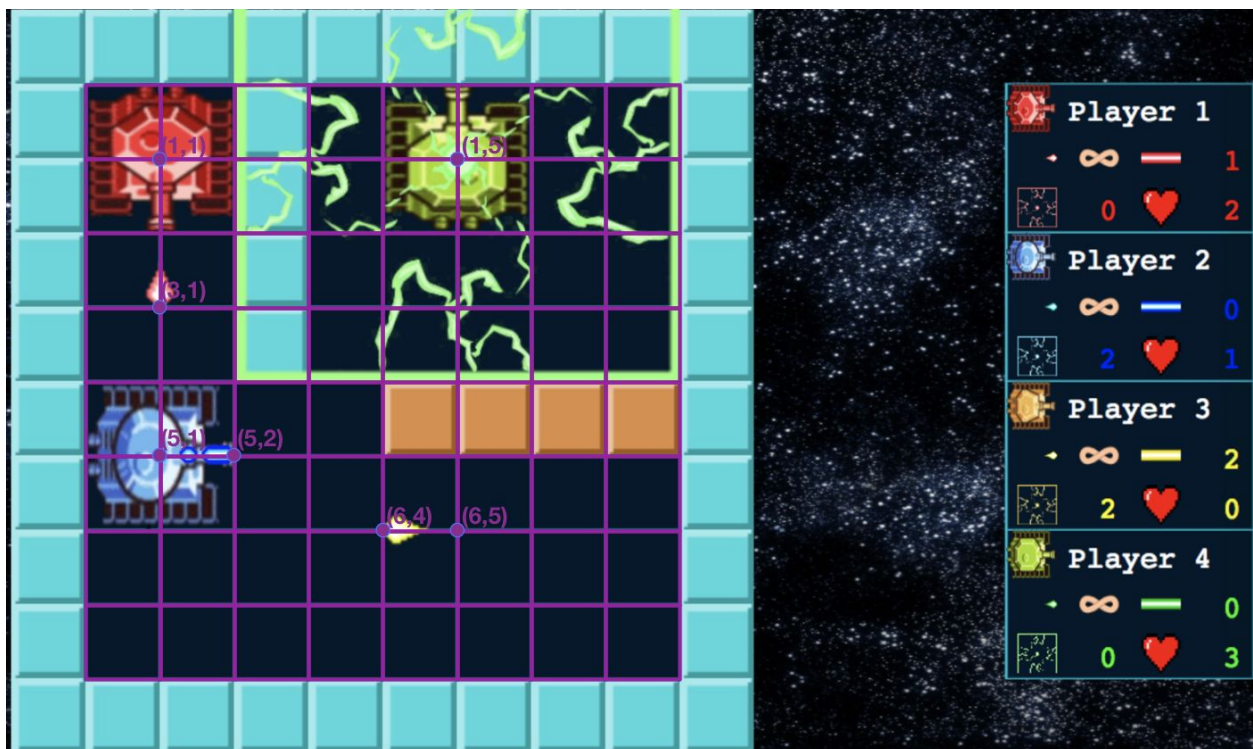
Um jogador é identificado pelo seu índice na lista (contido entre 0 e 3, havendo no máximo 4 jogadores em jogo), e possui informação tal como a posição e direção do seu tanque, um número de vidas maior ou igual a 0 e números de munições das suas armas maior ou igual a 0:

```
data Jogador = Jogador { posicaoJogador :: PosicaoGrelha
                        , direcaoJogador :: Direcao
                        , vidasJogador   :: Int
                        , lasersJogador  :: Int
                        , choquesJogador :: Int }
type PosicaoGrelha = (Int,Int)
```

Como o tanque de cada jogador ocupa sempre 4 peças do mapa, a sua posição (que corresponde ao ponto no centro do tanque) é dada num sistema de coordenadas sobreposto sobre a grelha do mapa (distinguido pelo tipo `PosicaoGrelha`).

## Disparos

Cada jogador possui um número ilimitado de munições de `Canhao`, mas números limitados de munições de `Laser` e de `Choque`, que são gastos a cada disparo. A seguinte imagem realça a grelha do mapa de exemplo e respetivas posições utilizadas com a cor **roxo**:





Neste estado temos também 4 disparos (note que, ao contrário dos jogadores, a ordem dos disparos na lista é irrelevante):

```
ds = [disparo1,disparo2,disparo3,disparo4]
disparo1 = DisparoCanhao 0 (3,1) B
disparo2 = DisparoLaser 1 (5,2) D
disparo3 = DisparoCanhao 2 (6,4) E
disparo4 = DisparoChoque 3 2
```

Um disparo é sempre efetuado por um jogador com uma determinada arma. Quando um disparo de `Canhao` ou `Laser` é efetuado (como resultado de uma jogada), ele é colocado na próxima posição e com a mesma direção do jogador que efetuou o disparo. No caso da arma de `Choque`, um disparo não possui uma posição ou uma direção, mas afeta uma área circundante do jogador por um determinado período de tempo (`Ticks`).

```
data Disparo
  = DisparoCanhao
    { jogadorDisparo :: Int
    , posicaoDisparo  :: PosicaoGrelha
    , direcaoDisparo :: Direcao bala
    }
  | DisparoLaser
    { jogadorDisparo :: Int
    , posicaoDisparo  :: PosicaoGrelha
    , direcaoDisparo :: Direcao
    }
  | DisparoChoque
    { jogadorDisparo :: Int
    , tempoDisparo   :: Ticks
    }
```

O processamento de uma jogada deverá ser efetuado de acordo com as seguintes regras:

- Um tanque muda apenas de direção quando se movimenta numa direção diferente da sua direção atual, sem mudar a sua posição.
- Um tanque desloca-se numa direção quando se movimenta na sua direção atual e as 2 novas peças de mapa que deseja ocupar se encontram livres (i.e., não existe outro tanque vivo ou parede a ocupar essa peça).
- Um tanque pode deslocar-se se não existe um disparo de `Choque` efetuado por outro jogador (vivo ou morto) que afete as 4 peças de mapa onde atualmente se encontra. Mesmo sob o efeito de um `Choque` um tanque pode mudar de direção.

- Um tanque que esteja morto (número de vidas igual a 0) não pode efetuar jogadas.
- Um disparo de `Laser` ou de `Canhao` só pode ser efetuado se o jogador em causa tiver munições suficientes, e gasta 1 munição.
- Um disparo de `Choque` imobiliza todos os tanques (que não o autor do disparo) que estejam numa peça numa área de 6x6 à volta do autor do disparo durante um período de 5 instantes de tempo.

Note que um tanque pode deslocar-se para uma direção onde exista um disparo, desde que não esteja imobilizado por um disparo de choque. Os tiros não se movem durante uma jogada, e os efeitos dos tiros (em jogadores e paredes) não são manifestados na jogada – isto será tratado na Tarefa 4.

## Funções a implementar

O objectivo desta tarefa é definir a função

```
jogada :: Int -> Jogada -> Estado -> Estado
```

que efetua uma jogada para um determinado jogador sobre um determinado estado, retornando o novo estado após a jogada ser efetuada. Consulte a documentação online da Tarefa 2 ([https://li1.lsd.di.uminho.pt/doc/src/Tarefa2\\_2018li1g000.html](https://li1.lsd.di.uminho.pt/doc/src/Tarefa2_2018li1g000.html)) para mais detalhes.

## Tarefa 3 - Comprimir o estado do jogo

O objectivo desta tarefa é, dada uma descrição do estado do jogo (idêntica à utilizada na tarefa anterior) implementar um mecanismo de compressão / descompressão que permita poupar caracteres e, desta forma, poupar espaço em disco quando o estado do jogo for gravado (permitindo, por exemplo, fazer pausa durante o jogo com o objectivo de o retomar mais tarde). Uma sugestão que pode implementar passa por eliminar redundâncias na representação do estado do jogo, nomeadamente na parte do mapa: tal como mencionado na tarefa 1 existem células do mapa cujo conteúdo é fixo, não sendo na prática necessário guardar informação sobre o conteúdo dessas células. Esta é apenas uma sugestão existindo muitas outras técnicas que podem implementar para obter melhores taxas de compressão.

## Funções a implementar

O objectivo desta tarefa é definir as funções

```
comprime    :: Estado -> String  
descomprime :: String -> Estado
```

cujo objectivo é, respectivamente, codificar o estado do jogo para uma `String` e voltar a decodificar. O mecanismo de codificação tem que ser invertível, ou seja, dado um estado do jogo qualquer `e`, tem que ser sempre verdade que `descomprime (comprime e) == e`. Uma forma trivial de satisfazer este critério é não fazer qualquer compressão, usando as definições:

```
comprime = show
descomprime = read
```

No entanto, pretende-se uma implementação de uma boa função de compressão, ou seja que, para um estado do jogo qualquer  $e$ , `length (comprime e)` seja bastante menor do que `length (show e)`. Consulte a documentação online da Tarefa 3 ([https://li1.lsd.di.uminho.pt/doc/src/Tarefa3\\_2018li1g000.html](https://li1.lsd.di.uminho.pt/doc/src/Tarefa3_2018li1g000.html)) para mais detalhes.

## Sistema de *Feedback*

O projecto inclui também um Sistema de *Feedback*, alojado em <http://li1.lsd.di.uminho.pt> que visa fornecer suporte automatizado e informações personalizadas a cada grupo e simultaneamente incentivar boas práticas no desenvolvimento de software (documentação, teste, controle de versões, etc). Esta página *web* permite aos alunos pedir feedback relativo ao seu trabalho de grupo presente no seu repositório SVN e fornece informação detalhada sobre vários tópicos, nomeadamente:

- Resultados de testes unitários, comparando a solução do grupo com um oráculo (solução ideal) desenvolvido pelos docentes;
- Relatórios de ferramentas automáticas (que se incentiva os alunos a utilizar) que podem conter sugestões úteis para melhorar a qualidade global do trabalho do grupo;
- Visualizadores gráficos de casos de teste utilizando as soluções do grupo e o oráculo.

Note que as credenciais de acesso ao Sistema de *Feedback* são as mesmas que as credenciais de acesso ao SVN.

Para receber *feedback* sobre as tarefas e qualidade dos testes, devem ser declaradas no ficheiro correspondente a cada tarefa as seguintes listas de testes:

```
testesT1 :: [Instrucoes]
testesT2 :: [(Int, Jogada, Estado)]
testesT3 :: [Estado]
```

Como exemplo, podem-se utilizar os exemplos acima definidos como casos de teste:

```
testesT1 = [is]
testesT2 = [(0, Movimenta B, e)]
testesT3 = [e]
```

Estas definições podem ser então colocadas no ficheiro correspondente a cada tarefa para que sejam consideradas pelo Sistema de *Feedback*. Note que uma maior quantidade e diversidade de testes garante um melhor *feedback* e ajudará a melhorar o código desenvolvido.

# Entrega e Avaliação

A data limite para conclusão de todas as tarefas desta primeira fase é **16 de Novembro de 2018 às 23h59m59s (Portugal Continental)** e a respectiva avaliação terá um peso de 50% na nota final do projeto. A submissão será feita automaticamente através do SVN: nesta data será feita uma cópia do repositório de cada grupo, sendo apenas consideradas para avaliação os programas e demais artefactos que se encontrem no repositório nesse momento. O conteúdo dos repositórios será processado por ferramentas de detecção de plágio e, na eventualidade de serem detectadas cópias, estas serão consideradas fraude dando-se-lhes tratamento consequente.

Para além dos programas Haskell relativos às 3 tarefas, será considerada parte integrante do projeto todo o material de suporte à sua realização armazenado no repositório SVN do respectivo grupo (código, documentação, ficheiros de teste, etc.). A utilização das diferentes ferramentas abordadas no curso (como Haddock, SVN, etc.) deve seguir as recomendações enunciadas nas respectivas sessões laboratoriais. A avaliação desta fase do projecto terá em linha de conta todo esse material, atribuindo-lhe os seguintes pesos relativos:

Componente	Peso
Avaliação automática da Tarefa 1	20%
Avaliação automática da Tarefa 2	20%
Avaliação automática da Tarefa 3	20%
Qualidade do código	15%
Qualidade dos testes	10%
Documentação do código usando o Haddock	10%
Utilização do SVN e estrutura do repositório	5%

A avaliação automática será feita através de um conjunto de testes que não serão revelados aos grupos. No caso da Tarefa 3, a avaliação automática também terá em conta o nível de compressão atingido. A avaliação qualitativa incidirá sobre aspectos de qualidade de código (por exemplo, estrutura do código, elegância da solução implementada, etc.), qualidade dos testes (quantidade, diversidade e cobertura dos mesmos), documentação (estrutura e riqueza dos comentários) e bom uso do SVN como sistema de controle de versões.