```
/* ZONE root */
newzone(root)?

entree(2,3).

sortie(9,8).

mur(1,1).
mur(1,2).
mur(1,3).
mur(1,4).
mur(1,5).
mur(1,6).
mur(1,7).
mur(1,8).
mur(1,9).
mur(1,10).
mur(2,1).
mur(2,4).
mur(2,5).
mur(2,10).
mur(3,1).
mur(3,3).
mur(3,7).
mur(3,8).
mur(3,10).
mur(4,1).
mur(4,5).
mur(4,7).
mur(4,10).
mur(5,1).
mur(5,3).
mur(5,4).
mur(5,5).
mur(5,6).
mur(5,7).
mur(5,9).
mur(5,10).
mur(6,1).
mur(6,5).
mur(6,9).
mur(6,10).
mur(7,1).
mur(7,2).
mur(7,3).
mur(7,5).
mur(7,7).
mur(7,8).
mur(7,9).
mur(7,10).
mur(8,1).
mur(8,10).
mur(9,1).
mur(9,2).
mur(9,4).
mur(9,5).
mur(9,6).
mur(9,7).
mur(9,9).
mur(9,10).
mur(10,1).
mur(10,2).
mur(10,3).
mur(10,4).
```

```
mur(10,5).
mur(10,6).
mur(10,7).
mur(10,8).
mur(10,9).
mur(10,10).

labyrinthe :-  clear_scr,  posx(0),  posy(0),  laby.

laby :-  entree(L,C),  go([[L,C]],Chemin_retourne),  reverse(Chemin_retourne,Chemin),  clear_scr,
         dessine(Chemin),  posy(17),  posx(0),  write('Un chemin trouve est: '),  nl,  write(Chemin),  getc(_),  fail.
laby :-  nl,  write('Fin de la recherche.').

go([[L,C]|R],[[L,C]|R]) :-  sortie(L,C),  !.
go([Point|Reste],Parcours) :-  avance(Point,[LS,CS]),  not mur(LS,CS),  not member([LS,CS],[Point|Reste]),
  go([[LS,CS],Point|Reste],Parcours).

avance([L,C],[LS,C]) :- LS is L - 1.
avance([L,C],[L,CS]) :- CS is C + 1.
avance([L,C],[LS,C]) :- LS is L + 1.
avance([L,C],[L,CS]) :- CS is C - 1.

member(X,[X|_]) :- !.
member(X,[_|R]) :- member(X,R).

reverse(X,Y) :- reverse1([],X,Y).

reverse1(S,[],S).
reverse1(Pile,[X|E],S) :- reverse1([X|Pile],E,S).

dessine(Chemin) :-  dessine_mur,  dessine1(Chemin).

dessine_mur :-  mur(Y,X),  posx(X),  posy(Y),  write('Û'),  fail.
dessine_mur.

dessine1([]).
dessine1([[Y,X]|R]) :-  posx(X),  posy(Y),  write(x),  dessine1(R).


/* END OF ZONE root */
```