utad UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

TRABALHO PRÁTICO PARTE 1

Programação Funcional | Engenharia Informática

Trabalho Realizado por:

Francisco Ruano, al78474

Índice

1. Introdução

No âmbito da unidade curricular de Programação Funcional, foi-nos proposto o desenvolvimento de uma aplicação, recorrendo à linguagem *Haskell*, que simule a operação de uma nave alienígena em um espaço tridimensional.

A primeira parte do projeto inclui a criação de diversas funções capazes de gerir o estado da nave, as suas movimentações e verificar colisões.

No presente relatório, serão apresentadas em detalhe as funcionalidades implementadas, os métodos utilizados para resolver cada problema específico e a lógica subjacente ao comportamento da nave, de forma a ilustrar as decisões tomadas no desenvolvimento da aplicação.

2. Tipos de Dados

Inicialmente, foram definidos tipos de dados específicos, utilizados ao longo do código, que representam os estados e movimentações de uma ou mais naves. Estes tipos de dados melhoram a legibilidade do código e facilitam a manipulação de informações complexas de forma clara e concisa.

a. **Localização:** Define a localização de uma nave no espaço tridimensional através de uma tupla de três inteiros. Cada elemento da tupla representa uma coordenada (x,y,z), permitindo que a nave seja posicionada em um ponto específico no espaço.

b. **EstadoNave:** Representa o estado completo de uma nave, combinando a localização (tipo Localizacao) e o estado de ligação da nave (valor Bool). O valor booleano indica se a nave se encontra "ligada" (*True*) ou "desligada" (*False*).

c. **Movimentacao:** Este define o deslocamento de uma nave em cada eixo (x, y, z) em uma única operação de movimento e é representada por uma dupla de inteiros.

d. **ID:** Este tipo de dado é um identificador único para cada nave, e facilita-nos na distinção das mesmas, caso haja movimentações ou colisões.

e. **ListaNaves:** Este tipo de dado faz a combinação de uma lista de movimentos (*[Movimentacao]*) e o identificador (*ID*) de uma nave especifica.

```
type Localizacao = (Int, Int, Int)
type EstadoNave = (Localizacao, Bool)
type Movimentacao = (Int, Int, Int)
type ID = Int
type ListaNaves = ([Movimentacao], ID)
```

Figura 1: Tipos de Dados

3. Funções Auxiliares

a. **atualiza_acao':** Esta função é responsável por atualizar o estado de ligação de uma nave, ou seja, recebe uma dupla com a ação "ligar" ou "desligar", através de um booleano (Bool), e o estado atual da nave. De seguida, devolve um novo estado (EstadoNave) atualizado, sem alterar a localização que já se encontrava definida.

- **Exemplo:** Se uma nave está desligada e a função é chamada com True, a nave passa a estar ligada, mantendo a localização.

b. **move':** Esta função está encarregue da movimentação da nave. Quando recebe as novas coordenadas de movimentação (Movimentação) e o estado atual da nave (EstadoNave), retorna um novo estado e executa a movimentação da mesma.

- **Exemplo:** Se a nave está na posição (1, 2, 3) e a movimentação é (2, 0, -1), a nova posição será (3, 2, 2).

c. **move_lista':** Esta função é responsável por aplicar uma lista de movimentos e um EstadoNave, e aplica cada movimento da lista ao estado atual, devolvendo o estado final da nave apos todos os movimentos

- **Exemplo:** Se a nave está na posição (0, 0, 0) e a lista de movimentos é [(1,1,1), (2,0,0)], a nave terminará na posição (3, 1, 1).

d. **move_varios':** Esta função aplica várias movimentações a várias naves. Sempre que esta recebe uma lista de naves (ListaNaves) e uma lista de estados correspondente às mesmas (EstadoNave), retorna uma lista de tuplas que representam o estado final de cada nave após as movimentações.

e. **verifica_embates':** Esta função verifica se há embates (colisões) entre naves, ou seja, compara a localização de uma nave com as localizações das outras naves, retornando True se houver mais de uma nave na mesma posição. Caso contrário, retorna False.

```haskell
atualiza_acao' :: Bool -> EstadoNave -> EstadoNave
atualiza_acao' ligacao (localizacao, _) = (localizacao, ligacao)

move' :: Movimentacao -> EstadoNave -> EstadoNave
move' (dx, dy, dz) ((x, y, z), ligacao) = ((x + dx, y + dy, z + dz), ligacao)

move_lista' :: [Movimentacao] -> EstadoNave -> EstadoNave
move_lista' [] estado = estado
move_lista' (mov:movs) estado =
    let novoEstado = move' mov estado
    in move_lista' movs novoEstado

move_varios' :: [ListaNaves] -> [EstadoNave] -> [(EstadoNave, ID)]
move_varios' [] _ = []
move_varios' _ [] = error "Número insuficiente de estados iniciais para as naves."
move_varios' ((movs, id):restoNaves) (estado:restoEstados) =
    let estadoFinal = move_lista' movs estado
    in (estadoFinal, id) : move_varios' restoNaves restoEstados

-- Função para verificar embates entre as naves
verifica_embates' :: EstadoNave -> [(EstadoNave, ID)] -> Bool
verifica_embates' (local, _) estados =
    length [id | ((loc, _), id) <- estados, loc == local] > 1
```

Figura 2: Definição das Funções Auxiliares

4. Função main

A função *main* atua como um ponto de entrada no programa, executando as operações iniciais e configurando o estado inicial da nave. (Figura 3)

Começa com uma introdução ao trabalho prático, incluindo o título e autores do mesmo. Em seguida, solicita ao utilizador que defina as coordenadas da nave, no espaço tridimensional, bem como o seu estado inicial (ligada ou desligada). Após estabelecer todos estes parâmetros, é invocada a função *menu*, que permite ao utilizador interagir com as demais funcionalidades do programa de forma dinâmica, proporcionando uma navegação contínua entre as opções.

Esta estrutura inicial garante que o utilizador tenha o controlo sobre a configuração e operação da nave desde o início da execução do programa.

```
 TRABALHO PRÁTICO PARTE 1 - PROGRAMAÇÃO FUNCIONAL

Trabalho realizado por :
 Francisco Ruano Al 78474

Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
True
O estado da aeronave inicialmente é: (1,2,3), True
```

Figura 3: Execução da Função main

5. Função menu

Esta função exibe um menu iterativo ao utilizador, permitindo-lhe escolher uma das várias opções apresentadas, tais como atualizar o estado da nave, mover a nave, mover uma lista de naves, verificar embates ou mover várias naves de forma atualizada (Figura 5).

Dependendo da escolha do utilizador, a função *menu* chama a função apropriada para determinada tarefa e reinicia o menu (Figura 4).

```haskell
-- Menu principal atualizado
menu :: EstadoNave -> IO ()
menu estadoAtual = do
    putStrLn "\nSelecione a opção que deseja visualizar: \n1- Atualiza Ação \n2- Move \n3- Move_Lista \n4- Move_Varios \n5- Verificar_Embates \n6- Move_Varios_Atualizada"
    putStrLn "Opção:"
    op <- getLine

    case op of
        "1" -> do
            estado_Up <- atualiza_acao estadoAtual
            menu estado_Up

        "2" -> do
            estado_move <- move estadoAtual
            menu estado_move

        "3" -> do
            estado_move <- move_lista estadoAtual
            menu estado_move

        "4" -> do
            resultados <- move_varios
            menu estadoAtual

        "5" -> do
            resultados <- move_varios
            verifica_embates resultados
            menu estadoAtual

        "6" -> do
            resultados <- move_varios_atualizada
            menu estadoAtual

        _ -> putStrLn "Opção Inválida. Tente novamente."
```

Figura 4: Função menu

```
Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
```

Figura 5: Compilação da Função menu

6. Função atualiza_acao

A função *atualiza_acao* tem como objetivo permitir que o utilizador atualize o estado de ligação de uma nave (ligada ou desligada). Esta lê um valor booleano que define esse parâmetro, e atualiza o EstadoNave correspondente, exibindo o novo estado após a atualização (Figuras 6 e 7).

```haskell
-- Função para atualizar o estado da nave (ligar/desligar)
atualiza_acao :: EstadoNave -> IO EstadoNave
atualiza_acao estadoAtual = do
    putStrLn "Qual é o Estado (Ligado ou Desligado)? (True ou False)"
    lig <- getLine
    let ligado = read lig :: Bool
    let estado_Up = atualiza_acao' ligado estadoAtual
    putStrLn $ "O novo estado da nave é: " ++ show estado_Up
    return estado_Up
```

Figura 6: Função atualiza_acao

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
True
O estado da aeronave inicialmente é: (1,2,3), True

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
1
Qual é o Estado (Ligado ou Desligado)? (True ou False)
False
O novo estado da nave é: ((1,2,3),False)
```

Figura 7: Compilação da Função atualiza_acao

7. Função move

A função *move* permite que uma determinada nave se mova no espaço, através da introdução de coordenadas (Figura 8).

Caso a nave já se encontre ligada, a função solicita ao utilizador as coordenadas de movimentação da nave (Figura 9).

Se a nave estiver desligada, é questionado ao utilizador se deseja ligá-la para utilizar a função. Se o utilizador decidir ligar a nave, deve inserir uma nova movimentação que será aplicada ao estado da nave (Figura 10). Caso contrário não pode efetuar movimentos (Figura 11).

```haskell
-- Função para receber a movimentação e atualizar o estado da nave
move :: EstadoNave -> IO EstadoNave
move estadoAtual@(coord, ligado) = do
    if not ligado
        then do
            putStrLn "A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)"
            ligar <- getLine
            let ligarNave = read ligar :: Bool
            if ligarNave
                then do
                    let estadoLigado = (coord, True)
                    putStrLn "A nave foi ligada."
                    move estadoLigado
                else do
                    putStrLn "A nave permanece desligada, logo nao se pode movimentar."
                    return estadoAtual
        else do
            putStrLn "Digite a movimentação que deseja ((1, 2, 3)):"
            mov <- getLine
            let moviment = read mov :: Movimentacao
            let estado_move = move' moviment estadoAtual
            putStrLn $ "O novo estado da nave é: " ++ show estado_move
            return estado_move
```

Figura 8: Função move

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
True
O estado da aeronave inicialmente é: (1,2,3), True

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
2
Digite a movimentação que deseja ((1, 2, 3)):
(1,3,4)
O novo estado da nave é: ((2,5,7),True)
```

Figura 9: Compilação da Função move quando a nave se encontra ligada

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
False
O estado da aeronave inicialmente é: (1,2,3), False

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
2
A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)
True
A nave foi ligada.
Digite a movimentação que deseja ((1, 2, 3)):
(1,2,4)
O novo estado da nave é: ((2,4,7),True)
```

Figura 10: Compilação da Função move quando o utilizador solicita a ligação da nave

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
False
O estado da aeronave inicialmente é: (1,2,3), False

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
2
A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)
False
A nave permanece desligada, logo nao se pode movimentar.
```

Figura 11: Compilação da Função move quando a nave se encontra desligada

8. Função move_lista

Esta função permite que uma determinada nave se movimente no espaço, com base numa lista de movimentações fornecidas pelo utilizador (Figura 12).

À semelhança da função *move*, esta função também verifica se a nave se encontra ligada antes de a mover e solicita ao utilizador que a ligue para executar a movimentação (Figuras 13, 14 e 15).

```haskell
-- Função para receber múltiplas movimentações
move_lista :: EstadoNave -> IO EstadoNave
move_lista estadoAtual@(coord, ligado) = do
    if not ligado
        then do
            putStrLn "A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)"
            ligar <- getLine
            let ligarNave = read ligar :: Bool
            if ligarNave
                then do
                    let estadoLigado = (coord, True)
                    putStrLn "A nave foi ligada."
                    move_lista estadoLigado
                else do
                    putStrLn "A nave permanece desligada, logo nao se pode movimentar."
                    return estadoAtual
        else do
            putStrLn "Digite uma lista de movimentos (Exemplo: (1,2,3) (3,2,1) ...):"
            mov <- getLine
            let movs = map read (words mov) :: [Movimentacao]
            let estadoNovo = move_lista' movs estadoAtual
            putStrLn $ "O novo estado da nave é: " ++ show estadoNovo
            return estadoNovo
```

Figura 12: Função move_lista

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
True
O estado da aeronave inicialmente é: (1,2,3), True

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
3
Digite uma lista de movimentos (Exemplo: (1,2,3) (3,2,1) ...):
(1,2,3) (3,2,1)
O novo estado da nave é: ((5,6,7),True)
```

Figura 13: Compilação da Função move_lista quando a nave se encontra ligada

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
False
O estado da aeronave inicialmente é: (1,2,3), False

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
3
A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)
True
A nave foi ligada.
Digite uma lista de movimentos (Exemplo: (1,2,3) (3,2,1) ...):
(1,2,3) (3,2,1)
O novo estado da nave é: ((5,6,7),True)
```

Figura 14: Compilação da Função move_lista quando o utilizador solicita a ligação da nave

```
Qual é a localização Inicial que pretendes? (1 2 3)
1 2 3
Qual é o Estado (Ligado ou Desligado)? (True ou False)
False
O estado da aeronave inicialmente é: (1,2,3), False

Selecione a opção que deseja visualizar:
1- Atualiza Ação
2- Move
3- Move_Lista
4- Move_Varios
5- Verificar_Embates
6- Move_Varios_Atualizada
Opção:
3
A nave Encontra-se Desligada. Deseja ligá-la para se mover? (True, False)
False
A nave permanece desligada, logo nao se pode movimentar.
```

Figura 15: Compilação da Função move_lista quando a nave se encontra desligada

9. Função move_varios

A função *move_varios* permite que o utilizador forneça uma lista de movimentos e estados iniciais para várias naves ao mesmo tempo, calculando e exibindo os estados finais de cada uma, após a aplicação das respetivas movimentações (Figura 16). Para isso, deve recolher informações interactivamente com o utilizador e, com o auxilio da função *move_varios'*, executa os cálculos necessários (Figura 17).

```haskell
-- Função para mover várias naves e retornar os estados finais
move_varios :: IO [(EstadoNave, ID)]
move_varios = do
    putStrLn "Digite a lista de naves (movimentos,ID) ( [([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)] ):"
    naves <- getLine
    let navesLista = read naves :: [ListaNaves]

    putStrLn "Digite os estados iniciais das naves ( [((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)] ):"
    ests <- getLine
    let estadosIniciais = read ests :: [EstadoNave]

    let resultados = move_varios' navesLista estadosIniciais
    putStrLn $ "\nOs estados finais das naves são: " ++ show resultados
    return resultados
```

Figura 16: Função move_varios

```
Opção:
4
Digite a lista de naves (movimentos,ID) ( [([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)] ):
 [([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)]
Digite os estados iniciais das naves ( [((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)] ):
[((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)]

Os estados finais das naves são: [(((6,9,12),True),12345),(((7,8,9),False),67890)]
```

Figura 17: Compilação da Função move_varios

10. Função verifica_embates

A função *verifica_embates* tem como objetivo verificar se ocorreram colisões entre as naves, após as suas movimentações (Figura 18).

Ao identificar eventuais embates, esta função informa o utilizador do sucedido, permitindo monitorar e gerir o posicionamento das naves para evitar possíveis sobreposições indesejadas. Nas Figuras 19 e 20 podemos observar uma simulação de embate e outra sem ocorrência do mesmo, respetivamente.

```haskell
-- Função para verificar embates após a movimentação
verifica_embates :: [(EstadoNave, ID)] -> IO ()
verifica_embates resultados = do
    let embates = [(estado, id) | (estado, id) <- resultados, verifica_embates' estado resultados]
    if null embates
        then putStrLn "Não houve embates."
        else putStrLn $ "Houve embates nas seguintes naves: " ++ show embates
```

Figura 18: Função verifica_embates

```
Opção:
5
Digite a lista de naves (movimentos,ID) ( [([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)] ):
[([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)]
Digite os estados iniciais das naves ( [((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)] ):
[((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)]

Os estados finais das naves são: [(((6,9,12),True),12345),(((7,8,9),False),67890)]
Não houve embates.
```

Figura 19: Exemplo da Função verifica_embates quando não ocorrem embates

```
Opção:
5
Digite a lista de naves (movimentos,ID) ( [([(1,2,3), (4,5,6)], 12345), ([(1,1,1), (2,2,2)], 67890)] ):
[([(1,0,0), (0,1,0)], 1), ([(1,1,0), (0,0,1)], 2)]
Digite os estados iniciais das naves ( [((1,2,3),True), ((4,5,6),False), ((1,1,1),True), ((2,2,2),False)] ):
[((1,2,3), True), ((1,2,2), True)]

Os estados finais das naves são: [(((2,3,3),True),1),(((2,3,3),True),2)]
Houve embates nas seguintes naves: [(((2,3,3),True),1),(((2,3,3),True),2)]
```

Figura 20: Exemplo da Função verifica_embates quando ocorrem embates

11. Função move_varios_atualizada

A função *move_varios_atualizada* executa a movimentação de várias naves e, em seguida, verifica se ocorreram colisões entre elas (Figura 21). O objetivo principal desta função é filtrar e exibir apenas os estados finais das naves que não colidiram, assegurando um relatório de posicionamento livre de embates (Figura 22 e 23).

```haskell
-- Função para mover várias naves e retornar os estados finais sem embates
move_varios_atualizada :: IO [(EstadoNave, ID)]
move_varios_atualizada = do
    putStrLn "Digite a lista de naves (movimentos,ID) ( [([(1,0,0), (0,1,0)], 1), ([(1,0,0), (0,0,1)], 2)] ):"
    naves <- getLine
    let navesLista = read naves :: [ListaNaves]

    putStrLn "Digite os estados iniciais das naves ([((1,2,3), True), ((1,2,3), True)] ):"
    ests <- getLine
    let estadosIniciais = read ests :: [EstadoNave]

    let resultados = move_varios' navesLista estadosIniciais
    let estadosFinais = filter (\(estado, id) -> not (verifica_embates' estado resultados)) resultados
    putStrLn $ "\nOs estados finais das naves sem embates são: " ++ show estadosFinais
    return estadosFinais
```

Figura 21: Função move_varios_atualizada

```
Digite a lista de naves (movimentos,ID) ( [([(1,0,0), (0,1,0)], 1), ([(1,0,0), (0,0,1)], 2)] ):
[([(1,0,0), (0,1,0)], 1), ([(1,0,0), (0,0,1)], 2)]
Digite os estados iniciais das naves ([((1,2,3), True), ((1,2,3), True)] ):
[((1,2,3), True), ((1,2,3), True)]

Os estados finais das naves sem embates são: [(((2,3,3),True),1),(((2,2,4),True),2)]
```

Figura 22: Quando não ocorre embates

```
Digite a lista de naves (movimentos,ID) ( [([(1,0,0), (0,1,0)], 1), ([(1,0,0), (0,0,1)], 2)] ):
[([(1,0,0), (0,1,0)], 1), ([(1,0,0), (0,0,1)], 2)]
Digite os estados iniciais das naves ([((1,2,3), True), ((1,2,3), True)] ):
[((1,2,3), True), ((1,3,2), True)]

Os estados finais das naves sem embates são: []
```

Figura 23: Quando ocorre embates

12. Conclusão

Em conclusão, a realização deste trabalho prático revelou-se fundamental para aprofundar a compreensão da linguagem de programação *Haskell* e das suas particularidades no contexto da programação funcional. Através do desenvolvimento de funcionalidades para a operação de uma nave alienígena, foi possível explorar conceitos-chave, como a imutabilidade de dados, recursão e a construção de funções puras e auxiliares, além de aprender a manipular estruturas de dados complexas.

Este trabalho também demonstrou como a linguagem *Haskell*, apesar da sua abordagem rigorosa e abstrata, pode ser eficaz na resolução de problemas concretos, promovendo um pensamento lógico e estruturado.

A experiência adquirida contribuiu para consolidar habilidades analíticas, aprimorar o raciocínio algorítmico e aplicar conceitos de programação funcional a cenários práticos, reforçando o valor do *Haskell* para desenvolver soluções robustas e eficientes.