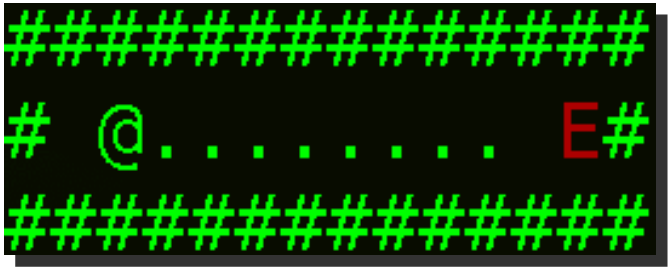
PRÁCTICAS DE MATEMÁTICAS-1

FASE 4

2018-2019

- VISIÓN LIMITADA
  - Nuevo sensor de visión
  - El sensor de lista
  - Probando el sensor
- LISTAS EN PROLOG
  - ¿Qué es una lista?
  - ¿Cómo funcionan?
  - **Resolver mapas Plman-F4**
- PREDICADOS para LISTAS
  - member/2
  - nth0/3
  - length/2
  - last/2

Enemigos Escurridizos



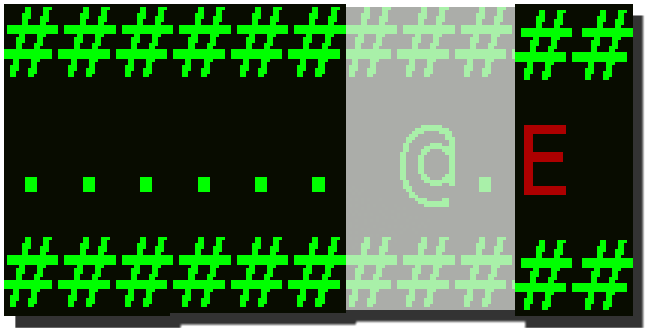
maps/fase4/mapa0.pl

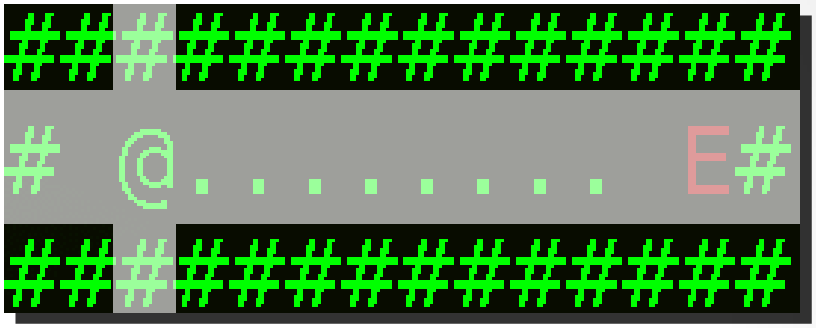
Movimiento enemigo:

→ Aleatorio, no más de 8 casillas a la izquierda

→ Vuelve a la esquina derecha cada 7-22 movimientos

Necesitamos ver más
El sensor de lista



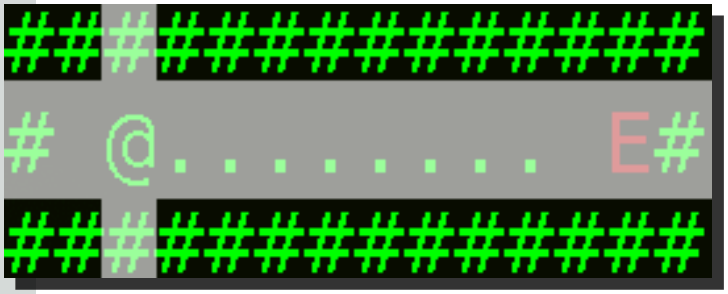


No podemos evitar algunos tipos de colisiones con enemigos

Necesitamos ampliar horizonte de visión y ser capaces de ver, por lo menos, hasta la pared más cercana

El sensor de lista

see(list, DIR, LISTA_VISION)



Direcciones Válidas
up down
left right

?- **see**(list, right, VEO).

VEO = [.,.,.,.,.,.,.,.,., ,E,#]

?- **see**(list, left, VEO).

VEO = [,#]

?- **see**(list, up, VEO).

VEO = [#]

?- **see**(list, down, VEO).

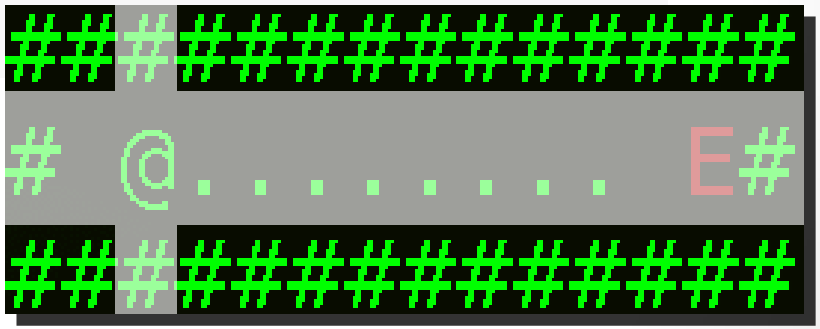
VEO = [#]

Probando el sensor

- Escribir/probar

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

plman :-   escribirLoQueVeoHacia(right), fail.
plman :-   see(normal, right, '.').
    .....
```



Listas en Prolog

- Las listas son estructuras de datos
- Guardan datos relacionados entre sí
- Permiten almacenar, operar y recuperar datos
- **Ejemplos**
  - Orden de turno de los jugadores en una partida

```
?- TURNO=[pedro, maria, juan, luis].
```

Mapa de plman

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

¿Cómo funcionan las listas?

- 2 formas de manejar los datos de las listas en PROLOG.

a) **MANEJO DIRECTO** Se escriben elementos de forma explícita, enumerando todos sus elementos en cada una de sus posiciones.

b) **SEPARANDO CABEZA/COLA**:

Permite dividir la lista en 2 elementos.

1º elemento sería el que ocupa la 1ª posición (cabeza)

2º lista con todos los demás elementos que no son el 1º (cola).

Manejo de listas

Prueba los ejemplos en el intérprete de SWI-PROLOG.

```
?- L = ['.','.','@']
?  [‘@’,‘#'] = Z.
?- ['.','.', X ] = ['.','.','@'].
?- [ A , B , C ] = ['.','.','@'].
?- [ _ , _ , X ] = ['.','.','@'].
?- [ _ , _ ,'@'] = ['.','.','@'].
?- [ _ ,'@', _ ] = ['.','.','@'].
?- ['.', X , _ ] = ['.','.','@'].
? [X] = [‘.’,’.’]
? [X,X] = [‘.’,’.’]
```

Listas: Cabeza / Cola

b) Separa las listas en cabeza / cola y comprueba

```
?- ['.','.', X ] = [CAB | CUERPO].
?- ['.','.','@'] = ['.' | ['.', '@'] ].
?- ['@','.','.'] = [CAB | _].
?- ['@'| ['.','.'] ] = [CAB,_,_].
?- ['@'| ['.','.'] ] = [E1,E2,E3].
?- ['@'| ['.','.'] ] = [CAB | CUERPO].
? ['.'] = [CAB|CUERPO].
? ['.','.','.' ] = ['.'| '.','.' ]
```

Importante: recuerda que las listas pueden tener cualquier longitud.

- Separar en cabeza / cola siempre funciona.
- Separa en cabeza / cola es más útil para preguntar por los contenidos.

member/2

- **member(ELEMENTO, LISTA)** tiene éxito si ELEMENTO pertenece a LISTA

```
?- 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','#'] ]).
```

nth0/3

nth0(POS,LISTA,ELEM) tiene éxito si ELEM es el elemento que ocupa la posición POS en la LISTA. Hay que tener en cuenta que el primer elemento está en la posición 0.

```
?- 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).
```

lenght/2

- **length(LISTA,LONG)** tiene éxito si LONG es el número de elementos (la longitud) de la LISTA.

```
?- length([' ','.','@'], 3).
?- length([' ','.','@'], 4).
?- L=[' ','.','@'], length(L, LONG).
?- L=[' ','@'], length(L, LONG).
?- length([], LONG).
?- L=['.' | ['a','E', '#'] ], length(L,4).
?- L=['.' | ['a','E', '#'] ], length(L,3).
?- L=['.' | ['a' | ['E', '#'] ] ], length(L,LON).
```

last/2

last(LISTA,ELEM) tiene éxito si ELEM es el último elemento de la LISTA.

```
?- last([' ','.','@'], '@').
?- 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).
```

Mapas de Ejemplo en maps/fase4

- maps/fase4 → 4 mapas sencillos equivalentes a nivel de dificultad 0-1
- En fase 4 sólo se podrán elegir dificultades 2,3 o 4.
- Sólo hay que **resolver 1 mapa** de fase 4
- Estos mapas requieren que hagáis comportamientos generales

Revisando fallos

Para **revisar un fallo**, volver a ejecutar no sirve, el **mapa cambia**

Script **launch** permite **ejecutar** la solución de una mapa **n veces**

$ ./launch n mapa.pl solucion.pl

El Script launch guarda logs de las que fallan

Plman permite **guardar el log** de una ejecución

$ ./plman mapa.pl solucion.pl -l archivoLog.log

Reproducir un log de una ejecución

$ ./plman -r archivoLog.log

Teclas:
P se avanza;
O se retrocede;
1-9 se cambia la velocidad de reproducción.
ESC para salir.