**Practice "Empty maze"**

Download the Mazes project and study it. Several fixed mazes have been prepared there. In each labyrinth, you need to take the robot to the exit - a cell marked with a green circle.

The maze itself is not known and the robot has no sensors to explore, but the type of maze and its size are known.

This is enough to build a route and give the necessary commands to the robot.

In this problem, a labyrinth is an empty room surrounded by a wall around the perimeter, in which from the upper left corner with coordinates (1, 1) you need to go to the lower right corner with coordinates (width-2, height-2).

Additional restrictions:

1. It is forbidden to use more than one cycle in one method.

2. It is forbidden to have methods longer than 6 lines of code.

3. It is forbidden to use the catch keyword

4. It is allowed to create helper methods, but only with friendly names, including argument names.

About comprehensibility of methods

It is important to be able to create such methods so that they are understandable without having to look inside the method. When working on a large project, this will allow you and your colleagues to navigate the code faster.

For example, such a method is incomprehensible, because it is not clear what i is, nor where the robot will move:

void Move(Robot robot, int i)

And this method is bad because it misleads the reader:

void MoveLeft(Robot robot, int stepCount)

{

for (int i = 0; i < stepCount - 2; i++)

robot.MoveTo(Direction.Left);

}

Indeed, in the method signature, it promises that it will take stepCount steps, and instead takes stepCount-2 steps.

Practice creating clear methods for this series of problems.

// Paste the final file content here EmptyMazeTask.cs

namespace Mazes

{

public static class EmptyMazeTask

{

public static void MoveOut(Robot robot, int width, int height)

{

for (int i = 0; i < width - 3; i++)

{ robot.MoveTo(Direction.Right); }

for (int i = 0; i < height - 3; i++)

{ robot.MoveTo(Direction.Down); }

}

}

}