PRÁCTICAS de Matemáticas 1

2023-2024

7

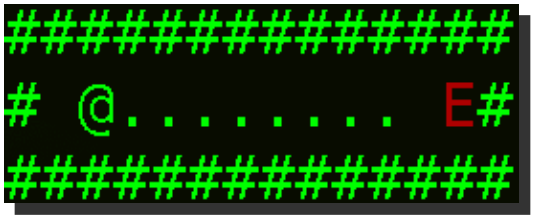
En la fase 4 Plman ve "todo" lo que hay en la fila /columna en la que se encuentra

El sensor de visión para ver una fila/columna completa hasta el primer objeto sólido es: list

see(list, D, O)

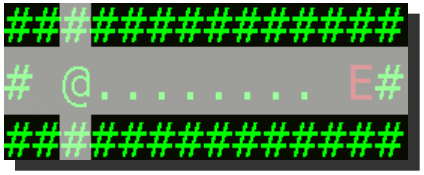
Predicados predefinidos en Prolog para manejar listas

Hasta ahora, plman sólo podía ver las
8 posiciones contiguas a él .



plman /maps/f4/mapa0.pl

Nos interesa ver toda la fila para **detectar** el enemigo



see(list, dirección, lista_visión)

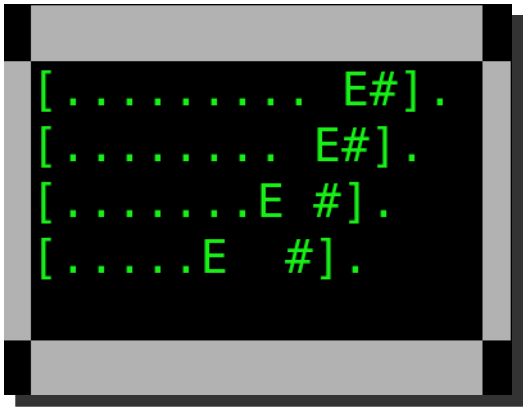
Direcciones Válidas: up down, left right

List : sensor de visión que permite a plman ver **toda la fila /columna** hasta el primer objeto sólido

Todo carácter, en el orden en que lo ve, lo almacena en una lista

Prueba a ejecutar este código para el mapa de plman/maps/fase4/mapa0.pl y verás cómo funciona el sensor de list.

El código escribe lo que plman ve




```
% Escribir por pantalla una lista
printList(L) :-
      write('[')
    , maplist(write, L)
    , writeln('].').
% Mirar y escribir lo que veo
seePrint(DIR, L) :-
      see(list, DIR, L)
    , printList(L).
% Si lo primero a mi derecha es
% un coco, mover a la derecha
do(move(right)) :-
      seePrint(right, L)
    , L = ['.'|_].
% Mover a la izquierda por defecto
do(move( left)).
```

Las listas en Prolog

Son **estructuras de datos** que permiten guardar datos relacionados, por ej., un mapa de plman

```
MAP=[
     ['#', '#', '#', '#', '#', '#', '#', '#', '#']
   , ['#', ' ', '.', '.', '.', '.', '.', ' ', '#']
   , ['#', '#', '#', '#', '#', '#', '#', '#', '#']
   ].
```

Los datos se escriben entre corchetes separados por comas, por ej.

L = ['.', '.', 'E']

L es una lista formada por 3 elementos, ordenados.

Las listas pueden tener cualquier longitud

En una lista se consideran dos partes: la **cabeza y la cola**.
La cabeza es el primer elemento; la cola es la lista que resulta de eliminar el primer elemento

L = ['.', '.', 'E'], L = [X | Y]

cabeza de la lista L: X = '.'
cola de la lista L: Y = ['.', 'E']

En una terminal abre SWIProlog ($ swipl) y prueba con estos ejemplos cuál es el valor de las variables para cada lista.
Comprueba el valor de la cabeza y cola de cada lista.

Prueba en swipl

```
?- [[' '], ['.', '#']]=[A, B].
?- L = [[' '], ['.', '#']],
   L = [A, B].
?- [[' '], ['.', '#']] = [A].
?- L = [[' '], ['.', '#']],
   L = [_, [X, Y]].
?- L = [[' ', 'P'],['.', '#']],
   L = [[X, Y], Z].
?- L = [[' ', ' '],[' ', ' ']],
   L = [[X, X], _].
?- L = [[' ', ' '],[' ', ' ']],
   L = [[X, X], X].
?- [['.', '.']] = ['.', '.'].
?- A = [['.', ' ']].
?- [A] = [['.', ' ']].
?- [[A]] = [['.', ' ']].
?- [[A, B]] = [['.', ' ']].
```

Algunos predicados predefinidos en Prolog para manejar listas

member /2

member(E, L)

True cuando el elemento, E, pertenece a la lista, L

Puedes usarlo, en el cuerpo de una regla, como una condición para hacer una acción .

```
do(get(DIR)) :- see(normal, DIR, '-'), member( DIR, [up, down, left, right] ).
```

Explicación: Con el sensor normal plman comprueba si ve la llave '-' en cualquier dirección DIR. DIR es una variable que toma uno de los valores de la lista definida en el 2o argumento del predicado member/2. Si la ve, la coge.

Prueba en swipl

```
?- member( '.', [' ','.','@']).
?- member( '@', [' ','.','@']).
?- L=['.' | ['a','E', '#'] ], member('E', L).
?- member(ELEM, [' ','.','@']).
?- L=['.' | ['.','a','E','#'] ], member('a', L).
?- member( '#', ['.' | ['.','a','E','#'] ]).
?- member( ' ', ['.' | ['.','a','E','#'] ]).
?- member(ELEM, ['.' | ['.','a','E','#'] ]).
?- member(['a'], ['a', 'b', 'c']).
?- member(['a'], [['a'], ['b', 'c']]).
?- member(ELEM, [['a'], ['b', 'c']]).
```

Algunos predicados predefinidos en Prolog para manejar listas

nth0/3

nth0(Pos, L, E)

True si el elemento, E, está en la posición, Pos, de la lista, L. La posición 0 es la 1ª

Puedes usarlo, en el cuerpo de una regla, como una condición para hacer una acción.

```
do(move(none)) :- see(list, up, L), nth0(1, L, 'E').
```

Explicación: Con el sensor list, plman mira arriba (up) y lo que ve, hasta el primer objeto sólido, lo escribe en la lista L. Con la condición del predicado nth0/3 comprueba si en la 2ª posición de L hay un enemigo 'E', si es así, no se mueve.

Prueba en swipl

```
?- nth0(  0, [' ','.','@'], ' ').
?- nth0(  1, [' ','.','@'], '.').
?- L=[' ','.','@'], nth0(  2, L, 'E').
?- nth0(  2, ['.','a','E', '#'], ELEM).
?- L=['.','a','E', '#'], nth0(  3, L, ELEM).
?- nth0(POS, ['.' | ['a','E', '#'] ], 'E').
?- L=['.' | ['a','E', '#'] ], nth0(POS, L, '#').
?- L=['.' | ['a','E', '#'] ], nth0(POS, L, ELEM).
?- nth0(10, ['.' | ['a','E', '#'] ], ELEM).
```

Algunos predicados predefinidos en Prolog para manejar listas

last/2

last(L, E)

True si el elemento, E, es el último elemento de la lista, L

Puedes usarlo, en el cuerpo de una regla, como una condición para hacer una acción.

```
do(use(right)) :- see(list, right, L), last( L, 'E').
```

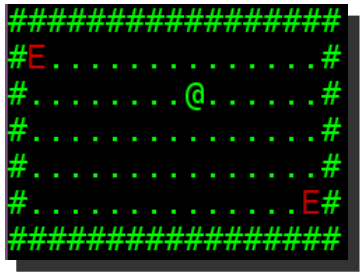
Explicación: Con el sensor list, plman mira a su derecha (right) y lo que ve, hasta el primer objeto sólido, lo escribe en la lista L. Con la condición del predicado last/2 comprueba si el ultimo element de la lista L, es un enemigo 'E', si es así, dispara.

Prueba en swipl

```
?- last([' ','@','E'], ELEM).
?- L=[' ','E',' '], last(L, ' ').
?- L=['.','.','a'], last(L, ELEM).
?- last([], ELEM).
?- L=['.' | ['a','E', '#'] ], last(L,'E').
?- L=['.' | ['a','E', '#'] ], last(L,'#').
?- L=['.' | ['a' | ['E', '#'] ] ], last(L,ELEM).
```

Resuelve algún mapa de entrenamiento plman/maps/f4

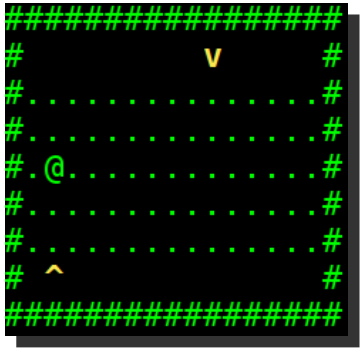
Mapas de dificultad: 2, 3, 4
En Fase 4 **sólo hay que resolver un mapa entregable**



Ejemplo de un código que usa predicados para manejar de listas y el sensor list

```
escribo_veo(DIR):- see(list, DIR, L), maplist(write,L), nl.

do(move(none)) :- escribo_veo(right), fail.
do(move(none)) :- see(list, right, [_|['E'|_]]).
```



% la regla anterior también se puede escribir, usando el predicado nth0/3

```
% do(move(none)) :- see(list, right, L), nth0(1,L,'E').
do(move(left))    :- see(normal, right, 'E').
do(move(right)) :- see(normal, right, '.').
do(move(right)) :- see(normal, right, ' ').
```