Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Laboratorio - Práctica 1

Sistemes Intel·ligents Distribuïts

Fuster Palà, Llum
Molina Sedano, Òscar
Orteu Aubach, Júlia
Puerta del Valle, Javier

1 Introducción

Este trabajo presenta una colección de agentes autónomos basados en el modelo BDI (*Beliefs-Desires-Intentions*) para el entorno `pyGOMAS`[1] . El objetivo principal ha sido desarrollar agentes inteligentes capaces de tomar decisiones racionales en un escenario competitivo de captura de bandera, donde cada agente debe cumplir con un rol específico (Soldado, Médico u Operador de Campo) mientras contribuye al objetivo global de su equipo.

Nuestra implementación se fundamenta en el uso del lenguaje *AgentSpeak* para definir el comportamiento deliberativo de los agentes, estableciendo un conjunto coherente de creencias, objetivos y planes que guían sus acciones en el entorno. Hemos puesto especial énfasis en diseñar comportamientos que combinen tanto reactividad (respuesta a cambios inmediatos en el entorno) como proactividad (persecución de objetivos a largo plazo), siguiendo los principios del modelo BDI. El desarrollo completo del proyecto puede consultarse en el siguiente repositorio de *GitHub*: `github.com/llumfp/sid_agentspeak`.

A continuación, en la Sección 2 se define la racionalidad adoptada, se describe la estrategia de los agentes y se detalla su implementación en *AgentSpeak*. La Sección 3 presenta el sistema automatizado desarrollado para facilitar la ejecución del entorno en distintos sistemas operativos. Finalmente, en la Sección 4, se recogen las conclusiones del trabajo y se proponen posibles líneas de mejora. El Apéndice A incluye el código `.asl` correspondiente a cada agente implementado en la práctica.

> **Nota:** Para esta práctica se ha optado por desarrollar cada agente de forma completamente individual, sin suponer coordinación entre ellos desde el inicio. Esta decisión se ha tomado considerando que, durante la evaluación, los agentes se mezclarán aleatoriamente con implementaciones de otros grupos. Por lo tanto, con esta estrategia esperamos maximizar su efectividad de manera independiente, sin depender de un comportamiento colectivo preestablecido.

2 Diseño e Implementación

La siguiente sección define la noción de racionalidad adoptada y presenta el diseño lógico y la implementación funcional de los agentes según su rol en el entorno. Se han implementado cuatro agentes distintos: un soldado, un operador de campo y dos médicos, siendo este último el rol escogido para el agente extra requerido en la práctica, dado su potencial estratégico en la supervivencia del equipo y la gestión del estado de los aliados.

2.1 Definición de Racionalidad

En el entorno `pyGOMAS`, los agentes participan en un escenario competitivo de captura de bandera, en el que dos equipos —*Allied* y *Axis*— se enfrentan en mapas delimitados, con recursos limitados y percepción parcial del entorno. La dinámica es simétrica: ambos equipos pueden asumir roles ofensivos o defensivos según el desarrollo de la partida. Sin embargo, por defecto, el equipo *Allied* tiene como objetivo capturar la bandera, mientras que el equipo *Axis* debe evitarlo. Cada agente actúa de forma autónoma bajo condiciones inciertas y cambiantes.

> Bajo este contexto, definimos la racionalidad como la capacidad de un agente para tomar decisiones deliberadas que optimicen su impacto en el objetivo global del equipo, maximizando su utilidad individual sin depender de coordinación explícita con otros agentes.

[1] Documentación oficial: `https://pygomas.webs.upv.es/wp-content/uploads/2024/02/pyGOMAS-Manual-v1_0.pdf`.

Esta racionalidad se implementa siguiendo el modelo BDI (*Beliefs-Desires-Intentions*), que estructura el comportamiento del agente en tres componentes base: las creencias representan el conocimiento que el agente tiene sobre el entorno; los deseos corresponden a los estados que desea alcanzar; y las intenciones son los planes comprometidos que guían su actuación. Bajo este esquema, el comportamiento racional de los agentes en `pyGOMAS` se concreta en los siguientes principios:

1. **Beliefs:** Toma de decisiones basada en la información disponible del entorno, construida a partir de percepciones como la posición de aliados, enemigos o recursos.
2. **Desires:** Priorización dinámica de objetivos según el contexto táctico inmediato, como capturar la bandera, asistir a un compañero o conservar la propia vida.
3. **Intentions:** Adaptación al rol asignado y aprovechamiento de sus capacidades específicas mediante la ejecución de planes que gestionan eficientemente los recursos disponibles (salud, munición) y permiten responder de forma autónoma a cambios en la situación de la partida.

2.2 Racionalidad por Rol

La implementación de cada rol se ha realizado exclusivamente con `AgentSpeak`, empleando una estructura modular que separa lógica común y planes específicos según el equipo. Se han utilizado patrones clásicos del modelo BDI, como la replanificación de objetivos y el uso de `belief-based context selection` para activar comportamientos específicos según el entorno. La integración con las acciones estándar del entorno `pyGOMAS` ha permitido una implementación clara, reactiva y funcional.

2.2.1 Soldado

El objetivo principal del soldado es derrotar a sus enemigos. Para ello, sin embargo, necesita estar vivo y tener munición suficiente. El código completo del agente puede consultarse en el Apéndice A.1.

Lógica común La lógica común para ambos equipos incorpora la **reacción de ataque**, todo aquello relacionado con la **munición y salud** y la **exploración contínua**.

- Al detectar enemigos en el espacio de visión, el soldado dispara 10 balas a la posición que ha recogido. Se hace uso de la creencia `enemies_in_fov(ID, TYPE, ANGLE, DIST, HEALTH, [X,Y,Z])` y de la acción `.shoot(10,[X,Y,Z])`.
- Se usa una estrategia de guardado de posiciones de packs de munición y salud. Se define un umbral de vida (`h_t`) y munición (`a_t`) límite para empezar a buscar paquetes de recuperación de estas propiedades. Al detectar mediante las creencias `health(H)` y `ammo(A)` nuevos valores por debajo de esos umbrales, se procede a la ejecución de objetivos prioritarios `!find_health` y `!find_ammo` que tienen una estructura simétrica: buscar en la memoria de packs de salud o munición e ir hacia el primero de los objetivos (mejorable aplicando ordenación por proximidad a la posición actual) y, en el caso de estar vacía, explorar.
- Una acción que es muy positiva a nivel individual para cada agente es el hecho de explorar constantemente. En este caso se ha implementado rotando $\pi/2$ (90^{o}) cada muy poco tiempo, para así ser consciente de todo aquello que lo envuelve y detectar rápidamente los cambios en su entorno cercano.

Estrategia en equipo *Axis* (defensivo) El soldado del equipo defensivo debe controlar la bandera, eso implica patrullarla y captar información del entorno para actuar en consecuencia, siempre teniendo en cuenta la definición de racionalidad para este agente, que es derrotar a los enemigos y permanecer vivo y con munición. Para ello la implementación se ha realizado de la siguente manera:

- Se inicia el agente patrullando la bandera de manera prioritaria y luego explorando. Este patrullado se ha adaptado para convertir la reactividad del agente base inicial a un esquema deliberativo.
- Al finalizar el patrullaje inicial, se añade como objetivo no prioritario de nuevo, para así priorizar un objetivo restauración de la salud o munición cuando sea necesario.

Estrategia en equipo *Allied* (ofensivo) El soldado del equipo atacante se comporta de manera racional si sigue la definición de racionalidad descrita para él: debe recoger la bandera como objetivo principal mientras a su vez derrota los enemigos y se ocupa de no quedarse sin vida o munición.

- Con una secuencia de objetivos se implementa el hecho de capturar la bandera.
- En caso de éxito, se empieza el subobjetivo "*volver a la base*".
- En caso de no recoger la bandera porque algun otro agente amigo lo ha logrado antes, se procede a explorar y a derrotar a los enemigos que detecte.

Proactividad y reactividad

- **Proactividad:** Se define una jerarquía de prioridad de objetivos dependiendo de las reacciones con el entorno y el momento de la partida en qüestión con objetivos y distintos planes para cumplirlos como `+!explore`, `+!find_health`, `+!bring_flag_home`, `+!capture_flag` o `+!patroll`
- **Reactividad:** El agente reacciona a cambios en sus creencias como `+flag(F)`, `+flag_taken`, `+target_reached(T)` o `+packs_in_fov(_, TYPE, _, _, _, POSITION)` para actualizar sus prioridades de actuación.

2.2.2 Operador de Campo

El objetivo principal del operador de campo es proporcionar munición a sus aliados y colaborar en el objetivo de su equipo mediante una adaptación dinámica según el contexto táctico. Este agente es crucial para mantener la capacidad operativa del equipo en combate. El código completo del agente puede consultarse en el Apéndice A.2.

Lógica común

- Si su salud baja a menos del 50%, activa un mecanismo de autopreservación mediante el plan `!cure_myself`, que lo lleva a buscar activamente packs de curación para asegurar su supervivencia.
- Cuando su munición desciende por debajo del 20%, genera un pack de munición e inicia el plan `!search_ammo` para recargar, realizando movimientos exploratorios y colocando packs adicionales. Solo se activa si no está buscando medicinas (`not searching_cures`).
- Al detectar aliados en su campo de visión, deposita automáticamente packs de munición para mantener la capacidad operativa del equipo, pero con un tiempo de espera (cooldown) de 10 segundos para evitar una generación excesiva.
- Al finalizar alguno de los dos objetivos de supervivencia (`!cure_myself` o `!search_ammo`), llama al objetivo `!goback_objective` para volver a la acción principal de su estrategia, sea defensiva o de ataque.
- Si detecta enemigos dentro de su campo de visión, reacciona inmediatamente disparando hacia su posición, independientemente del rol al que pertenezca, colaborando así en la defensa o el ataque.

- Si detecta amigos en su campo de visión, deja munición por si alguien la necesita. Después espera un tiempo para poder volver a dar munición mediante este evento.

Estrategia en equipo *Axis* (defensivo)

- Implementa un sistema de patrullaje en dos fases alrededor de la bandera. En la primera fase, crea puntos de control en un radio definido y se desplaza entre ellos dejando packs de munición. (`!go_patroll_flag`)
- Al detectar el primer enemigo o percibir daño en su nivel de vida, transita a la segunda fase (`!protect_flag`). Su comportamiento cambia a una estrategia defensiva más activa, centrándose en atacar enemigos y proveer munición a los defensores, estando cerca de la bandera.
- Si no detecta enemigos visibles durante la segunda fase, primero se gira hacia la bandera en busca de amenazas y sigue rotando. Si no encuentra a ningún enemigo tras un ciclo determinado, se desplaza a la posición donde detectó al primer enemigo para volver a empezar un ciclo de búsqueda desde ese punto. La posición se ha escogido de esta forma para asegurar que sea una posición cercana a la bandera y en la dirección hacia la base del equipo contrario.
- Se ha intentado una tercera fase donde, al detectar un enemigo con la bandera, lo persigue disparándole continuamente y siguiéndolo cuando está en su campo de visión. Sin embargo, no se cumple nunca la condición para activar este objetivo, hecho que deja esta tercera fase como trabajo de futuro.

Estrategia en equipo *Allied* (ofensivo)

- Se desplaza directamente hacia la posición de la bandera al iniciar (`!go_flag`).
- Si consigue capturar la bandera, invierte su dirección y se dirige de regreso a la base (`!go_base`).
- Si no ha cogido la bandera al llegar a su posición, realiza un patrón de búsqueda en espiral que le permite explorar el área circundante para encontrarla.
  - Si la encuentra, va hacia ella para cogerla o, en su defecto, empieza a perseguir a su compañero que la ha cogido.
  - Si no la encuentra, se dirige de vuelta a la base. En este trayecto, si ve en algun momento la bansera, cambiará su rumbo hacia ella y, al tenerla, volverá a dirigirse a la base.

Creencias, objetivos y planes El agente se basa en percepciones como `health(H)`, `ammo(M)`, `enemies_in_fov`, `friends_in_fov` y `packs_in_fov`. Estas creencias actualizan objetivos como `!cure_myself`, `!search_ammo`, `!go_base` o `!protect_flag`, ejecutados mediante planes que integran acciones como `.goto`, `.shoot`, `.reload` o `.turn`, según el contexto táctico actual.

El operador mantiene creencias internas como `last_health(H)` para detectar daños recibidos, `firstSeen(Position)` para recordarla posición del primer enemigo visto, y estados como `searching_cures`, `going_to_ammo`, `patrolling` o `looking_for_flag` que determinan su comportamiento actual. Para el equipo Axis, utiliza `control_points(C)` y `total_control_points(L)` para gestionar el patrullaje cíclico, mientras que para Allied maneja estados como `going_to_flag` y `going_to_base` para la navegación estratégica.

La transición entre estos estados se controla mediante guardas contextuales como `not searching_cures` o `patrolling`, que permiten priorizar tareas urgentes (como buscar salud) sobre objetivos tácticos (como patrullar). El sistema permite al agente reactivar su estrategia principal (`!goback_objective`) al finalizar cada submisión de supervivencia o apoyo, manteniendo así el equilibrio entre las necesidades individuales y los objetivos del equipo.

Proactividad y reactividad

- **Proactividad:** Se manifiesta en la continua generación de paquetes de munición, el patrullaje preventivo alrededor de la bandera (en *Axis*) o la búsqueda activa de ésta (en *Allied*), así como el retorno planificado a la base cuando sostiene la bandera o necesita reorganizarse. También consideramos un comportamiento proactivo cuando, satisfecha la urgencia, el agente vuelve a su objetivo anterior (`goback_objective`).
- **Reactividad:** El agente reacciona ante emergencias personales (vida o munición bajas) con búsquedas prioritarias de recursos. También responde de forma inmediata a la aparición de enemigos con disparos (`.shoot`) y maniobras de giro (`.turn`) para mantenerlos dentro del campo de visión. Nuestra implementación permite la detección de daño recibido como alerta temprana de peligro, lo que le permite anticiparse a amenazas incluso sin contacto visual directo.

2.2.3 Médico

El agente médico tiene como objetivo principal mantener con vida a los aliados y a sí mismo el mayor tiempo posible, incrementando la probabilidad de éxito en la partida. A su vez, debe colaborar a derrotar a los enemigos. El código completo del agente puede consultarse en el Apéndice A.3.

Lógica común

- Cuando ve a un aliado con un porcentaje de vida menor al 80%, el agente acude a su posición y deja un pack de sanación cerca del aliado herido. Cuando deje el pack de sanación, el médico realiza una patrulla cerca del agente por si precisa de más ayuda médica.
- Si su propia salud baja a menos del 20%, el agente se replega a buscar un pack de sanación guardado previamente. Si no tiene ninguna creencia de la posición de algún pack de sanación visto con anterioridad, este se replegará y creará un pack de sanación para consumo propio. Este evento también se activa cuando ve a un enemigo en su campo de visión y se cumple las anteriores condiciones. Una vez recopilado el paquete de sanación, este seguirá con el plan establecido en su estrategia de equipo.
- De la misma forma, si el agente obtiene una creencia de que su munición actual es inferior al 5%, este intentará aplicar el mismos proceso que con el pack de sanación pero esta vez con un pack de munición. Si no tiene ninguna creencia guardada sobre la posición de algun pack de munición, este seguirá realizando la estrategia asignada según el equipo al que pertenece.
- Si ve a un enemigo, le disparará si el médico tiene más de 5% de munición y más del 20% de vida.

Estrategia en equipo *Axis* (defensivo)

- El objetivo principal es el de patrullar de manera cíclica alrededor de un punto medio entre la bandera y su base. De esta forma, el agente cada vez que alcanza un punto, puede dejar packs de sanación preventivas para poder sanar a sus aliados. Además, el agente está a una distancia segura y no muy alejada para poderse protegerse pero también poder apoyar en la defensa de la bandera.
- Cada vez que el agente alcanza un punto de patrulla, este revisará sus creencias respecto a si quedan soldados aliados vivos o no. Dependiendo de si quedan o no, se modificará la estrategia defensiva o no.
  - Si quedan soldados aliados con vida: Se mantiene la intención del objetivo principal.

  - Si no quedan soldados aliados con vida : El agente cambiará su patrullaje cerca de la bandera, de esta forma adoptará una esterategia mucho más defensiva hasta el final de la partida.

Estrategia en equipo *Allied* (ofensivo)

- El objetivo principal es el mismo que el descrito en la estrategia defensiva pero esta vez, el punto de patrullaje cambia i es el punto a distancia 3/4 entre la base y la bandera. De esta forma, se acerca mucho más a la posible zona de combate y puede ser de mayor utilidad al equipo atacante.
- Cada vez que el agente alcanza un punto de patrulla, este revisará sus creencias respecto a si quedan soldados aliados vivos o no. Dependiendo de si quedan o no, se modificará la estrategia ofensiva o no.
  - Si quedan soldados aliados con vida: Se mantiene la intención del objetivo principal.
  - Si no quedan soldados aliados con vida : El agente modificará su comportamiento obviando su objetivo principal y replanificando hacia el objetivo de tomar la bandera y volver a base para poder ganar la partida. De esta forma nos aseguramos que, individualmente, el agente puede conseguir el objetivo de ganar la partida.

Creencias, objetivos y planes El agente utiliza percepciones como `friends_in_fov`, `enemies_in_fov`, `health(X)` y `ammo(X)` para actualizar sus objetivos y activar planes contextuales como moverse, curar o patrullar. Además, también se añaden creencias nuevas como `san_pack(Pos_san_pack)`, `ammo_pack(Pos_ammo_pack)`, `yendo_sanacion`, `yendo_municion`, `midpoint(Midpoint)` que permiten guardar la posición de los packs de sanación y munición, mantener el estado de ir hacia estos paquetes o guardar la información del punto de patrullaje inicial. Además, se incluyen nuevas intenciones como `reload_ammo o reload_san` que permiten activar el objetivo de recuperarse vida o munición, `patroll_at_midpoint o do_patroll` que implementan el patrullaje cíclico basándose en el código de los agentes base de PYGOMAS o `to_atack, to_defend o save_flag` que implementan las intenciones relacionadas con la replanificación de la estrategia comentada con anterioridad.

Proactividad y reactividad

- **Proactividad:** Estrategia de patrullaje con packs de curación preventivas, replanificación de la estrategia durante el patrullaje, ir a curar a aliados con baja vida, buscar munición o sanación cuando el agente necesita recuperarse o recargar
- **Reactividad:** Estrategia a seguir cuando ve a un enemigo, guardar la posición del paquete si ve uno durante el patrullaje.

2.2.4 Médico Extra

El agente extra desarrollado corresponde al rol de médico y también adapta su comportamiento dinámica- mente según el equipo asignado (*Allied* o *Axis*). Su objetivo es maximizar la supervivencia del equipo, proporcionando curación en momentos críticos y manteniendo su propia operatividad mediante gestión autónoma de recursos y reacción táctica. El código completo del agente puede consultarse en el Apéndice A.4.

> **Nota:** La implementación del médico extra es lógicamente más simple y menos granular que la del médico base. Esta diferenciación permite explorar dos enfoques de distinta complejidad para un mismo rol.

Lógica común

- Prioriza la curación de aliados cuya salud esté por debajo del 50%, desplazándose hacia su posición y generando un pack médico.
- Si su propia salud baja del umbral definido (40), se activa un plan de autopreservación que incluye curación inmediata y maniobras evasivas para evitar el enfrentamiento directo.
- Cuando el nivel de munición es inferior al 40%, busca packs de recarga si están disponibles en el campo de visión.
- Ante la presencia de enemigos, adapta su comportamiento según la distancia: si el enemigo está próximo, se limita a retroceder y curarse; si está a distancia segura y dispone de munición suficiente, dispara controladamente para apoyar al equipo sin comprometer su rol principal.

Estrategia en equipo *Axis* (defensivo)

- Al iniciar, genera puntos de patrulla alrededor de la bandera.
- Recorre estos puntos cíclicamente, dejando packs médicos en cada uno.
- Reinicia el patrullaje al finalizar el ciclo.
- Adapta su respuesta táctica según cercanía y munición ante enemigos.

Estrategia en equipo *Allied* (ofensivo)

- Se dirige directamente a la bandera al comenzar la partida.
- Si la bandera es capturada, se desplaza a la base para asistir en la retirada.
- No sigue soldados individualmente, pero puede adaptarse si fuera necesario.

Creencias, objetivos y planes El agente se basa en percepciones como `friends_in_fov`, `enemies_in_fov`, `health(X)` y `ammo(X)`. Estas creencias actualizan objetivos como moverse, curar o patrullar, ejecutados mediante planes que integran acciones como `.goto`, `.cure` o `.shoot`, según el contexto actual. Los planes están estructurados con contextos específicos como `patrolling & team(200) & patrol_point(P) & total_control_points(TP) & P + 1 < TP` que permiten acciones condicionadas a múltiples factores simultáneos, implementando así un comportamiento racional complejo mediante reglas simples concatenadas.

Proactividad y reactividad

- **Proactividad:** patrullaje, búsqueda de recursos y curación anticipada.
- **Reactividad:** adaptación inmediata ante enemigos, aliados heridos o eventos tácticos como la captura de la bandera.

3 Ejecución

Para verificar el correcto funcionamiento de los agentes implementados, se ha establecido un entorno de pruebas utilizando scripts de ejecución automatizados para sistemas Linux, macOS o Windows. Estos scripts se encuentran en el directorio `exec-scripts/` del repositorio y permiten iniciar todos los componentes necesarios para las simulaciones de forma secuencial y controlada. Estos scripts generan tres comandos con tiempo de espera entre ellos: el primero para inicializar el manager, el segundo para renderizar el mapa y el tercero para poner los agentes a atacar y defender respectivamente.

4 Conclusiones

El desarrollo de agentes racionales siguiendo el modelo BDI ha permitido obtener comportamientos autónomos coherentes con los objetivos tácticos de cada rol, en un entorno complejo y competitivo como `pyGOMAS`. La implementación se centró en maximizar la efectividad individual de cada agente, lo cual se alineaba con los criterios de evaluación definidos para la práctica.

No obstante, cabe señalar que esta aproximación individualista —tanto en la lógica de los agentes como en la organización del trabajo— implicó ciertas limitaciones prácticas. Cada miembro del equipo se especializó en un único rol, lo que facilitó la profundización técnica y la coherencia local del código. Sin embargo, en retrospectiva, esta decisión redujo las oportunidades de colaboración y el intercambio de soluciones comunes que habrían sido útiles entre agentes.

Detectamos que muchas estructuras de decisión, patrones de patrullaje o mecanismos de replanificación podrían haberse reutilizado y adaptado fácilmente entre roles. No haber compartido ni alineado estas lógicas desde el inicio nos hizo invertir tiempo en resolver problemas similares por separado, sin explotar al máximo el potencial conjunto del equipo.

Aunque la práctica no requería explícitamente optimizar el rendimiento global ni coordinar agentes, una estrategia colectiva en fases iniciales habría permitido mejorar la eficiencia del desarrollo y posiblemente, el rendimiento final de los agentes. Esta reflexión representa un valor añadido del trabajo: la toma de conciencia sobre cómo la organización del desarrollo impacta directamente en los resultados técnicos, incluso cuando el objetivo es puramente individual.

4.1 Líneas de mejora futura

De cara a futuras mejoras, se identifican diversas líneas que podrían potenciar aún más la autonomía, adaptabilidad y rendimiento de los agentes desarrollados:

- Implementar mecanismos de comunicación entre agentes del mismo equipo para compartir información relevante sobre el entorno.
- Introducir técnicas de aprendizaje durante la ejecución para que los agentes ajusten su comportamiento según experiencias previas.
- Desarrollar capacidades de reconocimiento de patrones enemigos para anticiparse a estrategias recurrentes.
- Optimizar la planificación de rutas para una navegación más eficiente en mapas con obstáculos.
- Integrar métodos de aprendizaje automático para ajustar parámetros tácticos de forma dinámica.

A Código ASL

A.1 Código del Soldado

```
/*
* ========================================
 * ESTRATEGIA DE AGENTE SOLDADO PARA PYGOMAS
 * ========================================
 * Agente Deliberativo     Ambos Equipos
*/
/* ================== CRENCIAS INICIALES ================== */

h_t(50).
a_t(50).

wait(1000).

+flag(F) : team(200) <-
  !patroll;
  !explore.

+flag(F) : team(100) <-
  !capture_flag;
  !explore.

/* ================== META PRINCIPAL (AXIS) ================== */

/* El soldado debe patrullar la bandera */

+!patroll : flag(F) <-
  .create_control_points(F, 20, 4, C);
  +control_points(C);
  .length(C, L);
  +total_control_points(L);
  +patrolling;
  +patrol_point(0);
  !to_point.

/* ================== PLANES PARA LA PATRULLA DE LA BANDERA (AXIS)
    ================== */

+!to_point : patrol_point(P) <-
  ?control_points(C);
  .nth(P,C,A);
  .print("Going to", A);
  .goto(A).

+target_reached(T): patrolling & total_control_points(TOTAL) <-
  ?patroll_point(P);
  if (P == TOTAL) {
    !!patroll;
  };
  if (P <= TOTAL) {
    -+patroll_point(P+1);
  };
  -target_reached(T).

/* ================== META PRINCIPAL (ALLIED) ================== */

/* El soldado debe capturar la bandera */

+!capture_flag : not flag_taken & team(100) <-
  .wait(1000);
  .print("Meta: capture_flag iniciada");
  ?health(H);
```

```
  ?ammo(A);
  .print("Helth =", H, " Ammo =", A);
  !assess_flag.

/* ==================== PLANES PARA LA CAPTURA DE LA BANDERA (ALLIED)
    ==================== */

+!assess_flag : flag(F) <-
  .print("Meta: assess_flag. Se conoce flag en: ", F);
  !take_flag(F).

+!take_flag(F) : true <-
  .print("Meta: take_flag. Moviendose hacia la bandera en: ", F);
  +to_flag;
  .goto(F).

+target_reached(T) : to_flag <-
  .print("Bandera alcanzada.");
  -to_flag;
  -target_reached;
  if (not flag_taken) {
    !!capture_flag;
  }.

/* ------------------------------------------------------------------------

`bring_flag_home` solo se dispara en caso de coger la bandera!

Si el agente soldado ha sido más lento que otro soldado porque ha
empezado en una posición de la base más alejada, no cogerá la bandera.

Ahora pues tocará diseñar qué hace en caso de no coger la bandera.

------------------------------------------------------------------------ */

+flag_taken <-
  // .wait(500);
  !bring_flag_home.

+!bring_flag_home : base(B) & flag(F) <-
  .print("He llegado a la bandera en: ", F, ". Capturando bandera...");
  .print("Meta: bring_flag_home. Moviendose hacia la base en: ", B);
  +returning;
  .goto(B).

+target_reached(T) : returning <-
  .print("He llegado a la base en: ", B, ". Entregando bandera.");
  -returning;
  -target_reached(T).

/*
============================================================
################### COMPORTAMIENTO COMUN ###################
============================================================
*/

/* ==================== REACCIÓN DE ATAQUE ==================== */

+enemies_in_fov(ID, TYPE, ANGLE, DIST, HEALTH, [X,Y,Z]) : true <-
  .print("Shooting to:", ID, TYPE);
  .shoot(10,[X,Y,Z]).

/* ==================== MUNICIÓN Y SALUD ==================== */

// Buscar munición
```

```
+ammo(A) : a_t(T) & A < T & not finding_ammo <-
  .print("No tengo suficiente municion.");
  +finding_ammo;
  !find_ammo.

// Buscar cura
+health(H) : h_t(T) & H < T & not finding_health <-
  .print("Necesito recuperar salud.");
  +finding_health;
  !find_health.

+!find_ammo : ammo_pack(P) <-
  .print("Dirigiendome hacia la municion.");
  +to_ammo;
  .goto(P).

+!find_ammo <-
  .print("No tengo constancia de ningun pack de municion.");
  -finding_ammo;
  !explore.

+!find_health : cure_pack(P) <-
  .print("Dirigiendome hacia la cura.");
  +to_cure;
  .goto(P).

+!find_health <-
  .print("No tengo constancia de ningun pack de cura.");
  -finding_health;
  !explore.

+target_reached(T) : to_ammo <-
  .print("Ammo cogida");
  -to_ammo;
  -finding_ammo;
  -target_reached(T).

+target_reached(T) : to_cure <-
  .print("Cura cogida");
  -to_cure;
  -finding_health;
  -target_reached(T).

+packs_in_fov(_, TYPE, _, _, _, POSITION) : TYPE == 1001 & not cure_pack(POSITION)
    <-
    .print("He detectado un CURE pack distinto en ", POSITION);
    +cure_pack(POSITION).

+packs_in_fov(_, TYPE, _, _, _, POSITION) :TYPE == 1002 & not ammo_pack(POSITION)
    <-
    .print("He detectado un AMMO pack distinto en ", POSITION);
    +ammo_pack(POSITION).

/* ================== EXPLORAR ================== */

+!explore <-
  // .print("Explorando...");
  .turn(1.571); // Girar pi/2 (90  )
  .wait(100);
  !!explore.
```

Listing 1: Código fuente del agente `bdisoldier.asl`

A.2 Código del Operador de Campo

```
/*
 * ====================================================
 * ESTRATEGIA DE AGENTE OPERADOR DE CAMPO PARA PYGOMAS
 * ====================================================
 * COMPORTAMIENTO GENERAL:
########################## URGENCIA MEDICA! ########################
Buscar medicinas.
-> EMPIEZA: cuando tenemos una vida inferior a 50
-> ACABA: cuando hemos recuperado vida
-> QUE HACE: dar una vuelta en busca de medicinas e ir hacia la medicina encontrada
*/

+health(Hnow): Hnow<50
  <-
  .reload;
  !cure_myself.

+!cure_myself
  <-
  +searching_cures.

+packs_in_fov(_, MedicPack, _, _, _, MedicinePos): searching_cures & MedicPack=1001 &
    not going_to_cure
  <-
  .print("He visto una cura, me dirijo hacia ella");
  +going_to_cure;
  .goto(MedicinePos).

+searching_cures: not going_to_cure
  <-
  .print("Explorando buscando medicinas");
  .wait(1000);
  .turn(-0.750).

+pack_taken(MedicPack,_): searching_cures & MedicPack=medic
  <-
  .print("Medicina tomada");
  -going_to_cure;
  -searching_cures;
  !goback_objective.

+target_reached(MedicinePos): searching_cures
  <-
  .print("He llegado donde estaba la medicina");
  -target_reached(MedicinePos);
  -going_to_cure;
  !check_cure.

+!check_cure: not searching_cures
  <-
  .print("Ya no esoy buscando curas").

+!check_cure: health(H) & H>20
  <-
  -searching_cures;
  !goback_objective.

+!check_cure: health(H) & H<20
  <-
  !cure_myself.

/*
########################## FALTA MUNICION! ########################
Poner municion y cogerla.
```

```
-> EMPIEZA: cuando tenemos una municion inferior a 20
-> ACABA: cuando hemos recuperado municion
-> QUE HACE: dejar municion e ir a cogerla
*/

+ammo(M): M<20 & not searching_cures
  <-
  .print("Voy a dejar muncion porque tengo poca");
  .wait(1000);
  .reload;
  !search_ammo.

+!search_ammo: team(100) & going_to_base
  <-
  .print("Tengo municion baja, pero ya estoy yendo de vuelta a la base");
  !goback_objective.

+!search_ammo
  <-
  .print("Voy a buscar municion");
  +searching_ammo.

+packs_in_fov(_, AmmoPack, _, _, _, AmmoPos): searching_ammo & AmmoPack=1002 & not
    going_to_ammo
  <-
  .print("Yendo hacia la municion vista");
  +going_to_ammo;
  .goto(AmmoPos).

+searching_ammo: not going_to_ammo
  <-
  .print("Explorando buscando municion");
  .wait(1000);
  .reload;
  .turn(-0.750).

+pack_taken(AmmoPack,_): searching_ammo & AmmoPack=fieldops
  <-
  .print("He encontrado la municion buscada");
  -going_to_ammo;
  -searching_ammo;
  !goback_objective.

+target_reached(AmmoPos): searching_ammo & going_to_ammo
  <-
  .print("He encontrado llegado donde estaba la municion pero no la he tomado");
  -going_to_ammo;
  +searching_ammo;
  .reload.

/*
########################## YA NO HAY NECESIDADES A CUBRIR ########################
Volver al objetivo que teníamos antes de tener poca vida
*/

+!goback_objective: team(200) & flag(F)
  <-
  .goto(F).

+!goback_objective: team(100) & going_to_flag
  <-
  !go_flag.

+!goback_objective: team(100) & going_to_base
  <-
```

```
  !go_base.

+!goback_objective
  <-
  .print("Estoy intentando volver de encontrar una necesidad y no se volver al
    objetivo principal");
  !goback_objective.

/*
########################## INICIO ########################
*/

// Inicialización de la variable para detectar que me están disparando
!first_health.

+!first_health: health(H)
  <-
  .print("La salud inicial es de", H);
  +last_health(H).

+!first_health
  <-
  .print("Esperando información de la salud.");
  .wait(1000);  // Esperar un poco antes de intentarlo de nuevo
  !first_health.  // Llamada recursiva para intentarlo de nuevo

// Activamos el objetivo inicial según si es un agente atacante o defensor

+team(100)
  <-
  .print("Soy atacante y me estoy dirigiendo a la bandera");
  !go_flag.


+team(200)
  <-
  .print("Soy defensor y me estoy dirigiendo a patrullar al rededor de la bandera");
  !go_patroll_flag.

/*
                                                                    #     #       #
     #####   ####   ####
#############################################################       # #    #       #
       #     #      #   #
######################## ALLIED(100) ######################      #####   #       #
       #     ###    #   #
#############################################################      #   #   #       #
       #     #      #   #
                                                                   #   #   #####   #####
     #####   ####   ####
*/

+!go_flag: flag(F)
  <-
  +going_to_flag;
  .goto(F).

+!go_flag
  <-
  wait(300);
  !go_flag.

+enemies_in_fov(ID,Type,Angle,Distance,Health,PositionEnemie): team(100)
  <-
  +batalla_empezada;
```

```
  .print("Disparando al enemigo");
  .shoot(5,PositionEnemie).

+friends_in_fov(_,_,_,_,_,_): not searching_cures & not recargado & batalla_empezada
  <-
  .print("Voy a dejar muncion porque vi un amigo");
  .reload;
  +recargado;
  .wait(10000);
  .print("Ahora puedo volver a dar municion");
  -recargado.

+flag_taken: team(100)
  <-
  .print("I got the flag");
  !go_base.

+target_reached(T): team(100) & going_to_flag
  <-
  .print("He llegado a la bandera pero no la he cogido, así que voy a buscarla");
  +turned(0);
  +looking_for_flag.

+looking_for_flag: going_to_flag & turned(P) & P<5 & not searching_cures & not
    going_to_moving_flag
  <-
  -+turned(P+1);
  .print("Buscando bandera");
  .wait(1000);
  .reload;
  .turn(-0.750).

+packs_in_fov(_, FlagType, _, _, _, FlagPos): looking_for_flag & not
    going_to_moving_flag
  <-
  .print("Bandera encontrada");
  .wait(1000);
  +turned(0);
  .goto(FlagPos);
  +going_to_moving_flag.

+looking_for_flag: going_to_flag & turned(P) & P>5 & not searching_cures
  <-
  .print("He dado vueltas y no veo la bandera. Vuelvo a la base.");
  !go_base.

+!go_base: base(B)
  <-
  +going_to_base;
  -going_to_flag;
  .goto(B).

+!go_base
  <-
  !go_base.

/*
                                                          #    #   #  #####
    ####
############################################################      # #    # #     #   #
######################### AXIS(200) #########################     #####    #      #
    ####
############################################################     #   #   # #     #
      #
                                                          #   #  #   #  #####
```

```
    ####
*/


/*
########################## FASE 1 ########################
Empezamos yendo a patrullar al rededor de la bandera.
-> EMPIEZA: inicio de la partida
-> ACABA: cuando vemos al primer enemigo
-> QUE HACE: crear puntos al rededor de la bandera y patrullar siguiendo estos puntos
*/

+!go_patroll_flag: flag(F)
  <-
  .create_control_points(F,25,3,C);
  +control_points(C);
  .length(C,L);
  +total_control_points(L);
  +patrolling;
  +patroll_point(0);
  .print("Got control points").

// Agregar un plan de respaldo para cuando las condiciones aún no se cumplen
+!go_patroll_flag
  <-
  .print("Esperando información del equipo y la bandera...");
  .wait(1000);  // Esperar un poco antes de intentarlo de nuevo
  !go_patroll_flag.  // Llamada recursiva para intentarlo de nuevo

+target_reached(T): patrolling & not searching_cures
  <-
  .print("Punto de patrulla alcanzado, dejando pack de municion");
  .reload;
  ?patroll_point(P);
  -+patroll_point(P+1);
  -target_reached(T).

+patroll_point(P): total_control_points(T) & P<T & patrolling & not searching_cures
  <-
  ?control_points(C);
  .nth(P,C,A);
  .goto(A).

+patroll_point(P): total_control_points(T) & P==T & patrolling & not searching_cures
  <-
  -patroll_point(P);
  +patroll_point(0).

// Cuando vemos al primer enemigo empezamos la FASE 2
+enemies_in_fov(ID,Type,Angle,Distance,Health,PositionFirstEnemie): patrolling
  <-
  .print("Empezamos FASE 2: primer enemigo visto en ", PositionFirstEnemie);
  -patroll_point(P);
  -patrolling;
  .stop;
  .shoot(5,PositionFirstEnemie);
  .print("Disparado");
  +firstSeen(PositionFirstEnemie);
  !protect_flag.

// Otra forma de empezar la FASE 2, cuando notamos que nos estan disparando
+health(Hcurrent): patrolling & last_health(H)
  <-
  .wait(1000);
  if (H-Hcurrent < 5) {
```

```
311       .print("Me están disparando. FASE 2");
312       -patrolling;
313       .stop;
314       !protect_flag;
315   }
316   else {
317       .print("Nadie me esta disparando. Sigo patrullando.");
318   }.
319
320 /*
321 ########################## AMIGO A LA VISTA ########################
322 Dejar munición cuando veo a un amigo, a partir de la fase 2.
323 -> EMPIEZA: cuando ve a un amigo
324 -> ACABA: nunca
325 -> QUE HACE: dejar munición siempre que veamos un amigo, esperando un tiempo por si se
        repite
326 */
327 +friends_in_fov(_,_,_,_,_,_): not searching_cures & not recargado & protecting
328   <-
329   .print("Voy a dejar muncion porque vi un amigo");
330   .reload;
331   +recargado;
332   .wait(10000);
333   .print("Ahora puedo volver a dar municion");
334   -recargado.
335
336 /*
337 ########################## FASE 2 ########################
338 Empezamos a proteger la bandera.
339 -> EMPIEZA: cuando vemos al primer enemigo
340 -> ACABA: cuando vemos a un enemigo que tiene la bandera
341 -> QUE HACE: atacar al enemigo, dar munición al soldado, encararse hacia la bandera
        por si está el enemigo
342 */
343
344 +!protect_flag
345   <-
346   +protecting.
347
348 +enemies_in_fov(ID,Type,Angle,Distance,Health,Position): protecting & not firstSeen(
        Any)
349   <-
350   -looking_for_enemies;
351   -searching_enemies(P);
352   .print("Disparado");
353   +firstSeen(Position);
354   .shoot(5,Position).
355
356 +enemies_in_fov(ID,Type,Angle,Distance,Health,Position): protecting & firstSeen(
        PositionFirstEnemie)
357   <-
358   -going_to_position_enemie;
359   -looking_for_enemies;
360   -searching_enemies(P);
361   .print("Disparado");
362   .shoot(5,Position).
363
364 +protecting: not enemies_in_fov(_,_,_,_,_,_) & flag(F) & not looking_for_enemies & not
        searching_cures
365   <-
366   .print("Estoy mirando hacia la bandera para ver si puedo encontrar a los enemigos");
367   .look_at(F);
368   +searching_enemies(0).
369
370 +looking_for_enemies: protecting & searching_enemies(P) & P<5 & not searching_cures
```

```
371   <-
372   -+searching_enemies(P+1);
373   .print("Buscando enemigos");
374   .wait(1000);
375   .reload;
376   .turn(-0.750).
377
378 +looking_for_enemies: protecting & searching_enemies(P) & P>4 & firstSeen(
      PositionFirstEnemie) & not searching_cures
379   <-
380   .print("Yendo a la posicion donde vimos el primer enemigo: ", PositionFirstEnemie);
381   +going_to_position_enemie;
382   .goto(PositionFirstEnemie).
383
384 +target_reached(PositionFirstEnemie): protecting & going_to_position_enemie & not
      searching_cures
385   <-
386   -going_to_position_enemie;
387   .print("He llegado a la posicion donde estaba el primer enemigo");
388   .reload;
389   +searching_enemies(0).
390
391
392 /*
393 ########################## FASE 3 ########################
394 Empezamos a perseguir al enemigo que cogió la bandera.
395 -> EMPIEZA: cuando vemos a un enemigo con la bandera
396 -> ACABA: hasta el final
397 -> QUE HACE: perseguir y atacar al enemigo
398 No lo pilla nunca, no estoy segura si así se puede encontrar el enemigo que ha cogido
      la bandera...
399 */
400
401 +packs_in_fov(_, FlagType, _, _, _, FlagPos): enemies_in_fov(ID,Type,Angle,Distance,
      Health,Position) & FlagPos=Position
402   <-
403   .print("He visto al enemigo con la bandera!");
404   !follow_enemie(ID).
405
406 +!follow_enemie(ID): not chasing_flag_carrier
407   <-
408   .print("Persiguiendo al enemigo con ID ", ID, " que tiene la bandera");
409   +chasing_flag_carrier;
410   +target_enemy(ID);
411   .reload.
412
413 +enemies_in_fov(ID,Type,Angle,Distance,Health,Position): chasing_flag_carrier &
      target_enemy(ID) & not following_enemy
414   <-
415   .print("El enemigo con la bandera sigue a la vista, actualizando posición");
416   .shoot(5,Position);
417   -searching_enemy_flag;
418   .goto(Position);
419   +following_enemy.
420
421 +target_reached(Position): following_enemy & chasing_flag_carrier
422   <-
423   -target_reached(Position);
424   .print("He llegado a donde estaba el enemigo");
425   .reload;
426   -following_enemy.
427
428 +chasing_flag_carrier: not enemies_in_fov(ID,_,_,_,_,_) & target_enemy(ID) & not
      searching_enemy_flag
429   <-
```

```
  .print("He perdido de vista al enemigo con la bandera, buscando");
  +searching_enemy_flag;
  .turn(-0.750).

+searching_enemy_flag: chasing_flag_carrier
  <-
  .print("Buscando al enemigo con la bandera");
  .wait(1000);
  .turn(0.750).
```

Listing 2: Código fuente del agente `bdifieldop.asl`

A.3 Código del Médico

```
/*
* ======================================
 * ESTRATEGIA DE AGENTE MÉDICO PARA PYGOMAS
 * ======================================
 *
//BasePoints(base_points) :- team(200) & base_points = [160, 0, 180].
//BasePoints(base_points) :- team(100) & base_points = [20, 0, 80].
*/
/*
########################## Lista de variables ######################
F = Flag
BasePoints_all = Puntos de control cercanos a la base
Pos = Posición a patrullar
L = Longitud de la lista de puntos de control
H = Vida del agente
M = Munición del agente
B = Posición Base
P = Punto de patrulla actual
T = Total de puntos de control
A = Nuevo Punto de control al que ir

TYPE_san_pack = Tipo del pack de sanación
Pos_san_pack = Posición del pack de sanación

TYPE_ammo_pack = Tipo del pack de munición
Pos_ammo_pack = Posición del pack de munición

ID_enemy = ID del enemigo
Position_enemy = Posición del enemigo

Health_ally = Vida del aliado
Position_ally = Posición del aliado
*/




// TEAM AXIS (defensor) and ALLIED (atacante)
+flag (F)
    <-
    .print("Iniciando patrulla cerca de la base");
    !set_base_random_points_patroll.



+!set_base_random_points_patroll : team(200)
    <-
    ?flag(F);
    ?base(B);

```

```
    // Manual midpoint calculation
    .nth(0, F, FX);
    .nth(1, F, FY);
    .nth(2, F, FZ);

    .nth(0, B, BX);
    .nth(1, B, BY);
    .nth(2, B, BZ);

    MidX = (FX + BX) / 2;
    MidY = (FY + BY) / 2;
    MidZ = (FZ + BZ) / 2;

    MidPoint = [MidX, MidY, MidZ];
    +midpoint(MidPoint);
    .create_control_points(MidPoint, 20, 3, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).

+!set_base_random_points_patroll : team(100)
    <-
    ?flag(F);
    ?base(B);

    // Manual midpoint calculation
    .nth(0, F, FX);
    .nth(1, F, FY);
    .nth(2, F, FZ);

    .nth(0, B, BX);
    .nth(1, B, BY);
    .nth(2, B, BZ);

    MidX = (FX + BX) * 3 / 4;
    MidY = (FY + BY) * 3 / 4;
    MidZ = (FZ + BZ) * 3 / 4;

    MidPoint = [MidX, MidY, MidZ];
    +midpoint(MidPoint);
    .create_control_points(MidPoint, 20, 3, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).




/*
############################ Bucle del patrullaje ######################
Plan principal, Patrullar alrededor de una posición intermedia entre
la base y la flag para poder explorar el mapa y observar posibles enemigos o amigos
########################################################################
*/

+!do_patroll(Pos)
    <-
    -+control_points(Pos);
    .length(Pos, L);
    -+total_control_points(L);
    -+patrolling;
    -+patroll_point(0);
    .print("Iniciando patrulla cerca de los puntos", Pos).

// Manejo de los puntos de patrulla
+patroll_point(P): total_control_points(L) & P<L & control_points(Pos) & patrolling
    <-
    .nth(P,Pos,A);
```

```
    .print("Yendo al punto", A, "(", P, " de ", L, ")");
    .goto(A).

+patroll_point(P): total_control_points(T) & P==T & patrolling
    <-
    -+patroll_point(0);
    -total_control_points(T);
    .print("Patrullaje terminado, volviendo a patrullar");
    !!patroll_at_midpoint.

+target_reached(A): patrolling & patroll_point(P) & total_control_points(L) & P + 1 <= L & patrolling
    <-
    +san_pack(A);
    .cure;
    .print("Dejando cura preventiva y guardando su ubicación en", A);
    .get_backups;
    .wait(550);
    .print("Revisando si hay soldados en la zona");
    ?patroll_point(P);
    -+patroll_point(P+1);
    -target_reached(A);
    .print("Punto alcanzado ", A).

+!patroll_at_midpoint : midpoint(MidPoint)
    <-
    .create_control_points(MidPoint, 20, 3, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).

/*
############################ Fin del bucle del patrullaje ######################
Cada vez que llega a un punto del patrullaje, deja una cura preventiva
################################################################################
*/

/*############################ Recarga de munición y sanación ####################
Intenciones para recargar munición y sanación, se activan cuando el agente tiene poca vida o munición
################################################################################
*/

+!reload_ammo : ammo_pack(Pos_ammo_pack)
    <-
    .print("Yendo a recargar munición");
    .goto(Pos_ammo_pack).


+!reload_san : san_pack(Pos_san_pack)
    <-
    .print("Yendo a curarme al pack de la posición ", Pos_san_pack);
    .goto(Pos_san_pack).

/*
############################ Reacción a los packs ##############################
Cuando ve un pack de sanación o munición, se dirige a él si tiene poca vida o munición respectivamente
Si tiene suficiente vida o munición, guarda la posición del pack
para ir a por él más tarde a traves de crear una intención con !reload_san o !reload_ammo
################################################################################
*/

+packs_in_fov(_, TYPE_san_pack, _, _, _, Pos_san_pack) : TYPE_san_pack == 1001 & health(H) & H > 50 & not san_pack(Pos_san_pack)
    <-
```

```
    .print("Guardando posición del sanity pack ", Pos_san_pack);
    +san_pack(Pos_san_pack).


+packs_in_fov(_, TYPE_ammo_pack, _, _, _, Pos_ammo_pack) :TYPE_ammo_pack == 1002 &
    ammo(M) & M > 50 & not ammo_pack(Pos_ammo_pack)
    <-
    .print("Guardando posición de la ammo pack ", Pos_ammo_pack);
    +ammo_pack(Pos_ammo_pack).



/*############################ Activar intenciones cuando vida baja o munición baja
    ######################
##############################################################################################*/

+health(H):  H < 20 & not yendo_sanacion & san_pack(Pos_san_pack) & not al_ataque
    <-
    .print("Vida baja, yendo a curarme");
    +yendo_sanacion;
    ?position(LastPos);
    +posicion_anterior(LastPos);
    .look_at(Pos_san_pack);
    !!reload_san.

// si no hay pack de sanación, se crea una cura preventiva
+health(H): H < 20 & not yendo_sanacion & not san_pack(Pos_san_pack) & not al_ataque
    <-
    .print("Vida baja, creando cura preventiva y volviendo al punto inicial");
    .cure;
    +san_pack(Pos_san_pack);
    ?position(LastPos);
    +posicion_anterior(LastPos);
    .create_control_points(LastPos, 10, 2, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).



+ammo(M): M < 20 & not yendo_municion & ammo_pack(Pos_ammo_pack) & not al_ataque
    <-
    .print("Munición baja, yendo a recargar munición");
    +yendo_municion;
    ?position(LastPos);
    +posicion_anterior(LastPos);
    .look_at(Pos_ammo_pack);
    !!reload_ammo.

// si no hay pack de munición, se crea una cura preventiva
+ammo(M): M < 20 & not yendo_municion & not ammo_pack(Pos_ammo_pack) & not al_ataque
    <-
    .print("Munición baja, creando cura preventiva y volviendo al punto inicial");
    .cure;
    ?position(LastPos);
    +posicion_anterior(LastPos);
    .create_control_points(LastPos, 10, 2, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).



// Cuando llega a un pack de munición o sanación, lo recoge y vuelve a patrullar

+target_reached(Pos_san_pack): yendo_sanacion & posicion_anterior(LastPos)
    <-
    .print("Sanación cogida");
    -target_reached(Pos_san_pack);
```

```
    -yendo_sanacion;
    .create_control_points(LastPos, 10, 3, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).

+target_reached(Pos_ammo_pack): yendo_municion & posicion_anterior(LastPos)
    <-
    .print("Munición cogida");
    -target_reached(Pos_ammo_pack);
    -yendo_municion;
    .create_control_points(LastPos, 10, 3, BaseMidPointAll);
    !do_patroll(BaseMidPointAll).


/* ############################ Reacción a los enemigos ######################
Cuando ve un enemigo, se activa la intención de atacar o recargar munición
o curarse dependiendo de la vida y munición del agente
*/

// Cuando ve a un enemigo y tiene suficiente vida
+enemies_in_fov(ID_enemy,_,_,_,_,Position_enemy) : health(H) & H > 20 & ammo(M) & M >
    5
    <-
    .shoot(5,Position_enemy);
    .print("Atacando al enemigo ", ID_enemy).

+enemies_in_fov(ID_enemy,_,_,_,_,Position_enemy) : health(H) & H > 20 & ammo(M) & M <
    5 & ammo_pack(Pos_ammo_pack)
    <-
    .look_at(Pos_ammo_pack);
    +yendo_municion;
    !reload_ammo;
    .print("Yendo a recargar munición, a la posición", Pos_ammo_pack).

// Cuando ve a un enemigo pero tiene poca vida
+enemies_in_fov(_,_,_,_,_,_) : health(H) & H <= 20 & san_pack(Pos_san_pack)
    <-
    .look_at(Pos_san_pack);
    +yendo_sanacion;
    !reload_san;
    .print("Yendo a curarme, a la posición", Pos_san_pack).


// Cuando ve a un amigo herido
+friends_in_fov(_,_,_,_,Health_ally,Position_ally) :  Health_ally < 80 & not
    curando_ally
    <-
    +curando_ally;
    .print("Yendo a curar a aliado a la posición", Position_ally);
    .goto(Position_ally).

+target_reached(Position_ally): curando_ally
    <-
    -curando_ally;
    -target_reached(Position_ally);
    .cure;
    .print("Curando al aliado ");
    .create_control_points(Position_ally, 10, 2, Position_ally_grouped); // Crear
    puntos cercanos al jugador
    !do_patroll(Position_ally_grouped).


/*############################ Replanificación si no hay soldados
    ######################

Para el equipo 100, si hay soldados en la zona, se vuelve a patrullar
```

```
y se crea una cura preventiva en la posición del último punto de patrullaje.
Si no hay soldados, se activa estrategia de ataque y se va a por la bandera

Para el equipo 200, si hay soldados en la zona, se vuelve a patrullar
y se crea una cura preventiva en la posición del último punto de patrullaje.
Si no hay soldados, se activa estrategia de defensa y se va a patrullar por la bandera

*******************************************************************************/


+myBackups(LISTBACKUPS) : not flag_taken & team(100)
    <-
    .print("Recibiendo lista de backups");
    .print(LISTBACKUPS);
    .length(LISTBACKUPS, LENGTHBACKUPS);
    -patrolling;
    if (LENGTHBACKUPS == 0) {
        .print("No hay backups disponibles, a por la bandera!");
        +al_ataque;
        -myBackups(LISTBACKUPS);
        !to_atack;
    }
    else {
        .print("Hay backups disponibles, a seguir patrullando!");
        -myBackups(LISTBACKUPS);
        +patrolling;
    }.

+myBackups(LISTBACKUPS) : flag_taken & team(100)
    <-
    .print("Recibiendo lista de backups");
    .print(LISTBACKUPS);
    .length(LISTBACKUPS, LENGTHBACKUPS);
    -patrolling;
    !save_flag;
    .print("Guardando bandera en la base!");
    -myBackups(LISTBACKUPS).


+myBackups(LISTBACKUPS) : team(200)
    <-
    .print("Recibiendo lista de backups");
    .print(LISTBACKUPS);
    .length(LISTBACKUPS, LENGTHBACKUPS);
    if (LENGTHBACKUPS == 0) {
        .print("No hay backups disponibles, a defender la bandera!");
        +a_defender;
        -myBackups(LISTBACKUPS);
        !to_defend;
    }
    else {
        .print("Hay backups disponibles, a seguir patrullando!");
        -myBackups(LISTBACKUPS);
        +patrolling;
    }.


+!to_atack
    <-
    ?flag(F);
    .goto(F).

+!to_defend : flag(F) & team(200)
    <-
    .create_control_points(F, 10, 2, F_points);
```

```
    !do_patroll(F_points);
    .print("Defendiendo la bandera en", F_points).


+target_reached(F) : al_ataque
    <-
    .print("Bandera alcanzada, volviendo a base");
    -al_ataque;
    !save_flag.


// plan para guardar la bandera en la base
+!save_flag : team(100) & flag_taken
    <-
    ?base(B);
    .print("Guardando bandera en la base");
    .goto(B).
```

Listing 3: Código fuente del agente `bdimedic.asl`

A.4 Código del Agente Extra (Médico)

```
/*
 * ================================================
 * ESTRATEGIA DE AGENTE MÉDICO EXTRA PARA PYGOMAS
 * ================================================
 *
 * Agente médico que proporciona soporte curativo al equipo y
 * participa en combate secundariamente según la situación táctica.
 *
 * COMPORTAMIENTO GENERAL:
 * - Prioriza la curación de aliados con salud crítica (<50)
 * - Tiene comportamientos específicos según el equipo (Axis o Allied)
 * - Mantiene distancia segura de enemigos cercanos
 * - Dispara a enemigos cuando tiene suficiente munición
 * - Busca packs de munición cuando sus reservas son bajas
 * - Se cura a sí mismo cuando su salud es crítica
 *
 * EQUIPO AXIS (DEFENSIVO):
 * - Crea puntos de patrulla alrededor de la bandera
 * - Patrulla sistemáticamente estos puntos dejando packs médicos
 * - Protege el área de la bandera
 *
 * EQUIPO ALLIED (OFENSIVO):
 * - Va directamente a la bandera (punto estratégico donde estarán compañeros)
 * - Se dirige a la base cuando se captura la bandera
 * - Prioriza estar cerca del objetivo sobre seguir a soldados específicos
 *
 * GESTIÓN DE RECURSOS:
 * - Cura proactivamente cuando su salud es baja (<40)
 * - Busca munición cuando está por debajo del 40%
 * - Responde tácticamente a enemigos según distancia y salud
 *
 * PATRONES DEFENSIVOS:
 * - Retrocede y cambia dirección ante enemigos cercanos
 * - Dispara a enemigos lejanos si tiene munición suficiente
 * - Prioriza curación sobre combate
 */

// Umbral para considerar mala salud propia
health_threshold(40).

// === INICIALIZACIÓN ===

// EQUIPO AXIS - DEFENSIVO: Patrullar cerca del objetivo
```

```
+flag(F): team(200)
  <-
  .print("Inicializando medico defensor");
  // Crear puntos de patrulla mas cercanos al flag (radio pequeño, mas puntos)
  .create_control_points(F, 20, 6, C);
  +control_points(C);
  .length(C, L);
  +total_control_points(L);
  +patrolling;
  +patrol_point(0);
  ?control_points(C);
  .nth(0, C, Position);
  .goto(Position).

// === COMPORTAMIENTO DE EQUIPO ALLIED ===

// EQUIPO ALLIED - OFENSIVO: Ir directamente a por la bandera,
// ya que es donde seguramente irán los compañeros
+flag(F): team(100)
  <-
  .print("Inicializando medico atacante, yendo hacia bandera");
  +objective(F);
  .goto(F).

// Si se da el caso que la bandera es capturada -- volver base
+flag_taken: team(100)
  <-
  .print("¡BANDERA CAPTURADA! Volviendo a base");
  ?base(B);
  +returning;
  .goto(B).

// Detectar soldados en el campo de visión y seguirlos -- pierde el foco de ir a la
    bandera
// puede que con un equipo con más soldados mejor descomentar
//+friends_in_fov(ID, Type, Angle, Distance, Health, Position): Type == 1 & team(100)
// <-
//.print("Soldado detectado en FOV, siguiendolo");
//.goto(Position).


// === PATRULLAJE (EQUIPO AXIS) ===

// Llegar a un punto de patrulla
+target_reached(T): patrolling & team(200) & patrol_point(P) & total_control_points(TP
    ) & P + 1 < TP
  <-
  // Dejar un pack médico en cada punto de patrulla
  .print("Punto de patrulla alcanzado, dejando pack medico");
  .cure;
  // Actualizar al siguiente punto
  -+patrol_point(P + 1);
  // Ir al siguiente punto
  ?control_points(CP);
  ?patrol_point(NP);
  .nth(NP, CP, NextPosition);
  .goto(NextPosition);
  -target_reached(T).

// Llegar al último punto de patrulla
+target_reached(T): patrolling & team(200) & patrol_point(P) & total_control_points(TP
    ) & P + 1 >= TP
  <-
  // Dejar un pack médico en cada punto de patrulla
  .print("Ultimo punto de patrulla alcanzado, reiniciando");
```

```
  .cure;
  // Volver al primer punto
  -+patrol_point(0);
  // Ir al primer punto
  ?control_points(CP);
  ?patrol_point(NP);
  .nth(NP, CP, NextPosition);
  .goto(NextPosition);
  -target_reached(T).

// === COMPORTAMIENTO DEFENSIVO ===

// Ver enemigos muy cerca - retroceder y curar
+enemies_in_fov(ID, Type, Angle, Distance, Health, Position): Distance < 15
  <-
  .print("Enemigo cerca, manteniendo distancia y creando packs");
  .cure;
  // Cambiar dirección de movimiento
  .turn(0.5).

// Ver enemigos a distancia segura - solo curar
+enemies_in_fov(ID, Type, Angle, Distance, Health, Position): Distance >= 40 & Health
    >= 30
  <-
  .print("Enemigo a distancia segura, creando pack medico");
  .cure.

// Ver enemigos a distancia segura - curar y dispararles
+enemies_in_fov(ID, Type, Angle, Distance, Health, Position): Distance >= 15  & ammo(A
    ) & A > 5
  <-
  .print("Enemigo detectado, disparando!");
  .cure;
  .shoot(5, Position).

// === DETECCIÓN Y CURACIÓN PROACTIVA ===

// Ver aliados con poca vida - PRIORIDAD ALTA
+friends_in_fov(ID, Type, Angle, Distance, Health, Position): Health < 50
  <-
  .print("Aliado con salud critica detectado! ID: ", ID, " Salud: ", Health);
  .goto(Position);
  .cure.

// Cuando estamos cerca de un aliado, crear pack médico proactivamente -- es despista
    si just esta a la bandera
//+friends_in_fov(ID, Type, Angle, Distance, Health, Position): Distance < 5
//<-
//.print("Aliado cercano, creando pack medico preventivo");
//.cure.

// === AUTO-PRESERVACIÓN ===

// Si tenemos poca vida, curarnos y buscar refugio
+health(H): health_threshold(T) & H < T
  <-
  .print("Salud critica! Curandome");
  .cure;
  // Retirarse a una posicion segura temporalmente
  .turn(1.0);
  .wait(1000).

// Ver packs de munición cuando tenemos poca munición
+packs_in_fov(ID, 1002, Angle, Distance, Health, Position): ammo(A) & A <= 40 & not
    yendo_municion
```

```
  <-
  .print("Pack de municion detectado! Yendo a por el");
  +yendo_municion;
  .goto(Position).

// Cuando llegamos al pack de munición
+target_reached(T): yendo_municion
  <-
  .print("He recargando balas, vamos a curar!");
  -yendo_municion;
  -target_reached(T).
```

Listing 4: Código fuente del agente `bdimedic_extra.asl`