

Problem-solving task

During the selection process, we require each candidate to complete a task in order to make sure you have the skills needed for this position.

Test duration

is 5 days from the date the task is received.

The task solution

must be sent as a GitHub URL (for each task).

To send us your task solution, you must upload it to a private git-repo and name it firstName_lastName_fatcat_task. Share it with the following accounts:

Zoran Lazić, username: **ezelohar**

Luka Patarčić, username: **LukaPatarcic**

If you have any questions about the task, you can write to us at this email address: careers@fatcatcoders.com

Important: Make sure that the GitHub link is also sent to careers@fatcatcoders.com

Please make sure to follow the instructions for sending the task solution carefully. Otherwise, your solution might not be reviewed. Thank you!

Disclaimer

The only purpose of this task is to assess candidate's skills. The sent code will not be used for commercial purposes.

Good luck, and have fun! 🍀

Task

Find a way through the $n * n$ matrix. You can use any algorithm you find on the internet, but make sure to understand it.

[https://en.wikipedia.org/wiki/Matrix_\(mathematics\)](https://en.wikipedia.org/wiki/Matrix_(mathematics)).

Let's define our items.

Moving object: the imaginary object we will move through our matrix. In the further text, we will use the short name **MO**.

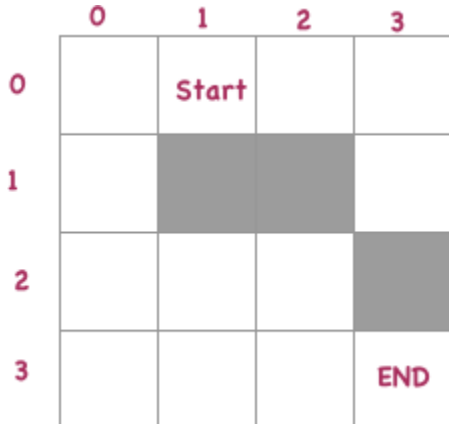
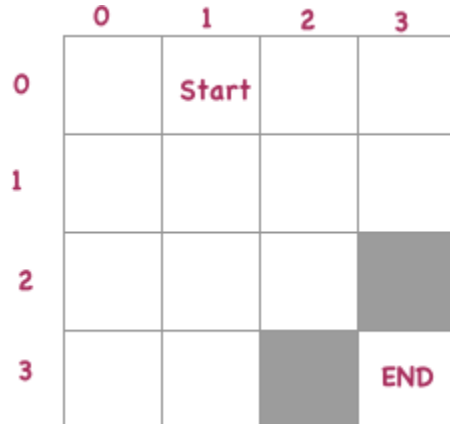
Blocking objects: the blocking object that will be used to block the path of our MO. In the further text, we will use the short name **BO**.

The goal is to bring our moving object (**MO**) from the start coordinates to the end coordinates while generating n blocking objects (**BO**) with every move the moving object (**MO**) makes. This means there is n **BO** created on each step where your **MO** cannot move to. It is important to note that **BOs** are placed randomly on the matrix at each step.

You can only move the **MO** *horizontally* or *vertically*, you cannot move it *diagonally*.

BO can be put anywhere on the matrix except on the path **MO** already passed through.

If the **BO** closes **all possible available paths**, you will retry the generation of **BO** until there is a path. If there are no combinations that will contain the path, reduce the number of **BO's** by **1** and retry path selection. This can be applied until the path is found.

Good blocking object example	Bad blocking object example,
	

The result should be an array of objects containing ***movingObjectCoordinates*** and ***blockingObjectsCoordinates*** keys. For each step of the algorithm, you will have an object. This object represents the current coordinates of our **MO** (**movingObjectCoordinates**) and all the **BO** and their coordinates (**blockingObjectCoordinates**).

We want to be able to run a full test of the path generation, including moments where there was no path due to number of generated BO's and then BO number had to be reduced.

movingObjectCoordinates - represents x and y coordinates where the MO has moved

blockingObjectCoordinates - an array of x and y coordinates for all the newly generated BO

```
[
  {
    movingObjectCoordinates: [x, y],
    blockingObjectCoordinates: [[BOx1, BOy1], [BOx2, BOy2]], // a newly generated
    blocking block based on the number of blocking blocks per move
  }
]
```

At the end, we also want to measure the speed needed to execute the code.

This algorithm needs to be visually represented using React. For each step, you need to show the position of the path of **MO** and the current **MO** location and all generated **BO**'s for the given step. The movement history of **MO** is kept while **BO** position is reset each after each turn. There should be inputs that can change

1. The size of the matrix
2. Start and end positions
3. The number of blocking objects

You can choose the design. See the example below

Step 1	Step 2
<div> <div>0123</div> <div> <div>0</div> <div>MO START</div> <div></div> <div></div> <div></div> </div> <div> <div>1</div> <div>BO</div> <div></div> <div>BO</div> <div></div> </div> <div> <div>2</div> <div>BO</div> <div></div> <div></div> <div></div> </div> <div> <div>3</div> <div></div> <div></div> <div></div> <div>END</div> </div> </div>	<div> <div>0123</div> <div> <div>0</div> <div>MO START</div> <div>MO</div> <div>BO</div> <div></div> </div> <div> <div>1</div> <div></div> <div></div> <div></div> <div></div> </div> <div> <div>2</div> <div></div> <div>BO</div> <div></div> <div>BO</div> </div> <div> <div>3</div> <div></div> <div></div> <div></div> <div>END</div> </div> </div>

Let's break down all items once again

1. **MO** moves 1 coordinate at each turn (horizontally or vertically)
2. Before **MO** moves, at the start of the turn, we generate our **BO**'s
3. After each **BO** generation, the **MO** needs to calculate the new shortest path and move one coordinate.
4. **MO** can move to any non blocked field
5. **BO** cannot be placed on the fields **MO** already passed through
6. If there are too much **BO**'s so they block all possible paths, reduce the **BO** number by 1 and try again. This can be repeated until **BO**'s reach 0, which is a very specific case that **MO** passed in his journey through all fields.

Requirements

You will have to complete all of these for your task to be considered done.

1. Typescript is mandatory
2. Set all default values for matrix, start end, and blocking objects.
3. Make sure you measure your program execution time.
4. Make sure you apply linter rules. The code should be clean.
5. You can use any algorithm you find on the internet, but make sure to understand it.
6. Visually represent your solution using React
7. All variables can be changed through HTML input fields
 - a. Size of matrix
 - b. Start and end position
 - c. Number of blocking objects
8. Create a button that runs the following matrix sizes in sequence (one after another) and store the time execution results in the table and number of steps it took to reach the end.
 - a. $5 * 5$
 - i. First Iteration: $BO = 5$
 - ii. Second Iteration: $BO = 10$
 - iii. Third Iteration: $BO = 15$
 - b. $10 * 10$
 - i. First Iteration: $BO = 10$
 - ii. Second Iteration: $BO = 30$
 - iii. Third Iteration: $BO = 81$
 - c. $20 * 20$
 - i. First Iteration: $BO = 30$
 - ii. Second Iteration: $BO = 100$
 - iii. Third Iteration: $BO = 361$

