



Prácticas AIN

pyGOMAS

Práctica 2: Comunicación y Coordinación

Índice

- ❖ Registro de Servicios
 - ❖ Comunicación y Coordinación
 - ❖ Trabajo a realizar
-

Recordatorio

- ✿ Hay definidos tres tipos de roles en los **agentes externos**:
 - ✿ *Soldier*: soldado de tipo general
 - ✿ ALLIED: va a por la bandera y vuelve a la base
 - ✿ AXIS: patrulla alrededor de la bandera
 - ✿ *Medic*: acude a curar
 - ✿ *FieldOps*: acude a dar munición
 - ✿ Un agente asume un único rol durante toda la partida
 - ✿ Cada rol tiene unas características y ofrece unos determinados **servicios**
-

Registro de servicios (I)

- Un rol debe registrar un **servicio** para que el resto de roles puedan solicitarlo:

.register_service("servicio_a")

Envia un mensaje al agente de servicio para registrar un servicio denominado “servicio_a” que estará disponible para su equipo.

- Ej:

.register_service("general");

registra el servicio “general” para su equipo.

Registro de servicios (II)

Registros que se hacen por defecto en todos los agentes:

*ALLIED

.register_service("allied");

Soldado: *.register_service("backup");*

Médico: *.register_service("medic");*

Fieldops: *.register_service("fieldops");*

*AXIS

.register_service("axis");

Soldado: *.register_service("backup");*

Médico: *.register_service("medic");*

Fieldops: *.register_service("fieldops");*

Registro de servicios (IV)

- * ¿Cómo saber que servicios hay disponibles desde un agente?
 - .get_medics*: Sigue al agente de servicios la lista de los médicos de su equipo.
Con la respuesta se crea una creencia: *myMedics(Medics_list)*
 - .get_fieldops*: Sigue al agente de servicios las lista de los operadores de campo de su equipo.
Con la respuesta se crea una creencia: *myFieldops(Fieldops_list)*
-

Registro de servicios (IV)

- ¿Cómo saber que servicios hay disponibles desde un agente?
 - .get_backups*: Sigue al Agente de Servicios los soldados de su equipo.
 - .get_service("servicio_a")*: Sigue al Agente de Servicios otro servicio (distinto de los tres anteriores) a los agentes tropa de su equipo que lo ofrezcan.
- La respuesta llega en forma de nueva creencia *servicio_a(L)*
- ```
+servicio_a(L)
<-
.print("Los agentes de mi equipo con el servicio_a son: ", L).
```
-

# Registro de servicios (V)

- NOTA: Todos las acciones .get siempre excluyen al propio agente que hace la solicitud de la lista que devuelven.
- Ejemplo de uso

Desde un plan se ejecuta: `.get_medics;`

Existe otro plan de la forma:

`+myMedics(M)`

```
<- .println("Mis médicos disponibles son: ", M);
 .length(M, X);
 if (X==0){ .println("No tengo médicos"); }.
```

Nota: si el agente que ejecuta este código fuese médico no aparecería en la lista M

# Registro de servicios (V)

---

- Ejemplo de uso de un servicio nuevo

Un agente A ejecuta: `.register_service("coronel");`

Otro agente B ejecuta: `.get_service("coronel");`

B además dispone del siguiente plan:

`+coronel(A)`

`<-`

`.print("Mi coronel es:", A);`  
`-coronel(_).`

---

# Registro de servicios (V)

---

- Ejemplo de uso de un servicio nuevo

Si el coronel ha muerto la lista estará vacía.

Una alternativa es que B ejecute lo siguiente:

```
.get_service("coronel");
.wait(2000); // un tiempo prudencial
if (coronel(A)){ .print("Mi coronel es:", A); -coronel(_); }.
```

---

# Coordinación (I)

---

- ✿ pyGOMAS dispone de mecanismos que permiten la coordinación entre agentes:
    - ✿ Sin comunicación (implícita):
      - ✿ Sensorización del entorno (ya visto en la práctica 1)
    - ✿ Con comunicación (explicita):
      - ✿ Mediante paso de mensajes
-

# Coordinación (II)

---

- ❖ Con comunicación
  - ❖ Envío de mensajes mediante la acción interna

*.send(Rec, Perf, Cont)*

Donde:

Rec → receptor del mensaje (puede ser una lista)

Perf → performativa (tell, untell, achieve, ...)

Cont → contenido

---

# Coordinación (III)

---

Ej: A1 quiere enviar un mensaje a los soldados médicos de su equipo diciendo que vayan a su posición (para ayudar, para coordinarse, para reagruparse, ...)

...

?position(Pos);

?myMedics(M); // se supone que antes he ejecutado .get\_medics

.send(M, tell, ir\_a(Pos));

---

# Coordinación (IV)

---

Ej: los médicos del equipo deberían disponer de un plan de la forma:

+*ir\_a(Pos)*[source(*A*)]

<-

.*println*("Recibido un mensaje de tipo *ir\_a* de ", *A*, "para ir a: ", *Pos*).

---

# Coordinación (V)

Ej: Si queremos que los soldados hagan algo más sofisticado

+*ir\_a(Pos)[source(A)]*

<-

*.println("Recibido mensaje ir\_a de: ", A, " para ir a: ", Pos);*

*+ayudando;*

*.goto(Pos).*

Mejoras:

- Comprobar si A tiene autoridad sobre el soldado
- Hacer caso sólo si tengo salud, armamento o las dos cosas
- Revisar antes otras tareas pendientes

# Coordinación (VI)

- ✿ Estrategias vistas (o por ver) en clase:
  - ✿ Organización jerárquica: El jefe manda !!!
  - ✿ Contract Net: Delegación de tareas
  - ✿ Social Choice: Votamos !!!
  - ✿ Subastas: quien me ofrece algo mejor !!!
  - ✿ ...

# Trabajo a Realizar (próximas sesiones)

---

- ✿ Objetivos:
    - ✿ Diseñar e implementar **un equipo de 10 agentes** con la distribución de tipos que deseéis (médicos, soldados y fieldops) para jugar a **capturar la bandera** en un mapa cualquiera como **atacante o como defensor**.
    - ✿ Es **necesario** emplear alguna técnica de **coordinación vía paso de mensajes** entre agentes del mismo equipo.
    - ✿ Se debe incluir al menos un **servicio nuevo por parte de un agente** y el uso del mismo por parte de otros agentes
    - ✿ Se deben realizar mejoras de **comportamientos** existentes (por ej. tratar de evitar el fuego amigo)
    - ✿ Se debe incluir al menos una nueva **acción interna** en Python.
-

# Trabajo a Realizar (próximas sesiones)

---

- ✿ ¿Qué os damos?
    - ✿ En Poliformat disponéis de una carpeta "práctica 2" con:
      - ✿ Un fichero json de ejemplo de configuración de la partida con 20 soldados
        - ✿ 10 allied y 10 axis (8 soldados, 1 médico y 1 fieldop)
      - ✿ Tres ficheros asl con la implementación **básica** de un soldado, un médico y un fieldop.
    - ✿ *Nota: la configuración de vuestro equipo es libre*
-

# Trabajo a Realizar (próximas sesiones)

---

## Posibles estrategias

### • ALLIED

- Elegir un capitán que coordine el ataque del resto
- Dividir el equipo en dos y atacar por oleadas
- Coordinar la retirada cuando se tiene la bandera

### • AXIS

- Elegir un capitán que coordine la defensa
  - Coordinar a los agentes para patrullar con distintos radios
  - Añadir algún agente vigía
  - Identificar que la bandera ha sido capturada y buscarla
-

# Trabajo a Realizar (Normas)

---

- ✿ **Reglas Básicas:**
    - ✿ No se puede consultar/solicitar información del sistema sobre el bando contrario que no sea suministrada por el entorno.
    - ✿ No puede existir comunicación entre agentes que no sea usando la acción interna *.send* y de acuerdo a la especificación proporcionada.
    - ✿ La práctica puede hacerse en grupo de dos alumnos.
-

# Trabajo a Realizar (Entrega)

- ❖ Entrega:
  - ❖ Ficheros \*.asl y \*.py desarrollados, así como el fichero json preparado para lanzar a los agentes del equipo.
  - ❖ **IMPORTANTE:** los nombres de vuestros agentes deben incorporar vuestro login para diferenciarlos del resto
  - ❖ El código, comentado y documentado debe seguir unas mínimas normas de estilo: tabulado y comentado.
  - ❖ Comprimir todo el directorio en un fichero <nombre\_equipo>.zip
  - ❖ Pequeña memoria, indicando las principales ideas de mejora aplicadas al equipo, así como unas breves conclusiones sobre los resultados obtenidos.

# Trabajo a Realizar (Entrega)

---

- ❖ Plazos
    - ❖ 19 de mayo (tarea en Poliformat)
    - ❖ Sesiones del 28 de abril, 5 y 12 de mayo serán para trabajar en la práctica.
-

# Prácticas AIN

---

pyGOMAS

Práctica 2: Comunicación y Coordinación

Añadiendo acciones internas

---

# Índice

---

- ✿ Recordatorio
  - ✿ Ejemplo de creación de una acción interna
    - ✿ Fichero .py
    - ✿ Fichero .asl
    - ✿ Fichero .json
-

# Recordatorio

---

- Una acción interna estará disponible desde el código JASON y comenzará con un ‘.’
  - Esta acción corresponderá a un método escrito en python, que pertenecerá a la clase de la que es nuestro agente.
-

# Ejemplo de creación de una acción interna

Fichero .py (I)

- ❖ Información disponible desde Python:
  - ❖ Atributos de AbstractAgent:
    - ❖ `team` : Número que identifica el equipo al que pertenece el agente
    - ❖ `services` : Lista con los identificadores de servicio que ofrece el agente.
  - ❖ Atributos de BDITroop:
    - ❖ `manager` : jid del Agente Manager.
    - ❖ `service` : jid del Agente de Servicios
    - ❖ `is_objective_carried` : (true/false) indica si lleva la bandera o no
    - ❖ `fov_objects` : lista de objetos actualmente en el campo de visión del agente
    - ❖ `aimed_agent` : agente al que actualmente está apuntando (o None)
    - ❖ `health` : salud actual del agente
    - ❖ `ammo` : munición actual del agente
    - ❖ `is_fighting` : indica si el agente está luchando en este momento (True/False)
    - ❖ `is_escaping` : indica si el agente está escapando en este momento (True/False)

# Ejemplo de creación de una acción interna

Fichero .py (I)

- ❖ Información disponible desde Python:
  - ❖ Atributos de AbstractAgent:
    - ❖ team : Número que identifica el equipo al que pertenece el agente
    - ❖ services : Lista con los identificadores de servicio que ofrece el agente.
  - ❖ Atributos de BDITroop:
    - ❖ Relativas al movimiento:
      - ❖ map
        - ❖ map.can\_walk(X, Z) : indica si es pisable la posición (X, 0, Z) (True / False)
        - ❖ map.allied\_base.get\_init\_x() , map.allied\_base.get\_init\_y() , map.allied\_base.get\_init\_z()
        - ❖ map.allied\_base.get\_end\_x() , map.allied\_base.get\_end\_y() , map.allied\_base.get\_end\_z()
        - ❖ map.axis\_base.get\_init\_x() , map.axis\_base.get\_init\_y() , map.axis\_base.get\_init\_z()
        - ❖ map.axis\_base.get\_end\_x() , map.axis\_base.get\_end\_y() , map.axis\_base.get\_end\_z()
      - ❖ velocity\_value : velocidad actual del agente
      - ❖ destinations : lista ordenada de los próximos destinos del agente.
      - ❖ movement
        - ❖ movement.velocity.x, movement.velocity.y, movement.velocity.z
        - ❖ movement.heading.x, movement.heading.y, movement.heading.z
        - ❖ movement.destination.x, movement.destination.y, movement.destination.z
        - ❖ movement.position.x, movement.position.y, movement.position.z

# Ejemplo de creación de una acción interna

Fichero .py (I)

- Información disponible desde Python:
  - Atributos de AbstractAgent:
    - `team` : Número que identifica el equipo al que pertenece el agente
    - `services` : Lista con los identificadores de servicio que ofrece el agente.
  - Atributos de BDITroop:
    - `self.soldiers_count = 0`
    - `self.medics_count = 0`
    - `self.engineers_count = 0`
    - `self.fieldops_count = 0`
    - `self.team_count = 0`
    - `threshold = Threshold()` Limits of some variables (to trigger some events)
      - `threshold.health`
      - `threshold.ammo`
      - `threshold.aim`
      - `threshold.shot`

# Ejemplo de creación de una acción interna

Fichero .py (II)

\* drunkenMonkey.py

```
import json
import random
from loguru import logger
from spade.behaviour import OneShotBehaviour
from spade.template import Template
from spade.message import Message
from pygomas.bditroop import BDITroop
from pygomas.bdfieldop import BDIFieldOp
from agentspeak import Actions
from agentspeak import grounded
from agentspeak.stdlib import actions as asp_action
from pygomas.ontology import DESTINATION

from pygomas.agent import LONG_RECEIVE_WAIT

class BDIDrunkenMonkey(BDIFieldOp):
 def add_custom_actions(self, actions):
 super().add_custom_actions(actions)
```

```
@actions.add(".drunkenMonkey", 0)
def _drunkenMonkey(agent, term, intention):
 randX = random.randrange(self.map.get_size_x() - 10)
 randZ = random.randrange(self.map.get_size_z() - 10)
 while (self.map.can_walk(randX, randZ) == False):
 randX = random.randrange(self.map.get_size_x() - 10)
 randZ = random.randrange(self.map.get_size_z() - 10)

 self.movement.destination.x = randX
 self.movement.destination.z = randZ
 self.bdi.set_belief(DESTINATION, tuple((randX, 0,
 randZ,)))
```

yield

# Ejemplo de creación de una acción interna Fichero .asl



- bdifieldop\_DM.asl

...

```
+enemies_in_fov(ID,Type,Angle,Distance,Health,Position)
```

<-

```
.drunkenMonkey;
```

```
.print("Drunken Monkey Fighting...");
```

```
.shoot(3,Position).
```

# Ejemplo de creación de acción interna Fichero .json



```
{
 "host": "gtirouter.dsic.upv.es",
 "manager": "ccc_m",
 "manager_password": "secret",
 "service": "ccc_s",
 "service_password": "secret",
 "axis": [
 {
 "rank": "BDISoldier",
 "name": "ccc_axis",
 "password": "secret",
 "amount": 8,
 "asl": "bdisoldier.asl"
 },
 {
 "rank": "BDIMedic",
 "name": "ccc_medic_axis",
 "password": "secret",
 "asl": "bdimedical.asl"
 },
 {
 "rank": "drunkenMonkey.BDIDrunkenMonkey",
 "name": "ccc_DM_fieldop_axis",
 "password": "secret",
 "asl": "bdifieldop_DM.asl"
 }
],
 "allied": [
 {
 "rank": "BDISoldier",
 "name": "ccc_allied",
 "password": "secret",
 "amount": 8,
 "asl": "bdisoldier.asl"
 },
 {
 "rank": "BDIMedic",
 "name": "ccc_medic_allied",
 "password": "secret",
 "asl": "bdimedical.asl"
 },
 {
 "rank": "BDIFieldOp",
 "name": "ccc_fieldop_allied",
 "password": "secret",
 "asl": "bdifieldop.asl"
 }
]
}
```

- Fichero JSON:  
Añadimos un nuevo tipo de agente que incorpora la acción comentada.



# Prácticas AIN

---

# pyGOMAS

## Práctica 2: Ejemplo de coordinación con contract-net



# Protocolo de red de contratos: Contract Net

---

Contract Net Protocol CNP (Smith, 1988).

Desarrollado para la resolución distribuida de problemas

Sigue usándose extensivamente en los SMAs

Sirve para que un agente contrate tareas a otros agentes

Suposiciones:

- 1 la negociación es un proceso local que no implica control centralizado,
  - 2 existe un medio bidireccional para intercambiar información,
  - 3 cada parte en la negociación evalúa la información desde su propia perspectiva
  - 4 el acuerdo final se alcanza mediante selección mútua.
-

# Contract Net

---

Un ejemplo ...

Un soldado solicita paquetes de medicina

- Pide ayuda a todos los médicos
  - Cada médico se propone o no a ayudarlo
  - El soldado elige el médico más adecuado  
(por distancia, por vida, ...)
-

# Ejemplo Pedir ayuda médica



Soldado



Médico 4



Médico 1



Médico 2



Médico 3

# Ejemplo Pedir ayuda médica



Soldado

cfp (posición)



Médico 4

cfp (posición)



Médico 1

cfp (posición)



Médico 2

cfp (posición)



Médico 3

# Ejemplo Pedir ayuda médica



Soldado



?

Médico 4



?

Médico 1



?

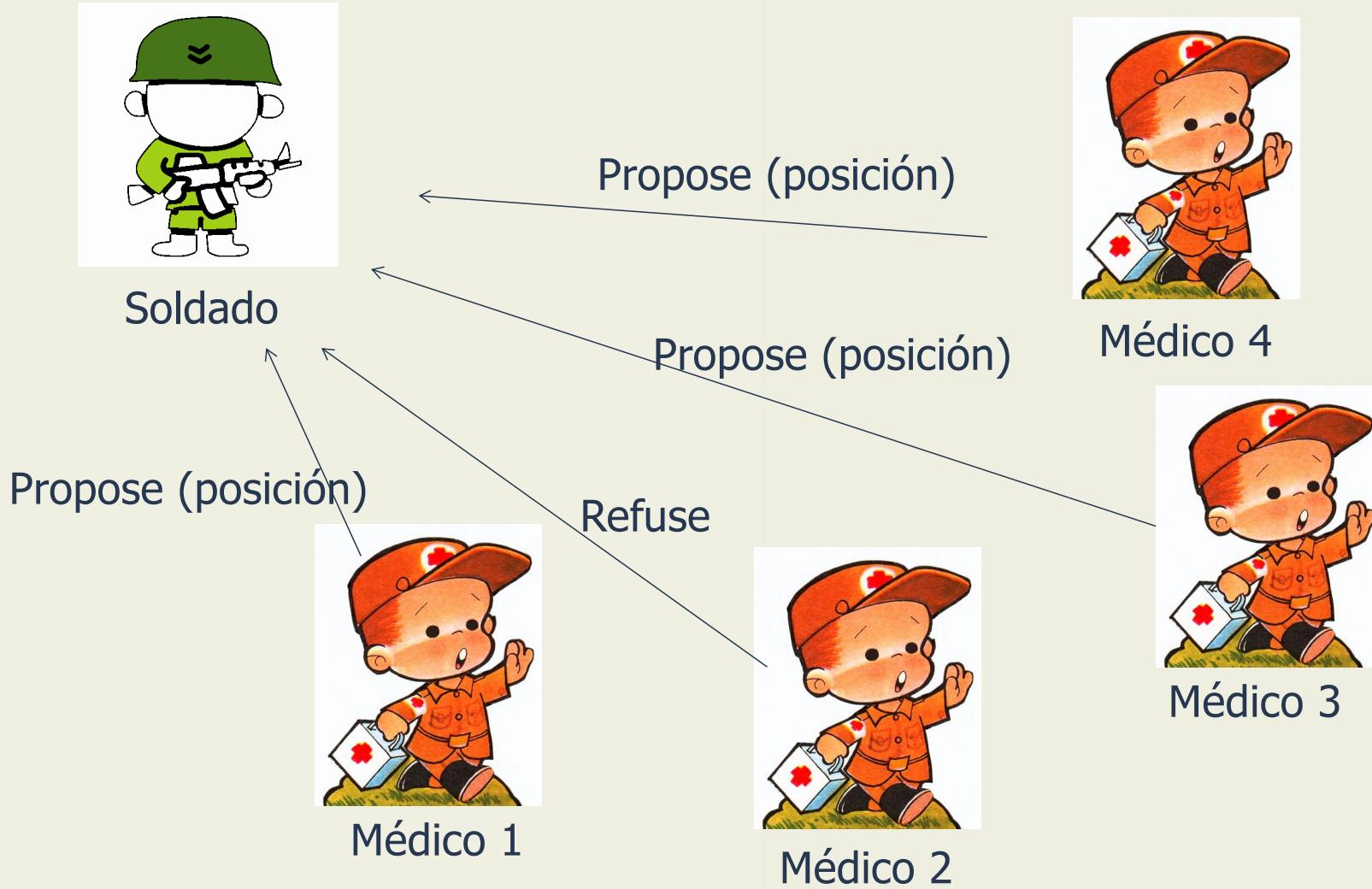
Médico 2



?

Médico 3

# Ejemplo Pedir ayuda médica



# Ejemplo Pedir ayuda médica



Soldado



Médico 4



Médico 1



Médico 2



Médico 3

# Ejemplo Pedir ayuda médica



Soldado

reject



Médico 4

accept



Médico 1

reject



Médico 2



Médico 3

# Ejemplo implementado



Médico 1



Médico 2



Médico 3



Médico 4

- 1º El soldado pedirá ayuda cuando llegue a una determinada posición
- 2º Todos los médicos le harán propuestas de ayuda

3º El soldado **elegirá** a uno de ellos (en este ejemplo al primero)

4º Todos los médicos **recibirán** una respuesta (positiva o negativa)

5º El médico elegido **acudirá** a dar ayudar



Soldado

# Implementación del ejemplo

Veamos el inicio del protocolo desde el iniciador (soldado)

1º El soldado **pedirá** ayuda cuando llegue a una determinada posición

```
+flag (F): team(100)
<-
.goto([100, 0, 100]);
+initpos.
```

```
+target_reached(T): team(100) & not (pedidaayuda)
<-
.print("He llegado al punto donde solicito ayuda");
-initpos;
+pedidaayuda;
.get_medics.
```

```
+myMedics(M): pedidaayuda
<-
.print("Pido ayuda");
?position(Pos);
+bids([]);
+agents([]);
.send(M, tell, savemeproposal(Pos));
.wait(1000);
!elegirmejor;
-myMedics(_) .
```



Soldado

# Implementación del ejemplo

---

Veamos el inicio del protocolo desde el médico

2º Todos los **médicos** le **harán propuestas** de ayuda

```
+savemeproposal(Pos)[source(A)]: not (ayudando(_,_))
<-
?position(MiPos);
.send(A, tell, mybid(MiPos));
+ayudando(A, Pos);
-savemeproposal(_);
.print("enviada propuesta de ayuda") .
```



Médico

# Implementación del ejemplo

---

Veamos el inicio del protocolo desde el médico

2º Todos los médicos le **harán propuestas** de ayuda  
Y el **soldado** las va recibiendo y almacenando

```
+mybid(Pos)[source(A)]: pedidaayuda
<-
 .print("Recibo propuesta");
 ?bids(B);
 .concat(B, [Pos], B1); --+bids(B1);
 ?agents(Ag);
 .concat(Ag, [A], Ag1); --+agents(Ag1);
 -mybid(Pos).
```



Soldado

# Implementación del ejemplo

Veamos el inicio del protocolo desde el médico

3º El soldado **elegirá** a uno de ellos (en este ejemplo al primero)

```
+!elegirmejor: bids(Bi) & agents(Ag)
```

```
<-
```

```
.print("Selecciono el mejor: ", Bi, Ag);
:nth(0, Bi, Pos); // no elijo el mejor, me quedo con el primero
:nth(0, Ag, A);
.send(A, tell, acceptproposal);
.delete(0, Ag, Ag1);
.send(Ag1, tell, cancelproposal);
-+bids([]);
-+agents([]).
```



Soldado

```
+!elegirmejor: not (bids(Bi))
```

```
<-
```

```
.print("Nadie me puede ayudar");
-pedidaayuda.
```

# Implementación del ejemplo

---

Veamos el inicio del protocolo desde el médico

4º Todos los médicos **recibirán** una respuesta (positiva o negativa)

```
+acceptproposal[source(A)]: ayudando(A, Pos)
<-
 .print("Me voy a ayudar al agente: ", A, "a la posicion: ", Pos);
 .goto(Pos).
```

```
+cancelproposal[source(A)]: ayudando(A, Pos)
<-
 .print("Me cancelan mi proposicion");
 -ayudando(A, Pos).
```



Médico

# Implementación del ejemplo

---

Veamos el inicio del protocolo desde el médico

5º El médico elegido **acudirá** a dar ayudar

```
+target_reached(T): ayudando(A, T)
<-
 .print("MEDPACK! para el agente:", A);
 .cure;
 ?miposicion(P);
 .goto(P);
 -ayudando(A, Pos).
```



Médico

# Implementación del ejemplo

---

**Veamos el ejemplo en ejecución**

# Dudas

---

Enviad vuestras dudas por:

- Teams (en horario de clase)
  - Correo: [vjulian@upv.es](mailto:vjulian@upv.es)  
[carrasco@dsic.upv.es](mailto:carrasco@dsic.upv.es)
  - o por Poliformat
-