Práctica 2 de TSI
Planificación Clásica

UNIVERSIDAD DE GRANADA

Antonio Manuel Fresneda Rodríguez

antoniomfr@correo.ugr.es

Martes, 22 de mayo de 2018

Índice

1 Ejercicio1

1.1 Ejercicio 1a

Para representar los objetos en el mundo, he declarado los siguientes objetos:

- Player: El robot encargado de entregar los objetos
- Obj: Objetos en el mundo.
- Character: Personajes que aparecen en el mundo.
- Place: Los distintos lugares que definen el mundo.
- Compass: Brújula.

1.2 Ejercicio 1b

Los predicados que he definido han sido los siguientes:

- *(PlayerLoc ?plyr - Player ?plc - Place)*: Tiene como entrada un objeto Player y uno Place. Especifica la posición del robot en un instante de tiempo.
- *(ObjectLoc ?obj - Obj ?plc - Place)*: Tiene como entrada un objeto Obj y uno Place. Especifica la posición de los objetos en el mundo.
- *(CharacterLoc ?chrctr - Character ?plc - Place)*: Tiene como entrada un objeto Character y otro Place. Especifica la posición de los personajes en el mundo,
- *(Orientation ?comps - Compass)*: Tiene como entrada un objeto Compass. Especifica la orientación del robot en un instante de tiempo.
- *(NeighborPlace ?place1 - Place ?place2 - Place ?orientation - Compass)*: Tiene como entrada dos objetos de tipo Place y otro de tipo Compass. Indica que dos ciudades están conectadas en una dirección indicada por la brújula
- *(HasObject ?obj - Obj)*: Tiene como entrada un objeto de tipo Obj. Especifica que el robot tiene el objeto en la mano.
- *(DeliveredObj ?charctr - Character ?obj - Obj)*:Tiene como entrada un objeto de tipo Character y otro tipo Obj. Especifica que un personaje ya ha recibido su objeto.
- *(HandEmpty)*: Especifica si la mano del robot está vacía.

1.3 Ejercicio 1c

La explicación de las acciones son las siguientes:

Girar

Girar izquierda y girar derecha son prácticamente iguales, lo único que cambia es el efecto. Por ejemplo si el robot mira al norte y gira a derecha pasa a estar en en dirección este mientras que si gira a izquierda estaría en dirección oeste

- *Parametros*: Orientación del robot (necesito saber la orientación actual ya que al girar voy a cambiarla dependiendo de la orientación actual).
- *Precondiciones*: Que la orientación actual del robot y la que se ha pasado por parámetro sea la misma.
- $Efecto$: Se cambia la orientación (si está mirando hacia el norte ahora pasa a estar mirando al oeste y a no estar mirando al norte)

Moverse

- *Parametros*:Nombre del robot, lugar origen, lugar destino y orientación
- *Precondiciones*: Que el robot este en el lugar de origen, que origen y destino sea distinto (o que el robot no este a la vez en origen y destino), que las dos ciudades tengan un camino entre ellas y que la orientación del robot sea igual a la orientación en la que están conectadas las ciudades.
- $Efecto$: El robot pasa de estar en la ciudad origen a la ciudad destino.

Coger

- *Parametros*:Nombre del robot, lugar y un objeto.
- *Precondiciones*: Que el robot y el objeto estén en el mismo lugar y que la mano del robot esté vacía
- $Efecto$: El objeto no esta en el lugar, la mano del robot pasa a no estar vacía y el robot tiene el objeto.

Tirar

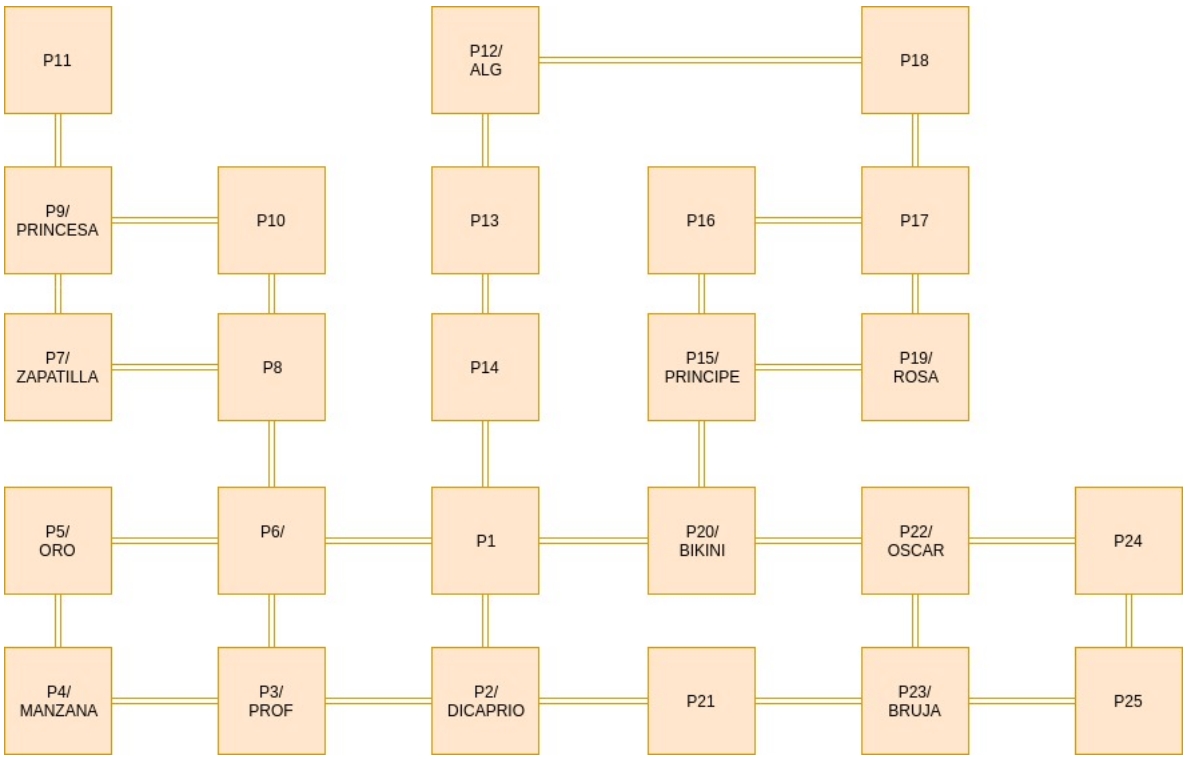
- *Parametros*:Nombre del robot, lugar y un objeto.
- *Precondiciones*: Que el robot este en ese lugar y que tenga ese objeto
- $Efecto$: El objeto esta en el lugar, la mano del robot pasa a estar vacía y el robot no tiene el objeto.

Entregar

- *Parametros*:Nombre del robot, lugar, un objeto y un personaje.
- *Precondiciones*: Que el robot y el personaje estén en ese lugar,que el robot tenga ese objeto y no se haya entregado.
- *Efecto*: El objeto es entregado al personaje, el robot ya no tiene ese objeto y la mano esta vacía.

1.4 Ejercicio 1d

He planteado el siguiente mapa para probar que el dominio funciona:



El plan esta en el archivo planEJ1.txt

2 Ejercicio 2

2.1 Ejercicio 2a

Para el ejercicio 2 he modificado el dominio anterior añadiendo dos funciones. Una que contabiliza el coste y que esta inicializada a 0 y cada vez que se pasa de una ciudad a otra se incrementa según el coste (*(increase (Cost) (Distance ?p1 ?p2))*) y la segunda que tiene dos parámetros P1 y P2 y que me devuelve la distancia entre los dos puntos. Para ello la forma en la que he declarado la distancia entre dos ciudades ha sido: *(=Distance(P1 P2) 1)*y así con las 25 ciudades. También se ha añadido al objetivo que minimice la distancia recorrida (*(:metric minimize (Cost))*).

2.2 Ejercicio 2b

He considerado que todas las zonas están a la misma distancia (5 metros). En el caso contrario tendríamos que cambiar los arcos que van tanto de A a B como de B a A por el valor correspondiente. Los pesos de la heurística usados han sido para g(n) igual a 1 y para h(n) igual a 2. He intentado ejecutar el problema con h(n) igual a 1 pero no terminaba de realizar el plan.
El plan obtenido esta en el fichero planEJ2.txt

3 Ejercicio 3

3.1 Ejercicio 3a

Para indicar el tipo de suelo que hay he declarado un nuevo objeto llamado Ground. Para especificar que de que tipo de suelo es una zona, he declarado un nuevo predicado: *(PlaceType ?place - Place ?type - Ground)*, para indicar el objeto necesario para pasar por un tipo de suelo he declarado el predicado: *(GroundObject ?type - Ground ?obj - Obj)*. Hay un problema con esto, y es que las zonas en las que no necesitas objeto la única forma de representarlo seria añadiendo un objeto o declarando un nuevo predicado que indicase que tipo de zonas no necesitan objeto para pasar por ellas. Yo había implementado la segunda de las opciones declarando el predicado: *(NoNeeddObject ?type - Ground)* pero no me funcionaba correctamente. Cuando se movía a zonas donde no necesitaba objeto siempre aparecía como si tuviese un zapato (aunque no lo tuviera). Creo que es porque como la acción de moverse necesita un parámetro de tipo objecto, coge el ultimo objeto declarado así que añadí el objeto NONE y este es el que utiliza cuando no necesita un objeto para moverse a una zona que no necesita objeto para pasar. La otra opcion era declarar otra acción para moverse y que no tuviera el parámetro objecto.
Para representar los precipicios no he añadido el predicado de NoNeedObject para precipicios, así consigo que el planificador no lo tenga en cuenta.

3.2 Ejercicio 3b

Para la representación de la mochila he añadido los siguientes predicados:

- *(SavedObject ?obj - Obj)*: Indica que el objeto obj esta guardado en la mochila.
- *(BagEmpty)*: Indica si la mochila está vacía.

También he añadido dos acciones más: **Sacar**

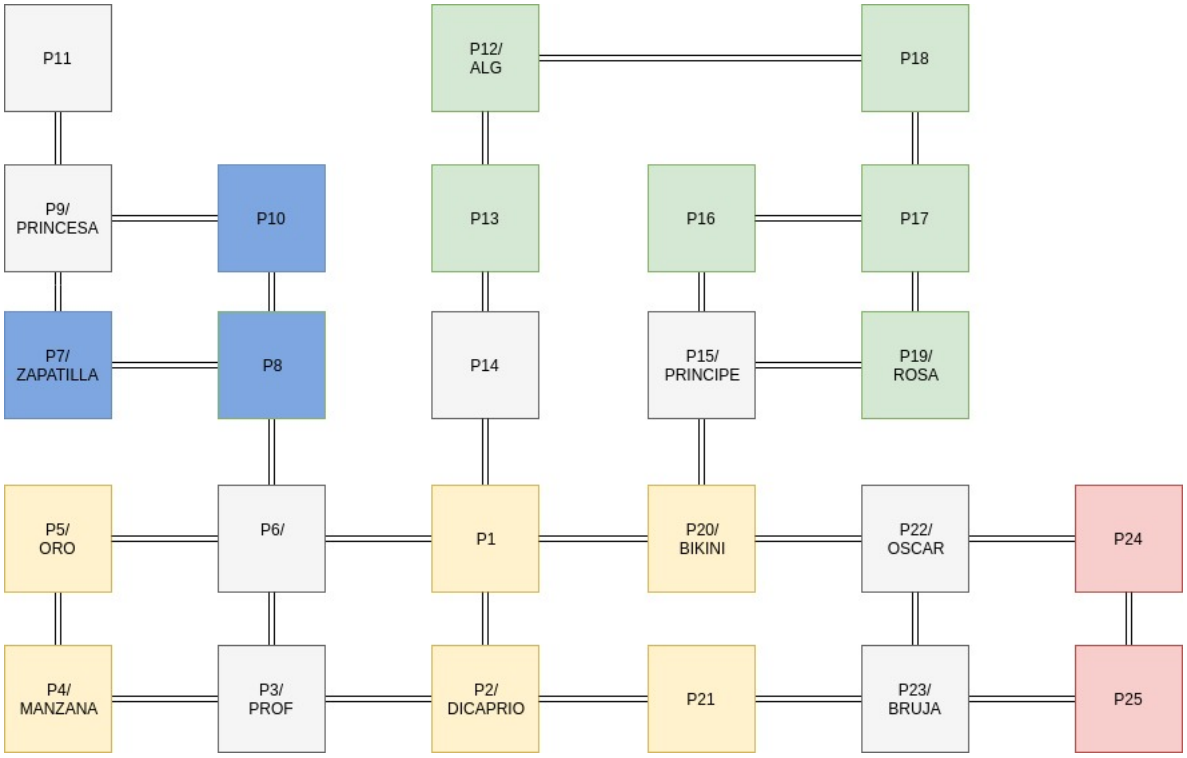
- *Parámetros*: Un objeto
- *Precondiciones*: Que el objeto esté en la mochila y que la mano esté vacía
- *Efectos*: El robot tiene el objeto en la mano, la mano no esta vacia, la mochila está vacía y no tiene guardado el objeto.

Meter

- *Parámetros*: Un objeto
- *Precondiciones*: Que la mochila esté vacía y que tenga en la mano el objeto.

- *Efectos*: El robot no tiene el objeto en la mano, la mano esta vacía, la mochila no está vacía y en la mochila esta el objeto guardado.

El mapa que se ha usado para probar el dominio ha sido el siguiente:



Donde las zonas azules representan agua, la verde bosque, la roja precipicio, la amarilla arena y la gris piedra. El plan encontrando está en el archivo PlanEJ3.txt. Cabe destacar que al principio parece

4 Ejercicio4

4.1 Ejercicio 4a

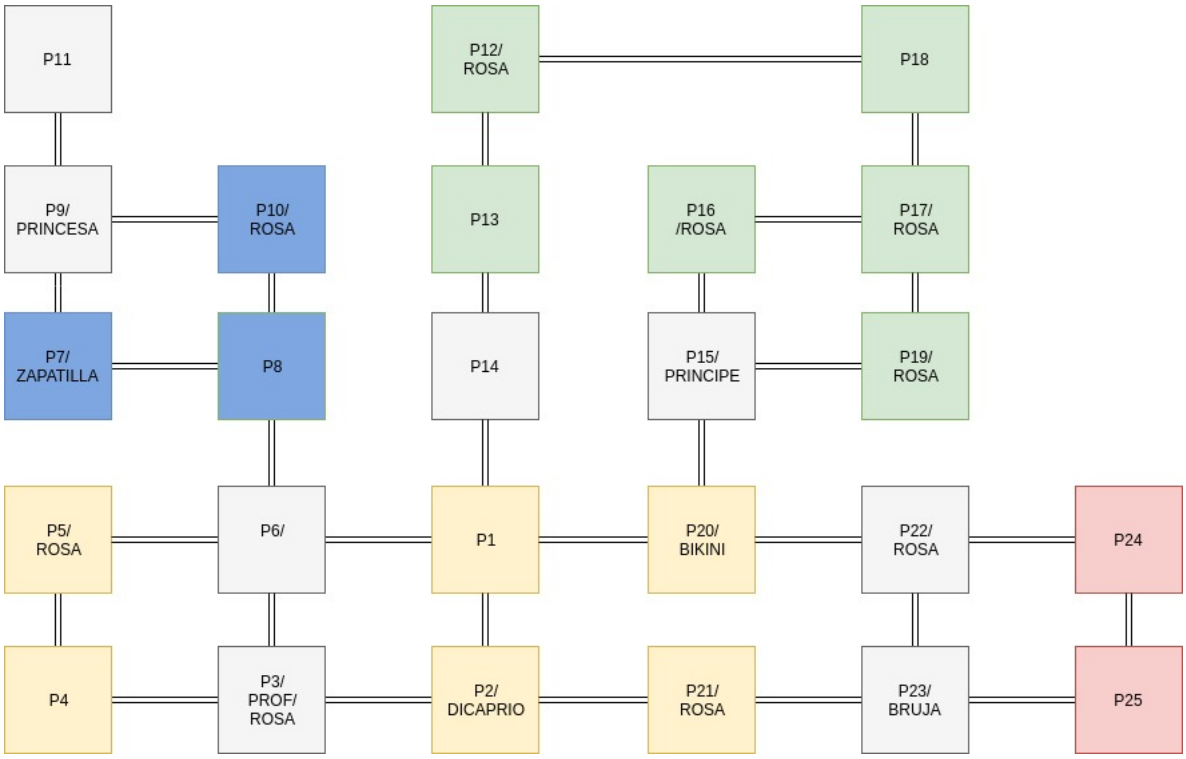
Para realizar el contabilizado de puntos he hecho algo parecido que lo que hice con la distancia. Me he declarado dos funciones: *(Points)*: que se incrementa cada vez que se entrega un objeto y *(Puntuation ?chrctr - Character ?obj - Obj)*: que devuelve el valor de puntuación que se obtiene al entregar un objeto. Para representar la puntuación voy a poner un ejemplo de como lo he hecho: *(= (Puntuation DICAPRIO OSCAR) 10)*, y esto es lo que devolvería la funcion *Puntuation*.

4.2 Ejercicio 4b

Para esto he realizado varios cambios:

1. He eliminado la restricción de que para entregar un objeto este no haya sido ya entregado previamente ya que si queremos llagar a 50 puntos si solo ponemos entregarle varios objetos a un personaje.
2. He cambiado el objetivo. En vez de que se deba de entregar un objeto a cada personaje lo que hacemos es que el numero total de puntos sea mayor o igual a 50.
3. He eliminado todos los objetos exceptuando un objeto de tipo Oscar y he colocado varios objetos manzana (hasta poder llegar a 50 puntos).

El problema planteado ha sido el siguiente:



El plan obtenido esta en el archivo planEJ4.txt

5 Ejercicio 5

5.1 Ejercicio 5a

Para realizar este ejercicio he hecho lo siguiente:

1. He declarado dos nuevas funciones: *NumberOfObjects* que especifica el tamaño del bolsillo de cada personaje y *(ObjectsDelivered ?chrctr - Character)* que especifica la cantidad de objetos que se han dado a un personaje. Un ejemplo de como se inician sería: *(= (ObjectsDelivered PRINCESS) 0)*.

2. La acción Entregar se ha cambiado:

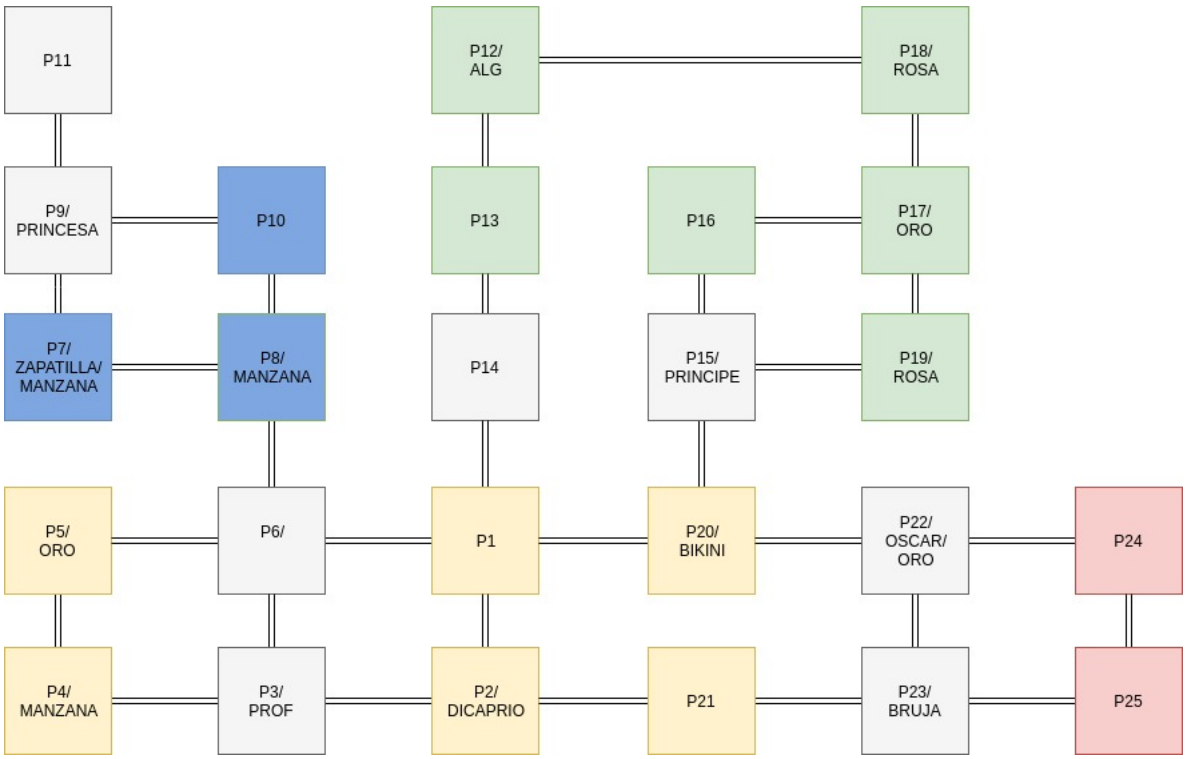
   - *Parametros*:Nombre del robot, lugar, un objeto y un personaje.
   - *Precondiciones*: Que el robot y el personaje estén en ese lugar,que el robot tenga ese objeto y no se hayan entregado más

objetos que el límite del bolsillo.

- *Efecto*: El objeto es entregado al personaje, se incrementa el contador de objetos entregados a ese personaje, el robot ya no tiene ese objeto y la mano esta vacía.

5.2 Ejercicio 5b

Para probar el dominio he planteado un ejemplo en el que hay bastantes objetos. Esto se debe a que si ponemos pocos objetos puede ocurrir que el plan no termine. Así conseguimos que el plan se ejecute bastante rápido. El mapa queda asi:



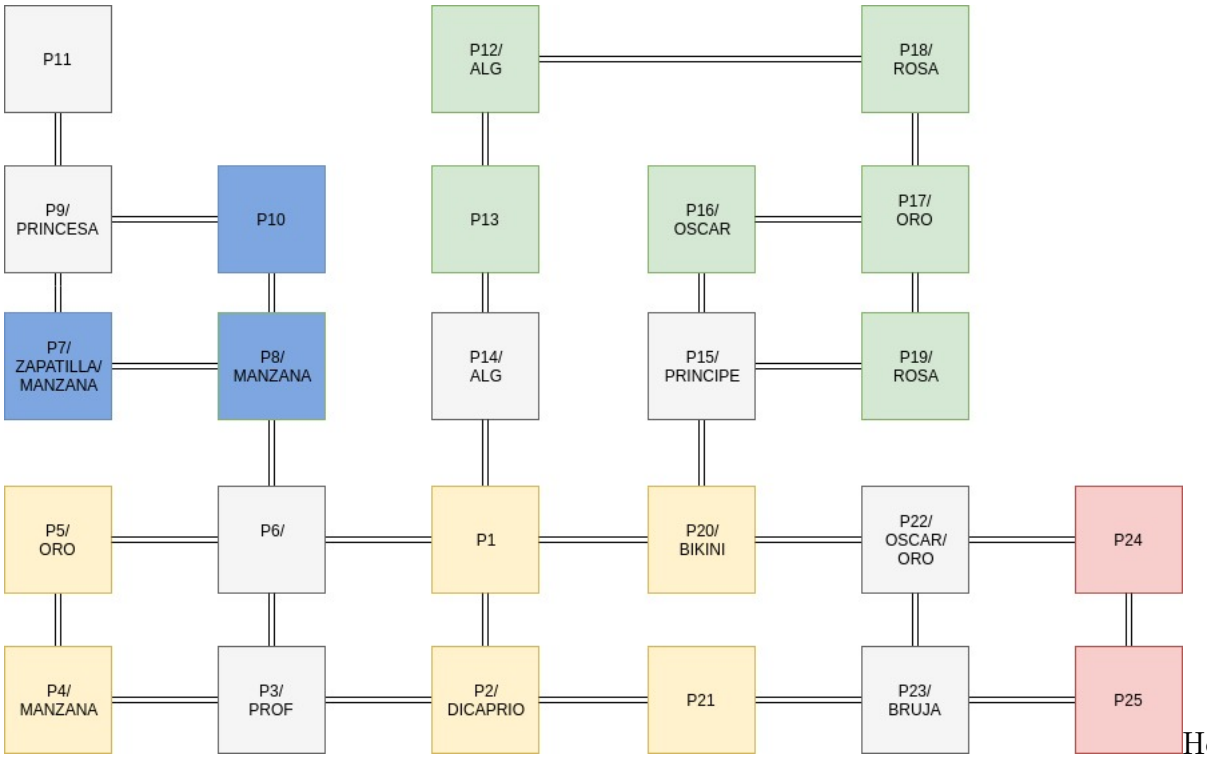
Y el plan esta en el archivo PlanEJ5.txt

6 Ejercicio 6

Para el este ejercicio lo que he hecho ha sido en todos los predicados que tienen que ver con el robot (*HandEmpty, BagEmpty, Location, HandObject, Orientation, SavedObject*) les he añadido una variable de tipo Player para indicar el jugador al que pertenece cada predicado. Aparte he declarado las siguiente función

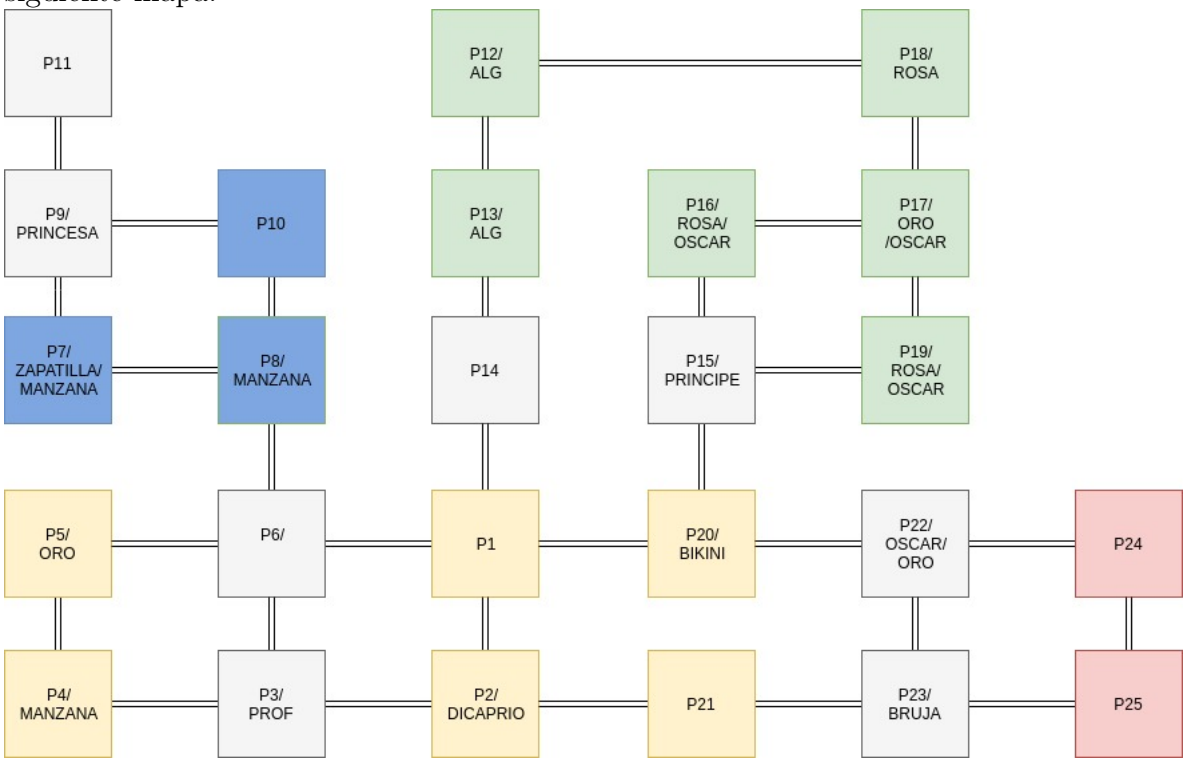
- *(PlayerPoints ?plyr - Player)*: Que indica los puntos que tiene el jugador en un momento de tiempo.

En el problema he añadido otro Robot y los puntos los he iniciado para ambos a 0. Una vez hecho esto, en el objetivo he añadido que cada jugador tenga un mínimo de puntos asi: *(and (>= (Points) 20) (>= (PlayerPoints PLAYER1) 5) (>= (PlayerPoints PLAYER1) 5)))* en el siguiente mapa:



He

reducido el numero de puntos para que tarde menos. El resultado de este plan está en el PlanEJ6-1.txt Tambien he probado con el siguiente mapa:



El objetivo para este problema es: *(and (>= (Points) 30) (>= (PlayerPoints PLAYER1) 2) (>= (PlayerPoints PLAYER1) 5))*. El plan está en PlanEj6-2.txt El jugador 1 parte de la posición P1 y el jugador 2 de la posición P2

7 Ejercicio 7

El principal problema de este ejercicio es que si solo modificamos la acción *PICK-UP* para que solo pueda coger objetos un jugador, el plan no termina ya que si tenemos que llevar un objeto a un personaje que este en sitios donde necesitan algun tipo de objeto, el robot que entrega los objetos no va a llegar. Para ello he hecho lo siguiente:

- *(Picker ?plyr - Player)*: Este predicado indica que robot es el que recoge.
- *(SpecialObject ?obj)*: Indica que son objetos que pueden coger ambos personajes (zapatillas y bikini).
- *PICK-UP*: En esta acción he añadido la precondición de que solo pueda coger un objeto si es el "picker" a no ser que sea un objeto especial.

El dominio de este problema lo he probado con el mapa del ejercicio 6-1 y con el mapa del ejercicio 6-2. La única diferencia es que donde hay una zapatilla y un bikini he puesto dos (uno para cada robot para que el planificador encuentre antes un plan si no tarda mucho tiempo). El resultado obtenido está en los ficheros PlanEJ7-1.txt y PlanEJ7-2.txt