Práctica 2 - Planificación Clásica

Técnicas de los Sistemas Inteligentes

David Gil Bautista

45925324M

Análisis del problema

La complejidad del problema reside en crear un planificador que permita que un agente reactivo, con los sensores que queramos, se mueva por un mundo virtual con la finalidad de llegar a conseguir un objetivo que fijaremos.

Para el problema podemos suponer un mundo en dos dimensiones, en las que nuestro agente se podrá ir moviendo por diferentes casillas y realizando tres movimientos; avanzar, girar hacia la izquierda y hacia la derecha. Contaremos también con una serie de personajes y objetos situados en el mundo con los cuales tendremos que interactuar, siendo algunas de estas interacciones el objetivo a cumplir por nuestro agente.

Definición del dominio del problema inicial

Lo primero que definimos en el problema serán las características que presenta nuestro agente, así como los objetos y personajes que apareceran en el mundo.

Nuestro agente tendrá una orientación {Norte, Sur, Oeste, Este}, el agente y los personajes serán de tipo persiana (persona), y una persiana y un objeto será localizable en una zona.

```
(define (domain ejercicio1-domain)
  (:requirements :strips :equality :typing)
  (:types
    orientacion
    persiana objeto - localizable
    personaje jugador - persiana
    zona

  )
```

Podemos establecer relaciones entre los distintos objetos que tenemos, y eso es lo que llamaremos predicados.

```
(:predicates
    ;Jugador no tiene objeto
    (manoVacia ?j - jugador)

    ;Jugador tiene un objeto

    (manoLlena ?j - jugador ?o - objeto)
```

```
    ;Localizacion
    (at ?l - localizable ?z - zona)

    ;Orientacion
    (orientado ?p - jugador ?o - orientacion)

    ;Personaje tiene objeto
    (tener ?p - personaje)

    ;Planificar ruta
    (ruta ?a - zona ?b - zona ?l - orientacion)
)
```

Dichos predicados nos permiten obtener el estado actual del mundo, y gracias a ellos podremos planificar para tomar acciones.

Para comprender como funciona un predicado podemos verlo de la misma forma que lo hace en el análisis sintáctico.**

(orientado jugador orientación) : Un jugador tiene una orientación.

(ruta zona1 zona2 orientación) : Hay un camino con una orientacion desde la zona 1 hasta la zona 2.

Para el primer ejercicio, las acciones son las más básicas, podremos mover nuestro agente, coger objetos y entregarselos a los distintos personajes. Para no saturar la memoria de código tan sólo mostraré una acción y explicaré las partes que tiene.

```
(:action IR
  :parameters (?j - jugador ?l - orientacion ?z - zona ?z2 - zona)
  :precondition (and (orientado ?j ?l) (at ?j ?z) (ruta ?z ?z2 ?l))
  :effect (and (not (at ?j ?z)) (at ?j ?z2))
)
```

Los parámetros que recibe esta acción deben ser el jugador que se moverá, una orientación y dos zonas, en las que se desarrolla la acción de moverse.

Para que se lleve a cabo la acción hay una serie de condiciones que deben cumplirse: El jugador debe poseer la orientación que se ha pasado por parámetro, debe estar en una de las posiciones y por último debe haber un camino posible entre esa posición con esa orientación hasta la siguiente posición.

El efecto que produce el movimiento será dejar de estar en la primera posición, lo que equivale a negar estar en esa posición, y desplazarse a la siguiente posición, creando un hecho en el que nuestro jugador ha cambiado de zona.

Para el resto de funciones no veo necesario explicar el porqué de cada parámetro y su condición, puesto que acciones básicas en las que la mayor dificultad es saber qué pasaría si se produce un giro hacia la izquierda cuando el agente se encuentra mirando hacia el norte.

Ejercicio 1

Para este ejercicio hemos definido el dominio anterior y se nos pide crear un mapa de 25 posiciones en las que coloquemos 5 personajes y al menos 5 objetos.

Lo primero que haremos será instanciar los objetos que tenemos.

```
  (:objects
      z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 z12 z13 z14 z15 z16 z17 z18 z19 z20 z21 z22 z23 z24 z25
- zona
      norte sur este oeste - orientacion
      oscar manzana rosas algoritmos oro - objeto
      Princesa Principe Bruja Profesor LeonardoDiCaprio - personaje
      HarryPotter - jugador
  )
```

Como se puede ver, tenemos 25 objetos de tipo zona que representarán las casillas del mundo virtual, los cuatro puntos cardinales como orientación, 5 objetos, 5 personajes y nuestro agente, Harry Potter.

Una vez tenemos nuestros objetos procedemos a darles un poco de información para poder interactuar con ellos.

```
    (ruta z1 z2 este)
    ;...
    (ruta z24 z20 norte)
    (ruta z25 z24 oeste)

    (at HarryPotter z1)
    (orientado HarryPotter sur)

    (at Principe z12)
    (at Princesa z19)
    (at Bruja z21)
    (at Profesor z17)
    (at LeonardoDiCaprio z10)

    (at manzana z6)
    (at oscar z25)
    (at rosas z4)
    (at oro z8)
    (at algoritmos z23)

    (manoVacia HarryPotter)
```

Esta es la parte más tediosa de la implementación, puesto que tener que escribir a mano los caminos disponibles en un mapa de 25 casillas puede hacernos tener un pequeño fallo que impida que algún camino se conecte y que nuestro planificador no sea capaz de encontrar un plan factible.

Una vez hemos inicializado nuestros objetos podemos crear al fin el objetivo que debe buscar nuestro planificador, en este caso será hacer que cada personaje tenga un objeto.

```
  (:goal (AND (tener Princesa)
              (tener Bruja)
              (tener Profesor)
              (tener LeonardoDiCaprio)
              (tener Principe)
         )
  )
```

Como se puede observar, crear un objetivo equivale a hacer que se de un hecho, en este caso será hacer que se cumplan los predicados de *"tener"* con todos los personajes. En caso de haber usado OR, en vez de AND, el objetivo sería entregar un objeto a cualquier personaje.

Para ver como funciona el primer ejercicio mostraré la salida obtenida por terminal, aunque para los siguientes ejemplos **adjuntaré el resultado obtenido para no sobrecargar la memoria**.

(Recomiendo usar: *$ cat belkan/planes/planX.txt*)

```
ff: parsing domain file
domain 'EJERCICIO1-DOMAIN' defined
 ... done.
ff: parsing problem file
problem 'EJERCICIO1-PROBLEMA2' defined
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:   19 into depth [1][2]
                                  18            [1][2]
                                  17            [1]
                                  16            [1][2][3][4][5]
                                  15            [1][2]
                                  14            [1]
                                  13            [1]
                                  12            [1][2][3][4][5]
                                  11            [1][2][3][4][5][6]
                                  10            [1]
                                   9            [1][2]
                                   8            [1][2]
                                   7            [1][2][3][4][5][6][7][8][9][10][11]
                                   6            [1]
                                   5            [1]
                                   4            [1][2][3][4][5][6][7][8][9][10][11]
                                   3            [1]
                                   2            [1]
                                   1            [1]
```

```
                                    0

ff: found legal plan as follows

step    0: IR HARRYPOTTER SUR Z1 Z6
        1: COGER HARRYPOTTER MANZANA Z6
        2: IR HARRYPOTTER SUR Z6 Z11
        3: IR HARRYPOTTER SUR Z11 Z16
        4: IR HARRYPOTTER SUR Z16 Z21
        5: GIRAR-IZQUIERDA HARRYPOTTER SUR
        6: ENTREGAR HARRYPOTTER MANZANA Z21 BRUJA
        7: IR HARRYPOTTER ESTE Z21 Z22
        8: IR HARRYPOTTER ESTE Z22 Z23
        9: COGER HARRYPOTTER ALGORITMOS Z23
       10: IR HARRYPOTTER ESTE Z23 Z24
       11: GIRAR-IZQUIERDA HARRYPOTTER ESTE
       12: IR HARRYPOTTER NORTE Z24 Z20
       13: IR HARRYPOTTER NORTE Z20 Z15
       14: IR HARRYPOTTER NORTE Z15 Z10
       15: GIRAR-DERECHA HARRYPOTTER NORTE
       16: GIRAR-DERECHA HARRYPOTTER ESTE
       17: GIRAR-DERECHA HARRYPOTTER SUR
       18: ENTREGAR HARRYPOTTER ALGORITMOS Z10 LEONARDODICAPRIO
       19: IR HARRYPOTTER OESTE Z10 Z9
       20: IR HARRYPOTTER OESTE Z9 Z8
       21: GIRAR-IZQUIERDA HARRYPOTTER OESTE
       22: COGER HARRYPOTTER ORO Z8
       23: IR HARRYPOTTER SUR Z8 Z13
       24: IR HARRYPOTTER SUR Z13 Z18
       25: GIRAR-DERECHA HARRYPOTTER SUR
       26: IR HARRYPOTTER OESTE Z18 Z17
       27: GIRAR-DERECHA HARRYPOTTER OESTE
       28: IR HARRYPOTTER NORTE Z17 Z12
       29: ENTREGAR HARRYPOTTER ORO Z12 PRINCIPE
       30: IR HARRYPOTTER NORTE Z12 Z7
       31: IR HARRYPOTTER NORTE Z7 Z2
       32: GIRAR-DERECHA HARRYPOTTER NORTE
       33: IR HARRYPOTTER ESTE Z2 Z3
       34: IR HARRYPOTTER ESTE Z3 Z4
       35: GIRAR-DERECHA HARRYPOTTER ESTE
       36: COGER HARRYPOTTER ROSAS Z4
       37: IR HARRYPOTTER SUR Z4 Z9
       38: IR HARRYPOTTER SUR Z9 Z14
       39: IR HARRYPOTTER SUR Z14 Z19
       40: GIRAR-DERECHA HARRYPOTTER SUR
       41: ENTREGAR HARRYPOTTER ROSAS Z19 PRINCESA
       42: GIRAR-DERECHA HARRYPOTTER OESTE
       43: GIRAR-DERECHA HARRYPOTTER NORTE
       44: IR HARRYPOTTER ESTE Z19 Z20
       45: GIRAR-DERECHA HARRYPOTTER ESTE
       46: IR HARRYPOTTER SUR Z20 Z25
       47: GIRAR-DERECHA HARRYPOTTER SUR

       48: COGER HARRYPOTTER OSCAR Z25
```

```
       49: IR HARRYPOTTER OESTE Z25 Z24
       50: IR HARRYPOTTER OESTE Z24 Z23
       51: IR HARRYPOTTER OESTE Z23 Z22
       52: GIRAR-DERECHA HARRYPOTTER OESTE
       53: IR HARRYPOTTER NORTE Z22 Z17
       54: ENTREGAR HARRYPOTTER OSCAR Z17 PROFESOR


time spent:    0.00 seconds instantiating 963 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 170 facts and 363 actions
               0.00 seconds creating final representation with 165 relevant facts, 0 relevant
fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 574 states, to a max depth of 11
               0.00 seconds total time
```

Una vez obtenido dicho plan podemos ver cómo funciona el planificador. A partir de un hecho inicial que declaramos al inicializar el problema, se van dando acciones que provocan nuevos hechos con el fin de obtener el predicado/o predicados de la meta.

En este caso, con un agente que tiene solo una mano y puede portar un único objeto, se dan 54 pasos hasta que consigue entregar los objetos a todos los personajes. En la siguiente imagen se puede observar el mapa usado en el que se ha planificado el problema.

(Dadas la claridad y representación de las imágenes escogidas no veo la necesidad de explicar qué objeto ocupa qué posición, pero en caso de duda se puede consultar la inicialización)

Ejercicio 2

Para este ejercicio se nos pide modificar el anterior añadiendo una serie de costes a las rutas con el objetivo de ejecutar el mismo plan, pero teniendo en cuenta esta vez la mejor y que menos coste presente.

Para representar este coste usaremos una función que llamaremos cada vez que nuestro agente se desplace.

```
(:functions
  (caminoActual)
  (coste ?a ?b - zona)
)
```

Dicha función se incrementa con el desplazamiento entre dos zonas, y en una variable iremos guardando el coste total, con la meta de minimizar el coste de dicho camino. Para usar esta función hemos actualizado la acción IR, añadiendo como efecto el incremento el coste del camino actual con el coste de ese movimiento.

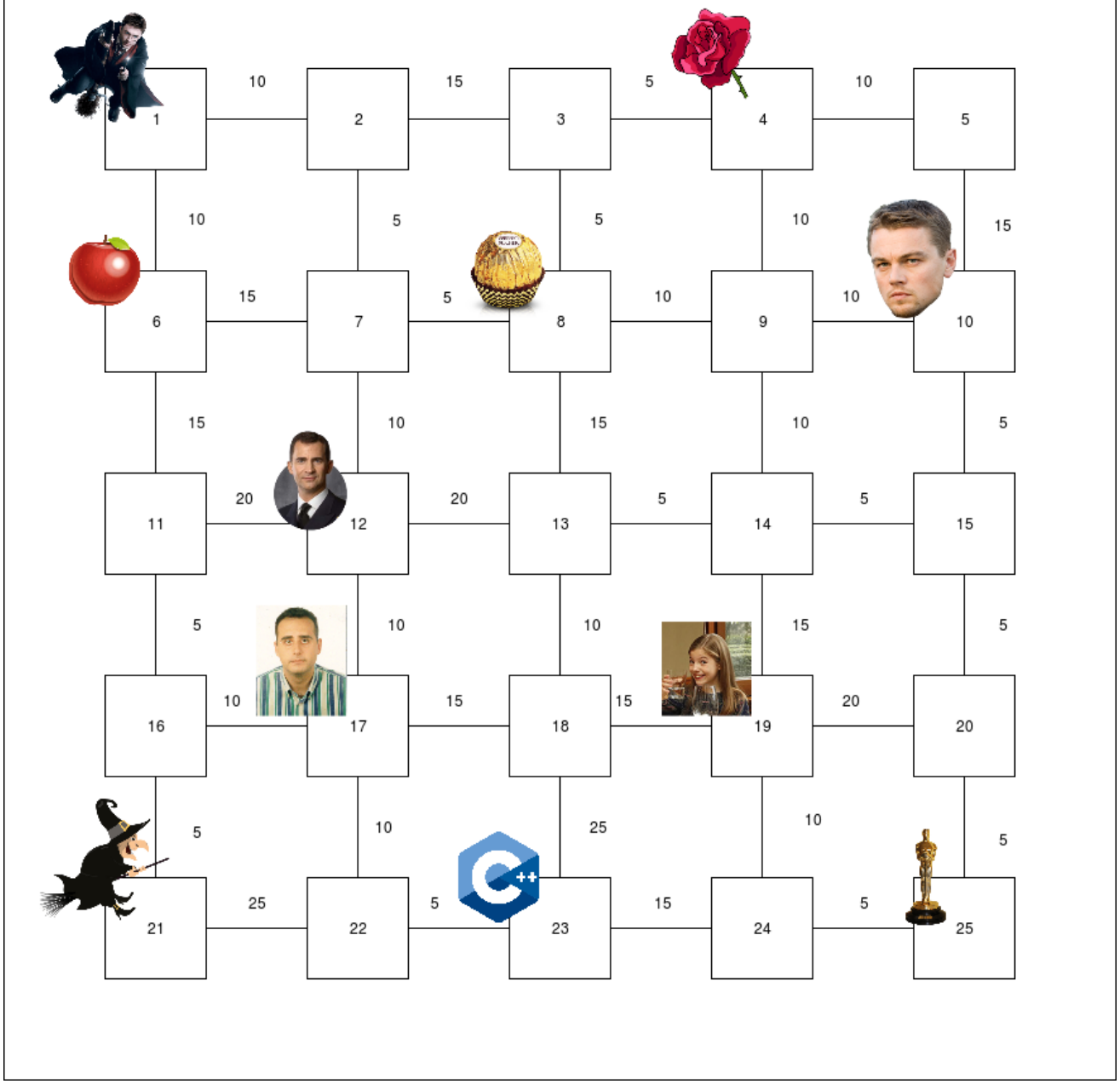
```
(:action IR
  :parameters (?j - jugador ?l - orientacion ?z - zona ?z2 - zona)
  :precondition (and (orientado ?j ?l) (at ?j ?z) (ruta ?z ?z2 ?l))
  :effect (and (increase (caminoActual) (coste ?z ?z2)) (not (at ?j ?z)) (at ?j ?z2))
)
```

Una vez sabemos que debemos minimizar un coste nos planteamos como representar dicho coste, y al igual que con el resto de la información del problema podemos representarlo dado valor al coste que tendrá cada camino entre dos zonas, esto lo haremos en la definición del problema de la siguiente forma.

```
  (= (coste z1 z2) 15)
  ;...
  (= (coste z25 z24) 5)

  (= (caminoActual) 0)
```

Fijaremos los distintos costes que tienen los caminos y el coste inicial.

Usaremos el siguiente mapa de costes.


1
10
2
15
3
5
4
10
5
10
5
5
10
15
6
15
7
5
8
10
9
10
10
15
10
15
10
5
11
20
12
20
13
5
14
5
15
5
10
10
15
10
5
16
10
17
15
18
15
19
20
20
5
10
25
10
5
21
25
22
5
23
15
24
5
25

Ejercicio 3

En este ejercicio debemos añadir una propiedad a las zonas. Una zona podrá ser de tipo Agua, Bosque, Arena, Piedra o Precipicio. También se añadirán dos objetos nuevos, Zapatillas y Bikini, y ahora nuestro agente dispondrá de una mochila que le permita guardar un objeto, aparte del equipado.

Lo primero que debemos hacer es representar la información con nuevos predicados:

```
    ;Mochila esta vacia
    (mochilaVacia ?m - mochila)

    ;Mochila tiene un objeto
    (mochilaLLena ?m - mochila ?o - objeto)

    ;Tipo zona (agua, bosque, piedra, precipici)
    (tipoZona ?z - zona ?t - tipo)
```

Estos predicados nos permiten saber el estado actual de la mochila y el tipo de terreno que tiene una zona. Teniendo la información representada se nos dan una serie de indicaciones sobre el funcionamiento del agente:

1. El agente no puede moverse a un precipicio
2. Podrá moverse a una casilla de tipo agua si equipa el bikini o lo guarda en la mochila
3. Podrá moverse a una casilla de tipo bosque si equipa las zapatillas o las guarda en la mochila

Dadas estas restricciones debemos modificar la acción de IR para evitar que nuestro agente muera en una estrepitosa caída por un precipicio.

```
  (:action IR
    :parameters (?j - jugador ?l - orientacion ?z - zona ?z2 - zona ?t - tipo ?m - mochila)
    :precondition (and (orientado ?j ?l) (at ?j ?z) (ruta ?z ?z2 ?l) (not (= ?t Precipicio))
(tipoZona ?z2 ?t)
                  (or
                    (and (not (= ?t Bosque)) (not (= ?t Agua)))
                    (and (= ?t Bosque) (or (mochilaLLena ?m zapatillas) (manoLlena ?j
zapatillas)))
                    (and (= ?t Agua) (or (mochilaLLena ?m bikini) (manoLlena ?j bikini)) )
                  ))
    :effect (and (increase (caminoActual) (coste ?z ?z2)) (not (at ?j ?z)) (at ?j ?z2))
  )
```

Ahora añadimos dos nuevos parámetros, el tipo de zona y la mochila del agente. Las condiciones para que se produzca la acción deben ser que el tipo de zona a desplazarse no sea un Precipicio y que en caso de ser Bosque, tenga unas zapatillas, en caso de ser Agua, un bikini, o que sea cualquiera que no sea Bosque o Agua.

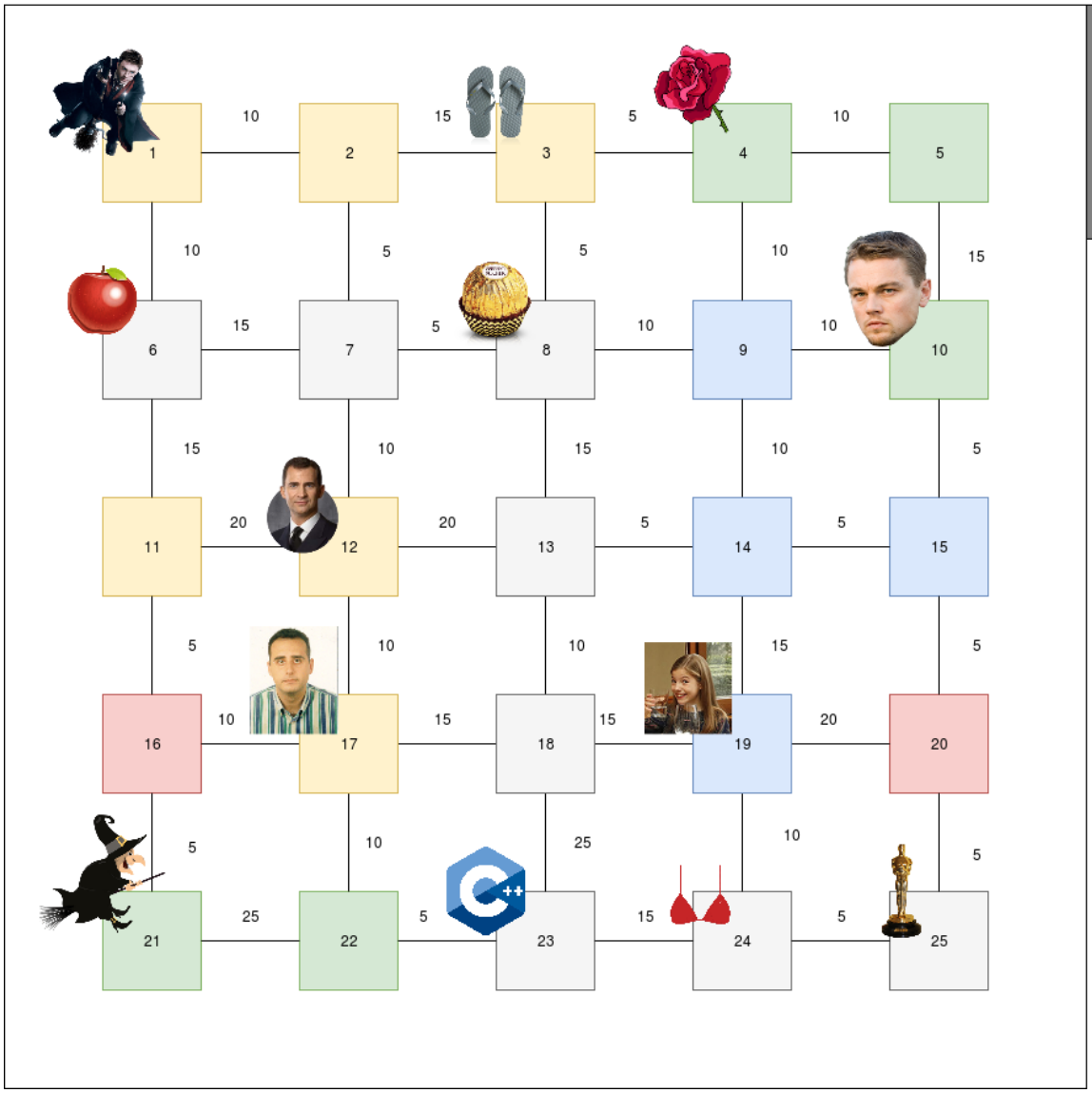
Como también hemos añadido una mochila debemos poder interactuar con ella:

```
  (:action GUARDAR-MOCHILA
    :parameters (?j - jugador ?m - mochila ?o - objeto)
    :precondition (and (manoLLena ?j ?o) (mochilaVacia ?m))
    :effect (and (not (mochilaVacia ?m)) (mochilaLLena ?m ?o) (not (manoLLena ?j ?o)) (manoVacia
?j))
  )

  (:action SACAR-MOCHILA
    :parameters (?j - jugador ?m - mochila ?o - objeto)
    :precondition (and (manoVacia ?j) (mochilaLlena ?m ?o))
    :effect (and (not (mochilaLLena ?m ?o)) (mochilaVacia ?m) (manoLLena ?j ?o) (not (manoVacia
?j)))
  )
```

Para guardar un objeto en la mochila debemos tener equipado el objeto en la mano, una vez guardado el objeto negamos que tenemos equipado el objeto, negamos que la mochila está vacía y creamos el hecho en el que guardamos el objeto en la mochila y el de la mano vacía.

Para sacar un objeto debemos tener las manos vacías. Negamos el hecho de tener la mochila llena, creamos uno indicando que la mochila está vacía, negamos que la mano está vacía y creamos el hecho en el que indicamos que tenemos equipado ese objeto.

Este es el mapa que he usado para los distintos terrenos.



LEYENDA

TIERRA
BOSQUE
AGUA
PRECIPICIO
PIEDRA

Ejercicio 4

En este ejercicio consideramos que la entrega de ciertos objetos a personajes determinados tendrá una puntuación (mostrada en la siguiente tabla).

	Leonardo	Princesa	Bruja	Profesor	Príncipe
Oscar	10	5	4	3	1
Rosa	1	10	5	4	3
Manzana	3	1	10	5	4
Algoritmo	4	3	1	10	5
Oro	5	4	3	1	10

Para conseguir esto debemos tratar con una nueva función que nos indique la puntuación que ha conseguido nuestro jugador:

```
  (:functions
    (puntos ?j - jugador)
  )
```

Una vez tenemos esto debemos actualizar la acción en la que entregamos los objetos, alterando los puntos tal y como muestra la tabla.

```
  (:action ENTREGAR
    :parameters (?j - jugador ?o - objeto ?z - zona ?p - personaje)
    :precondition (and (at ?j ?z) (at ?p ?z) (manoLLena ?j ?o)(not (= ?o bikini)) (not (= ?o
zapatillas)));No entregamos objeto usables
    :effect(and
              (when (and (= ?p LeonardoDiCaprio) (= ?o oscar)) (increase (puntos ?j) 10))
              (when (and (= ?p LeonardoDiCaprio) (= ?o rosas)) (increase (puntos ?j) 1))
              (when (and (= ?p LeonardoDiCaprio) (= ?o manzana)) (increase (puntos ?j) 3))
              ;...

              (when (and (= ?p Principe) (= ?o algoritmos)) (increase (puntos ?j) 5))
              (when (and (= ?p Principe) (= ?o oro)) (increase (puntos ?j) 10))

              (not (manoLLena ?j ?o)) (manoVacia ?j) (tener ?p)
            )
  )
```

Lo primero será añadir que el objeto que se entrega no sea una zapatilla o un bikini, sin hacer esto nuestro personaje entregaba todos los objetos y por culpa de haber entregado uno de estos no podría cumplir el plan. Al igual que en la tabla, debemos establecer todas las relaciones entre objetos y personajes, actualizando para cada uno de ellos el contador de puntos.

Habiendo implementado ya toda la información se nos exige como meta alcanzar 50 puntos, dado que solo hemos colocado 5 objetos y 5 personajes en nuestro mundo solo hay una forma de conseguir la máxima puntuación. Como se puede observar en el archivo adjuntado en la carpeta de planes, nuestro agente consigue alcanzar la máxima puntuación sin extraviarse.

Ejercicio 5

En este ejercicio se nos pide transformar la mochila en el bolsillo mágico de Doraemon. Para hacer esto modificaremos la mochila añadiéndole una capacidad que fijaremos en la inicialización. Para ello crearemos la siguiente función:

```
  (:functions
    (capacidad ?m - mochila)
  )
```

Con esta función actualizaremos el estado de la mochila cada vez que interactuemos con un objeto:

```
  ;Guardar objeto que tiene en la mano en la mochila
  (:action GUARDAR-MOCHILA
    :parameters (?j - jugador ?m - mochila ?o - objeto)
    :precondition (and (manoLLena ?j ?o) (mochilaVacia ?m) (> (capacidad ?m) 0))
    :effect (and (mochilaLLena ?m ?o) (not (manoLLena ?j ?o)) (manoVacia ?j) (decrease
(capacidad ?m) 1)
              (when (= (capacidad ?m) 0) (not (mochilaVacia ?m)))
            )
  )

  ;Sacar objeto de la mochila - Se equipa objeto en la mano
  (:action SACAR-MOCHILA
    :parameters (?j - jugador ?m - mochila ?o - objeto)
    :precondition (and (manoVacia ?j) (mochilaLlena ?m ?o) (< (capacidad ?m) 7))
    :effect (and (not (mochilaLLena ?m ?o)) (manoLLena ?j ?o) (not (manoVacia ?j)) (increase
(capacidad ?m) 1)
              (when (= (capacidad ?m) 5) (mochilaVacia ?m))
            )
  )
```

Para guardar un objeto en la mochila, la mochila debe estar vacía, lo que ahora equivale a que tiene una capacidad de al menos 1 objeto. Guardaremos el objeto creando el hecho mochilaLlena y decrementamos la capacidad de la mochila en uno.

Para sacar un objeto de la mochila, debe haber al menos un objeto en la mochila. Al sacar el objeto lo equipamos en la mano, negamos el hecho mochilaLlena de ese objeto, y aumentamos la capacidad en uno.

Una vez hemos cambiado el dominio debemos inicializar los valores del problema:

```
(= (capacidad mochila) 7)
```

Hecho esto ya podemos ejecutar el planificador y vemos encuentra un plan rápidamente y lo ejecuta a la perfección.

Ejercicio 6

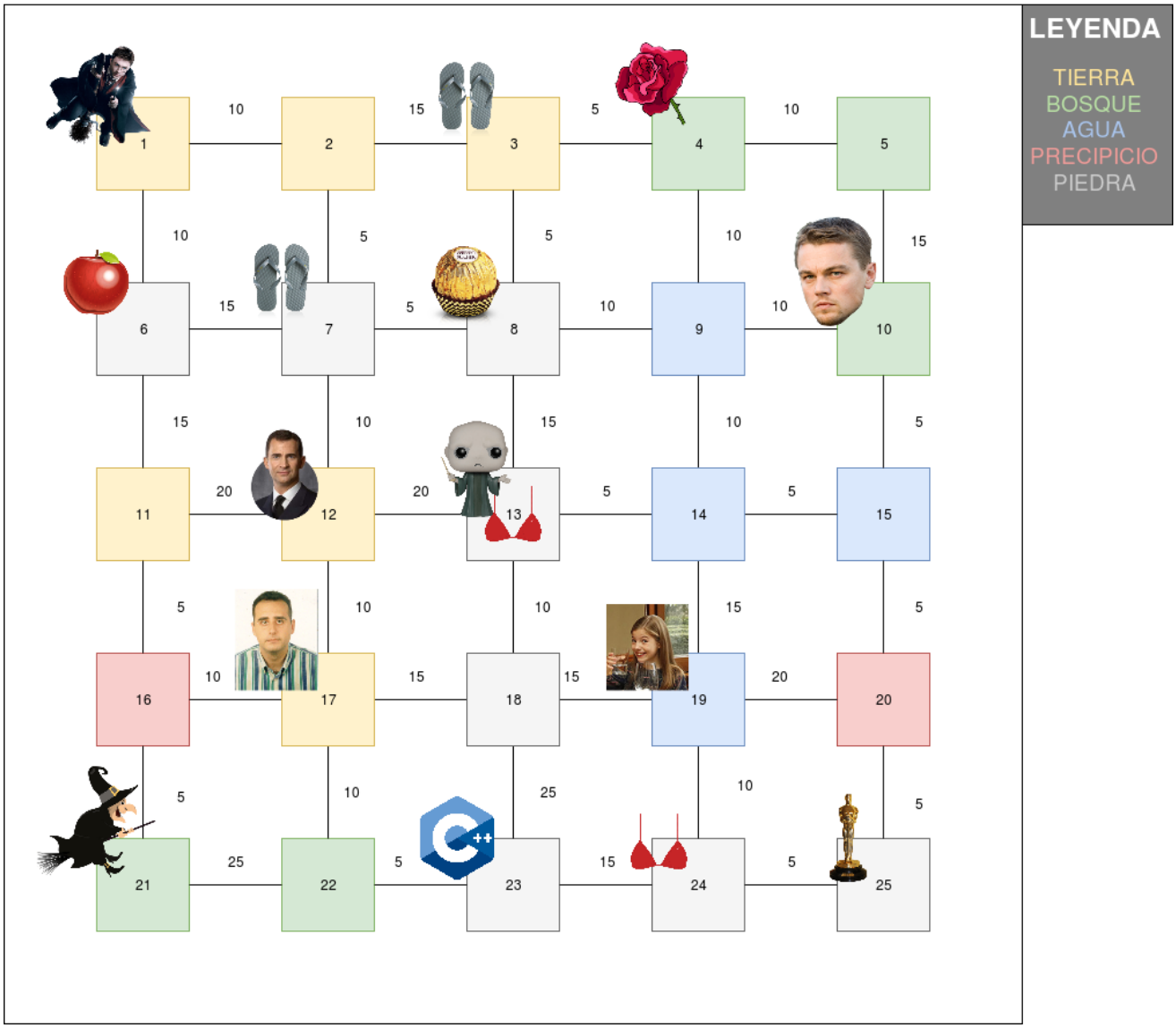
Ahora se nos pide introducir otro jugador en nuestro mundo, dicho jugador tendrá los mismos atributos que el primero por lo que no ha sido necesario cambiar el dominio del problema, tan solo crear otra instancia e inicializarla.

```
    ; Agente 1
    (at HarryPotter z1)
    (orientado HarryPotter sur)
    (= (puntos HarryPotter) 0)
    (= (capacidad mochila1) 7)
    (manoVacia HarryPotter)
    (mochilaVacia mochila1)

    ; Agente 2

    (at Voldemort z13)
    (orientado Voldemort norte)
    (= (puntos Voldemort) 0)
    (= (capacidad mochila2) 7)
    (manoVacia Voldemort)
    (mochilaVacia mochila2)
```

Cada jugador empieza en zonas distintas y tiene una mochila distinta con la misma capacidad.

El objetivo es hacer que los jugadores llegen a obtener un mínimo de puntos, y para este plan he supuesto que cada uno debe llegar a 20 puntos. Restringiendo la meta a esa puntuación hay pocas permutaciones que satisfagan la condición por lo que podemos apreciar como, al contrario que en los otros ejercicios, se demora un poco más para hallar un plan.

El mapa que he usado ha sido el siguiente, y el plan obtenido se encuentra en la carpeta de planes.



(He añadido unas zapatillas y un bikini para que ambos jugadores puedan recorrer los mismos caminos)

Conclusiones

La planificación basada en hechos y acciones es una herramienta muy potente que nos permite observar el comportamiento que tiene un agente dado un mundo ficticio. Que sea tan sencillo crear unas reglas básicas que permitan moverte por un mundo "desconocido" nos permite mandar robots a Marte que exploren la superficie y que recojan objetos, aunque no haya marcianos que te den puntos si les llevas una piedra. La pega de esta herramienta, por poner alguna, sería el caso de establecer un objetivo. En nuestro caso el objetivo ha sido cumplir una serie de misiones, en el caso de un robot en un planeta extraño, el objetivo sería recorrerlo recogiendo información que enviaría.