Prácticas de Matemáticas I 2018/2019

Fase 3

Semana 12 al 16 de noviembre

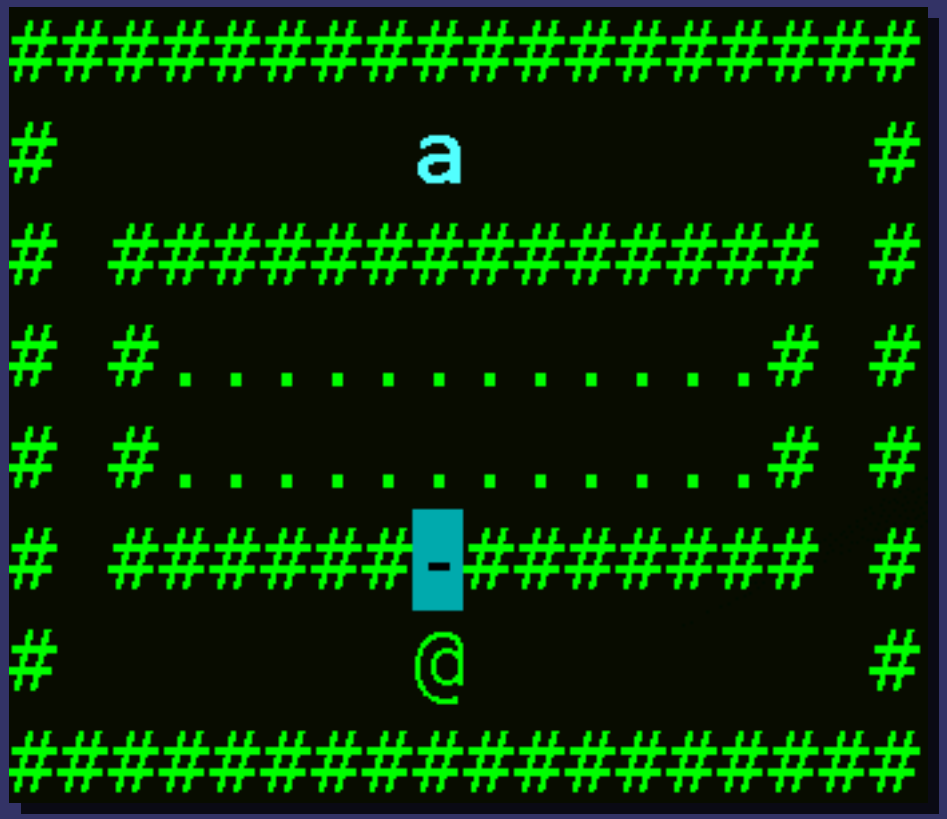
Second version – Doors of Hell
by Manwathiell

Habladurías

- Se comenta que en Fases 3 y 4 las mapas son **indeterministas**.
- ¿Qué es eso?
  - **Las cosas cambian** de una ejecución a otra.
  - Las cosas cambian de **posición**.
  - Los enemigos cambian de **comportamiento**.
  - **PLMan** no siempre empieza igual.
  - Hasta los cocos pueden **aparecer y desaparecer**.

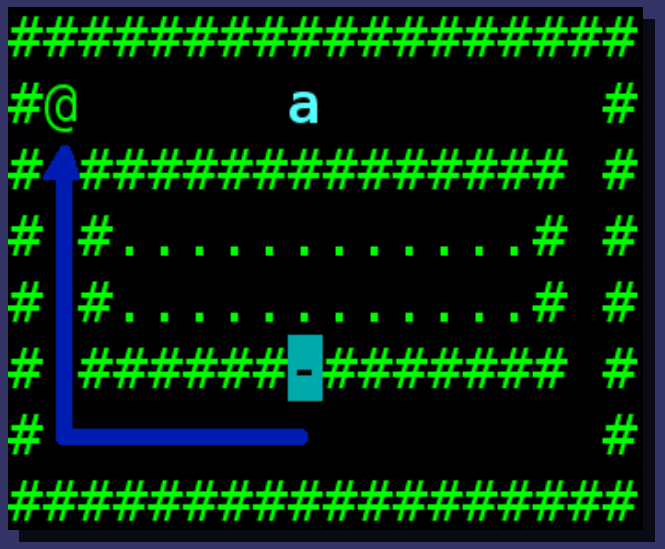
Condiciones adversas

maps/fase3/mapa1.pl



¿Cómo podemos resolver este mapa?

Nada cambia

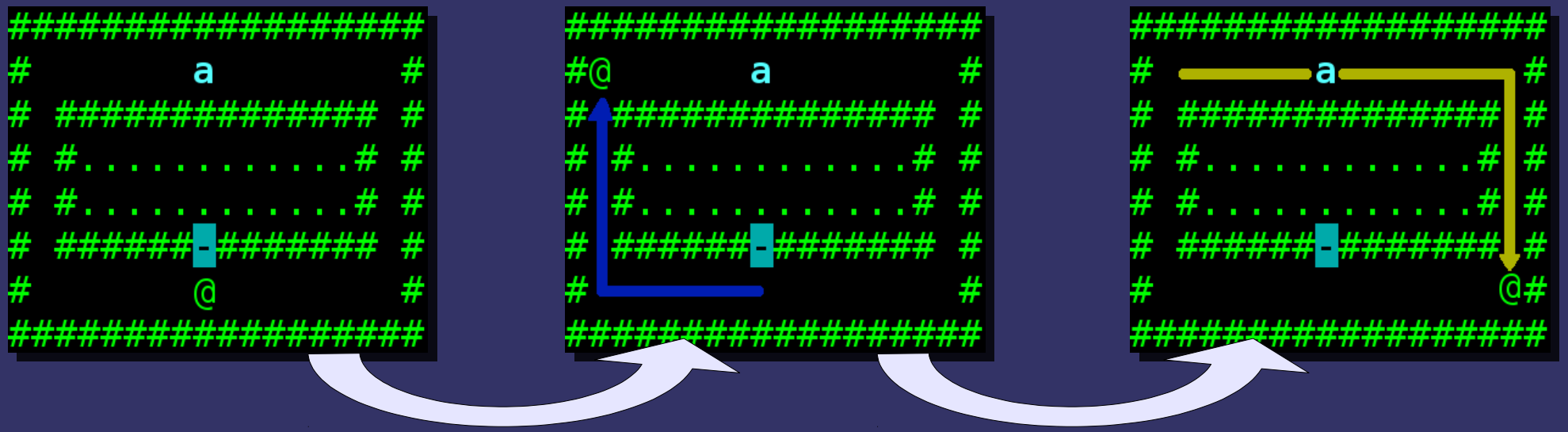



```
do(move(left)):- see(normal, left, ' ').
do(move(up)):- see(normal, up, ' ').
do(move(right)):- ????????
```

Necesitamos distinguir
cuándo estamos **arriba**
y cuándo estamos **abajo**

Predicados dinámicos

Permiten añadir/eliminar **hechos** durante la ejecución



Añadir
`estoy(abajo)`

Eliminar `estoy(abajo)`
Añadir `estoy(arriba)`

Manejando hechos en Prolog

:- dynamic predicado/aridad

Declara como dinámico un predicado,
sólo para la aridad (número de argumentos) indicada.

assert(HECHO)

añade **HECHO** a la Base de Conocimientos. **HECHO** debe ser un predicado dinámico.

Ejemplo:

```
:- dynamic estoy/1.
do(move(right)):-  see(normal, up, '#'), assert(estoy(arriba)).
```

Manejando hechos en Prolog

retract(HECHO)

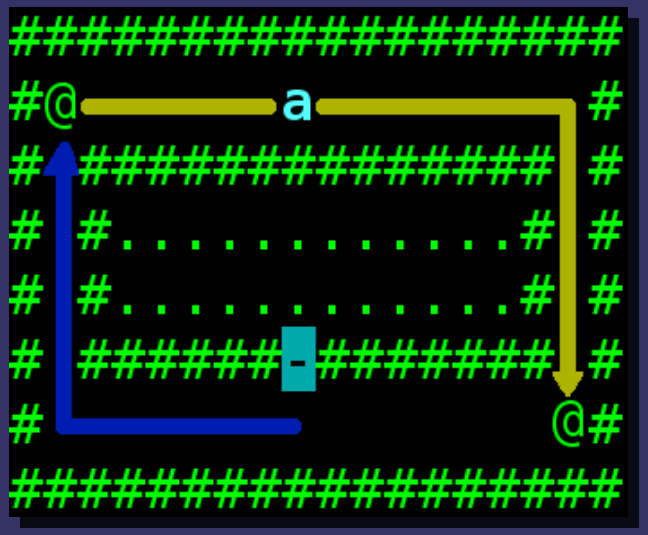
Elimina un **HECHO** de la Base de Conocimientos.
¡Ojo! Si **HECHO** no existe, **retract(HECHO) fracasa**.

retractall(HECHO)

Elimina todos los hechos que unifiquen con **HECHO** de la Base de Conocimientos. Siempre tiene éxito, incluso cuando ningún hecho unifica con **HECHO**.

Ejemplos: retract(estoy(arriba))
retractall(estoy(arriba))
retractall(estoy(_))

Añadiendo predicados dinámicos



¡Ojo!

Muchos detalles a los que prestar atención

```
:- dynamic estoy/1.

estoy(abajo).

%%reglas para cuando está abajo

do(…) :- estoy(abajo), see(…),...
do(…) :- estoy(abajo), ….
...
do(…) :- estoy(abajo), see(...),
                retract(estoy(abajo)), assert(estoy(arriba)).

%%reglas para cuando está arriba

do(…) :- estoy(arriba), see(…),...
…
do(…) :- estoy(arrriba), …
do(…) :- estoy(arrriba), …
```

Cosas que pueden fallar

Descuidos usando **dynamic**

```
dynamic estoy/1.
estoy(abajo).
```

```
:- dynamic estoy/2.
estoy(abajo).
```

```
%% dynamic no usado
estoy(abajo).
```

```
ERROR: retract/1: No permission to modify static procedure `estoy/1'
```

Forma correcta

```
:- dynamic estoy/1.
estoy(abajo).
```

Cosas que pueden fallar

Fallos utilizando **retract** / **assert**

```
1 ADVERTENCIA: La regla de control
de tu personaje ha fracasado!
```

Si hago retract(estoy): Si intenta borrar `estoy sin argumento`, fracasa.

Si hago retract(estoy(arriba)) y no existe ese hecho, fracasa.

Otro error típico es añadir un hecho de un predicado sin haber borrado otro hecho previo de ese predicado. No da error, pero tendríamos dos hechos del mismo predicado.

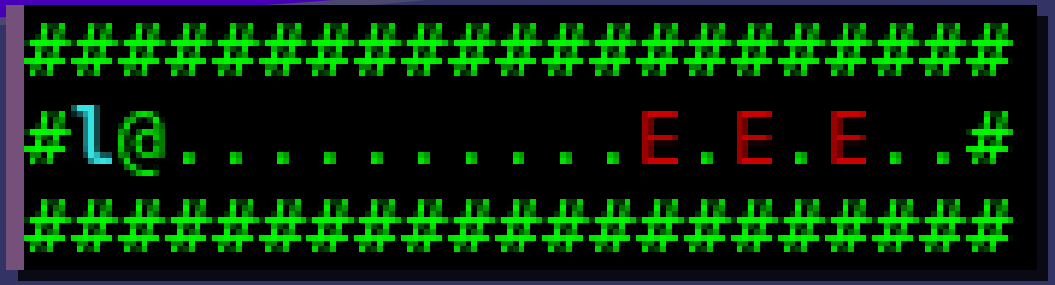
Comprobando fallos

Para saber si la ejecución pasa por un punto o no usaremos el write o writeln

```
do(…):- see(…),
        writeln('voy a hacer el retract'),
        retract(estoy(abajo)),
        writeln('Éxito. Ahora voy a hacer el assert'),
        assert(estoy(arriba)),
        writeln('regla ok').
```

Implementación de contadores con predicados dinámicos

maps/fase3/mapa5.pl



Después de coger la pistola tenemos que disparar 3 veces y luego ya ir para la derecha…

```
-dynamic cantidadMuertos/1.

cantidadMuertos(0).

do(get(left)):-  see(normal, left, 'l').
do(use(right)):- cantidadMuertos(N), N < 3, incCantidadMuertos.
do(move(right)).

incCantidadMuertos:- retract(cantidadMuertos(M)),N is M+1,
                     assert(cantidadMuertos(N)).
```

Lanzar muchas veces

Para revisar un fallo, volver a ejecutar no sirve si el mapa cambia. Lo que podemos hacer es lanzarlo n veces con el s**cript launch** y después reproducir las ejecuciones fallidas

https://logica.i3a.ua.es/downloads/launch.zip

```
$ ./launch 10 maps/fase3/mapa0.pl sol.pl
Ejecucion 1: MAPA SUPERADO 100%
Ejecucion 2: MAPA SUPERADO 100%
Ejecucion 3: MAPA SUPERADO 100%
Ejecucion 4: LIMITE DE MOVIMIENTOS SUPERADO
Ejecucion 5: MAPA SUPERADO 100%
Ejecucion 6: LIMITE DE MOVIMIENTOS SUPERADO
Ejecucion 7: MAPA SUPERADO 100%
Ejecucion 8: LIMITE DE MOVIMIENTOS SUPERADO
Ejecucion 9: MAPA SUPERADO 100%
Ejecucion 10: MAPA SUPERADO 100%
-------------------------------------------------------
-- RESULTADO FINAL:
-------------------------------------------------------
Total de ejecuciones superadas al 100%: 7 de 10 (70%)
```

Revisando fallos

- Cuando se lanza plman con el script launch se crea un archivo de log por cada ejecución fallida.

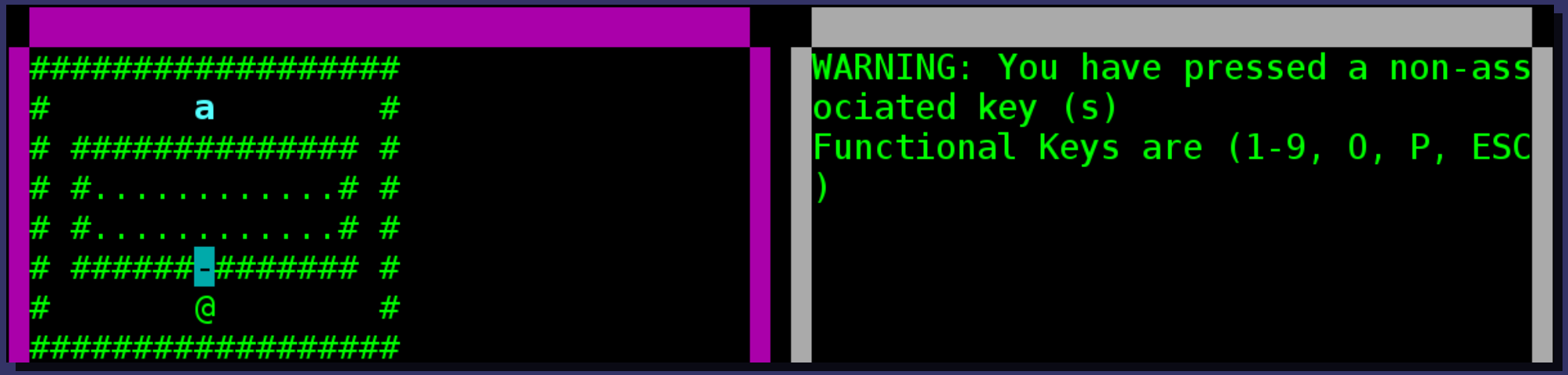
- PLMan permite guardar el **log** de una ejecución en concreto

  ./plman mapa.pl sol.p -l fichero.log

- Para reproducir un archivo de log: ./plman -r fichero.log

Reproduciendo un log

- **Teclas**
  - **P**: Avanzar un paso de ejecución
  - **O**: Retrodecer un paso de ejecución
  - **1-9**: Cambiar el tamaño del paso de ejecución (para ir más rápido o despacio)



Sugerencia

Recordad modularizar vuestro código

Organizar las reglas por grupos en función de vuestros predicados dinámicos y recordar que también podéis crear vuestros propios predicados

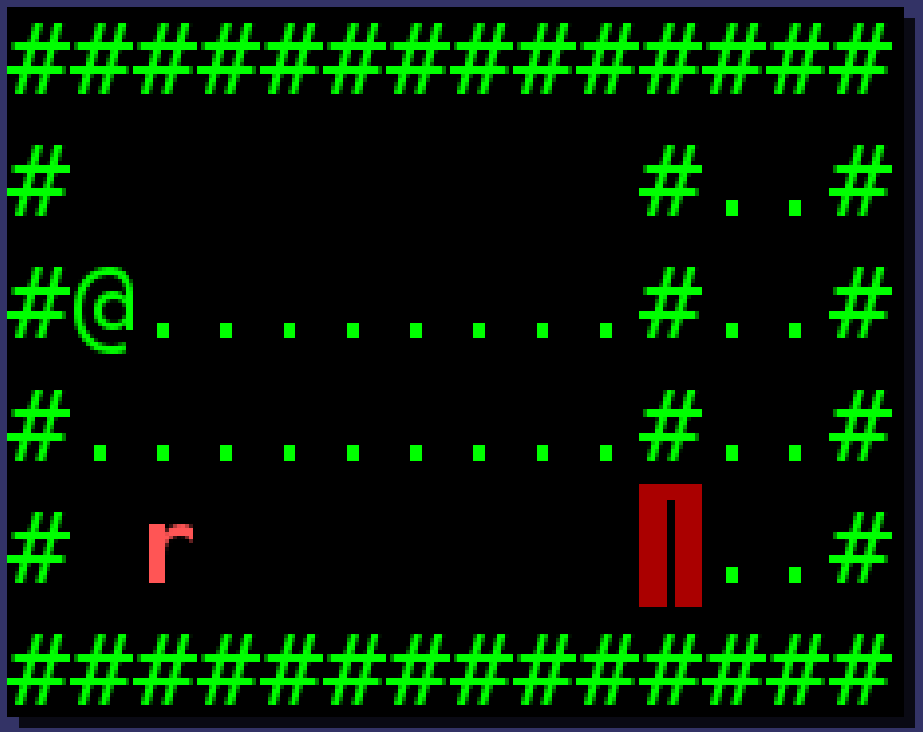
Cometeréis menos errores y más fáciles de localizar.

Importante

Programar bien
es resolver problemas
más difíciles
de forma
más fácil

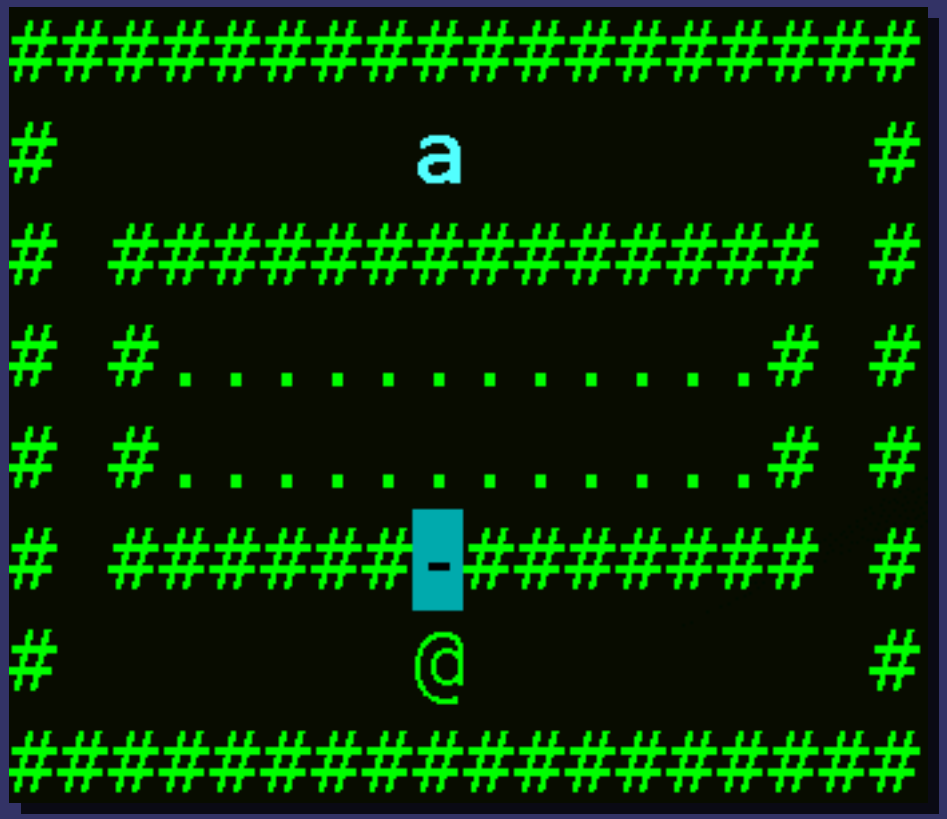
Intentad resolver el mapa0....

maps/fase3/mapa0.pl



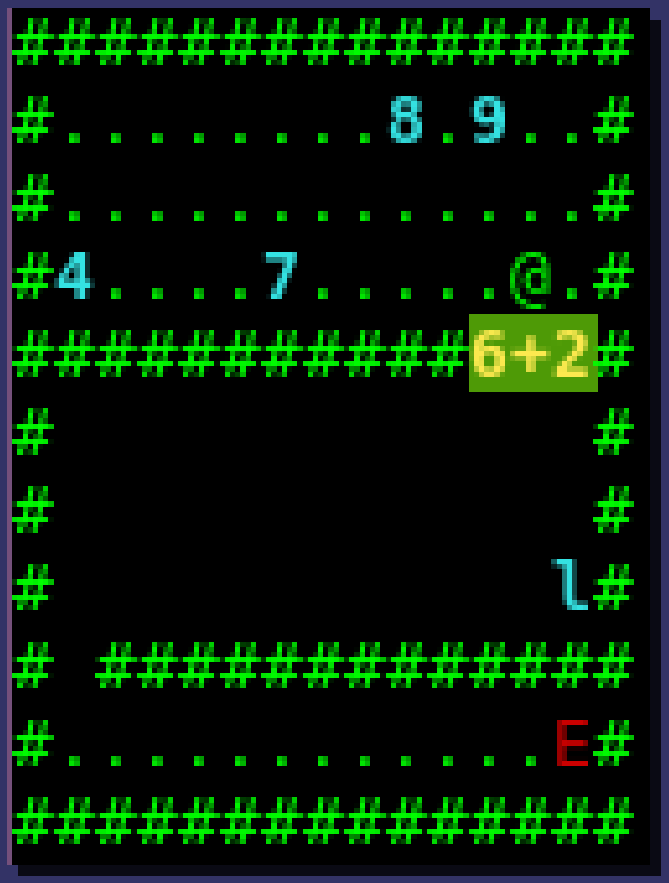
No os olvidéis del mapa1...

maps/fase3/mapa1.pl



¿Os atrevéis con el mapa7?

maps/fase3/mapa7.pl




```
:-dynamic objetoBuscar/1.

do(...):- ... see(normal,down,'+'),
                see(normal, down-left,Op1),
                see(normal, down-right,Op2),
                Suma is Op1+Op2,
                retractall(objetoBuscar(_)),
                assert(objetoBuscar(Suma)).


do(...):- ... objetoBuscar(X), see(normal, right, X).
```

¿Y con el mapa6?

maps/fase3/mapa6.pl