Plman Slides
(An incremental doc)

Francisco Escolano

Download & Installation

Download package pl-man.zip from: PLMan

Unzip the package (e.g, in Desktop).
It will create a folder **plman**, which in turn contains 2 subfolders:

1. *maps/* contains 12 examples of maps to solve.
2. *pl-man-game/* contains the Prolog code of the game Pl-Man.

The file plman is the script to execute the solutions to the maps.
The file launch is the script to execute many solutions.

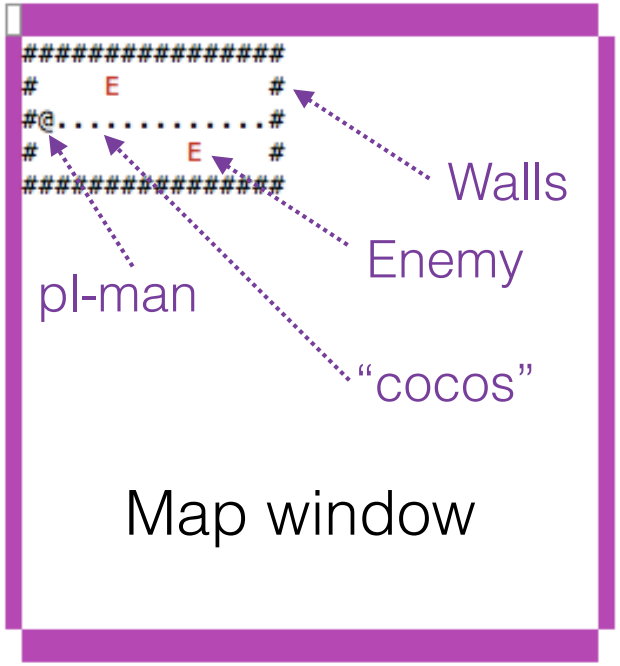
THESE FILES MUST BE EXECUTABLE in a UNIX/LINUX system: **$ chmod +x plman**

Launching Pl-man

Usual call to plman:

./plman MAP SOLUTION [PARAMETERS]

MAP: is the name of the **.pl file** which contains the map to solve,
SOLUTION is the name or path of the **.pl file** containing the rule that triggers the proposed solution to this map,



Launching PI-man

mapej0.pl

```
map([['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#'],
['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', '#'],
['#', ' ', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.',
'.', '#'],
['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', '#'],
['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#']]).
map_size(16, 5).
num_dots(13).
pacman_start(1, 2).
initMap:-
        addSolidObject('#'),
        createGameEntity(EID_0, 'E', mortal, 5, 1, active,
enemyBasicMovement, [appearance(attribs(normal, red, default))]),
        createGameEntity(EID_1, 'E', mortal, 10, 3, active,
enemyBasicMovement, [appearance(attribs(normal, red, default))]),
        enemyBasicMovement(init, EID_0, up-down, ['#']),
        enemyBasicMovement(init, EID_1, up-down, ['#']).
norule(_).
norule(_,_,_,_,_).
```

Launching Pl-man

solej0.pl

```
:- use_module('pl-man-game/main').

do(move(right)).
```

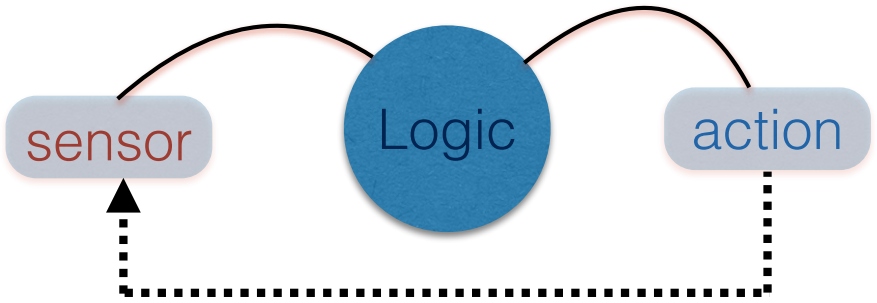
```
################
#    E         #
#@............#
#          E   #
################
```

```
./plman maps/ejemplos/mapaej0.pl maps/ejemplos/solej0.pl
```

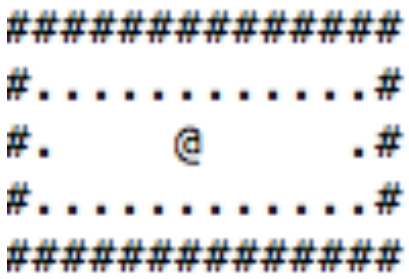
solej1.pl?

```
##############
#@..........#
############.#
#..........#
##############
```

```
:- use_module('pl-man-game/main').

do(move(right)):- see(normal,right,'.').
do(move(left)):- see(normal,down,'.').
do(move(down)):- see(normal,left,'.').
```



Launching Pl-man



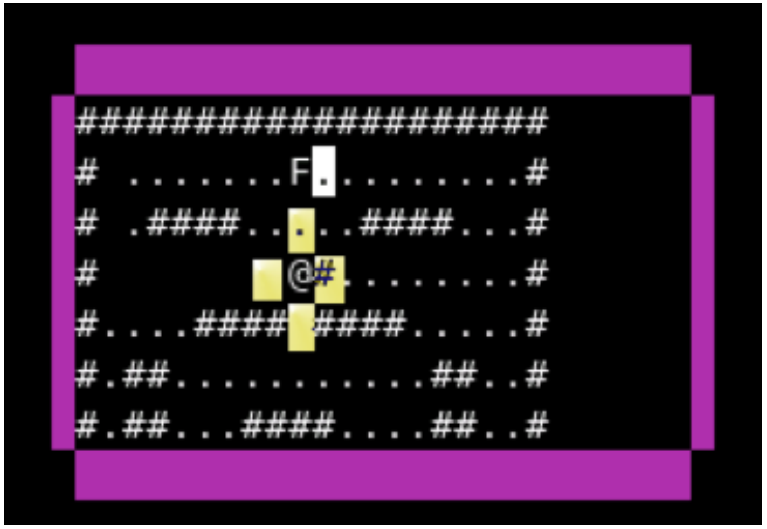
solej2.pl?

Strategy

1. Move left while seeing void ‘ ‘
2. Move left (“eating” points) ‘#’ blocks
3. Move up until being blocked by top ‘#’
4. Move right until being blocked by right ‘#’
5. Move down until being blocked by right ‘#’
6. Move left until being blocked by left ‘#’

```
:- use_module('pl-man-game/main').

do(move(right)):- see(normal,right,'.').
do(move(up)):- see(normal,left,'#').
do(move(left)):- see(normal,left,'.').
do(move(down)):- see(normal,right,'#').
do(move(left)):- see(normal,down,'#').
do(move(left)).
```

See & Move Commands



See “normal”

see(normal, DIR, ‘.’)

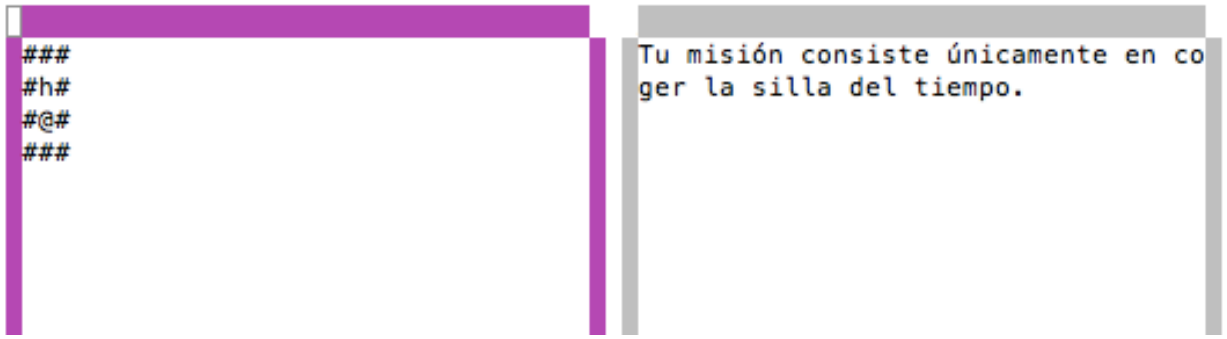
move

move(DIR)

```
DIR in
{up, down, right, left, up-left, none
up-right, down-left, down-right, here}
```

```
:- use_module('pl-man-game/main').

do(move(right)):- see(normal,right,'.').
do(move(up)):- see(normal,left,'#').
do(move(left)):- see(normal,left,'.').
do(move(down)):- see(normal,right,'#').
do(move(left)):- see(normal,down,'#').
do(move(left)).
```

Phase 0: get/drop/use objects



Your mission consists of getting the chair of time!

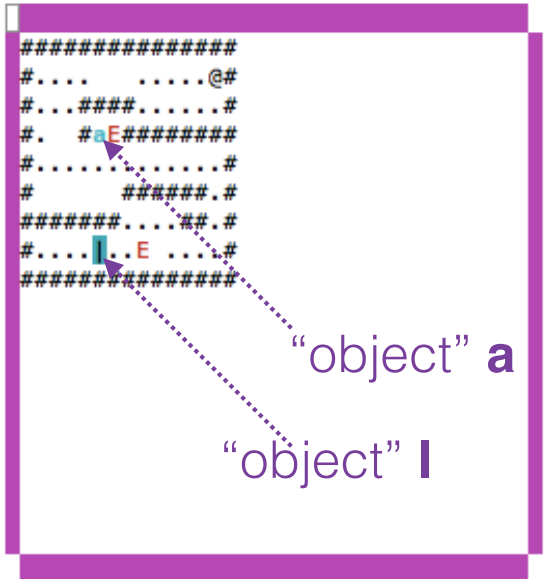
get/drop/use

get/drop/use(OBJ)

```
OBJ in
{up, down, right, left,
up-left, none up-right,
down-left, down-right, here}
```

```
map_format_version(1.0).
map([['#', '#', '#'],
['#', ' ', '#' ],
['#', ' ', '#'],
['#', '#', '#']]).
map_size(5, 3).
num_dots(1).
pacman_start(1, 2).
initMap:-
    addSolidObject('#'),
    createGameEntity('h', object, 1, 1, inactive, norule,
                     data(silla, solid, not_static, norule, 'Silla del tiempo: Te teletransporta al levantarla.')),
    createGameEntity('#', object, 2, 1, active, pickUpRule,
            data(checker, solid, static, norule, 'Checks end of the game')),
    msgWindowWrite('Tu misión consiste únicamente en coger la silla del tiempo.').

pickUpRule(_):-
    havingObject ->
    dotEaten.
pickUpRule(_).

norule(_).
norule(_,_,_,_,_).
```

Phases 0,1,2: **havingObject**



In these phases solving maps often requires object interaction to:

- Get a key to open a door
- Drop an object at a given place.

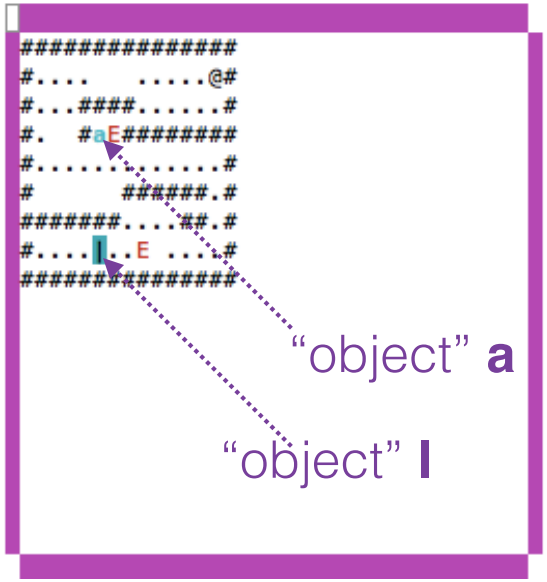
PI-man must know whether it has the required object or it doesn’t it.

- Use the predicate: **havingObject**
- It works on “name” and “appearance”

In the map’s code…

```
createGameEntity(OID_KEY, 'a', object, 5, 3, inactive, norule,
       [ name(llave_azul), solid(false), static(false), use_rule(basicDoorKey),
        description('Llave que abre puertas azules'), appearance(attribs(bold, cyan, default))]),
createGameEntity(OID_DOOR1, '|', object, 5, 7, inactive, norule,
                        [name(puerta_azul), solid(true), static(true), use_rule(norule),
                        description('Puerta azul 2'), appearance(attribs(bold, black, cyan))]),
createGameEntity(EID_0, 'E', mortal, 6, 3, active, enemyBasicMovement, [appearance(attribs(normal, red,
default))]),
createGameEntity(EID_2, 'E', mortal, 8, 7, active, entitySequentialMovement, [appearance(attribs(normal, red,
default))]),
```

Phases 0,1,2: **havingObject**



In these phases solving maps often requires object interaction to:

- Get a key to open a door
- Drop an object at a given place.

PI-man must know whether it has the required object or it doesn’t it.

- Use the predicate: **havingObject**
- It works on “name” and “appearance”

In the map’s code…

```
createGameEntity(OID_KEY, 'a', object, 5, 3, inactive, norule,
       [ name(llave_azul), solid(false), static(false), use_rule(basicDoorKey),
        description('Llave que abre puertas azules'), appearance(attribs(bold, cyan, default))]),
createGameEntity(OID_DOOR1, '|', object, 5, 7, inactive, norule,
                        [name(puerta_azul), solid(true), static(true), use_rule(norule),
                        description('Puerta azul 2'), appearance(attribs(bold, black, cyan))]),
createGameEntity(EID_0, 'E', mortal, 6, 3, active, enemyBasicMovement, [appearance(attribs(normal, red,
default))]),
createGameEntity(EID_2, 'E', mortal, 8, 7, active, entitySequentialMovement, [appearance(attribs(normal, red,
default))]),
```

Phase 3: **Dynamic Predicates**

state(up):

- Do the sum below and store in result()
- Read the result of the sum (get the corresponding number)
- Clean dots
- Go to the door (+ sign) and open.

state(down):

- Get the gun I
  Go down and shoot **E**
- Clean dots

Phase 3: **Dynamic Predicates**

```
:- use_module('pl-man-game/main').

%% Predicate for controlling the state
:- dynamic state/1.
state(up).
:- dynamic result/1.
result(0).
:- dynamic line/1.
line(0).

%% Symplifying rules
hO(OBJ)          :- havingObject(appearance(OBJ)).
s(DIR, OBJ)      :- see(normal, DIR, OBJ).
change(EST) :- retractall(state(_)), assert(state(EST)).
setresult(Z) :- retractall(result(_)), assert(result(Z)).
setline(L) :- retractall(line(_)), assert(line(L)), writeln(L).
```

```
%% Sub-Rules for doing the sum below me

% Rules to open and go down
```

Phase 3: **Dynamic Predicates**

```
################
#.|...#     v#m#
###...#      #.#
#.|...@       a#
###...#      #r#
#.|...#^     #.#
################
```

map6.pl

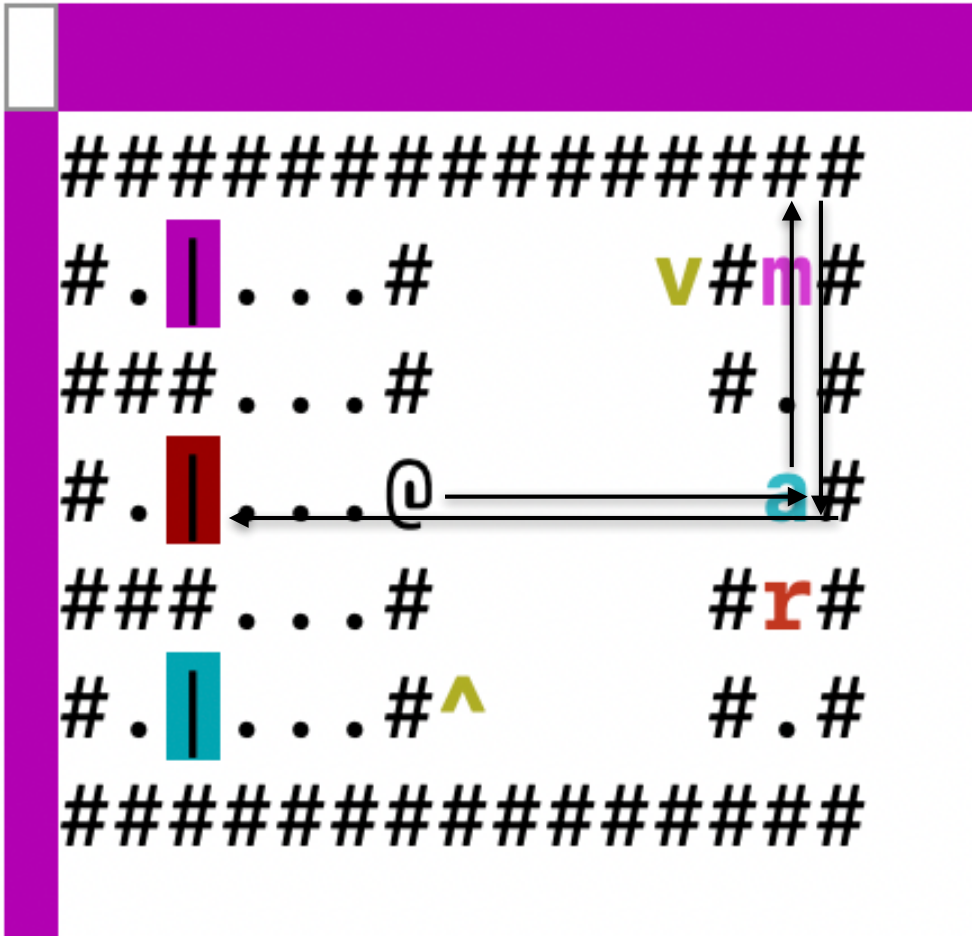
Description:

- The 3 doors appear always randomly.
- The keys are always on the right chamber. They can even be in the same place!
- Arcs are one up and one down but may appear in a random column: the down-left can be in cols 7 or 8 and the top-right can be at the cols 10 or 11

Strategy:

- Move in one direction until reaching a #
- Get the first key you find.
- Go to a position
- Try to open a door
- repeat…

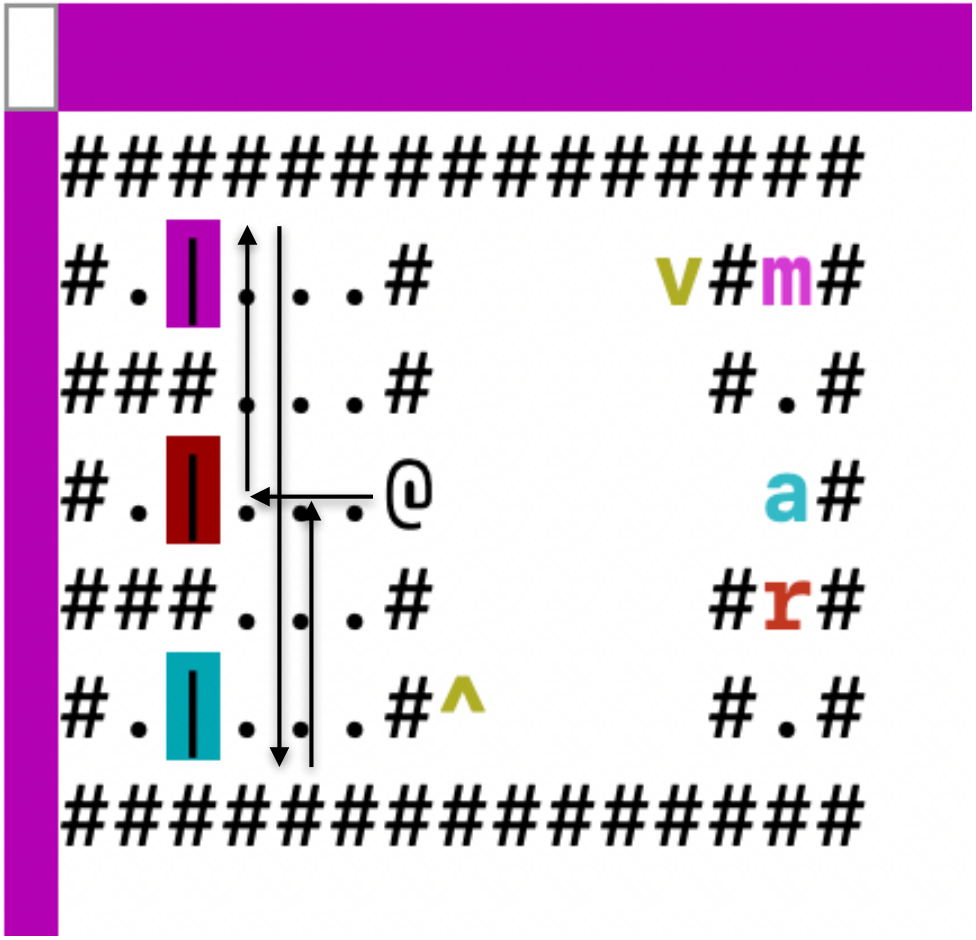
Phase 3: **Dynamic Predicates**



First Part:

- Go right.
- Go top.
- Go down.
- Sometimes you have to Go more down (more keys)
- Find the key.
- Return left.
- Use/drop key.
- Repeat

Phase 3: **Dynamic Predicates**



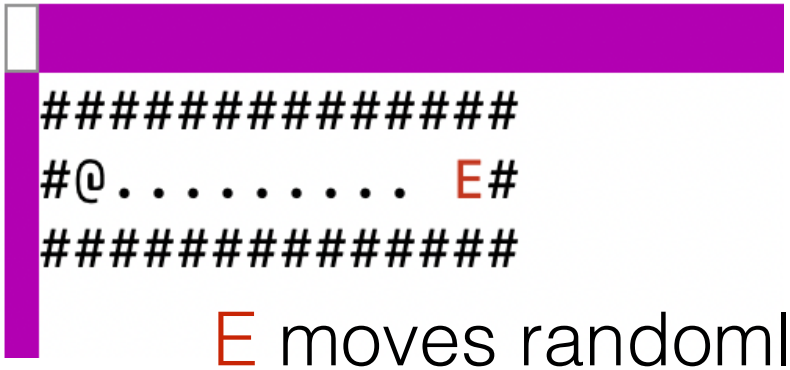
Second part:

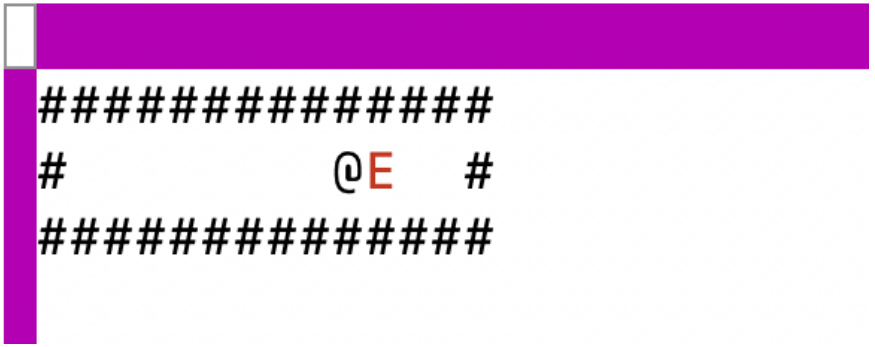
- Go to the middle door.
- Try to open.
- Find the key.
- Return left.
- Use/drop key.
- Repeat to the right UNTIL ALL DOORS ARE OPEN

Finally get the remaining dots.

Phase 4: **Using** List **Predicates**

map0.pl





E moves randomly, but…

```
map_format_version(1.0).
load_behaviour(entitySequentialMovement).
map([['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#'],
['#', ' ', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', ' ', ' ', '#'],
['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]).
map_size(14, 3).
num_dots(9).
pacman_start(1, 1).
initMap:-
        addSolidObject('#'),
        randomBetween(3,8,L), randomBetween(1,6,N),
        appendRepeatNTimes(  l,  L, [], L1),
        appendRepeatNTimes(  r,  L, L1, L2),
        appendRepeatNTimes(  n,  N, L2, L3),
        randomPermutation(L3, L3P),
        createGameEntity(EID_0, 'E', mortal, 12, 1, active,
entitySequentialMovement, [appearance(attribs(normal, red, default))]),
        entitySequentialMovement(init, EID_0, L3P, [ no_repeat_moves ]).
norule(_).
norule(_,_,_,_,_).
```

Never moves more than 8 cells left!

Phase 4: **Using** List **Predicates**

map0.pl

```
###############
#@.........  E#
###############
```

```
###############
#             @E   #
###############
```

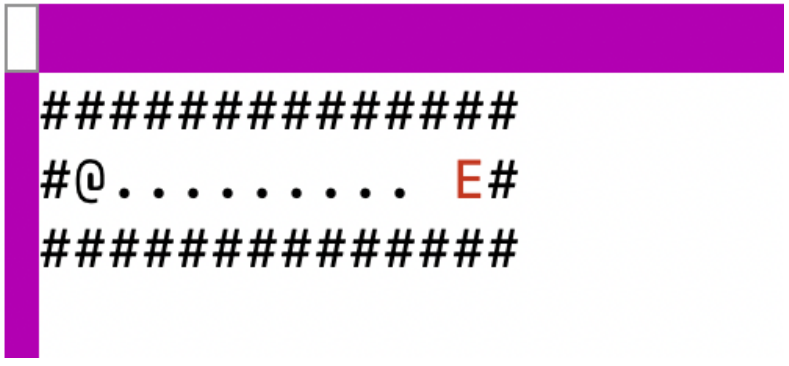
```
:- use_module('pl-man-game/main').

:- dynamic state/1.
state(initial).

sl(DIR,LIST):- see(list,DIR,LIST).
s(DIR,OBJ):- see(normal,DIR,OBJ).

doE(initial, move( right)):- sl(right,L), member('E',L), nth0(POS,L,'E'),POS>1.
doE(initial, move( none)):- sl(right,L), member('E',L), nth0(POS,L,'E'),POS<4.

%doE(enter, move( left)) :- s( left, '.').
%doE(enter, move(right)) :- s(right, '.').
%doE(enter, move( down)) :- s( down, '.').
%doE(enter, move(right)).

%% Main rule for launching sub-rules
%%
do(ACT) :- state(EST), doE(EST, ACT).
```

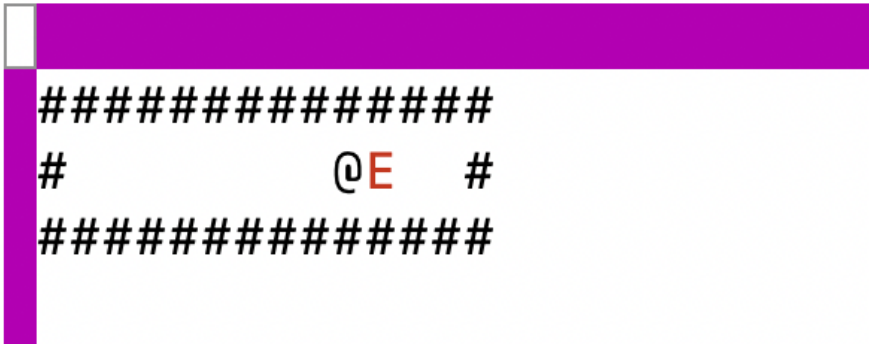
'E' belongs to list L

'E' is POS positions far from Plman starting by 0,1,…

Phase 4: **Using** List **Predicates**

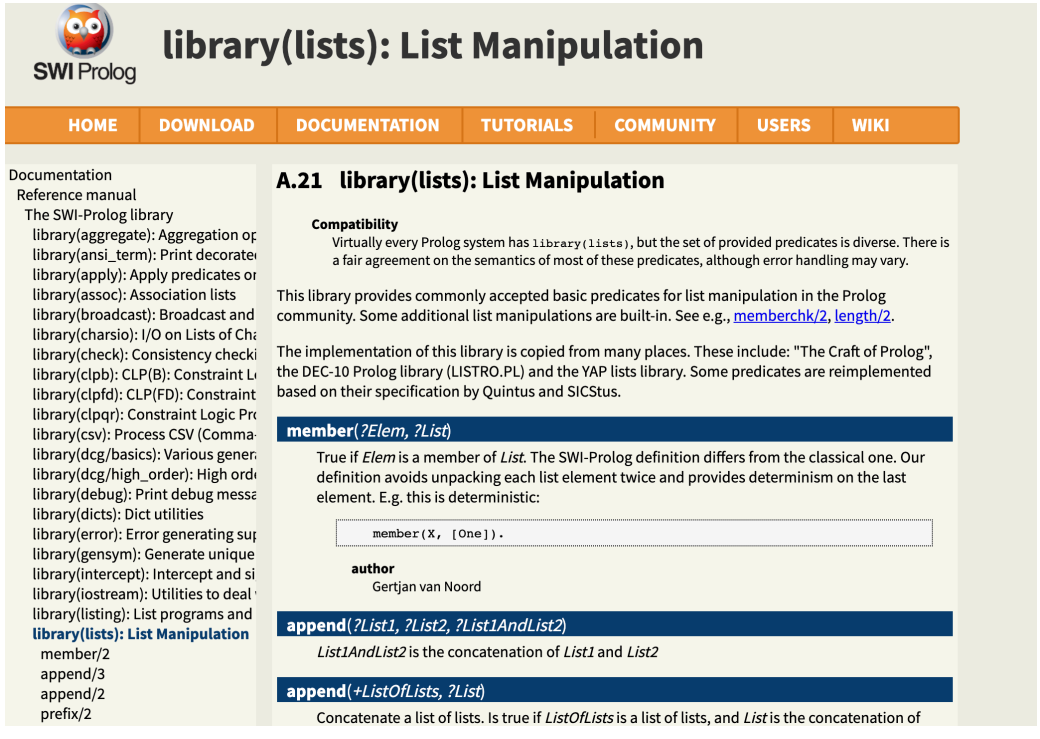
map0.pl





List predicates in swi-prolog:

https://www.swi-prolog.org/pldoc/man?section=lists



SWI Prolog

library(lists): List Manipulation

HOME | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | USERS | WIKI

Documentation
Reference manual
The SWI-Prolog library
library(aggregate): Aggregation op
library(ansi_term): Print decorate
library(apply): Apply predicates or
library(assoc): Association lists
library(broadcast): Broadcast and
library(charsio): I/O on Lists of Cha
library(check): Consistency checki
library(clpb): CLP(B): Constraint L
library(clpfd): CLP(FD): Constraint
library(clpqr): Constraint Logic Pro
library(csv): Process CSV (Comma-
library(dcg/basics): Various gener
library(dcg/high_order): High ord
library(debug): Print debug messa
library(dicts): Dict utilities
library(error): Error generating sup
library(gensym): Generate unique
library(intercept): Intercept and si
library(iostream): Utilities to deal
library(listing): List programs and
library(lists): List Manipulation
member/2
append/3
append/2
prefix/2

A.21 library(lists): List Manipulation

Compatibility
Virtually every Prolog system has `library(lists)`, but the set of provided predicates is diverse. There is a fair agreement on the semantics of most of these predicates, although error handling may vary.

This library provides commonly accepted basic predicates for list manipulation in the Prolog community. Some additional list manipulations are built-in. See e.g., memberchk/2, length/2.

The implementation of this library is copied from many places. These include: "The Craft of Prolog", the DEC-10 Prolog library (LISTRO.PL) and the YAP lists library. Some predicates are reimplemented based on their specification by Quintus and SICStus.

member(*?Elem, ?List*)

True if *Elem* is a member of *List*. The SWI-Prolog definition differs from the classical one. Our definition avoids unpacking each list element twice and provides determinism on the last element. E.g. this is deterministic:

```
member(X, [One]).
```

author
Gertjan van Noord

append(*?List1, ?List2, ?List1AndList2*)

List1AndList2 is the concatenation of *List1* and *List2*

append(*+ListOfLists, ?List*)

Concatenate a list of lists. Is true if *ListOfLists* is a list of lists, and *List* is the concatenation of