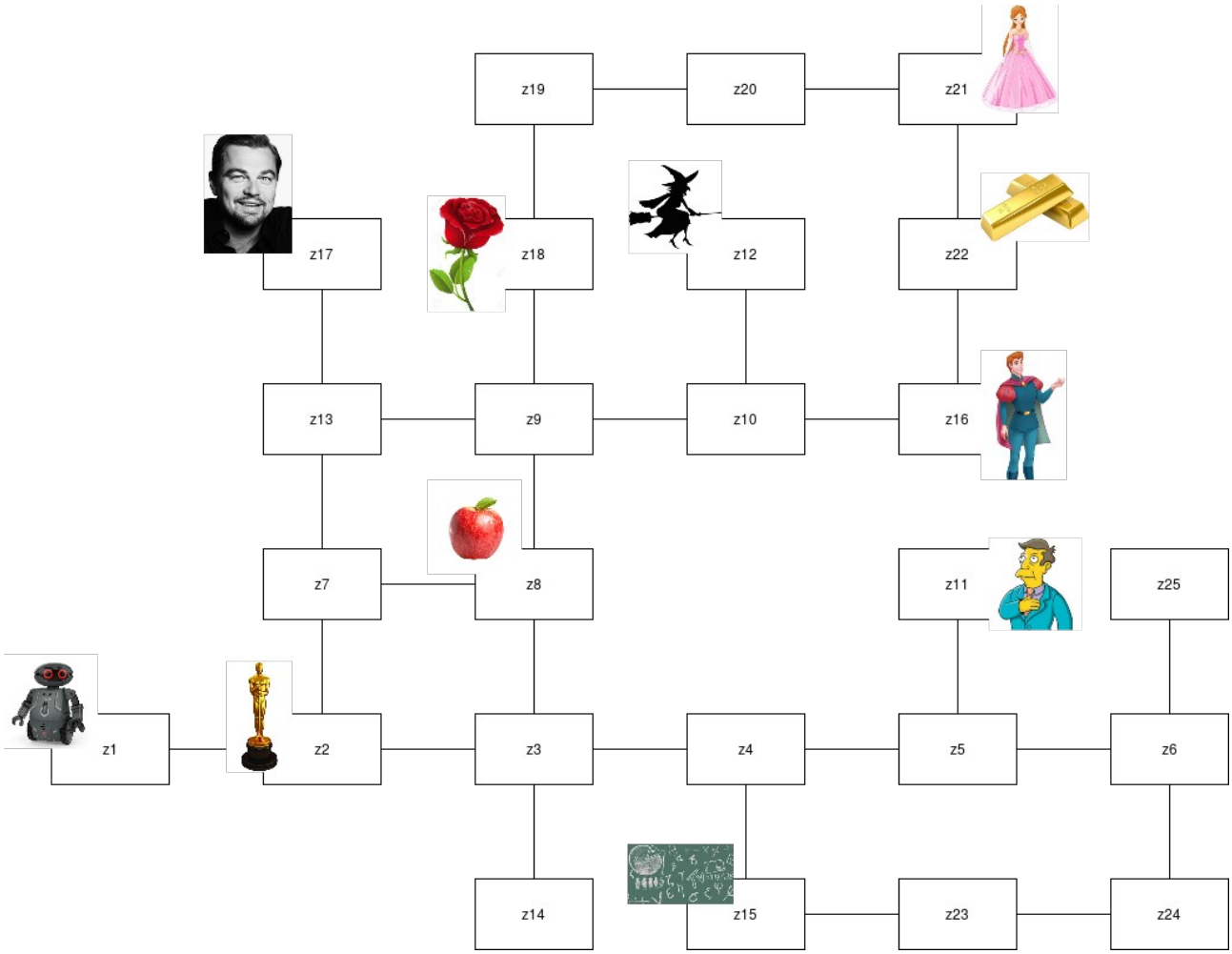
Técnicas de los Sistemas Inteligentes
Curso 2018-19

Práctica 2: Planificación Clásica (PDDL)

Alberto Estepa Fernández



Ejercicio 1.a: Para representar en el dominio los objetos del mundo de la forma mas general posibles hemos hecho una subdivisión en tipos de forma que si hay varios objetos de un subtipo de objetos, como por ejemplo varios Oscar, podamos identificarlos como Oscar que a la vez será un subtipo de Objetos. Así la división de tipos quedaría: Player (después se ampliará para distinguir entre Dealer y Picker en el ejercicio 6), Personaje (DiCaprio, Princesa, Principe, Profesor, Bruja), Objeto (Oscar, Oro, Algoritmo, Manzana, Rosa), Zona (después se ampliará en el ejercicio 3) y Orientacion

Ejercicio 1.b: Para expresar todas las acciones posibles que puede realizar el Jugador hemos creado una serie de predicados: PosicionJugador <Player> <Zona> que será verdad si el jugador esta en dicha zona, PosicionObjeto <Objeto> <Zona> que será verdad si el objeto se encuentra en dicha zona, PosicionPersonaje <Personaje> <Zona> que será verdad si el personaje se localiza en la zona correspondiente. OrientacionJugador <Orientacion> que será verdad si el jugador se orienta con la orientación dada (se implementó sin indicar el jugador en un principio puesto que hasta el ejercicio 6 no se incluyen varios jugadores en el problema, hasta entonces no se modificará), Conexion <Zona1> <Zona2> <Orientacion> que será verdad si la zona1 esta conectada con la zona2 en la orientación dada (Norte, Sur, Este, Oeste) de la zona1, ObjetoCogido <Objeto> que será verdad si el jugador tiene un objeto cogido de tipo objeto y por tanto tendrá la mano ocupada (al igual que OrientacionJugador, está pensada para un solo jugador, cosa que se modificará en el ejercicio 6 al introducir nuevos jugadores), TieneObjeto <Personaje> que será verdad si el personaje dado tiene algún objeto de los posibles en su poder (será el objetivo del problema), y por último ManoLibre que indicará si el jugador no dispone de ningún objeto en su mano (por una parte, al igual que los anteriores, se modificará en el ejercicio 6 al introducir nuevos jugadores y por otra parte, aunque es el opuesto de ObjetoCogido, en algunos momentos nos interesará saber que Objeto tiene cogido el Jugador (para lo que usaremos ObjetoCogido) mientras que en otros momentos nos interesará saber si el Jugador tiene algo cogido, independientemente de qué (para lo que usaremos ManoLibre)).

Ejercicio 1.c: Los predicados anteriores han sido creados en función de las acciones a realizar, así pues, como se han explicado en el apartado anterior y son funciones sencillas, entiendo que la explicación de como se ha implementado cada action es el propio código que se encuentra en el fichero Ej1dominio.ppdl

Ejercicio 1.d: Creamos el fichero problema pedido (de 25 zonas y a partir del cual vamos a generar los siguientes ficheros de los siguientes ejercicios) que será el problema1-1. Hemos creado un objeto de cada tipo posible y hemos iniciado los predicados de inicialización. El objetivo consiste en comprobar que todos los jugadores tengan algún objeto. Se entregará una visualización del mapa final en el ejercicio 7.

Ahora para analizar si podemos crear varios Objetos de un determinado tipo y varios Personajes de otro determinado tipo hemos creado el problema1-2. En este caso hemos creado 4 objetos tipo Oscar (oscar1, oscar2, oscar3 y oscar4) y hemos creado también 3 personajes de tipo Princesa (princesa1, princesa2 y princesa3).

Las salidas son coherentes, es verdad que no las mas cortas (pues no se ha ejecutado con la opción -O), pero sí son soluciones del problema.

En la portada se encuentra el mapa base sobre el problema que trabajaremos (se versionará también para probar cosas)

Ejercicio 1.f: Generador de problemas. No hecho

Ejercicio 2.a: Lo único que hemos modificado para introducir la característica pedida de introducir costes y distancias entre zonas es introducir dos funciones CosteDesplazamiento (que contabilizará el coste total) y DistanciaZonas <Zona1> <Zona2> (que indica la distancia entre dos zonas). DistanciaZonas se inicializará en el problema y CosteDesplazamiento se incrementará cada vez que se ejecuta la acción de ir de la Zona1 a la Zona2.

Ejercicio 2.b: Hemos añadido al init la inicialización de las funciones DistanciaZonas, así como la inicialización de la función CosteDesplazamiento (que siempre empezará por cero). En el problema1 hemos añadido la línea de la métrica al final del problema para que se ejecute con la opción -O y se pueda ver efectiva la optimización. En los problemas 2 y 3 sin embargo se ha eliminado esta línea para introducirlo en el objetivo. La diferencia entre 2 y 3 es que en el primero se pide que el CosteDesplazamiento sea menor o igual que 45 y en el 3 que sea menor o igual que 44. La diferencia en el resultado es abismal pues el primero se ejecuta en 94 pasos pero tarda 0.04 y el otro se ejecuta en 68 pasos pero tarda en la ejecución 0.17 segundos. Aún así cabe destacar que no es la opción más eficiente pues si con esas distancias lo ejecutaríamos con la opción -O y poniendo que la métrica sea CosteDesplazamiento (problema 4) obtenemos un resultado en 78 pasos pero con un tiempo de ejecución de 0.14 y un CosteDesplazamiento de 33 (coste optimo).

Ejercicio 3.a: Hemos añadido a los tipos TipoTerreno. Definirá si una zona es Bosque, Agua, Piedra, Arena o Precipicio. La decisión de crear un nuevo tipo que sea TipoTerreno ha sido para probar todas las posibilidades de PDDL, ya que bien pudiéramos haberlo metido como subtipos en Zona, hemos decidido hacerlo así.

Para cumplir las restricciones pedidas hemos decidido crear una serie de nuevos predicados: MochilaLibre, que será verdad si la mochila del jugador no tendrá ningún objeto dentro (análogo a ManoLibre); ObjetoEnLaMochila <Objeto>, predicado contrario a MochilaLibre pero que permite indicar qué objeto contiene la Mochila (análogo a ObjetoCogido); TipoZona <Zona> <TipoTerreno> que será verdad si la zona dada es de un tipo de los que hemos mencionado anteriormente; ObjetoTerreno <TipoTerreno> <Objeto> que será verdad si es necesarío tener un objeto característico dado para el tipo de terreno dado; TerrenoSinEquipamiento <TipoTerreno> que será cierto si el tipo de terreno dado no necesita ningun objeto específico para cruzarse; esZapatilla <Objeto> que será cierto si el objeto dado es del tipo Zapatillas y esBikini <Objeto> que será cierto si el objeto dado es del tipo Bikini.

Por último, tendremos que modificar las acciones pertinentes:
La acción Entregar ha sufrido una ligera modificación ya que tendremos que poner una precondición ya que no podremos entregar a un personaje ni bikinis ni zapatillas. Por otra parte la acción Ir sufre grandes modificaciones ya que tendremos que comprobar que el terreno no necesita objetos para pasar o que si no es así, disponemos de los objetos necesarios para pasar (en la mochila o cogido). Notamos que como el tipo Precipicio no entra en ninguno de estas opciones, no podrá nunca acceder a zonas con este tipo de terreno.

Ejercicio 3.b: Creamos las dos acciones pedidas: MeterEnMochila (necesitaremos tener la mochila libre y un objeto cogido, y el resultado será que habrá un objeto en la mochila por lo que la mochila ya no estará libre y tendremos la mano libre por lo que no tendremos ningún objeto cogido) y SacarDeMochila que será la acción contraria a MeterEnMochila.

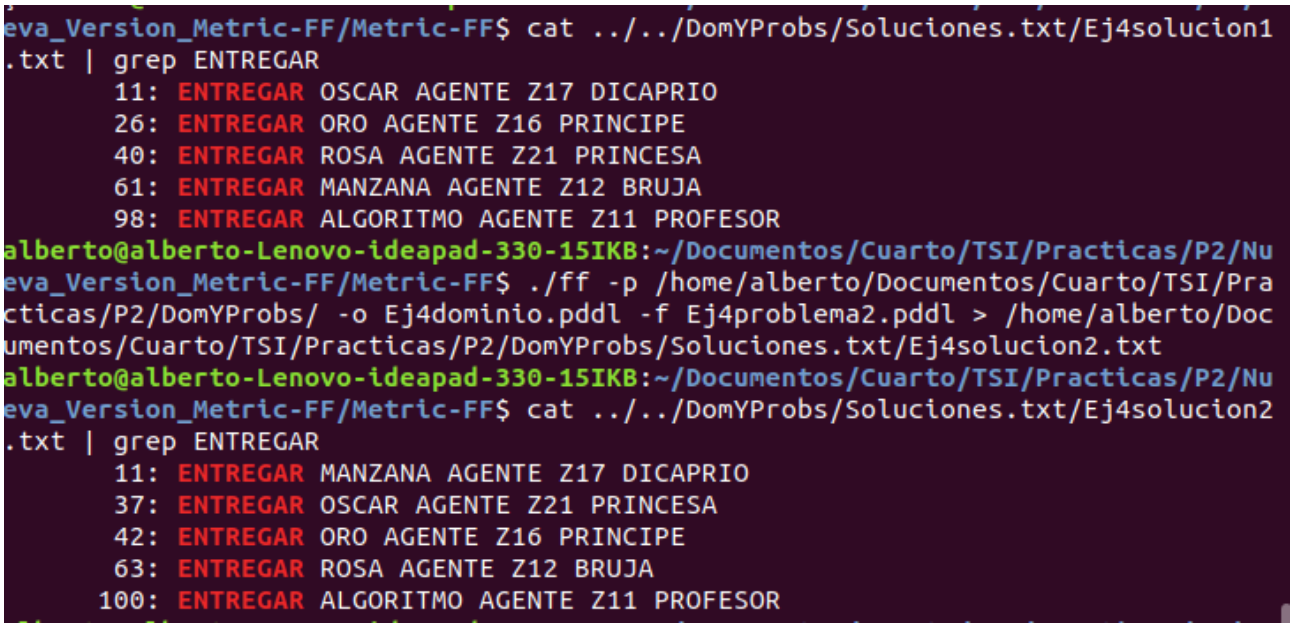
Ejercicio 3.c: Hemos creado tanto la mochila como los tipos de terreno pertinentes, le hemos definido el tipo de zona a cada zona y demás predicados necesarios (todo está en el problema 1 del ejercicio 3). Hemos añadido las únicas zapatillas en una zona de agua (z4) como aconsejaba el enunciado, además es necesario pasar por el bosque para completar el objetivo. La salida es coherente: no pasa por precipicios, consigue el objetivo, obtiene el bikini antes de pasar por el agua

y obtiene las zapatillas antes de pasar por el bosque, así damos por correcto el plan para el problema.

Hemos ejecutado el problema con -O, ya no ejecutaremos más con esa opción pues ha tardado 56.70 segundos en crear el plan.

Ejercicio 4.a: Hemos añadido dos funciones más: PuntosTotales que contabilizará los puntos del jugador en cuestión y PuntosObjetoPersonaje <Objeto> <Personaje> que contabilizará cuantos puntos se obtendrán al dar el objeto dado al personaje en cuestión. Así es obvio que la única acción a modificar sea Entregar, que incrementará los PuntosTotales del jugador al entregar el objeto al personaje dado.

Ejercicio 4.b: Hemos incluido la tabla dada mediante la funcion PuntosObjetoPersonaje y hemos inicializado PuntosTotales a cero. Por otro lado, hemos incluido en el objetivo la obtención de unos determinados puntos (problema 1), mientras que hemos creado otro problema sin especificar este objetivo (problema 2).



Vemos que en el problema 1 (que especificamos objeter 50 puntos (puntuación maxima)), los objetos entregados a los personajes son los que más valor pueden dar (10 puntos), mientras que en el problema 2, los objetos dados no tienen por que ser los de mayor valor (pues no se especifica nada en el objetivo). Consideramos correcto el plan de resolución.

Ejercicio 5.a: Hemos creado dos funciones para implementar la funcionalidad del Bolsillo: CapacidadBolsillo <Personaje> que será el máximo número de objetos que puede tener el personaje dado y ObjetosBolsillo <Personaje> que será el número de objetos actual del personaje dado.

La única acción que habrá que modificar será la de Entregar, ya que deberemos comprobar que el personaje al que entregamos un objeto dispone de espacio en el bolsillo, y si esto es cierto, incrementar en 1 los objetos actuales del personaje.

Ejercicio 5.b: Hemos mantenido el problema 1 del anterior ejercicio pero ahora le hemos dado a la bruja un bolsillo de capacidad 4 y a la princesa un bolsillo de capacidad 3, los otros 3 personajes no tienen capacidad en el bolsillo. La solución sería tal que:

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej5solucion1
.txt | grep ENTREGAR
      14: ENTREGAR ROSA AGENTE Z21 PRINCESA
      33: ENTREGAR MANZANA AGENTE Z12 BRUJA
      41: ENTREGAR ORO AGENTE Z21 PRINCESA
      43: ENTREGAR OSCAR AGENTE Z21 PRINCESA
```

Donde vemos que la princesa obtiene 3 objetos y la bruja obtiene 1.

Ejercicio 6.a: Para introducir más jugadores a la partida hemos tenido que modificar algunos de los predicados introduciéndole un parámetro que indique el jugador sobre el que es verdad, estos son: ObjetoCogido <Objeto> pasa a ser ObjetoCogido <Objeto> <Player>; ManoLibre pasa a ser ManoLibre <Player>; MochilaLibre se modifica a MochilaLibre <Player> y ObjetoEnLaMochila <Objeto> pasa a ser ObjetoEnLaMochila <Objeto> <Player>.

Además hemos creado una función nueva: PuntosJugador <Player> que cuantifica cuántos puntos tiene el jugador dado.

La acción a modificar para introducir esta información es Entregar (a parte de aquellas que usaban los predicados anteriores). Ahora, Entregar tendrá otro efecto: aumentar la función PuntosJugador del jugador que entrega el objeto al personaje.

Ejercicio 6.b: El problema base que hemos estado siguiendo se ha mejorado para introducir esta actualización: se crean dos Player (Agente1 y Agente2) y se inicializan sus puntos a cero y en el objetivo pedimos que tanto uno como otro sumen puntos.

Aparte hemos creado otros dos problemas en el que los objetivos varían: uno impide que un jugador consiga puntos y el otro impide que el otro jugador consiga puntos, veamos los resultados:

Problema en el se pide que ambos jugadores obtengan un mínimo de puntos (problema 1):

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej6solucion1
.txt | grep ENTREGAR
      18: ENTREGAR ORO AGENTE1 Z16 PRINCIPE
      28: ENTREGAR ROSA AGENTE2 Z21 PRINCESA
      45: ENTREGAR MANZANA AGENTE2 Z12 BRUJA
      67: ENTREGAR OSCAR AGENTE2 Z17 DICAPRIO
```

Problema en el se pide que Agente1 no obtenga ningún punto (problema 2):

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej6solucion2
.txt | grep ENTREGAR
       6: ENTREGAR ORO AGENTE2 Z16 PRINCIPE
      17: ENTREGAR ROSA AGENTE2 Z21 PRINCESA
      51: ENTREGAR OSCAR AGENTE2 Z17 DICAPRIO
      62: ENTREGAR MANZANA AGENTE2 Z12 BRUJA
```

Problema en el se pide que Agente2 no obtenga ningún punto (problema 3):

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej6solucion3
.txt | grep ENTREGAR
      18: ENTREGAR ORO AGENTE1 Z16 PRINCIPE
      42: ENTREGAR MANZANA AGENTE1 Z12 BRUJA
      64: ENTREGAR OSCAR AGENTE1 Z17 DICAPRIO
      77: ENTREGAR ROSA AGENTE1 Z21 PRINCESA
```

Damos los planes por correctos.

Ejercicio 7.a: Para introducir los dos tipos de Jugadores hemos incluido dos subtipos, Picker y Dealer. Hemos cambiado la función PuntosJugador <Player> a PuntosJugador <Dealer> ya que Picker no podrá nunca obtener puntos, pues no puede entregar objetos a un personaje. Hemos modificado la acción Coger para que ahora el Player deba ser un Picker; también hemos actualizado la acción Entregar para que ahora el Player deba ser un Dealer; y hemos creado una acción nueva PasarADealer que le otorga un objeto de Picker a Dealer.

Ejercicio 7.b: Para probar lo anterior, lo único que habría que modificar es quién de los player es Dealer y quien es Picker (en nuestro problema, Agente1 será el Dealer y Agente2 será el Picker). Inicializamos los puntos de Agente1 a cero (pues será el único que podrá conseguir puntos). Y ejercutamos:

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej7solucion1
.txt | grep ENTREGAR
       18: ENTREGAR ORO AGENTE1 Z16 PRINCIPE
       33: ENTREGAR ROSA AGENTE1 Z21 PRINCESA
       57: ENTREGAR MANZANA AGENTE1 Z12 BRUJA
       82: ENTREGAR OSCAR AGENTE1 Z17 DICAPRIO
```

Hemos probado a introducir otro Dealer, Agente3, en el mapa (problema2):

```
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej7solucion2
.txt | grep ENTREGAR
       10: ENTREGAR ORO AGENTE3 Z16 PRINCIPE
       25: ENTREGAR ROSA AGENTE3 Z21 PRINCESA
       46: ENTREGAR MANZANA AGENTE1 Z12 BRUJA
       66: ENTREGAR OSCAR AGENTE1 Z17 DICAPRIO
alberto@alberto-Lenovo-ideapad-330-15IKB:~/Documentos/Cuarto/TSI/Practicas/P2/Nu
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej7solucion2
.txt | grep COGER
        6: COGER ORO AGENTE2 Z22
       18: COGER ROSA AGENTE2 Z18
       37: COGER MANZANA AGENTE2 Z8
       57: COGER OSCAR AGENTE2 Z2
alberto@alberto-Lenovo-ideapad-330-15IKB:~/Documentos/Cuarto/TSI/Practicas/P2/Nu
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej7solucion2
.txt | grep DARADEALER
alberto@alberto-Lenovo-ideapad-330-15IKB:~/Documentos/Cuarto/TSI/Practicas/P2/Nu
eva_Version_Metric-FF/Metric-FF$ cat ../../DomYProbs/Soluciones.txt/Ej7solucion2
.txt | grep PASARADEALER
        9: PASARADEALER AGENTE2 AGENTE3 ORO Z16
       24: PASARADEALER AGENTE2 AGENTE3 ROSA Z21
       40: PASARADEALER AGENTE2 AGENTE1 MANZANA Z8
       62: PASARADEALER AGENTE2 AGENTE1 OSCAR Z9
```

En la imagen anterior podemos comprobar que solo Cogen objetos el Agente2 (Picker) y solo Entregan a personajes Agente1 y Agente3 (ambos Dealer) y los intercambios son correctos.