PRÁCTICAS de Matemáticas 1

2023-2024

6

Desde la Fase 2 se pueden programar las acciones de Plman condicionadas a los objetos del mapa.

Esto nos permitirá modularizar el código

- ¿Plman lleva objeto? ¿Qué objeto lleva?
- Predicado: **havingObject/n**
- Modulariza el código con subreglas
- En la explicación usaremos los mapas :

  plman/maps/fase2/**mapa0.pl** y

  plman/maps/fase2/**mapa00.pl**
- Resuelve otros mapas de entrenamiento de la colección plman/maps/fase2

- **Rutina** para resolver mapas: descarga/ver mapa/ejecutar solución.

Rutina : Descargar / ver mapa / ejecutar la solución

1º Abre terminal Linux / Ubuntu, ponte en Escritorio

$ cd Escritorio

2º Descarga la carpeta plman (Moodle) > **guárdala** en Escritorio.

3º Edita un fichero (con extensión **.pl**): **$ gedit sol.pl &**

Guárdalo en la carpeta: **Escritorio/plman**

4º Escribe en sol.pl

```
:- use_module('pl-man-game/main').
do(move(none)).
```

5º Descarga un mapa, de **mapas_init**, por ej: mapa1_init.pl

6º Comando para **ver mapa / ejecutar la solución**

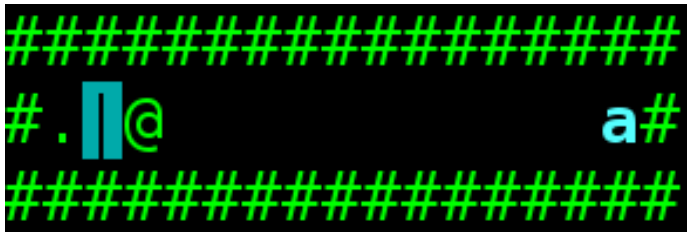
$ ./plman mapa.pl sol.pl

Abortar ejecución: **Esc + X**
Salir al finalizar ejecución: **X**

Abre terminal y descarga el mapa:

plman/maps/fase2/**mapa0.pl**

Resuélvelo…



En algunos mapas hay pasillos con espacios en blanco en los que Plman siempre ve lo mismo, en ambas direcciones, right - left. Aparece el movimiento TIC-TAC.

Si hay objetos, esta situación se puede solucionar introduciendo un nuevo predicado en el cuerpo de las reglas, llamado **havingObject/n**

Dicho predicado permite a Plman realizar **acciones** que estarán **condicionadas** a lo que **lleva.**

¿ Plman lleva objeto ? havingObject/n con n = 0

PREDICADO: **havingObject/0**

El predicado tiene **éxito** cuando Plman **lleva** un objeto (el que sea).

Fracasa si no lo lleva.

EJEMPLO

```
do(move(down)) :- havingObject.
do(move(up)) :- not(havingObject).
```

Plman se moverá abajo si lleva un objeto (el que sea)

Plman se moverá arriba si no lleva objeto

Acompañado del predicado see/3…

```
do(move(down)) :- havingObject, see(normal, down, ' ' ).

do(move(up)) :- not(havingObject), see(normal, up, ' ' ).
```

Plman se moverá abajo si lleva un objeto (el que sea) y ve espacio abajo

Plman se moverá arriba si no lleva objeto y ve espacio arriba

¿ Qué objeto lleva ? havingObject/n con n = 1

!Ojito! Tenemos en cuenta que Plman sólo puede llevar un objeto.

Si queremos que Plman **cambie** su comportamiento en **función del objeto** que lleve, usamos:

```
havingObject(appearance(OBJ
))
```

OBJ: objeto que lleva Plman
>> el predicado tiene **éxito** si Plman lleva el **objeto referido en el argumento (OBJ)**

EJEMPLO: Queremos que Plman use el objeto 'a' cuando vea la puerta a la izquierda
Si lo lleva encima la regla se ejecutará, si no, no, aunque vea la puerta.

```
do(use(left))  :- havingObject(appearance(a)), see(normal, left,
'|').
```

¿ Cómo elijes havingObject/n

1º : Si las acciones de Plman NO dependen del objeto que lleva, pero es necesario tener en cuenta si lleva, o no, el objeto, se usa:

havingObject/0

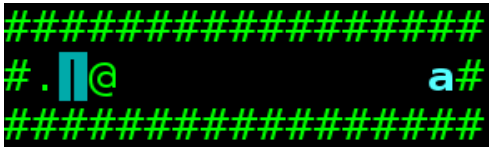
2º : Si las acciones de Plman Sí dependen del objeto que lleva se usa :

havingObject/1

Usamos havingObject para resolver plman/maps/fase2/mapa0.pl

>> Plman se mueve a la derecha cuando no lleva objeto y ve espacio:

do(move(right)) :- not(havingObject), see(normal,right,' ').



>> Plman coge el objeto 'a':

do(get(right)) :- see(normal,right,'a').

>> Plman se mueve a la izquierda cuando lleva objeto (llave 'a') y ve espacios

do(move(left)) :- havingObject(appearance(a)), see(normal,left, ' ').

>> Plman usa el objeto 'a' a la izquierda, si lo lleva, y ve una puerta:

do(use(left)) :- havingObject(appearance(a)), see(normal,left,'|').

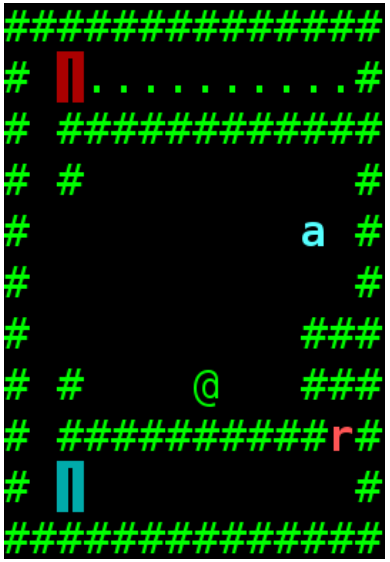
OJO: en este mapa no es necesario el argumento ¨appearance(a)¨ ya que sólo hay un objeto, por lo que las dos últimas reglas se podrían escribir:

do(move(left)) :- havingObject, see(normal,left, ' ').

do(use(left)) :- havingObject, see(normal,left,'|').

2º mapa : Descarga plman/maps/fase2/**mapa00.pl**

Resuélvelo…



plman/maps/fase2/mapa00.pl

Pista: considera las siguientes situaciones y define reglas para cada una.

1º Cuando Plman **no lleve** objeto

2º Cuando Plman **lleve** el objeto: **llave 'a'**

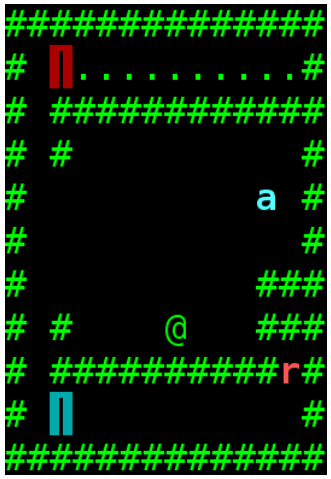
3º Cuando Plman **lleve** el objeto: **llave 'r'**

1° Cuando Plman **no lleve** objeto

```
%% 1° - Reglas para cuando Plman no lleva objeto

do(move(right))   :-  not(havingObject),   s(right, ' ').
do(  get( left))  :-  not(havingObject),   s( left, 'a').
do( get(   up))   :-  not(havingObject),   s(   up, r).
do( move(   up)).

...
```



plman/maps/fase2/mapa00.pl

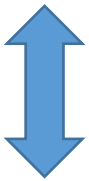
Para abreviar he usado la regla:

s(DIR, OBJ) :- see(normal, DIR, OBJ).

Reescribimos las reglas usando subreglas

```
%% 1 Reglas para cuando Plman no lleva objeto

do(move(right))   :- not(havingObject),   s(right, ' ').
do(  get( left))  :- not(havingObject),   s( left, 'a').
do( get(   up))   :- not(havingObject),   s(   up, r).
…
```



```
do(ACT) :- not(havingObject), do1(ACT).

%% Subreglas

do1(move(right))   :- s(right, ' ').
do1( get(   left))  :- s(   left, a).
do1( get(   up))    :- s(   up, r).
```

La variable **ACT será una de las acciones**
- > move(right))
- > get(left))
- > get(up))

Dependiendo de cuál sea la regla do1/1 que se ejecute (de arriba/abajo)

2º Cuando Plman lleva llave 'a'. Usamos havingObject(appearance(OBJ))

Reescribimos las reglas usando subreglas

```
%% 2º Reglas para cuando Plman lleva el objeto 'a'

do(use(right))  :-  havingObject(appearance(a)),  s(right, '|').
do(drop( left))  :-  havingObject(appearance(a)),  s(  up, 'r').
do(move(right)) :-  havingObject(appearance(a))  ,s( down,
'#').
....

do(ACT) :- havingObject(appearance(a)), do2(ACT).

%% Subreglas

do2(use(right))     :-  s(right, '|').
do2(drop( left))    :-  s(  up, r).
do2(move(right))  :-  s( down, '#').
...
```

plman/maps/fase2/mapa00.pl

2º Cuando Plman lleva llave 'r'. Usamos havingObject(appearance(OBJ))

Reescribimos las reglas usando subreglas

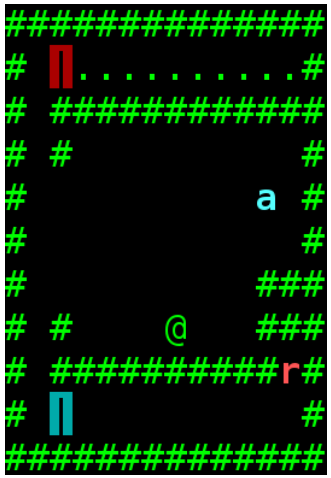
```
%% Reglas para cuando lleva el objeto 'r'

do(move(right))   :- havingObject(appearance(r)), s(right, '.').
do(move( left))   :- havingObject(appearance(r)), s( left, ' ').
do(move( left))   :- havingObject(appearance(r))  s( left, 'a').
...
```

```
do(ACT) :- havingObject(appearance(r)), do3(ACT).

%% Subreglas

do3(move(right))  :-  s(right, '.').
do3(move( left))  :-  s( left, ' ').
do3(move( left))  :-  s( left, 'a').
...
```



plman/maps/fase2/mapa00.pl

El código quedaría así.. (está incompleto)

```
%% Regla principal que lanza subreglas

(1) do(ACT) :- not(havingObject),                do1(ACT).
(2) do(ACT) :- havingObject(appearance(a)), do2(ACT).
(3) do(ACT) :- havingObject(appearance(r)),  do3(ACT).

%% Subreglas para cuando Plman no lleva objeto
do1(move(right))  :- s(right, ' ').
do1( get(   up))    :- s(   up, r).

%% Subreglas para cuando lleva 'a'
do2(use(right))     :- s(right, '|').
do2(drop( left))    :- s(   up, r).

%% Subreglas para cuando lleva 'r'
do3(move(right))  :- s(right, '.').
do3(move( left))   :- s( left, ' ').
…
```

Explicación:

Sabemos q Prolog recorre código de arriba abajo, ejecutando la primera regla que tiene éxito. Cuando Plman comienza no lleva objeto, luego se ejecuta do/1 de la regla (1). La ejecución se posiciona en el conjunto de subreglas do1/1. De éstas, de arriba/abajo se ejecutará la que tenga éxito. La variable ACT tomará el valor de la acción de dicha regla, que será la acción que realice do/1 (regla (1)). Si todas fracasan, fracasa (1). Mientras que Plman no lleve objeto estará en la regla (1).

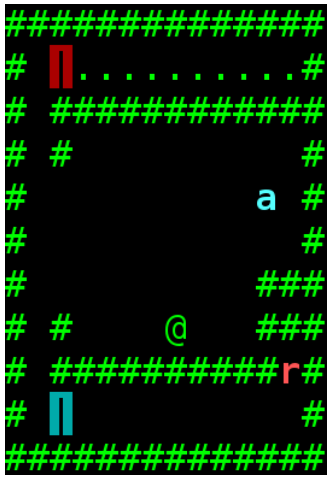
Si Plman lleva un objeto, 'r' por ej., la regla (1) fracasa, la (2) tb, luego pasará a ejecutar la regla (3). Esta regla manda la ejecución al conjunto de subreglas do3/1. Al igual que antes, la subregla de do3/1 que se ejecutará será la que tenga éxito, empezando siempre de arriba/abajo.

2ª versión del Código usando Subreglas y una variable para los objetos

```
%% Regla principal que lanza subreglas
do(ACT) :- not(havingObject),                  do1(ACT).
do(ACT) :- havingObject(appearance(Obj)), do2(Obj, ACT).

%% Reglas para cuando no lleva objeto
do1(move(right))     :- s(right, ' ').
do1( get(   up))        :- s(   up, 'r').

%% Reglas para cuando lleva 'a'
do2(a,  use(right))       :- s(right, '|').
do2(a, drop( left))       :- s(   up, 'r').
....
%% Reglas para cuando lleva 'r'
do2(r, move(right))     :- s(right, '.').
do2(r, move( left))       :- s( left, ' ').
...
```



plman/maps/fase2/mapa00.pl

Fijaros que ahora el predicado do2/2 tiene dos argumentos, el objeto y la acción y dependiendo del objeto que lleve entrará por un conjunto de reglas u otro

Más explicación en siguientes transpa …

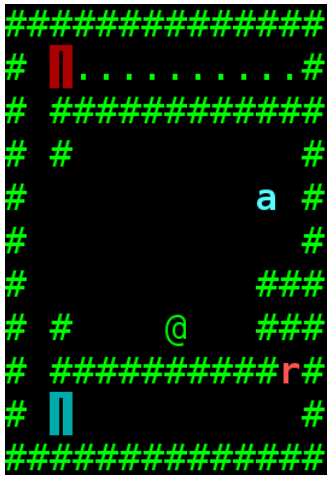
EXPLICACIÓN DE LAS REGLAS

%% Regla principal

(1) do(ACT) :- not(havingObject), do1(ACT).
do(ACT) :- havingObject(appearance(**Obj**)), do2(**Obj**, ACT).

Cuando Plman no lleve objeto será cierta la regla **(1)** y se ejecutarán las subreglas do1(ACT).

La variable ACT será la acción determinada por las subreglas de do1/1

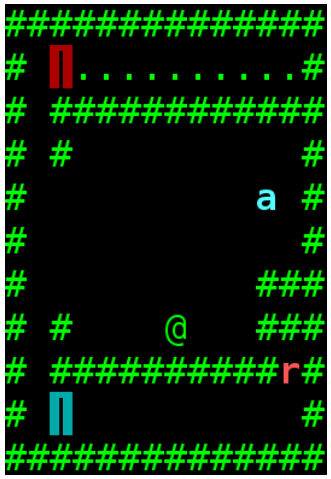


plman/maps/fase2/mapa00.pl

EXPLICACIÓN DE LAS REGLAS

%% Regla principal

```
    do(ACT) :- not(havingObject),                do1(ACT).
(2) do(ACT) :- havingObject(appearance(Obj)), do2(Obj, ACT).
```



plman/maps/fase2/mapa00.pl

Cuando Plman haya cogido un objeto (lo hará con las reglas definidas con el predicado do1/1), tendrá en cuenta el objeto que haya sido.
La variable Obj se instanciará a dicho objeto.

El predicado do2/2 ejecutará las reglas correspondientes al objeto de Obj.
Si ha cogido 'a' ejecutará, por ejemplo,

```
do2(a,  use(right))      :- s(right, '|').
```

Consejo para resolver mapas a partir de la Fase 2:

Si tienes la posibilidad de modularizar el código usando el predicado havingObject/n te recomendamos que lo hagas ya que te será más fácil elaborar la solución.

Ventaja: cuando alguna situación (conjunto de reglas) no te funcione (por ejemplo, en el mapa00.pl, cuando Plman lleve el objeto a) la revisarás y modificarás sin tener en cuenta las otras. De esta forma sólo te centras en cambiar el conjunto de reglas que has declarado para esa situación y no “estropeas” las que te funcionan bien.