Artificial Intelligence
Assignment 1

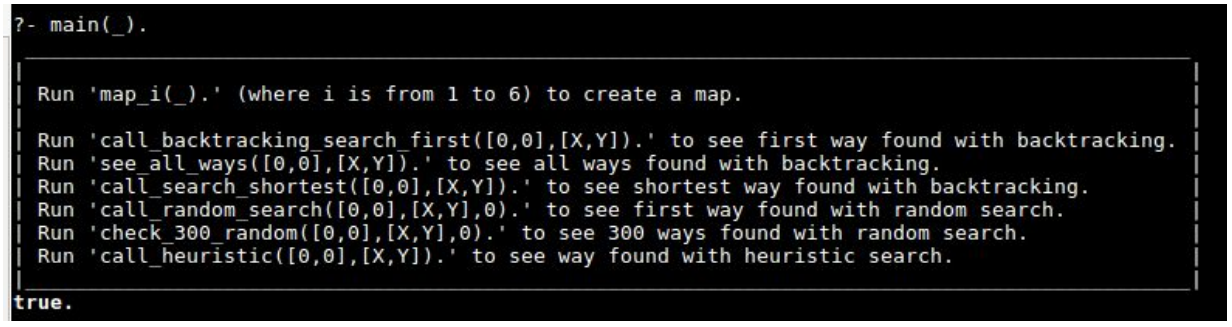
Alisa Martyanova BS18-05

Table of content

How to run the code

To run the code you need to open terminal and launch ***swipl Alisa_Martyanova.pl*.**

Then first of all you need to run ***?- main(_).*** It will print the list of all possible commands that you can run to see the work of different algorithms.



Then you need to run ***?-map_i(_).*** (where **i** is number from 1 to 6). This function will create a field with orcs, humans and touchdown.

Then you can run one of the following functions:

call_backtracking_search_first([0,0],[X,Y]). - to see first way found with backtracking.

see_all_ways([0,0],[X,Y]). - to see all ways found with backtracking.

call_search_shortest([0,0],[X,Y]). - to see shortest way found with backtracking.

call_random_search([0,0],[X,Y],0). - to see first way found with random search.

check_300_random([0,0],[X,Y],0). - to see 300 ways found with random search.

call_heuristic([0,0],[X,Y]). - to see way found with heuristic search.

(after each execution, it is better to run ***halt.*** and recompile)

[0,0] is starting point
[X,Y] will be finish point
0 is for Cycle, need for random search.

How to changing parameters

```
33  map_2(_):-
34      retractall(touchdown(X,U)),
35      retractall(wall_1(Y)),
36      retractall(orc(W,E)),
37      retractall(human(Z,S)),
38
39      assert(touchdown(3,2)),
40
41      assert(wall_1(4)),
42
43      assert(orc(3,1)),
44      assert(orc(2,3)),
45      assert(orc(2,2)),
46
47      assert(human(0,0)),
48      assert(human(0,3)).
```

There are six map functions, which creates six different fields with different sizes and distributions of characters.

map_2(_) - example of one of them.

Lines (34) - (37) - clear old distribution.

Line (39) - coordinates of touchdown.

Line (41) - upper wall.

Lines (43) - (45) - coordinates of orcs.

Lines (47) - (48) - coordinates of humans.

The field described above will look like this:

4					
3	H		O		
2			O	T	
1				O	
0	H				
	0	1	2	3	4

Basic functions

```
21
22  is_safe(X,Y):-
23      \+orc(X,Y).
24
25  win([X,Y]):-
26      touchdown([X,Y]).
27
28
```

is_safe(X,Y) - checks that at point [X,Y] there is no orc

win([X,Y]) - check if a point [X,Y] is touchdown

```
146 left([X,Y], [X2,Y2]):-
147     \+wall_2(X),
148     X2 is X-1,
149     Y2 is Y,
150     is_safe(X2,Y2).
151
```

left([X,Y], [X2,Y2]) - [X,Y] is starting point, [X2,Y2] is ending point. ***\+ wall_2(X)*** - it checks that player is not near left wall, ***is_safe(X2,Y2)*** - checks that yard where we want to move is without orc. If both conditions hold, player makes move.

Functions for **right(), forward()** and **back()** are done in the same way.

```
105
106 pass_right([X,Y], [X2,Y2]):-
107     right([X,Y], [X3,Y3]) -> (human(X3, Y3) -> (X2 is X3, Y2 is Y3); pass_right([X3, Y3], [X2,Y2])); false.
108
109 pass_left([X,Y], [X2,Y2]):-
110     left([X,Y], [X3,Y3]) -> (human(X3, Y3) -> (X2 is X3, Y2 is Y3); pass_left([X3, Y3], [X2,Y2])); false.
111
```

pass_right([X,Y], [X2,Y2]) - also make move from [X,Y] to [X2,Y2]. It calls function **right(),** if on new position there is a human, he catch the ball and these coordinates returns into [X2,Y2], if there is no human, no orc and no wall, function continues with recursive call. In such a way, if a ball moved till the wall and found nobody, no coordinates are returned and no move is done.
Functions ***pass_left(), pass_back()*** and ***pass_forward()*** are done in the same way.

```
117
118 pass_right_up_diag([X,Y], [X2,Y2]):-
119     (\+wall_1(X),
120     \+wall_1(Y),
121     X3 is X + 1,
122     Y3 is Y + 1,
123     is_safe(X3,Y3)) -> (human(X3, Y3) -> (X2 is X3, Y2 is Y3); pass_right_up_diag([X3, Y3], [X2,Y2])); false.
124
```

pass_right_up_diag([X,Y], [X2,Y2]) - make diagonal moves, so here it also needs to check to walls. If ball is placed in the right upper corner it cannot move by right upper diagonal. The rest implementation is as in previous functions.
pass_right_down_diag(), pass_left_up_diag() and pass_left_down_diag() works in the same way.

```
195
196 move([X,Y], [X2,Y2]):-
197     (nb_getval(pass_possible, T), T = 0) -> (pass_back([X,Y], [X2,Y2]), retractall(pass_possible), nb_setval(pass_possible, 1));false.
198
199
200 move([X,Y], [X2,Y2]):-
201     left([X,Y], [X2,Y2]).
202
```

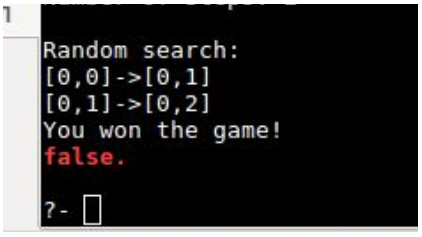
move([X,Y], [X2,Y2]) - choose and make one of the 12 possible moves. For pass it checks can pass be made or not (cause only one pass can be made per a round).

Part 1

Random search

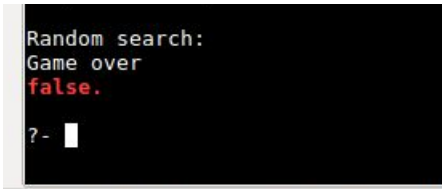
```
65
66  random_search(Start, Finish, Cycle):-
67      (Cycle \= 100 -> true; (writeln("Number of tries is over"), false)),
68
69      Cycle2 is Cycle + 1,
70      random(0,12,R),
71
72      (R = 2 -> (left(Start,Finish) -> (write(Start), write("->"), writeln(Finish)); (writeln("Game over"),false));
73      (R = 0 -> (right(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
74      (R = 1 -> (forward(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
75      (R = 3 -> (back(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
76          (R = 4 -> (pass_right(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
77          (R = 5 -> (pass_left(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
78          (R = 6 -> (pass_back(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
79          (R = 7 -> (pass_forward(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
80              (R = 8 -> (pass_right_up_diag(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
81              (R = 9 -> (pass_right_down_diag(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
82              (R = 10 -> (pass_left_up_diag(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false));
83              (R = 11 -> (pass_left_down_diag(Start,Finish)-> (write(Start), write("->"),writeln(Finish)); (writeln("Game over"),false))))))
84      (win(Finish) -> (writeln("You won the game!"), false);
85          random_search(Finish, [X2,Y2],Cycle2)).
86
```

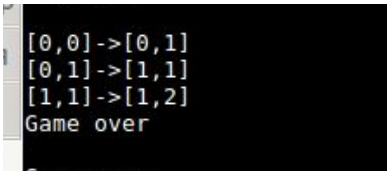
The idea is that it takes random number from 0 to 11 which are correspondent to moves (left, right, diagonal, etc.) and makes this move. If a player moves into touchdown point, he wins the game and algorithm shows his path.



(for touchdown in [0,2])

If player moves into yard with orc or outside the field - the game is over (as moves are random, the algorithm does not prevent such cases). He can lose either from the first move, or does more and lose at some point in the middle.





If player did 100 moves and did not find the touchdown the game is also over.

Backtracking search

```
50
51  backtracking(Start,Finish):-
52      nb_setval(pass_possible, 0),
53      dpth([Start],Finish,Way),
54      find_shortest(Way).
55
56  see_all_ways(Start,Finish):-
57      dpth([Start],Finish,Way),
58      show_answer(Way),
59      nl,
60      fail.
61
62  backtracking_search_first(Start,Finish):-
63      backtracking(Start,Finish) -> true.
64
65
66  backtracking_search_shortest(Start,Finish):-
67      backtracking(Start,Finish),
68      fail.
69
70  find_shortest(Way):-
71          length(Way, L),
72          L1 is L-1,
73          nb_getval(array_len,L2),
74          L1<L2 -> (nb_setval(array, Way), nb_setval(array_len,L1)); true.
75
76  prolong([Temp|Tail],[New,Temp|Tail]):-
77      move(Temp,New),
78      \+member(New,[Temp|Tail]).
79
80  dpth([Finish|Tail],Finish,[Finish|Tail]):-
81      win(Finish).
82
83  dpth(TempWay,Finish,Way):-
84      prolong(TempWay,NewWay),
85      dpth(NewWay,Finish,Way).
86
87  show_answer([_]):-!.
88
89  show_answer([A,B|Tail]):-
90      show_answer([B|Tail]),
91      nl,
92      write(B),
93      write(" -> "),
94      write(A).
```

The idea of **backtracking search** is that it is trying to find the path going in depth. If it moves on the yard with wall or orc, it goes back on one step and tries to find another way. Search starts in function ***backtracking(Start, Finish)***. It calls functions **dpth()** and **prolong()** recursively, which in turn go in depth and remember visited yards.
Function ***backtracking_search_first(Start, Finish)*** returns first found way in backtracking search, which is not necessarily the shortest.

```
Backtracking first found path:

[0,0] -> [1,1]
[1,1] -> [2,1]
[2,1] -> [3,1]
[3,1] -> [3,2]
[3,2] -> [2,2]
[2,2] -> [2,3]
[2,3] -> [3,3]
Number of steps: 7

Backtracking shortest path:

[0,0] -> [1,1]
[1,1] -> [2,1]
[2,1] -> [3,1]
[3,1] -> [3,2]
[3,2] -> [3,3]
Number of steps: 5
```

Function ***see_all_ways(Start, Finish)*** shows all the found ways using backtracking.

Function ***backtracking_search_shortest(Start, Finish)*** goes though the whole tree of solutions of backtracking and every iteration remember only the shortest path.

Backtracking will definitely find a path (if it exists).

(example for touchdown at [3,3])

Heuristic search

```
404
405  bdth([[Finish|Tail]|_],Finish,[Finish|Tail]):-
406      win(Finish).
407
408  bdth([TempWay|OtherWays],Finish,Way):-
409      findall(W,prolong(TempWay,W),Ways),
410      append(OtherWays,Ways,NewWays),
411      bdth(NewWays,Finish,Way).
412
413
414  search_bdth(Start,Finish):-
415      nb_setval(pass_possible, 0),
416      bdth([[Start]],Finish,Way),
417      length(Way, L1),
418      nb_setval(arr_h_l,L1),
419      nb_setval(arr_h,Way).
420
421  heuristic(Start, Finish):-
422      search_bdth(Start, Finish) -> true.
423
424
425  call_heuristic(Start,Finish):-
426      nb_setval(array_len,1000000),
427      writeln("Heuristic path:"),
428      statistics(runtime,[Start_t|_]), /*to count time*/
429      (\+heuristic(Start,Finish) -> statistics(runtime,[Stop|_]),Runtime is Stop - Start_t,writeln("No possible way"),write(Runtime), writeln("ms"),nl;
430          (statistics(runtime,[Stop|_]),
431          Runtime is Stop - Start_t,
432          nb_getval(arr_h, T), show_answer(T), nl, write("Number of steps: "), nb_getval(arr_h_l, L), writeln(L),
433                write(Runtime), writeln("ms"),
434          nl)).
435
```

The idea is that the algorithm checks neighbour yards, chose one and at further steps it will not consider neighbours of previously visited yards, which decreases the number of steps to find touchdown.

```
?- call_heuristic([0,0],[X,Y]).
Heuristic path:

[0,0] -> [0,3]
[0,3] -> [1,3]
[1,3] -> [1,4]
[1,4] -> [2,4]
[2,4] -> [3,4]
[3,4] -> [3,3]
[3,3] -> [3,2]
Number of steps: 7
18ms
```

Testing

Example of map and comparison of algorithms

```
?- call_backtracking_search_first([0,0],[X,Y]).
Backtracking first found path:

[0,0] -> [1,1]
[1,1] -> [2,1]
[2,1] -> [3,1]
[3,1] -> [4,1]
[4,1] -> [5,1]
[5,1] -> [6,1]
[6,1] -> [7,1]
[7,1] -> [8,1]
[8,1] -> [9,1]
[9,1] -> [10,1]
[10,1] -> [11,1]
[11,1] -> [11,2]
[11,2] -> [10,2]
[10,2] -> [9,2]
[9,2] -> [8,2]
[8,2] -> [7,2]
[7,2] -> [6,2]
[6,2] -> [5,2]
[5,2] -> [4,2]
[4,2] -> [3,2]
[3,2] -> [2,2]
[2,2] -> [1,2]
[1,2] -> [0,2]
[0,2] -> [0,3]
[0,3] -> [0,4]
[0,4] -> [1,4]
[1,4] -> [2,4]
[2,4] -> [3,4]
[3,4] -> [4,4]
[4,4] -> [5,4]
[5,4] -> [6,4]
[6,4] -> [7,4]
[7,4] -> [8,4]
[8,4] -> [9,4]
[9,4] -> [10,4]
[10,4] -> [11,4]
[11,4] -> [11,5]
[11,5] -> [10,5]
[10,5] -> [9,5]
[9,5] -> [8,5]
[8,5] -> [7,5]
[7,5] -> [6,5]
[6,5] -> [5,5]
[5,5] -> [5,6]
[5,6] -> [4,6]
[4,6] -> [3,6]
[3,6] -> [2,6]
[2,6] -> [2,7]
[2,7] -> [3,7]
Number of steps: 49
35ms

true.
```

11												
10												
9											O	
8												
7				T								O
6		O										
5		O			O							
4												
3		O										
2												
1	O	H										
0	H											
	0	1	2	3	4	5	6	7	8	9	10	11

```
?- call_random_search([0,0],[X,Y],0).
Random search:
Game over
42ms
true.
```

```
?- call_heuristic([0,0],[X,Y]).
Heuristic path:

[0,0] -> [1,1]
[1,1] -> [2,1]
[2,1] -> [3,1]
[3,1] -> [3,2]
[3,2] -> [3,3]
[3,3] -> [3,4]
[3,4] -> [3,5]
[3,5] -> [3,6]
[3,6] -> [3,7]
Number of steps: 9
128ms
```

(example of quite big field, with smaller fields it works better)

Backtracking shortest path: 9 steps, more than 30 mins
Backtracking first path: 49 steps, 35 ms
Heuristic: 9 steps, 128 ms
Random search: failed, 42 ms

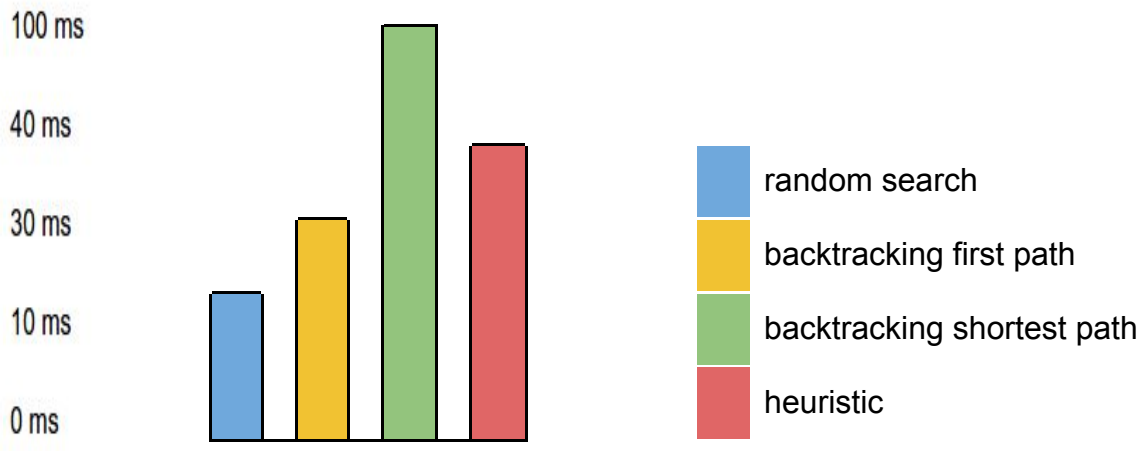
As we can see, the fastest was finding the first path of backtracking, the shortest path of backtracking was the slowest of all. Heuristic worked a bit slower than first backtracking but found shortest path. Random search failed.
So, comparing the time and number of steps, heuristic algorithm works best of all.

By the way, the chance of getting success and finding touchdown point with random search is very small (too much conditions for fail). For 4x4 field and touchdown located at [0,2] (which is very close to starting position) only 1-2 trials out of 100 are successful. So, for field 20x20 and touchdown located somewhere in the middle of the field the chance to find path is about 1 out of 1 000 000 trials.

Statistics

Everage algorithm's execution time

(for fields no more than 7x7 yards)



In general, Backtracking first path and Heuristic work quite good for most of maps. Backtracking shortest path works good for relatively small fields or for fields with small number of paths. For fields of size more than 10x10, it can take more that 20-30 mins to find the shortest path of backtracking algorithm.

Part 2

The ability to see orcs in two yards is not really helpful. The only case is when orc is placed between two humans, so if one human makes pass, the algorithm, for example backtracking, will understand that pass is not possible at earlier step, and cut this path, saving time. But if orc is placed near the touchdown, this ability will not help, cause player all the same will need to go near the orc.

H			T
O			
H			

Save time on pass

T			
O	H		
H			

Useless, needs to go near orc

The ability to see touchdown in two yards is really helpful, it reduces the time and sometimes the number of steps to find a path, especially for backtracking.

```
?- map_6(_).
true.

?- call_backtracking_search_first([0,0],[X,Y]).
Backtracking first found path:

[0,0] -> [1,0]
[1,0] -> [2,0]
[2,0] -> [3,0]
[3,0] -> [4,0]
[4,0] -> [4,1]
[4,1] -> [4,2]
[4,2] -> [3,2]
Number of steps: 7
20ms

true.
```

1 yard before touchdown

```
?- map_6(_).
true.

?- call_backtracking_search_first([0,0],[X,Y]).
Backtracking first found path:

[0,0] -> [1,0]
[1,0] -> [2,0]
[2,0] -> [3,0]
[3,0] -> [4,0]
[4,0] -> [4,1]
[4,1] -> [4,2]
[4,2] -> [3,2]
Number of steps: 7
15ms

true.
```

2 yards before touchdown

```
prolong([Temp|Tail],[New,Temp|Tail]):-
    ((left(Temp, [X,Y]), win([X,Y]));(left(Temp, [X,Y]),left([X,Y],[X2,Y2]), win([X2,Y2])) -> left(Temp, New);      /*if player is near
        ((right(Temp, [X,Y]), win([X,Y]));(right(Temp, [X,Y]),right([X,Y],[X2,Y2]), win([X2,Y2])) -> right(Temp, New);
            ((back(Temp, [X,Y]), win([X,Y]));(back(Temp, [X,Y]),back([X,Y],[X2,Y2]), win([X2,Y2])) -> back(Temp, New);
                ((forward(Temp, [X,Y]), win([X,Y]));(forward(Temp, [X,Y]),forward([X,Y],[X2,Y2]), win([X2,Y2])) -> forward(Temp, New);
                    (move(Temp,New))))),
    \+member(New,[Temp|Tail]).
```

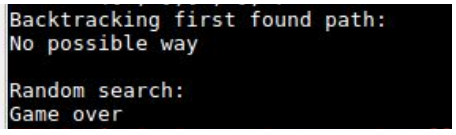
Part 3

Map is impossible when orcs are placed in circle around the initial position of human or in circle around the touchdown. Also the case when the only free way to touchdown is diagonal.

H		O	
		O	T
		O	O
H			

H		O	
		O	T
			O
H			

H			
			T
O	O		
H	O		



It is hard to say which map can be considered as difficult. I looked at two different cases: when the field is big and there is a little number of orcs, so there are a lot of possible paths and field when there are a lot of obstacles and there is only one way to go.

O				
O		O	O	
O		O	T	
O	H	O	O	O
H	O	O	O	O

backtracking: 10 steps, 15 ms
heuristic: 10 steps, 35ms
random: failed

			T	
			O	
H				

backtracking (first path): 7 steps, 35 ms
heuristic: 5 steps, 20ms
random: failed

As we can see, backtracking and heuristic algorithms behave differently. But both of them find a path. Random search failed as usual.

```
?- call_backtracking_search_first([0,0],[X,Y]).
Backtracking first found path:

[0,0] -> [1,1]
[1,1] -> [1,2]
[1,2] -> [1,3]
[1,3] -> [1,4]
[1,4] -> [2,4]
[2,4] -> [3,4]
[3,4] -> [4,4]
[4,4] -> [4,3]
[4,3] -> [4,2]
[4,2] -> [3,2]
Number of steps: 10
15ms

true.
```

```
?- call_backtracking_search_first([0,0],[X,Y]).
Backtracking first found path:

[0,0] -> [1,0]
[1,0] -> [2,0]
[2,0] -> [3,0]
[3,0] -> [4,0]
[4,0] -> [4,1]
[4,1] -> [4,2]
[4,2] -> [3,2]
Number of steps: 7
35ms

true.
```

```
?- call_heuristic([0,0],[X,Y]).
Heuristic path:

[0,0] -> [1,1]
[1,1] -> [1,2]
[1,2] -> [1,3]
[1,3] -> [1,4]
[1,4] -> [2,4]
[2,4] -> [3,4]
[3,4] -> [4,4]
[4,4] -> [4,3]
[4,3] -> [4,2]
[4,2] -> [3,2]
Number of steps: 10
35ms
```

```
?- call_heuristic([0,0],[X,Y]).
Heuristic path:

[0,0] -> [1,0]
[1,0] -> [2,0]
[2,0] -> [2,1]
[2,1] -> [2,2]
[2,2] -> [3,2]
Number of steps: 5
20ms
```