Labirintos

Trabalho de Avaliação de Programação II

2016/17

Conteúdo

Resumo

Um *labirinto* é passa-tempo milenar que apresenta desafios cognitivos, matemáticos e informáticos.

1 Enunciado

Implemente um programa `java` para ver, criar, resolver e estudar labirintos.

A sua resolução deve respeitar os contratos definidos na subsecção 3.1. Além disso, deve ter um *interface* gráfico adequado às seguintes tarefas:

1. Mostrar um labirinto definido num ficheiro de texto, de acordo com o formato especificado na sub-secção 3.2.
2. Resolver o labirinto mostrado. O resultado deve mostrar o percurso desde o **início** até ao **fim**.
3. Modo de Jogo. O utilizador pode movimentar o **peão**.
4. Modo Gerador. O programa gera um labirinto de acordo com um conjunto de parâmetros (por exemplo, número de linha e colunas).

2 Labirinto

Um **labirinto** é definido por uma grelha, `maze`, com `numCols` colunas e `numRows` linhas. Em cada posição da grelha está um dos seguintes valores:

- `EMPTY`: a posição está livre (pode ser ocupada pelo peão).

- `WALL`: a posição está ocupada (não pode ser ocupada pelo peão).
- `START`: uma posição inicial do peão.
- `EXIT`: uma posição final do peão.

O labirinto é habitado por um **peão**, que se desloca desde uma posição inicial (`START`) até atingir uma saída (`EXIT`). É necessário descrever o peão, as suas ações e o seu percurso. O peão **movimenta-se** no labirinto nas seguintes direções:

- `NORT` para "cima".
- `EAST` para a "direita".
- `SOUTH` para "baixo".
- `WEST` para a "esquerda".
- `NOOP` para ficar na mesma posição.

Um movimento, antes de ser aplicado, deve ser avaliado tendo em conta o labirinto, a posição do peão e o movimento, de forma a impedir que este se desloque para uma posição ocupada (`WALL`).

O **percurso** que o peão desenvolve deve ficar registado numa (instância de uma) classe apropriada.

3 Implementação

3.1 Contrato

O contrato para o labirinto:

```
public enum Move {NORT, EAST, SOUTH, WEST, NOOP};
public enum MazeCell {EMPTY, WALL, START, EXIT};
public interface IMaze {
    boolean canMove(Pawn, Move); // true IFF pawn can do move in this maze.
    Move[] getOptions(Pawn);     // returns the possible moves of pawn.
    void move(Pawn, Move);       // if pawn can move, change his position.
    boolean isSolvedBy(Pawn);    // true IFF pawn is in EXIT position.
}
public interface IPawn {
    void move(Move);  // changes this pawn position according to move.
    Route getRoute(); // returns the current route.
}
public interface IRoute {
    int getCol();      // returns the column of the current position.
    int getRow();      // returns the row of the current position.
    int getCol(int i); // returns the column of the i-th position.
    int getRow(int i); // returns the row of the i-th position.
    int length();      // returns the number of moves in this route.
    void move(Move);   // append a new position to this route.
}
```

Condições de avaliação:

1. `IMaze.canMove(pawn, move)` deve devolver `false` quando o `move` leva o `pawn` para "fora" da grelha. Isto é, as posições "fora da grelha" são, implicitamente, `WALL`.

2. `IMaze.move(pawn, move)` deve começar por verificar (usando `canMove`) se o movimento `move` é possível na posição de `pawn`. Se o movimento for possível, o **percurso** do peão regista a nova posição. Caso contrário, o percurso regista a posição actual (isto é, a posição actual fica repetida).

3.2 Formato `.maze`

O formato `.maze` permite definir labirintos em ficheiros de texto.

1. Todas as linhas têm o mesmo número de caracteres. Se esta condição falhar, deve ser levantada uma exceção do tipo `MazeFileNumCols(int)` a indicar a linha do ficheiro onde ocorre o problema.

2. Os carateres possíveis em cada linha são:

   - `S` para indicar posições `START`.
   - `E` para indicar posições `EXIT`.
   - `W` para indicar posições `WALL`.
   - `_` para indicar posições `EMPTY`.
   - `\n` para indicar o fim de uma linha (**n.b.** `\n` é um único caracter, que indica "fim de linha").

3. Se, numa linha, ocorrer algum outro caracter, deve ser levantada uma excepção do tipo `MazeFileWrongChar(int row, int col)` a indicar a linha e coluna onde está esse caracter.

Exemplo de um labirinto em formato `.maze`, com três linha e nove colunas, com uma posição inicial em `[0,1]` e saída em `[2,8]`:

```
_SW_______\n
_WWW_W_WW\n
______W__E\n
```