Lefkopoulos Miltiadis

Problem Representation

Pairs were used to represent the problem. More specifically, the representation of parked cars was implemented with a vector that contains each car, in the following form, *pair<pair<int, int>, bool>>* which translates to *pair<pair<x, y>, horizontal>*

In the first part of the pair, I save a pair that has the coordinates (x, y) of the car, while in the second part I save the orientation of the car (true for horizontal – false for vertical).

To represent a labyrinth state, I have the Maze class, in which the cars are stored, the positions of the obstacles in a two-dimensional table (the positions of the cars are also considered obstacles), the previous move, and finally the previous maze from which the program arrived at the state it is in.

Implementation of basic methods

So, we can proceed to solve the problem with the BFS algorithm. BFS primarily uses a queue, and the expand() method of the Maze class.

Method expand()

For each car, all its possible movements should be found. So, by going through vector cars, I control every movement of every car.

- The car has a horizontal orientation
  ○ The car can go left ○ The car can go right
- The car has a vertical orientation ○ The car can go up ○ The car can go down

These checks are performed using the goLeft(), goRight(), goUp(), goDown() methods.

In each of these methods, it is checked whether the car is on the edge in order to erase it from the table with the cars (it has successfully exited the maze), and remove it from an obstacle, or whether the position in which the car wants to move is free or not in order to move successfully and change the position of the obstacle it causes.

The BFS algorithm works with a simple vector to store child movements (BFS()), but also with the use of unordered_map (BFS2()).

The key to unordered_map is a long long number which consists of the coordinates of each car and its orientation.

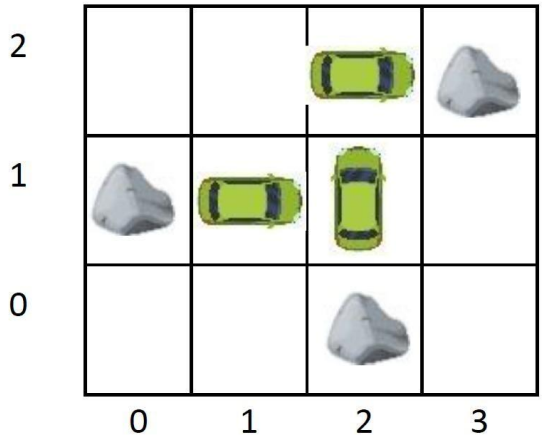
B.C.

221210111

- Car 1 (221) ○ x (2) ○ and (2) ○ horizontal (1)
- Car 2 (210) ○ x (2) ○ and (1) ○ horizontal (0)
- Car 3 (111) ○ x (1) ○ and (1) ○ horizontal (1)

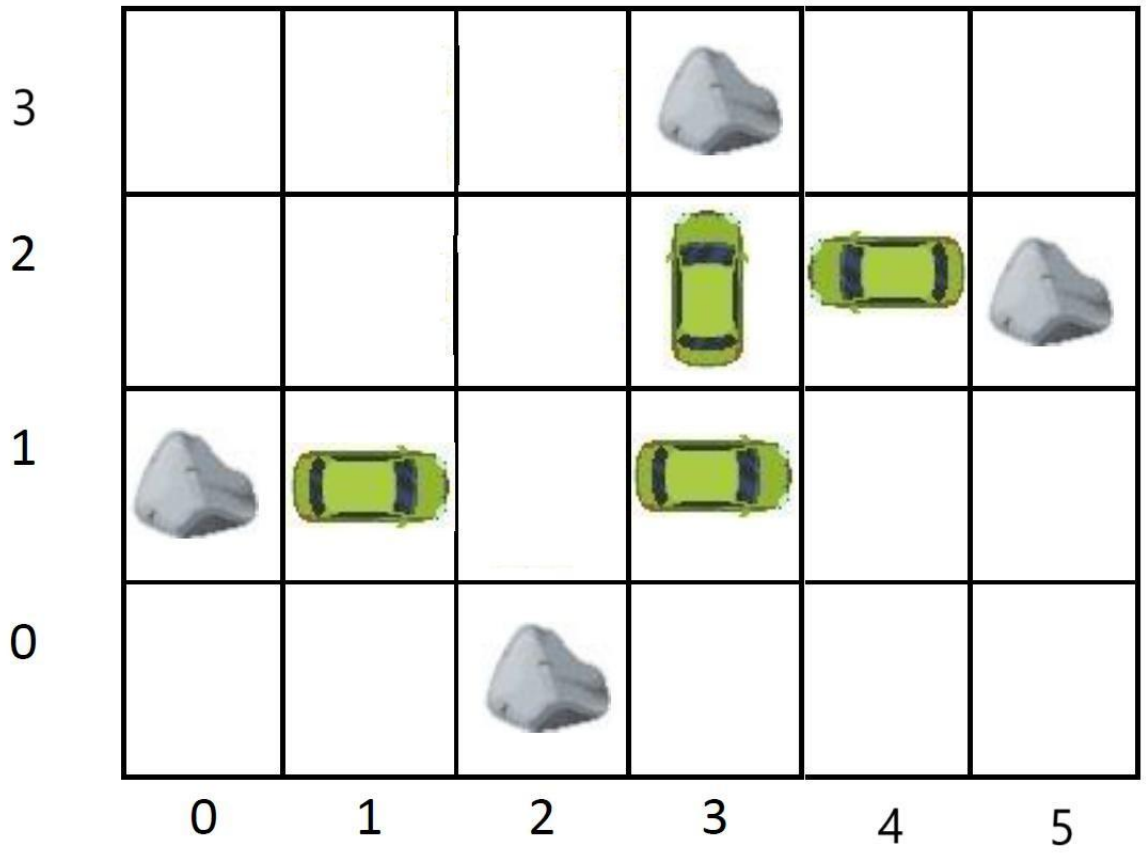
Examples

- Example 1 (Task)



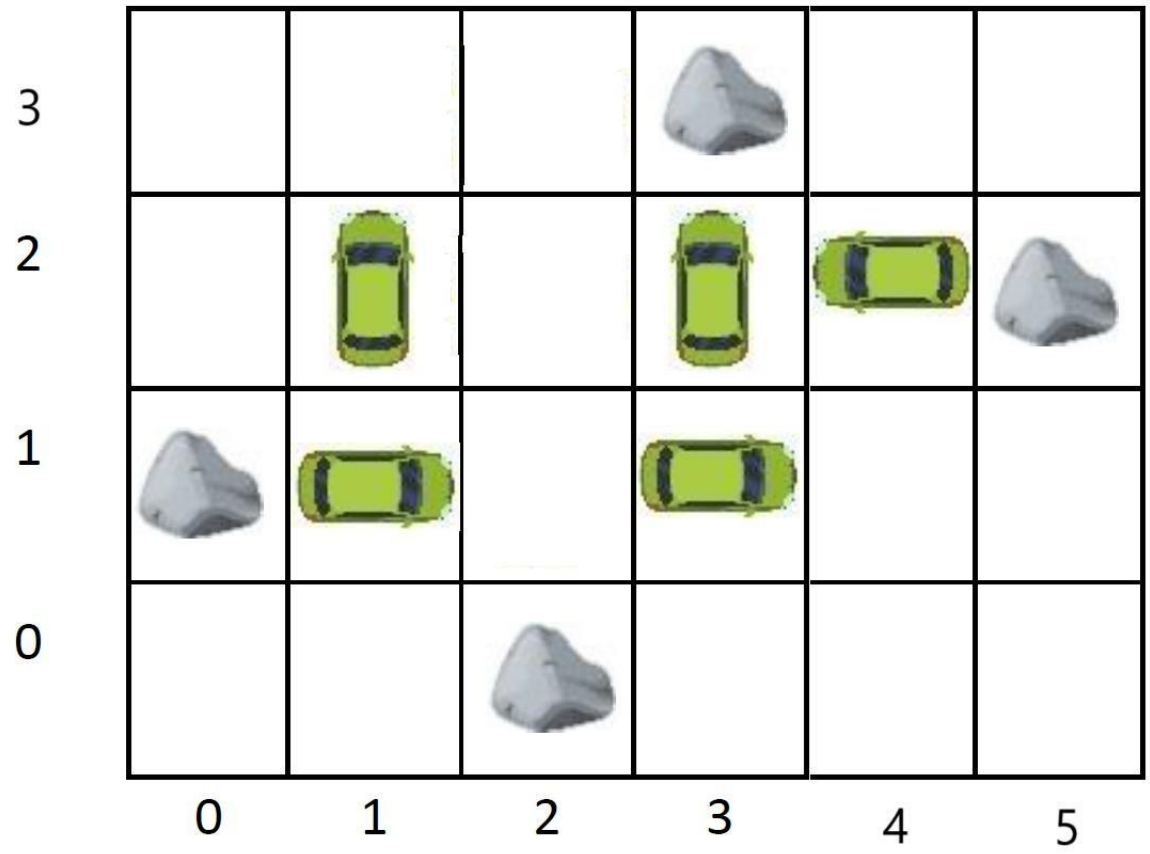

```
BFS: depth = 8, Mem: 51, Examined: 39
Left <- 2-2
Left <- 1-2
Left <- 0-2
Up <- 2-1
Up <- 2-2
Right <- 1-1
Right <- 2-1
Right <- 3-1
```

- Example 2



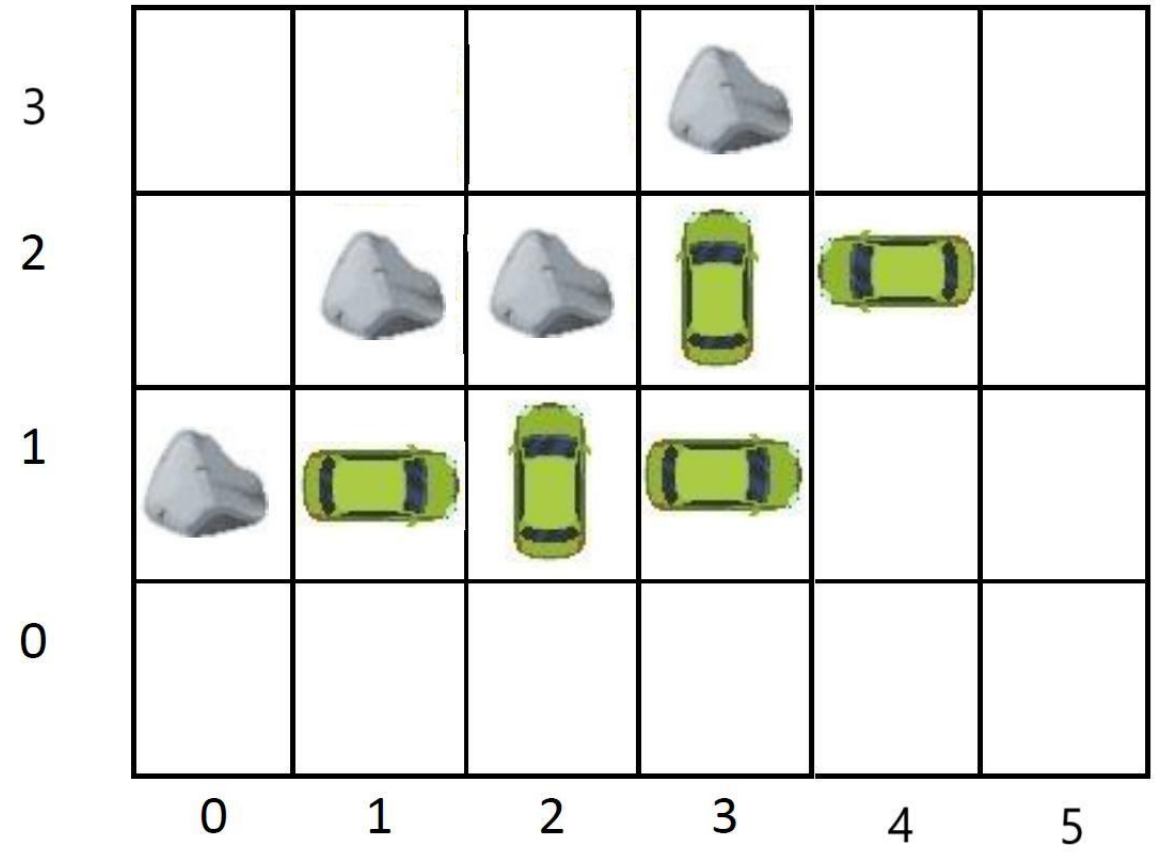

```
BFS: depth = 16, Mem: 373, Examined: 338
Right <- 1-1
Right <- 3-1
Right <- 2-1
Right <- 4-1
Right <- 3-1
Right <- 5-1
Right <- 4-1
Right <- 5-1
Down <- 3-2
Down <- 3-1
Down <- 3-0
Left <- 4-2
Left <- 3-2
Left <- 2-2
Left <- 1-2
Left <- 0-2
```

- Example 3




```
BFS: depth = 18, Mem: 1803, Examined: 1520
Right <- 1-1
Right <- 3-1
Right <- 2-1
Right <- 4-1
Right <- 3-1
Right <- 5-1
Right <- 4-1
Right <- 5-1
Down <- 3-2
Down <- 3-1
Down <- 3-0
Left <- 4-2
Left <- 3-2
Up <- 1-2
Left <- 2-2
Left <- 1-2
Left <- 0-2
Up <- 1-3
```

- Example 4




```
BFS: depth = 15, Mem: 699, Examined: 589
Right <- 3-1
Right <- 4-1
Right <- 5-1
Down <- 3-2
Down <- 3-1
Down <- 3-0
Right <- 4-2
Right <- 5-2
Down <- 2-1
Right <- 1-1
Right <- 2-1
Right <- 3-1
Right <- 4-1
Right <- 5-1
Down <- 2-0
```