**MEMORIA PRÁCTICA 1 AIN**

**1. INTRODUCCIÓN Y OBJETIVOS**

El presente proyecto tiene como objetivo principal el desarrollo de un sistema multi-agente inteligente para simulaciones de combate utilizando la plataforma PyGOMAS. El sistema implementa agentes autónomos con capacidades de coordinación, comunicación y toma de decisiones estratégicas en tiempo real.

Los objetivos específicos del proyecto incluyen:

* Implementar un sistema de coordinación jerárquica entre agentes
* Desarrollar servicios especializados para diferentes tipos de unidades
* Mejorar los comportamientos de combate para evitar fuego amigo
* Crear funciones personalizadas en Python para cálculos tácticos avanzados

**2. METODOLOGÍA Y ARQUITECTURA DEL SISTEMA**

**2.1 Arquitectura Multi-Agente**

El sistema está basado en la arquitectura BDI (Beliefs, Desires, Intentions) proporcionada por PyGOMAS, extendida con funcionalidades personalizadas. Se han implementado tres tipos de agentes especializados:

* **BDISuperSoldier**: Unidad de combate principal con capacidades de liderazgo
* **BDIMedic**: Unidad médica especializada en curación y soporte
* **BDIFieldOp**: Unidad de apoyo logístico y suministro de munición

**2.2 Lenguaje de Programación**

El proyecto utiliza dos lenguajes complementarios:

* **Python**: Para la implementación de la clase base y funciones matemáticas complejas
* **AgentSpeak**: Para la definición de comportamientos reactivos y planes de los agentes

**3. IMPLEMENTACIÓN TÉCNICA**

**3.1 Clase Python Personalizada (soldier.py)**

Se ha desarrollado la clase BDISuperSoldier que extiende BDISoldier con las siguientes funciones personalizadas:

**Función de Flanqueo Táctico:**

(Python)

@actions.add\_function(".calculate\_flanking\_position", (tuple, tuple, float))

Esta función calcula posiciones de flanqueo óptimas utilizando trigonometría avanzada. Recibe la posición del agente, la posición del enemigo y la distancia deseada, retornando coordenadas que permiten atacar desde un ángulo perpendicular.

**Función de Movimiento Circular:**

(Python)

@actions.add\_function(".circle", (tuple, tuple, float))

Implementa un algoritmo de movimiento circular alrededor de un punto específico, incrementando el ángulo en 15 grados por cada llamada para crear patrones de movimiento fluidos.

**Función de Verificación de Distancia Segura:**

(Python)

@actions.add\_function(".safe\_distance\_check", (tuple, tuple, float))

Calcula la distancia euclidiana entre dos puntos y verifica si supera un umbral mínimo de seguridad, esencial para evitar fuego amigo.

**3.2 Sistema de Coordinación (bdisoldier.asl)**

**Liderazgo Jerárquico:**  
El sistema implementa un mecanismo de elección de líder dinámico. El primer agente en activarse registra el servicio "team\_leader" y coordina las acciones del resto del equipo.

(ASL)

+flag (F): team(100)

<-

.get\_service("team\_leader");

.wait(1000);

if(team\_leader(L)){

+following\_leader(L)

}

else{

.register\_service("team\_leader");

+i\_am\_leader

};

**Comunicación Táctica:**  
El líder envía órdenes específicas a cada tipo de unidad:

* Soldados: Órdenes de formación y posicionamiento
* Médicos: Posiciones de apoyo médico
* FieldOps: Coordenadas para apoyo logístico

**3.3 Servicios Especializados**

**Servicios Médicos (bdimedic.asl):**

* emergency\_support: Atención médica prioritaria en combate
* combat\_medic: Soporte médico especializado en zona de guerra

**Servicios Logísticos (bdifieldop.asl):**

* ammo\_specialist: Suministro especializado de munición
* tactical\_support: Apoyo táctico con reabastecimiento

**Servicios de Reconocimiento (bdisoldier.asl):**

* scout: Misiones de exploración y reconocimiento
* team\_leader: Coordinación y liderazgo del equipo

**3.4 Comportamientos Avanzados de Combate**

**Evitar Fuego Amigo:**  
Se ha implementado un sistema multicapa para prevenir daño a aliados:

1. **Verificación de línea de fuego**: Compara ángulos entre enemigos y aliados
2. **Cálculo de distancia segura**: Utiliza la función safe\_distance\_check
3. **Maniobras evasivas**: Flanqueo táctico o movimiento circular según la situación

text

+enemies\_in\_fov(IDE,TypeE,AngE,DistanceE,HealthE,[Xe, Ye, Ze]):

friends\_in\_fov(IDA,TypeA,AngA,DistanceA,HealthA,[Xa, Ya, Za]) & position([Xs, Ys, Zs])

<-

.safe\_distance\_check([Xs, Ys, Zs], [Xa, Ya, Za], 10, SafeDistance);

if(AngA == AngE & SafeDistance == false){

.calculate\_flanking\_position([Xs, Ys, Zs], [Xe, Ye, Ze], 20, FlankPos);

.goto(FlankPos)

}

**Gestión Inteligente de Recursos:**  
El sistema diferencia entre situaciones de combate y no combate para optimizar las peticiones de recursos:

* En combate: Prioriza servicios de emergencia
* Fuera de combate: Utiliza servicios estándar