Práctica 2

Planificación Automática

"Modelado de un sistema de Planificación Automática que permita el control de un robot que gestione videollamadas entre pacientes y familiares"

Grado en Ingeniería Informática - Ingeniería de Conocimiento - 4º Curso
Diciembre 2022

100429073@alumnos.uc3m.es - Enrique Mateos Melero - 83 - EPS
100429025@alumnos.uc3m.es - Mario Maroto Jiménez - 83 - EPS

Grupo 3

Tabla de Contenidos

Introducción y Objetivo

Uno de los desafíos más importantes de la robótica social es el comportamiento inteligente en entornos estocácticos y dinámicos como es el caso de los entornos reales. En este caso, el robot será utilizado en una residencia atendiendo a ancianos donde se pueden dar muchos problemas como cancelaciones de citas o problemas de comunicación con los pacientes.

El objetivo de este trabajo es modelar un sistema de Planificación Automática que permita el control de un robot que gestione videollamadas entre pacientes y familiar. La planificación se basará en la visita por las salas de la residencia de un robot que vaya avisando al paciente que tiene una reserva acordada. Una vez haya avisado al paciente y haya realizado la llamada, acudirá al siguiente paciente o esperará en la base de carga hasta la siguiente reserva.

Este documento se organiza siguiendo las directrices de la práctica. En primer lugar se expondrá la implementación realizada con la justificación de los conceptos y reglas aplicados. Posteriormente, se explicará la forma de usar el sistema y cómo realizar pruebas con él. Seguidamente, se expondrán las pruebas realizadas, los resultados obtenidos y el análisis de los mismos. Finalmente, se plantearán las conclusiones derivadas de la realización de la práctica.

Manual Técnico

Este apartado pretende explicar en profundidad la implementación finalmente llevada a cabo, hablando del planificador y del dominio. Para dicha justificación, se ha usado un análisis del texto de la práctica y los cambios realizados al planteamiento inicial mientras se realizaba la práctica.

Planificador usado

Al comienzo del desarrollo de la práctica se tomó la decisión de usar *Fast Downard* como el planificador. Esta decisión se tomó sabiendo que se encontrarían dificultades en, al menos, dos aspectos:

- Imposibilidad de usar números: las funciones, usadas para implementar lo que son efectivamente variables numéricas, no están disponibles en *Fast Downward*.
- Forall limitado: *Fast Downard* no tiene soporte para usar *forall* como una precondición, lo que supuso un reto en varios puntos del desarrollo.

Pese a estas limitaciones, *Fast Forward* resultó más atractivo que otras alternativas como *Metric-FF* y *CBP* dada su habilidad para proporcionar soluciones óptimas. Además, al ser el planificador que se usó en otras tareas de clase, se tenía conocimiento previo sobre la salida de consola de *Fast Downward* el cual permitió acelerar el desarrollo.

Creación del dominio: Tipos

Se ha considerado que, únicamente, tres tipos son necesarios para la implementación del sistema de videollamadas:

- *location*: usado para representar los posibles lugares que el robot puede visitar
- *patient*: usado para identificar los diferentes pacientes a los que el robot puede asistir.
- *time*: usado para la creación de un sistema numérico debido a la falta de soporte para funciones en *Fast Downard.* Representa el tiempo del sistema.

Creación del dominio: Predicados

Aunque inicialmente se plantearon menos, un total de 14 predicados fueron creados para implementar el sistema satisfactoriamente:

- *(current-time ?curr_time - time)*
  Este predicado es usado para llevar una cuenta del tiempo actual. Solo debe haber una instancia en cualquier momento.

- *(next-time ?curr_time - time ?next_time - time)*
  La implementación del sistema numérico mencionado en el apartado de tipos. Asocia dos tiempos para obtener el siguiente dado un tiempo.

- *(robot-at ?loc - location)*
  Indica la localización del robot. Inicialmente se generalizó junto con la localización de los pacientes, pero se decidió separarlos debido a la complejidad añadida. Solo debe haber una instancia en cualquier momento.

- *(maybe-patient-at ?patient - patient ?loc - location)*
  Este predicado se usa para crear la "lista" de localizaciones donde se puede encontrar un paciente.

- *(call-reservation ?patient - patient ?start_time - time ?end_time - time)*
  Las reservas de llamada constan del paciente que llama, el tiempo en el que empieza la llamada, y el tiempo en el que termina. Indicar el tiempo en el que termina en lugar de cuanto tiempo dura la llamada simplifica la lógica.

- *(call-cancelled ?patient - patient ?time - time)*
  Indicador de que la llamada de un paciente ha sido cancelada junto con el tiempo de cancelación.

- *(search-time ?patient - patient ?time - time)*
  Este predicado indica el tiempo en el que un robot debe salir a buscar a un paciente de tal modo que llegue a la llamada tiempo.

- *(searching ?patient - patient)*
  Denota cuál es el paciente que está siendo buscado actualmente.

- *(occupied)*
  Este predicado marca que el robot está ocupado, ya sea buscando a un paciente, llamando, etc. Inicialmente, su funcionalidad era dada por *searching*, pero se convirtió en su propio predicado como solución a no poder usar *forall* en las precondiciones, entre otras cosas.

- *(stop-time)*
  Usado para bloquear el paso del tiempo genérico. Inicialmente este predicado no existía, dado que se estableció que visitar todas las habitaciones de cada paciente era una meta, pero la cancelación de llamadas hacía que esto no fuese realista.

- *(can-call ?patient)*
  Este predicado es un indicador de que el robot ha saludado al paciente y puede comenzar la llamada.

- *(calling ?patient - patient ?end_time - time)*
  Establece que un paciente está llamando y el tiempo en el que la llamada terminará.

- *(call-ended)*
  Teniendo un propósito similar a *stop-time,* siendo un mismo predicado inicialmente, pero *call-ended* también asegura que solo se pueda volver a base tras cancelar o terminar una llamada.

- *(completed ?patient - patient)*
  Usado para indicar la meta, este predicado sirve para establecer que un paciente ha completado su llamada, ya sea porque se ha terminado o porque se ha cancelado.

Estos predicados, en conjunción con las acciones que se detallarán a continuación, generan algunas limitaciones:

- Si se quiere ampliar el espacio temporal, es necesario crear las instancias de *time* y de *next-time* manualmente.
- *search-time* debe ser manualmente actualizado si se cambia el número de habitaciones donde puede estar un paciente.
- Cada paciente solo puede hacer una llamada, incluso si esta se cancela, debido a que *completed* solo puede tener una instancia por paciente

Creación del dominio: Acciones

Varias acciones han sido creadas para que, en conjunción con los predicados y los tipos, el robot pueda visitar las habitaciones y realizar las llamadas. En total hay 10 acciones.

Paso del tiempo

El tiempo actual debe ser actualizado cuando sea posible. Tres acciones están relacionadas con el paso del tiempo:

- *pass-time*: la acción principal para el paso del tiempo, aplicada cuando el tiempo no está parado y la llamada no acaba de terminar.
- *announce-patient*: esta acción debe pasar el tiempo ya que este se para cuando un paciente se está buscando. Será explicada en el apartado de Búsqueda del paciente.
- *greet-patient*: esta acción se encuentra en una posición idéntica a *announce-patient*. Debe pasar el tiempo ya que se para al saludar un paciente, y será explicada en el apartado de Realización de la llamada.

Las tres acciones emplean el mismo método para pasar el tiempo: determinar cuál es el tiempo siguiente usando los predicados *current-time* y *next-time*, eliminar la instancia del tiempo actual y crear otra con el tiempo nuevo.

Búsqueda del paciente

Antes de que el robot salga de la base de carga, se debe encontrar un paciente a partir del cual poder determinar qué localizaciones se han de visitar. Tres acciones están involucradas en este proceso:

- *search-patient*: el primer paso del proceso de búsqueda del paciente es determinar qué paciente buscar. Para ello, esta acción comprueba que el robot no esté ocupado y que se deba buscar a algún paciente en el tiempo actual. Si es así, se elimina el predicado *search-time* del paciente, el robot pasa a estar ocupado, buscando al paciente y finalmente se para el tiempo para evitar que el robot no se mueva a las habitaciones.
- *go-to-place*: una vez se determina el paciente a buscar, el robot debe visitar las posibles localizaciones de este. Para ello, si se está buscando a un paciente, el robot puede cambiar su localización por alguna otra determinada por el planificador. Es clave limitar el movimiento a cuando se esté buscando a un paciente ya que sino el planificador se anticipa y mueve al robot antes de buscar un agente.
- *announce-patient*: finalmente, en cada localización se debe anunciar que se busca al paciente. Si se está buscando a un paciente y el robot está en la misma localización que una de las posibles del paciente, se elimina esa localización de la lista de posibles localizaciones del paciente y se actualiza el tiempo.

go-to-place y *announce-patient* funcionan en armonía. Dado que el tiempo se para en *search-patient*, la única forma de que avance el tiempo es a través de *announce-patient*, por lo que *go-to-place* sólo llevará al robot a aquellas localizaciones que permitan que *announce-patient* sea parte del plan. Además, este sistema asegura que se visiten todas las localizaciones donde pueda estar el paciente.

Realización de la llamada

Una vez se han visitado todas las localizaciones posibles del paciente se debe realizar la llamada. Esta vez, cuatro acciones hacen esto posible:

- *greet-patient*: previo a la llamada, el robot debe saludar al paciente. Si el robot está buscando al paciente y está en la sala de vídeo, y el paciente tiene una reserva de llamada al tiempo siguiente, se establece que el paciente puede llamar y se actualiza el tiempo, el cuál sigue parado. En este caso, añadir paso del tiempo a *greet-patient* sirve para que aunque se cancele la llamada justo antes de empezar el saludo tenga lugar.
- *call-family*: una vez se establece que es el paciente puede llamar y ha llegado el tiempo de la reserva, el paciente pasa a estar llamando teniendo en cuenta el tiempo de terminar la llamada, se activa el paso del tiempo, se deja de buscar al paciente y se elimina la reserva.
- *end-call*: al llegar el tiempo de final de llamada, el paciente pasa a haber completado su llamada, es decir, se cumple la meta, y se marca la llamada como terminada.
- *cancel-call*: también es posible que la llamada no pueda comenzar ya que se ha cancelado. Esencialmente, esta acción condensa las 2 previas en una, reanudando el paso del tiempo, marcando la llamada como terminada y la meta del paciente como completada.

Siguiente paciente

Por último, el robot debe continuar tras terminar la llamada, lo cual se implementa a través de dos acciones:

- *return-to-base*: debido a que el robot no tiene ninguna motivación para volver a la base de carga (se asume que no necesita cargar), este movimiento se debe forzar. Si el robot está ocupado, la llamada ha terminado y no hay que buscar a otro paciente inmediatamente después, el robot deja de estar ocupado y vuelve a la base.
- *next-patient*: en el caso opuesto a *return-to-base*, si hay un paciente que buscar inmediatamente después de que termine la llamada, simplemente se marca al robot como no ocupado y no se cambia su posición.

Manual de Usuario

Dada la explicación de los detalles técnicos del proyecto, también es necesario explicar como hacer uso de los archivos y ejecutar las pruebas basándose en la jerarquía presente en el Anexo 1.

La distribución de los archivos está basada en la agrupación de las pruebas en subcarpetas, mientras que elementos únicos se mantienen en la carpeta raíz, de entre los cuales destaca *domain.pddl*, contiene los tipos, predicados y acciones necesarios, es decir, el

dominio, mientras que la carpeta *./problems/* contiene las 6 pruebas requeridas para la realización de la práctica.

Los archivos *problema-X.pddl* contienen los estados iniciales y metas que verifican el funcionamiento del dominio. Se crearon una serie de scripts para facilitar su ejecución:

- *run.sh*: este script fue usado para realizar pruebas generales durante el desarrollo de la práctica. Ejecuta el problema *problema.pddl* con el dominio *domain.pddl*, y escribe el plan obtenido en *salida-problema.pddl*.
- *run_i.sh*: este script permite ejecutar una prueba *X* elegida por el usuario de entre las disponibles en *./problems/* para mejorar el proceso de debugging de cada una de las pruebas. Tanto el plan como el SAS que resulta de la ejecución se almacenan en la carpeta *./outputs/salida-problema-X.txt y ./sas/sas-problema-X.txt* respectivamente.
- *run_all.sh*: este es un script que ejecuta todas las pruebas *./problems/problema-X.pddl* automáticamente con el fin de generar todos los planes y almacenarlos en la carpeta correspondiente, *./outputs/salida-problema-X.txt*. Los archivos SAS no son generados.

Cabe resaltar que los tres scripts asumen que el planificador *Fast Downward* se encuentra en la carpeta *../fast_downward/fast-downward.py*, por lo que deben ser modificados si el planificador se encuentra en otro lugar.

Pruebas realizadas

Las pruebas realizadas se han hecho intentando contemplar todas las posibilidades que se pueden dar en el sistema siempre y cuando tengan una solución. La descripción de las pruebas en orden respectivo son:

- *problema1.pddl*: Una sola llamada a un paciente para ver el funcionamiento básico del planificador.
- *problema2.pddl*: Dos llamadas, una exactamente después de la otra. Permite mostrar el funcionamiento del robot cuando no puede volver a la base de carga.
- *problema3.pddl*: Tres llamadas separadas entre ellas. Muestran el caso en el que el robot ha de volver a la base.
- *problema4.pddl*: Dos llamadas, la primera se cancela mientras el robot busca al paciente y la segunda antes de empezar la llamada. Hay tiempo entre las llamadas.
- *problema5.pddl*: Dos llamadas, la primera se cancela antes de empezar la llamada y empieza una de manera inmediata.
- *problema6.pddl*: Se aumenta la complejidad con cinco llamadas y una combinación de los casos mostrados en los problemas anteriores. Busqueda de la robustez del sistema.

Los planes obtenidos para cada uno de los problemas se muestran a continuación:

```
(pass-time t01 t02)
(pass-time t02 t03)
(pass-time t03 t04)
(search-patient p1 t04)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t04 t05)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t05 t06)
(go-to-place p1 room2 room3)
(announce-patient p1 room3 t06 t07)
(go-to-place p1 room3 video-room)
(greet-patient p1 t07 t08 t12)
(call-family p1 t08 t12)
(pass-time t08 t09)
(pass-time t09 t10)
(pass-time t10 t11)
(pass-time t11 t12)
(end-call p1 t12)
cost = 18 (unit cost)
```

Figura 1: Plan del *problema1.pddl*

```
(pass-time t01 t02)
(pass-time t02 t03)
(pass-time t03 t04)
(search-patient p1 t04)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t04 t05)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t05 t06)
(go-to-place p1 room2 video-room)
(greet-patient p1 t06 t07 t10)
(call-family p1 t07 t10)
(pass-time t07 t08)
(pass-time t08 t09)
(pass-time t09 t10)
(end-call p1 t10)
(next-patient p2 t10)
(search-patient p2 t10)
(go-to-place p2 video-room room2)
(announce-patient p2 room2 t10 t11)
(go-to-place p2 room2 room3)
(announce-patient p2 room3 t11 t12)
(go-to-place p2 room3 room4)
(announce-patient p2 room4 t12 t13)
(go-to-place p2 room4 video-room)
(greet-patient p2 t13 t14 t20)
(call-family p2 t14 t20)
(pass-time t14 t15)
(pass-time t15 t16)
(pass-time t16 t17)
(pass-time t17 t18)
(pass-time t18 t19)
(pass-time t19 t20)
(end-call p2 t20)
cost = 33 (unit cost)
```

Figura 2: Plan del *problema2.pddl*

```
(pass-time t01 t02)
(pass-time t02 t03)
(pass-time t03 t04)
(search-patient p1 t04)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t04 t05)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t05 t06)
(go-to-place p1 room2 video-room)
(greet-patient p1 t06 t07 t10)
(call-family p1 t07 t10)
(pass-time t07 t08)
(pass-time t08 t09)
(pass-time t09 t10)
(end-call p1 t10)
(return-to-base p1 t10 video-room)
(pass-time t10 t11)
(pass-time t11 t12)
(search-patient p2 t12)
(go-to-place p2 charging-base room2)
(announce-patient p2 room2 t12 t13)
(go-to-place p2 room2 room3)
(announce-patient p2 room3 t13 t14)
(go-to-place p2 room3 room4)
(announce-patient p2 room4 t14 t15)
(go-to-place p2 room4 video-room)
(greet-patient p2 t15 t16 t20)
(call-family p2 t16 t20)
(pass-time t16 t17)
(pass-time t17 t18)
(pass-time t18 t19)
(pass-time t19 t20)
(end-call p2 t20)
(return-to-base p1 t20 video-room)
(pass-time t20 t21)
(pass-time t21 t22)
(pass-time t22 t23)
(pass-time t23 t24)
(pass-time t24 t25)
(pass-time t25 t26)
(pass-time t26 t27)
(pass-time t27 t28)
(search-patient p3 t28)
(go-to-place p3 charging-base room3)
(announce-patient p3 room3 t28 t29)
(go-to-place p3 room3 video-room)
(greet-patient p3 t29 t30 t35)
(call-family p3 t30 t35)
(pass-time t30 t31)
(pass-time t31 t32)
(pass-time t32 t33)
(pass-time t33 t34)
(pass-time t34 t35)
(end-call p3 t35)
cost = 54 (unit cost)
```

Figura 3: Plan del *problema3.pddl*

```
(pass-time t01 t02)
(search-patient p1 t02)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t02 t03)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t03 t04)
(go-to-place p1 room2 room5)
(announce-patient p1 room5 t04 t05)
(go-to-place p1 room5 video-room)
(greet-patient p1 t05 t06 t10)
(cancel-call p1 t06 t06 t10)
(next-patient p2 t06)
(search-patient p2 t06)
(go-to-place p2 video-room room2)
```

```
(announce-patient p2 room2 t06 t07)
(go-to-place p2 room2 room5)
(announce-patient p2 room5 t07 t08)
(go-to-place p2 room5 video-room)
(greet-patient p2 t08 t09 t15)
(call-family p2 t09 t15)
(pass-time t09 t10)
(pass-time t10 t11)
(pass-time t11 t12)
(pass-time t12 t13)
(pass-time t13 t14)
(pass-time t14 t15)
(end-call p2 t15)
cost = 27 (unit cost)
```

Figura 4: Plan del *problema4.pddl*

```
(pass-time t01 t02)
(search-patient p1 t02)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t02 t03)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t03 t04)
(go-to-place p1 room2 room5)
(announce-patient p1 room5 t04 t05)
(go-to-place p1 room5 video-room)
(greet-patient p1 t05 t06 t10)
(cancel-call p1 t06 t06 t10)
(next-patient p2 t06)
(search-patient p2 t06)
(go-to-place p2 video-room room2)
(announce-patient p2 room2 t06 t07)
(go-to-place p2 room2 room5)
(announce-patient p2 room5 t07 t08)
(go-to-place p2 room5 video-room)
(greet-patient p2 t08 t09 t15)
(call-family p2 t09 t15)
(pass-time t09 t10)
(pass-time t10 t11)
(pass-time t11 t12)
(pass-time t12 t13)
(pass-time t13 t14)
(pass-time t14 t15)
(end-call p2 t15)
cost = 27 (unit cost)
```

Figura 5: Plan del *problema5.pddl*

```
(pass-time t01 t02)
(pass-time t02 t03)
(pass-time t03 t04)
(search-patient p1 t04)
(go-to-place p1 charging-base room1)
(announce-patient p1 room1 t04 t05)
(go-to-place p1 room1 room2)
(announce-patient p1 room2 t05 t06)
(cancel-call p1 t06 t07 t10)
(return-to-base p1 t06 room2)
(pass-time t06 t07)
(pass-time t07 t08)
(pass-time t08 t09)
(pass-time t09 t10)
(pass-time t10 t11)
(pass-time t11 t12)
(search-patient p2 t12)
(go-to-place p2 charging-base room2)
(announce-patient p2 room2 t12 t13)
(go-to-place p2 room2 room3)
(announce-patient p2 room3 t13 t14)
(go-to-place p2 room3 room4)
(announce-patient p2 room4 t14 t15)
(go-to-place p2 room4 video-room)
(greet-patient p2 t15 t16 t20)
(cancel-call p2 t16 t16 t20)
(return-to-base p1 t16 video-room)
(pass-time t16 t17)
(search-patient p5 t17)
(go-to-place p5 charging-base room4)
(announce-patient p5 room4 t17 t18)
(go-to-place p5 room4 room5)
(announce-patient p5 room5 t18 t19)
(go-to-place p5 room5 video-room)
(greet-patient p5 t19 t20 t26)
(call-family p5 t20 t26)
(pass-time t20 t21)
(pass-time t21 t22)
(pass-time t22 t23)
(pass-time t23 t24)
(pass-time t24 t25)
(pass-time t25 t26)
(end-call p5 t26)
(return-to-base p1 t26 video-room)
(pass-time t26 t27)
(pass-time t27 t28)
(search-patient p3 t28)
(go-to-place p3 charging-base room3)
(announce-patient p3 room3 t28 t29)
(go-to-place p3 room3 video-room)
(greet-patient p3 t29 t30 t35)
(call-family p3 t30 t35)
(pass-time t30 t31)
(pass-time t31 t32)
(pass-time t32 t33)
(pass-time t33 t34)
(pass-time t34 t35)
(end-call p3 t35)
(next-patient p4 t35)
(search-patient p4 t35)
(go-to-place p4 video-room room1)
(announce-patient p4 room1 t35 t36)
(go-to-place p4 room1 room3)
(announce-patient p4 room3 t36 t37)
(go-to-place p4 room3 room4)
(announce-patient p4 room4 t37 t38)
(go-to-place p4 room4 room5)
(announce-patient p4 room5 t38 t39)
(go-to-place p4 room5 video-room)
(greet-patient p4 t39 t40 t43)
(call-family p4 t40 t43)
(pass-time t40 t41)
(pass-time t41 t42)
(pass-time t42 t43)
(end-call p4 t43)
cost = 75 (unit cost)
```

Figura 6: Plan del *problema6.pddl*

Análisis de los resultados

Como se puede observar, todos los planes son óptimos, además de por ser ejecutados con *Fast Downward*, porque solo va a las habitaciones cuando tiene que hacerlo, solo anuncia una vez por habitación y cancela las llamadas en los momentos correctos y evita realizar la llamada con el paciente.

Esto se puede observar, por ejemplo, en los distintos funcionamientos de los planes 2 y 3. En el caso del primero, una vez ha terminado la llamada no vuelve a base porque supondría una acción más y continúa directamente con el siguiente paciente que tiene una llamada justo después. En el caso del segundo, si decide volver a base ya que no tiene nada que hacer por el momento.

A pesar de que todos los problemas están definidos con el mismo espacio de tiempo, de 1 a 45, todos terminan su planificación si han conseguido lograr los objetivos, es decir, realizar o cancelar las llamadas de los pacientes.

También se puede observar que las únicas acciones repetidas son las del paso del tiempo pero no son relevantes para el plan en el sentido de que supongan un cambio en el entorno (moverse, llamar, anunciar…). Es por ello, que se podrían agrupar en espacios de tiempo de distinta longitud y de esa manera obtener planes más cortos.

Conclusiones

Se han obtenido una serie de conocimientos derivados de la realización de la práctica. Se ha observado que para que el planificador haga ciertas acciones que el programador quiere que haga, es necesario o bien incluirlas en las metas o que sea la única forma que haya de conseguir un predicado necesario para cumplir el objetivo.

Ejemplo de esto es el caso de visitar las salas, ya que al principio se realizaba la primera opción, pues de otra manera, el mejor plan era no visitar las salas e ir directamente a la de videollamadas. Aplicando la segunda, mediante cortar el paso del tiempo durante el movimiento del robot, se conseguía el mismo objetivo sin incluir los sub-objetivos de visitar las salas.

Por último, destacar que vuelve a ser una manera distinta de trabajar a la que estamos acostumbrados. Incluso ya nos habíamos acostumbrado a pensar en el formato declarativo de CLIPS. En este caso es mucho menos directo que CLIPS ya que solo se basa en determinar un estado inicial y una serie de posibles acciones y permitir que busque un plan. Sin embargo, esta práctica nos ha permitido comprender esta nueva forma de trabajo.

Anexo 1

```
┌─ IC_P2
├──── README.md
├──── run.sh
├──── run_i.sh
├──── run_all.sh
├──── domain.pddl
├──── problema.pddl
├──── salida-problema.txt
│
├─┬── problems
│ └──── problema-[1-6].pddl
│
├─┬── outputs
│ └──── salida-problema-[1-6].txt
│
└─┬── sas
  └──── sas-problema-[1-6].txt
```

Anexo 2

Todo el proyecto se encuentra en: https://github.com/enrique-soft-dev/IC_P2