INGENIERIA MULTIMEDIA

PRÁCTICAS

2016-2017

SESIÓN 6

INDETERMINISMO

Comportamientos que cambian en ejecución

PREDICADOS DINÁMICOS De PROLOG

¿Qué son?

¿Para qué sirven?

ESTADOS FINITOS

¿Qué son?

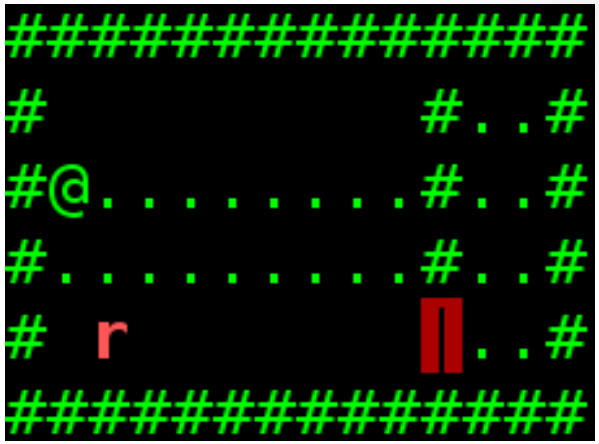
¿Cómo se usan?

Resolver mapas Fase3

MATEMÁTICAS

Este mapa es indeterminista…???

maps/fase3/mapa0.pl



Descargar el script launch.sh: https://logica.i3a.ua.es/downloads/launch.zip

$ ./launch 10 maps/fase3/mapa0.pl sol.pl r

```
$ ./launch 10 maps/fase3/mapa0.pl sol.pl r
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%)
```

INDETERMINISMO, ¿ Qué es ?

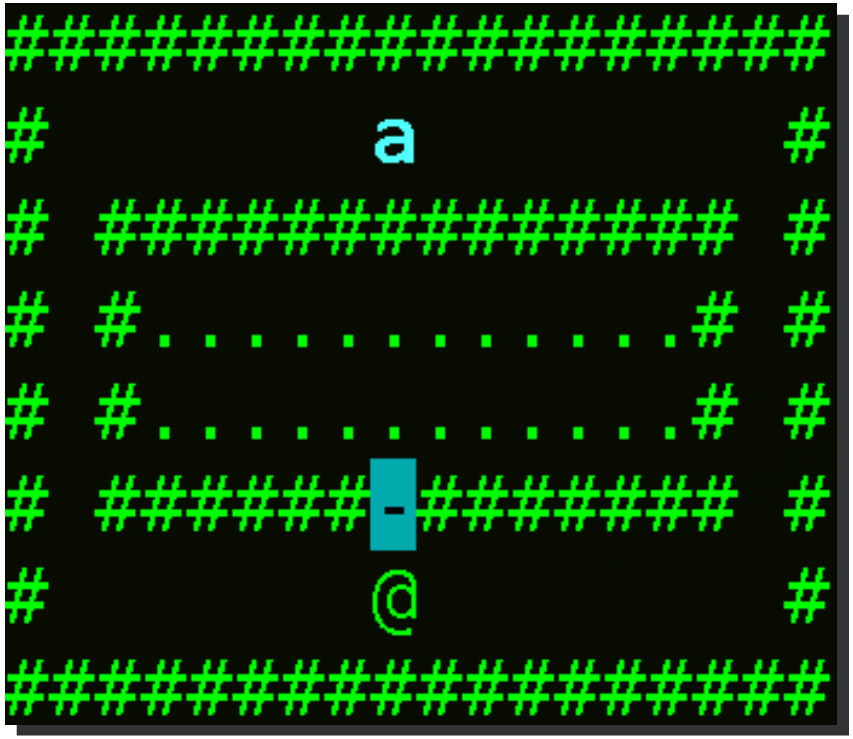
Mapas de Fase 3, Fase 4

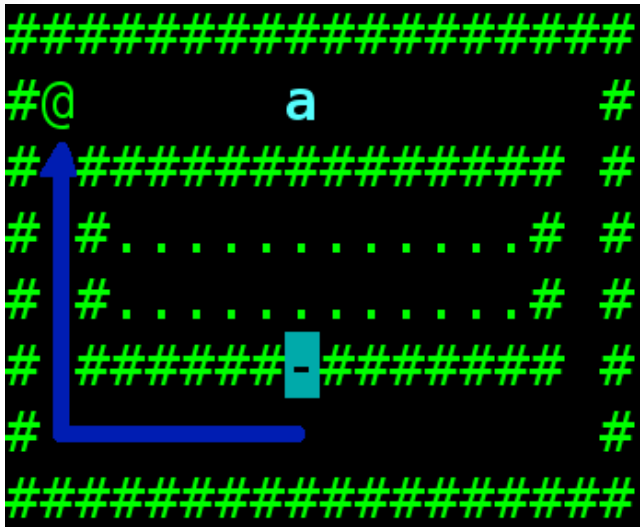
En cada ejecución de un mapa:

- Un objeto puede estar en **distinta posición.**
- Un enemigo puede **comportarse diferente.**
- Plman podría **empezar** en un sitio diferente.
- Los cocos pueden **aparecer / desaparecer.**

Este mapa es determinista pero … con condiciones adversas

/maps/fase3/mapa1.pl



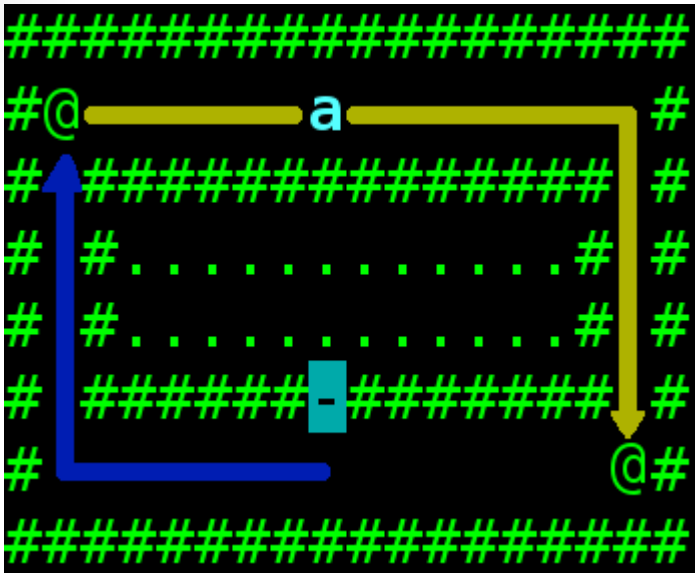



```
:- use_module('pl-man-game/main').

s(D,O) :- see(normal, D, O).
m(D)   :- doAction(move(D)).

r :- s(left, ' '),   m(  left).
r :- s( up, ' '),    m(   up).
r :-                 m(right).
```

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

Modificamos código …

Usamos subreglas…




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

s(D,O) :- see(normal, D, O).
m(D)   :- doAction(move(D)).
 g(D)   :- doAction(get(D)).

r_arriba :-  s( right, ' ' ),  m(   right).
r_arriba :-  s( right, 'a ' ),  g(    right).
r_arriba :-  s( down, '  '),  m( down).


r_abajo :-  s( left, '  '),  m( left).
r_abajo :-  s(  up, '  '),  m(  up).


r :-  estoyArriba, r_arriba.
r :-  estoyAbajo, r_abajo.
```

Estos predicados ?????

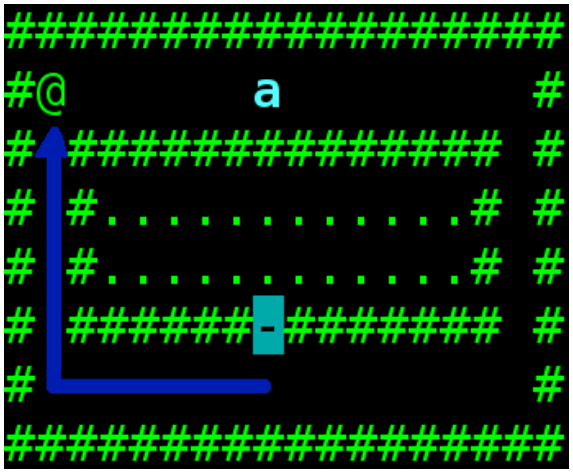
ponemos como un HECHO → dónde está Plman

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

s(D,O) :- see(normal, D, O).
m(D)   :- doAction(move(D)).
g(D)   :- doAction(get(D)).

r_arriba :- s( right, ' '), m( right).
r_arriba :- s( right, 'a '), g( right).
r_arriba :- s( down, ' '), m( down).


r_abajo :- s( left, ' '), m( left).
r_abajo :- s( up, ' '), m( up).


r :- estoyArriba, r_arriba.
r :- estoyAbajo, r_abajo.
```

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

s(D,O) :- see(normal, D, O).
m(D)   :- doAction(move(D)).
g(D)   :- doAction(get(D)).

estoy(abajo).

r(arriba) :- s( right, ' '), m( right).
r(arriba) :- s( right, 'a '), g( right).
r(arriba) :- s( down, ' '), m( down).


r(abajo) :- s( left, ' '), m( left).
r(abajo) :- s( up, ' '), m( up).

% regla principal

r :- estoy(D), r(D).
```

¿ Crees que falta otro HECHO para “arriba” ?

Añade y comprueba



:- use_module(‘pl-man-game/main’).

estoy(abajo).
estoy(arriba).

r(arriba) :- s(right, ’ ’), m(right).
r(arriba) :- s(right, ’a ’), g(right).
r(arriba) :- s(down, ’ ’), m(down).

r(abajo) :- s(left, ’ ’), m(left).
r(abajo) :- s(up, ’ ’), m(up).

% regla principal

r :- estoy(D), r(D).

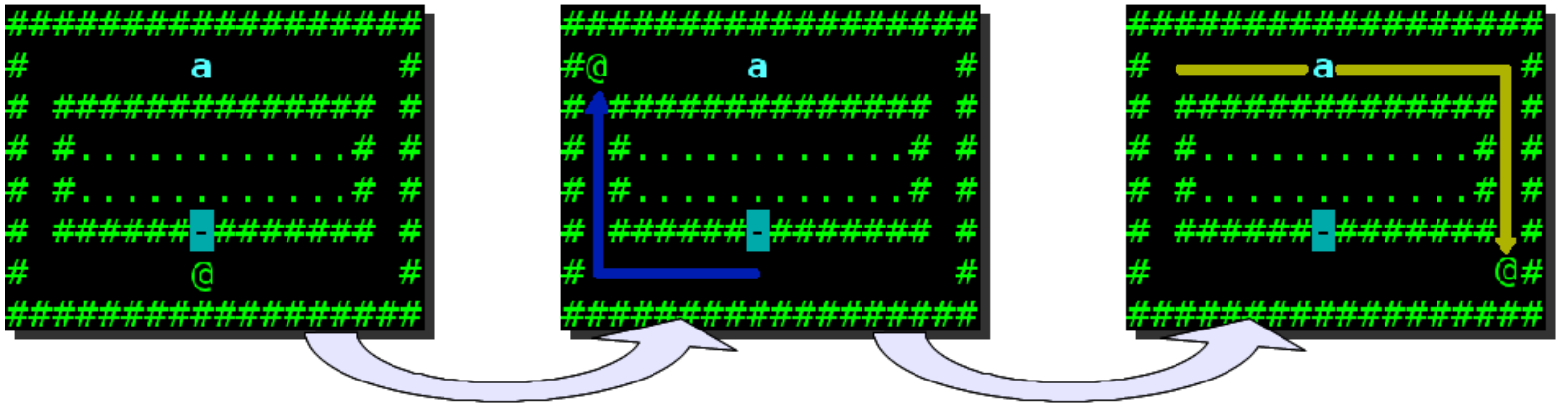
Sería conveniente:

Cuando esté abajo → hacer reglas **r(abajo)** → añadir hecho **estoy(abajo).**

Cuando esté arriba → hacer reglas **r(arriba)** → añadir hecho **estoy(arriba).**

Cómo se puede hacer….

*Se **añade / elimina el hecho** en tiempo de ejecución → **Predicado Dinámico**.*



Añadir estoy(abajo)

Eliminar estoy(abajo)
Añadir estoy(arriba)

```
r :- estoy(E), r(E).
```

```
estoy(abajo).

r :- estoy(E), r(E).
```

```
estoy(arriba).

r :- estoy(E), r(E).
```

Predicados dinámicos en Prolog

→ Permiten **añadir / eliminar** cláusulas (hechos, reglas) en la BC durante la ejecución**.**

→ Sólo trabajaremos con **hechos**

→ Se declaran mediante la directiva **dynamic/1**

:- dynamic predicado / n

Predicado ISO_Standard para **añadir** HECHOS dinámicos**:**

assert/ 1, assert(hecho).

Predicados ISO_Standard para **eliminar** HECHOS dinámicos**:**

retractt/ 1, retract(hecho).

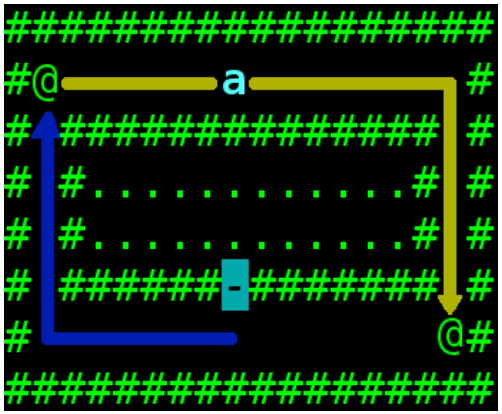
Elimina un hecho de la BC. Si hecho no existe→ fracaso.

retractall/1, retractall(hecho).

Elimina todos los hechos que unifiquen con **HECHO** de la BC.

Siempre tiene éxito, incluso cuando ningún hecho unifica con **HECHO**.

Solución con Predicados dinámicos




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

:- dynamic estoy/1.

%añades el hecho inicial, por donde quieres empezar

estoy(abajo).

r(arriba) :-  s( right, ' '),   m( right).
r(arriba) :-  s( right, 'a '),  g( right).
r(arriba) :-  s( down, '  '),  m( down).


r(abajo) :-  s( left, '  '),  m( left).
r(abajo) :-  s( up, '  '),  m(  up).
r(abajo) :-  m(right),
             retract(estoy(abajo)),
              assert(estoy(arriba)).

% regla principal

r :-  estoy(D), r(D).
```

Descuidos al usar 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).
```

Fallos al usar retract y assert

```
r(abajo)  :- retract(estoy),
             assert(estoy(arriba)).
```

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

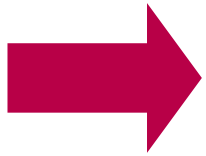
Se intenta borrar `estoy/0`, que no existe. `retract` fracasa.

```
r(abajo)  :- retract(estoy(arriba)),
             assert(estoy(arriba)).
```

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

Se intenta borrar `estoy(arriba)`, que no existe
(existe `estoy(abajo)`). `retract` fracasa.

```
r(abajo)  :- assert(estoy(arriba)).
```



```
estoy(abajo).
estoy(arriba).
```

`assert` añade un hecho, pero no borra el anterior.
La Base de Conocimientos ahora tiene 2 hechos `estoy/1`.

Comprobar fallos escribiendo "mensajes" con write/1 y writeln/1

```
r :- estoy(E),
     write('Estoy '),
     writeln(E),
     r(E).
```

```
Estoy abajo
Estoy abajo
Estoy abajo
Estoy abajo
Estoy arriba
Estoy arriba
Estoy arriba
```

Para saber si la ejecución pasa por un punto o no

```
r(abajo)  :- write('Lanzamos retract...'),
             retract(estoy),
             write('Exito. Lanzamos assert...'),
             assert(estoy(arriba)).
```

```
Lanzamos retract...1 ADVERTENCIA: L
a regla de control de tu personaje
ha fracasado!
```

retract *se lanza y fracasa.*
La ejecución nunca llega al segundo ***write.***

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 regla

Script launch guarda logs de las que fallan

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

$ ./plman mapa.pl solucion.pl regla -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.

Más…

La regla de control se lanza en cada ciclo.

PLMan tiene una regla especial de inicialización que se lanza **sólo 1 vez**, al empezar.

Ejemplo

$ ./plman mapa.pl sol.pl rrcc

rrcc se lanzará en cada ciclo de ejecución.

rrcc(init) se lanzará 1 sola vez, al principio.

Es útil por si se quiere lanzar alguna regla que inicialice los hechos al comienzo.

Estado: conjunto de subreglas que resuelven un comportamiento de Plman

Resolución de un mapa usando estados y predicados dinámicos:

- Estudiamos los diferentes comportamientos de Plman en el mapa.
- Para cada comportamiento Plman definir sub-reglas con diferentes acciones
- Cada conjunto de sub-reglas conformará un estado.
- Las sub-reglas aparecerán en el cuerpo de la regla principal.
- Cuando Plman cambia de comportamiento se cambia de estado.
- El **estado** se declara en el programa mediante **un hecho** con predicado **dinámico** con el fin de que su/s argumento/s pueda variar durante la ejecución siempre que sea conveniente.

¿Cuántos estados se pueden/deben declarar para resolver un mapa?

Tantos como se estimen oportunos según los diferentes comportamientos de Plman.

En cada momento de la resolución se debe saber el estado en que se encuentra Plman.

Modulariza la solución usando estados:

- **Al**macena en la BC **el estado** en el que se encuentra Plman
- Para **cada estado** crea **subreglas** distintas, una para cada estado.
- C**ambia de estado** cuando sea conveniente.

```
% Predicado dinámico para almacenar el estado
:- dynamic estado/1.
% Cláusulas de mi regla
miregla :- estado(ESTADO), ....
```

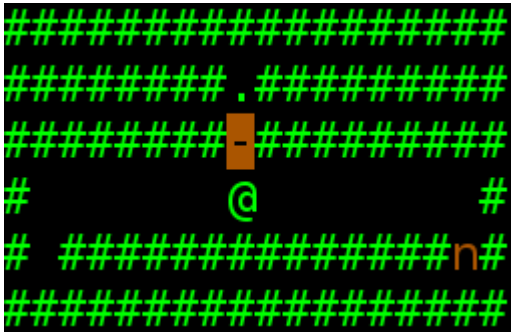
En cada ciclo de ejecución la regla principal se debe inicializar

miregla(init) **se ejecuta** sólo 1 vez al comienzo de la ejecución.

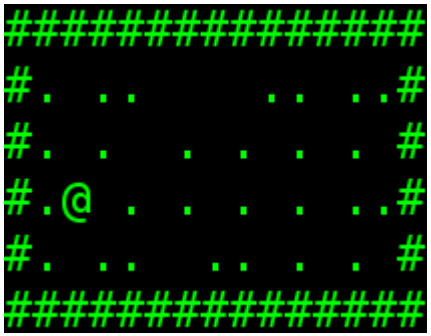
(útil por si se quiere lanzar alguna regla que inicialice los hechos al comienzo)

miregla 1 vez en cada ciclo de ejecución.

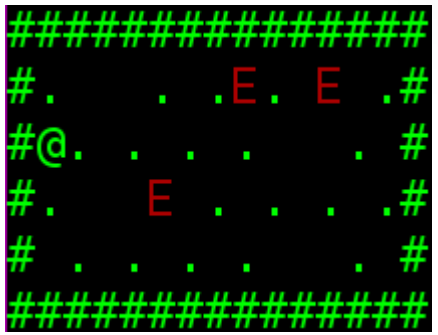
HACER MAPAS ejemplos de Fase 3 (plman/maps/fase3)



mapa2

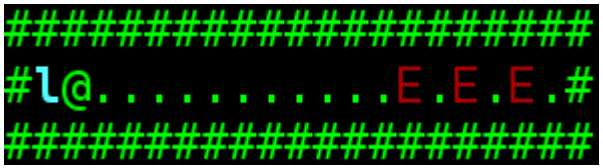


mapa3



mapa4

mapa6



mapa5

El primero que los resuelva, O el que tenga menor nº de cláusulas en total , gana….

SUGERENCIAS

Recordad modularizar vuestro código

Cread reglas que os permitan gestionar vuestros predicados dinámicos, encargándose de modificar los valores de los hechos.

Cometeréis menos errores y más fáciles de localizar.

Programar bien

es resolver problemas

más difíciles

de forma

más fácil