UNIVERSIDAD
DE GRANADA

Técnicas de los Sistemas Inteligentes
grado en ingeniería informática

PRÁCTICA 2

Planificación Clásica

Autor
Vladislav Nikolov Vasilev

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

Curso 2018-2019

Índice

1. Ejercicio 1

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 1.

1.1. Ejercicio 1.a

Para representar los objetos del mundo se han utilizado tipos, y para dar un mayor nivel de abstracción, se han utilizado supertipos. Por ejemplo, los distintos tipos de objetos (Manzana, Algoritmo, etc.) se han englobado bajo el supertipo *items* para poder facilitar referirse a cualquiera de ellos. Lo mismo pasa con los personajes, los cuáles han sido englobados bajo el supertipo *npc* para referirse a cualquiera de ellos. Y para todos los objetos que pueden ser puestos en alguna posición (jugador, personaje al que dar un objeto y objeto) se han englobado bajo el supertipo *locatable*. Todo esto se ha hecho, tal y como se ha mencionado hace un momento, con el objetivo de facilitar la generalización al poder referirnos posteriormente a cualquier tipo de personaje u objeto mediante un supertipo, en vez de tener que indicar el tipo concreto de un objeto.

A continuación se puede ver el dominio:

Listing 1: Representación del dominio.

```
(:types items npc Player - locatable
        Oscar Manzana Algoritmo Oro Rosa - items
        Princesa Principe Bruja Profesor Leonardo - npc
        Player
        zone
)
```

1.2. Ejercicio 1.b

En este apartado se han representado los predicados básicos del dominio. Para representar las orientaciones del personaje y las posiciones relativas de dos zonas adyacentes (conexas), se han utilizado constantes que hacen referencia a cada uno de los cuatro puntos cardinales (norte, sur, este y oeste). Se han elegido constantes debido a que son valores que no se van a instanciar (no se van a crear variables de éstos ya que no tiene mucho sentido tenerlas), y como tal representan más un atributo o una propiedad que un concepto u objeto.

En cuanto a los predicados, se han definido predicados para la orientación del jugador, las conexiones de zonas, las posiciones de objetos, personajes y jugador (objetos con posición) y algunos predicados más que comentaremos.

Para la orientación del jugador, se utiliza el predicado **oriented**, el cuál indica qué jugador tiene qué orientación. Para la localización de un objeto con posición se ha utilizado el predicado **at** que indica que el objeto con posición se encuentra en una determinada zona. Se puede ver que aquí se ha utilizado una abstracción sobre un conjunto de conceptos para facilitar la representación (como se indicó en el apartado anterior). Para representar las conexiones entre zonas se utiliza el predicado **connected**, el cuál especifica que la primera zona está conectada zon la segunda zona mediante un punto cardinal (el cuál reutiliza las constantes de orientación). Por ejemplo, si tuviésemos que desde una zona *z1* se puede llegar a una zona *z2* la cuál se sitúa en el sur, tendríamos **(connected z1 S z2)**. Esto se ha hecho así para facilitar más adelante la acción del giro, para poder comprobar más fácil si la orientación del jugador coincide con la de la zona a la que se quiere ir. El predicado **emptyhand** indica que un determinado jugador tiene la mano vacía, y por tanto puede coger un objeto. Es importante indicar esto, ya que si no, no habría forma de controlar si un jugador no tiene ya un objeto en la mano (recordemos que de momento solo puede llevar un objeto). El predicado **taken** representa que un objeto (tipo *item*) ha sido cogido por un jugador determinado. El predicado **given** indica que un determinado objeto ha sido entregado a cualquier personaje. Esto se hace con el objetivo de que el jugador no pueda quitarles a los personajes aquellos objetos que les hayan sido entregados. A parte de estos predicados básicos, se tiene una función numérica **received** que indica cuántos objetos ha recibido un determinado personaje. Cada vez que recibe uno, se incrementa el valor. Esto se ha hecho así para tener una especie de contador de objetos recibidos, el cuál nos será útil más adelante. A continuación se pueden ver los predicados representados:

Listing 2: Representación de predicados básicos.

```
(:constants N S E W - orientation)
(:predicates
        (oriented ?p - Player ?o - orientation)
        (at ?l - locatable ?z - zone)
        (connected ?z1 - zone ?o - orientation ?z2 - zone)
        (emptyhand ?p - Player)
        (taken ?obj - items ?p - Player)
        (given ?obj - items)
)
(:functions
        (received ?n - npc)
)
```

1.3. Ejercicio 1.c

Para este apartado se han creado acciones para girar al jugador tanto a la derecha como a la izquierda, desplazarse de una zona a otra e interactuar con objetos y personajes.

Debido a que el código de las acciones es muy extenso, no se va a adjuntar en la memoria, si no que se va a comentar su funcionamiento a algo nivel:

- Para las acciones de giro, se comprueba la orientación del personaje, y según esta orientación se le asigna una nueva dependiendo del tipo de giro. Antes de esto, obviamente, se elimina la orientación antigua.
- Para moverse de una zona *z1* a una zona *z2*, se tiene que dar que el jugador se encuentre en la zona *z1*, que las dos zonas estén conectadas y que el jugador esté orientado correctamente, ya que si no, tendrá que girar antes para orientarse correctamente. Al aplicar la acción, se cambia la posición del jugador y se elimina la anterior.
- Para coger un objeto se tiene que dar que el jugador tenga la mano vacía, ya que el jugador no puede coger más de un objeto simultáneamente (tendría que dejar el otro). Una vez que se aplica el objeto, el jugador deja de tener la mano vacía, el objeto no está en la zona en la que se encontraba y se indica que el objeto ha sido cogido por un determinado jugador.
- Para dejar un objeto, se tiene que dar que el jugador tenga cogido el objeto. Al dejarlo, se indica que el jugador ya no tiene cogido el objeto, que tiene la mano vacía y que el objeto se encuentra en la zona donde se encuentra actualmente el jugador.
- Para entregar un objeto, el jugador tiene que encontrarse en la misma zona que el personaje y tiene que tener cogido un objeto. Cuando se lo da, se indica que el jugador ya no tiene el objeto y que vuelve a tener la mano vacía. También se indica que el objeto se encuentra en la zona del personaje para intentar seguir una coherencia con la descripción del mundo, ya que al ser entregado no "desaparece", si no que el objeto se encuentra en una determinada zona del espacio, solo que ahora en posesión de un personaje. Para que no pueda volver a ser cogido y para representar que un personaje tiene posesión del objeto, se indica que el objeto ha sido entregado a un determinado personaje, y se incrementa el número de objetos que ha recibido dicho personaje. Ésto último, como se ha comentado antes, será muy significativo posteriormente.

1.4. Ejercicio 1.d

En este apartado se han planteado 2 problemas. El primero es un problema con 25 zonas con la forma de una matriz 5×5, donde cada zona está conectada con sus zonas adyacentes. Se han situado 5 personajes en el mapa (uno de cada clase) y 5 objetos (uno de cada tipo), de tal forma que estuviesen más o menos dispersos de igual forma. Se ha situado al jugador en el centro del mapa y orientado hacia el sur (como en todos los problemas, ya que no se especifica nunca que el jugador tenga una determinada orientación inicial). Se han incializado el número de objetos que ha recibido cada personaje a 0 y se ha especificado que el jugador tiene la mano vacía. El objetivo era que cada personaje tuviese al menos un objeto. Para representar este objetivo sin decir explícitamente qué jugador debe tener qué objeto, aquí ha venido muy bien el contador **received** del que se habló anteriormente, ya que simplemente bastaba con indicar que cada personaje tenía que recibir una cantidad de objetos mayor o igual que 1. Con esto, se han creado 2 planes: uno sin optimizar y uno optimizando. El resultado que se ha obtenido es que las dos planificaciones han obtenido el mismo plan, con lo cuál en este caso, la optimización no ha sido un factor muy importante.

Se ha planteado un segundo problema con 12 zonas y de nuevo 5 objetos y 5 personajes, cada uno de una clase. Se han incializado el número de objetos que ha recibido cada personaje a 0 y se ha especificado que el jugador tiene la mano vacía. El objetivo ha sido, de nuevo, que cada personaje tuviese un objeto, pero además de eso, que el jugador estuviese en la zona 3 y orientado al norte. En este caso se han obtenido de nuevo dos planes: uno sin optimización y uno optimizando. En este caso, el plan sin optimización es más largo que el que se ha obtenido optimizando con *BFS*, con lo cuál aquí si que influye la optimización, posiblemente debido a las conexiones de las zonas y a las distribuiciones de los personajes y objetos.

2. Ejercicio 2

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 2.

2.1. Ejercicio 2.a

Para considerar las distancias, se han añadido dos nuevas funciones a las que ya teníamos antes. La primera es la función **distance**, que indica el coste del camino que une una zona con la otra. La segunda es **traveled**, que indica cuánta distancia ha recorrido un determinado jugador, en caso de que haya más de uno. Se han

utilizado funciones debido a que son lo único que nos permite trabajar con valores numéricos en PDDL. A continuación se pueden ver las nuevas funciones:

Listing 3: Representación de las funciones de coste y distancia viajada.

```
(:functions
    (distance ?z1 ?z2 - zone)
    (traveled ?p - Player)
)
```

Adicionalmente se ha modificado la acción **move**, para que cuando un jugador se mueva de una zona a otra se incremente el valor de **traveled** de dicho jugador con el valor de **distance** correspondiente.

2.2. Ejercicio 2.b

En este apartado se han planteado de nuevo dos problemas. El primero de ellos tiene exactamente la misma descripción que el problema de las 25 zonas del ejercicio anterior, solo que se han añadido costes entre las zonas y se inicialización la distancia recorrida por el jugador a 0. El plan que se obtuvo del planificador sin ningún tipo de optimización era el más largo, pero a la vez también el más rápido, ya que no buscaba minimizar la distancia total viajada. Se ha probado también a obtener un plan con optimización por *BFS* y por búsqueda con $g = 1$ y $h = 1$. Los resultados es que en el primer caso se obtuvo un plan más corto que el original con un coste de 46. En el segundo caso se obtuvo un plan aún más corto que el caso anterior con un coste de 34, pero se ha tardado mucho en obtener el plan. Por tanto, podemos concluir que la búsqueda con $g = 1$ y $h = 1$ es la que más tarda en obtener un resultado, pero que siempre obtiene el camino de coste óptimo, y que el peso que se le asigna a la heurística es importante, ya que en el primer caso hace más una búsqueda heurística mientras que en el segundo caso hace una búsqueda más equilibrada.

El segundo problema sigue la misma descripción que el problema de las 12 zonas del ejercicio anterior, solo que se han añadido costes y se ha indicado que la distancia total recorrida inicialmente es 0. El plan que se ha obtenido sin optimizar es el más corto, pero no tiene en cuenta las distancias. El que se ha obtenido por *BFS* es una acción más largo que el anterior, pero obtiene el camino óptimo (a diferencia del caso anterior). La búsqueda equilibrada obtiene un plan con el mismo coste pero con 2 acciones más que el anterior, además de que tarda más tiempo en obtenerlo.

3. Ejercicio 3

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 3.

3.1. Ejercicio 3.a

En este apartado se han añadido una serie de nuevas constantes, tipos y predicados. Las constantes que se han añadido son para representar el tipo de superficie de cada zona. Se han hecho contantes ya que no interesa tener instancias de ellas debido a que son más una propiedad o atributo de un concepto que una entidad como tal.

Los tipos nuevos representan los dos nuevos tipos de objetos: la zapatilla y el bikini. Estos dos nuevos tipos han sido englobados por el supertipo *items*, ya que así es más fácil referirse a un objeto en general a la hora de cogerlo y/o dejarlo.

Los nuevos predicados permiten relacionar una zona con el tipo de superfície (esto es importante, ya que se tiene que comprobar qué superfície tiene la zona a la que se va a mover el jugador para saber si puede ir por ahí o no) y si un objeto es una zapatilla o un bikini, para saber si ese objeto se puede entregar o no a un personaje (no tiene sentido entregar estos objetos ya que sin ellos el jugador pierde capacidad para desplazarse), lo cuál se comprobará en las acciones correspondientes.

Lo dicho anteriormente se puede ver a continuación:

Listing 4: Modificación del dominio para permitir superfícies y nuevos objetos para el movimiento.

```
(:types
 Oscar Manzana Algoritmo Oro Rosa Zapatilla Bikini - items
)
(:constants
        Bosque Agua Precipicio Arena Piedra - surface
)
(:predicates
        (terrain ?z - zone ?s - surface)
        (is_zapatilla ?o - items)
        (is_bikini ?o - items)
)
```

En cuanto a las acciones, a la acción de entregar un objeto a un personaje se han añadido las precondiciones de que el objeto no tiene que ser ni una zapatilla ni un bikini (lo cuál tiene sentido por lo que se ha comentado anteriormente, además de que el hecho de tener esos dos predicados hace que sea más fácil saber si un objeto se puede entregar o no). A la acción de moverse se le pasa como parámetro también un objeto y se comprueba si a la hora de desplazarse necesita ese objeto o no (por ejemplo, si necesita ir a una zona de Bosque se comprueba si tiene un objeto de tipo zapatilla cogido). También se comprueba que la zona a la que va a desplazarse no es un precipicio, ya que el jugador no puede desplazarse por ese tipo de superfícies.

3.2. Ejercicio 3.b

En este apartado se ha añadido la posibilidad de que el jugador lleve una mochila. Esto se ha representado con dos nuevos predicados: uno que representa que un jugador determinado tiene la mochila vacía, y que por tanto puede insertar objetos dentro; y otro predicado que indica que un jugador determinado tiene un objeto en su mochila (y que por tanto no está vacía). Se ha decidido hacer esto para emular una especie de segunda mano con los predicados, ya que si hay múltiples jugadores, cada uno de ellos tiene una mochila individuale en la que pueden meter un objeto como máximo, con lo cuál no tiene sentido que haya predicados referentes a la mochila de manera global, si no específica para cada jugador (tal y como se hacía con la mano). A continuación se pueden ver estos nuevos predicados:

Listing 5: Modificación del dominio para permitir el uso de mochilas a los jugadores.

```
(:predicates
        (emptybag ?p - Player)
        (inbag ?o - items ?p - Player)
)
```

Respecto al apartado anterior, hace falta modificar algunas acciones de nuevo y añadir algunas nuevas. Se han añadido acciones para meter y sacar un objeto de la mochila. En el primer caso, el jugador tiene que tener la mochila vacía y el objeto en la mano, y pasa a tener el objeto en la mochila y la mano vacía. En el segundo caso, tiene que tener la mano vacía y un objeto en la mochila, y pasa a tener el objeto en la mano y la mochila vacía. A la hora de mover un jugador, se comprueba también si tiene el objeto en la mochila. A la hora de soltar un objeto, se comprueba antes si tiene espacio en la mochila, ya que no tiene mucho sentido soltar un objeto y coger otro cuando el personaje tiene la mochila vacía.

3.3. Ejercicio 3.c

De nuevo se han probado dos problemas. El primero es el problema de las 25 zonas del ejercicio anterior, añadiendo ahora superfícies que obliguen al jugador a plantearse si necesita coger un objeto que le permita desplazarse, además de un bikini y una zapatilla, los cuáles se usarán para desplazarse, y se ha indicado que el jugador tiene la mochila vacía. El planificador encontró un plan en muy poco tiempo sin dar ningún item que no tocaba a ningún personaje. Se ha intentado optimizar el problema, pero debido al tamaño de este se obtenía una violación de segmento, con lo cuál a partir de aquí se dejará de intentar optimizar este problema.

El segundo problema es una modificación del de 12 zonas de antes añadiendo una nueva zona que es un precipicio. Como en el caso anterior, se han añadido superfícies, los nuevos objetos y la mochila vacía. El planificador encontró un plan tanto sin optimizar como con los dos tipos de optimizaciones en un tiempo muy pequeño. Pero, a medida que se iba optimizando la distancia recorrida, el tamaño del plan iba aumentando. De hecho, entre los dos planes optimizados, a pesar de tener el mismo coste, hay más acciones en el plan con la heurística equilibrada que con la de *BFS*.

4. Ejercicio 4

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 4.

4.1. Ejercicio 4.a

En este apartado se han creado dos nuevas funciones. La primera es una función que almacena la cantidad de puntos totales que se han obtenido. De momento, esta función solo se puede utilizar para un jugador, y sirve como un contador de la cantidad de puntos que ha obtenido este jugador. Sin embargo, en ejercicios posteriores esta función será utilizada para acumular la cantidad de puntos que han conseguido todos los jugadores en conjunto, ya que como tal la función no hace referencia a ningún jugador en concreto, si no a todos los que hay en el mapa. También tiene la ventaja de que esta función se puede utilizar también para especificarle al jugador cuántos puntos se quieren conseguir (al ponerla en *goal*. La segunda función que se ha implementado es una que permite asociar una cantidad de puntos que ortorga uno de los personajes del mapa al entregarle uno de los objetos que hay en el mapa. Se ha diseñado de esta forma ya que en PDDL no se pueden representar funciones entre dos tipos (el tipo de personaje y el tipo de

objeto, por ejemplo), si no que se tiene que representar con las instancias. Esto, por una parte, implica que las descripciones de los problemas serán más grandes, ya que para cada instancia de objeto, se tiene que escpecificar cuántos puntos proporciona **cada una de las instancias de los personajes**. Sin embargo, esto facilita algunas cosas, y es que no se tiene que comprobar de qué tipo es el objeto que se entrega y de qué tipo es el personaje para poder obtener la cantidad de puntos que se ortorgan, si no que simplemente basta ver la cantidad de puntos asociada a las dos instancias.

A continuación se pueden ver las dos nuevas funciones:

Listing 6: Modificación del dominio para permitir puntuaciones.

```
(:functions
    (total_score)
    (score ?n - npc ?o - items)
)
```

Las acciones también se han visto modificadas, ya que ahora al entregar un objeto a un personaje, se tiene que contabilizar cuántos puntos se le proporcionan. Por tanto, cada vez que se le entregue un objeto a un personaje, **total_score** se va a ver incrementado en una cantidad especificada por la función **score**.

Para ilustrar mejor esto, si por ejemplo el jugador le entrega un objeto **manzana1** a un personaje **bruja1**, se verá que la función **(total_score)** se ha incrementado en la cantidad de puntos dada por la función **(score bruja1 manzana1)**.

4.2. Ejercicio 4.b

En este apartado se han probado tres problemas. El primero es una modificación del problema de 25 zonas, dejando las mismas zonas y personajes pero cambiando los objetos para que sean solo manzanas y oscars. También se han generado todos los puntos que ortorga cada uno de los personajes por cada objeto. El objetivo era conseguir 80 puntos como mínimo. El planificador es capaz de encontrar un plan muy rápidamente en el que en total consigue 84 puntos.

El segundo es una modificación del problema de las 13 zonas, dejando las zonas tal y como estaban y cambiando todos los objetos por oscars y manzanas y añadiendo más objetos de estos tipos. También se han generado todos los puntos que ortorga cada uno de los personajes por cada objeto. Se especificaba que se consiguiesen 50 puntos mínimo, y se consiguieron exactamente 50 puntos. Además,

se probó a optimizar el plan con una búsqueda equilibrada, y se consiguió un plan con la misma longitud. Sin embargo, el tiempo que se ha necesitado para obtener el plan, a pesar de que el problema no era muy grande, fue de poco más de 2 minutos. Con lo cuál, a partir de ahora no se probará a optimizar ningún problema de nuevo, ya que los tiempos de ejecución se han visto incrementados demasiado.

El tercero es una extensión del problema de 25 zonas, solo que esta vez conservando los mismos objetos, personajes y zonas de antes y añadiendo objetos nuevos de distintas clases. También se han generado todos los puntos que ortorga cada uno de los personajes por cada objeto. Se ha especificado que se obtuviesen 80 puntos, y el planificador encontró un plan en el que se entregaban todos los objetos y el jugador conseguía 82 puntos en total, consiguiendo por tanto el objetivo propuesto.

5. Ejercicio 5

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 5.

5.1. Ejercicio 5.a

En este apartado se ha modificado el dominio para poder representar que existen personajes con bolsillo y cuál es su capacidad.

Para lo primero, se ha utilizado un predicado nuevo que indica que un determinado personaje tiene bolsillo, y por tanto, que tiene una capacidad limitada de recibir objetos. Esto se ha hecho para poder distinguir a los personajes con bolsillos de los que no tienen, lo cuál influirá en la acción que se aplique en cada caso al entregar un objeto, ya que ahora hay dos.

Para lo segundo, se ha creado una función que indica cuál es la capacidad del bolsillo del personaje. Esto se ha hecho así para aprovechar la función que teníamos antes que contaba cuántos objetos había recibido un personaje. Si se da el caso de que el número de objetos recibidos coincide con la capacidad del bolsillo, no se le pueden entregar más objetos a ese personaje (en caso de disponer de bolsillo; los personajes sin bolsillo pueden recibir infinitos objetos).

Las acciones también se han visto modificadas. Ahora existen dos acciones para entregar objetos. Una es una modificación de la que ya teníamos antes, solo que ahora se comprueba que el personaje en cuestión no tiene bolsillo. La segunda es una extensión para los personajes con bolsillos, donde se comprueba que el personaje tiene bolsillo y que la capacidad del bolsillo del personaje es mayor que la cantidad

de objetos que se le han entregado hasta el momento. Lo que se modifica dentro de estas dos acciones, sin embargo, no ha cambiado.

Los nuevos predicados y funciones que se añaden se pueden ver a continuación:

Listing 7: Modificación del dominio para permitir el uso de bolsillos para los personajes.

```
(:predicates
        (has-pocket ?n - npc)
)
(:functions
    (pocket-capacity ?n - npc)
)
```

5.2. Ejercicio 5.b

En este apartado se han probado tres problemas. El primero es una modificación del problema de las 25 zonas del ejercicio anterior en el que había muchos objetos de todos los tipos (el tercer problema). En este caso, se ha indicado que **leonardo1** y **profesor1** ambos tenían bolsillos y una capacidad de bolsillo de 2. El objetivo era conseguir 60 puntos entregando objetos. El planificador consiguió encontrar un plan en el que entregaba la mayoría de objetos, sin pasarse nunca de la cantidad de objetos que tenía que entregar a los personajes con bolsillos consiguiendo en total 61 puntos.

El segundo problema es una modificación del problema de las 13 zonas del ejercicio anterior (solo con manzanas y oscars). Todos los personajes tienen bolsillos, y el personaje **witch1** tiene un bolsillo de capacidad 0. El resto tienen bolsillos de capacidades iguales o superiores a 1. El personaje **leo1** tiene un bolsillo de capacidad 3. El objetivo era conseguir 20 puntos, forzando que no pueda entregar manzanas a la bruja y que entregase oscars a Leonardo, ya que es el que mayores puntos permitía obtener. El planificador obtuvo un plan en el que se entregaban 4 objetos y conseguía 24 puntos. En este caso, solo entregó un objeto a **leo1** (el objeto final), posiblemente por como estuviesen distribuidos los objetos y los personajes.

El tercer problema es una modificación del anterior, solo que ahora **leo1** tiene una capacidad de bolsillo de 0 (no puede recibir objetos) y el personaje **witch1** tiene una capacidad de bolsillo de 1. El resto de personajes tiene una capacidad de bolsillo de 1. El objetivo es de nuevo conseguir 20 puntos, solo que esta vez el jugador está mucho más limitado a la hora de entregar objetos a personajes, ya que muchos de los personajes le darán una cantidad de puntos subóptima con los

objetos que hay en el mapa, ya que solo la bruja le podrá aportar 10 puntos. El planificador encontró un plan en el que consguía que el jugador obtenía exactamente 20 puntos, entregando eso sí una de las manzanas a la bruja.

6. Ejercicio 6

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 6.

6.1. Ejercicio 6.b

En este apartado se ha modificado el dominio del problema y se han planteado 3 problemas al planificador para ver si era capaz de resolverlos.

En cuanto a las modificaciones, se ha añadido una nueva función que permita ir acumulando la puntuación que recibe cada jugador, aprovechando la función **total** para acumular la puntuación que han conseguido entre todos los jugadores. De esta forma, se pueden contabilizar los puntos específicos de cada jugador y los globales por separado, pudiendo por tanto especificar objetivos sobre cuántos puntos debe conseguir cada jugador y cuántos puntos se deben conseguir en total.

Este añadido se puede ver en la siguiente función:

Listing 8: Modificación del dominio para permitir contabilizar los puntos de cada jugador.

```
(:functions
    (player-score ?p)
)
```

En cuanto a las acciones modificadas, se han modificado las dos acciones de entregar un objeto. Ahora cada una de ellas acumula en los puntos del jugador que le ha entregado el objeto, los puntos que se reciben por entregar ese objeto a ese personaje.

Pasando ahora a hablar de los problemas, el primero de ellos es una modificación del problema de las 25 zonas del ejercicio anterior, solo que ahora con 2 jugadores y cada uno de ellos tiene que obtener un mínimo de 20 puntos, consiguiendo en total entre los dos 60 puntos. También se ha especificado que los puntos que tienen

inicialmente los jugadores son 0. El planificador fue capaz de encontrar un plan en el que ambos consiguiesen la cantidad mínima de puntos, aun teniendo que pasar por ejemplo por un bosque. Sin embargo, la cantidad individual de puntos que conseguía cada uno no estaba equilibrado (había un jugador que conseguía más puntos que el otro).

El segundo y tercer problema son una modificación del problema de las 13 zonas con solo manzanas y oscars. Todos los personajes tienen bolsillos de capacidad mayor o igual a 1, y en los dos problemas los dos jugadores tienen que obtener un mínimo de 10 puntos cada uno. También se ha especificado que los puntos que tienen inicialmente los jugadores son 0. En uno de los problemas se tienen que obtener 20 puntos en total, mientras que en el otro 30. El planificador es capaz de encontrar en ambos casos planes para resolver los problemas, resolviéndolos satisfactoriamente y consiguiendo las cantidades necesarias de puntos.

7. Ejercicio 7

A continuación se ofrece una descripción de las decisiones de diseño tomadas a la hora de realizar el ejercicio 7.

7.1. Ejercicio 7.a

En este apartado se ha modificado el dominio de los ejercicios anteriores para representar los dos nuevos tipos de jugadores: el *Piker* y el *Dealer*. Estos dos nuevos tipos se han creado como subtipos del tipo *Player* que teníamos anteriormente con el objetivo de abstraer los dos tipos para las acciones comunes que tienen (como por ejemplo girar, moverse, meter un objeto en la mochila y sacarlo y soltar un objeto). Estos nuevos tipos se pueden ver a continuación:

Listing 9: Modificación del dominio para permitir crear nuevos tipos de jugadores.

```
(:types
        Picker Dealer - Player
)
```

En lo referente a las acciones, se ha modificado la acción de coger un objeto del suelo para que solo la pueda llevar a cabo un jugador del tipo *Picker*. Las dos acciones de entregar objetos a personajes con y sin bolsillo han sido modificadas para que solo las pueda llevar a cabo un personaje de tipo *Dealer*. Se ha creado

una nueva acción para que un jugador de tipo *Picer* pueda darle un objeto a un jugador de tipo *Dealer*. Para ello, los dos jugadores tienen que estar en la misma zona, el *Picker* tiene que tener el objeto cogido y el *Dealer* tiene que tener la mano vacía. El resultado es que el *Picker* pasa a no tener el objeto cogido y a tener la mano vacía y el *Dealer* pasa a tener el objeto cogido y deja de tener la mano vacía.

7.2. Ejercicio 7.b

De nuevo, como se ha realizado anteriormente, se han probado tres problemas. El primero es una modificación del problema de las 25 zonas del apartado anterior, cambiando los tipos de los jugadores para que uno haga el papel de *Picker* y el otro el papel de *Dealer*. También se ha especificado que el *Dealer*, como mínimo, tiene que conseguir como mínimo 20 puntos y un mínimo de 60 puntos en total (tendrá que hacer todo el tabajo en este caso al no tener otro *Dealer*). También se ha especificado que los puntos que tiene inicialmente el *Dealer* son 0 puntos. El planificador fue capaz de encontrar un plan, tardando sin embargo aproximadamente 18 segundos en encontrarlo.

El segundo y el tercer problema son una modficiación del problema de las 13 zonas del ejercicio anterior. En el primero de ellos hay de nuevo un *Picker* y un *Dealer*, y el *Dealer* tiene que conseguir un mínimo de 10 puntos, y en total se tienen que conseguir 20 puntos (de nuevo tiene que hacer todo el trabajo). En cambio, en el otro problema hay un *Picker* y dos jugadores de tipo *Dealer*, y cada uno de ellos tiene que conseguir un mínimo de 10 puntos, llegando entre los dos a 30 en total. En los dos problemas se especificó que la cantidad de puntos que tenían los *Dealer* inicialmente era 0. En ambos casos, el planificador consiguió encontrar un plan que satisfaciese las restricciones, demostrando que funciona incluso para más de un *Dealer*. Sin embargo, en el segundo caso tardó poco más de 1 minuto en obtener el plan, con lo cuál, parece que aún para un problema simple, si se empiezan a tener demasiados elementos en el problema el planificador tiene problemas para encontrar planes que cumplan los objetivos en un tiempo pequeño.