Técnicas de los Sistemas Inteligentes (2016-2017)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 3

Definición de Dominios y Problemas de Planificación

Relación de ejercicios prácticos 1

Javier Béjar Méndez
45337539p
javierbejarmendez@correo.ugr.es

6 de junio de 2017

Índice

Índice de figuras

1. Relación de Ejercicios Prácticos 1

Relación de ejercicios de definición de dominios y problemas en el lenguaje de planificación PDDL[1], usando el planificador Metric-FF para el cálculo de planes. El problema sobre el que trabajaremos se llama *Los extraños mundos de Belkan*.

1.1. Ejercicio 1

Para este ejercicio se ha definido un mapa cuadrícula $4x4$ con todas las ciudades interconectadas, simplificando el aportado por el profesor, con los 5 personajes y los 5 objetos, para ver la distribución exacta ver el docuento *problema.pddl*.

- Apartado a. En esta sección definimos los distintos objetos del mundo, como podemos observar a continuación:

```
(:types localizable zona orientacion - object
        persona objeto - localizable
        jugador personaje - persona)
```

Figura 1.1: Definición de los distintos objetos del mundo.

He definido los tipos mediante una jerarquía, el supertipo localizable representa cualquier objeto que puede situarse en una zona, que son los objetos y las personas. Las personas son entes que pueden tener objetos, distinguiendo entre personajes (npcs) y jugador. Por ultimo tenemos el tipo zona para la definición de las zonas y el tipo orientación para representar los puntos cardinales. Se ha seguido esta metodología para evitar crear predicados redundantes del tipo *(en ?p - personaje ?z - zona)*, *(en ?o - objeto ?z - zona)*, simplificandolos, *(en ?l - localizable ?z - zona)*.

- Apartado b. Se definen los predicados necesarios para representar el estado del mundo en un momento dado, que son los siguientes:

```
(:predicates
    ;orientacion
    (siguienteDER ?x1 - orientacion ?x2 - orientacion)
    (siguienteIZQ ?x1 - orientacion ?x2 - orientacion)

    ;posicion
    (en ?l - localizable ?z - zona)

    ;tenencia
    (tiene ?p - persona ?o - objeto)
    (quiere ?p - personaje ?o - objeto)

    ;orientado
    (orientado ?p - persona ?x - orientacion)

    ;zonas
    (camino ?z1 - zona ?x - orientacion ?z2 - zona)
    |
    )

(:functions
    (mano ?j - jugador)
    (coste ?z1 - zona ?z2 - zona)
)
```

Figura 1.2: Definición de los distintos predicados.

Los predicados *siguienteDER* y *siguienteIZQ* nos permiten calcular la orientación resultante al girar a la derecha e izquierda respectivamente, siendo ?$x1$ la orientación inicial, y ?$x2$ la orientación resultante.

El predicado *en* nos permite describir que un objeto de tipo *localizable* se encuentra en una determinada *zona*.

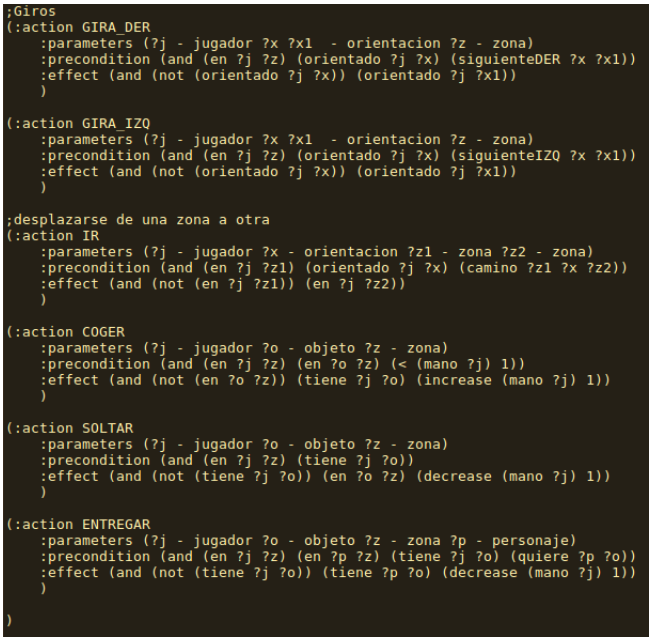
El predicado *tiene* nos indica que una *persona* tiene un *objeto*, se utiliza sobre todo para representar que objeto tiene el jugador en la mano y definir el goal para entregar los objetos a los distintos personajes, siendo el predicado *quiere* el que describe qué *personaje* debe recibir qué *objeto*, que se usará en la acción *ENTREGAR*.

El predicado *orientado* se utiliza para describir la orientación del *jugador*.

Por último tenemos el predicado *camino* que describe la existencia de un camino entre dos zonas en una orientación determinada (Norte, Este, Oeste y Sur).

También he definido las funciones *mano* y *coste*, que se utilizan para comprobar que el jugador no pueda tener más de un objeto en la mano y para obtener el coste de un camino respectivamente.

- Apartado c. En este apartado se definen las siguientes acciones:



```
;Giros
(:action GIRA_DER
    :parameters (?j - jugador ?x ?x1  - orientacion ?z - zona)
    :precondition (and (en ?j ?z) (orientado ?j ?x) (siguienteDER ?x ?x1))
    :effect (and (not (orientado ?j ?x)) (orientado ?j ?x1))
    )

(:action GIRA_IZQ
    :parameters (?j - jugador ?x ?x1  - orientacion ?z - zona)
    :precondition (and (en ?j ?z) (orientado ?j ?x) (siguienteIZQ ?x ?x1))
    :effect (and (not (orientado ?j ?x)) (orientado ?j ?x1))
    )

;desplazarse de una zona a otra
(:action IR
    :parameters (?j - jugador ?x - orientacion ?z1 - zona ?z2 - zona)
    :precondition (and (en ?j ?z1) (orientado ?j ?x) (camino ?z1 ?x ?z2))
    :effect (and (not (en ?j ?z1)) (en ?j ?z2))
    )

(:action COGER
    :parameters (?j - jugador ?o - objeto ?z - zona)
    :precondition (and (en ?j ?z) (en ?o ?z) (< (mano ?j) 1))
    :effect (and (not (en ?o ?z)) (tiene ?j ?o) (increase (mano ?j) 1))
    )

(:action SOLTAR
    :parameters (?j - jugador ?o - objeto ?z - zona)
    :precondition (and (en ?j ?z) (tiene ?j ?o))
    :effect (and (not (tiene ?j ?o)) (en ?o ?z) (decrease (mano ?j) 1))
    )

(:action ENTREGAR
    :parameters (?j - jugador ?o - objeto ?z - zona ?p - personaje)
    :precondition (and (en ?j ?z) (en ?p ?z) (tiene ?j ?o) (quiere ?p ?o))
    :effect (and (not (tiene ?j ?o)) (tiene ?p ?o) (decrease (mano ?j) 1))
    )

)
```

Figura 1.3: Definición de las distintas acciones.

Las acciones *GIRA_DER* y *GIRA_IZQ* permiten girar al jugador, haciendo uso de los predicados descritos anteriormente, el resultado de esta acción es que se niega el predicado con la orientación del jugador existente y se define un predicado con la nueva orientación

La acción *IR* permite al jugador desplazarse de una zona a otra, siempre y cuando exista un camino entre las zonas mediante una orientación igual a la del jugador, el resultado es la negación de la situación del jugador y la creación del predicado con la nueva situación del jugador.

La acción *COGER* permite al jugador coger un objeto en la mano, siempre y cuando el objeto se encuentre en la misma zona que el jugador, y la función mano sea menor a uno, es decir, que no tenga ya ningún objeto en la mano. El resultado es la negación del predicado donde estaba el objeto, y la inclusión del predicado de tenencia para el jugador con dicho objeto, aumentando en uno la función *mano*.

La acción *SOLTAR* hace lo contrario que la de coger, siempre y cuando tenga un objeto en la mano es capaz de soltarlo, negando la tenencia de dicho objeto e incluyendo la presencia del objeto en la zona, disminuyendo la función *mano*.

La acción *ENTREGAR* permite dar objetos que el jugador posea a los distintos personajes, mediante el predicado *quiere* la acción entrega el objeto al personaje correspondiente. El resultado es negación de tenencia del jugador y la inclusión de

tenencia del personaje, dismiyundo la función *mano*.

- Apartado d. En el siguiente apartado se describe el problema con 25 zonas,, 5 personajes y 5 objetos, la definición del prblema se muestra en las siguientes figuras:

```
(define (problem ejercicio1)

  (:domain Belkan)

  (:objects
    ZONE_00_00 ZONE_00_01 ZONE_00_02 ZONE_00_03 - zona
    ZONE_01_00 ZONE_01_01 ZONE_01_02 ZONE_01_03 - zona
    ZONE_02_00 ZONE_02_01 ZONE_02_02 ZONE_02_03 - zona
    ZONE_03_00 ZONE_03_01 ZONE_03_02 ZONE_03_03 - zona


    NORTE - orientacion
    SUR - orientacion
    ESTE - orientacion
    OESTE - orientacion

    zes - jugador
    Bruja - personaje
    Principe - personaje
    Princesa Profesor Leonardo - personaje

    oro - objeto
    manzana - objeto
    algoritmo - objeto
    oscar - objeto
    rosa - objeto

  )
```

Figura 1.4: Definición de los distintos objetos del mundo.

```
(:init

  ;giros
  (siguienteDER NORTE ESTE)
  (siguienteDER ESTE SUR)
  (siguienteDER SUR OESTE)
  (siguienteDER OESTE NORTE)

  (siguienteIZQ NORTE OESTE)
  (siguienteIZQ OESTE SUR)
  (siguienteIZQ SUR ESTE)
  (siguienteIZQ ESTE NORTE)

  ;Iniciales
  (en zes ZONE_00_00)
  (orientado zes NORTE)
  (= (mano zes) 0)
  (quiere Principe oro)
  (quiere Princesa rosa)
  (quiere Profesor algoritmo)
  (quiere Leonardo oscar)
  (quiere Bruja manzana)

  ; Posion de los personajes
  (en Principe ZONE_00_01)
  (en Bruja ZONE_01_03)
  (en Profesor ZONE_00_03)
  (en Princesa ZONE_01_01)
  (en Leonardo ZONE_02_02)


  ; Posicion objetos
  (en oro ZONE_03_01)
  (en manzana ZONE_00_00)
  (en Oscar ZONE_02_01)
  (en Algoritmo ZONE_03_03)
  (en rosa ZONE_01_02)
```

Figura 1.5: Definición de los predicados iniciales.

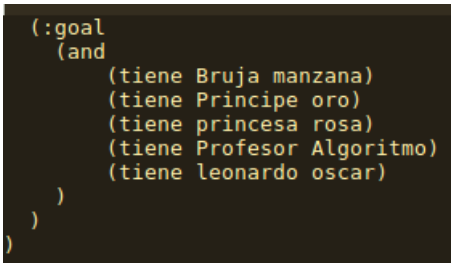


Figura 1.6: Definición del goal, objetivo a alcanzar por el planificador.

Se ha omitido la definición del mapa debido a su longitud, pero solo consiste en definir los predicados iniciales *camino* para definir todas las conexiones entre las zonas. Tras ejecutar el planifiador con optimización y los parámetros $-g$ 1 y $-h$ 1 se ha obtenido el siguiente plan, minimizando el número de acciones por defecto:

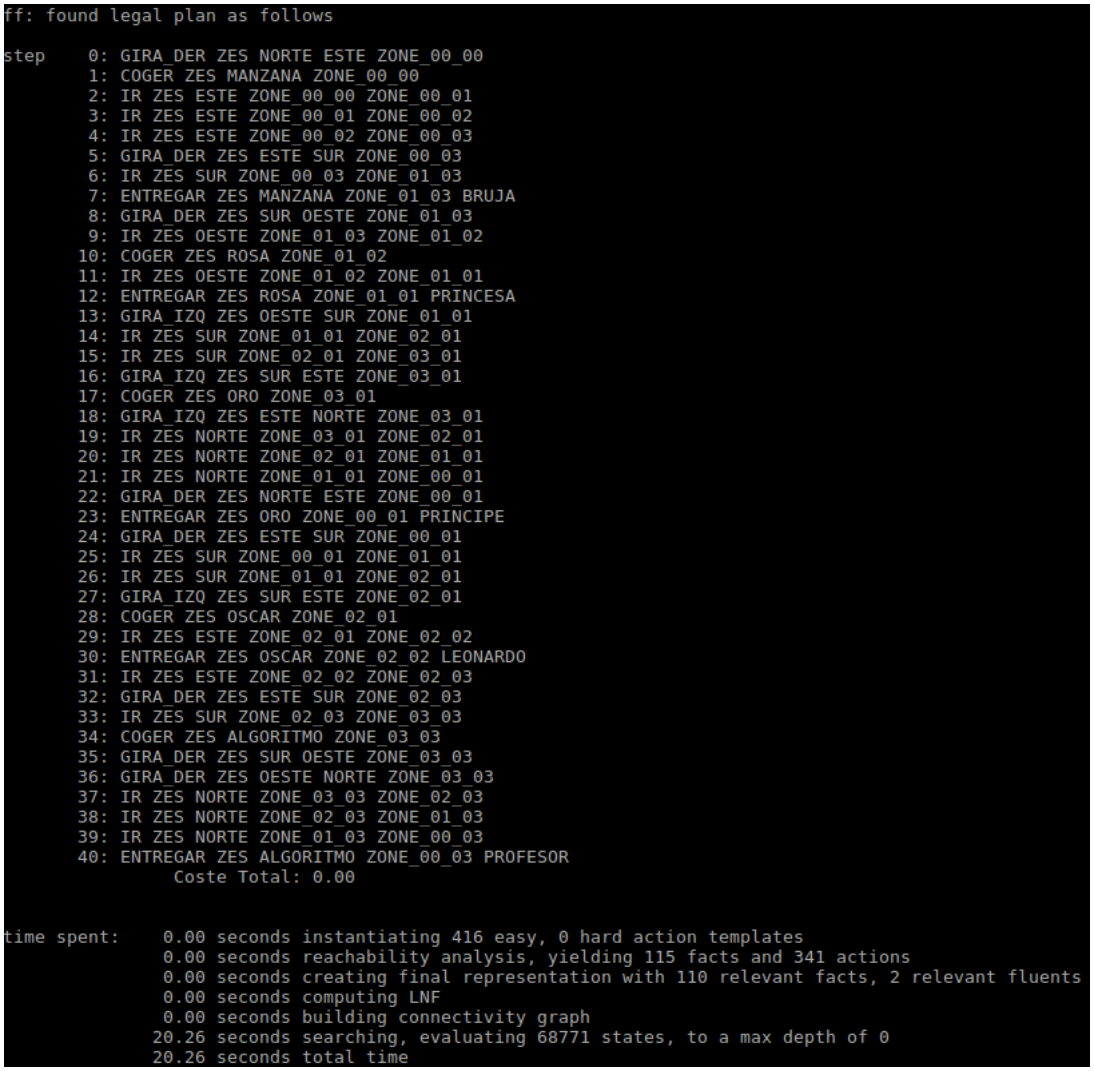


Figura 1.7: Plan obtenido por el planificador Metric-FF.

1.2. Ejercicio 2

En este ejercicio se introduce que desplazarse de una ciudad a otra conlleva un coste igual a la longitud del camino. Se ha utilizado el mismo mapa que el anterior, facilitando la distribución de los objetos y personajes, ya que con optimización no es capaz de resolverlo si el mapa es muy complejo, una vez más consultar la descripción del problema para ver la distribución exacta.

- Apartado a. Los cambios presentes en el dominio son la introducción de la función *coste*, descrita en el ejercicio anterior, y la función $distancia-total$ que acumula el coste de los caminos recorridos hasta el momento, como observamos en la siguiente figura:

```
(:functions
    (mano)
    (coste ?z1 - zona ?z2 - zona)
    (distancia-total)
)
```

Figura 1.8: Funciones asociadas al coste.

Y la actualización de la acción IR para que contabilice el coste del camino, como se muestra a continuación:

```
;desplazarse de una zona a otra
(:action IR
    :parameters (?j - jugador ?x - orientacion ?z1 - zona ?z2 - zona)
    :precondition (and (en ?j ?z1) (orientado ?x) (camino ?z1 ?x ?z2))
    :effect (and (not (en ?j ?z1)) (en ?j ?z2) (increase (distancia-total) (coste ?z1 ?z2)))
    )
```

Figura 1.9: Acción IR actualizada para contabilizar los costes de los caminos.

- Apatado b. Se ha introducido la inicialización de la funcion *coste* para todos los caminos y la inicialización de la función $distancia-total$ a 0, como se muestra en la siguiente figura:

```
    ;Iniciales
    (en zes ZONE_00_00)
    (orientado NORTE)
    (= (mano) 0)
    (= (distancia-total) 0)
    (quiere Principe oro)
    (quiere Princesa rosa)
```

Figura 1.10: Funcion $distancia-total$ asociada al coste.

Ahora ejecutamos el planificador sin opciones de optimización, obteniendo el siguiente plan:

```
ff: found legal plan as follows

step    0: GIRA_DER NORTE ESTE ZONE_04_04
        1: IR ZES ESTE ZONE_00_00 ZONE_00_01
        2: COGER ZES ORO ZONE_00_01
        3: IR ZES ESTE ZONE_00_01 ZONE_00_02
        4: ENTREGAR ZES ORO ZONE_00_02 PRINCIPE
        5: IR ZES ESTE ZONE_00_02 ZONE_00_03
        6: GIRA_DER ESTE SUR ZONE_04_04
        7: COGER ZES MANZANA ZONE_00_03
        8: IR ZES SUR ZONE_00_03 ZONE_01_03
        9: GIRA_DER SUR OESTE ZONE_04_04
       10: ENTREGAR ZES MANZANA ZONE_01_03 BRUJA
       11: IR ZES OESTE ZONE_01_03 ZONE_01_02
       12: COGER ZES OSCAR ZONE_01_02
       13: IR ZES OESTE ZONE_01_02 ZONE_01_01
       14: GIRA_IZQ OESTE SUR ZONE_04_04
       15: ENTREGAR ZES OSCAR ZONE_01_01 LEONARDO
       16: IR ZES SUR ZONE_01_01 ZONE_02_01
       17: GIRA_DER SUR OESTE ZONE_04_04
       18: IR ZES OESTE ZONE_02_01 ZONE_02_00
       19: GIRA_IZQ OESTE SUR ZONE_04_04
       20: COGER ZES ALGORITMO ZONE_02_00
       21: IR ZES SUR ZONE_02_00 ZONE_03_00
       22: GIRA_IZQ SUR ESTE ZONE_04_04
       23: ENTREGAR ZES ALGORITMO ZONE_03_00 PROFESOR
       24: IR ZES ESTE ZONE_03_00 ZONE_03_01
       25: COGER ZES ROSA ZONE_03_01
       26: IR ZES ESTE ZONE_03_01 ZONE_03_02
       27: ENTREGAR ZES ROSA ZONE_03_02 PRINCESA
             Coste Total: 0.00


time spent:    0.00 seconds instantiating 623 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 115 facts and 413 actions
               0.00 seconds creating final representation with 110 relevant facts, 3 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 72 states, to a max depth of 3
               0.00 seconds total time
```

Figura 1.11: Plan sin optimización para las nuevas restricciones.

- Apartado c. Ahora le indicamos al planificador que tenga en cuenta los costes de los caminos y minimice el coste total, mediante la sentencia : *metric* como se observa en la siguiente figura:

```
  (:goal
    (and
        (tiene Bruja manzana)
        (tiene Principe oro)
        (tiene princesa rosa)
        (tiene Profesor Algoritmo)
        (tiene leonardo oscar)

    )
  )
  (:metric minimize (distancia-total))
)
```

Figura 1.12: Como indicar al planificador que minimize una función.

Una vez realizado esto, lanzamos el planificador con opciones de optimización óptimas, obteniendo el siguiente plan:

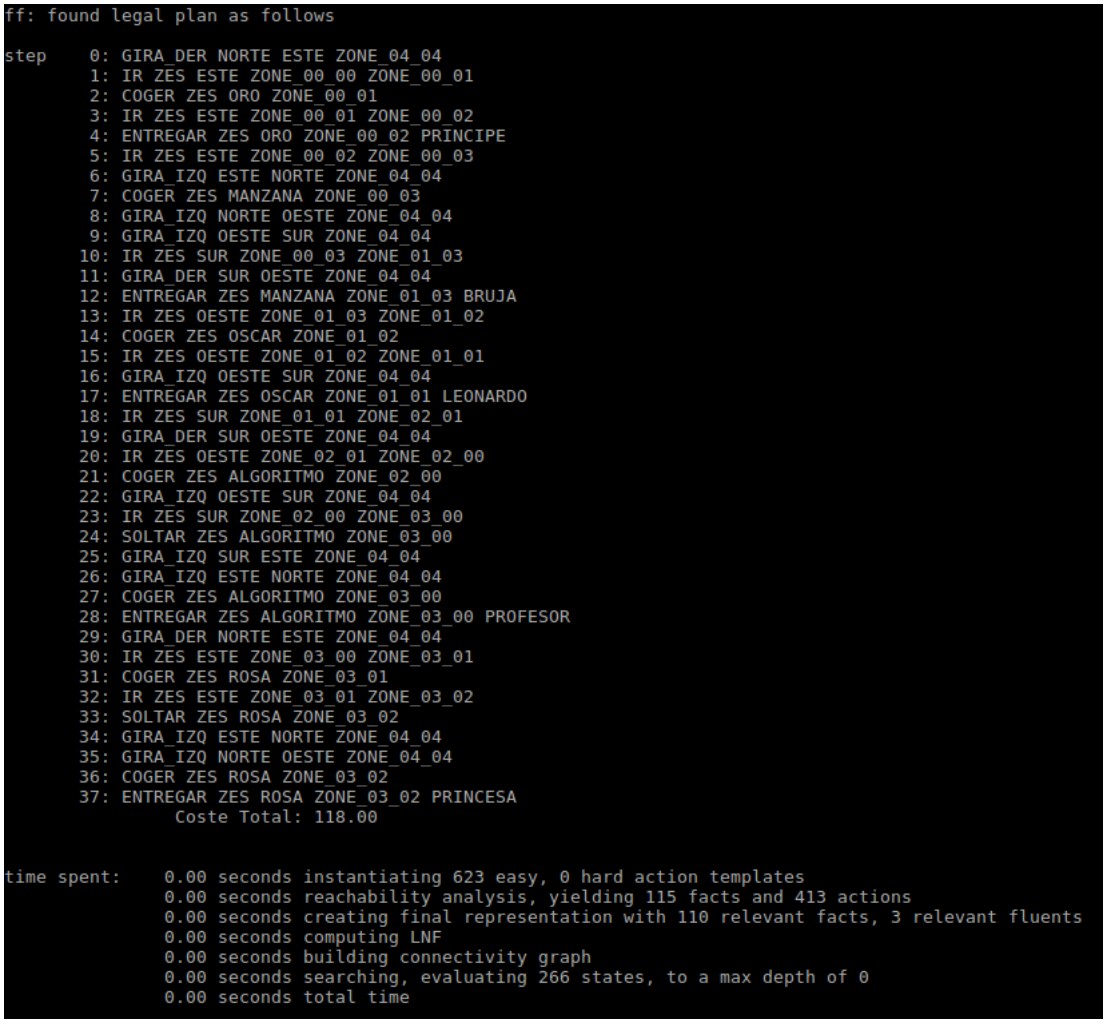


Figura 1.13: Plan minimizando el coste del recorrido total.

Podemos observar que el plan resultante contiene mayor número de acciones que el anterior ya que busca el camino óptimo, obteniendo un coste óptimo de 118.

1.3. Ejercicio 3

En este ejercicio introduciremos los distintos tipos de zonas (bosque, agua, arenoso, pedregoso y precipicio) y dos nuevos objetos (zapatillas y bikini), done para poder ir a una zona de agua necesitamos el bikini, y para ir a una zona de bosque las zapatillas. Las zonas de precipicio son intransitables. Ademas el jugador cuenta con una mochila de capacidad 1, donde puede guardar un objeto. Para la ejecución se ha utilizado el mismo mapa, situando una zapatilla en terreno AGUA y un bikini en terreno BOSQUE, y situando personajes en BOSQUE y AGUA para forzar que tenga que pasar por dichos terrenos.

- Apartado a. Los cambios introducidos al dominio para satisfacer las nuevas restricciones son los siguientes:

```
(:types localizable zona orientacion tipoterreno - object
        persona objeto - localizable
        jugador personaje - persona
        )

(:constants
    AGUA PRECIPICIO BOSQUE - tipoterreno
    BIKINI ZAPATILLAS - objeto
    )

(:predicates
    ;orientacion
    (siguienteDER ?x1 - orientacion ?x2 - orientacion)
    (siguienteIZQ ?x1 - orientacion ?x2 - orientacion)

    ;posicion
    (en ?l - localizable ?z - zona)

    ;tenencia
    (tiene ?p - persona ?o - objeto)
    (guardado ?o - objeto)
    (quiere ?p - personaje ?o - objeto)

    ;orientado
    (orientado ?x - orientacion)

    ;zonas
    (camino ?z1 - zona ?x - orientacion ?z2 - zona)
    (isZone ?z - zona ?t - tipoterreno)


    )
```

Figura 1.14: Dominio con los nuevos tipos de terreno y objetos.

Se ha introducido el tipo *tipoterreno* para especificar el terreno de una zona. Se han declarado las constantes para los nuevos terrenos y objetos, veremos su uso en la acción IR. Se ha añadido la función *mochila*, que tiene el mismo funcionamiento que la función *mano*, calcula que en la mochila solo se introduzca un objeto. Se ha añadido el predicado *guardado* que hace referencia al objeto guardado en la mochila. Por último se ha añadido el predicado *isZone* que relaciona una *zona* con su *tipoterreno*.

En cuanto a la acción IR se ha redefinido como sigue:

```
;desplazarse de una zona a otra
(:action IR
    :parameters (?j - jugador ?x - orientacion ?z1 - zona ?z2 - zona ?t - tipoterreno ?o - objeto)
    :precondition (and (en ?j ?z1) (orientado ?x) (camino ?z1 ?x ?z2) (isZone ?z2 ?t)
                  (not (isZone ?z2 PRECIPICIO))
    )

    :effect (and
                (when (and (= ?t AGUA) (or (tiene ?j BIKINI) (guardado BIKINI)))
                    (and (not (en ?j ?z1)) (en ?j ?z2) (increase (distancia-total) (coste ?z1 ?z2))) )
                (when (and (= ?t BOSQUE) (or (tiene ?j ZAPATILLAS) (guardado ZAPATILLAS)))
                    (and (not (en ?j ?z1)) (en ?j ?z2) (increase (distancia-total) (coste ?z1 ?z2))) )
                (when (or (= ?t ARENOSO) (= ?t PEDREGOSO))
                    (and (not (en ?j ?z1)) (en ?j ?z2) (increase (distancia-total) (coste ?z1 ?z2))) )
        )
    )
```

Figura 1.15: Función *IR* actualizada a los nuevos terrenos.

En la precondición se ha introducido el predicado *isZone* y se comprueba que el valor de ?*t* - *tipoterreno* no sea precipicio, haciendo uso de las constantes previamente declaradas. En el efecto de la acción hacemos uso de los condicionales *when* para comprobar que si el tipo de terreno es agua o bosque el jugador tenga los respectivos objetos, ya sea en la mochila o en la mano, para poder avanzar a la zona. Cuando el tipo es arenoso o pedregoso puede desplazarse normalmente.

- Apartado b. Para poder guardar y sacar objetos de la mochila se han definido las siguientes acciones:

```
(:action GUARDAR
    :parameters(?o - objeto ?j - jugador)
    :precondition(and (tiene ?j ?o) (< (mochila) 1))
    :effect(and (not (tiene ?j ?o)) (increase (mochila) 1) (decrease (mano) 1) (guardado ?o))
    )

(:action SACAR
    :parameters (?o - objeto ?j - jugador)
    :precondition(and (< (mano) 1) (guardado ?o))
    :effect(and (not (guardado ?o)) (increase (mano) 1) (decrease (mochila) 1) (tiene ?j ?o))
    )
```

Figura 1.16: Acciones para sacar y guardar objetos de la mochila.

La función *GUARDAR* tiene como precondición que el jugador tenga el objeto en la mano y que la función de la mochila devuelva menor que uno, es decir, que no tenga ningún objeto. El resultado es la negación del predicado *tiene* y la inclusión del predicado *guardado*, incrementando la función *mochila* y decrementando la función *mano*.

La función *SACAR* hace exactamente la operación inversa, comprueba que *mano* sea menor que uno (tiene la mano libre) y el resultado es la negación de *guardado* y la inclusión de *tiene*, decrementando *mochila* e incrementando *mano*.

- Apartado c. La definición del problema resultante es la siguiente:

```
(define (problem ejercicio1)

  (:domain Belkan)

  (:objects
    ZONE_00_00 ZONE_00_01 ZONE_00_02 ZONE_00_03 - zona
    ZONE_01_00 ZONE_01_01 ZONE_01_02 ZONE_01_03 - zona
    ZONE_02_00 ZONE_02_01 ZONE_02_02 ZONE_02_03 - zona
    ZONE_03_00 ZONE_03_01 ZONE_03_02 ZONE_03_03 - zona

    ARENOSO PEDREGOSO AGUA BOSQUE PRECIPICIO - tipoterreno

    NORTE SUR ESTE OESTE - orientacion

    zes - jugador
    Bruja Principe Princesa Profesor Leonardo - personaje

    oro manzana algoritmo oscar rosa zapatillas bikini - objeto
  )
```

Figura 1.17: Declaración de los objetos, incluyendo los tipos de terreno y los nuevos objetos.

```
    ; Posion de los personajes
    (en Principe ZONE_00_02)
    (en Bruja ZONE_01_03)
    (en Profesor ZONE_03_00)
    (en Princesa ZONE_03_02)
    (en Leonardo ZONE_01_01)


    ; Posicion objetos
    (en oro ZONE_00_01)
    (en manzana ZONE_00_03)
    (en Oscar ZONE_01_02)
    (en Algoritmo ZONE_02_00)
    (en rosa ZONE_03_01)
    (en zapatillas ZONE_00_03)
    (en bikini ZONE_03_03)
    (en bikini ZONE_00_02)

(isZone ZONE_00_00 ARENOSO)
(isZone ZONE_00_01 PEDREGOSO)
(isZone ZONE_00_02 PEDREGOSO)
(isZone ZONE_00_03 AGUA)
(isZone ZONE_01_00 PEDREGOSO)
(isZone ZONE_01_01 BOSQUE)
(isZone ZONE_01_02 PEDREGOSO)
(isZone ZONE_01_03 PEDREGOSO)
(isZone ZONE_02_00 PEDREGOSO)
(isZone ZONE_02_01 PRECIPICIO)
(isZone ZONE_02_02 Pedregoso)
(isZone ZONE_02_03 AGUA)
(isZone ZONE_03_00 BOSQUE)
(isZone ZONE_03_01 BOSQUE)
(isZone ZONE_03_02 PEDREGOSO)
(isZone ZONE_03_03 BOSQUE)
```

Figura 1.18: Posición de los personajes, objetos y asignación de los terrenos a las distintas zonas.

```
;Iniciales
(en zes ZONE_00_00)
(orientado NORTE)
(= (mano) 0)
(= (distancia-total) 0)
(= (mochila) 0)
```

Figura 1.19: Inicialización de la nueva función mochila.

La ejecución del planificador obtiene un plan con coste óptimo y uso correcto de los nuevos elementos introducidos al problema, debido a la longitud de dicho plan no lo he introducido en la memoria, puede consultarse en la carpeta $E3$, nombrado como *plan_optimo.txt*.

1.4. Ejercicio 4

En este ejercicio cambiaremos el goal, representando que la entrega de un objeto conlleva unos puntos dependiendo del personaje al que le sea entregado, según esta tabla:

	Leonardo	Princesa	Bruja	Profesor	Príncipe
Oscar	10	5	4	3	1
Rosa	1	10	5	4	3
Manzana	3	1	10	5	4
Algoritmo	4	3	1	10	5
Oro	5	4	3	1	10

Figura 1.20: Tabla de puntuación.

- Apartado a. La función que calcula los puntos se define como sigue:

```
(:functions
    (mano)
    (mochila)
    (coste ?z1 - zona ?z2 - zona)
    (distancia-total)
    (puntos-entrega ?p - personaje ?o - objeto)
    (puntos-total)
)
```

Figura 1.21: Función que calcula los puntos por entrega de objetos.

La función *puntos − entrega ?p ?o* devuelve los puntos por entregar el objeto *?o* al personaje *?p*. Mientras que la función *puntos − total* acumula los puntos

conseguidos. Tambien se ha modificado la acción *ENTREGAR* de la siguiente manera:

```
(:action ENTREGAR
    :parameters (?j - jugador ?o - objeto ?z - zona ?p - personaje)
    :precondition (and (en ?j ?z) (en ?p ?z) (tiene ?j ?o))
    :effect (and (not (tiene ?j ?o)) (decrease (mano) 1) (increase (puntos-total) (puntos-entrega ?p ?o)))
    )
```

Figura 1.22: Acción *ENTREGAR* actualizada para registrar puntos.

Se ha eliminado la condición de que un determinado personaje quiera un determinado objeto, y se ha añadido que se incremente la función $puntos - total$ con la entrega correspondiente.

- Apartado b. Se ha modificado el problema para representar las nuevas restricciones, en la siguiente figura podemos observar las inicializaciones de las nuevas funciones y la situación de los objetos:

```
; Posion de los personajes
(en Principe ZONE_00_02)
(en Bruja ZONE_01_03)
(en Profesor ZONE_03_00)
(en Princesa ZONE_03_02)
(en Leonardo ZONE_01_01)


; Posicion objetos
(en manzana ZONE_00_01)
(en manzana ZONE_00_03)
(en oro ZONE_01_02)
(en manzana ZONE_02_00)
(en oro ZONE_03_01)
(en oro ZONE_02_03)
(en manzana ZONE_00_00)

(en zapatillas ZONE_00_03)
(en bikini ZONE_03_03)
(en bikini ZONE_00_02)

;puntos
(= (puntos-total) 0)
(= (puntos-entrega leonardo oscar) 10)
(= (puntos-entrega leonardo rosa) 1)
(= (puntos-entrega leonardo manzana) 3)
(= (puntos-entrega leonardo algoritmo) 4)
(= (puntos-entrega leonardo oro) 5)

(= (puntos-entrega princesa oscar) 5)
(= (puntos-entrega princesa rosa) 10)
(= (puntos-entrega princesa manzana) 1)
(= (puntos-entrega princesa algoritmo) 3)
(= (puntos-entrega princesa oro) 4)

(= (puntos-entrega bruja oscar) 4)
(= (puntos-entrega bruja rosa) 5)
(= (puntos-entrega bruja manzana) 10)
(= (puntos-entrega bruja algoritmo) 1)
(= (puntos-entrega bruja oro) 3)

(= (puntos-entrega profesor oscar) 3)
(= (puntos-entrega profesor rosa) 4)
(= (puntos-entrega profesor manzana) 5)
(= (puntos-entrega profesor algoritmo) 10)
(= (puntos-entrega profesor oro) 1)

(= (puntos-entrega principe oscar) 1)
(= (puntos-entrega principe rosa) 3)
(= (puntos-entrega principe manzana) 4)
(= (puntos-entrega principe algoritmo) 5)
(= (puntos-entrega principe oro) 10)
```

Figura 1.23: Definición del problema para representar el nuevo sistema de puntuaciones.

En el goal indicamos simplemente que se alcancen 50 puntos, como podemos observar a continuación:

```
(:goal
  (= (puntos-total) 50)

)
```

Figura 1.24: Goal para alcanzar la puntuación de 50 puntos.

Para este ejercicio se ha lanzado el clasificador sin optimización, ya que no es capaz de resolver el problema con un mapa de 4x4 con parámetros de optimización. El

resultado obtenido es el siguiente:

```
ff: found legal plan as follows

step    0: GIRA_IZQ NORTE OESTE ZONE_03_03
        1: GIRA_IZQ OESTE SUR ZONE_03_03
        2: GIRA_IZQ SUR ESTE ZONE_03_03
        3: IR ZES ESTE ZONE_00_00 ZONE_00_01 PEDREGOSO BIKINI
        4: COGER ZES MANZANA ZONE_00_01
        5: IR ZES ESTE ZONE_00_01 ZONE_00_02 PEDREGOSO BIKINI
        6: GIRA_DER ESTE SUR ZONE_03_03
        7: IR ZES SUR ZONE_00_02 ZONE_01_02 PEDREGOSO BIKINI
        8: GIRA_IZQ SUR ESTE ZONE_03_03
        9: IR ZES ESTE ZONE_01_02 ZONE_01_03 PEDREGOSO BIKINI
       10: ENTREGAR ZES MANZANA ZONE_01_03 BRUJA
       11: GIRA_IZQ ESTE NORTE ZONE_03_03
       12: GIRA_IZQ NORTE OESTE ZONE_03_03
       13: IR ZES OESTE ZONE_01_03 ZONE_01_02 PEDREGOSO BIKINI
       14: GIRA_DER OESTE NORTE ZONE_03_03
       15: COGER ZES ORO ZONE_01_02
       16: IR ZES NORTE ZONE_01_02 ZONE_00_02 PEDREGOSO BIKINI
       17: ENTREGAR ZES ORO ZONE_00_02 PRINCIPE
       18: COGER ZES BIKINI ZONE_00_02
       19: GIRA_DER NORTE ESTE ZONE_03_03
       20: IR ZES ESTE ZONE_00_02 ZONE_00_03 AGUA BIKINI
       21: GIRA_DER ESTE SUR ZONE_03_03
       22: GUARDAR BIKINI ZES
       23: COGER ZES MANZANA ZONE_00_03
       24: IR ZES SUR ZONE_00_03 ZONE_01_03 PEDREGOSO BIKINI
       25: ENTREGAR ZES MANZANA ZONE_01_03 BRUJA
       26: IR ZES SUR ZONE_01_03 ZONE_02_03 AGUA BIKINI
       27: GIRA_DER SUR OESTE ZONE_03_03
       28: COGER ZES ORO ZONE_02_03
       29: IR ZES OESTE ZONE_02_03 ZONE_02_02 PEDREGOSO BIKINI
       30: GIRA_DER OESTE NORTE ZONE_03_03
       31: IR ZES NORTE ZONE_02_02 ZONE_01_02 PEDREGOSO BIKINI
       32: IR ZES NORTE ZONE_01_02 ZONE_00_02 PEDREGOSO BIKINI
       33: ENTREGAR ZES ORO ZONE_00_02 PRINCIPE
       34: GIRA_IZQ NORTE OESTE ZONE_03_03
       35: IR ZES OESTE ZONE_00_02 ZONE_00_01 PEDREGOSO BIKINI
       36: IR ZES OESTE ZONE_00_01 ZONE_00_00 ARENOSO BIKINI
       37: GIRA_IZQ OESTE SUR ZONE_03_03
       38: GIRA_IZQ SUR ESTE ZONE_03_03
       39: COGER ZES MANZANA ZONE_00_00
       40: IR ZES ESTE ZONE_00_00 ZONE_00_01 PEDREGOSO BIKINI
       41: IR ZES ESTE ZONE_00_01 ZONE_00_02 PEDREGOSO BIKINI
       42: IR ZES ESTE ZONE_00_02 ZONE_00_03 AGUA BIKINI
       43: GIRA_DER ESTE SUR ZONE_03_03
       44: IR ZES SUR ZONE_00_03 ZONE_01_03 PEDREGOSO BIKINI
       45: ENTREGAR ZES MANZANA ZONE_01_03 BRUJA
                Coste Total: 0.00


time spent:    0.00 seconds instantiating 926 easy, 44 hard action templates
               0.00 seconds reachability analysis, yielding 92 facts and 306 actions
               0.00 seconds creating final representation with 87 relevant facts, 7 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 249 states, to a max depth of 12
               0.00 seconds total time
```

Figura 1.25: Plan obtenido para alcanzar 50 puntos.

1.5. Ejercicio 5

En este ejercicio se implementa que la mochila tenga una capacidad n.

- Apartado a. El unico cambio requerido para implementar una mochila de tamaño n es definir una función $tam - mochila$, inicializarla a n y que en la función guardar

se compare que la función *mochila* sea menor que $tam - mochila$, como se observa en las siguientes figuras:

```
(:functions
    (mano)
    (mochila)
    (coste ?z1 - zona ?z2 - zona)
    (distancia-total)
    (puntos-entrega ?p - personaje ?o - objeto)
    (puntos-total)
    (tam-mochila)
)
```

Figura 1.26: Declaración de función $tam - mochila$.

```
;Iniciales
(en zes ZONE_00_00)
(orientado NORTE)
(= (mano) 0)
(= (distancia-total) 0)
(= (mochila) 0)
(= (tam-mochila) 4)
```

Figura 1.27: Inicialización de función $tam - mochila$.

```
(:action GUARDAR
    :parameters(?o - objeto ?j - jugador)
    :precondition(and (tiene ?j ?o) (< (mochila) (tam-mochila)))
    :effect(and (not (tiene ?j ?o)) (increase (mochila) 1) (decrease (mano) 1) (guardado ?o))
    )
```

Figura 1.28: Cambios en la acción $GUARDAR$ para implementar una mochila de tamaño n.

- Apartado b. Se ha definido un problema sencillo con 3 manzanas y un principe para observar la capacidad de la mochila y su eficacia. Los estados iniciales usados son los siguientes:

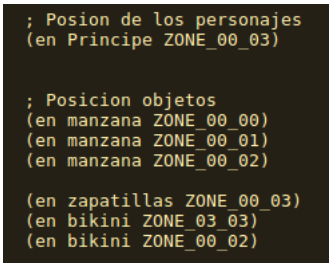


Figura 1.29: Estados iniciales para la comparativa de la eficacia de una mochila de tamaño n.

Se especifica en el enunciado del ejercicio que el plan tiene que contener menos acciones con un tamaño de mochila mayor, pero sin parametros de optimización realiza exactamente las misma acciones que con una mochila de tamaño 1, y al planificar con parámetros de optimización realiza más acciones cuando el tamaño de la mochila es mayor, pese a que el coste sea menor con un tamaño de mochila mayor. En base a esto solo puedo demostrar la eficacia de la mochila con un tamaño mayor comparando los costes. En los resultados obtenidos la mochila con tamaño 5 obtiene un coste de desplazamiento igual a 89, mientras que con tamaño 1 se ha obtenido 111 de coste. Los resultados pueden observarse en los archivos $plan\_mochila\_5.txt$ y $plan\_mochila\_1.txt$, situados en la carpeta $E5$, para mochilas de tamaño 5 y 1 respectivamente.

1.6. Ejercicio 6

En este ejercicio se definen 3 problemas con distintas capacidades de la mochila y con distinto número de objetos a entregar, de forma que el planificador encuentre un plan para ellos maximizando el número de puntos a conseguir.

Para este problema se ha definido la función (*entregado*) para definir como goal que se entreguen un número determinado de objetos y observar el comportamiento del planificador indicandole que minimize los puntos de entrega (ya que la métrica *maximize* no funciona), luego el resultado óptimo buscado es la entrega del objeto al personaje con menor puntuación. En un planteamiento inicial fuí simplificando el mapa de 4x4 interconectado, pero el planificador era incapaz de encontrar solución óptima. Así que he decidido definir un mapa muy sencillo ya que la más mínima complejidad hace que el planificador no encuentre solución. Para aplicar el nuevo goal he redefinido la acción $ENTREGAR$ para contabilizar el número de objetos entregados como se muestra a continuación:

```
(:action ENTREGAR
    :parameters (?j - jugador ?o - objeto ?z - zona ?p - personaje)
    :precondition (and (en ?j ?z) (en ?p ?z) (tiene ?j ?o))
    :effect (and (not (tiene ?j ?o)) (decrease (mano) 1)
        (increase (puntos-total) (puntos-entrega ?p ?o)) (increase (entregado) 1))
    )
```

Figura 1.30: Función *ENTREGAR* redefinida para contabilizar el número de objetos entregados.

El mapa es una columna de 3 casillas siendo la casilla central de terreno *AGUA*, en un extremo de la columna estan situados los 5 personajes, en otro extremo estan situados los 5 objetos distintos y un bikini, y la casilla central vacia. El jugador empieza en la casilla de los objetos. Los problemas definidos y los resultados obtenidos son los siguientes:

- Problema 1. Una mochila tamaño 5, siendo el goal entregar 5 objetos. Podemos observar la descripción del problema en la siguiente figura:

```
;Iniciales
(en zes ZONE_00_00)
(orientado NORTE)
(= (mano) 0)
(= (distancia-total) 0)
(= (mochila) 0)
(= (tam-mochila) 5)
(= (entregado) 0)

; Posion de los personajes
(en Principe ZONE_02_00)
(en princesa ZONE_02_00)
(en Leonardo ZONE_02_00)
(en Profesor ZONE_02_00)
(en Bruja ZONE_02_00)

; Posicion objetos
(en manzana ZONE_00_00)
(en oro ZONE_00_00)
(en oscar ZONE_00_00)
(en algoritmo ZONE_00_00)
(en rosa ZONE_00_00)
(en bikini ZONE_00_00)

(isZone ZONE_00_00 ARENOSO)
(isZone ZONE_01_00 AGUA)
(isZone ZONE_02_00 ARENOSO)
```

Figura 1.31: Descripción del problema 1 para optimización de entregas.

El plan obtenido sin optimización es el siguiente:

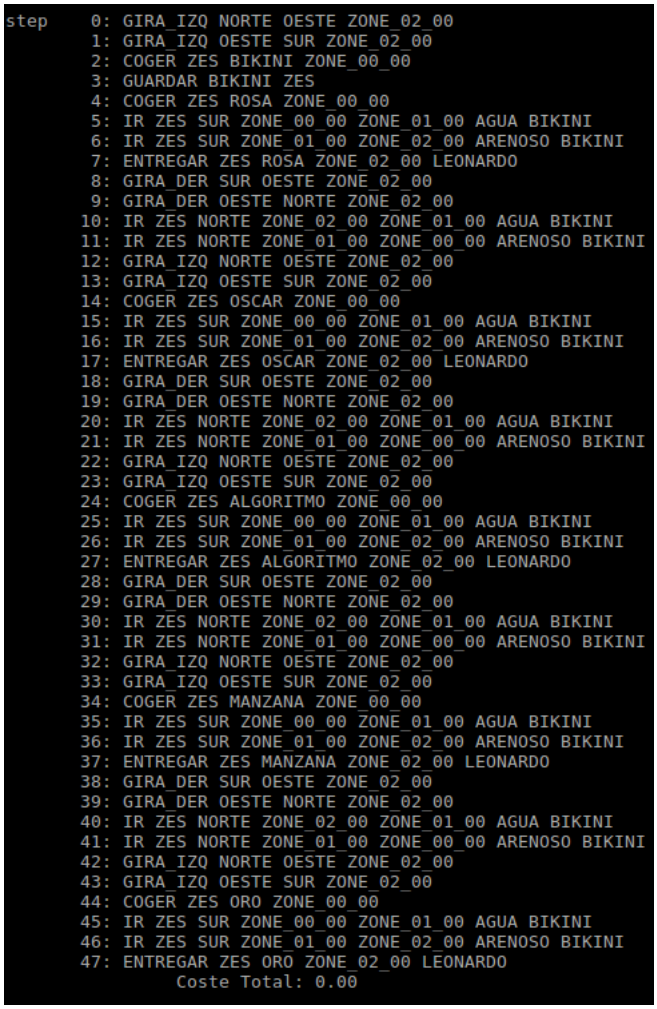


Figura 1.32: Plan problema 1 sin optimización.

Podemos observar que le entrega todos los objetos a leonardo, no alcanzando el resultado óptimo. Tampoco hace uso de la mochila, llevando los objetos uno a uno.

```
step    0: COGER ZES ORO ZONE_00_00
        1: GUARDAR ORO ZES
        2: COGER ZES MANZANA ZONE_00_00
        3: GUARDAR MANZANA ZES
        4: COGER ZES ALGORITMO ZONE_00_00
        5: GUARDAR ALGORITMO ZES
        6: COGER ZES OSCAR ZONE_00_00
        7: GUARDAR OSCAR ZES
        8: COGER ZES ROSA ZONE_00_00
        9: GUARDAR ROSA ZES
       10: COGER ZES BIKINI ZONE_00_00
       11: GIRA_DER NORTE ESTE ZONE_02_00
       12: GIRA_DER ESTE SUR ZONE_02_00
       13: IR ZES SUR ZONE_00_00 ZONE_01_00 AGUA BIKINI
       14: IR ZES SUR ZONE_01_00 ZONE_02_00 ARENOSO BIKINI
       15: SOLTAR ZES BIKINI ZONE_02_00
       16: SACAR ORO ZES
       17: ENTREGAR ZES ORO ZONE_02_00 PROFESOR
       18: SACAR MANZANA ZES
       19: GIRA_DER SUR OESTE ZONE_02_00
       20: ENTREGAR ZES MANZANA ZONE_02_00 PRINCESA
       21: SACAR ROSA ZES
       22: ENTREGAR ZES ROSA ZONE_02_00 LEONARDO
       23: SACAR OSCAR ZES
       24: ENTREGAR ZES OSCAR ZONE_02_00 PRINCIPE
       25: SACAR ALGORITMO ZES
       26: ENTREGAR ZES ALGORITMO ZONE_02_00 BRUJA
                Coste Total: 5.00
```

Figura 1.33: Plan problema 1 con optimización.

En el plan con optimización hace uso de toda la capacidad de la mochila y entrega todos los objetos al personaje que menor puntuación suma, alcanzando el resultado óptimo.

- Problema 2. Para este problema hemos definido la mochila de tamaño 1 y que entregue 1 objeto (con tamaño de mochila dos el planificador solo encuentra óptimo entregando 1 o 5 objetos). Los resultados obtenidos por el planificador son los siguientes:

```
step    0: GIRA_IZQ NORTE OESTE ZONE_02_00
        1: GIRA_IZQ OESTE SUR ZONE_02_00
        2: COGER ZES BIKINI ZONE_00_00
        3: GUARDAR BIKINI ZES
        4: COGER ZES ROSA ZONE_00_00
        5: IR ZES SUR ZONE_00_00 ZONE_01_00 AGUA BIKINI
        6: IR ZES SUR ZONE_01_00 ZONE_02_00 ARENOSO BIKINI
        7: ENTREGAR ZES ROSA ZONE_02_00 LEONARDO
                Coste Total: 0.00
```

Figura 1.34: Plan problema 2 sin optimización.

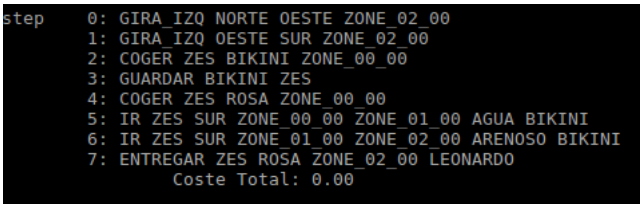


Figura 1.35: Plan problema 2 con optimización.

Puestos a analizar los resultados el planificador con o sin optimización siempre le entrega a la rosa a leonardo, asique si solo entrega un objeto obtiene el resultado óptimo en ambas planificaciones.

- Problema 3. Para éste problema hemos definido la mochila a tamaño 3 y que entregue 3 objetos. El plan sin optimización obtenido parece un plan razonable, mientras que el plan con optimización tiene 239 acciones, puede consultarse en el archivo *plan_problema3_optimo.txt* situado en la carpeta *E6*. El plan sin optimización es el siguiente:

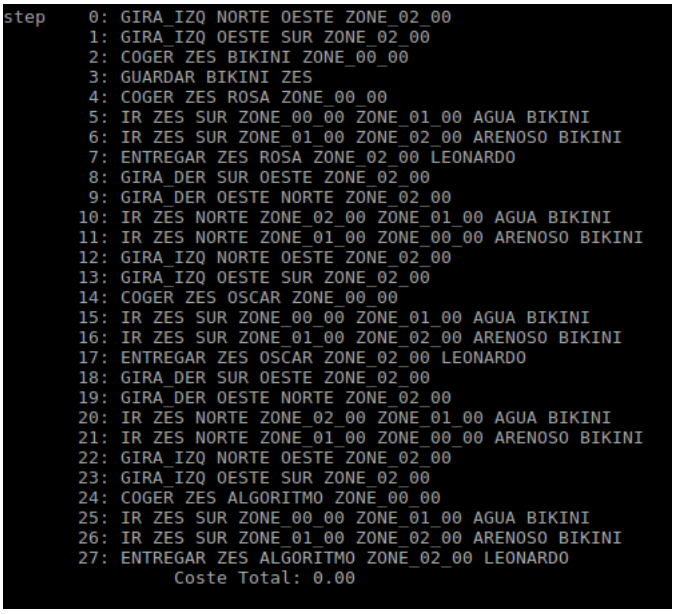


Figura 1.36: Plan problema 3 con optimización.

1.7. Inciso

El planificador parece funcionar correctamente en la minimización de los caminos, con una capacidad limitada, en cuanto el problema aumenta la complejidad no es capaz de encontrar solución óptima, quedando atrapado en mínimos locales o tiempos de cálculo extremadamente altos, pero cuando cambiamos a que minimice la puntuación de las

entregas funciona erroneamente y solo con problemas muy sencillos, suponiendo que mi código está escrito correctamente, esta es la conclusión a la que he llegado tras varias horas probando muchos mundos y revisando a mas no poder mi código e intentando diversas estrategias con varios compañeros más, que también han sufrido de estos problemas.

Referencias

[1] http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.