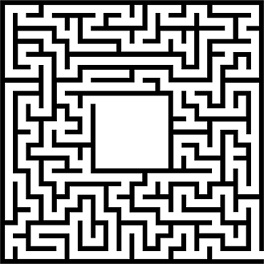
**Artificial Intelligence BTI7510 BFH TI Bern**

Prolog project for BFH TI "Artifical Intelligence", BTI7510 by Bruno Fernandez & Claudio Polo Date: 15.06.2018

[](https://github.com/brunofernandez1/Artificial_Intelligence_BTI7510/blob/master/SRC/simple-maze.jpg)

**About**

The idea of this project is to find a way from room x to room y. Problems to solve ->

* the maze and rooms are bi-directed graphs -> problem of looping
* return only the shortest way
* police officer appears randomly in room8, room10, room11 or room12
* there is a bomb that can explode a wall (%%%%)

Some help from:

* <http://tcl.sfs.uni-tuebingen.de/~cornell/prolog/Graphs003.html>
* <https://www.cpp.edu/~jrfisher/www/prolog_tutorial/2_15.html>

**Labyrinth**

+--------+--------+--------+--------+--------+

| START Room2 # Room3 Room24 Room4 |

+--------+--------+--------+--------+-- --+

| Room5 | | Room6 |

+-- --+--------+--------+--------+-- --+

| Room7 Room23 Room22 Room11 Room8 |

+-- --+--------+--------+-- --+-- --+

| Room9 | | Room12 Room10 |

+-- --+--------+ +--------+--%%%%--+

| Room13 Room14 | | Room21 |

+--------+-- --+--------+ +-- --+

| Room15 End | | Room20 |

+-- --+-- --+--------+-- --+

| Room16 | Room17 Room18 Room19 |

+--------+--------+--------+--------+

%%%% -> wall to explode

# -> bomb to collected

room8, room10, room11, room12 -> possible rooms the police officer can be located

**How to run**

To run the code please start the following main method with 3 variables. For example:

main(Waylist, Policepos, Cost)

Then it will return the shortest path with the following information

* Waylist: the list of the shortest path
* Policepos: random position of the police officer
* Cost: how much hops you needed to reach the end
* Depending on whether the figure is caught or not, you will receive one of those messages to the console:
  + You have found a way to escape!
  + The police caught you!

Here are two examples:

?- main(Waylist, Policepos, Cost).

You have found a way to escape!

Waylist = [start,room2,room3,room24,room4,room6,room8,room10,room21,room20,room19,room18,room17,end],

Policepos = room12,

Cost = 14.

?- main(Waylist, Policepos, Cost).

The police caught you!

Waylist = [start,room2,room3,room24,room4,room6,room8,room10,room21,room20,room19,room18,room17,end],

Policepos = room10,

Cost = 14

Hint: to see the whole list, run the following command on the console

set\_prolog\_flag(answer\_write\_options,[max\_depth(100)]).

## Some explanations for the code

### Rooms

A door connects two rooms. So this is the logical fact or relationship of both rooms. Here an example for the first room:

door(start,room2).

If room A is connected with room B, then room B also connected with room A. With this rule, we bypassed the long list of creating both relations:

connected(X,Y) :- door(X,Y) ; door(Y,X).

### Ways and walk

To find all possible ways, we use a recursive depth first search function. As input

way(X,Y,Waylist) :-

walk(X,Y,[X],Z),

reverse(Z,Waylist).

walk(X,Y,Roomlist,[Y|Roomlist]) :-

(connected(X,Y);blowwall(X,Y,Roomlist)) .

walk(X,Y,Roomlist,Path) :-

(connected(X,Z);blowwall(X,Z,Roomlist)),

Z \== Y,

\+member(Z,Roomlist),

walk(Z,Y,[Z|Roomlist],Path).

As we only want the shortest path, we return only the path with the smallest number of rooms (hops):

shortestPath(X, Y, L, N) :-

length(L, N),

way(X, Y, L),!.

### Bomb

The bomb rule works as follow:

* X is the room where the bomb is located
* Y and Z are the rooms which can be connected if you explode the wall with a bomb

bomb(X,Y,Z).

Now if you have passed the room where the bomb was located and you reach the room where you can explode the wall, the new room will be added do the possible rooms.

blowwall(X,Y,Roomlist):-

(bomb(B, X, Y); bomb(B, Y, X)),

member(B,Roomlist), !.

### Police

The police rule defines a list of possible rooms where the police officer can be locaded:

police([room11,room8,room12,room10]).

Now, each time you start the main code, the police is randomly set in on those rooms

getPolList(X) :- police(Y), random\_member(X, Y).

**Known errors**

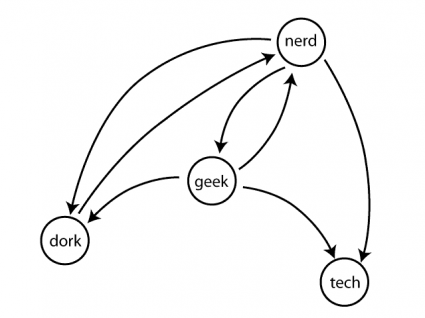
Those errors/problems are known and will be resolved in a later release

* Some times it just returns a false message and it can't find any way to the end room. No idea why
* If we have found multiple ways with the same path length, it just returns the first one.
* extend the code to find another way if the police officer is blocking the path

**Personal review**

In this project we worked for the first time with a "logical programming language" like Prolog. We noticed that you have to leave behind the knowhow of the procedural development and you have to change your approach how to solve the problem. It takes a lot of rethinking in the beginning to handle this language.

As long as you only work with prolog facts, it is still manageable. But as soon as you start with prolog rules, it quickly becomes difficult. Especially if you have to solve certain problem with recursion, it quickly becomes unmanageable.

We spend a lot of time with the traversing of the rooms. At first we had the problem that the traversing program ended up in an infinite loop and finally crashed. After finding out that the rooms are actually a directed graph, we skipped the rooms that have already been visited. To realize this in prologue was more difficult than expected. [](https://github.com/brunofernandez1/Artificial_Intelligence_BTI7510/blob/master/SRC/directed-cyclic-graph-425x318.png)

In one of the linked URLs, our problem was explained in detail and we could implement it with their help.

The remaining functions were a lot easier to implement and fast realized. However, we have both noticed that this is not our favourite way to program and will probably make a big bow to Prologue in the future.