Package FunctionA_CreateMaze

Class MazeGenerator

java.lang.Object
FunctionA_CreateMaze.MazeGenerator

```
public class MazeGenerator
extends Object
```

The MazeGenerator class is to generate a maze with multiple paths, one ENTRY and one EXIT on the opposite edge randomly The algorithm for generating the maze is Depth-First-Search Algorithm (DFS) DFS starts with the randomly created ENTRY on edge (initial current Cell) while all the remaining Cells are BLOCKS Then it expands the PATHs on maze by randomly selecting one of the valid neighbouring Cells of the current Cell which is also the next current cells Also, it finds the EXIT during the expansion process which is the first neighbouring cell of the current cell on the opposite edge of ENTRY

Field Summary

Fields

Modifier and Type	Field	Description
Cell	EntryPoint	
Cell	ExitPoint	

Constructor Summary

Constructors

Constructor	Description
MazeGenerator(int rows, int cols)	Constructs a MazeGenerator object with the specified number of rows and columns.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
Boolean	cellOnCorner(Cell cell)	Checks if a cell is on the corner of the maze.
Boolean	cellOnEdge(Cell cell)	Checks if a cell is on the edge of the maze.
Boolean	cellOnGrid(int row, int col)	Checks if a cell is on the grid.
Boolean	CellOnOppositeEdge(Cell cell_1, Cell cell_2)	Checks if two cells are on the opposite edges of the maze.
Boolean	checkIfEntryPoint(Cell cell)	Checks if a cell is the entry point of the maze.
Boolean	checkIfExitPoint(Cell cell)	Checks if a cell is the exit point of the maze.
boolean	checkValidNeighbors(Cell cell)	Checks if a cell is a valid neighboring cell.
void	EntryPointGenerator()	Generates the entry point of the maze.
void	generateMaze()	Generates the maze by DFS algorithm.
Cell[][]	getMaze()	Retrieves the maze array.
List <Cell>	getValidNeighbors(Cell cell)	Retrieves the valid neighboring cells of a given cell and set information for EXIT if found.
void	initializeMaze()	Initializes the maze grid with cells.

Methods inherited from class java.lang.Object

clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait

Field Details

EntryPoint

```
public Cell EntryPoint
```

ExitPoint

```
public Cell ExitPoint
```

Constructor Details

MazeGenerator

```
public MazeGenerator(int rows,
                     int cols)
```

Constructs a MazeGenerator object with the specified number of rows and columns.

Parameters:

`rows` - The number of rows in the maze.

`cols` - The number of columns in the maze.

Method Details

EntryPointGenerator

```
public void EntryPointGenerator()
```

Generates the entry point of the maze.

initializeMaze

```
public void initializeMaze()
```

Initializes the maze grid with cells.

generateMaze

```
public void generateMaze()
```

Generates the maze by DFS algorithm.

checkValidNeighbors

```
public boolean checkValidNeighbors(Cell cell)
```

Checks if a cell is a valid neighboring cell. Criteria for a valid neighboring cell if the Exit of the maze is not yet found The less than 4 of the 8 connected cells of the cell is PATH, the cell is not yet visited by the DFS algorithm, the cell is not on the edge (excluding the

opposite edge of ENTRY) Criteria for a valid neighbouring cell if the Exit of the maze is found The less than 4 of the 8 connected cells of the cell is PATH, the cell is not yet visited by the DFS algorithm, the cell is not on the edges (all)

Parameters:

`cell` - The cell to check.

Returns:

True if the cell is a valid neighboring cell, false otherwise.

getValidNeighbors

```
public List <Cell> getValidNeighbors(Cell cell)
```

Retrieves the valid neighboring cells of a given cell and set information for EXIT if found.

Parameters:

`cell` - The cell to retrieve the neighbors for.

Returns:

A list of valid neighboring cells.

cellOnGrid

```
public Boolean  cellOnGrid(int row,
                           int col)
```

Checks if a cell is on the grid.

Parameters:

`row` - The row index of the cell.

`col` - The column index of the cell.

Returns:

True if the cell is on the grid, false otherwise.

cellOnEdge

```
public Boolean  cellOnEdge(Cell cell)
```

Checks if a cell is on the edge of the maze.

Parameters:

`cell` - The cell to check.

Returns:

True if the cell is on the edge, false otherwise.

cellOnCorner

```
public Boolean  cellOnCorner(Cell cell)
```

Checks if a cell is on the corner of the maze.

Parameters:

`cell` - The cell to check.

Returns:

True if the cell is on the corner, false otherwise.

checkIfEntryPoint

```
public Boolean  checkIfEntryPoint(Cell cell)
```

Checks if a cell is the entry point of the maze.

Parameters:

`cell` - The cell to check.

Returns:

True if the cell is the entry point, false otherwise.

checkIfExitPoint

```
public Boolean  checkIfExitPoint(Cell cell)
```

Checks if a cell is the exit point of the maze.

Parameters:

`cell` - The cell to check.

Returns:

True if the cell is the exit point, false otherwise.

CellOnOppositeEdge

```
public Boolean  CellOnOppositeEdge(Cell cell_1,
                                   Cell cell_2)
```

Checks if two cells are on the opposite edges of the maze.

Parameters:

`cell_1` - The first cell.

`cell_2` - The second cell.

Returns:

True if the cells are on opposite edges, false otherwise.

getMaze

```
public Cell[][] getMaze()
```

Retrieves the maze array.

Returns:

The maze array.