Técnicas de los Sistemas Inteligentes

Universidad de Granada

Práctica 2

Planificación Clásica

Ignacio Vellido Expósito

Grupo Lunes

Ejercicio 1

Apartado a

Se definen los tipos siguiendo las características del problema. **Objeto** y **personaje** cuentan con subtipos para cada posible instancia de estos. También se crea un tipo **orientacion** para no tener que depender de funciones (además este tipo nos sirve para los predicados del problema).

```
(:types
  jugador - jugador

  princesa principe bruja profesor leonardo - personaje
    princesa - princesa
    principe - principe
    bruja - bruja
    profesor - profesor
    leonardo - leonardo

  oscar manzana rosa algoritmo oro - objeto
    oscar - oscar
    manzana - manzana
    rosa - rosa
    algoritmo - algoritmo
    oro - oro

  zona - zona

  n s e o - orientacion
)
```

Apartado b

Por lo pronto con almacenar los caminos, la orientación, la posición de los personajes y la situación de los objetos es suficiente para representar el mundo.

```
(:predicates
  ; ?z1 conectada con ?z2 y situada a orientación ?o de esta
  ; Acordarse de que el grafo es dirigido, se deben representar las dos
  ; direcciones
  (camino ?z1 - zona ?z2 - zona ?o - orientacion)

  ; Indica la orientación del jugador
  (orientacion ?j - jugador ?o - orientacion)

  ; Indican la posición de un personaje, jugador y objeto
  (posicion_personaje ?p - personaje ?z - zona)
  (posicion_jugador ?j - jugador ?z - zona)
  (posicion_objeto ?o - objeto ?z - zona)

  ; Indica que el jugador tiene un objeto
  (cogido ?o - objeto ?j - jugador)

  ; Indica qué objetos se han entregado a qué personaje
  (entregado ?o - objeto ?p - personaje)
)
```

Apartado c

Las acciones son las siguientes:

- Las acciones de girar tienen en cuenta la orientación del jugador y la cambian según al valor que tenían.
- Las de movimiento comprueban que existe un camino en la dirección a la que está orientado.
- Para **coger**, se comprueba que el objeto está en la misma zona que el jugador y no tiene cogido ninguno actualmente.
- Para **dejar**, basta con saber que tiene el objeto cogido, y puede por tanto dejarlo en la zona.
- Y para **entregar** el personaje y el jugador deben estar en la misma zona y este último tiene que tener cogido el objeto.

Se ha probado con los problemas:

Ej1problema1.pddl	Prueba de movimiento entre zonas, define un mapa e indica al agente a desplazarse a una en concreto, cambiando de orientación en el proceso.
Ej1problema2.pddl	Prueba con objetos, obliga a entregar uno y trasladar otro a otra zona.
Ej1problema3.pddl	Corresponde al apartado *d* del ejercicio. Entrega un objeto a cada personaje.

Ejercicio 2

Apartado a

Para incluir esta nueva funcionalidad se han definido dos funciones:

- **coste-total**: Que almacena el coste del plan actual.
- **coste-camino**: Que almacena el coste de transición de una zona a otra.

Para los cálculos, en las acciones de movimiento se hace que aumente el **coste-total** en base al **coste-camino** entre las dos zonas implicadas.

```
(:functions
  (coste-total)
  (coste-camino ?z1 ?z2 - zona)
)
```

```
(:action ir_norte
  :parameters (?j - jugador ?z1 - zona ?z2 - zona)
  :precondition (and  (posicion_jugador ?j ?z1)
                      (orientacion ?j n)
                      (camino ?z2 ?z1 n)
                )
  :effect (and  (not (posicion_jugador ?j ?z1))
                (posicion_jugador ?j ?z2)
                (increase (coste-total) (coste-camino ?z1 ?z2))
          )
)
```

Aunque no se indica en el enunciado, se ha añadido coste 1 a los movimientos de giro para evitar que el planificador haga más movimientos de los necesarios.

```
(:action girar-derecha
  :parameters (?j - jugador ?o - orientacion)
  :precondition (and (posicion_jugador ?j ?z)
                     (orientacion ?j ?o)
                )
  :effect (and  (not (orientacion ?j ?o))
                (when (= ?o n)
                      (orientacion ?j e)
                )
                (when (= ?o e)
                      (orientacion ?j s)
                )
                (when (= ?o s)
                      (orientacion ?j o)
                )
                (when (= ?o o)
                      (orientacion ?j n)
                )
                ; Para que no dé más giros que los necesarios
                (increase (coste-total) 1)
          )
)
```

Se ha probado con los problemas:

Ej2problema1.pddl	Prueba de movimiento entre zonas, define un mapa e indica al agente a desplazarse a una en concreto, buscando el camino más corto posible.
Ej2problema2.pddl	Corresponde al apartado *b* del ejercicio. Entrega un objeto a cada personaje. Utiliza la ruta más corta posible.

Ejercicio 3

Apartado a

Se añade los subtipos **zapatilla** y **bikini**, y un nuevo tipo **terreno** para poder almacenar las distintas zonas posibles. Adicionalmente, se incluye un predicado para asignar a una zona un tipo de terreno concreto.

```
; Indica qué objeto tiene guardado en la mochila
(en_mochila ?o - objeto ?j - jugador)

; Indica de qué tipo es la zona
(tipo_zona ?z - zona ?t - terreno)
```

Como queremos ser capaces de manejar los objetos en la mochila, se añade otro predicado adicional.

Para mantener las restricciones en los desplazamientos, se dividen las acciones de movimiento en tres:

- Cuando la zona contigua se trata de **piedra** o **arena**, la acción se define de manera similar a la del ejercicio anterior.
- Cuando se quiere desplazar hacia un **bosque**, debemos tener un objeto **zapatilla** cogido o en la mochila.
- Para desplazarse hacia **agua**, se actúa de manera similar a **bosque**, pero con la necesidad de un **bikini**.

Al no estar definida una acción con una zona de precipicio, el movimiento hacia ellas será imposible.

```
(:action ir_oeste
  :parameters (?j - jugador ?z1 - zona ?z2 - zona)
  :precondition (and  (posicion_jugador ?j ?z1)
                      (orientacion ?j o)
                      (camino ?z2 ?z1 o)
                      (or (tipo_zona ?z2 piedra)
                          (tipo_zona ?z2 arena)
                      )
                )
  :effect (and  (not (posicion_jugador ?j ?z1))
                (posicion_jugador ?j ?z2)
                (increase (coste-total) (coste-camino ?z1 ?z2))
          )
)
```

```
(:action ir_norte_bosque
  :parameters (?j - jugador ?z1 - zona ?z2 - zona ?o - zapatilla)
  :precondition (and  (posicion_jugador ?j ?z1)
                      (orientacion ?j n)
                      (camino ?z2 ?z1 n)

                      (tipo_zona ?z2 bosque)
                      (or (cogido ?o ?j)
                          (en_mochila ?o ?j)
                      )
                )
  :effect (and  (not (posicion_jugador ?j ?z1))
                (posicion_jugador ?j ?z2)
                (increase (coste-total) (coste-camino ?z1 ?z2))
          )
)
```

Además, existen dos acciones para la mochila:

- Una para almacenar el objeto que tenemos cogido, teniendo en cuenta que la mochila está vacía.
- Otra para sacar el objeto de la mochila, comprobando que no tenemos ningún objeto cogido.

```
(:action sacar_de_mochila
  :parameters (?j - jugador ?o1 - objeto ?o2 - objeto)
  :precondition (and  (en_mochila ?o1 ?j)
                      (forall (?o2 - objeto)
                            (not (cogido ?o2 ?j))
                      )
                )
  :effect (and (cogido ?o1 ?j)
               (not (en_mochila ?o1 ?j))
          )
)
```

```
(:action guardar_en_mochila
  :parameters (?j - jugador ?o1 - objeto ?o2 - objeto)
  :precondition (and  (cogido ?o1 ?j)
                      (forall (?o2 - objeto)
                            (not (en_mochila ?o2 ?j))
                      )
                )
  :effect (and (en_mochila ?o1 ?j)
               (not (cogido ?o1 ?j))
          )
)
```

Se ha probado con los problemas:

Ej3problema1.pddl	Igual que **Ej2problema1.pddl**, pero incluyendo el dominio nuevo y haciendo el camino completo **piedra**.
Ej3problema2.pddl	Probando que uno de los caminos no es transitable por tener un **precipicio** en él.
Ej3problema3.pddl	Debe recoger una **zapatilla** y un **bikini** para llegar a la zona objetivo, pues el único camino posible incluye **bosque** y **agua**.
Ej3problema4.pddl	A pesar de ser necesaria en el apartado anterior, se obliga a que el jugador coja y saque objetos de la mochila.
Ej3problema5.pddl	Corresponde al apartado c del ejercicio. Entrega un objeto a cada personaje. Utiliza la ruta más corta posible.

Ejercicio 4

Apartado a

Para incluir el manejo de puntos se añaden dos funciones.

- **puntos**: Que almacena los puntos totales del jugador.
- **conseguir-puntos**: Que almacena los puntos obtenidos por entregar un objeto a un personaje.

```
(puntos)
(conseguir-puntos ?o - objeto ?p - personaje)
```

La tabla de puntos por tipo de objeto se debe definir en el problema, pues, aunque resulte engorroso, esta no es una cuestión del dominio. El generador se ha adecuado de manera acorde a esta restricción.

Se ha probado con los problemas:

Ej4problema1.pddl	Prueba simple con puntos en la que se pide que el jugador alcance un mínimo bajo (pero no lo suficiente para entregar cualquier objeto a cualquier personaje).
Ej4problema2.pddl	Corresponde al apartado *b* del ejercicio. Se debe conseguir un mínimo de 25 puntos.

Ejercicio 5

Apartado a

Para este ejercicio se incluyen dos funciones nuevas:

- **bolsillo-maximo**: Indica cuántos objetos acepta el personaje como máximo.
- **bolsillo-actual**: Indicando cuántos objetos han sido entregados al personaje.

```
(bolsillo-maximo ?p - personaje)
(bolsillo-actual ?p - personaje)
```

Y en la acción de entregar se añade la comprobación de que el bolsillo del personaje no esté lleno.

```
(:action entregar
  :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
  :precondition (and (posicion_jugador ?j ?z)
                     (posicion_personaje ?p ?z)
                     (cogido ?o ?j)
                     (< (bolsillo-actual ?p) (bolsillo-maximo ?p))
                )
  :effect (and  (not (cogido ?o ?j))
                (entregado ?o ?p)
                (increase (puntos) (conseguir-puntos ?o ?p))
                (increase (bolsillo-actual ?p) 1)
          )
)
```

Se ha probado con los problemas:

Ej5problema1.pddl	Prueba en la que la única manera de obtener la puntuación es entregando objetos a un personaje con un bolsillo igual a cero, por tanto PDDL no encuentra solución.
Ej5problema2.pddl	Similar al anterior pero en esta ocasión si puede entregarle objetos al personaje y por tanto encontrar solución.
Ej5problema3.pddl	Corresponde al apartado *b* del ejercicio. Algunos personajes aceptan objetos y otros no.

Ejercicio 6

Apartado b

La única modificación necesaria para este apartado es la inclusión de una nueva función **puntos-jugador** para almacenar los puntos que lleva cada jugador durante la ejecución.

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

El resto de predicados y acciones fueron generalizados desde el principio para la existencia de múltiples jugadores y por tanto no es necesario hacerles cambios.

Se ha probado con los problemas:

Ej6problema1.pddl	Corresponde al apartado *b* del ejercicio. Cada jugador tiene su puntuación mínima a obtener. Puesto que el problema estaba generalizado desde el ejercicio 1 no veo necesaria la prueba con más problemas.

Ejercicio 7

Apartado a

En este caso es necesario crear subtipos de jugador para cada tipo de robot y restringir las acciones según las posibilidades de cada uno.

```
(:types
  jugador - jugador
    repartidor - jugador
    recogedor - jugador
```

```
(:action coger
  :parameters (?j - recogedor ?z - zona ?o1 - objeto ?o2 - objeto)
```

```
(:action dejar
  :parameters (?j - recogedor ?z - zona ?o - objeto)
```

```
(:action entregar
  :parameters (?j - repartidor ?z - zona ?o - objeto ?p - personaje)
```

Además, se añade una acción para que dos jugadores intercambien objetos entre sí:

```
; j1 se lo da a j2
(:action traspasar_objeto
  :parameters (?j1 - jugador ?j2 - jugador ?o1 - objeto ?o2 - objeto)
  :precondition (and  (cogido ?o1 ?j1)
                      (forall (?o2 - objeto)
                              (not (cogido ?o2 ?j2))
                      )
                )
  :effect (and (cogido ?o1 ?j2)
               (not (cogido ?o1 ?j1))
          )
)
```

Nota: A excepción de **Ej7problema1.pddl**, todos los ejercicios probados en la práctica funcionan activando la optimización de FF. En el caso de este problema, es tan grande que (a mi parecer) el sistema no es capaz de manejar todas las posibilidades y se origina un fallo de segmento.

Se ha probado con los problemas:

Ej7problema1.pddl Corresponde al apartado *b* del ejercicio. Dos personajes de cada tipo.