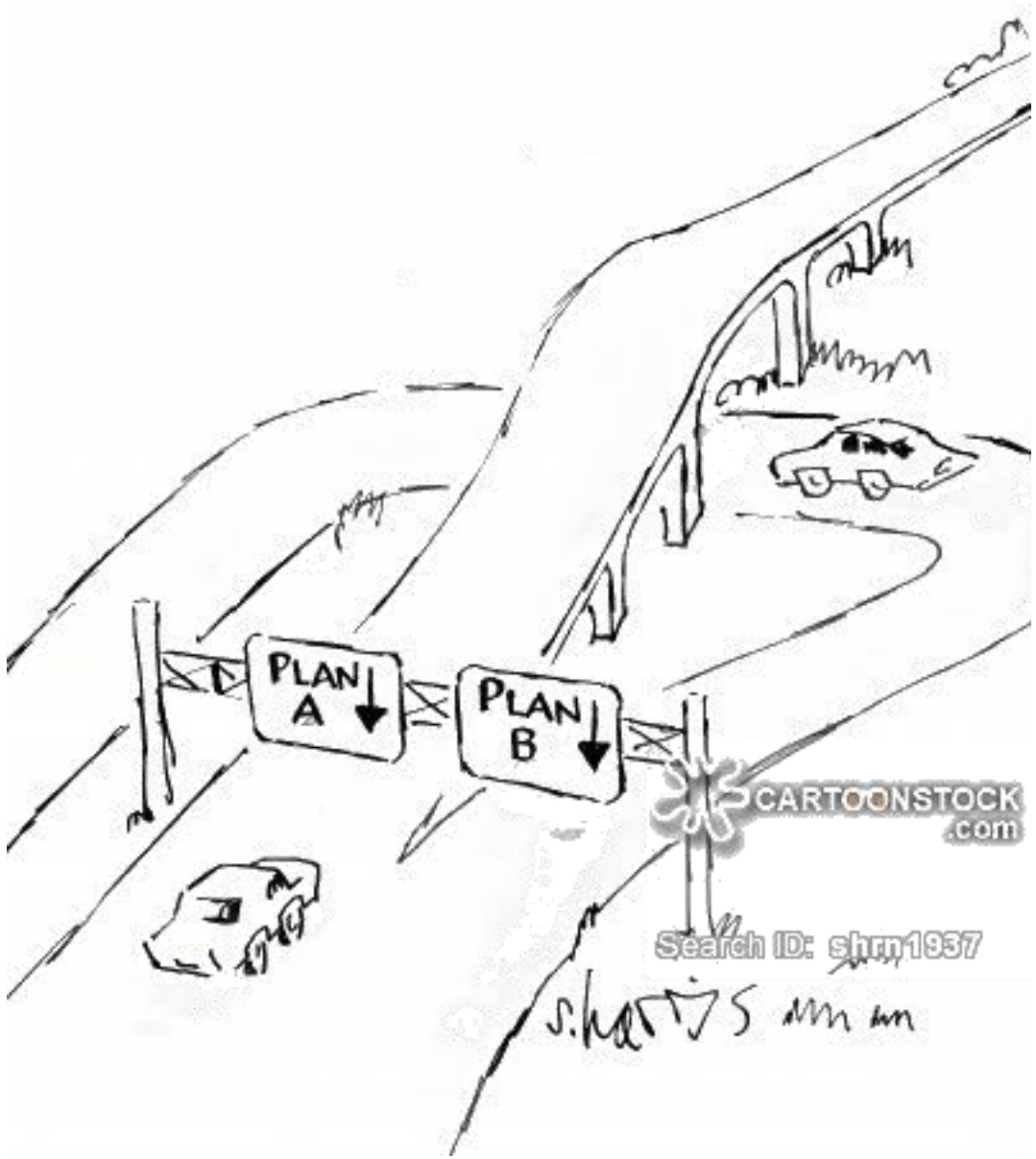


Backtracking

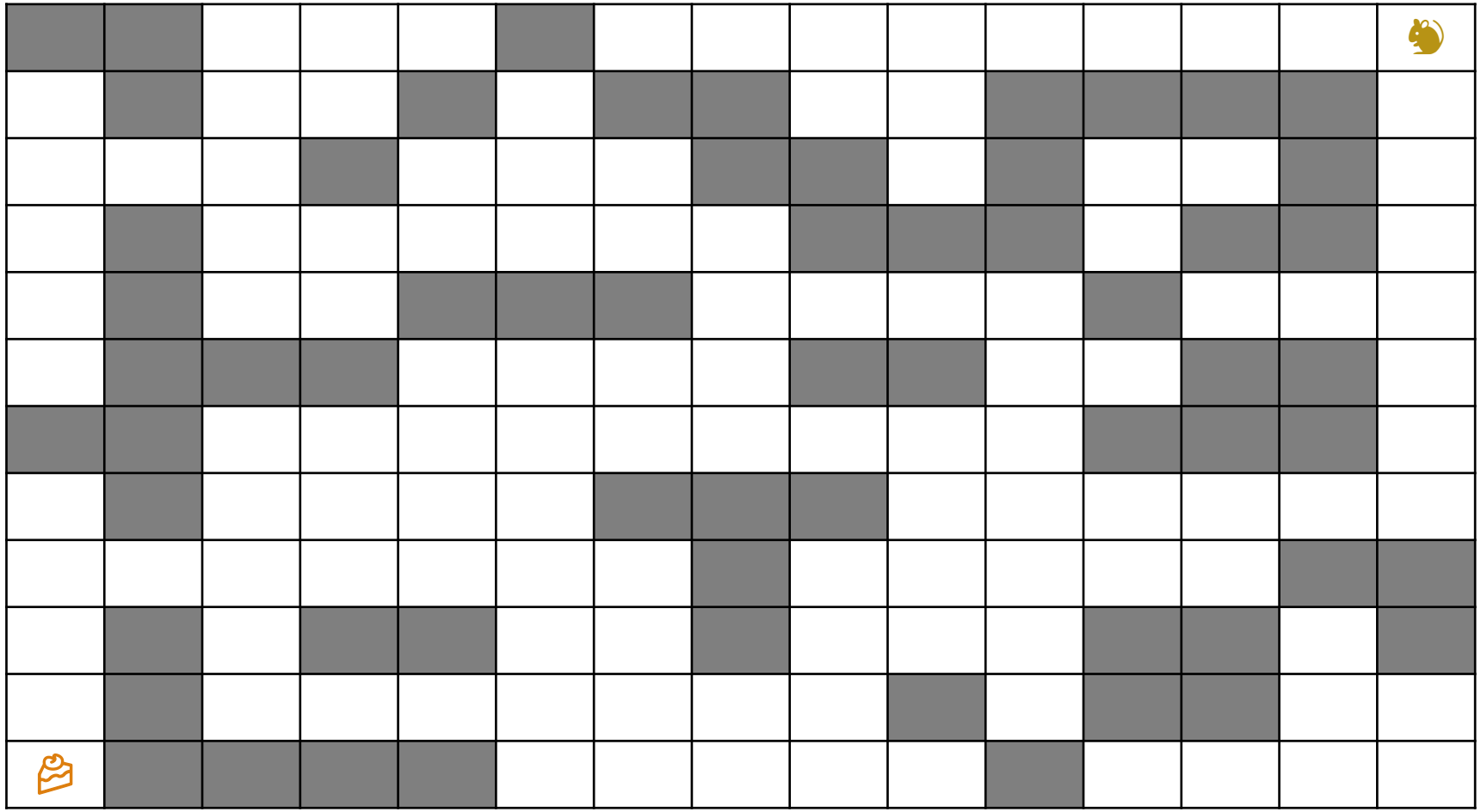
Dr. Tamal Chakraborty

Rat In A Maze

"This place is hard to find . . . but you'll see . . . it's A-MAZE-ING!"

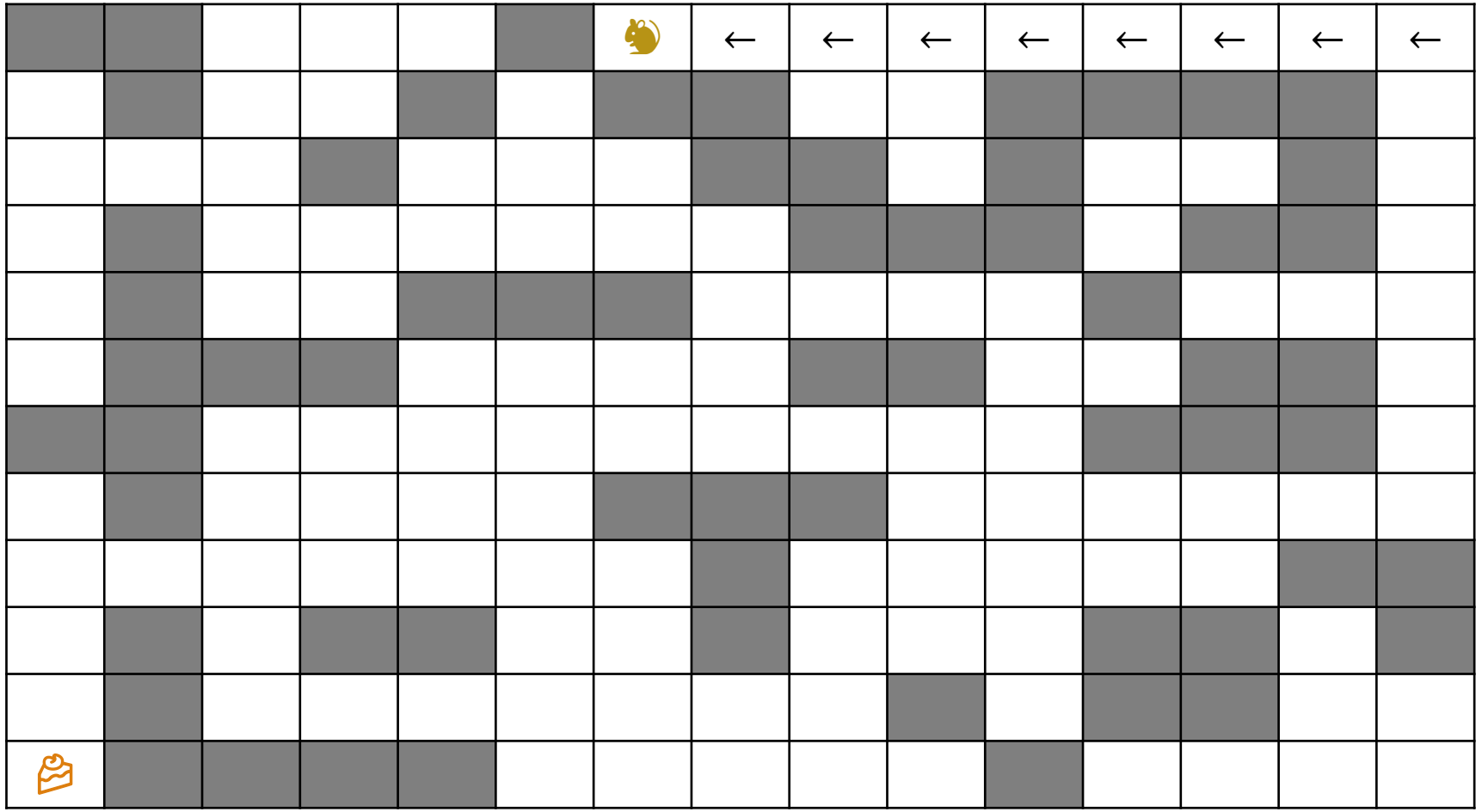
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



Rat In A Maze

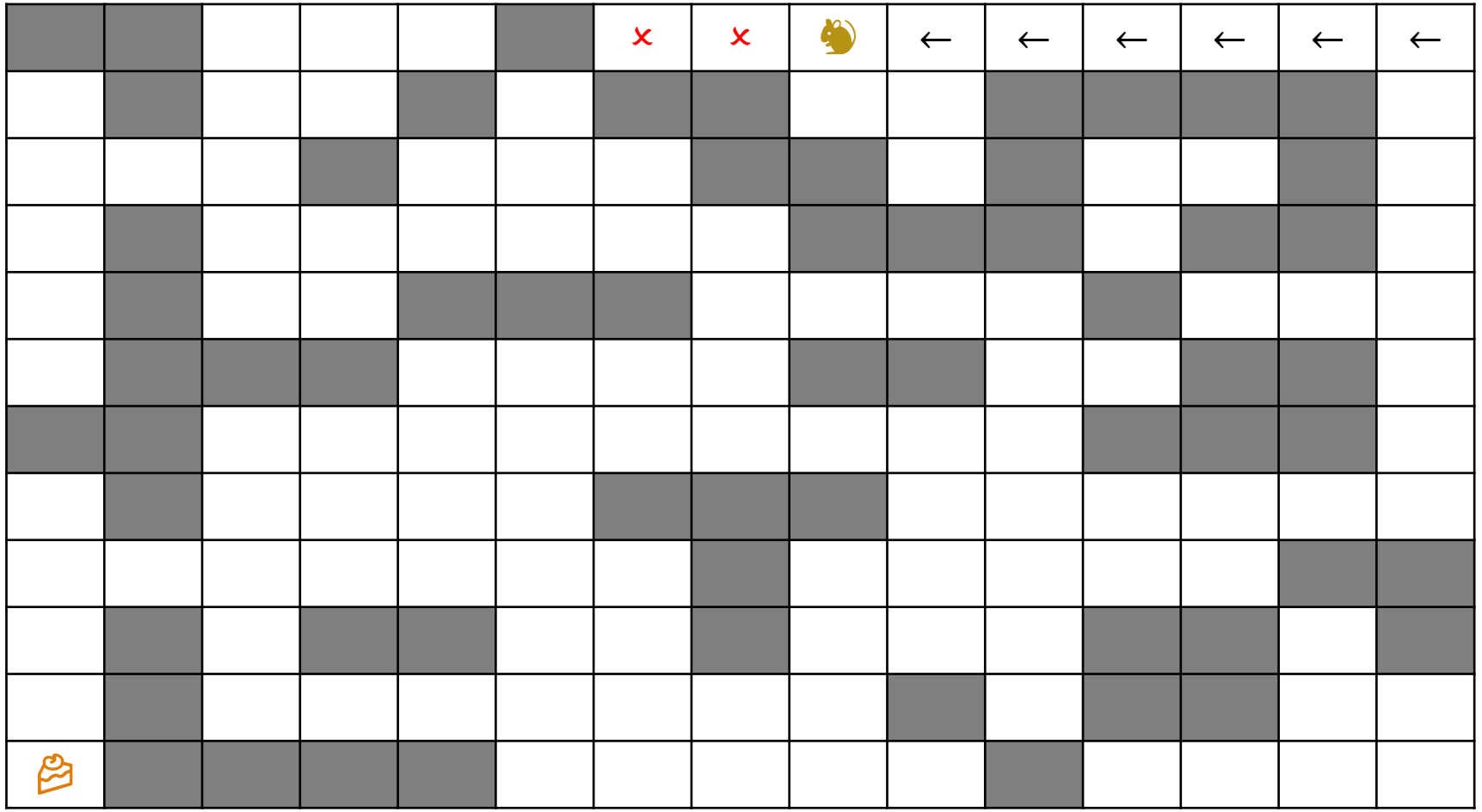
Move Order	
←	left
↓	down
→	right
↑	up



Dead End:
Move backward until we reach a square from which a forward move is possible.

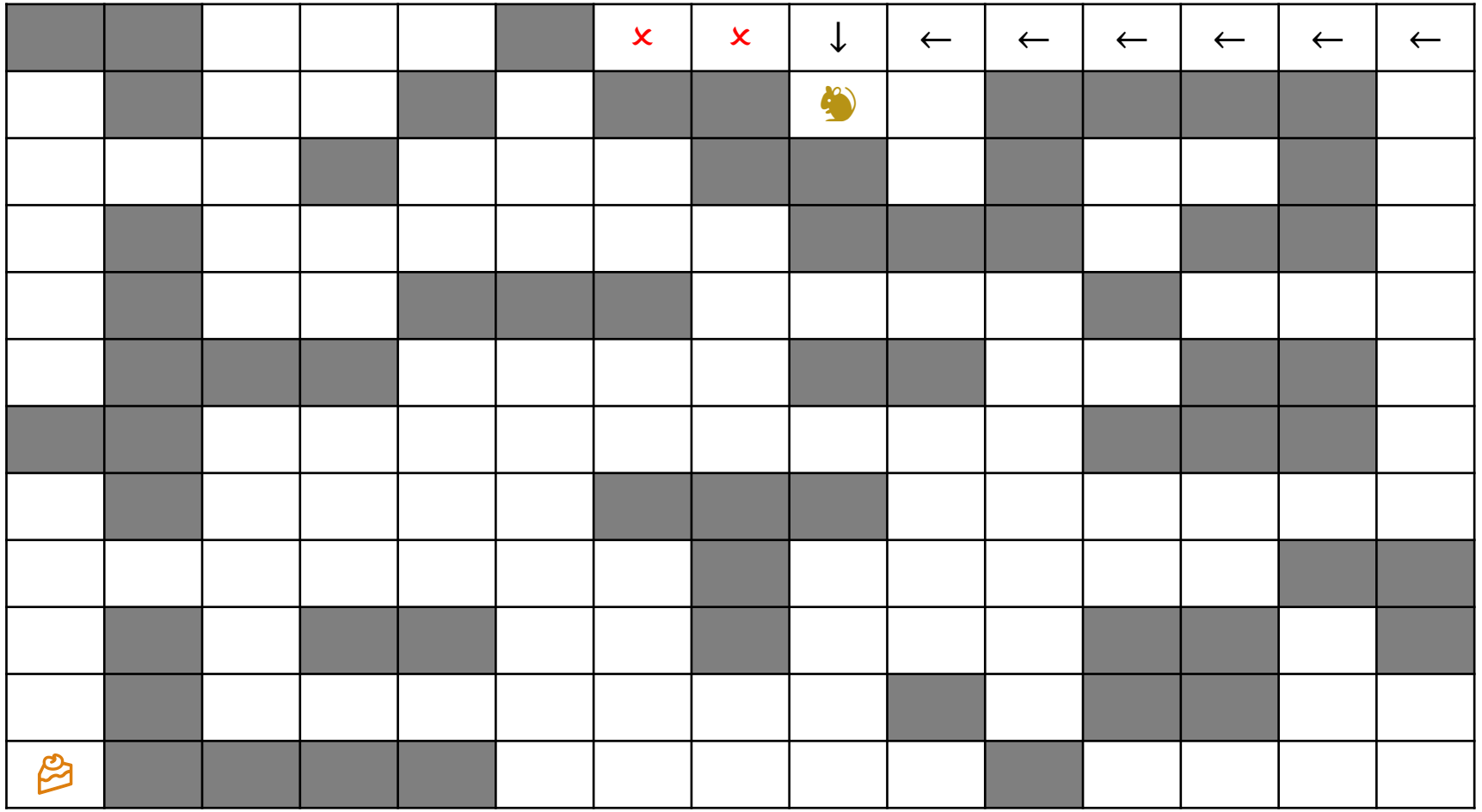
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



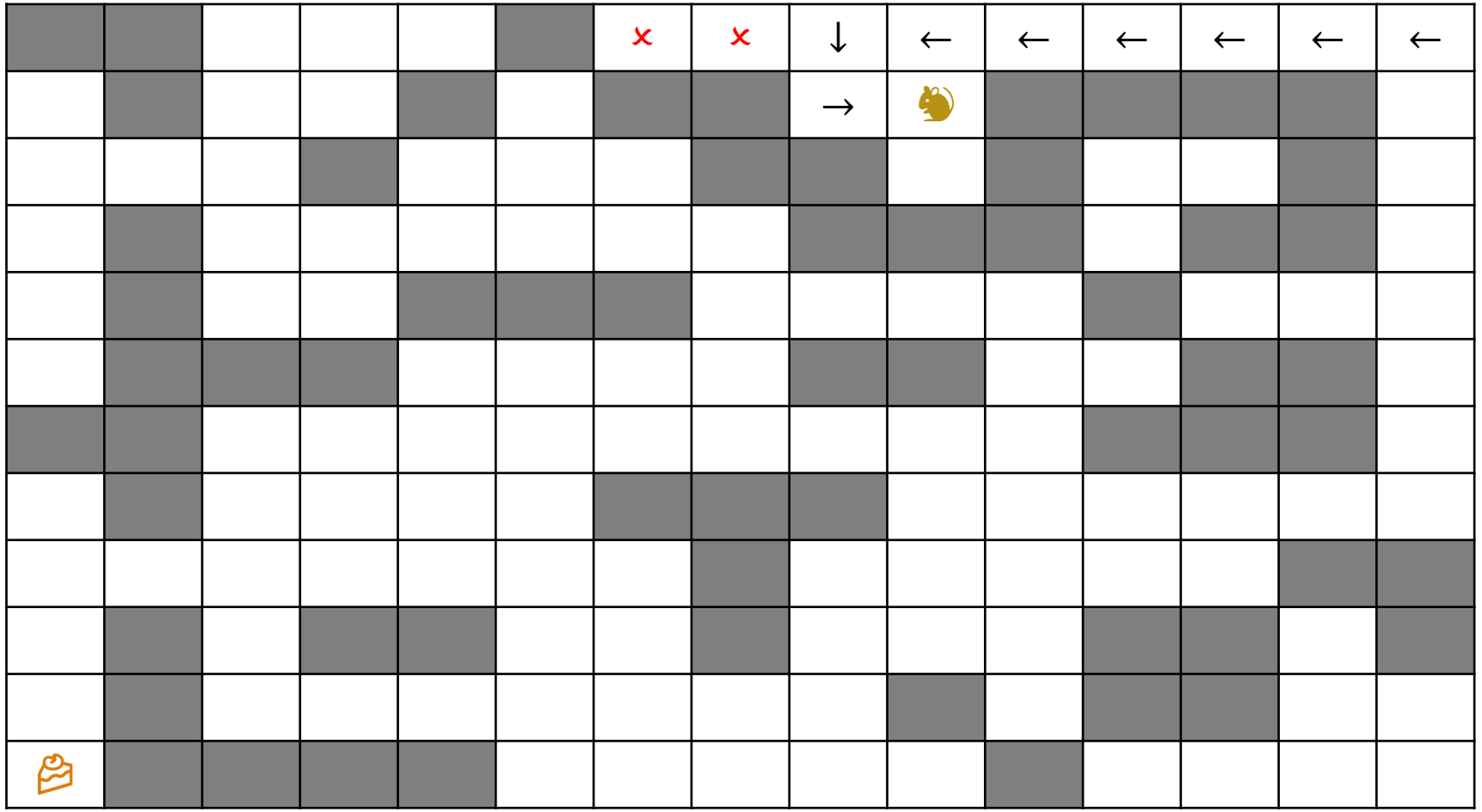
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



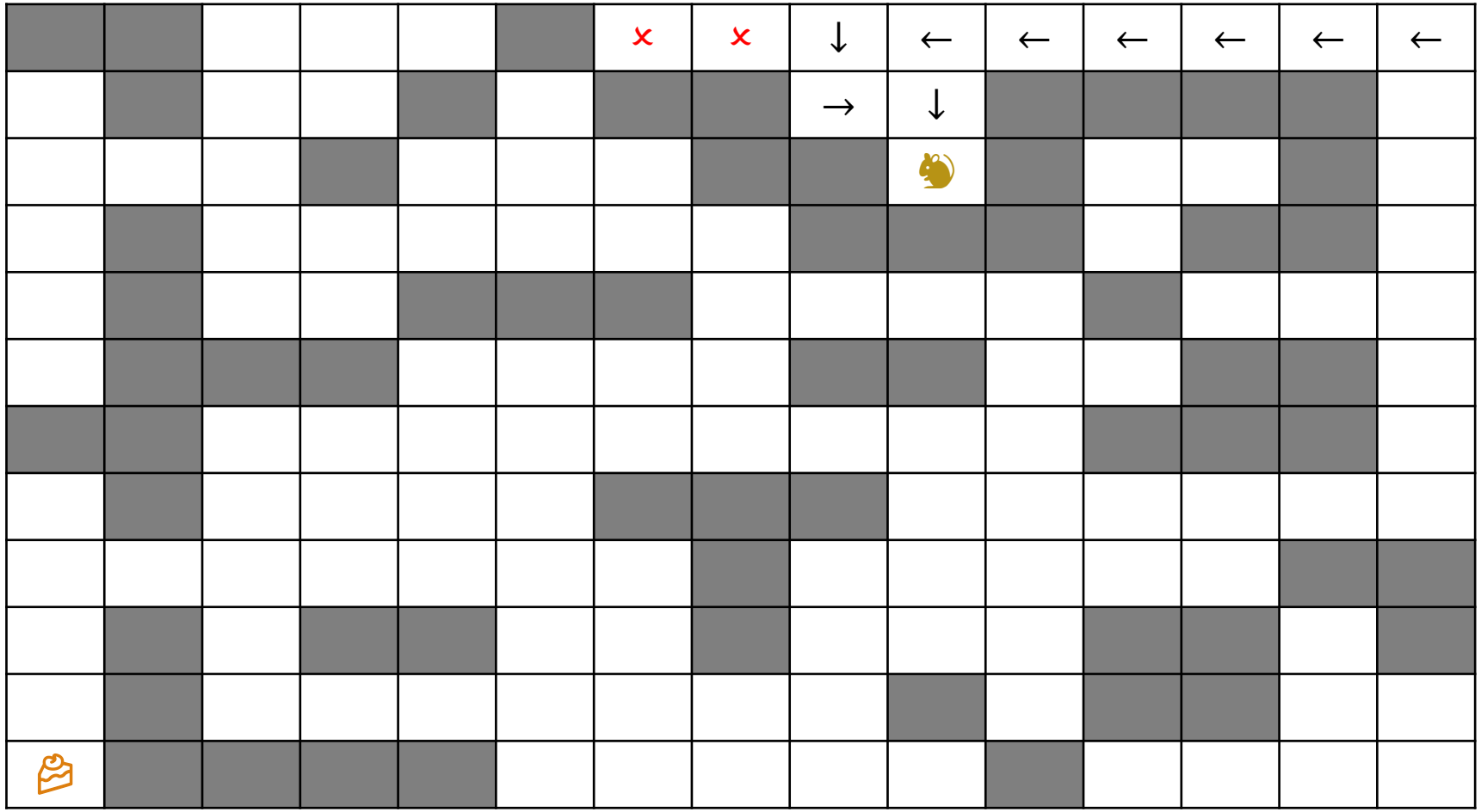
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



Rat In A Maze

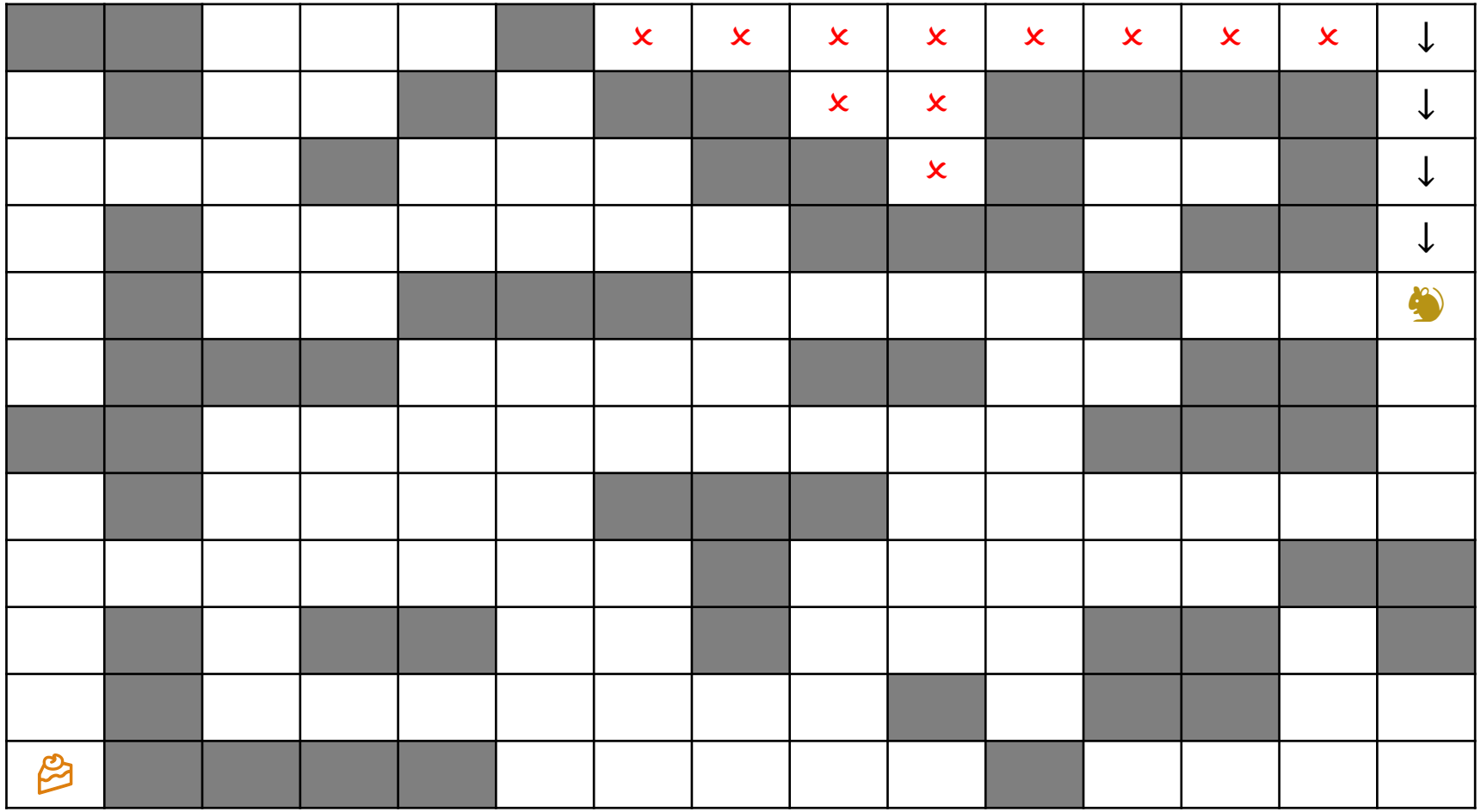
Move Order	
←	left
↓	down
→	right
↑	up



<u>Dead End</u>:
Move backward until we reach a square from which a forward move is possible.

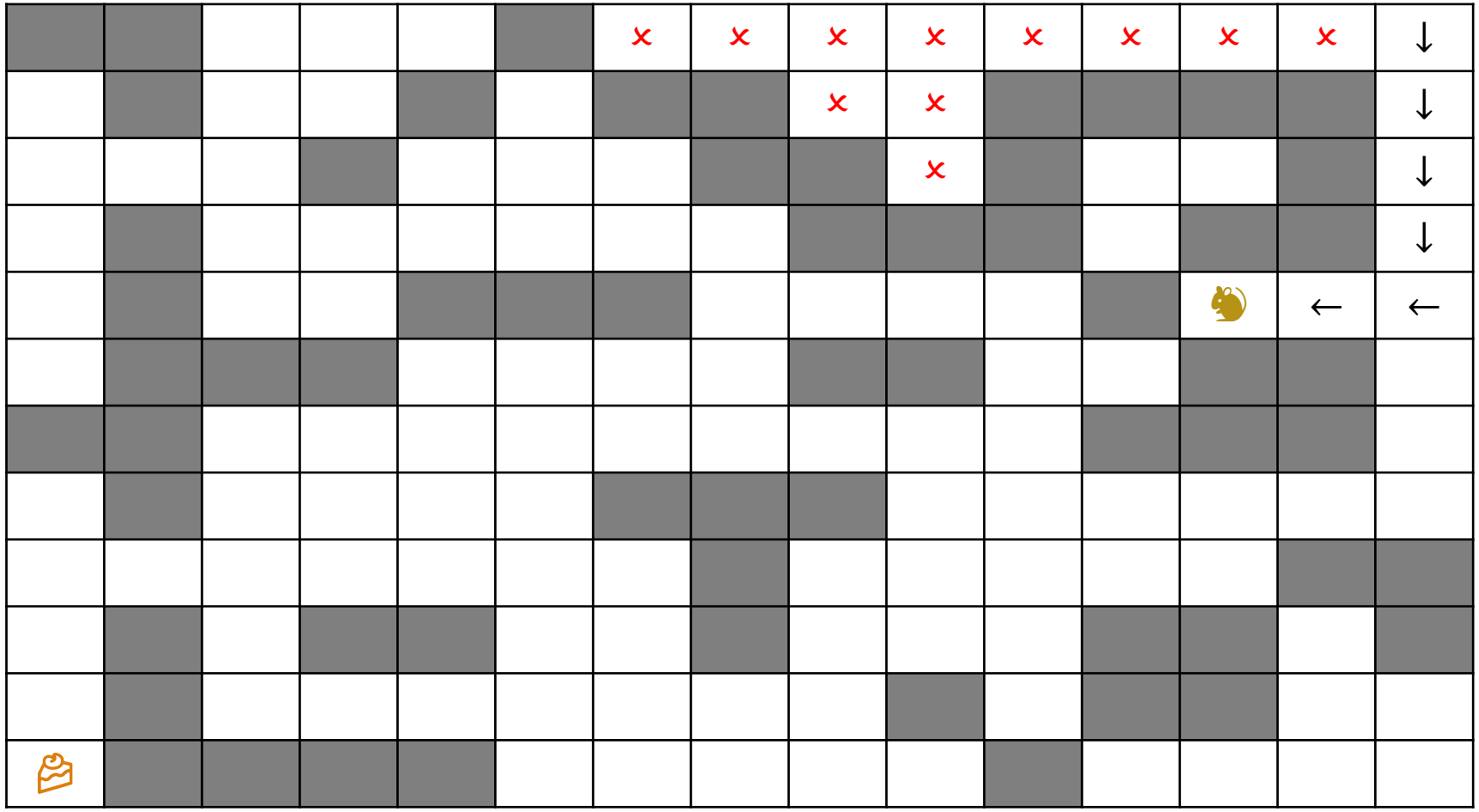
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



Rat In A Maze

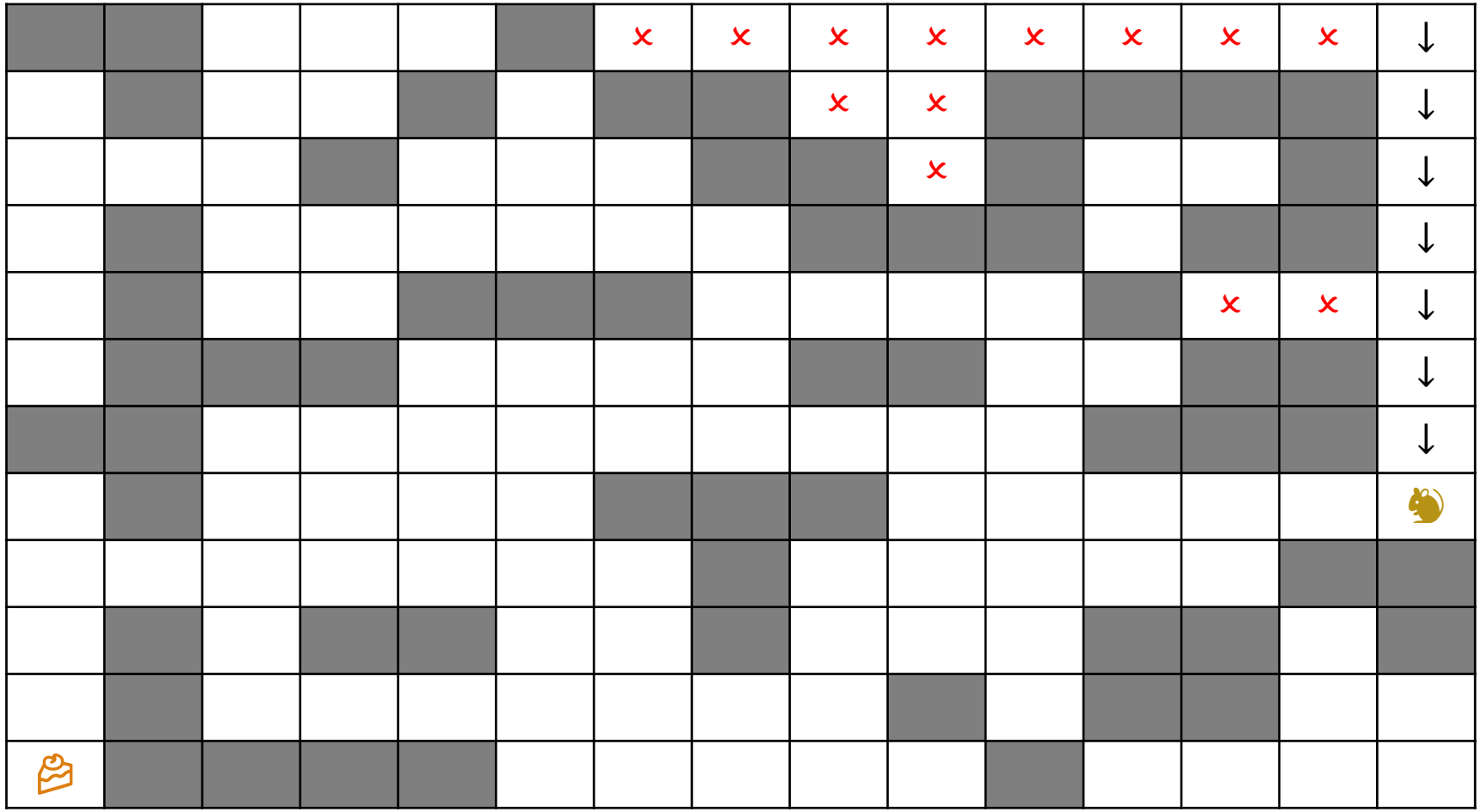
Move Order	
←	left
↓	down
→	right
↑	up



Dead End:
Move backward until we reach a square from which a forward move is possible.

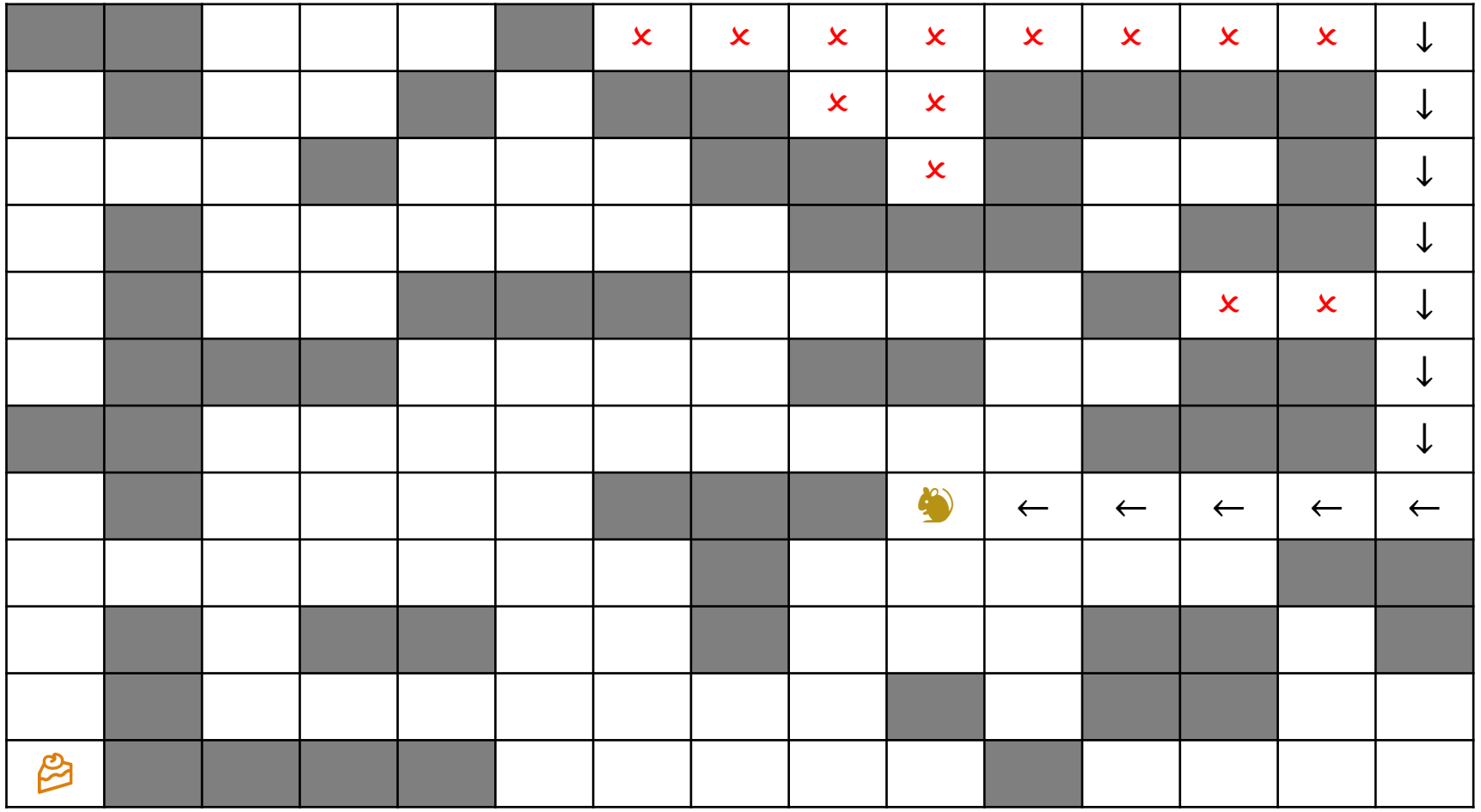
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



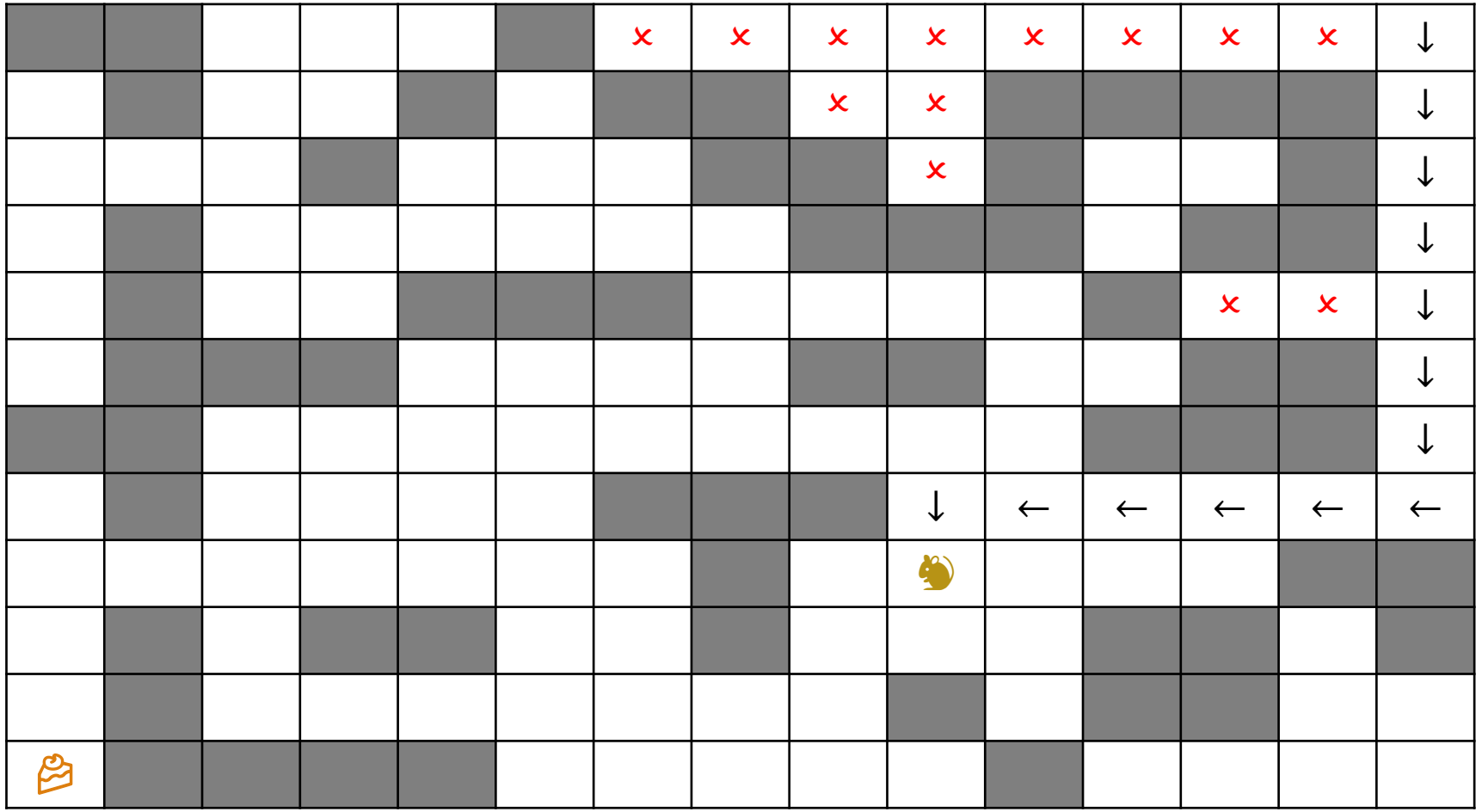
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



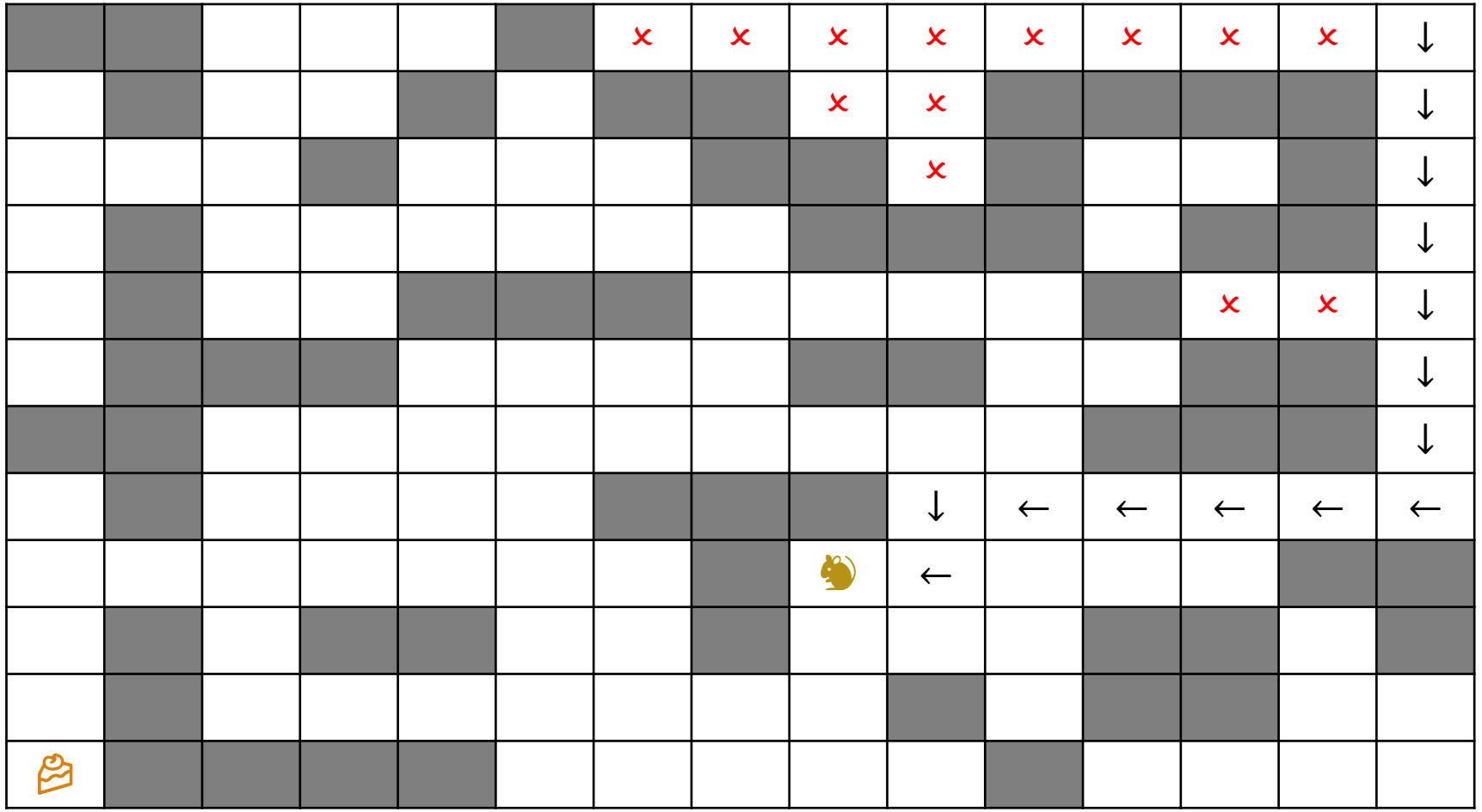
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



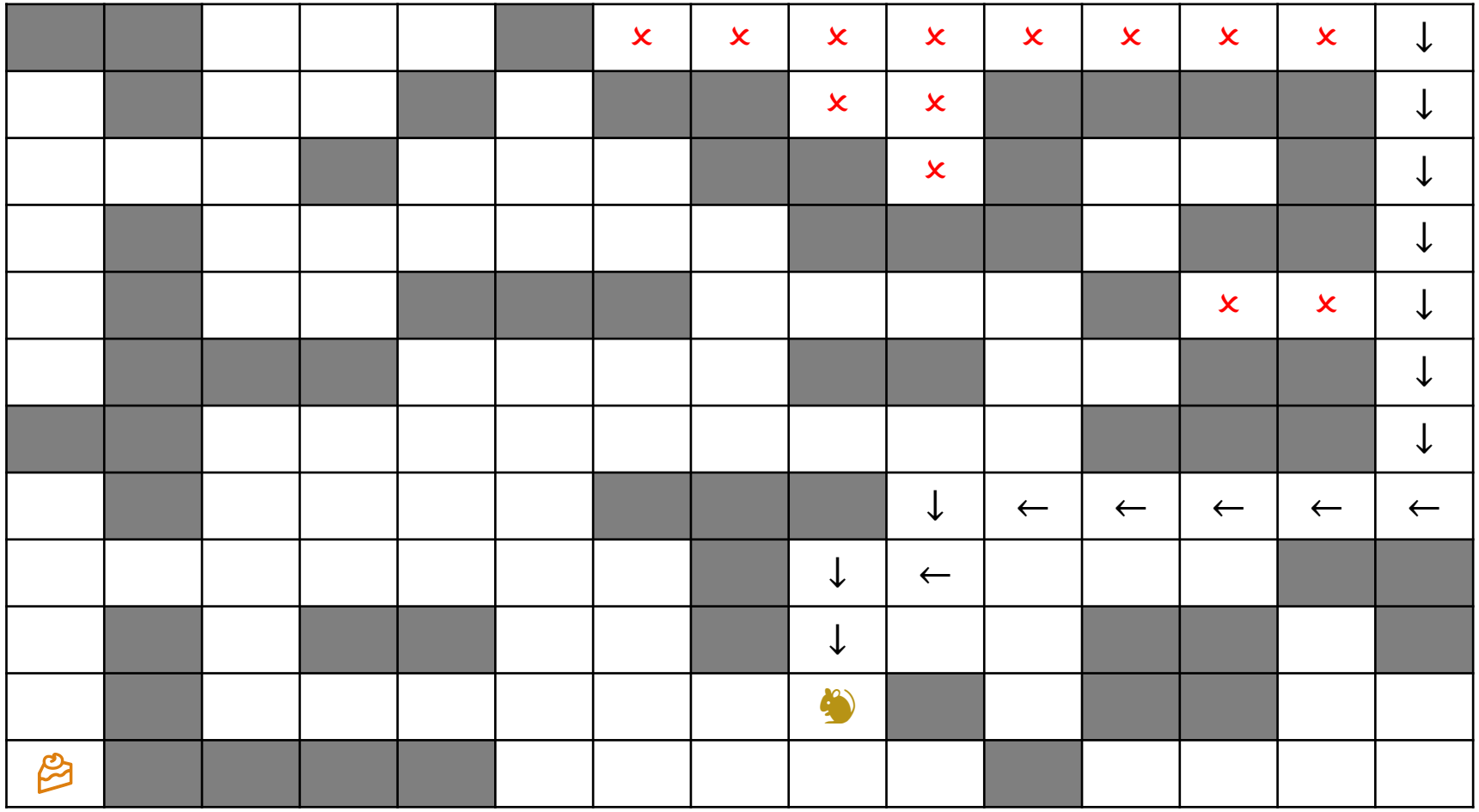
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



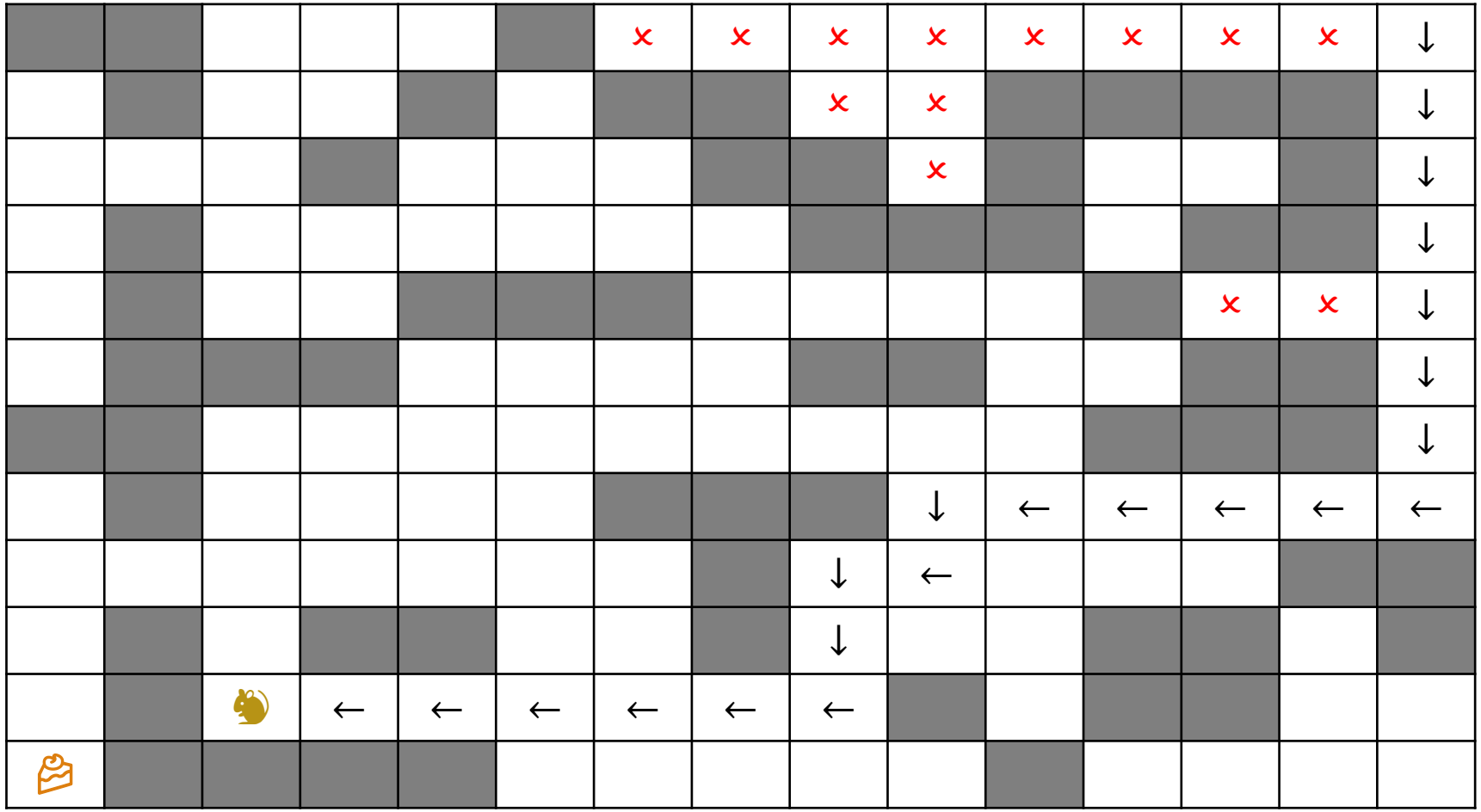
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



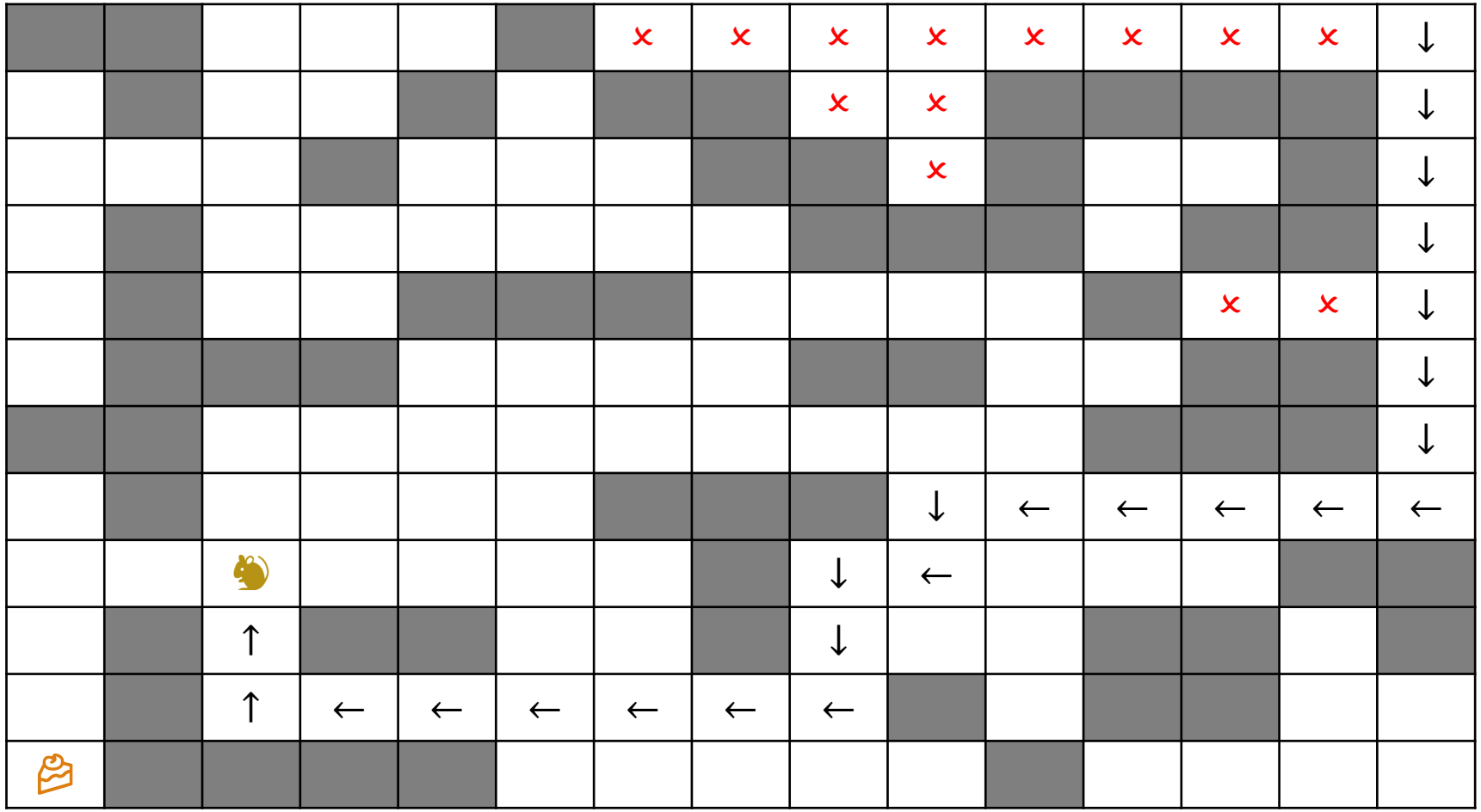
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



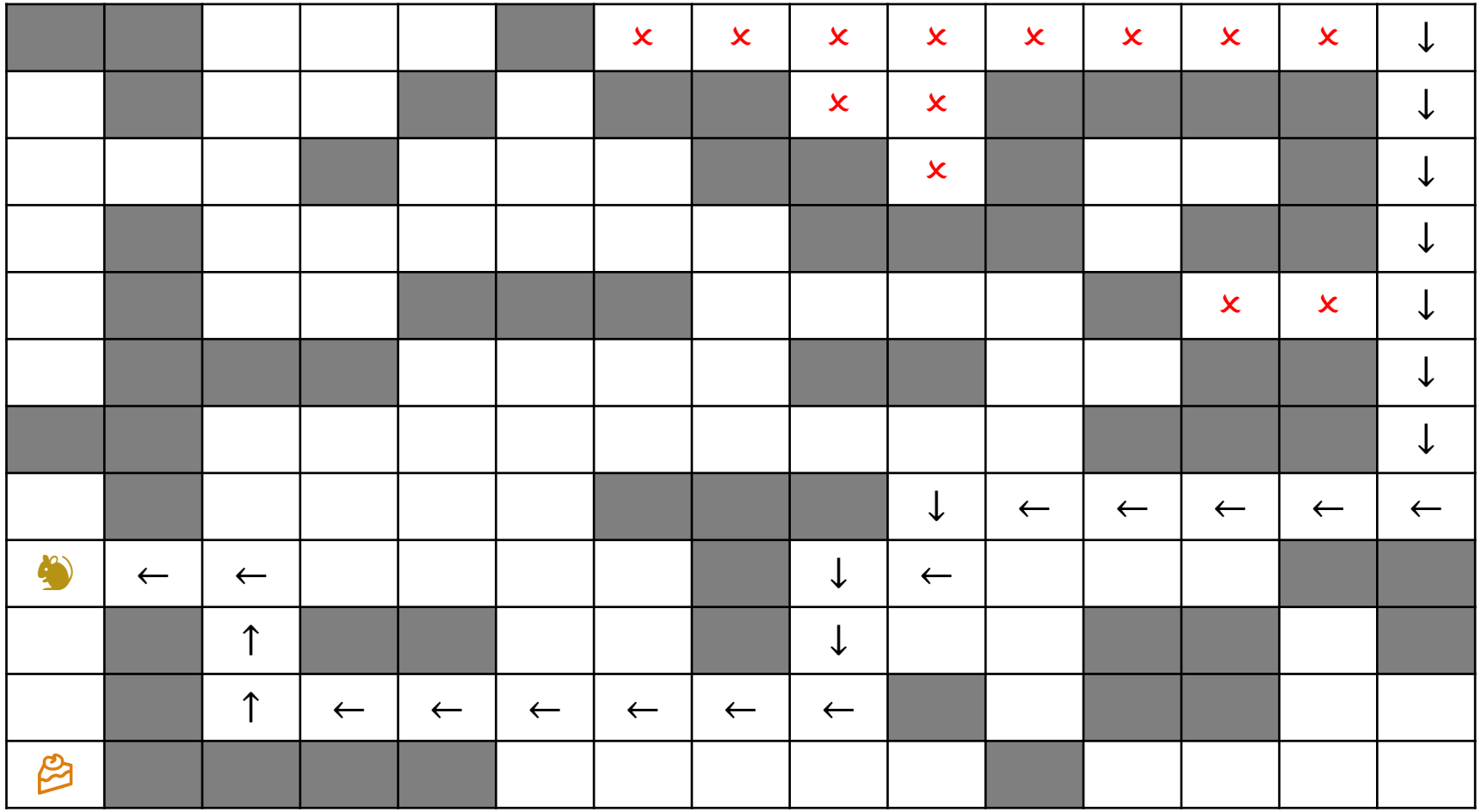
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up



Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up

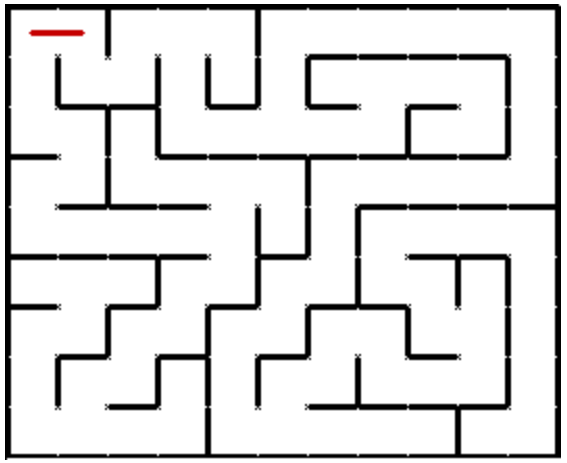


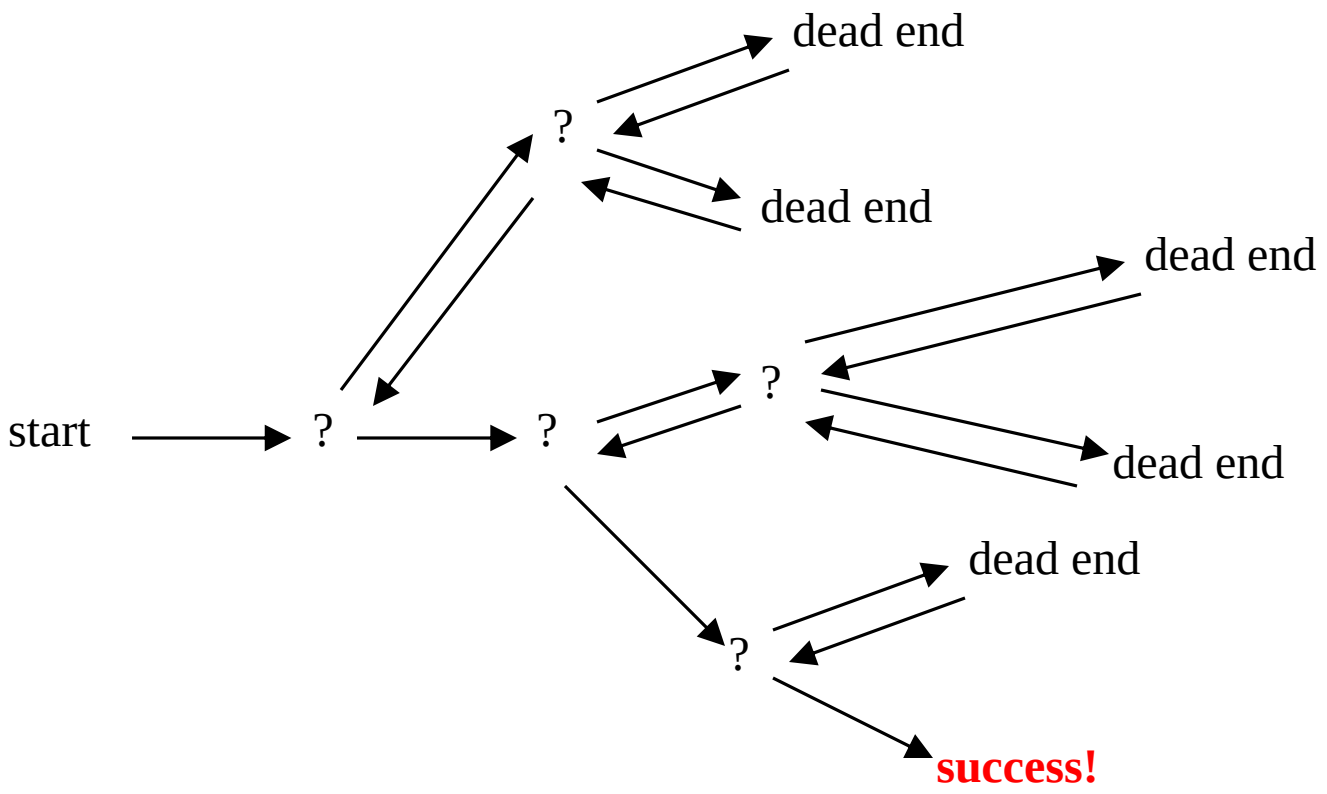
Rat In A Maze

Move Order	
←	left
↓	down
→	right
↑	up

						×	×	×	×	×	×	×	×	↓
								×	×					↓
									×					↓
														↓
												×	×	↓
														↓
														↓
									↓	←	←	←	←	←
↓	←	←						↓	←					
↓		↑						↓						
↓		↑	←	←	←	←	←	←						
■														

Backtracking

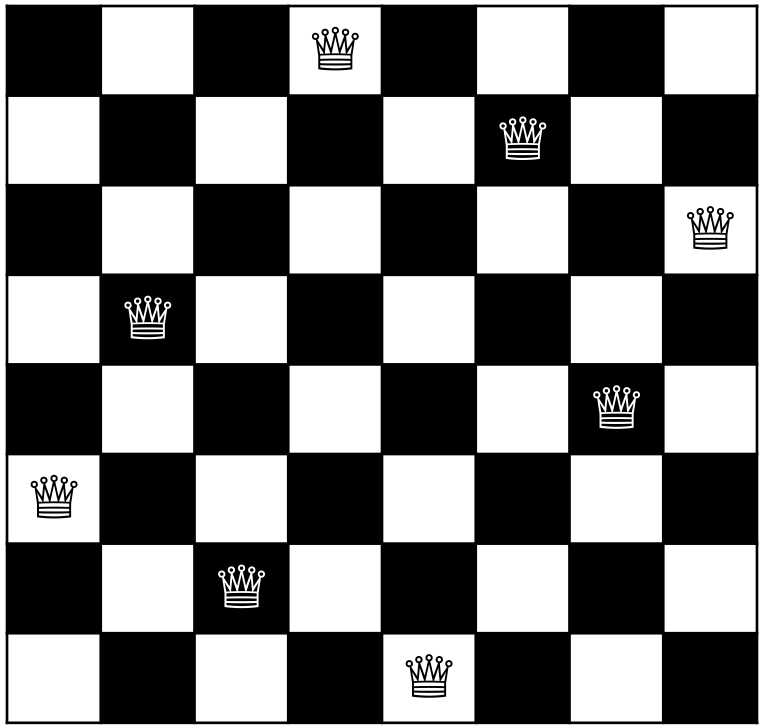




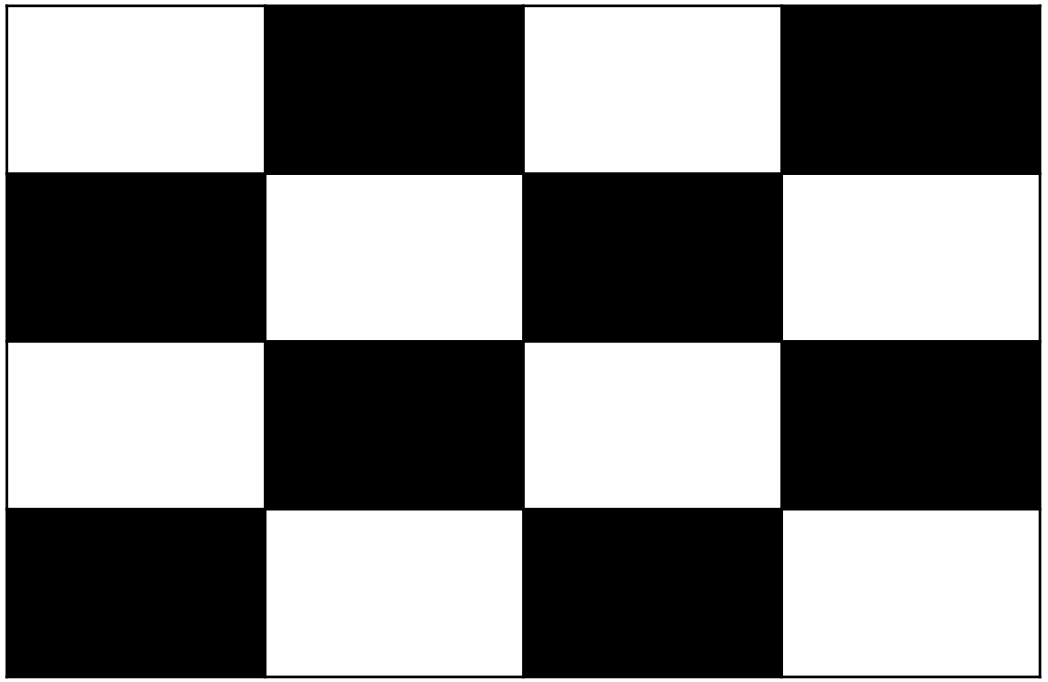
The n Queens Problem

We have an 8x8 chessboard, our job is to place eight queens on the chessboard, so that no two of them can attack. That is, no two of them are in the same row, column or diagonal.

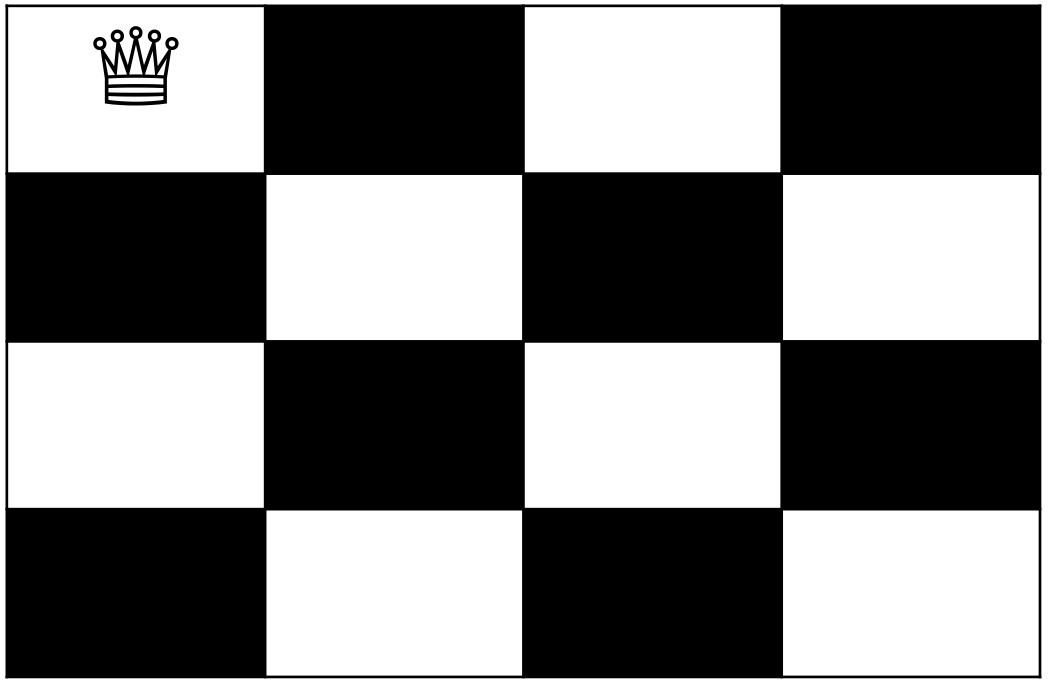
The generalized version of the 8 Queens problem is the n Queens problem, where n is a positive integer.



Backtracking to solve the 4-queens problem

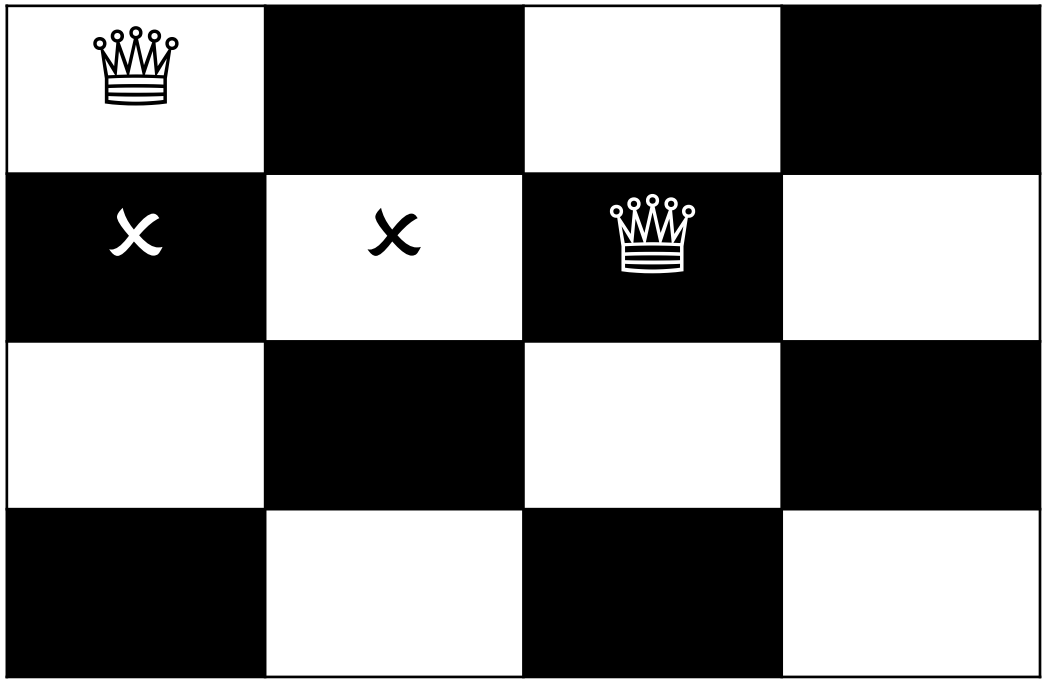


Backtracking to solve the 4-queens problem



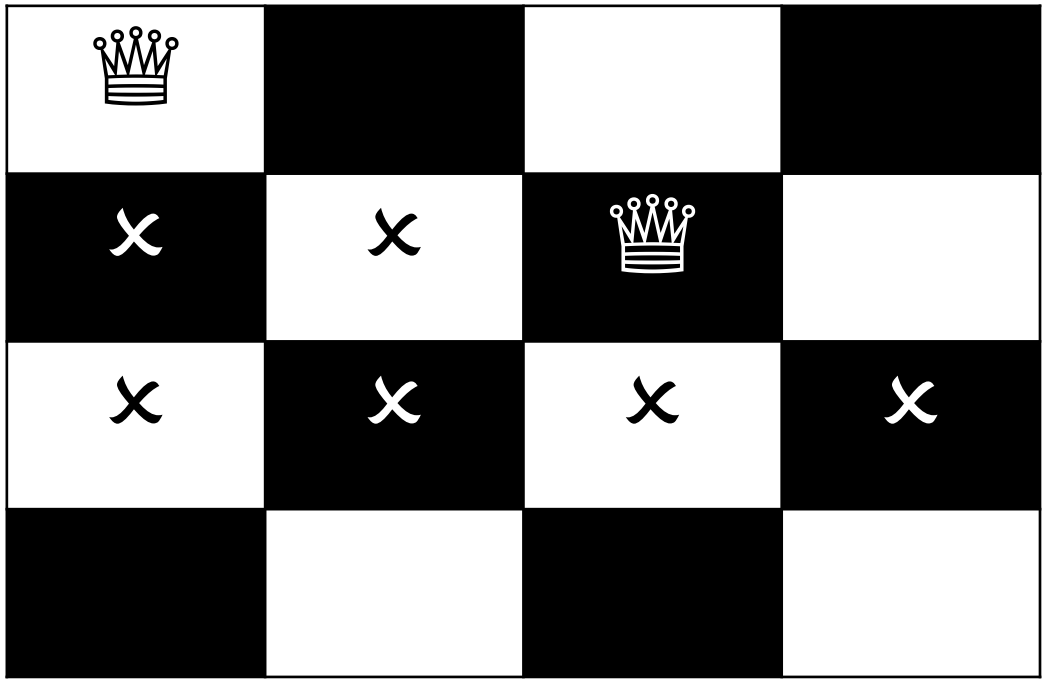
Place Queen1 at row1

Backtracking to solve the 4-queens problem



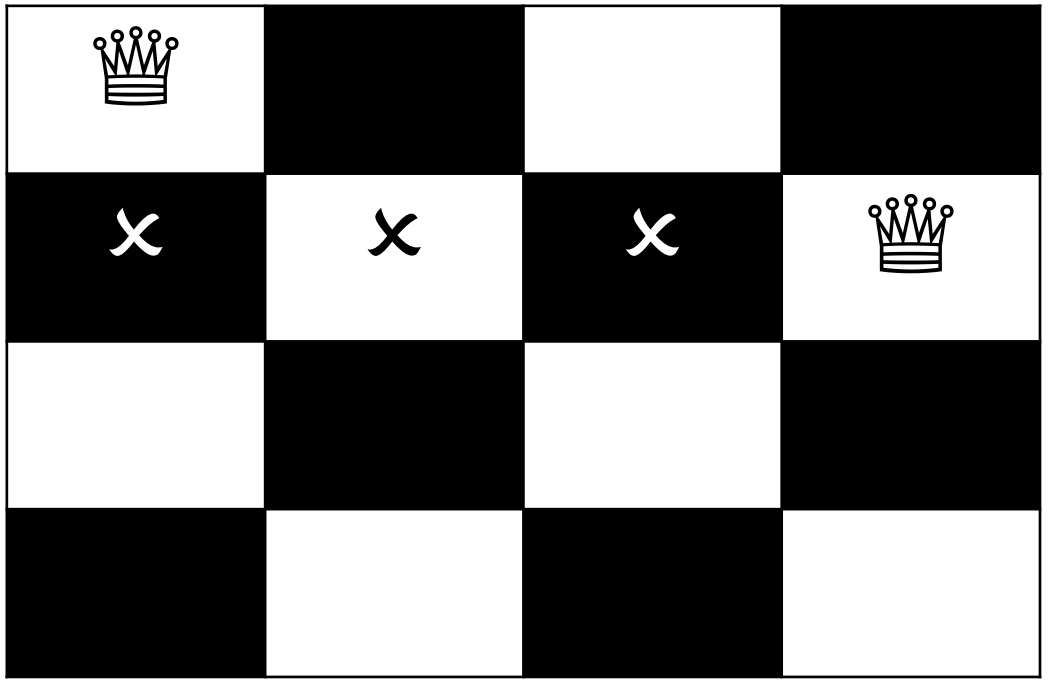
Place Queen2 at row2

Backtracking to solve the 4-queens problem



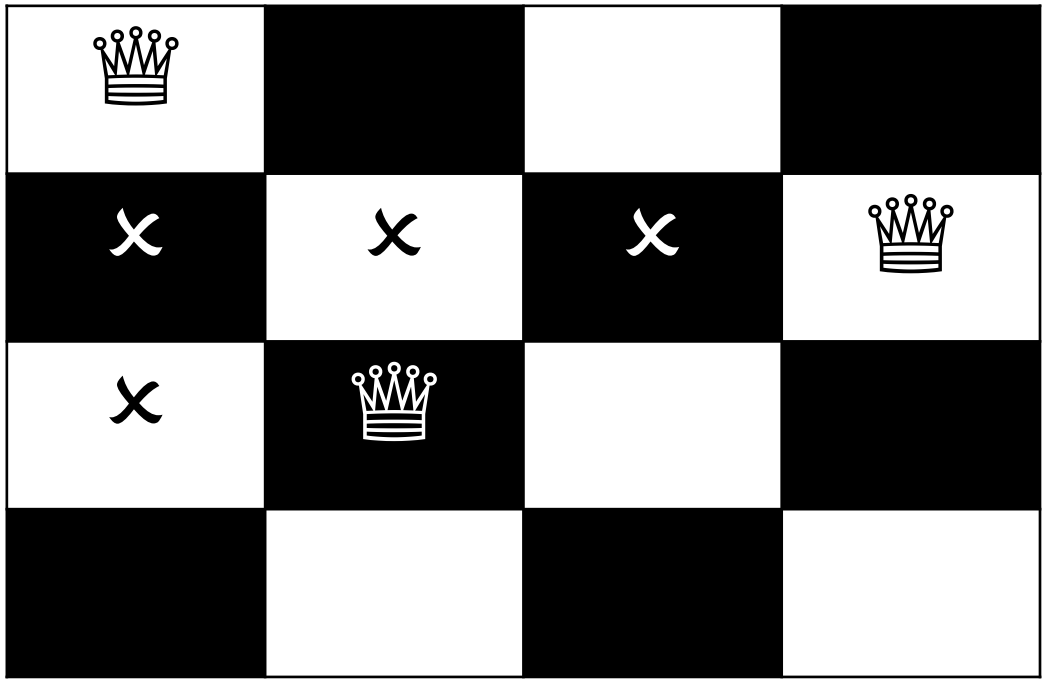
Dead End: can't place Queen3

Backtracking to solve the 4-queens problem



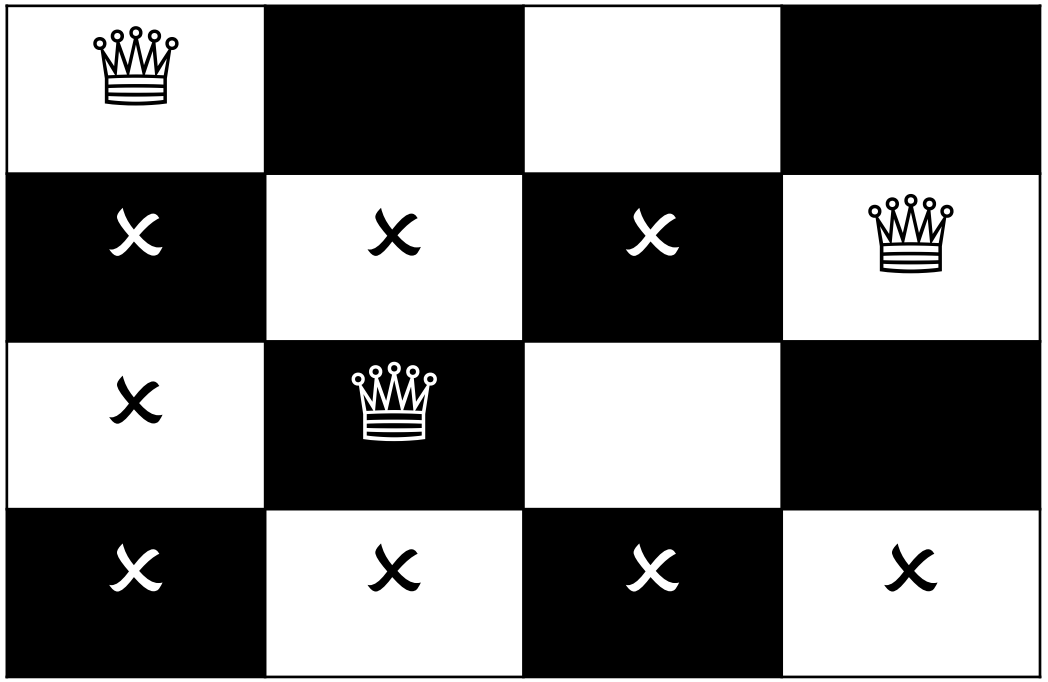
Backtrack and alter Queen2

Backtracking to solve the 4-queens problem



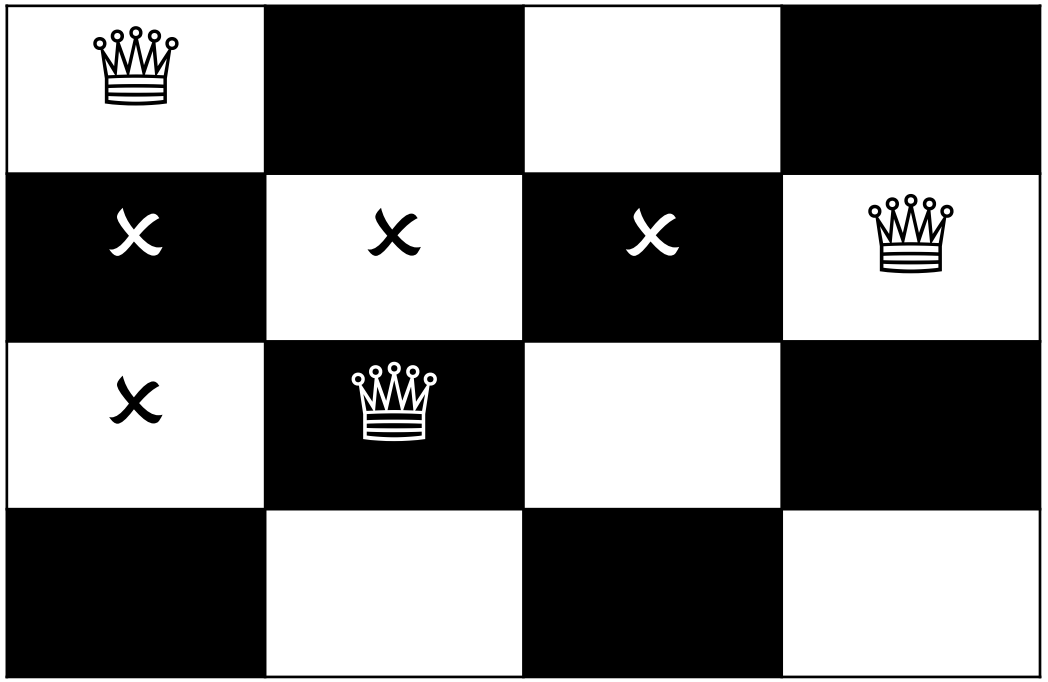
Place Queen3 at row3

Backtracking to solve the 4-queens problem



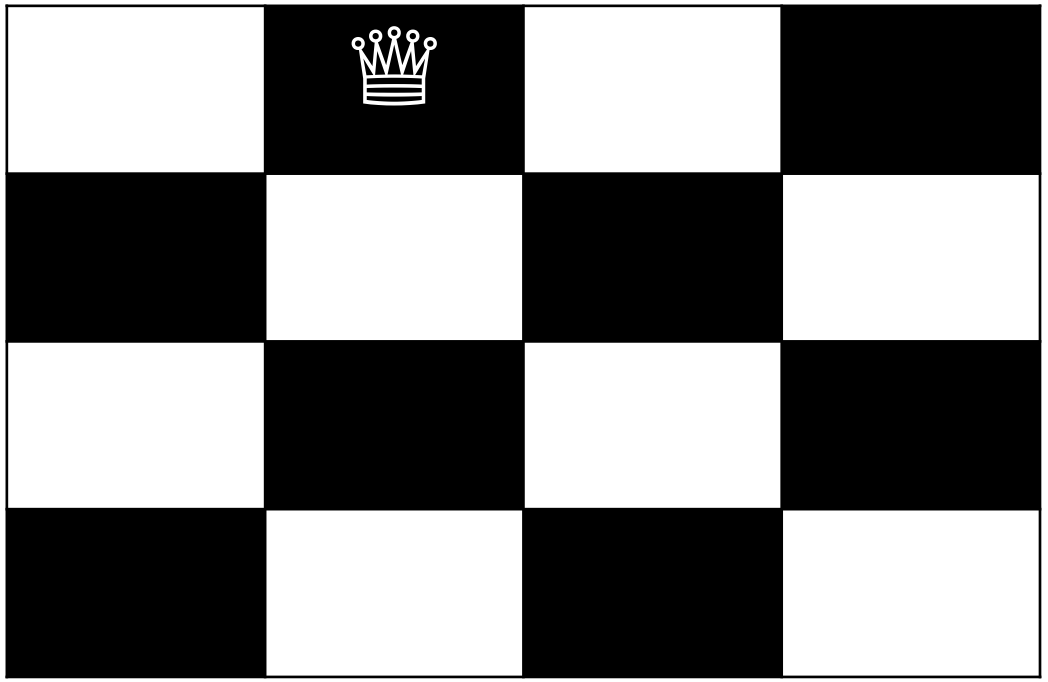
Dead End: can't place Queen4

Backtracking to solve the 4-queens problem



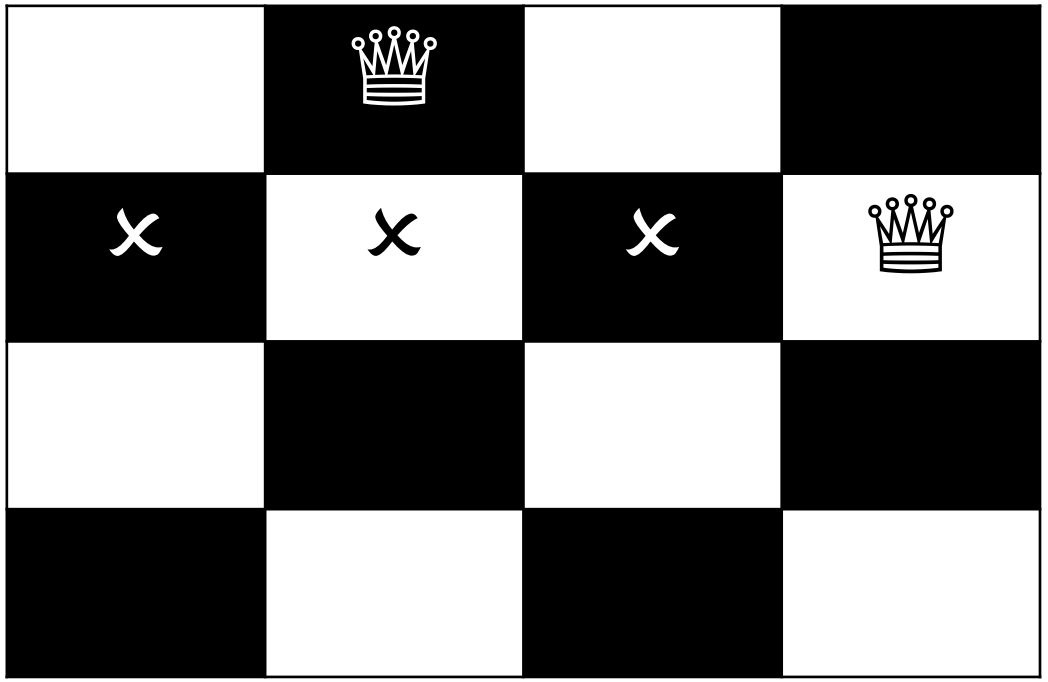
Can’t alter Queen3 or Queen2

Backtracking to solve the 4-queens problem



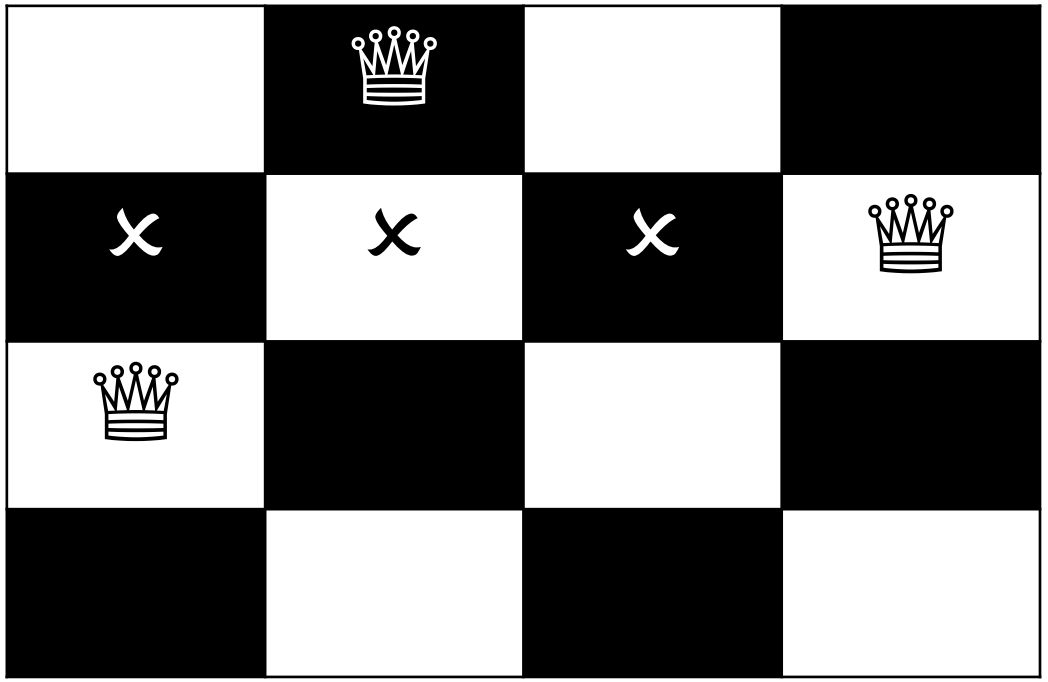
Backtrack and alter Queen1

Backtracking to solve the 4-queens problem



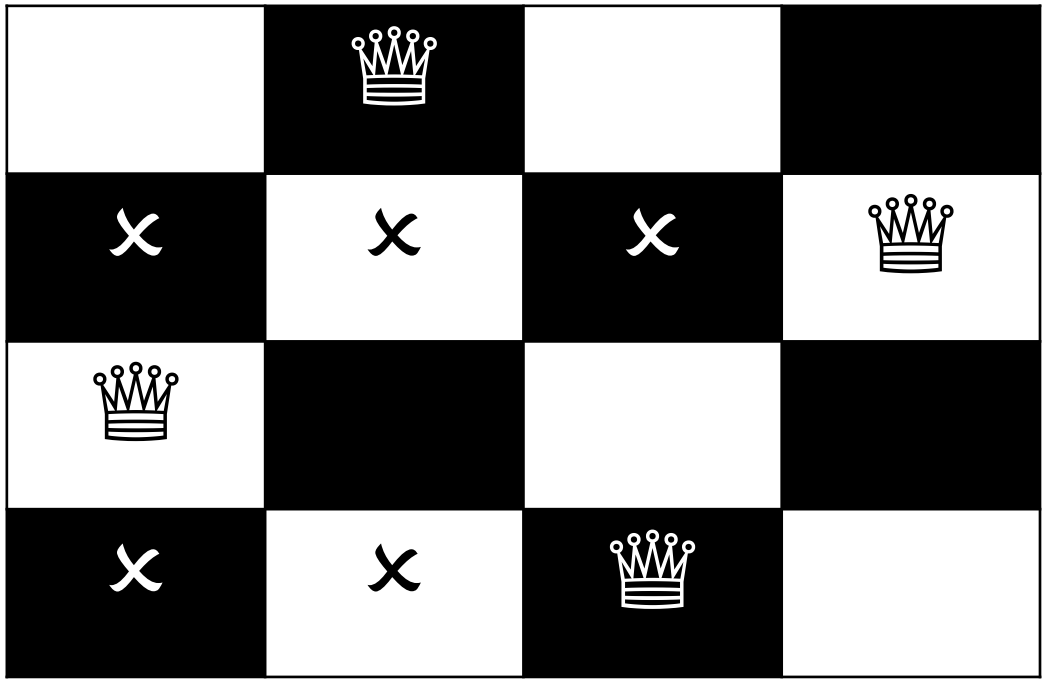
Place Queen2 at row2

Backtracking to solve the 4-queens problem



Place Queen3 at row3

Backtracking to solve the 4-queens problem



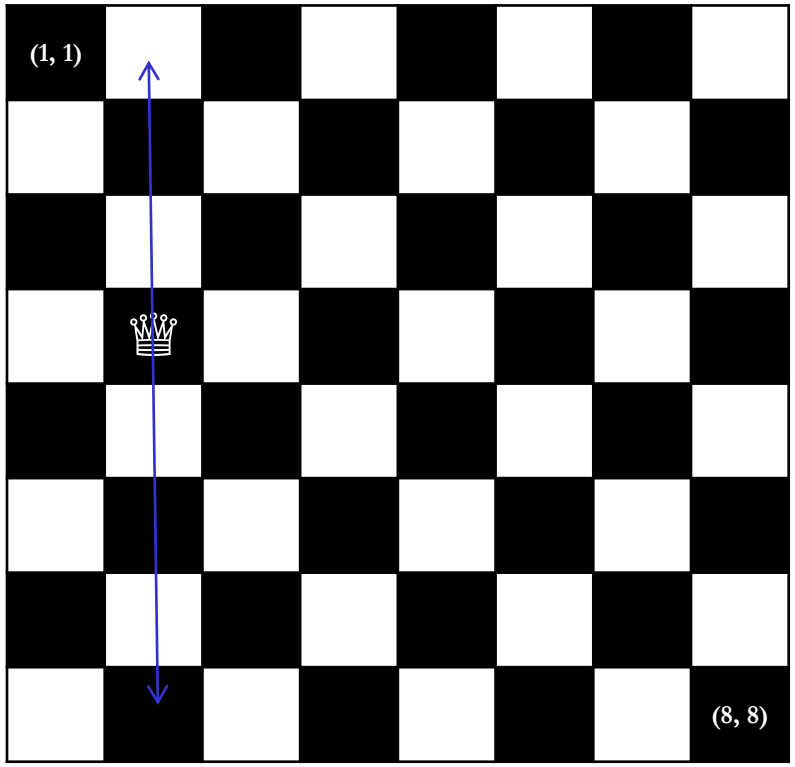
Place Queen4 at row4

Detecting a column attack

Think of the chessboard as a matrix of n rows and n columns.

Say, Queen1 is at position(r_1, c_1)
and, Queen2 is at position(r_2, c_2)

Under what condition does Queen1 attack Queen2 along the same column?



$$c_1 == c_2$$

Note that we need not check for the row equality because by our backtracking strategy we are only placing one queen per row. Hence there is no chance of two queens being in the same row.

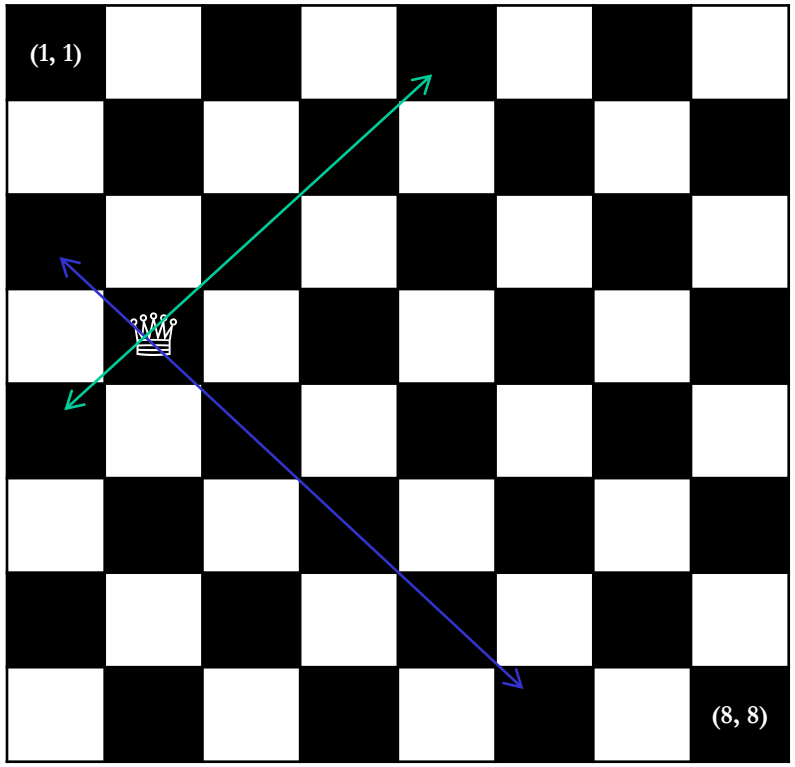
Detecting a diagonal attack

Now, say Queen1 is at (r_1, c_1) : e.g. (4, 2)

It can be attacked by Queen2 at (r_2, c_2) along two diagonals

a) Along the diagonal going from bottom-left to top-right
   - Possible positions of Queen2 are (5, 1), (3, 3), (2, 4), (1, 5).
   - In all the cases we observe that
   - $r_1 + c_1 == r_2 + c_2$
   - i.e. $\mathbf{r_1 - r_2 == c_2 - c_1}$ … (i)

b) Along the diagonal going from top-left to bottom-right
   - Possible positions of Queen2 are (3, 1), (5, 3), (6, 4), (7, 5), (8, 6).
   - In all the cases we observe that
   - $r_1 - c_1 == r_2 - c_2$
   - i.e. $\mathbf{r_1 - r_2 == c_1 - c_2}$ … (ii)



Combining (i) & (ii) we get the condition for diagonal attack as

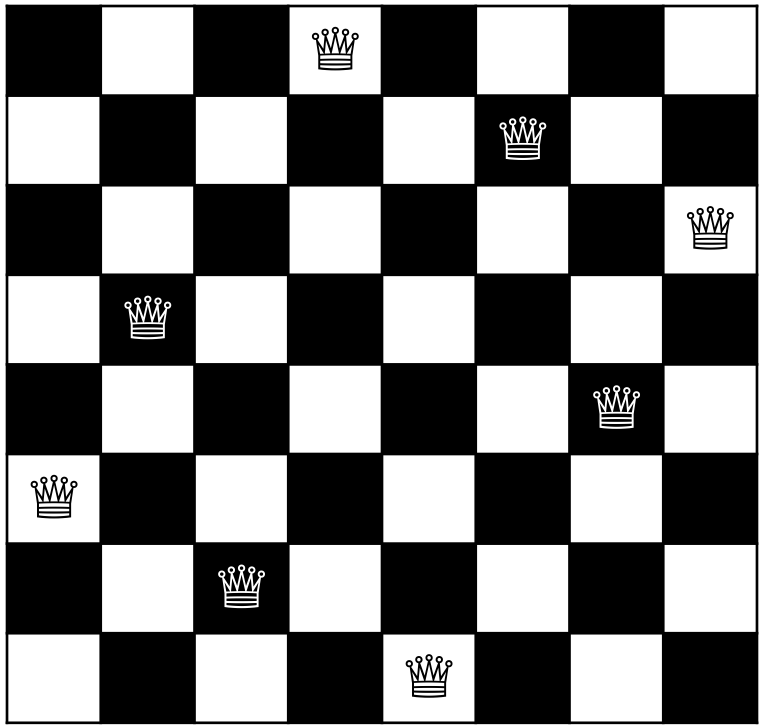
$$\mathbf{abs(r_1 - r_2) == abs(c_1 - c_2)}$$

Representing the solution

n Queens on an n x n chessboard can be represented by a one dimensional array q[1 … n], where

The index j represents the row of the Queen
The corresponding value q[j] represents the column of the Queen

e.g. q[4] = 2 means queen at row 4 is at column 2,
q[2] = 6 means queen at row 2 is at column 6.



This chessboard
Is represented by this array

	1	2	3	4	5	6	7	8
q	4	6	8	2	7	1	3	5

Placing a Queen

- Let's write an algorithm, which will place a Queen at position (r_2, c_2) in the chessboard.

`place(r2, c2)`

- Our algorithm should return true, if the Queen can be placed at (r_2, c_2) and false otherwise
- Now, a Queen is being placed at (r_2, c_2) :- what information does it give us?
- Hint: recall the backtracking strategy.
- Remember how we solved the 4-Queens problem? We tried to place the first Queen in row1, then the second Queen in row2 and then the third Queen at row3 and so on…
- In other words, we only tried to place a Queen in a row if and only if we were able to place Queens in all the previous rows.
- So if we are trying to place a Queen at (r_2, c_2), then it must mean that we have successfully placed Queens in all previous rows, i.e. 1, 2, …, $r_2 - 1$ already.
- Then our job is to simply check if any of those (previously placed $r_2 - 1$) Queens attack the Queen to be placed at (r_2, c_2) along a column or along a diagonal

Placing a Queen

```
place(r2, c2)
      // for all queens in the previous rows
      //      if (the new Queen at (r2, c2) attacks along a column or a diagonal)
      //            This Queen can't be placed at (r2, c2)
      // otherwise (r2, c2) is a good position to place the Queen
   1. for (r1 = 1 to r2 - 1)
            //  column of the Queen at row r1 is
   2.       c1 = q[r1]
   3.       if (c1 == c2 OR abs(r1 - r2) == abs(c1 - c2))
   4.          return false
      // since we are out of the for loop this must be a valid place
   5. return true
```

The n-Queens Algorithm

```
place(r2, c2)
1. for (r1 = 1 to r2 - 1)
2.       c1 = q[r1]
3.       if (c1 == c2 OR abs(r1 - r2) == abs(c1 - c2))
4.           return false
5. return true
```

```
nQueens(r)
1. for (c = 1 to n)
2.       if (place(r, c))
3.           q[r] = c
4.           if (r == n) displayQueens()
5.           else nQueens(r + 1)
```

Happy Backtracking

Thank You!