ISEL INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Programação
Repetição do teste final

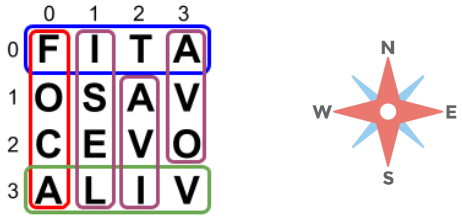
Época de Recurso - 27 janeiro 2024

ISEL - DEETC
LEIC - PG
T2 23/24 - Inv

Entregar as respostas de cada grupo em folhas separadas, identificadas com o número e o nome do aluno.
Em cada resposta, o texto deve ser legível e compreensível e o código deve estar indentado coerentemente.
Em todas as respostas será valorizada a legibilidade e a simplicidade da solução.

Grupo I [12]

Pretende-se um programa para jogar ao "**Sopa de letras**", em que o jogador tenta descobrir as palavras orientadas em qualquer direção inscritas numa matriz de letras.
Considera-se apenas as quatro orientações principais (N-Norte, S-Sul, E-Este e W-Oeste).




```
FITA
OSAV
CEVO
ALIV
Por descobrir: [ISEL, VILA, FITA, AVO, FOCA, IVA]
Palavra RCO? IVA 32N
Por descobrir: [ISEL, VILA, FITA, AVO, FOCA]
Palavra RCO? ISEL 50X
Jogada inválida
Palavra RCO? ISEL 00S
Não existe ISEL no local indicado.
Por descobrir: [ISEL, VILA, FITA, AVO, FOCA]
Palavra RCO? ISEL 01S
Por descobrir: [VILA, FITA, AVO, FOCA]
...
Por descobrir: [FOCA]
Palavra RCO? foca 00s
Parabéns.
```

À direita é apresentada uma execução do programa.
O programa apresenta a matriz de letras e a lista de palavras por descobrir e o utilizador tenta descobrir cada palavra indicando a palavra, a linha e a coluna da primeira letra e a orientação da palavra (*RCO - Row, Column and Orientation*).
Por exemplo: ***ISEL 01S*** indica que a palavra ***ISEL*** começa na linha ***0***, coluna ***1*** com orientação ***S***ul.

Para realizar a totalidade do programa, implemente cada uma das funções seguintes, incluindo a `main`.

Nas restantes questões considere o tipo enumerado `Ori`, para representar as quatro orientações, e os tipos agregados `RCO` e `Guess`, para representar cada tentativa de descoberta de uma palavra.

```
enum class Ori(val dRow: Int, val dCol: Int) {
   N(-1,0), S(+1,0), E(0,+1), W(0,-1)
}
data class RCO(val row: Int, val col: Int, val ori: Ori)
data class Guess(val word: String, val rco: RCO)
```

Considere também que, para definir um jogo existem os valores globais *mat* e *words*, que representam a matriz de letras e as palavras a descobrir.

```
val mat = listOf("FITA", "OSAV", "CEVO", "ALIV")
val words = listOf("ISEL", "VILA", "FITA", "AVO", "FOCA", "IVA")
```

Assumindo que cumprem sempre a seguinte condição:
`mat.size in 3..10 && mat.all{it.length==mat.size} && words.all{it.length in 3..mat.size}`

1. [2] Implemente a função `Char.toOri(): Ori?` que transforma um valor do tipo `Char` (uma das letras *N*, *S*, *E* ou *W*) numa orientação. A função retorna `null` se o valor do tipo `Char` não corresponde a uma orientação.
   Por exemplo, a expressão `'N'.toOri()` tem o valor `Ori.N` , mas a expressão `'A'.toOri()` tem o valor `null`.

2. [2] Implemente a função `Char.toRowOrCol(): Int?`, que retorna o valor inteiro correspondente a um dígito dentro do intervalo `0..<mat.size`. A função retorna `null` se o `Char` não for um dígito ou estiver fora do intervalo.
   Note que, `mat.size` pode ter qualquer valor entre `3` e `10`.

3. [2] Considerando a função `readGuess` que retorna uma tentativa de descoberta introduzida pelo jogador, ou `null` caso a tentativa não seja válida. Implemente a função *`toRCO`* chamada por `readGuess`.
   A função `toRCO` deve chamar as funções `toOri` e `toRowOrCol` e deve retornar um valor do tipo `RCO` ou `null` se a `String` em causa não corresponde a um `RCO` válido. Um `RCO` válido tem exatamente 3 símbolos, o 1º e 2º símbolos são os números da linha e da coluna e o 3º símbolo corresponde a uma orientação.

```
fun readGuess(): Guess? {
   print("Palavra RCO? ")
   val ln = readln().uppercase().split(' ')
   if (ln.size!=2 || ln[0].length !in 3..mat.size)
       return null
   val rco = ln[1].toRCO() ?: return null
   return Guess(ln[0], rco)
}
```

4. [2] Implemente a função `readValidGuess(toFind: List<String>): Guess`. Esta função deve chamar a função `readGuess` até a tentativa de descoberta lida ser válida e a palavra da tentativa pertencer à lista de palavras `toFind`, recebida como parâmetro. Apresenta o texto `Jogada inválida` quando for necessário ler novamente.

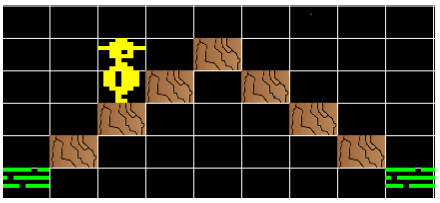
5. [2] Implemente a função **`Guess.isCorrect(): Boolean`** que apenas retorna **`true`** se a palavra da tentativa de descoberta corresponde a uma palavra na matriz ***`mat`***, com início na posição indicada e com a orientação indicada.
Na implementação desta função, admita a existênca da função **`getWordOf(rco: RCO): String`** que retorna a palavra na matriz ***`mat`*** que começa na posição e orientação indicadas em **`rco`** e que termina quando acaba a matriz.
Por exemplo, a chamada ***`getWordOf`*****`(RCO(1,0,Ori.S))`** retorna a string **`"OCA"`**.

6. [2] Usando as funções anteriores, implemente a função **`main`** do programa. O exemplo apresentado inicialmente demonstra uma possível utilização do programa e qual o *output* rigorosamente pretendido.
O programa deve manter uma lista com as palavras por descobrir iniciada com ***`words`***, deve apresentar inicialmente a matriz de letras definida em ***`mat`*** e depois repetir as seguinte operações enquanto existirem palavras por descobrir:
- Apresentar as palavras ainda por descobrir;
- Ler mais uma tentativa válida de descoberta, introduzida pelo jogador (**`readValidGuess`**);
- Se a tentativa lida estiver correta (**`isCorrect`**), retira da lista a palavra indicada na tentativa;
- Caso contrário, indica que a palavra não existe no local indicado.

No final, apresenta o texto **`Parabéns.`**

Grupo II [8]

No contexto do jogo *Chuckie Egg*, realizado este semestre, considere que foi adicionada uma montanha ao nível, sendo possível ao homem, escalá-la (subindo ou descendo) em qualquer direção, dependendo da sua orientação (esquerda ou direita). A figura seguinte ilustra uma possível situação de jogo com o homem subindo a montanha:



Considere ainda as seguintes declarações:

```
data class Game(
   val man: Man,
   val floor: List<Cell>,
   val hill: List<Cell>,
   val stairs: List<Cell>
)

data class Cell(
   val row: Int,
   val col: Int
)
```

```
data class Man(
   val pos: Point,
   val faced: Direction = Direction.RIGHT,
   val speed: Speed = Speed(dx=CELL_WIDTH, dy=-CELL_HEIGHT)
)

enum class Direction(val dRow: Int, val dCol: Int) {
   LEFT(0,-1), RIGHT(0,+1), UP(-1,0), DOWN(+1,0)
}

data class Point(val x: Int, val y: Int)
data class Speed(val dx: Int, val dy: Int)
```

O tipo **`Game`** reúne toda a informação do jogo. O tipo **`Man`** representa o homem, contendo a sua posição atual (**`pos`**), direção (**`faced`**) e velocidade (**`speed`**). As propriedades **`floor`** e **`hill`** contêm respectivamente a lista de células do chão e de uma (e apenas uma) montanha, sendo a propriedade **stairs** irrelevante para este exercício.
O tipo **`Point`** representa uma posição (**`x`**, **`y`**) no Canvas, o tipo **`Direction`** uma direção e o tipo **Speed** um deslocamento ao longo das colunas e das linhas (**`dx`**, **`dy`**) da grelha do jogo.

1. [2] Implemente a função `onTheHill(game:Game):Boolean`, que verifica se o homem se encontra na montanha. Lembre-se que a função `Point.toCell():Cell` retorna a célula correspondente à posição, e que a função `Cell.plus(dir: Direction):Cell` permite adicionar uma linha/coluna à célula.

2. [3] Implemente a função `topOfHill(game:Game):Cell`, que retorna a célula correspondente ao cume da montanha.

3. [3] Implemente a função `stepClimbHill(game:Game):Man`, extensão de **`Man`**, que retorna o homem na nova posição, caso este se encontre na montanha, deslocando-se uma posição (subindo ou descendo) por cada passo, à velocidade definida em **`speed`**, de acordo com a sua direção (**`faced`**). Se o homem estiver no cume, inverte a subida ou descida. Esta função deve chamar as funções implementadas nos pontos anteriores (`onTheHill` e `topOfHill`). Lembre-se que a função `Point.plus(speed: Speed):Cell` permite adicionar uma velocidade à posição.