AI: Assignment 1

Due on March 9, 2021

l.lymarenko@innopolis.university

BS19-02

Lev Lymarenko

PEAS

- Performance Measure - the number of steps the agent should take to reach the house
- Environment - 9*9 (or less) square lattice
- Actuator - move horizontally, vertically and diagonally
- Sensor - sensor to get home location, ability to perceive COVID

Description of the environment

- *Partially Observable* since we can only perceive COVID when we are nearby
- *Deterministic* - world is fully determined by the rules, no random
- *Single-agent* - there is no any other agents in the maze
- *Static* - the environment does not change while an agent is acting
- *Discrete* - there are a limited number of states of the environment
- *Sequential* - since current move decision has consequences on future decisions
- *Known* - designer of the agent have full knowledge of the rules of the environment

Algorithms description

I implemented two algorithms: backtracking and A*

Backtracking

Backtracking solves problem recursively, considering every possible combination in order find path to home. In order to increase performance of the algorithm, it is reasonable to introduce some optimization. For instance, on a recursion step, we can check validness of next movement, introduce maximum length of the path and do not enter already visited cells.

A*

As the second algorithm, I decided to choose A*. Basically, the algorithm uses heuristic to find path. So, for every point we assign a number that shows a certain heuristic value. The logic for selecting a number may vary. In my algorithm I use the number of cells from point to the home:

```
heuristic(Point, Result) :-
    Point = Cell-_,
    home(Home),
    mooreDistance(Cell, Home, Result).
mooreDistance(X1-Y1, X2-Y2, D) :-
    D is max(abs(X1 - X2), abs(Y1 - Y2)).
```

However, we should consider mask/doctor effect, so we need some modification of A*. In order to do it, I decided to introduce the meaning of covid-flag. The thing is that every cell in a maze actually represented in the algorithm twice: a cell that was entered by actor without a mask/doctor effect and entered with a mask/doctor effect. So, technically, maze is not 2D array anymore but 3D with only two options for 3rd value: zero (no mask/doctor) and one (have mask/doctor).

Comparison or two algorithms

Actually, there is no difference between two scenarios of perceiving COVID in my algorithms:

- **Backtracking** takes into account only the adjacent cells and completely ignoring information about COVID , except that it removes infected cells adjacent cells. So, it can not look ahead to the COVID
- For **A***, it is possible to consider forbidden cells from the COVID in heuristic function. However it is completely not clear how to add this information to the function. I tried

different ways of adding vision of COVID in heuristic function, and it did not affect the time anyhow.

Consequently, since I found no difference between those two scenarios, I will discuss only **Backtracking (variant 1) compared to 2nd algorithm (variant 1)**

Legend

- . - Free cell
- A - Initial actor cell
- M - Mask cell
- D - Doctor cell
- C - Covid cell
- H - Home cell
- All lowercase letters is visited cells

Maps

In order to compare performance of two algorithms, let's construct 3 maps with different difficulty levels:

```
 EASY MAP              NORMAL MAP                  HARD MAP
  0 1 2 3           0 1 2 3 4 5 6 7 8           0 1 2 3 4 5 6 7 8
0 A . . M         0 A H . . M D . . .         0 A . . . . . M . D
1 . . . .         1 . . C . . . . . .         1 . . . . . . C . .
2 D . . C         2 . . . . . . . . .         2 . . . . . . . . .
3 . . C H         3 . . . . . . . . .         3 . . . . . . . . .
                  4 . . C . . . . . .         4 . . . . . . . . .
                  5 . . . . . . . . .         5 . . . . . . . . .
                  6 . . . . . . . . .         6 . . . . . . . . .
                  7 . . . . . . . . .         7 . C . . . . . . .
                  8 . . . . . . . . .         8 H . . . . . . . .
```

And let's run all 3 algorithms 20 times on all 3 maps:

Map\Algorithm	Backtracking		A*	
	# of steps	average time	# of steps	average time
Easy map	6	0.0342 s	6	0.0204 s
Normal map	17	78.3215 s	17	0.041 s
Hard map	-	too long	19	0.0857 s

P-value test

Since there is no way to calculate mean time for backtracking in Hard map in reasonable time, let's calculate t-value and p-value for first two maps. (You can find the data in **data.txt** attached file.) Let's choose null hypothesis as "Two algorithms are the same and have the same distribution of time-consuming" and significance level as 0.05
Using `https://www.socscistatistics.com/tests/studenttest/default2.aspx` we get

$$t_{value} = -6.24526, \text{ so } p_{value} < 0.05$$

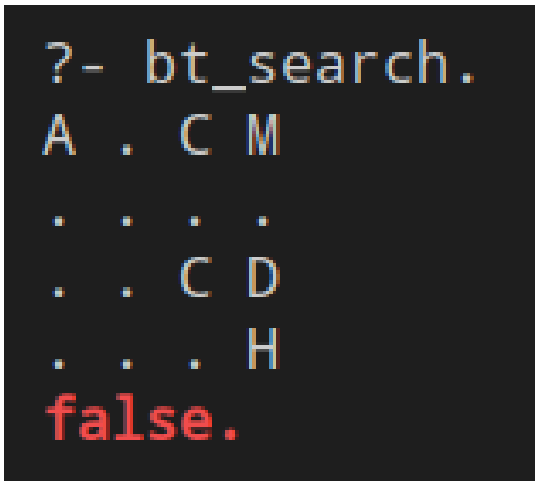
Therefore, we **reject** the null hypothesis.

Conclusion

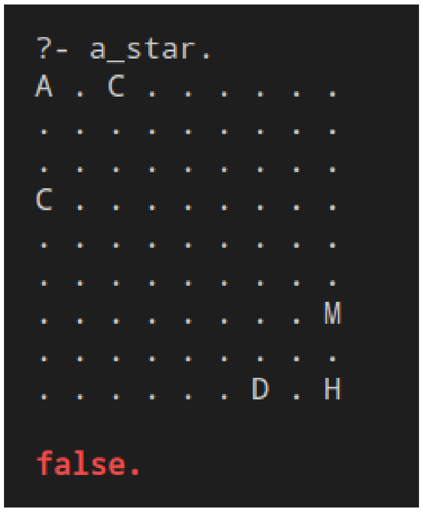
So, as you can see, **backtracking** is much worse than **A*** in terms of time-consuming. Actually, A* can find path in rather big maze in a fairly reasonable time (500x500 in 16 seconds), while backtracking will try to find all possible combinations and will not be able to meet a reasonable amount of time.
However, it is not fair to say that A* is better than backtracking. Prolog is not intended for implementing algorithms such as A* or Dijkstra. To achieve this goal, you have to use a lot of workarounds, creating complex data structures and turning each loop into a recursion. Thus, the implementation of a simple algorithm turns into a complex challenge. While backtracking is built into the prolog and its implementation takes several lines. Therefore, you have to spent more time to create A*, rather than backtracking.

Impossible to solve maps

It is possible to solve any map without COVID , so, the only option for us to create imposible map is block the path to home/mask/doctor with COVID
Examples:



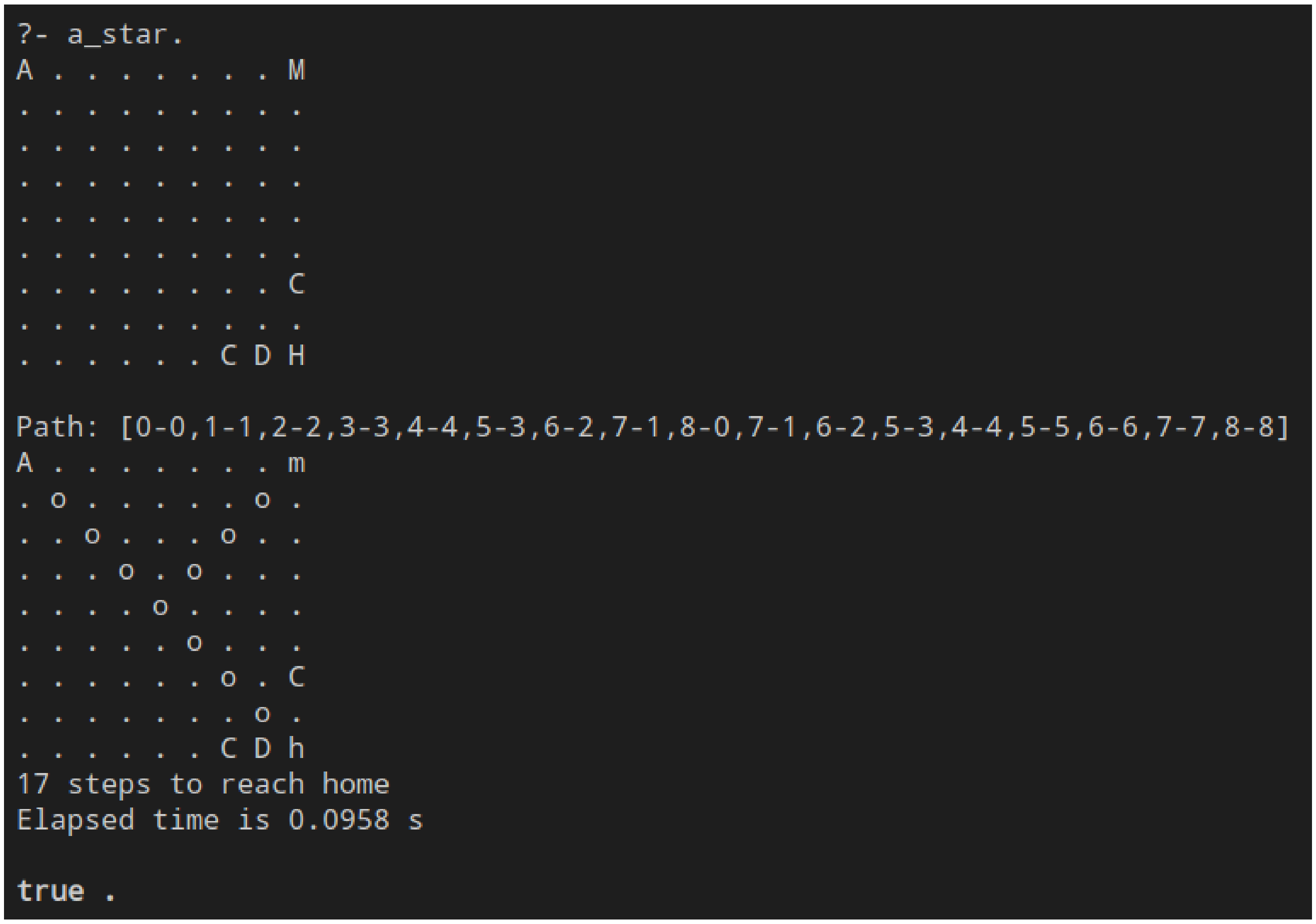
```
?- bt_search.
A . C M
. . . .
. . C D
. . . H
false.
```



```
?- a_star.
A . C . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
C . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . M
. . . . . . . . . .
. . . . . . . D . H

false.
```

Interesting outcome/map



```
?- a_star.
A . . . . . . . M
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . C
. . . . . . . . .
. . . . . . C D H

Path: [0-0,1-1,2-2,3-3,4-4,5-3,6-2,7-1,8-0,7-1,6-2,5-3,4-4,5-5,6-6,7-7,8-8]
A . . . . . . . m
. o . . . . . o .
. . o . . . o . .
. . . o . o . . .
. . . . o . . . .
. . . . . o . . .
. . . . . . o . C
. . . . . . . o .
. . . . . . C D h
17 steps to reach home
Elapsed time is 0.0958 s

true .
```

When I built this map, I expected that the solution would be a path around the perimeter of the map, however, due to the allowed diagonal movements, allowed movement in visited cells and the heuristic that the path to the home is more profitable, we get this solution!

Code

You can find the code and README file on github: `https://github.com/sevenzing/AI_assignment_1`
Or in attached files of submission