

Projeto de LI1 - *LightBot*

João Coelho A74859

Fábio Baião A75662

16 de Fevereiro de 2015

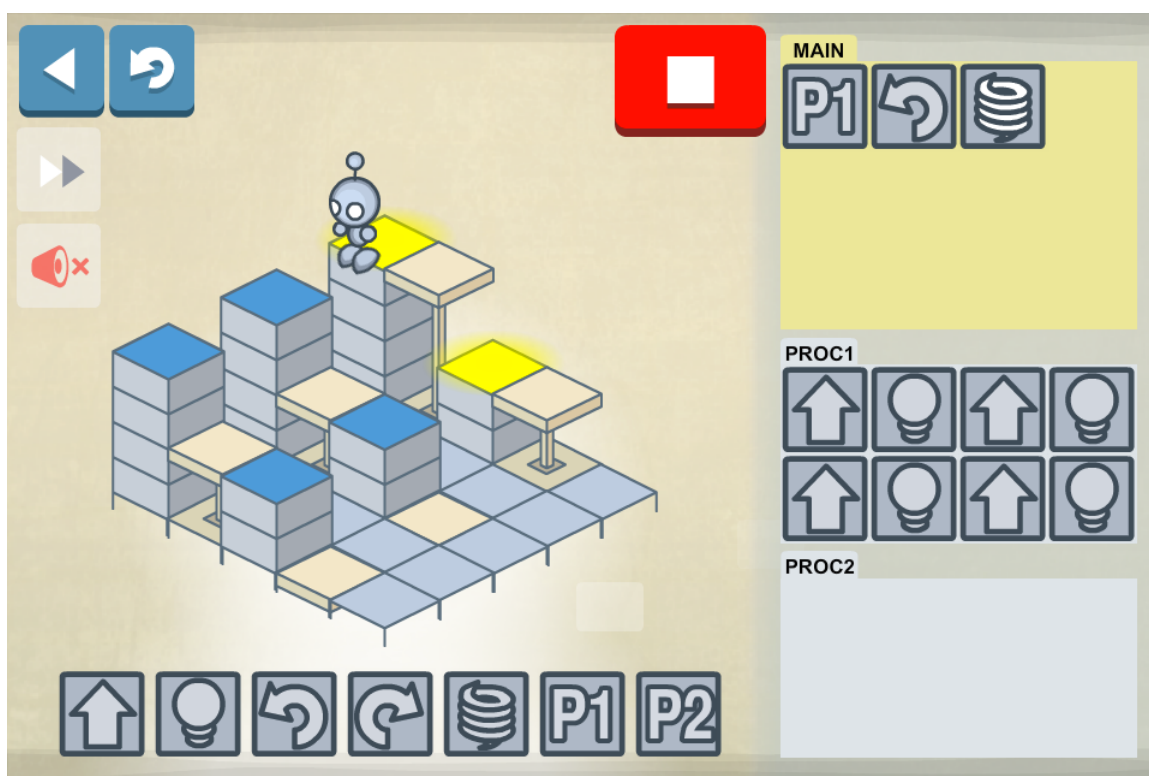


Figura 1: Tabuleiro do jogo *LightBot*

Conteúdo

1	Introdução ao Projeto	3
1.1	<i>LightBot</i>	3
1.2	Objetivos do grupo	3
1.3	Especificações globais - Entrada/Saída de Dados	3
2	Tarefa 1	4
3	Tarefa 2	6
4	Tarefa 3	9
5	Tarefa 4	12
6	Tarefa 5	15
7	Conclusão	18

1 Introdução ao Projeto

1.1 *LightBot*

- O projeto baseia-se na criação de um jogo virtual à semelhança do puzzle *LightBot*, onde se controla um *robot* num tabuleiro de blocos, com recurso a comandos muito simples, com vista a acender todas as lâmpadas disponíveis.
- Consoante o nível, o jogador depara-se com um novo tabuleiro, com nível de dificuldade de resolução crescente, isto é, com lâmpadas cuja localização se torna progressivamente mais desafiante alcançar.
- Este projeto é constituído por duas fases com diferentes tarefas.

1.2 Objetivos do grupo

- Este projeto de LI1 proporcionar-nos-á a criação do nosso primeiro jogo, algo que é sempre marcante para estudantes de Engenharia Informática. Como tal, esperamos conseguir neste projeto ser criativos, tendo em vista não só uma melhor classificação como também a gratificação pelo trabalho realizado quando terminada a criação do jogo.
- Por outro lado, e não menosprezando a avaliação, pensamos que a satisfação pessoal é também relevante aquando da realização de um projeto, pelo que um dos nossos objetivos estipulados é precisamente desfrutarmos da criação deste jogo.

1.3 Especificações globais - Entrada/Saída de Dados

- O **formato de entrada** é essencialmente comum a todas as tarefas (com uma pequena exceção na tarefa 4), representando o tabuleiro onde o *robot* se move, para além da sua posição e orientação inicial e do programa para o controlar. Por exemplo,

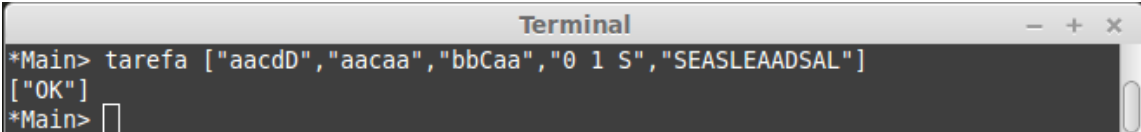
```
aacdD
aaca
bbCaa
0 1 S
SEASLEAADSAL
```

onde as três primeiras linhas correspondem a um tabuleiro representado por 3 linhas, cada uma delas com 5 caracteres alfabéticos, os quais estão associados a diferentes níveis de altura (o carácter **a** ou **A** corresponde ao nível 0, o **b** ou **B** ao 1 e assim sucessivamente). Note-se ainda que a colocação de letra maiúscula ocorre apenas se a posição tem uma "lâmpada"; a linha seguinte dá-nos a informação relativa às coordenadas x e y, respetivamente, da posição inicial do robot, bem como a sua orientação (N, S, E ou O); a linha final são os comandos que definem a movimentação do *robot* no tabuleiro, variando entre A (avançar), S (saltar), E (esquerda), D (direita) e L (luz).

- O **formato de saída**, contrariamente ao de entrada, é específico para cada uma das tarefas.

2 Tarefa 1

1. O objetivo desta primeira tarefa passava pela criação de um programa que validasse se o *input* fornecido cumpria os requisitos impostos pelo formato de entrada. Após verificar se as linhas do tabuleiro possuíam o mesmo tamanho, se a linha das coordenadas da posição inicial apresentava a estrutura devida e se os comandos eram válidos, o respetivo programa deveria imprimir "OK", se o formato do *input* estivesse de acordo com o estabelecido, ou o número da linha em que primeiramente os dados divergiram do formato prescrito.



```
*Main> tarefa ["aacdD","aacao","bbCaa","0 1 S","SEASLEAADSAL"]
["OK"]
*Main> 
```

Figura 2: Resposta impressa quando o *input* é válido

2. Para solucionar o problema, criou-se um programa suportado essencialmente em seis funções auxiliares:
 - i. Duas responsáveis por verificar a validade do tabuleiro, ou seja, se as linhas têm o mesmo tamanho.

```

verificaTab :: Entrada -> Saida
verificaTab [] = "1"
verificaTab l = if (validaTab tab) == 0
                then verificaCoord (dropWhile (all isAlpha)l) (length (head tab)) (length tab) (length tab + 1)
                else show (validaTab tab)
  where tab = takeWhile (all isAlpha) l

{- |'validaTab' verifica se todas as linhas do tabuleiro tem o mesmo tamanho. Se sim devolve o valor 0 (zero).
Se não, devolve o número da linha em que ocorre o erro.

* Exemplo:

>>>validaTab ["aacdD","aacao","bbCaa"] = 0
-}

validaTab :: Tabuleiro -> Int
validaTab [] = 1
validaTab (x:xs) = aux (length x) 1 (x:xs)
  where aux tam n [] = 0
        aux tam n (y:ys) = if length y == tam
                             then aux tam (n+1) ys
                             else n

```

Figura 3: Funções auxiliares que verificam se o tabuleiro é válido

- ii. Outras duas que testam a validade da posição inicial, verificando se as coordenadas são maiores ou iguais que 0 e menores que o tamanho do tabuleiro e se a orientação respeita a prescrição do formato de entrada.

```

verificaCoord :: [String] -> Int -> Int -> Int -> Saida
verificaCoord [] _ _ _ = show e
verificaCoord (h:t) x y e = if validaCoord x y h then verificaCom t (e+1)
                             else show e

{- |'validaCoord' verifica se as coordenadas são maiores ou iguais que 0 e menores que o tamanho do tabuleiro e verifica se a orientação
é válida.

* Exemplo:

>>>validaCoord 5 3 "0 1 S" = True
-}

validaCoord :: Int -> Int -> Coordenadas -> Bool
validaCoord xtam ytam [] = False
validaCoord xtam ytam lcoord = case words lcoord of
  [xcoord,ycoord,o] -> eNumero xcoord &&
                        eNumero ycoord &&
                        read xcoord < xtam &&
                        read ycoord < ytam &&
                        read xcoord >= 0 &&
                        read ycoord >= 0 &&
                        elem o "NSEO"
  where eNumero [] = False
        eNumero [x] = isDigit x
        eNumero (x:xs) | isDigit x = eNumero xs
                        | otherwise = False
  _ -> False

```

Figura 4: Funções auxiliares que verificam a validade da linha das coordenadas da posição inicial

- iii. E ainda duas outras que atuam sobre a(s) última(s) linha(s) do *input*: uma que verifica a validade dos comandos, retornando o número da linha que os inclui se não forem válidos, e outra que, no caso de estes serem válidos, verifica se não existe mais nada em

baixo, devolvendo o número da linha abaixo da linha de comandos se existir ou "OK" se não. É, portanto, nesta fase que o programa devolve "OK" em caso de o *input* ser aprovado, já que a aprovação na verificação dos comandos pressupõe que as prescrições anteriores impostas pelo formato de entrada foram respeitadas.

```
verificaCom :: [String] -> Int -> Saida
verificaCom [] x = show x
verificaCom (h:t) x = if validaComand h then if t == [] then "OK"
                        | else show (x+1)
                        | else show x

{- | 'validaComand' verifica se a linha dos comando é válida.

* Exemplo:

>>> validaComand "SEASLEAADSAL" = True
-}

validaComand :: Comandos -> Bool
validaComand [] = False
validaComand [x] = elem x "ASEDL"
validaComand (x:xs) | elem x "ASEDL" = validaComand xs
                    | otherwise = False
```

Figura 5: Funções auxiliares que testam a linha de comandos

3. Nesta tarefa, o objetivo traçado pelo Projeto foi alcançado na plenitude pelo grupo, obtendo a classificação máxima de pontos na avaliação do *Mooshak*. É de referir também que, associados a esta primeira tarefa, foram realizados e aprovados mais de trinta testes, contendo cada um deles um *input* diferente, com o intuito de pôr a prova o programa criado. Note-se que os testes são não mais do que uma simulação da funcionalidade do programa, logo em qualquer tarefa passam pela criação de um ficheiro com o *input* e outro com o respetivo *output*, estando a aprovação no teste dependente de o *output* manualmente criado coincidir com o ficheiro resultado dado pela *Makefile*.

3 Tarefa 2

1. No que à segunda tarefa diz respeito, o Projeto apresentava como objetivo a implementação de um programa que determinasse a posição do *robot* após a execução do primeiro comando, fornecido por um *input* que se assumia estar de acordo com as especificações estipuladas pelo formato de entrada. Após identificar qual o primeiro comando, o respetivo programa deveria retornar "ERRO" no caso do comando não ser aplicável (por exemplo, comando L sobre uma posição do tabuleiro sem *lâmpada* ou comando S quando o nível da posição final é

superior em mais de uma unidade em relação ao da posição inicial)ou, sendo aplicável, deveria apresentar a posição seguinte do *robot* de forma análoga à linha da posição inicial.

```
posf = proxPosSA coordxi coordyi orient
-- |Coordenada x na posição calculada por 'posf' (coordenada final)
coordxf = read ((words posf) !! 0)
-- |Coordenada y na posição calculada por 'posf' (coordenada final)
coordyf = read ((words posf) !! 1)
-- |Linha tabuleiro a que corresponde a coordenada final y
linhaTabf = tab !! (length tab - (coordyf + 1))
-- |Caracter a que corresponde as coordenadas finais
caracterf = linhaTabf !! coordxf
-- |Caracter transformado em maiusculo para efeitos de comparacao
final = toUpper caracterf
-- |Verifica se a posição que deverá corresponder à seguinte pertence ao tabuleiro e se é possível saltar
verificaS = (validaCoordf (length (head txt)) (length txt - 2) (words posf)) && (validaS inicial final)
-- |Verifica se a posição que deverá corresponder à seguinte pertence ao tabuleiro e se é possível avançar
verificaA = (validaCoordf (length (head txt)) (length txt - 2) (words posf)) && (inicial == final)
proxCoords = if verificaS
then posf
else "ERRO"
proxCoordA = if verificaA
then posf
else "ERRO"
proxCoordL = if isUpper caracteri
then linhaCoord
else "ERRO"
```

Figura 6: Evidência de como o programa obedece às exigências formais da tarefa

2. Este novo programa, respeitando o estipulado, inicialmente encontra o primeiro comando. Em seguida, ocorre a verificação da aplicabilidade desse comando:
 - i. A verificação dos comandos S e A acarretam um grau de dificuldade acrescido, dado ser necessário, em primeiro lugar, verificar se a posição seguinte pertence ao tabuleiro e depois comparar os níveis das duas posições, sendo que: só é possível saltar se o nível da posição inicial for superior ao da final ou inferior em um nível apenas; só é aplicável o A se o nível das duas posições for o mesmo.

```
validaS :: CaracterTab -> CaracterTab -> Bool
validaS x y | x == 'A' = y == 'B'
            | otherwise = ord y < ord x || ord y == ord x + 1
```

Figura 7: Função que verifica se é possível saltar

- ii. Já a verificação do comando L resume-se a analisar se o carácter que representa a posição inicial no tabuleiro é maiúsculo, uma vez

que, como foi anteriormente referido, as posições das lâmpadas são ilustradas no tabuleiro por um carácter maiúsculo.

- iii. Quanto aos comandos O e E, a sua aplicabilidade é universal, logo o programa ignora a verificação e devolve de imediato as coordenadas da posição final.

```
proxPosD :: CoordenadaX -> CoordenadaY -> Orientacao -> ProxPosicao
proxPosD x y o | o == 'N' = show x ++ " " ++ show y ++ " E"
                | o == 'S' = show x ++ " " ++ show y ++ " O"
                | o == 'E' = show x ++ " " ++ show y ++ " S"
                | o == 'O' = show x ++ " " ++ show y ++ " N"

{- |'proxPosE' calcula a próxima posição para o comando esquerda.

*Exemplo:

>>> proxPosE 0 1 S = "0 1 E"
-}

proxPosE :: CoordenadaX -> CoordenadaY -> Orientacao -> ProxPosicao
proxPosE x y o | o == 'N' = show x ++ " " ++ show y ++ " O"
                | o == 'S' = show x ++ " " ++ show y ++ " E"
                | o == 'E' = show x ++ " " ++ show y ++ " N"
                | o == 'O' = show x ++ " " ++ show y ++ " S"
```

Figura 8: Função que calcula a próxima posição para O e E

3. Para os diferentes comandos segue-se:

- i. Após a verificação, para S e A, se a posição final pertencer ao tabuleiro, a função verifica se é possível saltar e avançar, respetivamente. Caso seja possível, o programa devolve a posição final já calculada, senão retorna "ERRO".

```
proxPosSA :: CoordenadaX -> CoordenadaY -> Orientacao -> ProxPosicao
proxPosSA x y o | o == 'N' = show x ++ " " ++ show (y+1) ++ " N"
                | o == 'S' = show x ++ " " ++ show (y-1) ++ " S"
                | o == 'E' = show (x+1) ++ " " ++ show y ++ " E"
                | o == 'O' = show (x-1) ++ " " ++ show y ++ " O"
```

Figura 9: Função que calcula a próxima posição para S e A

- ii. Para L, caso se confirme a validade deste comando, a próxima posição será coincidente com a posição inicial.

```
proxCoordL = if isUpper caracteri
              then linhaCoord
              else "ERRO"
```

Figura 10: Função que devolve a linha das coordenadas da posição inicial se o carácter for maiúsculo

- iii. As coordenadas da posição final, se o comando inicial for E ou O, divergirão das da inicial apenas na parcela referente à orientação.
- 4. Nesta segunda tarefa, foi também possível ao grupo obter pontuação máxima na avaliação levada a cabo pelo *Mooshak*, o que demonstra que foram cumpridos os objetivos definidos pelo enunciado do Projeto para esta tarefa, algo passível de se concluir também pelo facto de os testes realizados para esta segunda etapa do trabalho terem recebido na totalidade resposta positiva.

4 Tarefa 3

1. A tarefa 3, última desta primeira fase do Projeto, requeria um programa capaz de fazer o *robot* executar a sequência de comandos, fornecida pelo *input*, até acender todas as lâmpadas disponíveis. Novamente, assumia-se que o *input* estaria de acordo com as prescrições do formato de entrada. Considerando que os comandos eram executados em sequência e que aqueles que não fossem aplicáveis deveriam deixar o estado do *robot* inalterado, ao programa criado era exigido que imprimisse uma linha contendo as coordenadas "x" e "y" da posição do *robot* no momento (separadas por um único espaço), sempre que um comando L fosse executado com sucesso. Se todas as *lâmpadas* do tabuleiro fossem acesas, o programa deveria terminar, independentemente da existência de comandos não executados, retornando a mensagem "FIM «Num»", sendo «Num» o número de comandos válidos executados. Caso contrário, se a sequência de comandos terminasse sem que todas as *lâmpadas* se encontrassem acesas, deveria imprimir "INCOMPLETO".

```

proxPos :: Tabuleiro -> LuzesTabuleiro -> Dimensao -> Comandos -> Acumulador -> Resultado
proxPos tab l (dimx,dimx) [] (coord,coordL,coordLT,y) = if pertence l coordL
then coordLT ++ ["FIM " ++ show y]
else coordLT ++ ["INCOMPLETO"]
proxPos tab l (dimx,dimx) (h:t) (coord,coordL,coordLT,y) = if pertence l coordL
then coordLT ++ ["FIM " ++ show y]
else proxPos tab l (dimx,dimx) t (proxPos tab (dimx,dimx) h (coord,coordL,coordLT,y)

```

Figura 11: Programa retorna FIM «Num» ou INCOMPLETO como é suposto

2. O funcionamento do programa criado pelo grupo para dar resposta ao problema colocado por esta terceira tarefa baseia-se em:
 - i. Calcular sucessivamente a próxima posição dependendo do comando a executar, pelo que parcelas do programa da tarefa anterior são

aplicadas também nesta. Dado que, para efeito dos dados finais devolvidos pelo programa, é necessário contabilizar os comandos executados, foi adicionado um acumulador à função que calcula a próxima posição, que aumenta uma unidade sempre que um comando seja aplicável.

```
validaS :: CaracterTab -> CaracterTab -> Bool
validaS x y | x == 'A' = y == 'B'
            | otherwise = ord y < ord x || ord y == ord x + 1

{- |'proxPosD' calcula a próxima posição para o comando direita.
*Exemplo:
>>> proxPosD 0 1 S = "0 1 0"
-}
proxPosD :: CoordenadaX -> CoordenadaY -> Orientacao -> Coordenadas
proxPosD x y o | o == 'N' = show x:show y:"E":[]
               | o == 'S' = show x:show y:"O":[]
               | o == 'E' = show x:show y:"S":[]
               | o == 'O' = show x:show y:"N":[]

{- |'proxPosE' calcula a próxima posição para o comando esquerda.
*Exemplo:
>>> proxPosE 0 1 S = "0 1 E"
-}
proxPosE :: CoordenadaX -> CoordenadaY -> Orientacao -> Coordenadas
proxPosE x y o | o == 'N' = show x:show y:"O":[]
               | o == 'S' = show x:show y:"E":[]
               | o == 'E' = show x:show y:"N":[]
               | o == 'O' = show x:show y:"S":[]
```

Figura 12: Algumas funções comuns à tarefa anterior

```
proxPos :: Tabuleiro -> Dimensao -> Comando -> Acumulador -> Acumulador
proxPos tab (dimx,dimy) x (coord,coordL,coordLT,y) | x == 'S' && verificaS = (proxPosAS,coordL,coordLT,y+1)
                                                    | x == 'A' && verificaA = (proxPosAS,coordL,coordLT,y+1)
                                                    | x == 'L' && isUpper caracteri = (coord,verificaRep (coordAcende) coordL,coordLT ++ [coordAcende],coordLT,y+1)
                                                    | x == 'D' = (proxPosD coordxi coordyi orient,coordL,coordLT,y+1)
                                                    | x == 'E' = (proxPosE coordxi coordyi orient,coordL,coordLT,y+1)
                                                    | otherwise = (coord,coordL,coordLT,y)
where verificaS = validaCoord dimx dimy proxPosAS && validas inicial final
      verificaA = validaCoord dimx dimy proxPosAS && (inicial == final)
      proxPosAS = proxPosSA coordxi coordyi orient
      coordxi = read (coord !! 0)
      coordyi = read (coord !! 1)
      orient = head (coord !! 2)
      caracteri = (tab !! (length tab - (coordyi + 1))) !! coordxi
      inicial = toUpper caracteri
      coordxf = read ((proxPosAS) !! 0)
      coordyf = read ((proxPosAS) !! 1)
      caracterf = (tab !! (length tab - (coordyf + 1))) !! coordxf
      final = toUpper caracterf
      coordAcende = show coordxi ++ " " ++ show coordyi
```

Figura 13: Função com acumulador mencionada em i.

- ii. Todavia, antes de calcular a posição seguinte há uma verificação se as *lâmpadas* estão todas acesas. Caso estejam, o programa termina e imprime o resultado final, dentro das características do output para a terceira tarefa.

- iii. É também de mencionar o modo como o programa atua sobre as listas das coordenadas das *lâmpadas*. As três primeiras funções apresentadas na figura 14 calculam a lista das onde existe *lâmpada*, ou seja, carácter maiúsculo no tabuleiro. À medida que o *robot* acende as *lâmpadas*, as coordenadas das que já estão acesas são guardadas numa lista, porém sempre que o *robot* executa o comando L nas coordenadas de uma *lâmpada* já acesa, ela é apagada e essas coordenadas são removidas da lista. Se o comando L voltar a ser executado nessa coordenadas, elas são reintroduzidas na lista. Para salvaguardar isto, foi necessário criar uma função que verificasse se as coordenadas onde foi executado com sucesso o comando L já existiam na lista só com as coordenadas de *lâmpadas* acesas. É esta lista, apenas de coordenadas de *lâmpadas* já acesas, que é comparada à que inclui as de todas as *lâmpadas* do tabuleiro, para verificar se foram acesas todas elas.

```

luzesTab :: Tabuleiro -> Luzes
luzesTab l = coordLuzes l (length l - 1)
{- |
*Exemplo:

>>> coordLuzes ["aacdD","aacaa","bbCaa"] 2
1. linhas "aacdD" 0 2 ++ coordLuzes ["aacaa","bbCaa"] (2-1)
-}
coordLuzes :: Tabuleiro -> Int -> Luzes
coordLuzes [] _ = []
coordLuzes (h:t) y = linhas h 0 y ++ coordLuzes t (y-1)
{- |
*Exemplo:

>>> linhas "aacdD" 0 2 = ["4 2"]
-}
linhas :: String -> Int -> Int -> Luzes
linhas [] _ _ = []
linhas (h:t) x y | isUpper h = [show x ++ " " ++ show y] ++ linhas t (x+1) y
                  | otherwise = linhas t (x+1) y

{- | 'verificaRep' recebe a coordenada em que foi executado o comando L com sucesso
na lista das coordenadas que só tem as luzes que estão acesas.

*Exemplo:

>>> verificaRep "4 2" ["2 0"] = ["2 0","4 2"]

>>> verificaRep "2 0" ["2 0"] = []
-}
verificaRep :: String -> Luzes -> Luzes
verificaRep x [] = [x]
verificaRep x (h:t) | x == h = t
                    | otherwise = h : verificaRep x t

```

Figura 14: Funções auxiliares que atuam sobre as coordenadas das *lâmpadas*

3. Essencialmente por questões de eficiência do programa criado, nesta fase a classificação que o *Mooshak* atribuiu ao grupo esteve dois pontos abaixo daquela que era a classificação máxima. Não obstante este facto, os múltiplos ficheiros de teste criados para a tarefa 3 foram aprovados quando corrida a *Makefile*.

5 Tarefa 4

1. A tarefa 4 marca o início da segunda fase do Projeto. Nela pretendia-se a realização de um programa que sintetizasse um programa para o *robot* de modo que este conseguisse, a partir da sua posição inicial, acender todas as *lâmpadas* do tabuleiro. Desta forma, era notório que, dependendo do tabuleiro, da posição inicial do *robot* e das posições das *lâmpadas*, a solução para o problema poderia ser simples (quando a diferença de altura, para quaisquer duas posições adjacentes do tabuleiro, fosse de um nível no máximo), complicada (quando nem todos os caminhos entre duas posições do tabuleiro fossem possíveis) ou até mesmo impossível de encontrar (quando existisse pelo menos uma *lâmpada* inalcançável). Tal como nas duas tarefas anteriores, assumia-se que o *input* obedeceria ao formato de entrada estabelecido, com a particularidade de ser omitida a linha referente aos comandos. O *output* esperado corresponderia à linha referente ao programa sintetizado.
2. Para esta tarefa, o programa criado pelo grupo cobre todos os casos simples e boa parte dos complicados. À semelhança da tarefa anterior, é necessário encontrar as coordenadas das posições das *lâmpadas*, pelo que as funções da tarefa 3 responsáveis por esta ação foram reaproveitadas nesta tarefa. No entanto, há diferenças na abordagem a cada uma das situações (casos simples ou complexos):
 - i. Uma vez que não existe a necessidade de verificar se é possível ou não ao *robot* saltar, os casos mais simples foram solucionados com recurso a funções que essencialmente atuam sobre a orientação do *robot*, de modo a orientá-lo no caminho até cada *lâmpada*, e seleccionam, tanto para os movimentos verticais como horizontais, o comando A (avançar) ou S(saltar), dependendo das posições inicial e final.

```

rodaN :: Char -> String
rodaN o | o == 'N' = ""
        | o == 'E' = "E"
        | o == 'S' = "EE"
        | o == 'O' = "D"

rodaS :: Char -> String
rodaS o | o == 'S' = ""
        | o == 'O' = "E"
        | o == 'N' = "EE"
        | o == 'E' = "D"

moveASE :: [String] -> (Int,Int) -> (Int,Int) -> String
moveASE tab (xi,yi) (xf,yf) | xi == xf = []
                             | otherwise = (verificaAS tab (xi,yi) (xi+1,yi)) ++ (moveASE tab (xi+1,yi) (xf,yf))

moveASO :: [String] -> (Int,Int) -> (Int,Int) -> String
moveASO tab (xi,yi) (xf,yf) | xi == xf = []
                             | otherwise = (verificaAS tab (xi,yi) (xi-1,yi)) ++ (moveASO tab (xi-1,yi) (xf,yf))

```

Figura 15: Algumas das funções que intervêm na orientação do *robot* e na seleção entre os comandos A e S

- ii. Já para os casos mais complexos, foi necessário criar funções auxiliares que, inicialmente, calculam todas as combinações possíveis entre as *lâmpadas*, pois a morfologia do tabuleiro pode não permitir ao *robot* acendê-las pela ordem em que surgem na lista das suas coordenadas, e depois calculam a lista de comandos capaz de acender todas as *lâmpadas* para cada uma das combinações anteriores.

```

fun :: [Coordenadas] -> [[Coordenadas]] -> [[Coordenadas]]
fun [] l = l
fun (h:t) l = fun t (auxP h l)
{- |
*Exemplo

>>>auxP (4,3) []
1. [(4,3)]

>>>auxP (1,2) [(4,3)]
1. aux (1,2) [(4,3)] 0
-}
auxP :: Coordenadas -> [[Coordenadas]] -> [[Coordenadas]]
auxP x [] = [[]]
auxP x [l] = aux x l 0
auxP x (h:t) = (aux x h 0) ++ auxP x t
{- |
*Exemplo

>>>aux (1,2) [(4,3)] 0
1. [aux2 (1,2) [(4,3)] 0] ++ aux (1,2) [(4,3)] (0+1)
-}
aux :: Coordenadas -> [Coordenadas] -> Int -> [[Coordenadas]]
aux x l n | n == length l = [aux2 x l n]
          | otherwise = [aux2 x l n] ++ (aux x l (n+1))
{- |
*Exemplo

>>>aux2 (1,2) [(4,3)] 0 = [(1,2),(4,3)]

>>>aux2 (1,2) [(4,3)] 1
1. (4,3): (aux2 (1,2) [(4,3)] (1-1))
-}
aux2 :: Coordenadas -> [Coordenadas] -> Int -> [Coordenadas]
aux2 x [] = [x]
aux2 x (h:t) n | n == 0 = (x:h:t)
               | otherwise = h : (aux2 x t (n-1))

```

Figura 16: Funções que calculam todas as combinações possíveis entre as *lâmpadas*

Para o cálculo da lista de comandos, as funções do programa atuam da seguinte forma: testam se, deslocando-se o *robot* primeiro na horizontal e em seguida na vertical (ou vice-versa), este consegue chegar à primeira *lâmpada*. Se sim, avança para a segunda pelo mesmo trajeto (horizontal-vertical ou vice-versa), caso contrário tenta chegar lá pelo trajeto não testado. Esta abordagem é feita para todas as *lâmpadas*, restando porém mencionar que se, na tentativa de alcançar uma delas, nos depararmos com uma situação em que não é possível nem pelo caminho horizontal-vertical nem pelo vertical-horizontal, o programa conclui que a combinação de *lâmpadas* em teste não é possível e passa para o teste da seguinte combinação.

```

move :: Tabuleiro -> Coordenadas -> Char -> [Coordenadas] -> String
move _ [] = []
move tab (x,y) o (h:t) = if elem '1' l1
                        then if elem '1' l2
                            then "1"
                            else l2 ++ move tab h o2 t
                        else l1 ++ move tab h o1 t
where l1 = orientX tab (x,y) o h ++ orientY tab (fst h,y) oY h ++ "L"
      l2 = orientY tab (x,y) o h ++ orientX tab (x,snd h) oX h ++ "L"
      o1 | y < (snd h) = 'N'
          | y > (snd h) = 'S'
          | x < (fst h) = 'E'
          | x > (fst h) = 'O'
          | x == (fst h) = o
      o2 | x < (fst h) = 'E'
          | x > (fst h) = 'O'
          | y < (snd h) = 'N'
          | y > (snd h) = 'S'
          | y == (snd h) = o
      oY | x < (fst h) = 'E'
          | x > (fst h) = 'O'
          | x == (fst h) = o
      oX | y < (snd h) = 'N'
          | y > (snd h) = 'S'
          | y == (snd h) = o

```

Figura 17: Função que possibilita a abordagem às *lâmpadas* pelos diferentes trajetos

É também de mencionar que, por uma questão de eficiência, optou-se por, para os casos em que o tabuleiro apresente menos de sete *lâmpadas*, colocar o programa a imprimir como resultado a menor sequência de comandos de entre aquelas obtidas através das diferentes combinações, possíveis, de *lâmpadas*. Para os tabuleiros com mais de sete *lâmpadas*, o programa retorna a lista de comandos derivada da primeira combinação possível.

```

tarefa :: [String] -> [String]
tarefa txt = [msg]
  where tab = takeWhile (all isAlpha) txt
        linhaCoord = txt !! (length txt - 1)
        coordx = read ((words linhaCoord) !! 0)
        coordy = read ((words linhaCoord) !! 1)
        coord = (coordx, coordy)
        orient = head ((words linhaCoord) !! 2)
        msg | length (luzesTab tab) < 7 = menor (movePm tab coord orient (fun (luzesTab tab) []))
            | otherwise = movePM tab coord orient (fun (luzesTab tab) [])

```

Figura 18: Evidência do que foi mencionado anteriormente

3. Nesta quarta tarefa, dado que o programa criado não cobre todos os casos mais complexos, o grupo obteve, e há semelhança da classificação atribuída pelo *Mooshak* à tarefa 3, dezoito em vinte pontos totais. Pensa-se que os casos que ficam por cobrir pelo programa referem-se àqueles em que o *robot*, para alcançar uma das *lâmpadas*, necessita de se mover ora na horizontal ora na vertical, alternando sucessivamente.

6 Tarefa 5

1. A tarefa 5 é a segunda tarefa da segunda fase, sendo a última do Projeto. Nela pretendia-se visualizar o jogo criado com recurso ao formato *X3dom*, o qual permite a visualização de cenas tridimensionais em *browsers web*. Como foi dito introdutoriamente, o formato de entrada é comum a todas as tarefas, logo o *input* assemelhar-se-ia aos das tarefas anteriores. Já como *output*, o programa deveria imprimir o código **xhtml** de uma página web que permitisse visualizar o jogo. Dado o carácter subjetivo desta tarefa, os *outputs* poderiam divergir tanto quanto a criatividade dos alunos, pelo que a avaliação seria sempre essencialmente qualitativa.
2. Na quinta tarefa, começou-se por criar o *robot* numa página *html*, explorando o formato *X3dom*, enquanto em simultâneo se geravam as

primeiras funções do programa desta tarefa, destinadas à criação de tabuleiros.

Robot "The Snowman"

Trying



Figura 19: *Robot*

Em seguida passou-se às adaptações *html* no programa, necessárias para ser possível, a partir do executável da tarefa e do terminal, obter a visualização do jogo numa página *html*:

- i. Parte destas adaptações passaram por copiar, do enunciado do Projeto para esta tarefa 5, um prefixo e um sufixo comuns a ficheiros *html* deste tipo.

```
prefixo = [ "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\"",
  "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">",
  "<html xmlns=\"http://www.w3.org/1999/xhtml\">",
  "<head>",
  "<meta http-equiv=\"X-UA-Compatible\" content=\"chrome=1\" />",
  "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\" />",
  "<title>Hello World</title>",
  "<script type=\"text/javascript\" src=\"http://www.x3dom.org/release/x3dom.js\"></script>",
  "<link rel=\"stylesheet\" type=\"text/css\" href=\"http://www.x3dom.org/release/x3dom.css\">",
  "</head>",
  "<body>",
  "<h1>Mundo de Cubos</h1>",
  "<p class=\"case\">",
  "<X3D xmlns=\"http://www.web3d.org/specifications/x3d-namespace\" id=\"boxes\"",
  "showStat=\"false\" showLog=\"false\" x=\"0px\" y=\"0px\" width=\"400px\" height=\"400px\">",
  "<Scene>",
  "<Shape DEF=\"tile\">",
  "<Appearance>",
  "<Material diffuseColor='0 1.0 1.0' />",
  "</Appearance>",
  "<Box size='3.45 3.45 1.45' />",
  "</Shape>",
  "<Shape DEF=\"tile\">",
  "<Appearance>",
  "<Material diffuseColor='0 0 1.0' />",
  "</Appearance>",
  "<Box size='3.45 3.45 1.45' />",
  "</Shape>"
]
```

Figura 20: Prefixo mencionado em i.

- ii. Em seguida, para além da criação de funções que permitem colocar o *robot* no tabuleiro fornecido pelo *input*, fizeram-se adaptações *html* às funções que criam os tabuleiros e inseriu-se o código *html* do *robot* no programa sob a forma de sufixo.

```
mostra :: Coordenada -> Coordenada -> Char -> [String]
mostra x y h = if nivel h == 0
               then [mostraP x y h]
               else [mostraP x y h] ++ mostra x y (chr (ord (toLower h) - 1))

mostraP :: Coordenada -> Coordenada -> Char -> String
mostraP x y h = if isUpper h then "<Transform translation=" ++ (show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5) ++ " " ++ show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5))
                else "<Transform translation=" ++ (show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5) ++ " " ++ show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5))

nivel :: Char -> Float
nivel c = (fromIntegral (ord (toLower c) - ord 'a'))*1.5

robot :: Tabuleiro -> Coordenadas -> Orientacao -> [String]
robot tab (x,y) c = ["<Transform translation=" ++ (show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5) ++ " " ++ show ((fromIntegral x)*3.5) ++ " " ++ show ((fromIntegral y)*3.5) ++ " " ++ show ((fromIntegral h)*3.5))
                    , "<Transform translation='0 0 2.2' rotation='1 0 0 1.57>"]

altura :: Tabuleiro -> Coordenadas -> Float
altura tab (x,y) = 1.5 + nivel caracter
  where caracter = linha !! x
        linha = tab !! (length tab - (y+1))

orientacao :: Orientacao -> Float
orientacao c | c == 'S' = 0
              | c == 'O' = 4.71
              | c == 'N' = 3.14
              | c == 'E' = 1.57
```

Figura 21: Adaptações *html*

- iii. Por fim, após elaborar num ficheiro *html* a animação do *robot* num tabuleiro específico, procurou-se criar funções *Haskell* que permitam ao programa devolver numa página *html* uma visualização fiel do *robot* a executar os comandos que lhe foram aplicados, isto depois de efetuadas as devidas adaptações.

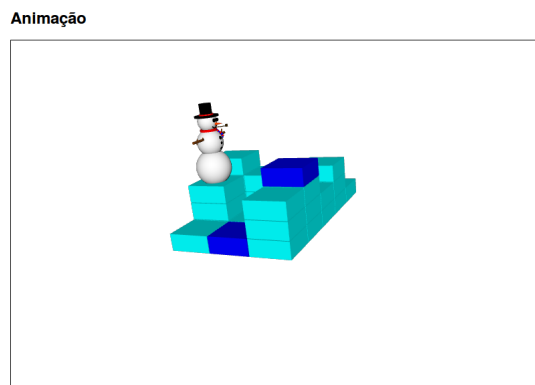


Figura 22: Exemplo de uma animação

```

prefixoanimacao = ["<transform DEF=\\"robot\\">"]
animacao :: Tabuleiro -> Coordenadas -> Orientacao -> Comandos -> [String]
animacao tab (x,y) o l =
  [
    "<timeSensor DEF=\\"time\\" cycleInterval=\\" ++ show (fromIntegral (length l)/4) ++ "\\" loop=\\"true\\"> </timeSensor>",
    "<PositionInterpolator DEF=\\"movePOS\\" key=\\"0\\" ++ key (length l + extra l,1/(fromIntegral (length l + extra l))) 1 ++ "\\" keyValue=\\"0 0 0\\">",
    ++ fst (proxPos tab (x,y) o l) ++ "\\"> </PositionInterpolator>",
    "<OrientationInterpolator DEF=\\"moveROT\\" key=\\"0\\" ++ key (length l,1/(fromIntegral (length l))) 1 ++ "\\" keyValue=\\"0 0 0\\">"
    ++ snd (proxPos tab (x,y) o l) ++ "\\"> </OrientationInterpolator>"
  ]

key :: (Int,Float) -> Float -> String
key (x,y) n |x == 1 = " 1.0"
              otherwise = " " ++ show (n*y) ++ key (x-1,y) (n+1)

extra :: String -> Int
extra l = length (filter (\c -> c == 'L')l) + length (filter (\c -> c == 'S')l)

proxPos :: Tabuleiro -> Coordenadas -> Orientacao -> Comandos -> (String,String)
proxPos tab (x,y) o l | o == 'S' = ((proxPosSul tab (x,y) o l ["", "0", "0", "0"]), (roda l ["", "0", "0", "0", "0", "0"]))
                        | o == 'E' = ((proxPosEste tab (x,y) o l ["", "0", "0", "0"]), (roda l ["", "0", "0", "0", "0", "0"]))
                        | o == 'N' = ((proxPosNorte tab (x,y) o l ["", "0", "0", "0"]), (roda l ["", "0", "0", "0", "0", "0"]))
                        | o == 'O' = ((proxPosOeste tab (x,y) o l ["", "0", "0", "0"]), (roda l ["", "0", "0", "0", "0", "0"]))

```

Figura 23: Adaptações *html* para possibilitarem a animação do *robot*

3. Como foi referido, a avaliação desta tarefa é essencialmente qualitativa, pelo que não é possível apontar a maior ou menor correção do programa da tarefa. Contudo, os testes do *Mooshak* fazem uma verificação estrutural do código gerado pelo programa, com vista a perceber se este devolve ou não uma página *html*. Nesse teste, o programa criado pelo grupo para esta quinta tarefa apresenta um "Presentation Error".

7 Conclusão

- Em jeito de conclusão, diga-se que desfrutamos da criação deste jogo. Por um lado, pelo toque pessoal que lhe pudemos impor. Por outro, pela experiência que ele nos proporcionou, nomeadamente no que diz respeito ao contacto com a plataforma \LaTeX e o formato *html*, sem nunca esquecer o aprofundar de conhecimentos nas aplicações *Haskell*.

