



## Tarea 2

# Programación Lógica para el Razonamiento

Fecha de entrega: jueves 10 de abril a las 23:59 hrs

---

### Aspectos generales

#### Formato y plazo de entrega

El formato de entrega son archivos con extensión .lp con un PDF para las respuestas teóricas. El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el jueves 10 de abril a las 23:59 hrs. Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas. Por último, recuerda que los cupones de atraso son días **no hábiles** extra.

#### Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo hecho en esta tarea debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente a través de las [issues en GitHub](#).

Por otra parte, sabemos que estás utilizando material hecho por otras personas, por lo que es importante reconocerlo de la forma apropiada. Todo lo que obtengas de internet debes citarlo de forma correcta (ya sea en APA, ICONTEC o IEEE). Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

#### Comentarios adicionales

El objetivo de esta tarea es que puedan desarrollar la capacidad de modelar problemas a partir del lenguaje natural y resolverlos utilizando ASP en Clingo. Es fundamental que pongan énfasis en las justificaciones de sus respuestas, cuidando la redacción, ortografía; manteniendo el código ordenado y comentado. Aquellas respuestas que solo presenten resultados o código (sin contexto ni comentarios) no serán consideradas, mientras que tareas desordenadas pueden ser objeto de descuentos.

Se recomienda fuertemente apoyarse de los [apuntes del curso](#) para el desarrollo de la tarea, además de la bibliografía oficial del curso. En particular el uso de Clingo y modelación de problemas.

## 1. Reflexión y Teoría (1 pts.)

Actualmente existen diversos podcasts disponibles en YouTube que discuten la teoría y aplicaciones de la IA. Dos de ellos son *Machine Learning Street Talk* y el podcast de Lex Fridman. En esta pregunta te ayudaremos a familiarizarte con este tipo de contenido en el contexto de nuestro curso.

1. (0,5 puntos) El tema de la conciencia en la Inteligencia Artificial ha ocasionado debate desde los inicios del área. Ve el extracto de la entrevista (click [aquí](#)) del podcast *Machine Learning Street Talk* a François Chollet a principios de este año. Escucha entre 58:30 y 1:01:15 (aunque recomendamos seguir escuchando al menos 3 minutos más!). Escribe de la mejor forma que puedas cuál es la manera en que Chollet define al Sistema 2 como intrínsecamente ligado a la conciencia. ¿Es la definición de Chollet convincente para ti? Justifica. Ahora, toma una definición de conciencia entregada por la psicología, la filosofía o la espiritualidad (citando fuentes) y contrástala con la de Chollet. ¿Consideras que es posible que las máquinas adquieran ese tipo de conciencia? Escribe tu respuesta en no más de media página<sup>1</sup>.



2. (0,5 puntos) Ve ahora la entrevista de Lex Fridman a Melanie Mitchell (click [aquí](#)) específicamente la discusión respecto de los *conceptos*. Da un ejemplo de un concepto que se pueda representar en ASP. Especula sobre qué sistema computacional sería necesario implementar para permitir que un sistema de programación en lógica realice analogías; da un ejemplo concreto. Escribe tu respuesta en no más de media página.



**Qué entregar en esta parte**

En un subdirectorío Parte1/ un archivo `respuesta.pdf` con tu respuesta.

---

<sup>1</sup>Escrita a 11pt, fuente Arial/Times/Computer Modern. Recomendamos (no obligamos!) usar L<sup>A</sup>T<sub>E</sub>X para el informe.

## 2. DCCesafíos de la Ingeniería (2.5 puntos)

En el curso ING1004 Desafíos de la Ingeniería, que se dicta cada semestre, es necesario formar equipos a partir de un gran número de estudiantes, cumpliendo con diversas restricciones. Dichas restricciones tienen que ver con que no se quiere configurar equipos en que estén “solos” estudiantes de algunas cohortes minoritarias. Específicamente, no pueden haber equipos que tengan una mujer sola, que tengan un estudiante de una vía de admisión especial solo (p.ej., talento e inclusión) o que tengan un solo estudiante proveniente de una región fuera de la metropolitana. Adicionalmente, no se quiere incluir en un equipo a un par de estudiantes que provengan del mismo colegio. Como podrás imaginar, esta tarea resulta tediosa y compleja si se realiza manualmente.

Se te ocurre que el problema puede resolverse sencilla y eficientemente aplicando los conceptos de *Answer Set Programming*. Por eficiencia, considerando los avances en inteligencia artificial, decides probar Deepseek para evaluar si es capaz de generar un código desde el cual comenzar a trabajar. En el prompt, le explicas a Deepseek que toda la información de los estudiantes está descrita en un predicado `info`, de tal forma que si el programa contiene un hecho de la forma:

```
info(S, Sec, G, H, R, A).
```

se indica la existencia de un estudiante cuyo *id* es *S*, en la sección *Sec*, con genero  $G \in \{m, f\}$ , que proviene del colegio de *id* *H*, de la región  $R \in \{rm, otra\}$ , y donde la vía de admisión es *A*.

A continuación, se presenta el código que te entregó Deepseek.<sup>2</sup>

```
student(S)      :- info(S, _, _, _, _, _).
section(S, Sec) :- info(S, Sec, _, _, _, _).
gender(S, G)    :- info(S, _, G, _, _, _).
high_school(S, H) :- info(S, _, _, H, _, _).
region(S, R)    :- info(S, _, _, _, R, _).
admission(S, A) :- info(S, _, _, _, _, A).
admission_type(A) :- admission(_, A).
team(T)         :- section_of_team(T, _).

{ assign(S, T) : section_of_team(T, Sec) } = 1 :- student(S), section(S, Sec).

:- team(T), #count { S : assign(S, T), gender(S, f) } = 1.

:- team(T), #count { S : assign(S, T), region(S, otra) } = 1.

:- team(T), admission_type(A), #count { S : assign(S, T), admission(S, A) } = 1.

:- team(T), assign(S1, T), assign(S2, T), high_school(S1, H1), high_school(S2, H2), S1 !=
   ↪ S2.

size_of_team(T, Size) :- team(T), #count {S : assign(S,T)} = Size.
```

---

<sup>2</sup>Para desarrollar esta pregunta, es fundamental que comprendas el uso del operador `#count`. Se recomienda revisar el libro para familiarizarte con su funcionamiento.

```
:- team(T), size_of_team(T, Size), Size>8.
:- team(T), size_of_team(T, Size), Size<7.

#show assign/2.
```

Los archivos proporcionados para resolver el problema son los siguientes:

- `assign_teams.lp`: Archivo que el código generado por Deepseek para la formación de equipos.
- `sections.lp`: Archivo que contiene las secciones del curso Desafíos de la Ingeniería y los números de los equipos asignados a cada una. No debes modificar este archivo.
- `students.lp`: Archivo que contiene la lista de estudiantes del curso junto con su información relevante. No debes modificar este archivo.

## 2.1. *Debugging* del código generado por IA (0.75 puntos)

Al analizar la solución generada por Deepseek, sospechas de la corrección de algunas reglas y del enfoque en general dado que el programa te entrega configuraciones no factibles de equipos.

Como aún deseas explorar el potencial de la inteligencia artificial, decides que la mejor forma de *debuggear* el código es consultando a Chat GPT sobre los posibles errores. Tras tu solicitud, la IA te proporciona la siguiente respuesta:

### Resumen de Problemas y Soluciones

A continuación se presenta un resumen de los problemas identificados en el código ASP, junto con las respectivas soluciones propuestas.

#### 1. Restricción de tipo de admisión

**Problema:** La regla

```
:- team(T), admission_type(A),
   #count { S : assign(S, T), admission(S, A) } = 1.
```

impide que en un equipo haya exactamente un estudiante de cierto tipo de admisión, lo que puede dificultar la formación de equipos si existen pocos estudiantes de ese tipo.

**Solución:**

- Si se desea permitir solo cero o al menos dos estudiantes de un tipo de admisión, se puede mantener la regla.
- Si se prefiere penalizar pero no prohibir esta situación, se puede usar una *weak constraint*:

```
:~ team(T), #count { S : assign(S, T), admission(S, A) } = 1. [1@2, T]
```

## 2. Restricción de colegios

**Problema:** La siguiente regla prohíbe que *cualquier par* de estudiantes esté en el mismo equipo, sin verificar que provengan del mismo colegio.

```
:- team(T), assign(S1, T), assign(S2, T),  
   high_school(S1, H1), high_school(S2, H2), S1 != S2.
```

**Solución:** Verificar que los estudiantes pertenezcan al mismo colegio:

```
:- team(T), assign(S1, T), assign(S2, T),  
   high_school(S1, H), high_school(S2, H), S1 != S2.
```

## 3. Restricción del tamaño del equipo

**Problema:** Las restricciones

```
:- team(T), size_of_team(T, Size), Size > 8.  
:- team(T), size_of_team(T, Size), Size < 7.
```

exigen que cada equipo tenga entre 7 y 8 miembros, lo que puede resultar en la ausencia de soluciones.

**Solución:** Utilizar *weak constraints* para penalizar tamaños fuera del rango en lugar de prohibirlos:

```
:~ team(T), size_of_team(T, Size), Size > 8. [Size-8@1, T]  
:~ team(T), size_of_team(T, Size), Size < 7. [7-Size@1, T]
```

## 4. Definición de equipos

**Problema:** La regla

```
team(T) :- section_of_team(T, _).
```

puede no generar todos los equipos, especialmente si hay equipos que solo se generan al asignar estudiantes.

**Solución:** Asegurar que se reconozca como equipo cualquier T en el que se asigne al menos un estudiante:

```
team(T) :- section_of_team(T, _).  
team(T) :- assign(_, T).
```

## 5. Balance de género en equipos con muchas mujeres

**Problema:** Si existen demasiadas mujeres, la regla anterior no evita que se formen equipos compuestos únicamente por mujeres.

**Solución:** Para asegurar equipos mixtos, se pueden agregar restricciones que prohíban equipos completamente unisex:

```
:- team(T), #count { S : assign(S, T), gender(S, f) } = Size,  
   size_of_team(T, Size).  
:- team(T), #count { S : assign(S, T), gender(S, m) } = Size,  
   size_of_team(T, Size).
```

O, si se desea limitar el número máximo de mujeres por equipo:

```
:- team(T), #count { S : assign(S, T), gender(S, f) } > X.
```

Por ejemplo, para no tener más de 5 mujeres:

```
:- team(T), #count { S : assign(S, T), gender(S, f) } > 5.
```

## Conclusión

El código presenta restricciones que pueden resultar demasiado rígidas en función de la distribución de estudiantes, tipos de admisión y género. Se proponen modificaciones para:

- Verificar condiciones específicas (por ejemplo, misma escuela).
- Utilizar *weak constraints* para penalizar en lugar de prohibir configuraciones.
- Garantizar que la definición de equipos cubra todos los casos.
- Ajustar la distribución de género para evitar equipos unisex no deseados.

Como primera actividad, revisa la respuesta proporcionada y analiza cada uno de los siete problemas identificados. Para cada caso, determina si realmente representa un error en la formulación del problema. Si consideras que un problema debe ser corregido en el código, evalúa si la solución propuesta por Chat GPT es adecuada y justifica. En caso afirmativo, implementa la corrección correspondiente. En caso que consideres que hay una mejor solución, menciónalo y justifica. Tus respuestas deben ser concisas, con un máximo de cinco líneas por problema.

Para redactar tu respuesta, considera el contexto real de la universidad y la carrera de ingeniería al analizar los problema sugeridos.

## 2.2. Documentación de código generado por IA (0.75 puntos)

Después de identificar y corregir todos los errores en el código, decides entregarlo a la universidad como una herramienta útil para las futuras generaciones. No obstante, consideras que podrían surgir preguntas sobre su funcionamiento.

Por esta razón, determinas que lo más adecuado es documentarlo. De este modo, no solo estarás preparado para aclarar cualquier interrogante, sino que también garantizarás que otros usuarios puedan comprender y utilizar el código de manera efectiva.

En esta actividad, deberás llevar a cabo lo siguiente:

1. Documenta el código que depuraste en la etapa anterior, proporcionando para cada restricción una breve descripción (de no más de dos líneas) sobre su función y la razón por la que opera correctamente. Como se discutió en clases, la explicación de las reglas puede ser ‘literal’ o precisa y en buen castellano. Apunta a lo segundo.
2. Solicita a ChatGPT que realice la misma tarea y adjunta su respuesta.
3. Compara tu documentación con la generada por ChatGPT y reflexiona sobre las diferencias y similitudes, señalando qué elementos de cada versión podrían mejorarse y cuál consideras más efectiva para futuros usuarios.

## 2.3. Expansión y mejora de código generado por IA (0.5 puntos)

El código ahora funciona correctamente y cumple con el objetivo establecido. Sin embargo, los profesores han decidido añadir una nueva restricción para la formación de equipos y te piden que la integres en el código.

Para seguir evaluando las capacidades de ChatGPT, le solicitas que incorpore la siguiente restricción:

*Para promover la heterogeneidad de los equipos, no puede haber un equipo con 4 o más mujeres al mismo tiempo que existen equipos con 0 mujeres.*

En esta actividad, deberás llevar a cabo lo siguiente:

1. En un archivo nuevo llamado `p1_3_alumno.lp` copia el código anterior e implementa la restricción solicitada. Documenta tu solución proporcionando una breve descripción (de no más de dos líneas) sobre su función y la razón por la que opera correctamente. Verifica que funciona correctamente.
2. En un archivo nuevo llamado `p1_3_gpt.lp` copia el código anterior y solicita a alguna IA que realice la misma tarea y adjunta su respuesta. Verifica que funciona correctamente y explica por qué.
3. Compara tu solución con la generada por ChatGPT y reflexiona sobre las diferencias y similitudes, señalando qué elementos de cada versión podrían mejorarse.

## 2.4. Reflexión y conclusiones (0.5 puntos)

Al desarrollar esta actividad, habrás notado que, a pesar de los grandes avances en inteligencia artificial, esta tecnología aún puede cometer errores. Por ello, es fundamental utilizarla con criterio y no depender ciegamente de sus resultados.

A continuación, plantea qué estrategia utilizarías para verificar la precisión y fiabilidad de los resultados y códigos generados por un programa de inteligencia artificial al momento de utilizarlo para generación, *debugging* y expansión de código.

Luego, reflexiona acerca del uso de estas tecnologías en el ámbito de tu educación, y de las consecuencias de utilizarla sin seguir la estrategia planteada anteriormente.

### Qué entregar en esta parte

En un subdirectorío DCCesafios/:

- Un archivo `respuesta.pdf` con tus respuestas ordenadas.
- Para la parte 1 crea un archivo `p1_1.lp` con su respectiva respuesta y para la 3 crea los archivos `p1_3_alumno.lp` y `p1_3_gpt.lp` especificados anteriormente.



### 3. DCC Bodega (2.5 puntos)

#### 3.1. Introducción

El DCC está construyendo un nuevo edificio en donde ubicarán todos los computadores que tienen para entrenar modelos de Inteligencia Artificial. Debido a lo anterior, el profesor Jorge Baier ha decidido implementar un sistema para ordenar las bodegas durante la construcción. Para esto, se inspiró en la empresa *Kiva Systems*, la cual es conocida por tener un sistema de automatización de empaque bastante impresionante, que se puede ver acá:

<https://www.youtube.com/watch?v=3UxZDJ1HiPE>

Debido a que eres un especialista en ASP, el profesor ha decidido pedirte ayuda para construir un programa en Clingo capaz de hacer razonamiento sobre un ambiente simulado inspirado en los problemas que resuelve Kiva Systems.

Para evaluar rápidamente la factibilidad de la solución, deberás implementar un programa que considere lo siguiente:

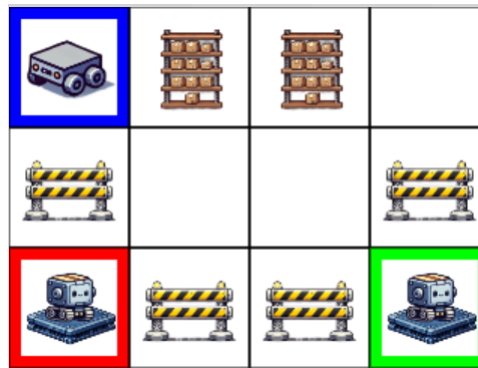


Figura 1: Imagen de la grilla

1. Representaremos la bodega como una grilla de  $N \times M$  celdas.
2. En la bodega pueden encontrarse  $X$  robots, junto con  $Y$  estantes. Cada uno de estos estantes puede llevar  $Z$  productos. Puedes considerar que estos estantes llevan un número ilimitado de unidades de estos productos, es decir, no se les acaban las unidades.



(a) Imagen de un robot



(b) Imagen de un estante

Figura 2: Componentes del problema

3. Un robot puede cargar estanterías, pero solo una a la vez. Asimismo, una estantería solo puede ser cargada por un robot a la vez.
4. Pueden haber  $P$  estaciones de descarga (donde se arman los paquetes), donde cada uno de estas puede necesitar una lista de  $K$  productos distintos. Puedes considerar que solo necesita una unidad de estos productos, es decir, solo se necesita que se descargue una vez cada producto.

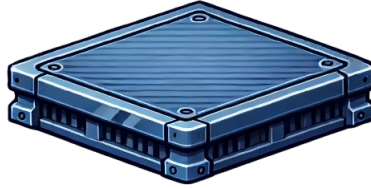
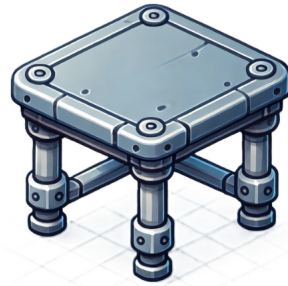


Figura 3: Imagen de una estación de descarga

5. La bodega tiene 2 tipos de obstáculos. El primero de ellos son paredes, las que impiden el paso de los robots. El segundo obstáculo son mesas, por las cuales los robots pueden pasar por debajo **solo** si no está cargando un estante.



(a) Imagen de una pared



(b) Imagen de una mesa

Figura 4: Obstáculos del problema

6. El objetivo del problema es que cada estación de descarga obtenga los productos solicitados en sus respectivas listas de manera óptima (es decir, en el menor tiempo posible).

A continuación se describen las acciones posibles que cada robot puede ejecutar en una unidad de tiempo:

1. **action(up)**: incrementa la coordenada  $y$  en la que se encuentra el robot. Es ejecutable si la celda de destino se encuentra desocupada o si, no teniendo el robot un estante encima, contiene una mesa.
2. **action(down)**: decrementa la coordenada  $y$  en la que se encuentra el robot. Es ejecutable si la celda de destino se encuentra desocupada o si, no teniendo el robot un estante encima, contiene una mesa.
3. **action(left)**: decrementa la coordenada  $x$  en la que se encuentra el robot. Es ejecutable si la celda de destino se encuentra desocupada o si, no teniendo el robot un estante encima, contiene una mesa.

4. `action(right)`: incrementa la coordenada  $x$  en la que se encuentra el robot. Es ejecutable si la celda de destino se encuentra desocupada o si, no teniendo el robot un estante encima, contiene una mesa.
5. `action(wait)`: el robot no se mueve.
6. `action(liftShelf)`: el robot levanta una estantería. Es ejecutable solo si el robot está ubicado debajo de un estante y el estante no está siendo levantado ya por otro robot.
7. `action(dropShelf)`: el robot deja una estantería en la posición actual. Es ejecutable solo si el robot tiene un estante encima.

Asimismo, para definir un problema específico, se utilizarán los siguientes predicados estáticos.

1. `rangeX(0..ancho)`: define las coordenadas  $X$  que componen la grilla.
2. `rangeY(0..alto)`: define las coordenadas  $Y$  que componen la grilla.
3. `robot(0..n)`: define la cantidad de robots que hay, con IDs  $0, 1, \dots, n$ .
4. `object(0..n)`: define la cantidad de productos distintos que existen.
5. `shelf(0..n)`: define los IDs, entre  $0$  y  $n$ , de estantes en el mapa.
6. `robotOn(A,B,C,0)`: define en qué ubicación parte el robot  $A$ , siendo  $B$  la coordenada  $X$  y  $C$  la coordenada  $Y$  en el tiempo  $0$ .
7. `shelfOn(A,B,C,0)`: define en qué ubicación parte el estante  $A$ , siendo  $B$  la coordenada  $X$  y  $C$  la coordenada  $Y$  en el tiempo  $0$ .
8. `goalOn(A,B,C)`: define que el objetivo  $A$  está en  $(B,C)$ . No tiene tiempo debido a que no se puede mover.
9. `wallOn(A,B)`: define que hay una pared en  $(A,B)$ , por donde no puede pasar ni el robot ni un estante.
10. `tableOn(A,B)`: define que hay una mesa en  $(A,B)$ , por donde un robot puede pasar solo si no tiene un estante encima.
11. `shelfHas(A,B)`: define que el estante con ID  $A$  tiene el objeto con ID  $B$ .
12. `goalWants(A,B)` dice que el objetivo con ID  $A$  quiere el objeto con ID  $B$ .
13. **(Para el bonus)** `diagRobotDefined(A)`: hace que el robot con ID  $A$  se convierta en un robot que sólo puede moverse diagonalmente.

### 3.2. Actividades

Dicho esto, tu tarea será implementar un programa que sea capaz de hacer lo siguiente:

#### 1. Acciones (0.8 puntos)

- Los robots deben poder moverse en las 4 direcciones cardinales. Asimismo, el robot puede decidir no moverse.
- Los robots deben poder tomar y soltar estantes si es válido.
- Cuando los robots se mueven y están llevando un estante encima suyo, el estante debe moverse junto a ellos.

- Las acciones toman un tiempo  $T = 1$  en realizarse.

## 2. Restricciones (1 punto)

- El robot debe mantenerse dentro del rango del tablero.
- El robot no puede superponerse a otros robots (ni cruzarse).
- Los estantes no pueden superponerse entre sí
- El robot no puede cruzar paredes.
- Un estante no puede estar en la misma altura que una mesa, es decir, un estante no puede cruzar una mesa.
- El programa debe encontrar una solución dentro del bound definido dentro de cada test.

## 3. Objetivos (0.7 puntos)

- Los pedidos de los objetivos tienen que ser cumplidos, es decir, cada lista de pedidos de cada estación de descarga debe ser completada.
- Para que un producto de la lista de pedido sea marcado como entregado, un estante con el producto correspondiente debe estar, en algún momento, encima de la respectiva estación de descarga.

4. **Bonus (0.3 puntos)** Deberás implementar un nuevo modelo de robot que solo es capaz de moverse en diagonal. Para esto debes implementar la lógica detrás del robot y agregar sus movimientos, de manera que sean permitidos únicamente si son válidos.



Figura 5: Imagen de un robot diagonal

Como base se te entregará el siguiente código que ejecuta una acción para cada robot en un tiempo.

```
% Existirá un máximo de instantes de tiempo, denotado por la constante bound
time(1..bound).

% Los robots cuentan con varias acciones posibles, moverse en las 4 direcciones, no hacer
↳ nada o recoger/soltar una caja
action(up).
action(down).
action(left).
action(right).
action(wait).
action(liftShelf).
action(dropShelf).
```

```
% Para cada instante de tiempo T, cada robot R puede ejecutar una única acción de las 7/11
↪ posibles:
1 {exec(R, A, T) : action(A)} 1 :- robot(R), time(T+1).
```

## Carpetas y Archivos

### Carpetas

- **tests:** Contiene 4 subcarpetas con distintos tests, fáciles, medianos y difíciles, además de una carpeta con problemas simples del bonus. Puedes crear más tests si te son de utilidad.
- **images:** Contiene los sprites de la visualización.

### Archivos

- **dccbodega.lp** contiene el modelo base con las acciones mínimas que debe realizar los robots, el límite de tiempo y la forma en que los robots eligen sus acciones. En este archivo debes implementar tu solución.
- **dccbodega\_bonus.lp** contiene el modelo base con las acciones mínimas que debe realizar los robots, el límite de tiempo y la forma en que los robots eligen sus acciones, junto con la definición del robot diagonal. En este archivo debes implementar tu solución para el bonus.
- **process.py** contiene la lógica en python para procesar una solución en clingo y pasarla a un archivo **output.txt** que pueda leer el visualizador.
- **visualizador.html** contiene el visualizador de las soluciones generadas por tu programa. Lo puedes abrir en tu navegador de preferencia y entregarle un archivo **output.txt**. Luego, podrás utilizar las flechas para avanzar o retroceder en los tiempos T para ver los movimientos realizados por los robots.

## DCCbodega Visualizador

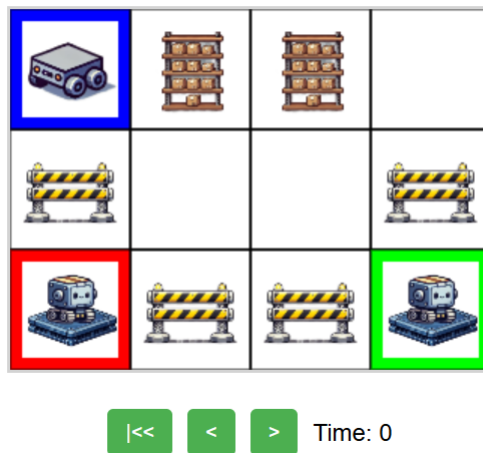


Figura 6: Imagen del visualizador

## Comandos a utilizar

- Para comenzar a resolver el problema puede ser útil querer manipular solamente el archivo de Clingo y ver la salida en la consola. Para esto se puede utilizar el siguiente comando:

```
clingo tests/<difícultad>/test<X>.lp dccbodega.lp
```

Con esto, se podrá ver la salida de Clingo en la consola y verificar si las reglas están funcionando correctamente. En esta parte, puedes modificar los `statements #show` para que te sea más fácil depurar el código.

- Una vez que tengas una solución que te parezca correcta, puedes ejecutar el siguiente comando para generar el archivo `output.txt`:

```
clingo dccbodega.lp tests/<difícultad>/test<X>.lp -c bound=N | python3 process.py
```

Este comando ejecutará clingo y luego procesará la salida para generar el archivo `output.txt`. Es muy importante que, en esta parte, mantengas los *statements #show* que se entregaron inicialmente en el archivo `dccbodega.lp` para que el proceso de parseo funcione correctamente. Si el comando `python` no funciona, puedes intentar con `python3` o `py`. Si a pesar de esto, sigue sin funcionar, es posible que se deba a que tu programa no encuentra solución.

- A continuación se incluyen los *statements #show* inicialmente entregados en el archivo `.lp`:

```
% Statements visualizer
#show rangeX/1.
#show rangeY/1.
#show time/1.
#show goalOn/3.
#show shelfOn/4.
#show robotOn/4.
#show carrying/3.
#show tableOn/2.
#show wallOn/2.
#show button/3.
#show diagRobotDefined/1.
```

- Finalmente, puedes abrir el archivo `visualizador.html` en un navegador y cargar el archivo `output.txt` que se generó en el paso anterior. De esta forma, podrás visualizar el mapa y la solución entregada por clingo.

## Entrega

Tu programa será corregido por otro programa para la corrección. Por lo tanto, es importante que utilices los mismos predicados sugeridos en este enunciado y que sigas las siguientes instrucciones de entrega.

- Tu programa **no deberá** incluir ninguna regla que defina un problema específico. Es decir, no debe incluir hechos que describan ordenes, posiciones, etc.
- En un subdirectorío `DCCbodega/`: Un archivo llamado `dccbodega.lp` con el programa sin el bonus y un archivo llamado `dccbodega_bonus.lp` para el programa con el bonus.

No se usarán criterios cualitativos para corregir la tarea. Usaremos un corrector automático sobre varios problemas. En cada problema, su programa obtendrá 1 punto si es capaz de encontrar una solución válida en el tiempo especificado por `bound` y obtendrá 0 puntos en otro caso.