Técnicas de los Sistemas Inteligentes (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 2

UNIVERSIDAD
DE GRANADA

Antonio Jesús Heredia Castillo

23 de mayo de 2019

Índice

Índice de figuras

1. El mapa

El mapa que he usado para todos es el siguiente. Creo que con este mapa se puede cubrir todas los posibles casos que se den.

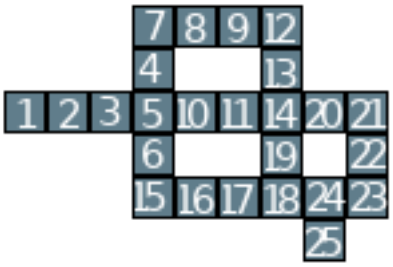

Figura 1:

Los personajes en este mapa se encuentran en:

Personaje	Zona
bruja1	1
profesor1	25
principe1	14
princesa1	21
leonardo	5
jugador	13

Los objetos se encuentran en las siguientes zonas:

Objeto	Zona
oro1	22
oscar1	19
algoritmo1	15
manzana1	24
rosa1	9

2. Ejercicio 1

2.1. Tipos

Tenemos definida la siguiente lista de tipos:

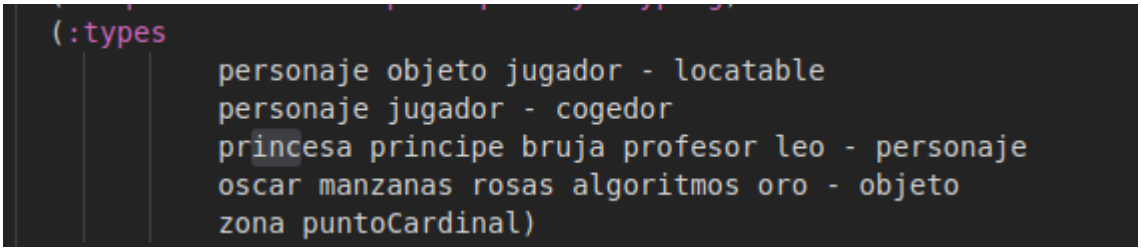

Figura 2:

He creado un tipo llamado "locatable", que incluirá todos los tipos que pueden estar ubicados en una posición, como son los personajes, los objetos y el jugador. Los objetos del tipo "cogedor" serán los que puedan tener objetos cogidos, aunque esto cambiara en los siguientes ejercicios en un principio lo diseñe así. Luego los tipos "personaje" y "objeto" que son los que pedían en el ejercicio. Cambien tengo "zona" que se usara para identificar las diferentes zonas del mapa y "puntoCardinal" para saber en que posición se encuentra una zona o la orientación del jugador. .

2.2. Predicados

Los predicados que tengo son los siguientes:

- zonaVecina ?zona - zona ?vecino - zona ?puntoCar - puntoCardinal : sirve para saber donde se encuentra una zona respecto a la otra. Por ejemplo zonavecina z1 z2 N, quiere decir que z2 se encuentra al norte de z1 y están conectados.
- orientacionJug ?j - jugador ?orienta - puntoCardinal: indica hacia que punto cardinal se encuentra mirando un jugador.
- enZona ?obj - locatable ?zone - zona: indica que objeto/personaje/jugador se encuentra posicionado en una zona.
- tieneObjeto ?obj - objeto ?j - cogedor: indica que objeto tiene cogido un "cogedor"
- tienenUnObjeto ?j - jugador: si un jugador tiene un objeto.

2.3. Acciones

2.3.1. Giro a derecha

Para el giro a la derecha solo necesito dos cosas, una que el jugador se encuentre en una zona y saber la orientación actual del jugador. Sabiendo

esto simplemente, si esta por ejemplo mirando al Norte, el jugador perderá su orientación anterior y su nueva orientación sera Este. Como los giros a derecha e izquierda son diferentes vi que seria mejor tener dos acciones diferentes.

```
(:action GIROr-JUGADOR
        :parameters (?j - jugador ?z - zona ?o - puntoCardinal)
        :precondition (and (enZona ?j  ?z)(orientacionJug ?j ?o))
        :effect(and
        (not(orientacionJug ?j ?o))
        (when (= ?o N)
                (orientacionJug ?j E)
        )
        (when (= ?o S)
                (orientacionJug ?j O)
        )
        (when (= ?o E)
                (orientacionJug ?j S)
        )
        (when (= ?o O)
                (orientacionJug ?j N)
        )
        )
)
```

2.3.2. Giro a izquierda

Exactamente igual que el giro a la derecha. Solo que ahora el calculo de la nueva orientación cambia. Si estas mirando al norte y giras a la izquierda, la nueva posición del jugador sera Oeste.

```
(:action GIROl-JUGADOR
        :parameters (?j - jugador ?z - zona ?o - puntoCardinal)
        :precondition (and (enZona ?j  ?z)(orientacionJug ?j ?o))
        :effect(and
                (not(orientacionJug ?j ?o))
                (when (= ?o N)
                        (orientacionJug ?j O)
                )
                (when (= ?o S)
                        (orientacionJug ?j E)
                )
                (when (= ?o E)
                        (orientacionJug ?j N)
                )
                (when (= ?o O)
                        (orientacionJug ?j S)
                )
        )
)
```

2.3.3. Ir a una zona

Esta acción es muy sencilla. Lo único que realiza es teniendo en cuenta, que la zona a la que queremos ir se encuentra en el mismo punto cardinal

respecto a la zona en la que estamos que el punto cardinal al que esta orientado el jugador. Eso es lo que tenemos en las precondiciones. Como efecto el jugador deja de estar en la zona en que esta y pasa a ir a la zona que quería.

```
(:action IR-ZONA
        :parameters (?j - jugador ?z - zona ?z2 - zona ?o - puntoCardinal)
        :precondition (and (enZona ?j  ?z)(orientacionJug ?j ?o)(zonaVecina
            ?z ?z2 ?o))
        :effect(and
                (not(enZona ?j ?z))
                (enZona ?j ?z2)
        )
)
```

La acción de coger objeto tiene como condición que el objeto y el jugador se encuentre en la misma zona y ademas que el jugador no tenga ya un objeto en la mano. El efecto es que el objeto no se encuentra en la zona y pasa a estar en el jugador.

2.3.4. Coger objeto

```
(:action COGER-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto)
        :precondition (and (enZona ?j  ?z)(enZona ?o ?z)(not(tienenUnObjeto
            ?j)))
        :effect(and
                (not(enZona ?o ?z))
                (tieneObjeto ?o ?j)
                (tienenUnObjeto ?j)
        )
)
```

2.3.5. Dejar objeto

Para esta acción debe encontrarse en una zona el jugador y tener un objeto. El efecto sera que el jugador no tiene ya el objeto y pasa a estar en la zona.

```
(:action DEJAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto)
        :precondition (and (enZona ?j  ?z)(tieneObjeto ?o ?j)(tienenUnObjeto
             ?j))
        :effect(and
                (not(tieneObjeto ?o ?j))
                (not(tienenUnObjeto ?j))
                (enZona ?o ?z)
        )
)
```

2.4. Dar objeto

Aqui el jugador se debe encontrar en la misma zona que un personaje, ademas de que el jugador tenga un objeto. El efecto sera que el jugador ya no tiene el objeto y ahora lo tiene el personaje.

```
(:action ENTREGAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
        :precondition (and (enZona ?p  ?z)(enZona ?j  ?z)(tieneObjeto ?o ?j)
            (tienenUnObjeto ?j))
        :effect(and
                (not(tieneObjeto ?o ?j))
                (tieneObjeto ?o ?p)
                (not(tienenUnObjeto ?j))
        )
)
```

2.5. Explicación de las decisiones

He intentado crear el mínimo posible de acciones. Pero a costa de tener posiblemente mas predicados o tipos. He intentado generalizar al maximo los tipos para que predicados como el de "tieneObjeto" me valga tanto para un personaje como para el jugador.

2.6. Problema

En el fichero de problema, tendremos como objetos todas las zonas del mapa, los personajes que vamos a colocar, los objetos a repartir y los 4 puntos cardinales.
En la parte de inicialización, estarán todas las zonas conectadas con sus zonas vecinas. Ademas de en que zona se encuentra los personajes, los objetos y el propio jugador.
Estos dos puntos nos los genera completos el parser creado en python. Que recibe como entrada lo especificado en el ejercicio y da como salida el fichero de problema para pddl.
Lo unico que hay que añadir al fichero generado por el parser es el "goal" que queremos. Que en este caso va a ser dar un objeto especifico a cada jugador. Lo de un objeto especifico para cada jugador esta hecho asi para que cuando en el ejercicio dos tengamos que optimizar no tarde demasiado.

3. Ejercicio 2

3.1. Tipos

Los tipos son exactamente los mismo que en el ejercicio anterior, no han cambiado en nada.

3.2. Predicados

Los predicados también son iguales. No he necesidad cambiarlos.

3.3. Funciones

Este apartado lo he tenido que añadir nuevo. ya que son necesarios para realizar la minimizacion del coste en distancia recorrida.

```
(:functions
        (distancia ?x ?y - zona)
        (distanciaTotal)
)
```

La función “distancia ?x ?y - zona” nos servirá para saber cuanta distancia hay de una zona a otra con la que tiene conexión.
En ”distanciaTotal” tendremos la distancia total recorrida por el jugador. Esta es la que nos servirá para minimizar.

3.4. Accioens

En esta ocasión hago uso de todas la acciones anteriormente descritas. Solo que añado una modificación a la accion de ir a una zona.

3.4.1. Ir a una zona

El único cambio que realizo en esta acción es añadir el “increase” que incrementara la distancia total recorrida por el jugador en la cantidad definida en “distancia zona1 zona2”. Al haber añadido esa nueva función nos facilita mucho el cambio que había que realizar en este ejercicio.

```
(:action IR-ZONA
        :parameters (?j - jugador ?z - zona ?z2 - zona ?o - puntoCardinal)
        :precondition (and (enZona ?j  ?z) (orientacionJug ?j ?o)(zonaVecina
             ?z ?z2 ?o))
        :effect(and
                (increase (distanciaTotal) (distancia ?z ?z2))
                (not(enZona ?j ?z))
                (enZona ?j ?z2)
        )
)
```

3.5. Problema

En el fichero del problema no cambia mucho, simplemente el parser nos generara las funciones de distancia entre dos zonas y inicializara la distancia total recorrida a 0.
Ademas añadimos como "metric" que se minimize la distanciaTotal.

4. Ejercicio 3

En este ejercicio, existirán distintos tipos de zonas por las que el agente solo podra pasar con determinados objetos.

4.1. Mapa

El mapa sigue siendo el mismo, pero añadimos los nuevos objetos y los tipos de suelo. El color verde es el bosque, el color amarillo la arena, el color azul el agua y el color gris la piedra. Hay que tener en cuenta que hay objetos y personajes en la zona de bosque y agua. Para comprobar si se meten o no.

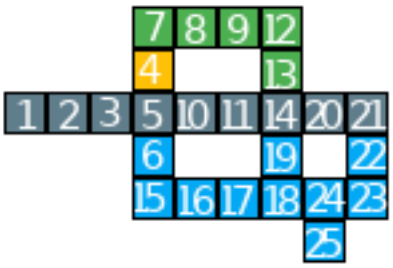

Figura 3:

Nuevos objetos:

Objeto	Zona
bikini1	20
zapatillas1	4

Los tipos de suelo son:

Zona	Tipo
1	Piedra
2	Piedra
3	Piedra
4	Arena
5	Piedra
6	Agua
7	Bosque
8	Pricipicio
9	Bosque
10	Piedra
11	Piedra
12	Bosque
13	Bosque
14	Piedra
15	Agua
16	Agua
17	Agua
18	Agua
19	Agua
20	Piedra
21	Piedra
22	Agua
23	Agua
24	Agua
25	Agua

4.2. Tipos

En este ejercicio añadimos algunos tipos nuevos, que danto asi la lista de tipos:

```
(:types
        personaje objeto jugador - locatable
        personaje jugador - cogedor
        princesa principe bruja profesor leo - personaje
        oscar manzanas rosas algoritmos oro zapatillas bikini zapatillas
            bikini - objeto
        tiposSuelo zona puntoCardinal
)
```

Los nuevo tipos serán “tipoSuelo” que nos servira para definir los distintos tipos de suelo que tenemos en el problema.
Ademas añadimos los dos objetos nuevos, el bikini y las zapatillas

4.2.1. Predicados

Para este ejercicio he tenido que añadir algunos predicados conforme a los de ejercicios anteriores. Para realizar una reorganización en algunas acciones.

```
(:predicates
        (zonaVecina ?z - zona ?vecino - zona ?puntoCar - puntoCardinal)
        (orientacionJug ?j - jugador ?orienta - puntoCardinal)
        (enZona ?obj - locatable ?zone - zona)
        (tieneObjeto ?obj - objeto ?prs - personaje)
        (manoOcupada  ?j - jugador)
        (tieneObjetoJ ?obj - objeto ?j - jugador)
        (mochilaOcupada  ?j - jugador)
        (enMochila ?obj - objeto  ?j - jugador)
        (enMano ?obj - objeto ?j - jugador)
        (tipozona ?tipe - tiposSuelo ?z - zona)

)
```

Voy a describir a continuación cuales he añadido.

- (tipozona ?tipe - tiposSuelo ?z - zona) : nos servira para saber que tipo de suelo tiene una zona expecificamente. Es decir la zona ?z tendra sera del tipo ?tipe
- (enMano ?obj - objeto ?j - jugador): Nos sirve para saber que objeto tiene en la mano el jugador
- (enMochila ?obj - objeto ?j - jugador): Nos sirve para saber que objeto tiene en la mochila el jugador.
- (mochilaOcupada ?j - jugador): Para saber si tiene la mochila ocupada el jugador.
- (manoOcupada ?j - jugador): Para saber si tiene la mano ocupada el jugador.
- (tieneObjetoJ ?obj - objeto ?j - jugador): Este lo utilizo para acceder rapidamente si el jugador tiene un objeto en concreto, dando igual si esta en la mano en la mochila. Muy útil para usar las zapatilals y el bikini.

4.2.2. Funciones

No he añadido ninguna función en este ejercicio. Aunque mantengo las anteriores en caso de que se quisiera minimizar la distancia recorrida por el jugador.

4.2.3. Acciones

En este ejercicio he tenido que añadir varias acciones, aunque he intentado de minimizar la cantidad.

4.3. Ir a una casilla sin tener objeto

Cuando defino ir a una casilla sin tener un objeto, me refiero a sin tener un objeto de "movilidad" ya sea el bikini o las zapatillas. Con esta acción solo se puede mover a través de piedras y por encima de arena.

```
(:action IR-ZONA-SIN-OBJ
        :parameters (?j - jugador ?z - zona ?z2 - zona ?o - puntoCardinal ?t
             - tiposSuelo)
        :precondition (and (enZona ?j  ?z) (orientacionJug ?j ?o)(tipozona ?
            t ?z2 )(zonaVecina ?z ?z2 ?o))

        :effect(and
                (when (or (= ?t Piedra)(= ?t Arena))
                        (and
                                (increase (distanciaTotal) (distancia ?z ?z2
                                    ))
                                (not(enZona ?j ?z))
                                (enZona ?j ?z2)
                        )
                )

        )
)
```

Esto lo realizamos comprobando cual es el tipo de suelo de la zona a la que se quiere avanzar. El resultado sera el cambio de casilla.

4.4. Ir a una casilla con zapatillas

En esta acción nuestro jugador tendrá las zapatillas, ya sea en la mochila o en la mano, no importa. Al tener las zapatillas ademas de por las anteriores podrá ir por suelo de tipo bosque.

```
(:action IR-ZONA-OBJ-MOCHN-zapatillas
        :parameters (?j - jugador ?z - zona ?z2 - zona ?o - puntoCardinal ?t
             - tiposSuelo ?obj - zapatillas)
        :precondition (and (enZona ?j  ?z) (orientacionJug ?j ?o)(tipozona ?
            t ?z2 )(tieneObjetoJ ?obj  ?j )(zonaVecina ?z ?z2 ?o))
        :effect(and

                (when (or (= ?t Piedra)(= ?t Arena)(= ?t Bosque))
                        (and
                        (increase (distanciaTotal) (distancia ?z ?z2))
                        (not(enZona ?j ?z))
                        (enZona ?j ?z2)
                )
        )
        )
```

```
)
```

En caso de que la zona siguiente sea por una de la que se puede mover, el jugador cambiara de casilla.

4.5. Ir a una casilla con bikni

Como en el anterior, si el jugador tiene el bikini en su posesión, podra avanzar ademas de por la piedra y por la arena, por el agua.

```
(:action IR-ZONA-OBJ-MOCHN-bikini
        :parameters (?j - jugador ?z - zona ?z2 - zona ?o - puntoCardinal ?t
             - tiposSuelo ?obj - bikini)
        :precondition (and (enZona ?j  ?z) (orientacionJug ?j ?o)(tipozona ?
            t ?z2 )(tieneObjetoJ ?obj  ?j )(zonaVecina ?z ?z2 ?o))

        :effect(and

                (when (or (= ?t Piedra)(= ?t Arena)(= ?t Agua))
                        (and
                        (increase (distanciaTotal) (distancia ?z ?z2))
                        (not(enZona ?j ?z))
                        (enZona ?j ?z2)
                        )
                )
        )
)
```

Si combinamos esta acción y la anterior, si el jugador tiene el bikini y las zapatillas podrá pasar tanto por el bosque como por el agua.

4.6. Coger/Dejar objetos

Las siguientes acciones de coger y dejar objetos han sufrido unos pequeños cambios. Para que el jugador pueda coger un objeto, este tiene que tener como precondiciñon la mano no ocupada. Esto lo especificamos con un predicado definido anteriormente. El jugador pasara a tener la mano ocupada y a tener en la mano el objeto definido y a tener el objeto de forma global.

```
(:action COGER-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto)
        :precondition (and (enZona ?j  ?z)(enZona ?o ?z) (not(manoOcupada  ?
            j)))
        :effect(and
                (not(enZona ?o ?z))
                (enMano ?o ?j)
                (manoOcupada ?j)
                (tieneObjetoJ ?o  ?j )
        )
)
```

En el caso de dejar es parecido, solo que ahora como precondición el jugador tendra que tener el objeto necesariamente en la mano. No podrá soltarlo directamente desde la mochila.

```
(:action DEJAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto)
        :precondition (and (enZona ?j  ?z) (manoOcupada ?j)(enMano ?o ?j))
        :effect(and
                (not(enMano ?o ?j))
                (enZona ?o ?z)
                (not(manoOcupada ?j))
                (not(tieneObjetoJ ?o  ?j ))
        )
)
```

El jugador dejara de tener la mano ocupada, de tener el objeto en la mano y de tener el objeto de forma global. Ahora pasara ha estar el objeto en el suelo.

4.6.1. Entregar el objeto

A diferencia de en ejercicios anteriores, el jugador tendrá que tener el objeto necesariamente en la mano. No valdrá tenerlo en la mochila.

```
(:action ENTREGAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
        :precondition (and (enZona ?p  ?z)(enZona ?j  ?z)(enMano ?o ?j))
        :effect(and
                (not(enMano ?o ?j))
                (not(tieneObjetoJ ?o  ?j ))
                (tieneObjeto ?o ?p)
                (not(manoOcupada ?j))
        )
)
```

El objeto como efecto pasara a estar en posesión del personaje en cuestión y dejara de estar en la mano del jugador.

4.7. Guardar/Sacar objeto

Ahora al disponer de una mochila, el jugador podra guardar y sacar objetos de ella. Para guardar un objeto, debera tener la mochila vacía ademas de tener un objeto en la mano.

```
(:action GUARDAR-OBJETO
        :parameters (?j - jugador  ?o - objeto)
        :precondition (and (not(mochilaOcupada ?j))(manoOcupada  ?j) (enMano
             ?o ?j))
        :effect(and
                (not (enMano ?o ?j))
                (not (manoOcupada ?j))
                (mochilaOcupada ?j)
                (enMochila ?o ?j)
        )
```

```
)
```

Como efecto el objeto pasara a la mochila y esta a estar ocupada. Ademas de liberar la mano del jugador.
Para sacar un objeto de la mochila la mano del jugador deberá estar vaciá y la mochila contener algún objeto.

```
(:action SACAR-OBJETO
        :parameters (?j - jugador  ?o - objeto)
        :precondition (and (not(manoOcupada ?j))(mochilaOcupada  ?j) (
            enMochila ?o ?j))
        :effect(and
        (not (enMochila ?o ?j))
        (not (mochilaOcupada ?j))
        (manoOcupada ?j)
        (enMano ?o ?j)
        )
)
```

4.8. Problema

En el fichero de problema no cambian ni los objetivos ni la métrica. El parser solo añadirá a los objetos los distintos tipos de suelo y también añadirá los dos nuevos objetos(bikini y zapatilla)
En la zona de inicialización se inicializara de que tipo de zona es cada zona.

5. Ejercicio 4

En este ejercicio se añade diferentes puntuaciones a la entrega de objetos.

5.1. Tipos

No añado ningun tipo nuevo al dominio.

5.2. Predicados

No añado ningun predicado nuevo al dominio.

5.3. Funciones

Aquí si añado algunas funciones nuevas que paso a explicar a continuación.

- (puntosMaximos): Aqui se guardara los puntos necesarios para que el "juego" acabe.

- (puntosTotales): La cantidad de puntos que tiene el jugador por ahora.
- (puntosDa ?x - objeto ?y - personaje) : la determinada cantidad de puntos que da un objeto a un personaje.

5.4. Acciones

En este ejercicio solo cambia una acción y es la de entregar objeto al personaje. Añadimos un "increase" que añadira a la función puntosTotales la cantidad de puntos definida por (puntosDa ?x - objeto ?y - personaje).

```
(:action ENTREGAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
        :precondition (and (enZona ?p  ?z)(enZona ?j  ?z)(enMano ?o ?j))
        :effect(and
                (not(enMano ?o ?j))
                (not(tieneObjetoJ ?o  ?j ))
                (tieneObjeto ?o ?p)
                (not(manoOcupada ?j))
                (increase (puntosTotales) (puntosDa ?o ?p))
        )
)
```

Con esto conseguimos tener la función puntosTotales actualizada cada vez que se entrega un objeto.

5.5. Problema

Ahora en el problema cambiaremos el goal.Ahora el objetivo del jugador sera obtener al menos 50 puntos.

```
(:goal
        (AND
                (>= (puntosTotales) (puntosMaximos))
        )
)
```

Ademas el parser añadira al fichero de problema cuantos puntos dará cada objeto a cada personaje. Por ejemplo un oscar le da a la bruja 4 puntos.

$$(= (puntosDaoscar1bruja1)4)$$

El parser también inicializara el valor de

$$(= (puntosTotales)0)$$

y

$$(= (puntosMaximos)50)$$

6. Ejercicio 5

En este ejercicio se limitara la cantidad de objetos que puede tener un determinado personaje.
En este ejercicio no añado ni predicados ni tipos nuevo, pero si funciones.

6.1. Funcioness

Tengo dos nuevas funciones.
La función ($tamanioBolsillo?x - personaje$) no dirá el tamaño del bolsillo de un personaje.
La función ($cantidadObjetos?x - personaje$) nos dirá la cantidad de objetos que lleva ya el personaje.
Estas dos funciónes nos servira para no dar mas objetos de los que tiene capacidad a un personaje. Esto lo controlaremos con las acciones.

6.2. Acciones

6.2.1. Dar objeto

La función dar objeto es la que tiene que controlar que no demos mas objetos que los que tiene capacidad un determinado personaje. El tamaño del bolsillo mágico de cada personaje se instancia en el archivo de problema y de ahí se coge. La acción comprobara antes de dar el objeto que el personaje tiene capacidad y si tiene, se lo da, ademas incrementa en 1 la cantidad total de objetos que tiene el personaje.

```
(:action ENTREGAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
        :precondition (and (enZona ?p  ?z)(enZona ?j  ?z)(enMano ?o ?j))
        :effect(and
        (when   (and (< (cantidadObjetos ?p) (tamanioBolsillo ?p)))
                (and
                        (not(enMano ?o ?j))
                        (not(tieneObjetoJ ?o  ?j ))
                        (tieneObjeto ?o ?p)
                        (not(manoOcupada ?j))
                        (increase (puntosTotales) (puntosDa ?o ?p))
                        (increase (cantidadObjetos ?p) 1)
                )
        )
        )
)
```

6.3. Problema

En el fichero de problema el parser añadirá la capacidad que tiene cada personaje ademas de inicializar a 0 la cantidad de objetos que tiene cada uno. El objetivo no cambia.

7. Ejercicio 6

En este ejercicio habrá varios jugadores. Y ambos tendrá que llegar a un mínimo de puntos, que este puede cambiar de uno a otro. Ademas se tendrá que llegar al objetivo de mínimo de puntos.

7.1. Funciones

En este ejercicio volvemos a añadir dos funciones muy parecidas a las del ejercicio anterior. La función $(puntosJugador?j - jugador)$ nos indicara cuantos puntos tiene cada jugador.
La función $(puntosMinimoJugador?j - jugador)$ nos indicara cuantos puntos necesita cada jugador.
Con estas dos funciones podremos controlar el objetivo necesario.

7.2. Acciones

7.3. Dar objeto

Volvemos a modificar solo la acción de dar objeto a un personaje. En esta función añadiremos un incremento en los puntos que tiene cada jugador. Los puntos se incrementan en función de la cantidad de puntos que da dar un determinado objeto a un determinado personaje $((puntosDa?o?p))$.

```
(:action ENTREGAR-OBJETO
        :parameters (?j - jugador ?z - zona ?o - objeto ?p - personaje)
        :precondition (and (enZona ?p  ?z)(enZona ?j  ?z)(enMano ?o ?j))
        :effect(and
        (when (and (< (cantidadObjetos ?p) (tamanioBolsillo ?p)))
                (and
                        (not(enMano ?o ?j))
                        (not(tieneObjetoJ ?o  ?j ))
                        (tieneObjeto ?o ?p)
                        (not(manoOcupada ?j))
                        (increase (puntosTotales) (puntosDa ?o ?p))
                        (increase (cantidadObjetos ?p) 1)
                        (increase (puntosJugador ?j) (puntosDa ?o ?p))
                        )
                )
        )
)
```

7.4. Problema

El objetivo ahora ademas de tener una cantidad mínima de puntos totales necesarios también tendrá la cantidad mínima de puntos por jugador.

```
(:goal
        (AND
                (>= (puntosTotales) (puntosMaximos))
                (>= (puntosJugador player1) (puntosMinimoJugador player1))
                (>= (puntosJugador player2) (puntosMinimoJugador player2))
        )
)
```

Ademas el parser añade los puntos minimos que necesita cada jugador y inicializa a 0 los puntos que tiene hasta ahora.

8. Parser

El parser esta escrito en Python. Hay un parser y un fichero sin procesar y otro procesado del problema para cada ejercicio. Aunque he intentado realizarlo lo mejor posible, es muy rigido, es decir no te puedes dejar un espacio o lineas en blanco. No obstante funciona de forma adecuada para todos los ejercicios. La parte de los objetivos no la completa ya que entiendo que es algo que puede variar mucho mas entre ejercicio.