Memoria Práctica 2:

Dominios y Problemas de Clasificación Clásica en PDDL

Técnicas de los Sistemas Inteligentes

Christian Vigil Zamora

3º - Grupo 2

Índice

1. EJERCICIO 1

APARTADO A)

Para la representación de los objetos del mundo, he definido una serie de Objetos. La zona **'zone'** y la orientación **'orientation'** , las he considerado de tipo **'object'**, al igual que el tipo **'locatable'** el cual va a representar tanto a personajes como objetos. El hecho de que su tipo sea **'locatable'** quiere decir que son localizables dentro del mapa. Por lo tanto, los objetos que agrupa el tipo **'locatable'** son **'person'** e **'item'**, los cuales identifican a personas u objetos. Dentro del tipo 'person' se encuentran representados tanto los posibles jugadores como los personajes, mediante los tipos **'player'** y **'character'** respectivamente. Como se puede ver, los tipos de objetos elegidos son muy genéricos, de forma que permite que pueda varios de ellos. Captura:

```
(:types zone orientation locatable - object
        person item - locatable
        player character - person)
```

APARTADO B)

En cuanto a los predicados, he necesitado definir 5 para representar todos los posibles estados del mundo. El primero de ellos es **'rute'**, el cual representa la conexión entre 2 zonas. Su sintaxis es de tipo '**rute** zone1 zone2 orientation', entendiéndose como que para ir desde zona1 hasta zona2, se debe tener la orientación indicada. Para representar la localización de jugador/es, personajes y objetos, he definido el predicado '**location** locatable zone', que quiere decir que un objeto elemento localizable está en 'x zona. La orientación del jugador es crucial, por tanto aparece el predicado 'looking jugador orientación', que expresa que el jugador está orientado hacia 'x orientación. Tanto el jugador como los personajes pueden tener un objeto, es por ello que defino el predicado 'has person item' que indica que el jugador o un personaje posee un objeto. Por último, para expresar justo lo contrario, cuando el jugador o un personaje no tienen ningún objeto, he definido el predicado 'handempty person'. Captura:

```
(:predicates
        (rute ?z1 - zone ?z2 - zone ?o - orientation)
        (location ?l - locatable ?z - zone)
        (looking ?pl - player ?o - orientation)
        (has ?p - person ?i - item)
        (handempty ?p - person)
)
```

APARTADO C)

A continuación describo las acciones que pueden ser llevadas a cabo por el jugador. La primera de ellas es **'turn_left'**, acción usada cuando el jugador quiere girar a la izquierda. Los parámetros necesarios sería el jugador por supuesto, y su orientación. La única que precondición exigida en ésta acción es que el jugador esté orientado, sea cual sea su orientación. El efecto de dicha acción sería comprobar la orientación actual del jugador e ir alternando en función de ello. Es decir, si el jugador está orientado hacia el Norte y quiere girar a la izquierda, su nueva orientación será el Oeste, y así sucesivamente con todas las posibilidades. Una vez se establece la nueva orientación del jugador, se niega la que tenía anteriormente. Queda mejor reflejado en la captura siguiente:

```
(:action turn-left
       :parameters (?pl - player ?o - orientation)
       :precondition (and (looking ?pl ?o))
           :effect (and (when (and (looking ?pl Norte))(and (looking ?pl Oeste)
                              (not (looking ?pl Norte))))
                        (when (and (looking ?pl Sur))(and (looking ?pl Este)
                              (not (looking ?pl Sur))))
                        (when (and (looking ?pl Este))(and (looking ?pl Norte)
                              (not (looking ?pl Este))))
                        (when (and (looking ?pl Oeste))(and (looking ?pl Sur)
                              (not (looking ?pl Oeste))))
                    )
)
```

La acción **'turn_right'**, usada cuando el jugador quiere girar a la derecha, toma como descripción la misma que la anterior comentada, a excepción de que en éste caso, se gira en sentido contrario, y por tanto la orientación se comporta de forma contraria a la acción anterior. Captura:

```
(:action turn-right
       :parameters (?pl - player ?o - orientation)
       :precondition (and (looking ?pl ?o))
           :effect (and (when (and (looking ?pl Norte))(and (looking ?pl Este)
                              (not (looking ?pl Norte))))
                        (when (and (looking ?pl Sur))(and (looking ?pl Oeste)
                              (not (looking ?pl Sur))))
                        (when (and (looking ?pl Este))(and (looking ?pl Sur)
                              (not (looking ?pl Este))))
                        (when (and (looking ?pl Oeste))(and (looking ?pl Norte)
                              (not (looking ?pl Oeste))))
                    )
)
```

La acción **'go'** es la que se activa cuando el jugador quiere ir de una zona a otra. Por lo tanto, sus parámetros son: el jugador, la zona en la que se encuentra el jugador, la zona a la que desea ir y su orientación. Para que pueda activarse deben cumplirse 3 precondiciones principalmente, siendo una que el jugador se encuentre en la primera zona pasada como parámetro, otra que el jugador esté orientado en la orientación correcta para poder ir desde su zona a la otra, y la última que exista una relación entre la zona en la que se encuentra y la que desea ir, con la orientación correcta. Sus efectos son bastantes obvios, el jugador deja de estar en la zona inicial y pasa a estar en la otra zona. Captura:

```
(:action go
       :parameters (?pl - player  ?z1 - zone ?z2 - zone ?o - orientation)
       :precondition (and (location ?pl ?z1) (looking ?pl ?o) (rute ?z1 ?z2 ?o))
       :effect (and (not (location ?pl ?z1)) (location ?pl ?z2))
  )
```

Para representar la opción de coger un objeto, he definido la acción **'pick-up'**, la cual recibe como parámetros el jugador, una zona y un objeto. Para su activación, debe cumplirse que tanto el jugador como el objeto que desea coger se encuentren en la misma localización, es decir en la misma zona. El efecto de dicha acción sería que el jugador deja de tener las manos vacías, lo que indica que el jugador pasa a tener el objeto deseado y se indica que dicho objeto ya no se encuentra en esa zona. Captura:

```
(:action pick-up
       :parameters (?pl - player ?z - zone ?i - item)
       :precondition (and (location ?pl ?z) (location ?i ?z)
                          (handempty ?pl))
       :effect
       (and (not (handempty ?pl)) (has ?pl ?i)
                          (not (location ?i ?z)))
  )
```

Para representar justo la opción contraria, dejar un objeto, he definido la acción **'put-down'**, cuyos parámetros serían de nuevo el jugador, una zona y un objeto. En ésta ocasión, se debe cumplir para que se active, que el jugador posea el objeto y que el jugador se encuentre en una zona. Si se cumplen ambas, el efecto que produce ésta acción sería que el jugador deja de poseer el objeto, que el objeto pasa a estar en esa zona y que el jugador vuelve a tener las manos vacías. Captura:

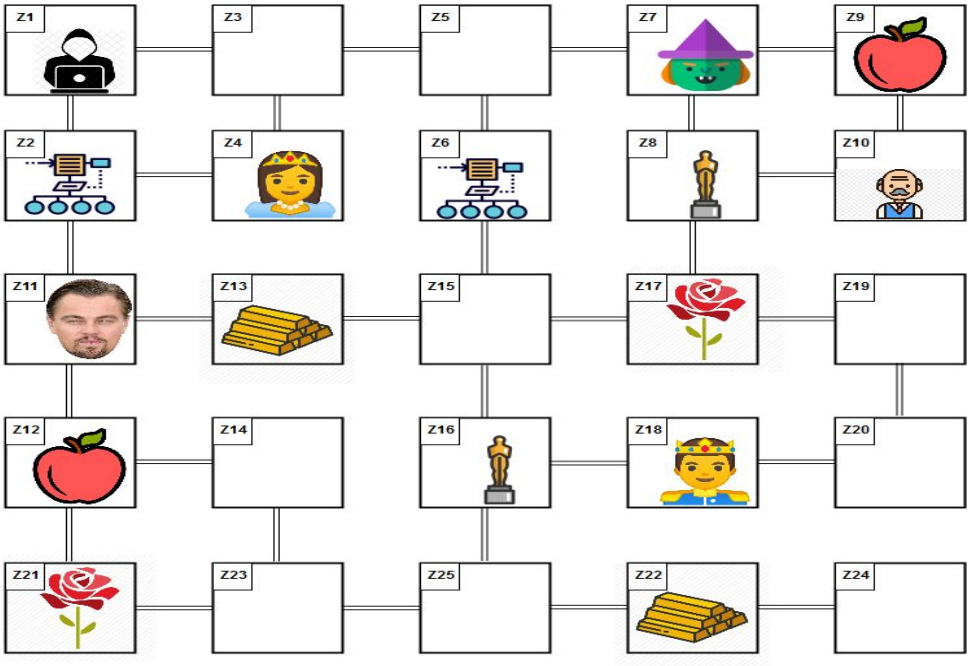
```
(:action put-down
       :parameters (?pl - player ?z - zone ?i - item)
       :precondition (and (has ?pl ?i) (location ?pl ?z))
       :effect
       (and (not (has ?pl ?i)) (location ?i ?z) (handempty ?pl))
  )
```

La última acción definida es **'give'**, encargada de representar la acción de cuando un jugador entrega un objeto a un personaje. Los parámetros que recibe son el jugador, un personaje, un objeto y una zona. Se debe cumplir que tanto el jugador como el personaje estén en la misma zona, además el jugador debe tener un objeto y el personaje para poder recibirlo debe tener las manos vacías. Su efecto sería que el jugador deja de tener ese objeto y lo pasa a tener el personaje, además el jugador pasaría a tener las manos vacías, mientras que el personaje pasaría a tener el objeto. Captura:

```
(:action give
       :parameters (?pl - player ?c - character ?i - item ?z - zone)
       :precondition (and (location ?pl ?z) (location ?c ?z)
                          (has ?pl ?i) (handempty ?c))
       :effect
       (and (not (has ?pl ?i)) (has ?c ?i) (handempty ?pl) (not (handempty ?c)))
  )
```

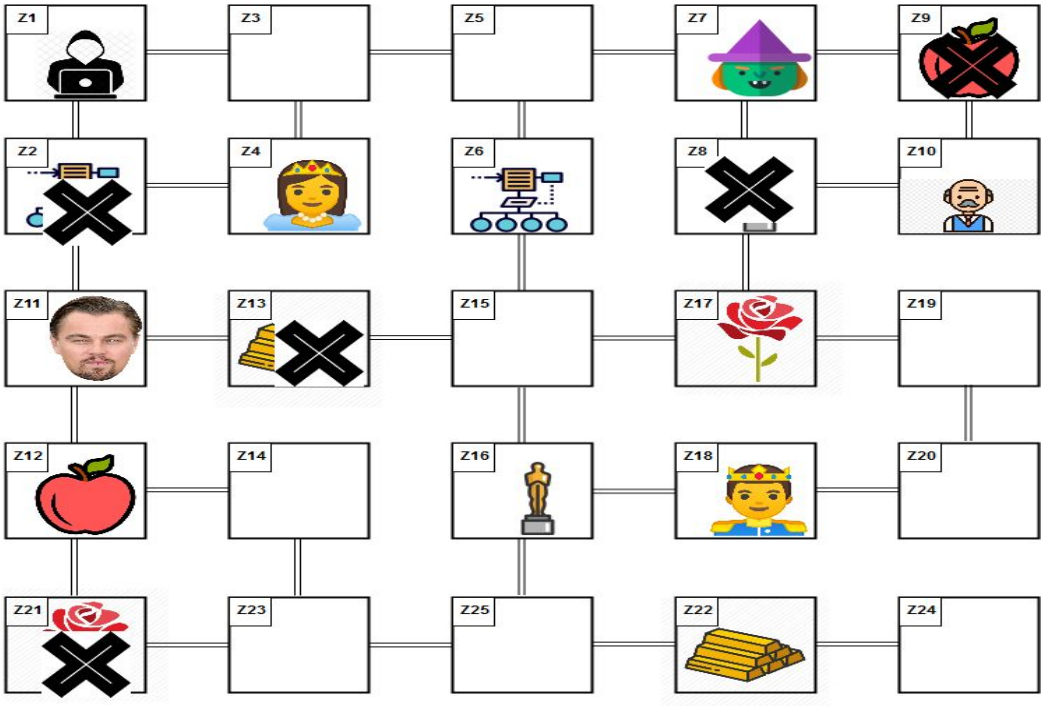
APARTADO D)

Para la ejecución del ejercicio he definido 2 problemas adaptados al dominio descrito anteriormente. El primero de ellos tendría la siguiente forma:



Como se puede apreciar, el mapa permite perfectamente que el objetivo del ejercicio pueda ser llevado a cabo, pues cada personaje debe tener al menos un objeto. Para ello, he creado un mapa compuesto por 25 zonas, por las que he distribuido aleatoriamente 5 personajes. El jugador se encuentra en la primera casilla, en mi caso el jugador es un Hacker. También he definido 10 objetos por el mapa, siendo sólo 5 necesarios para lograr el objetivo. Para indicar el objetivo del problema he recurrido a la cláusula 'exists', la cual utilizo para indicar que existen al menos 5 objetos posibles que deben ser distribuidos a los personajes.

Para el segundo problema, mantengo todo lo del anterior a excepción de que reduzco el número de objetos en el mapa de 10 a 5, de forma que sólo existe 1 objeto posible para cada personaje. Mapa:



APARTADO E)

He empleado el mapa proporcionado en el pdf, es decir, un problema bastante básico compuesto por 7, en la que hay 1 jugador, 2 personajes y 2 objetos.
De momento nada relevante a destacar del generador de problemas. Más adelante sí.

2. EJERCICIO 2

APARTADO A)

Para satisfacer la nueva característica introducida en éste ejercicio, he declarado funciones. La declaración que he realizado ha sido la siguiente:

```
(:functions
  (distance ?x ?y - zone)
  (total_distance)
)
```

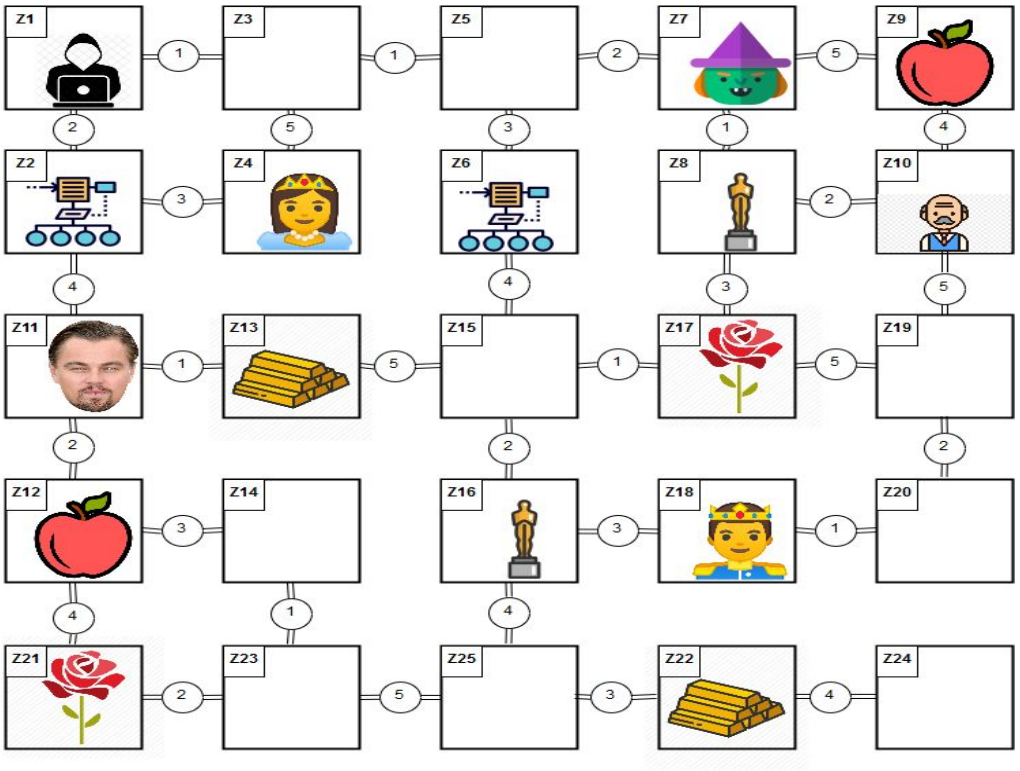
A partir de ahora, la existe una distancia entre las zonas, es por ello que recurro a la función '**distance** ?x ?y', ya que me permite definir la distancia que existe entre 2 zonas. En mi caso, la variable ?x representa una zona y la variable ?y representa otra. De esa forma, podemos declarar las distancias entre zonas. En el siguiente apartado comentaré la sintaxis de dicha función en el problema. La función **'total_distance'** simplemente ejerce como contador y representa la distancia total acumulada hasta el momento. Como ahora la acción de desplazamiento entre zonas tiene un coste igual a la longitud del camino entre cada zona, modifico la acción **'go'**, quedando así:

```
(:action go
      :parameters (?pl - player  ?z1 - zone ?z2 - zone ?o - orientation)
      :precondition (and (location ?pl ?z1) (looking ?pl ?o) (rute ?z1 ?z2 ?o))
      :effect (and (not (location ?pl ?z1)) (location ?pl ?z2) (increase (total_distance) (distance ?z1 ?z2)))
 )
```

La novedad que he incorporado con respecto al ejercicio anterior es que cada vez que se ejecute la acción de ir de una zona a otra, por medio del operador **'increase'** aumento tanto como sea la longitud del camino entre ambas zonas.

APARTADO B)

Para los problemas de éste ejercicio, sigo conservando el mapa anterior aunque le he añadido la novedad de que entre las ciudades existe una distancia, quedando así:



En cuanto a la implementación del problema, también hay novedades. Ahora, inicializo el valor de las distancias entre las ciudades por medio de la función comentada anteriormente **'distance'**, siendo su sintaxis la siguiente:

```
(= (distance z25 z22) 3)
(= (distance z25 z16) 4)
```

También es importante comentar que puesto que ahora tenemos un contador almacenando la distancia total recorrida hasta el momento, inicializo a 0 la función **'total_distance'**:

```
(= (total_distance) 0)
```

Por último, comentar que en éste problema, como la novedad es la distancia entre zonas, voy a utilizar la métrica **'minimize'** para que la resolución del ejercicio sea llevada a cabo en la menor distancia posible, es decir, minimizando la distancia total recorrida. Para ello, he ejecutado con '-O -g 1 -h 1'.

APARTADO C)

Mantengo el mismo mapa e introduzco la distancia entre zonas.
De momento nada relevante a destacar del generador de problemas. Más adelante sí.

3. EJERCICIO 3

APARTADO A)

En primer lugar, para considerar que hay distintos tipos de zonas en función de la superficie, añado un tipo para representar los objetos, en éste caso los tipos de superficie, por lo que el tipo añadido es **'type'**. Definido el tipo para representar los tipos de superficie, llego a la conclusión de que hace falta un predicado, en concreto, **'surface** zone type' empleado para indicar el tipo de superficie de cada zona. Los nuevos objetos Zapatilla y Bikini los considero de tipo **'item'** también, al igual que el resto de objetos. Como ahora el jugador está dotado de una mochila, defino el predicado **'bag** player item) mediante el cual represento cuando el jugador tiene un objeto en la mochila. Con la aparición de la mochila, creo otro predicado para representar el estado de que la mochila está vacía: '**bagempty** pl'. Puesto que aparecen 2 restricciones nuevas, defino el predicado **'necessary** type item', es decir, que para una superficie concreta, hace falta el elemento que se especifique, de esa forma puedo expresar que para moverse a una zona de Bosque sea necesario tener una Zapatilla y que para moverse a una zona de Agua sea necesario tener un Bikini. Por último, para facilitar la gestión de las superficies y evitar comprobaciones, he declarado el predicado **'allow** type' con el que indico las superficies por las que es posible pasar sin restricción alguna. De esa forma, cumplo también la condición de que no pueda moverse a un precipicio, pues si no declaro con 'allow' el precipicio, descartará pasar por zonas que lo sean.

```
(surface ?z - zone ?t - type)
(bag ?pl - player ?i - item)
(bagempty ?pl - player)
(necessary ?t - type ?i - item)
(allow ?t - type)
```

Todas éstas novedades conllevan actualizar la acción **'go'**, quedando así:

```
(:action go
      :parameters (?pl - player  ?z1 - zone ?z2 - zone ?o - orientation ?t - type ?i - item)
      :precondition (and (location ?pl ?z1) (looking ?pl ?o) (rute ?z1 ?z2 ?o)
                         (surface ?z2 ?t) (or (allow ?t) (and (or (has ?pl ?i) (bag ?pl ?i)) (necessary ?t ?i))))
      :effect (and (not (location ?pl ?z1)) (location ?pl ?z2) (increase (total_distance) (distance ?z1 ?z2)))
 )
```

Ahí quedan reflejadas las nuevas condiciones, puesto que en las precondiciones para ir de una zona a otra se incluye que la zona a la que se desea sea de una superficie sin restricciones (allow) o en su defecto, si es una zona cuya superficie requiere algún objeto especial, se compruebe que tiene ese objeto o bien cogido o en la mochila. También es necesario actualizar la acción **'give'**, quedando así:

```
(:action give
       :parameters (?pl - player ?c - character ?i - item ?z - zone)
       :precondition (and (location ?pl ?z) (location ?c ?z)
                          (has ?pl ?i) (handempty ?c) (not (has ?pl Zapatillas)) (not (has ?pl Bikini)))
       :effect
       (and (not (has ?pl ?i)) (has ?c ?i) (handempty ?pl) (not (handempty ?c)))
  )
```

Puesto que ahora tenemos nuevos objetos: Zapatillas y Bikini, tenemos que incluir en las precondiciones de dicha acción que si el jugador posee las Zapatillas o el Bikini, dichos objetos no pueden ser entregados a personajes, por tanto se comprueba que para que se active la acción no posea esos objetos.

APARTADO B)

Para contemplar que el jugador pueda meter y sacar objetos en/de la mochila he considerado que la mejor opción es definir 2 acciones. La primera es:

```
(:action put
       :parameters (?pl - player ?i - item)
       :precondition (and (has ?pl ?i) (bagempty ?pl) (not (handempty ?pl)))
       :effect
       (and (not (has ?pl ?i)) (bag ?pl ?i) (not (bagempty ?pl)) (handempty ?pl))
  )
```

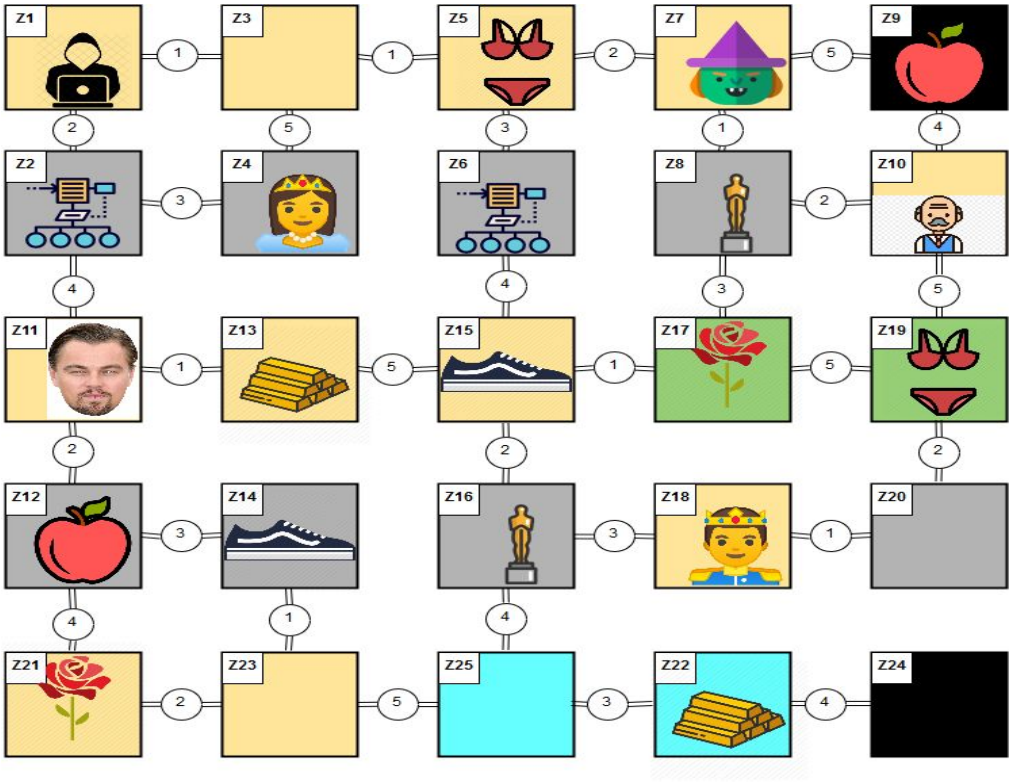
Ésta acción se ejecuta cuando el jugador quiere meter un objeto en la mochila, por tanto los parámetros que recibe son un jugador y un objeto. Para que se active dicha acción, el jugador debe tener el objeto cogido además de tener la mochila vacía, pues sólo puede guardar uno. Si se activa, el efecto es que el jugador deja de tener el objeto cogido, por lo tanto pasa a tener las manos vacías y la mochila deja de estar vacía. La segunda acción es:

```
(:action take
       :parameters (?pl - player ?i - item)
       :precondition (and (bag ?pl ?i) (handempty ?pl) (not (bagempty ?pl)))
       :effect
       (and (not (bag ?pl ?i)) (has ?pl ?i) (not (handempty ?pl)) (bagempty ?pl))
  )
```

Con ésta acción se lleva a cabo el hecho de sacar un objeto de la mochila, por ello recibe como parámetros un jugador y un objeto. Para que se active, el jugador debe poseer el objeto que se desea soltar, en la mochila, además debe tener las manos vacías pues el objeto pasará de la mochila a sus manos. El efecto de esta acción sería que el objeto pasa a no encontrarse en la mochila, el jugador pasa a tenerlo, por tanto el jugador deja de tener las manos vacías y la mochila pasa a estarlo.

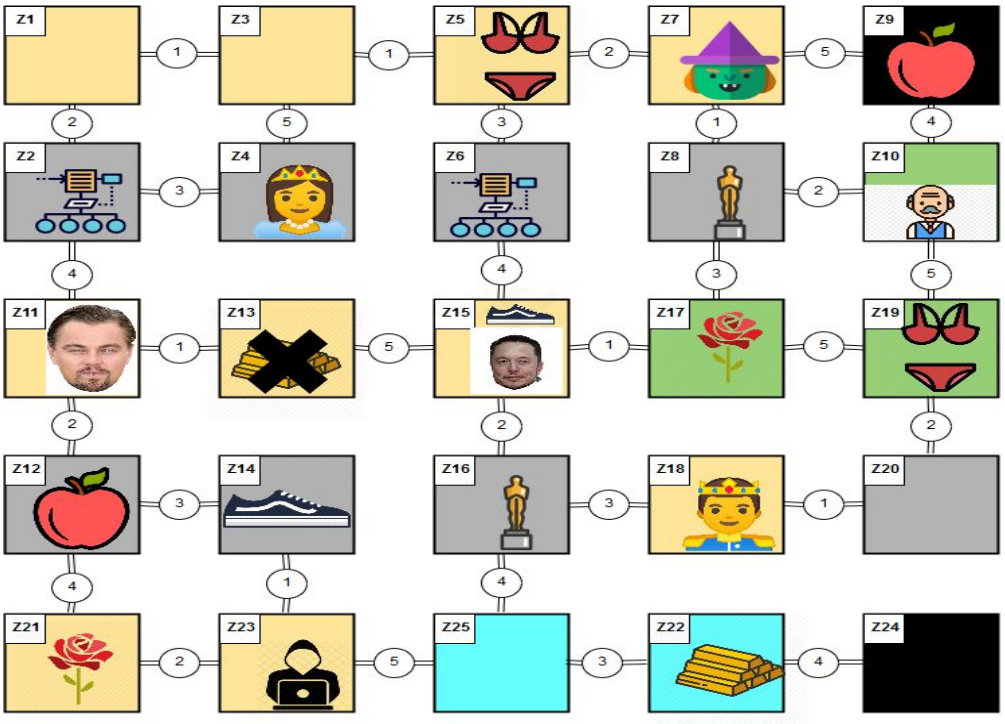
APARTADO C)

La extensión del problema que he llevado a cabo ha sido añadir predicados de la superficie de las zonas en formato: '(**surface** z1 Arena)', además de indicar cuáles son las superficies sin restricciones, pe: '(**allow** Arena)' y cuáles son las superficies con restricciones, pe: '(**necessary** Bosque Zapatilla)'. Por lo demás, sigue siendo igual, además de tener el mismo objetivo. El mapa ha quedado así:



He cumplido también con la condición de que al menos haya un bikini en una zona de Bosque o una zapatilla en una zona de Agua.

También he considerado un segundo problema, en el que evidencio aún más la obligación de ser necesario tener 'x' objeto para pasar por algunas zonas, es por ello que he creado ésta versión del mapa:



El primer cambio es que he cambiado la posición del jugador, pasando a empezar en una posición totalmente opuesta con respecto al anterior problema. También, he añadido que la superficie de la zona en la que se encuentra el Profesor sea Bosque, de forma que para cumplir el objetivo de que cada personaje tenga mínimo un objeto, va a tener que coger las Zapatillas si o sí para poder entregarle al Profesor un objeto. Por otra parte, para obligarle a usar el Bikini, he reducido el número de objetos de Oro a 1 y en el objetivo del problema he indicado que el Príncipe tiene que recibir el objeto Oro a la fuerza, de forma que el jugador va a tener coger un Bikini para poder coger el Oro. De esa forma, muestro el cumplimiento de las nuevas condiciones.

APARTADO D)

Con respecto a éste apartado, puesto que en mi dominio no he considerado añadir los objetos en formato 'oro1, manzana2...etc' sino que como los objetos son un tipo genérico de representación, basta con indicar 'Manzana' a secas y añadir tantas como se desee, entonces a la hora de generar un problema, mi dominio puede trabajar con los objetos en formato 'oro1,manza2...' sin problemas excepto con las Zapatillas y el Bikini, y eso es debido a que tengo en mi dominio el predicado '(**necessary**)' con el que generalizo que para Bosque por ejemplo hace falta un objeto de tipo Zapatillas, además que para futuros ejercicios en los que modifico algunas acciones, incluyo en las precondiciones Zapatillas y Bikini. Por lo tanto, para solucionar eso, lo único que he hecho es que conforme leo del mapa a generar Zapatillas o Bikinis en formato 'zapatillas1, bikini2...' los adapto a mi dominio, pasando a ser Zapatillas o Bikini a secas, y añadiendo tantos como encuentre. El motivo de hacer ésto es porque el generador de problemas lo realicé teniendo ya la práctica hecha, y por motivos temporales me era imposible

adaptar mi dominio para ese formato. Aún así, con mi solución, funciona a la perfección. Con respecto al Mapa, conservo el mismo que para el ejercicio anterior añadiendo las novedades de los tipos de superficie de las zonas y añadiendo objetos Bikini y Zapatillas.

4. EJERCICIO 4

APARTADO A)

Poder registrar los puntos acumulados por el agente es una tarea fácil si recurrimos a funciones, por ello, he definido:

```
(points ?c - character ?i - item)
(total_points)
```

La función (**points** character item) es la que nos permite expresar la puntuación que obtiene el agente según qué objeto objeto le entregue a cada personaje, osea que con dicha función podemos representar la tabla de puntos dada en el ejercicio. También, para ir registrando los puntos acumulados por el agente he definido la función (**total_points**), la cuál ejerce el papel de un contador, que se va incrementando conforme el agente va entregando objetos a los personajes.
Lo más razonable es que los puntos acumulados (total_points) se incrementen en la acción definida anteriormente **'give'**, que es la que representa la acción de entregar un objeto a un personaje. Tras la modificación, la acción queda así:

```
(:action give
       :parameters (?pl - player ?c - character ?i - item ?z - zone)
       :precondition (and (location ?pl ?z) (location ?c ?z)
                          (has ?pl ?i) (not (has ?pl Zapatillas)) (not (has ?pl Bikini)))
       :effect
       (and (not (has ?pl ?i)) (has ?c ?i) (handempty ?pl) (increase (total_points) (points ?c ?i)))
  )
```

Como vemos, se aprecian algunas novedades con respecto a la versión anterior. Las precondiciones siguen siendo las mismas, no se puede entregar unas Zapatillas o un Bikini a un personaje, pero la novedad se ha incluido en el efecto de la acción, ya que cada vez que se lleve a cabo la acción, se incrementa el número de puntos total acumulado, y ¿cuántos puntos se incrementa?, tantos como indique la función comentada anteriormente, por la que sabiendo el objeto y el personaje al que se le va a entregar, nos devuelve los puntos que acumulamos por ello.

APARTADO B)

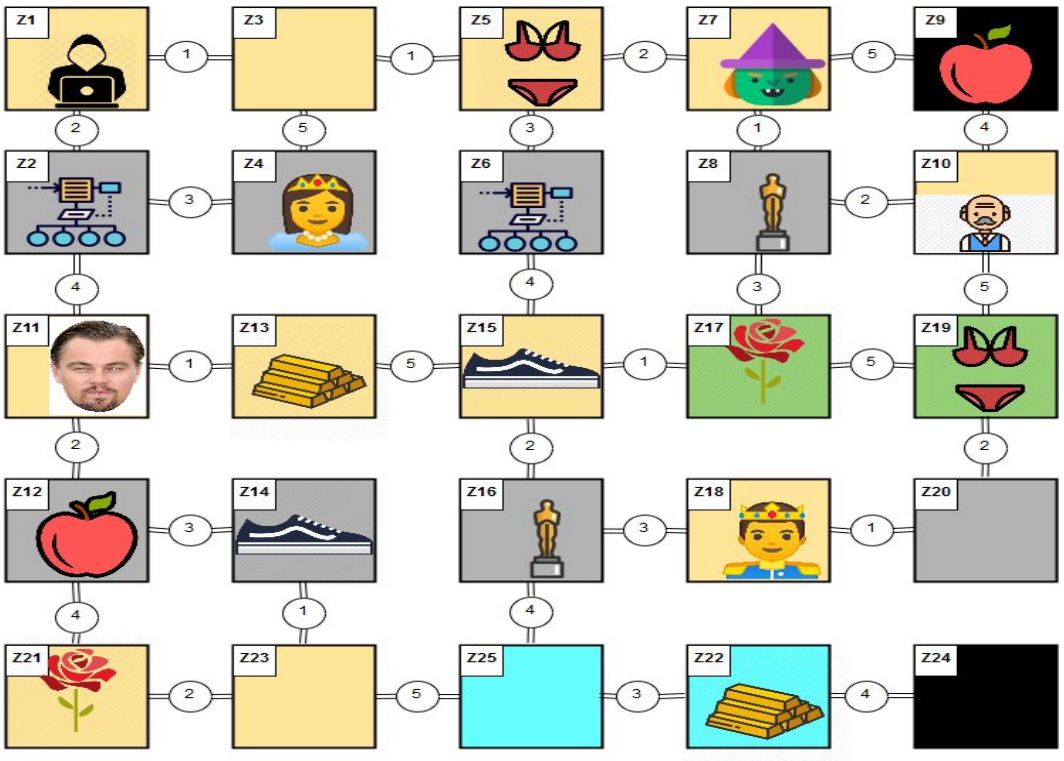
En primer lugar, para representar la tabla anterior, he recurrido a la función (**points**), ya comentada, añadiéndola el problema con el siguiente formato:

```
(= (points Leonardo Oscar) 10)
(= (points Leonardo Rosa) 1)
(= (points Leonardo Manzana) 3)
(= (points Leonardo Algoritmo) 4)
(= (points Leonardo Oro) 5)
```

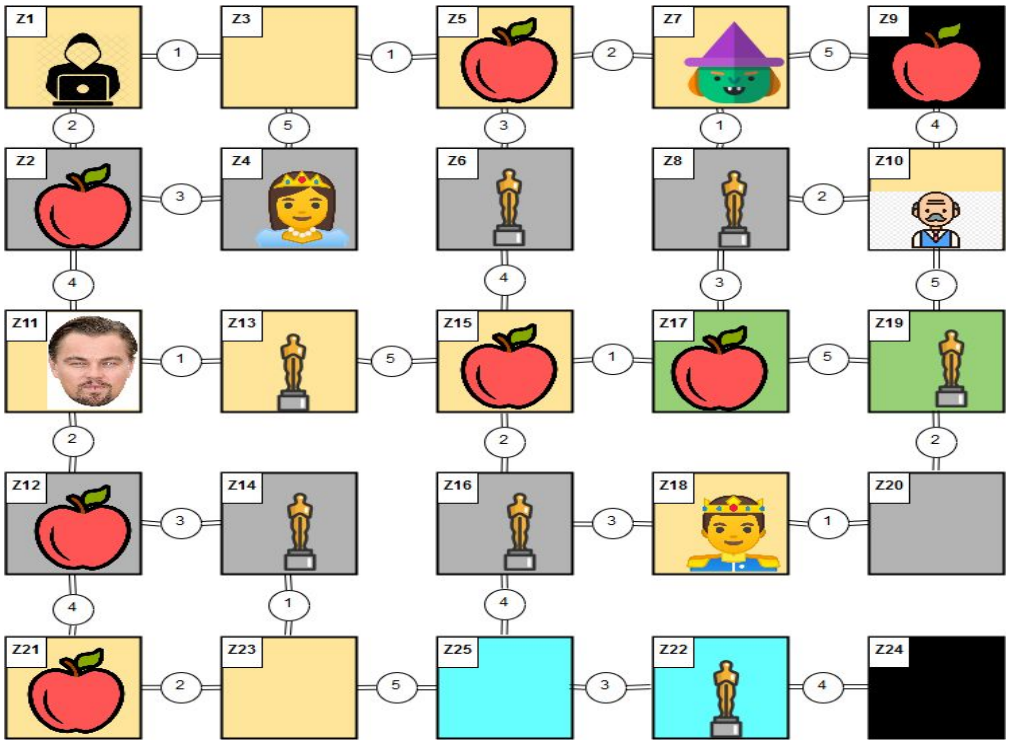
En la imagen se aprecia sólo la puntuación para el personaje Leonardo, pero se haría igual con el resto de personajes. Por otro lado, el problema debe iniciarse de partida con que el jugador tiene 0 puntos, por lo tanto, la función (**total_points**) se inicializa a 0:

```
(= (total_points) 0)
```

El primer problema que he definido para éste ejercicio es el que mantiene la distribución ya vista en ejercicios anteriores, con la particularidad de que en ésta ocasión el objetivo es exclusívamente obtener una puntuación mayor o igual a 50 puntos.



El segundo mapa que he definido es uno en el que los únicos objetos distribuidos por el mapa son de tipo oscar y manzana, lo que restringe la facilidad del problema, pues si hubiera objetos de cada tipo, sería fácil satisfacer a los personajes con los objetos que más puntuación dan, y por tanto llegar con más facilidad a la solución. Mapa:



He de decir que el objetivo es el mismo que para el ejercicio anterior:

```
(:goal
    (and (>= (total_points) 50))
)
```

APARTADO C)

Mantengo el mismo mapa e introduzco los puntos mínimos que se deben alcanzar. Ningún aspecto mayor a relevar.

5. EJERCICIO 5

APARTADO A)

Para poder adecuarme a la nueva característica de los personajes, he definido 2 nuevas funciones:

```
(magic_pocket ?c - character)
(magic_pocket_size ?c - character)
```

La primera de ellas es (**magic_pocket** character), dicha función es la encargada de contabilizar cuántos objetos tiene almacenados un personaje en su bolsillo mágico. La segunda es la función que se emplea para configurar el máximo número de objetos que puede almacenar un personaje en su bolsillo mágico. Dichas funciones tienen que ser usadas por tanto en la acción **'give'**, pues se deben activar cuando un personaje trate de recibir o reciba un objeto. Acción 'give' modificada:

```
(:action give
      :parameters (?pl - player ?c - character ?i - item ?z - zone)
      :precondition (and (location ?pl ?z) (location ?c ?z)
                         (has ?pl ?i) (not (has ?pl Zapatillas)) (not (has ?pl Bikini)) (< (magic_pocket ?c) (magic_pocket_size ?c)))
      :effect
      (and (not (has ?pl ?i)) (has ?c ?i) (handempty ?pl) (increase (total_points) (points ?c ?i)) (increase (magic_pocket ?c) 1))
 )
```

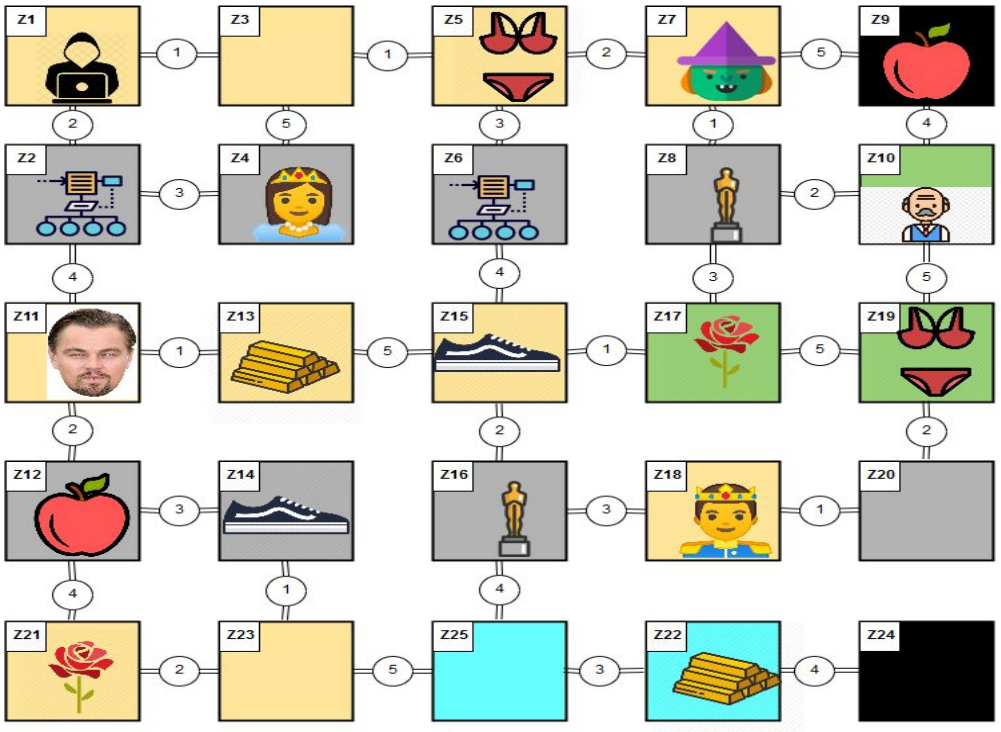
Puesto que el bolsillo mágico de cada personaje tiene un máximo de objetos a almacenar, antes de que un personaje pueda recibir un objeto, se debe comprobar la precondición de que el número de objetos almacenados actualmente por dicho personaje sea inferior al número máximo permitido. En caso de tener capacidad, uno de los efectos de la acción será incrementar el número de objetos almacenados en el bolsillo mágico de dicho personaje en 1 unidad. De esa forma conseguimos adecuarnos a la nueva característica. En el problema se debe inicializar el número de objetos almacenados en la mochila de cada personaje a 0 y establecer el máximo posible a almacenar, de la siguiente forma:

```
; Inicializo el bolsillo mágico de los personajes a 0
(= (magic_pocket Leonardo) 0)
(= (magic_pocket Princesa) 0)
(= (magic_pocket Bruja) 0)
(= (magic_pocket Profesor) 0)
(= (magic_pocket Principe) 0)

; Indico el tamaño máximo del bolsillo mágico de los personajes
(= (magic_pocket_size Leonardo) 2)
(= (magic_pocket_size Princesa) 2)
(= (magic_pocket_size Bruja) 2)
(= (magic_pocket_size Profesor) 2)
(= (magic_pocket_size Principe) 2)
```

APARTADO B)

Para éste ejercicio, mi mapa sigue siendo el mismo:



Además en éste primer problema he mantenido el objetivo de que obtenga mínimo 50 puntos y he indicado que la capacidad máxima del bolsillo mágico de los personajes sea 2, por lo que a primera vista no debe presentar dificultad alguna. En el segundo problema que he definido, el mapa sigue siendo el mismo, la máxima capacidad de los bolsillos mágicos la misma también, pero el número mínimo de puntos a conseguir pasa a ser 70, de forma que si tenemos 5 personajes, a la fuerza para conseguir resolver el problema, algunos personajes tienen que exprimir la máxima capacidad de su bolsillo mágico.

APARTADO C)

Mantengo el mismo mapa e introduzco el número de objetos admitido por cada personaje, es decir, la capacidad máxima del bolsillo mágico de los personajes. Además, para poner a prueba la nueva característica, indico que la princesa no tiene capacidad en su bolsillo mágico, por lo tanto no puede recibir el objeto.

6. EJERCICIO 6

APARTADO A)

Para considerar que hay dos jugadores cooperantes no es necesario añadir nada, pues puede haber tantos jugadores como se desee, ya que el dominio está preparado para ello. La nueva característica por la que vamos a tener que modificar el dominio es porque entre los dos tienen que obtener una cantidad de puntos dada y cada uno tiene que conseguir un mínimo de puntos. Para ello, defino la función:

```
(player_points ?pl - player)
```

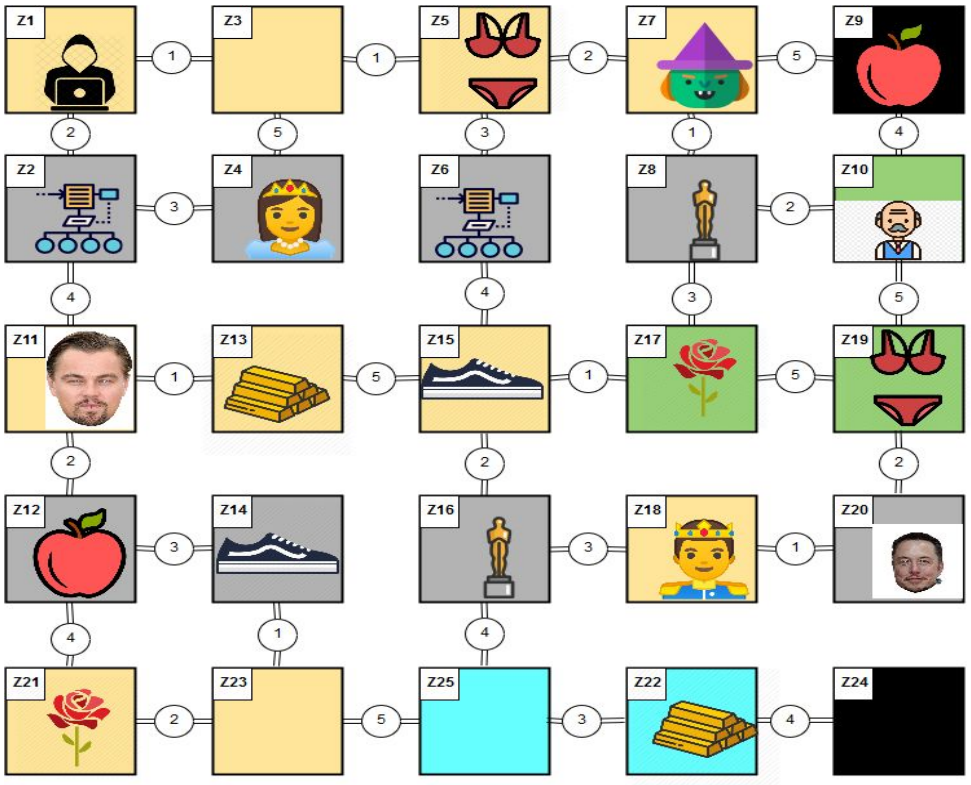
Dicha función es la encargada de llevar el recuento de puntos que lleva cada personaje conseguidos. Por ello, debemos modificar de nuevo la acción **'give'**, para ir incrementando los puntos de los jugadores conforme vayan entregando objetos. La acción queda así:

```
(:action give
       :parameters (?pl - player ?c - character ?i - item ?z - zone)
       :precondition (and (location ?pl ?z) (location ?c ?z)
                          (has ?pl ?i) (not (has ?pl Zapatillas)) (not (has ?pl Bikini)))
       :effect
       (and (not (has ?pl ?i)) (has ?c ?i) (handempty ?pl) (increase (total_points) (points ?c ?i)) (increase (magic_pocket ?c) 1)
                                                           (increase (player_points ?pl) (points ?c ?i)))
 )
```

La única diferencia con respecto a la anterior versión es que siempre y cuando un jugador, sea cual sea, entregue un objeto a un personaje, mediante la función comentada anteriormente aumentamos el número de puntos conseguidos por el jugador, tantos como nos indique la función que extrae los puntos de la tabla del ejercicio 4. De esa forma, vamos actualizando los puntos conseguidos por cada jugador.

APARTADO B)

Al problema anterior, he añadido un segundo jugador, en mi caso ElonMusk, quedando el mapa así:



También debemos modificar el objetivo del problema para mostrar que el dominio es correcto y cumple con la nueva condición, por lo tanto nuestro nuevo objetivo será:

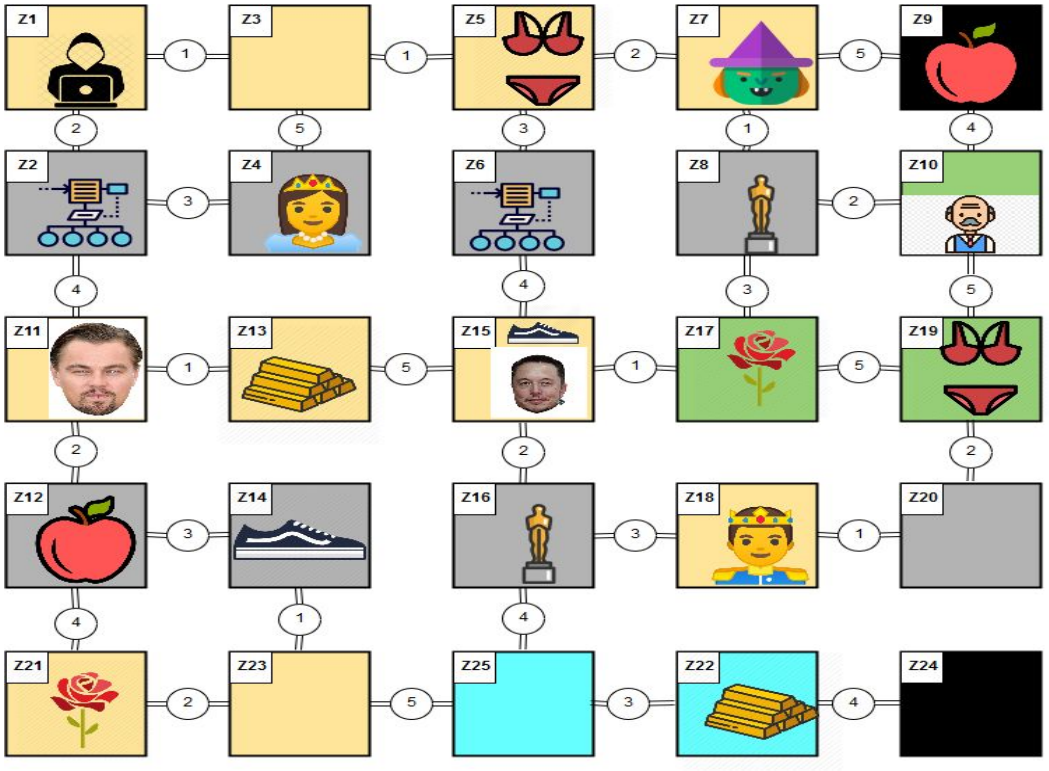
```
(:goal
    (and (>= (total_points) 70) (>= (player_points Hacker) 20) (>= (player_points ElonMusk) 30))
)
```

Es decir, para resolver el problema, entre los dos jugadores tienen que conseguir mínimo 70 puntos, pero además el jugador Hacker debe conseguir un mínimo de 20 puntos mientras que el jugador ElonMusk debe conseguir mínimo 30.

Para éste ejercicio, he definido un segundo problema, similar al primero pero en el que he incrementado el número de puntos a conseguir por el jugador ElonMusk, además, lo he cambiado de posición para comprobar si le cuesta más o menos. Nuevo objetivo:

```
(:goal
    (and (>= (total_points) 70) (>= (player_points Hacker) 20) (>= (player_points ElonMusk) 35))
)
```

Mapa:



APARTADO C)

De nuevo, conservo al mapa anterior. La capacidad del bolsillo mágico de la princesa sigue siendo 0. Con respecto a las novedades, se añade 1 jugador más y se indica cuántos puntos como mínimo debe conseguir cada uno.

7. EJERCICIO 7

APARTADO A)

Adecuarse a la nueva característica puede ser resuelto de forma tan sencilla como añadiendo el siguiente predicado:

```
(type_player ?pl - player ?t - type)
```

Con éste predicado podemos tener conocimiento con respecto al tipo de jugador que es cada uno, en nuestro caso Picker o Dealer. De nuevo, las acciones deben modificarse, es por ello que la acción 'give' ha sido separada en 2, según la acción que se quiera realizar sea dar un objeto a un personaje o a un jugador. Acciones:

```
(:action give-player
       :parameters (?pl1 - player ?pl2 - player ?i - item ?z - zone)
       :precondition (and (location ?pl1 ?z) (location ?pl2 ?z)
                          (has ?pl1 ?i) (type_player ?pl1 Picker) (type_player ?pl2 Dealer) (not (has ?pl1 Zapatillas)) (not (has ?pl1 Bikini)))
       :effect
       (and (not (has ?pl1 ?i)) (has ?pl2 ?i) (handempty ?pl1))
  )
```

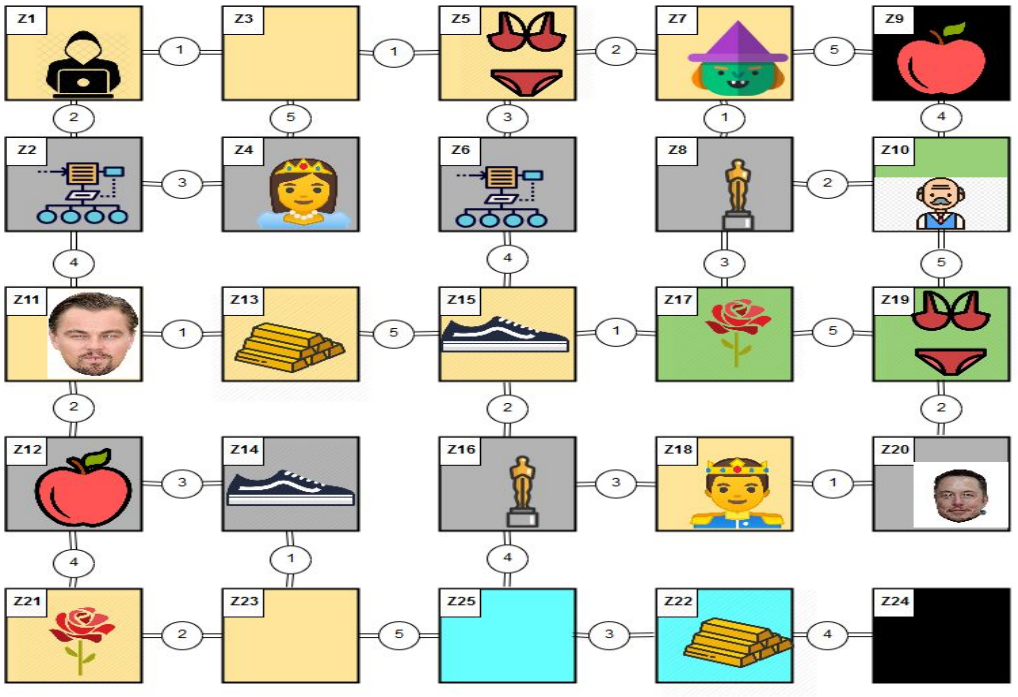
Ésta es la primera de ellas, la acción de entregar un objeto a un jugador, por lo tanto es la acción que se activará cuando el jugador Picker quiera entregar un objeto al jugador Dealer. Recibe 2 jugadores como parámetros obviamente, para realizar el intercambio, un objeto y una zona. Para que la acción se active, ambos jugadores deben estar en la misma localización, además el jugador que posea el objeto debe ser de tipo Picker mientras que el otro debe ser Dealer. El resto de precondiciones se mantienen de la versión anterior. Los efectos de dicha acción sería que el jugador Picker deja de tener el objeto y pasa a tenerlo el jugador Dealer, además de pasar a tener las manos vacías el jugador Picker.

```
(:action give-character
       :parameters (?pl2 - player ?c - character ?i - item ?z - zone)
       :precondition (and (location ?pl2 ?z) (location ?c ?z)
                          (has ?pl2 ?i) (type_player ?pl2 Dealer) (not (has ?pl2 Zapatillas)) (not (has ?pl2 Bikini)))
       :effect
       (and (not (has ?pl2 ?i)) (has ?c ?i) (handempty ?pl2) (increase (total_points) (points ?c ?i)) (increase (magic_pocket ?c) 1)
            (increase (player_points ?pl2) (points ?c ?i)))
  )
```

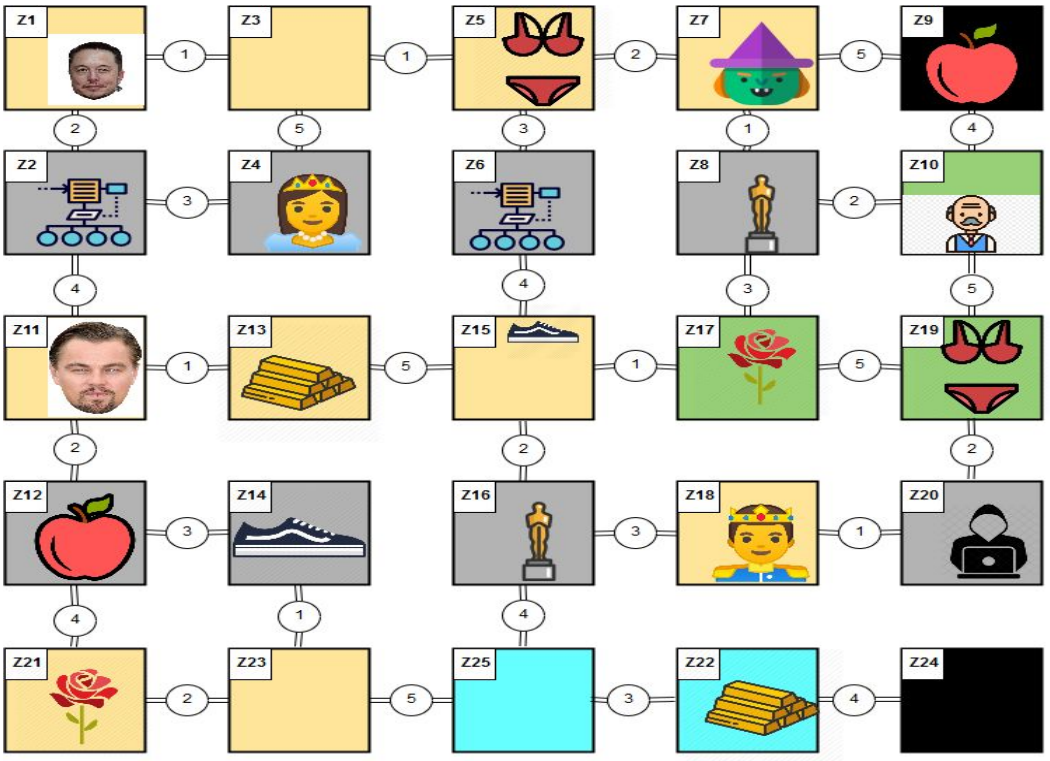
Ésta acción por tanto, es lleva a cabo cuando el jugador de tipo Dealer quiere entregar un objeto a un personaje. Las nueva precondición con respecto a la acción ‘give’ del ejercicio anterior sería que el jugador debe ser tipo Dealer. Esa sería la única novedad en la acción y hasta ahí todo lo que habría que añadir para cumplir con la nueva característica.

APARTADO B)

El primer problema usado para éste ejercicio considera el mismo mapa que el problema 1 del ejercicio anterior. La única diferencia es que he reducido el número mínimo de puntos a realizar a 50, para evitar un gasto innecesario de cómputo puesto que son 50 es suficiente para comprobar que cumple con la nueva característica. Mapa:



El segundo problema que he definido es una leve modificación del primero hecha a concia para demostrar la importancia de la posición de los jugadores. Es por ello que para éste nuevo problema, he intercambiado la posición de ambos, quedando el mapa así:



APARTADO C)

De nuevo, a fin de simplificar la corrección, he reutilizado el mismo mapa, el cuál considero bastante sencillo, simplemente añadiendo la nueva característica de éste ejercicio y es que puedan ser admitidos jugadores tanto de tipo Picker como Dealer.