Práctica 2
Técnicas de los Sistemas Inteligentes

Ignacio Aguilera Martos

18 de mayo de 2019

Índice

1. Ejercicio 1

El primero de los apartados nos pide definir los objetos, personajes y demás tipos utilizados en el dominio. Los tipos que yo he representado son zona, orientación, agente, princesa, principe, bruja, profesor, leonardo, oscar, manzana, rosa, algoritmo, oro, personaje, objeto y posicionable.

El diagrama de tipos que he empleado es un árbol de la forma:

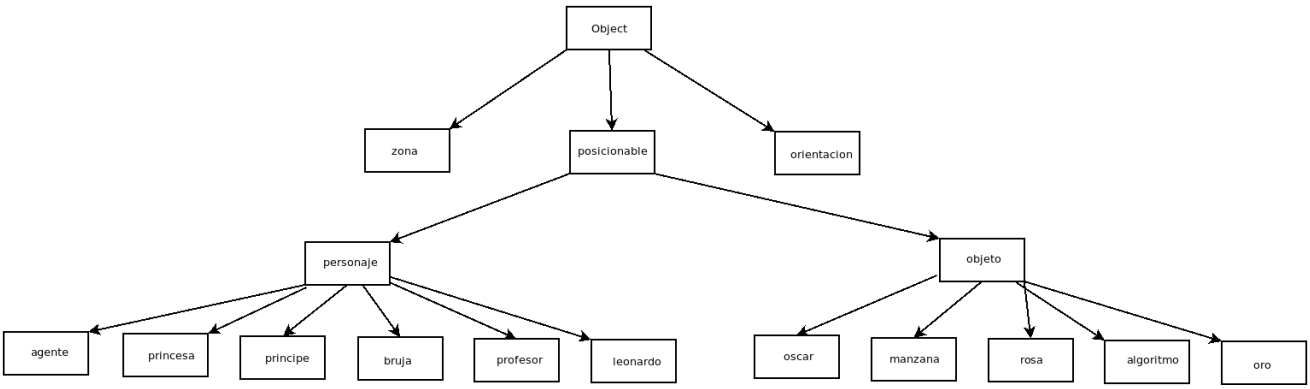


Figura 1: Diagrama de tipos

En cuanto a los predicados que he tomado para resolver el ejercicio son:

- (en ?x - posicionable ?y - zona): este predicado lo que nos indica es que el objeto o personaje ?x está en la zona ?y.
- (orientado ?a - personaje ?ori - orientacion): este predicado nos dice que el personaje (incluido el propio agente) ?a está orientado según la orientación ?ori.
- (conectado ?z1 ?z2 - zona ?ori - orientacion): este predicado nos dice que las zonas ?z1 y ?z2 están conectadas por la orientación ?ori, esto es si z2 está a la derecha de z1 entonces están conectadas por el este, si z2 está debajo de z1 entonces z1 está conectada a z2 por el sur y así para el resto de orientaciones.
- (manovacia ?a - agente): este predicado nos dice si el agente ?a tiene o no la mano vacía para llevar un objeto.
- (enlamano ?a - agente ?obj - objeto): este predicado nos dice que el agente ?a tiene un objeto ?obj en la mano.
- (tieneobjeto ?per - personaje): este predicado nos indica que el personaje ?per tiene un objeto.

Cabe decir que además de los predicados y tipos he definido 4 constantes para las orientaciones que son este, oeste, norte y sur.

Para las acciones las he definido de la siguiente forma:

- girar-izquierda: es la acción que nos permite rotar hacia la izquierda, requiere como único parámetro un agente y el efecto es, comprobando la orientación actual del agente cambiar dicha orientación de forma adecuada al giro a la izquierda.

- girar-derecha: es la misma acción que girar a la izquierda pero cambiando el sentido de la rotación.
- ir: recibe como parámetros un agente, dos zonas y una orientación. Como precondiciones establecemos que el agente esté en la primera de las posiciones, que el agente esté orientado según la orientación dada y que las zonas estén conectadas en esa dirección. El efecto de esta acción es que el agente pasa a estar en la segunda zona y deja de estar en la primera.
- coger: esta acción recibe como parámetros un agente, una zona y un objeto. Se pide como precondición que el agente esté en la zona, el objeto también y que el agente tenga la mano vacía. Como efecto se produce que el agente deja de tener la mano vacía, el objeto deja de estar en la zona y pasa a estar en la mano del agente.
- dejar: recibe como parámetros un agente, una zona y un objeto. Como precondiciones se pide que el agente esté en la posición y tenga el objeto en la mano. El efecto que se produce es que el objeto pasa a estar en la zona y desaparece de la mano del agente y éste pasa a tener la mano vacía.
- entregar: recibe como parámetros un agente, una zona, un objeto y un personaje. Como precondiciones se pide que el agente esté en la zona, el personaje también y que el agente tenga el objeto en la mano. El efecto producido es que el agente deja de tener el objeto en la mano, pasa a tener la mano vacía y el objeto pasa a tenerlo el personaje.

Lo siguiente que se nos pide es que planteemos de forma manual un problema con 25 zonas, 5 personajes mas el agente y 5 objetos. El objetivo será que cada uno de los 5 personajes tenga un objeto.

La disposición del mapa que he dado es la siguiente:

z1 Agente	z2	z3 Princesa	z4	z5 Principe			
z6 Oro	z7	z8 Oscar	z9	z10 Leonardo			
z11 Bruja	z12	z13	z14 Manzana	z15			z25
z16	z17	z18 Profesor	z19	z20	z21 Rosa	z22	z23 Algoritmo
							z24

Partimos además de que el agente está orientado hacia el norte, tiene la mano vacía y el objetivo es que todos los personajes tengan un objeto.

Por último se nos pide un parser que he implementado en Python3 que dada una entrada de la forma:

numero de zonas:7

Dominio:Ejercicio2

Problema:Problema2

V -> z1[bruja1-bruja] z3[] z6[]

H -> z2[player1-agente] z3[] z4[]

H -> z5[oscar1-oscar,manzana1-manzana] z6[] z7[princesa1-princesa]

Cabe decir que tenemos que poner los nombres de los tipos en minúscula para que casen con los que yo he definido.

Además, aunque el enunciado dice que los objetos de cada zona (entre corchetes) deben estar separados por espacios, he tomado la decisión de diseño de que los elementos estén separados por comas, pues de esta forma los separadores son únicos para cada caso lo que hace que su división sea más sencilla.

El resto es simplemente un programa en Python comentado que convierte este fichero en un fichero de problema PDDL completo usando el dominio descrito anteriormente.

2. Ejercicio 2

Para ajustar el dominio al nuevo problema tenemos que definir dos funciones:

- (coste ?z1 ?z2): nos indica cuál es el coste de pasar de la zona ?z1 a la zona ?z2.
- (costeTotal): es la suma del coste del camino que estamos recorriendo, es decir, cada vez que entremos a la acción ir sumaremos a esta función el coste del arco (coste ?z1 ?z2) tomado.

Para terminar de adecuar esto tenemos que modificar la función "ir" para que en los efectos se incremente (costeTotal) de la forma (increase (costeTotal) (coste ?z1 ?z2)) donde "ir" parte de que el agente está en la zona ?z1 y quiere desplazarse a la zona ?z2.

Para la extensión del problema podemos tomar todo lo hecho y únicamente añadir en el bloque "init" la declaración de los pesos de las conexiones e inicializar (costeTotal) a 0.

En el problema anterior de 25 zonas vamos a añadir los siguientes pesos:

z1 Agente	z2	z3 Princesa	z4	z5 Principe			
z6 Oro	z7	z8 Oscar	z9	z10 Leonardo			
z11 Bruja	z12	z13	z14 Manzana	z15			z25
z16	z17	z18 Profesor	z19	z20	z21 Rosa	z22	z23 Algoritmo
							z24

3. Ejercicio 3

En primer lugar, para representar los tipos de terrenos he declarado un nuevo tipo llamado "superficie" del que he declarado las constantes: bosque, agua, precipicio, arena y piedra. Además he declarado los tipos bikini y zapatilla que son subtipos de objeto con lo que se considera que los bikinis y zapatillas también son objetos.

Para poder representar esta nueva información he añadido el predicado (es ?sup ?z) que nos indica que la zona ?z es del tipo de superficie ?sup y los predicados (esbikini ?obj) y (eszapatilla ?obj) que nos indican si los objetos son zapatillas o bikinis.

Para representar el conocimiento de la mochila ahora tenemos dos predicados nuevos que son (mochilavacia ?age) que nos dice que el agente ?age tiene la mochila vacía y (enlamochila ?age ?obj) que nos dice que el agente ?age tiene el objeto ?obj en la mochila.

En cuanto a las acciones que han sido modificadas son dos "ir" y las nuevas acciones de "meter-mochila" y "sacar-mochila".

La acción de ir ahora tiene nuevas precondiciones. Sumadas a las anteriores se pide ahora que la zona a la que se va a ir no sea precipicio, que no sea agua o que en caso de serlo tengamos un bikini en la mochila o en la mano y que o no sea bosque o que si lo es tengamos unas zapatillas en la mochila o en la mano. De esta forma no nos moveremos a los precipicios y sólo pasaremos por el agua con un bikini y por el bosque con unas zapatillas.

La acción "meter-mochila" requiere que tengamos un objeto en la mano y la mochila vacía. Como efecto se cumple que tenemos la mano vacía, el objeto en la mochila y ya no tenemos la

mochila vacía ni el objeto en la mano.

La acción “sacar-mochila” requiere que tengamos un objeto en la mochila y la mano vacía. Como efecto se cumple que tenemos la mochila vacía, el objeto en la mano y ya no tenemos la mano vacía ni el objeto en la mochila.

El problema para ser extendido requiere que definamos las superficies de cada zona y añadamos un bikini y unas zapatillas al menos, con lo que el mapa queda:

z1 Agente Piedra 2	z2 5 Piedra 5	z3 Princesa 4 Agua 6	z4 3 Piedra 5	z5 Principe 8 Bosque 2			
z6 Oro 3 Arena 1	z7 Arena 1	z8 Oscar 6 Arena 1	z9 5 Arena 1	z10 Leonardo 1 Arena 1			
z11 Bruja Agua 4	z12 7 Piedra 3	z13 2 Piedra 2	z14 Manzana 6 Piedra 6	z15 Zapatilla 2 Agua 8			z25 Piedra 8
z16 Bikini 8 Arena	z17 Arena	z18 9 Profesor Arena	z19 2 Precipicio	z20 5 Arena	z21 Rosa 2 Bosque	z22 1 Arena	z23 Algoritmo 4 Piedra 6
							z24 Piedra

En este caso para el parser he modificado un poco la sintaxis para adecuarla de la siguiente forma:

numero de zonas:7

Dominio:Ejercicio3

Problema:Problema3

V -> z1[bruja1-bruja](bosque)=10=z3[bikini1-bikini](arena)=5=z6[zapatilla1-zapatilla](piedra)

H -> z2[player1-agente](piedra)=10=z3[](arena)=5=z4[](piedra)

H -> z5[oscar1-oscar,manzana1-manzana](bosque)=10=z6[](piedra)=5=z7[princesa1-princesa](piedra)

Con lo que para el tipo de terreno lo situamos después de los personajes y objetos entre paréntesis. El cambio de corchetes a paréntesis es por simplicidad, puesto que son separadores distintos a los usados hasta el momento y por tanto podemos reutilizar de mejor manera el código previo.

4. Ejercicio 4

Para poder mantener un registro de los puntos del agente debemos declarar dos nuevas funciones. La primera de ellas es (puntosTotales) que mantendrá el conteo del número de puntos que lleva el agente por el momento y la función (puntos ?obj ?personaje) que nos indica cuántos puntos cuenta entregar el objeto ?obj al personaje ?personaje.

Una vez definidas estas funciones tenemos que modificar la función entregar para que al entregar un objeto a un personaje mantengamos el conteo de puntos. Para ello es tan sencillo como sumar a la función (puntosTotales) el valor de (puntos ?obj ?per) donde ?obj es el objeto que se va a entregar y ?per el personaje al que se le va a entregar. Esta implementación nos obliga a definir en el problema los puntos que nos sumará entregar cada uno de los objetos (cada instancia) a cada uno de los personajes (cada instancia de personaje).

El problema anterior es fácilmente extensible. Debemos inicializar en primer lugar la función (puntosTotales) a cero y además para cada objeto y personaje que tenemos debemos definir el número de puntos que nos va a repercutir entregar cada uno de estos objetos a cada personaje presente en el mapa. Además para que se tenga en cuenta el número de puntos debemos añadir alguna restricción en el goal o indicar al planificador que se maximice la métrica de puntos. En mi caso he optado por introducir en el goal una condición del tipo (¿= (puntosTotales) x) para que la solución obtenga al menos x puntos.

El problema de 25 zonas que he planteado es de la forma:

z1 Agente Piedra 2	z2 5 Piedra 5	z3 Princesa 4 Arena 6	z4 3 Piedra 5	z5 Principe 8 Piedra 2			
z6 Oro 3 Arena 1	z7 Arena 1	z8 Oscar 6 Arena 1	z9 5 Arena 1	z10 Leonardo 1 Arena 1			
z11 Bruja Arena 4	z12 7 Piedra 3	z13 2 Piedra 2	z14 6 Manzana Piedra 6	z15 Zapatilla 2 Agua 8			z25 Piedra 8
z16 Bikini 8 Arena	z17 Arena	z18 9 Profesor Arena	z19 2 Precipicio	z20 5 Arena	z21 Rosa 2 Piedra	z22 1 Arena	z23 Algoritmo 4 Piedra 6
							z24 Piedra

He quitado algunas restricciones en cuanto a superficies con respecto al apartado anterior para facilitar la resolución y no agotar demasiado tiempo.

En cuanto al parser en este caso se añade una línea de cabecera que indica el mínimo de puntos

que debe tener el agente al final de la resolución y además en la escritura del problema debemos tener en cuenta que mi implementación requiere que para cada objeto y personaje se establezca el número de puntos que otorga entregar dicho objeto a cada personaje.

De esta forma he introducido los datos de puntuaciones como un diccionario de diccionarios que contiene en el más general un diccionario con los nombres de los tipos de personajes y como valor en dicho diccionario un nuevo diccionario que contiene como clave el nombre de los tipos de los objetos excluyendo zapatillas y bikini que no tienen ningún tipo de puntuación. Gracias a este diccionario podemos escribir para cada personaje y objeto declarados la relación de puntaje que otorgan.

5. Ejercicio 5

Ahora tenemos que modificar el hecho de que los personajes sólo puedan recibir un objeto pues ahora puede ser más de uno. Para controlar esto he decidido declarar una función que se llama (numeroObjetos ?personaje) que nos indica la capacidad de dicho personaje.

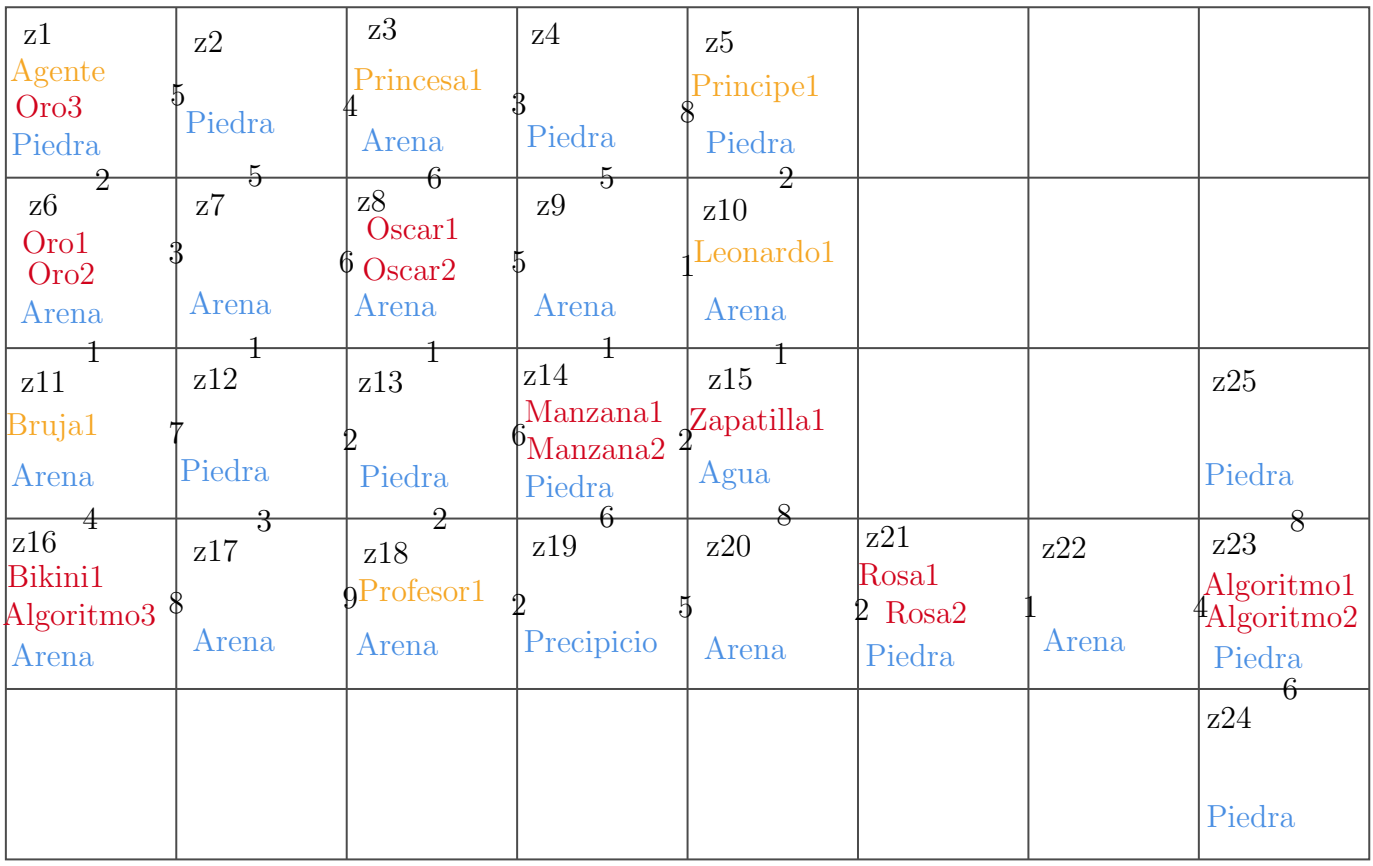
Junto con esta función he modificado la función entregar para que se añada como precondición que (numeroObjetos ?personaje) sea mayor que 0 y además que al entregar un objeto se decremente en una unidad la función para llevar el conteo de los espacios libres que le quedan a dicho personaje.

De esta forma al declarar ahora un nuevo problema debemos especificar como funciones para cada personaje el número de objetos que pueden almacenar cada uno.

Sólo queda por tanto probarlo con ejemplos para comprobar que funciona. Téngase en cuenta que los objetivos de los apartados anteriores han sido simplificados para que se pueda obtener un resultado de forma rápida y no tener así que esperar varios minutos o incluso más.

Los dos problemas que he propuesto mantienen la estructura de mapa del ejercicio anterior y se añade ahora las condiciones sobre la capacidad de los personajes. Además para simplificar la solución he añadido un mayor número de objetos con la intención de que el objetivo de puntaje sea fácilmente completable.

El mapa queda por tanto así:



Para los dos problemas que he propuesto he dado como capacidades las siguientes:

Problema / Capacidades	Princesa1	Principe1	Bruja1	Leonardo1	Profesor1
1	1	2	1	1	2
2	3	2	1	1	3

En cuanto al parser el único cambio es añadir el hecho de que se añade una nueva línea de la forma:

bolsillo:[bruja1:4 princesa1:5]

Que se traduce en mi caso en dos nuevas líneas en el problema:

(= (numeroObjetos bruja1) 4)

(= (numeroObjetos princesa1) 5)

6. Ejercicio 6

La primera modificación que he tenido que realizar ha sido añadir una función nueva llamada (puntosObtenidos ?agente) que indicará el número de puntos que lleva el agente ?agente. De esta forma podemos controlar el puntaje individual manteniendo la función (puntosTotales) que llevará la cuenta del total.

Como segunda modificación en el dominio he alterado la función entregar que ahora añade como efecto incrementar la función (puntosObtenidos ?agente) con el valor (puntos ?obj ?per)

que no es más que los puntos que le corresponden por entregar el objeto ?obj al personaje ?per.

En cuanto al problema ahora podemos inicializar un nuevo agente que en mi caso requiere definir su orientación, indicar su posición e indicar que tanto su mano como la mochila están vacías.

Además ahora el goal incluye que las funciones (puntosObtenidos ?agente) sean mayor que una cierta cota para que los dos agentes obtengan cada uno por separado un mínimo de puntos.

El mapa definido para el problema es el siguiente:

z1 Agente1 Oro3 Piedra 2	z2 5 Piedra 5	z3 Princesa1 4 Arena 6	z4 3 Piedra 5	z5 Principe1 8 Piedra 2			
z6 Oro1 Oro2 Arena 1	z7 3 Arena 1	z8 Oscar1 6 Oscar2 Arena 1	z9 5 Arena 1	z10 Agente2 1Leonardo1 Arena 1			
z11 Bruja1 Arena 4	z12 7 Piedra 3	z13 2 Piedra 2	z14 Manzana1 6 Manzana2 Piedra 6	z15 Zapatilla1 2 Agua 8			z25 Piedra 8
z16 Bikini1 Algoritmo3 Arena	z17 8 Arena	z18 9Profesor1 Arena	z19 2 Precipicio	z20 5 Arena	z21 Rosa1 2 Rosa2 Piedra	z22 1 Arena	z23 Algoritmo1 4Algoritmo2 Piedra 6
							z24 Piedra

En el primero de los problemas he definido como restricciones que el puntaje total sea mayor que 50 y que los puntajes individuales sean mayores que 20, de esta forma al menos uno de los dos agentes deberá exceder esta cota para cumplir la condición del puntaje total.

En el segundo de los problemas he definido las restricciones de forma opuesta al primer problema. En este caso he definido el problema para que el agente 1 tenga al menos 25 puntos y el agente 2 tenga como mínimo 30 puntos siendo el mínimo de puntos totales 30, es decir que sólo con cumplir la cota del segundo agente ya se cumple la cota del puntaje total.

En cuanto al parser la modificación es exactamente igual que la implementada en el apartado anterior pues el formato es el mismo y sólo cambia la función que utiliza los datos de entrada en PDDL.

7. Ejercicio 7

Para este ejercicio no he definido ningún tipo extra, en su defecto he cogido y he definido dos predicados (espicker ?agente) y (esdealer ?agente) que nos indican si un agente es picker o dealer.

En cuanto a modificación en las funciones he modificado la función entrega dividiéndola en dos: entrega-agente y entrega-personaje.

La primera de ellas (entrega-agente) es la acción de que un agente entregue un objeto a otro, en concreto un picker a un dealer. De esta forma como precondiciones debemos exigir que el agente que tiene el objeto lo tenga en la mano y sea un picker y el agente al que se le va a dar el objeto sea un dealer y además tenga la mano vacía. Además ambos deben de estar en la misma zona.

De esta forma los efectos que produce la acción son que el agente que entrega el objeto queda con la mano vacía y ya no posee el objeto y el segundo agente (el dealer) ya no tiene la mano vacía y posee el objeto.

La segunda es la acción que modela que un dealer entregue a un personaje un objeto. De esta forma se requiere que el agente sea un dealer y el resto del comportamiento es idéntico al definido previamente.

En cuanto a la modificación del problema en primer lugar tenemos que especificar qué es cada agente (picker o dealer) y cambiar el goal, pues ahora la restricción de puntosObtenidos para un agente sólo se puede pedir si ese agente es de tipo dealer, pues el picker nunca podrá entregar un objeto a un personaje.

Además la función de coger ahora requiere que el agente que va coger el objeto sea un picker, de esta forma un se completa el ciclo de funcionamiento donde para dejarle un objeto a un personaje primero el picker lo coge, después se lo entrega al dealer y por último es el dealer el que entrega el objeto al personaje final.

Para probar el dominio he diseñado dos problemas de forma manual con los siguiente mapas:

En el ejemplo 1:

z1 Agente1 Oro3 Piedra 2	z2 5 Piedra 5	z3 Princesa1 4 Arena 6	z4 3 Piedra 5	z5 Principe1 8 Piedra 2			
z6 Oro1 Oro2 Arena 1	z7 3 Arena 1	z8 Oscar1 6 Oscar2 Arena 1	z9 5 Arena 1	z10 Agente2 1 Leonardo1 Arena 1			
z11 Bruja1 Arena 4	z12 7 Piedra 3	z13 2 Piedra 2	z14 Manzana1 6 Manzana2 Piedra 6	z15 Zapatilla1 2 Agua 8			z25 Piedra 8
z16 Bikini1 Algoritmo3 Arena	z17 8 Arena	z18 9 Profesor1 Arena	z19 2 Precipicio	z20 5 Arena	z21 Rosa1 2 Rosa2 Piedra	z22 1 Arena	z23 Algoritmo1 4 Algoritmo2 Piedra 6
							z24 Piedra

En el ejemplo 2:

z1 Agente1 Piedra 2	z2 5 Piedra 5	z3 Princesa1 4 Arena 6	z4 3 Piedra 5	z5 Principe1 8 Piedra 2			
z6 Oro1 Arena 1	z7 3 Arena 1	z8 Oscar1 6 Arena 1	z9 5 Arena 1	z10 Agente2 1 Leonardo1 Arena 1			
z11 Bruja1 Arena 4	z12 7 Piedra 3	z13 2 Piedra 2	z14 Manzana1 6 Piedra 6	z15 Zapatilla1 2 Agua 8			z25 Piedra 8
z16 Bikini1 Arena	z17 8 Arena	z18 9 Profesor1 Arena	z19 2 Precipicio	z20 5 Arena	z21 Rosa1 2 Piedra	z22 1 Arena	z23 Algoritmo1 4 Piedra 6
							z24 Piedra

Donde en el ejemplo 1 se coloca al agente 1 como el picker y al 2 como el dealer y se piden como goals que los puntos totales sean mayor o igual a 30, el coste total menor o igual que 1500

y los puntos obtenidos por el segundo agente mayores o igual que 20.

En el problema 2 se han colocado muchos menos objetos y además el coste total se pide que sea menor que 1500, los puntos totales mayor o igual que 20 y los puntos obtenidos por el agente 2 al menos 15 donde el agente 1 es picker y el 2 el dealer.