Evidencia 2 - Revisión 3

Team members:

Andrés Gallego López - A1645740

Ali Lopez Sarabia - A01645060

Sebastian Alett Oliva Aranda - A01645073

Mauricio Gael Villalobos Aguayo - A01644972

Sebastián Alejandro Veilleux Amaya - A01644977

Guillermo Sainz Lizárraga - A01644854

29/11/2025

Diagramas UML y AUML finales.

- ## Plan de Trabajo

1. Agente Manager (Decisor-Planificador)

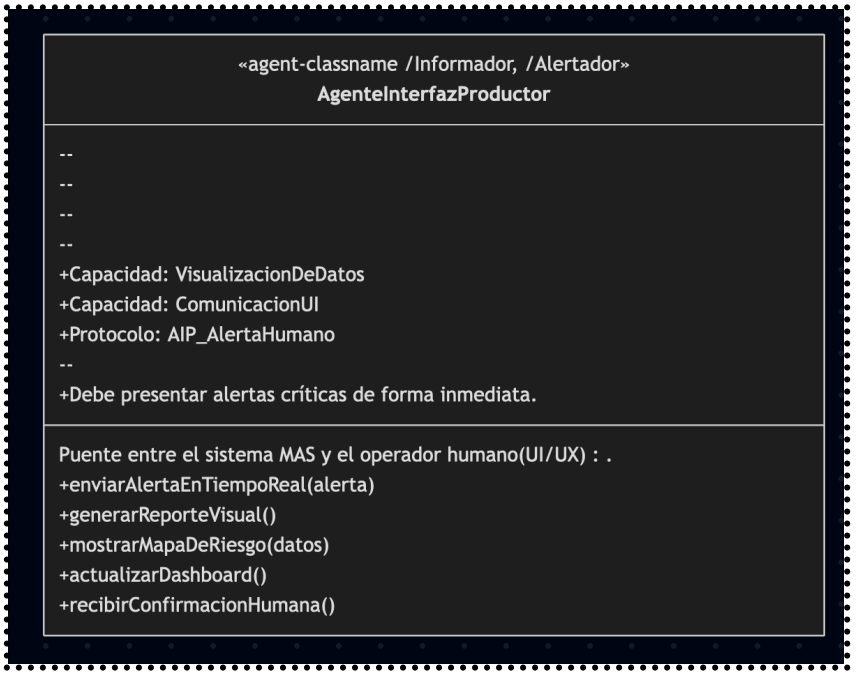
Descripción: Este es el diagrama de clase para el agente lógico central. Define sus roles como /Decisor y /Planificador, sus capacidades de análisis y los protocolos que utiliza para recibir datos y enviar órdenes.

2. Agente Físico Unificado (Explorador-Ejecutor-Sensor)

Descripción: Este diagrama modela al agente físico. Sus tres roles principales (/Explorador, /Sensor, /Ejecutor) están claramente definidos. Nota cómo sus capacidades (Navegación, Sensado) y acciones (recorrer, aplicarTratamiento) reflejan directamente sus funciones en el invernadero.

3. Agente de Interfaz con el Productor (UI/UX)

Descripción: Este diagrama representa al agente que se comunica con el humano. Es más simple, enfocado en roles de /Informador y /Alertador, y sus acciones son para generar visualizaciones y alertas.

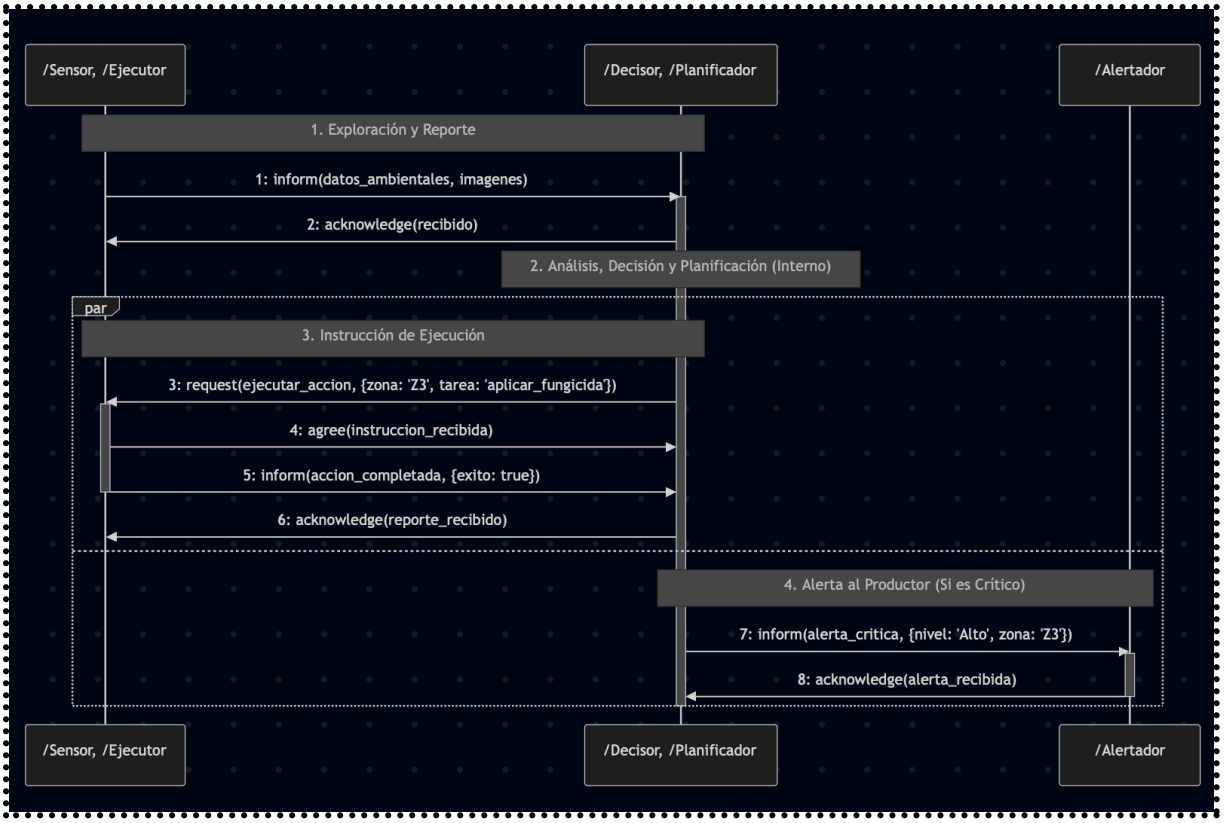


- Diagramas de Interacción (AIP) sobre todas las interacciones que consideren en su simulación.

Descripción del Diagrama

El flujo se divide en cuatro fases principales:

1. **Exploración y Reporte:** El AgenteFisicoUnificado (en su rol de /Sensor) envía proactivamente los datos del invernadero al AgenteManager.
2. **Análisis y Decisión:** El AgenteManager (como /Decisor) procesa esta información (esto se representa por su barra de activación continua).
3. **Ejecución y Alerta (Paralelo):** Una vez tomada la decisión, el AgenteManager (como /Planificador) inicia dos hilos de conversación en paralelo (usando el bloque par):
    - **Instrucción de Ejecución:** Envía una orden al AgenteFisico (en su rol de /Ejecutor) para que realice una tarea física.
    - **Alerta al Productor:** Si el riesgo es crítico, simultáneamente informa al AgenteInterfaz (como /Alertador) para que notifique al usuario.



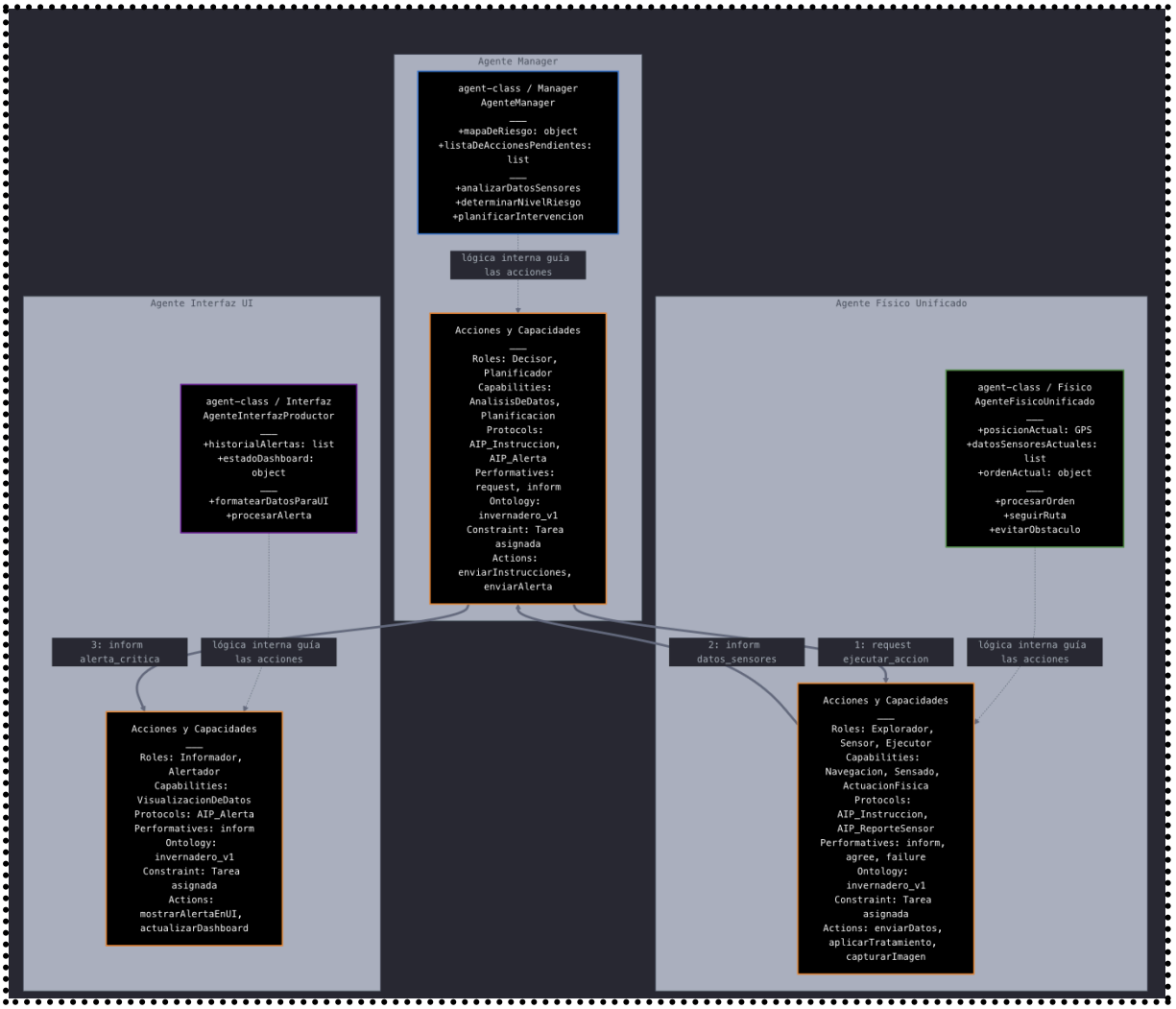
- Diagramas de clase estándar para describir los subsistemas de los agentes.

Descripción: Para cada agente, se separa en dos componentes:

1. **«agent-class» (El Pensamiento):** Este es el recuadro que simula una clase UML. Contiene los atributos internos (estado, memoria, datos) y los métodos internos (lógica, procesamiento, "pensamiento"). Representa la lógica deliberativa del agente.
2. **«Acciones y Capacidades» (La Interfaz al Entorno):** Este es el recuadro externo. Define los roles que el agente puede jugar, los protocolos que entiende, y lo más importante, las acciones que puede iniciar y que afectan a otros agentes o al entorno.

El diagrama muestra cómo el AgenteManager inicia la comunicación enviando un request al AgenteFisico y un inform de alerta al AgenteInterfaz. A su vez, el AgenteFisico envía un inform de vuelta al Manager con los datos de sus sensores.

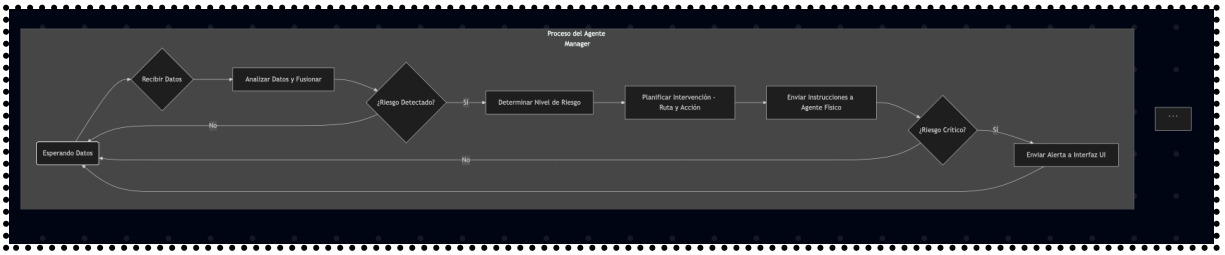
- **Comportamiento de los subsistemas de los agentes.**



  - Diagramas de Actividad o de Estado para describir el comportamiento de los subsistemas de los agentes.

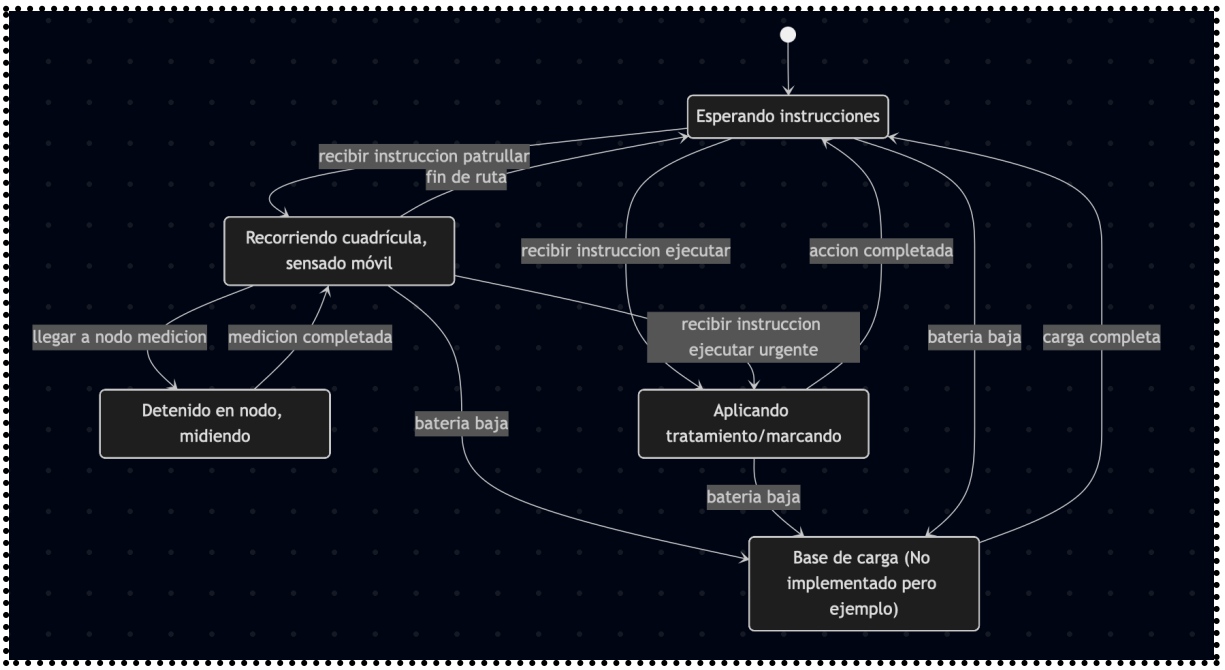
1. Diagrama de Actividad: Proceso del Agente Manager

Descripción: Este diagrama de actividad (presentado como un diagrama de flujo) describe el bucle de procesamiento principal del AgenteManager. Muestra el flujo lógico desde que recibe datos hasta que decide si debe actuar, alertar, o simplemente seguir esperando.



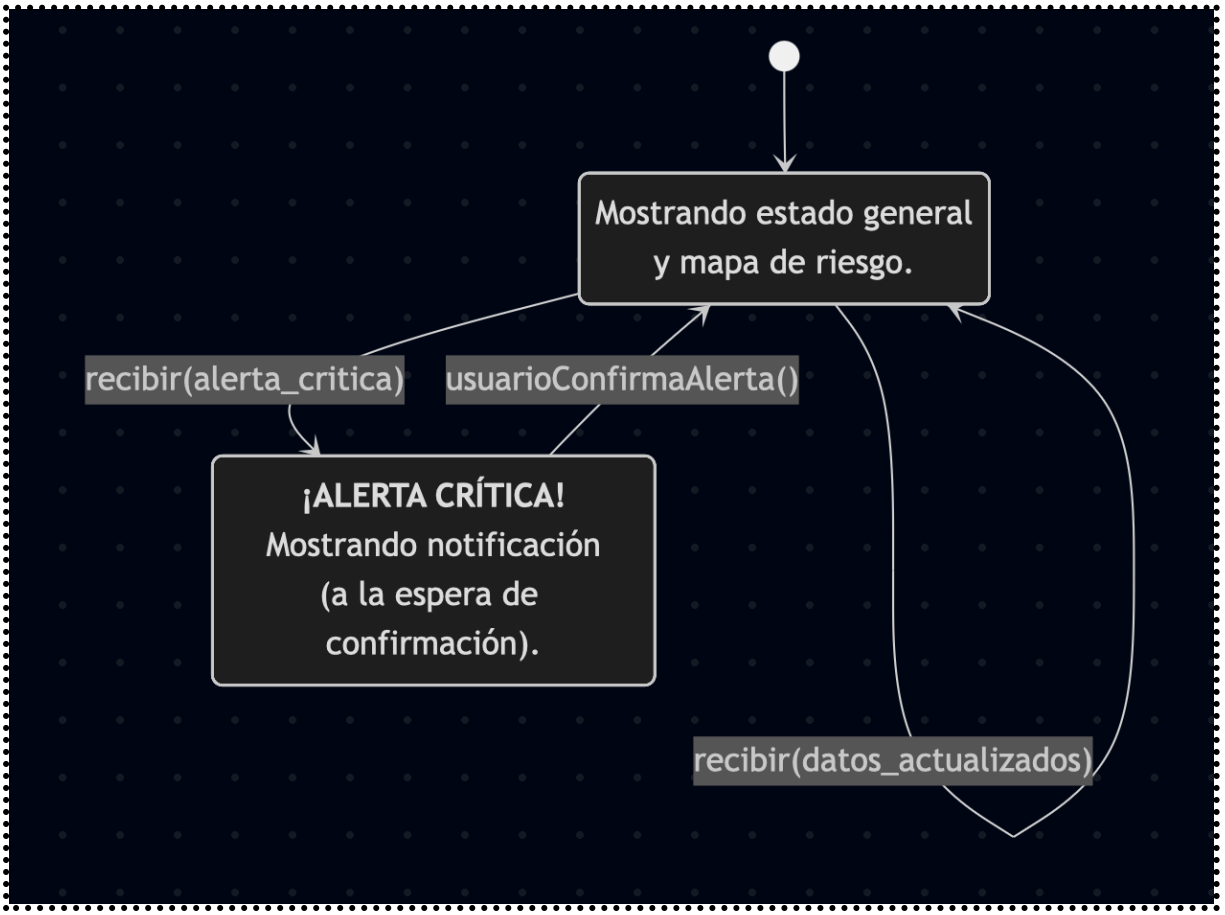
2. Diagrama de Estado: Estados del Agente Físico Unificado

Descripción: Este diagrama de estado modela los diferentes estados en los que puede encontrarse el AgenteFisicoUnificado. Es crucial para entender cómo el agente maneja diferentes tareas concurrentes (como explorar y luego ser interrumpido por una orden de ejecución).



3. Diagrama de Estado: Estados del Agente Interfaz con el Productor (UI/UX)

Descripción: Este diagrama de estado modela el comportamiento del Agente de Interfaz. Es un agente relativamente simple cuyo estado está determinado principalmente por la información que recibe del AgenteManager.



Código de la Implementación de los Agentes

Se han desarrollado e integrado los scripts para los tres agentes principales definidos en la arquitectura del sistema:

- **Agente Físico (Explorador/Ejecutor):**
  - ***Implementación:*** Script `AgenteFisico.cs` y Canvas de Unity.
  - ***Funcionalidad:***
    - Navegación autónoma (búsqueda de objetivos `FindObjectsByType`), sistema de sensores mediante `Raycast`, toma de decisiones local (detenerse, analizar) y ejecución de acciones (marcar planta como tratada/enferma).
    - Despliegue de datos en tiempo real (`MostrarAnalisisUI`) y alertas críticas (`RegistrarAlerta`) para el usuario humano.

```
public class AgenteFisico : MonoBehaviour
{
    [Header("Configuración Vuelo")]
    public float velocidad = 3.0f;        // Velocidad del dron
    public float alturaVuelo = 1.5f;      // Altura a la que vuela sobre
las plantas
    public float distanciaParaParar = 0.5f; // Qué tan cerca se detiene
del tomate

    [Header("Configuración Análisis")]
    public float tiempoAnalisis = 2.0f;  // Cuánto tarda en analizar
    public Transform puntoSensor;

    private List<PlantaData> listaDeTomates;
    private int indiceActual = 0;
    private bool estaViajando = true;
    private bool tareaCompletada = false;

    void Start()
    {
        // Encuentra todos los objetos que tengan el script PlantaData
        var todosLosTomates = FindObjectsOfType<PlantaData>();

        //  Los ordena por cercanía para crear una ruta lógica
        listaDeTomates = todosLosTomates.OrderBy(t =>
Vector3.Distance(transform.position, t.transform.position)).ToList();

        Debug.Log($"[DRON] Ruta calculada: {listaDeTomates.Count}
tomates encontrados.");
    }

    void Update()
    {
        // Si terminamos o estamos parados analizando, no hacer nada
        if (tareaCompletada || !estaViajando || listaDeTomates.Count ==
0) return;

        // Obtener el objetivo actual de la lista
        PlantaData objetivo = listaDeTomates[indiceActual];
```

```
        // Definir el punto de destino (misma posición X/Z del tomate,
pero a nuestra altura Y)
        Vector3 destino = new Vector3(objetivo.transform.position.x,
alturaVuelo, objetivo.transform.position.z);

        // Moverse hacia el destino
        transform.position = Vector3.MoveTowards(transform.position,
destino, velocidad * Time.deltaTime);

        // Mirar hacia el destino
        transform.LookAt(destino);

        // Calcular distancia
        float distancia = Vector3.Distance(transform.position,
destino);

        // Si estamos cerca, iniciar análisis
        if (distancia < distanciaParaParar)
        {
            StartCoroutine(AnalizarRutina(objetivo));
        }
    }

    IEnumerator AnalizarRutina(PlantaData planta)
    {
        estaViajando = false;


        if (planta.tienePlaga)
        {
            planta.MarcarComoEnferma();
            if (AgenteManager.Instance != null)

AgenteManager.Instance.RegistrarAlerta(planta.nombreComun);
        }
        else
        {
            planta.MarcarComoSana();
        }

        // Comunicar estado normal
        if (AgenteManager.Instance != null)
            AgenteManager.Instance.MostrarAnalisisUI(planta);

        // Dibujar láser
        if (puntoSensor != null)
            Debug.DrawLine(puntoSensor.position,
planta.transform.position, Color.red, tiempoAnalisis);

        // Ahora sí, esperar simulando recolección de datos detallados
        yield return new WaitForSeconds(tiempoAnalisis);

        planta.yaAnalizada = true;
        indiceActual++;

        if (indiceActual < listaDeTomates.Count)
        {
            estaViajando = true;
```

```
        }
        else
        {
            Debug.Log("[DRON] Misión Completa.");
            tareaCompletada = true;
        }
    }
}
```

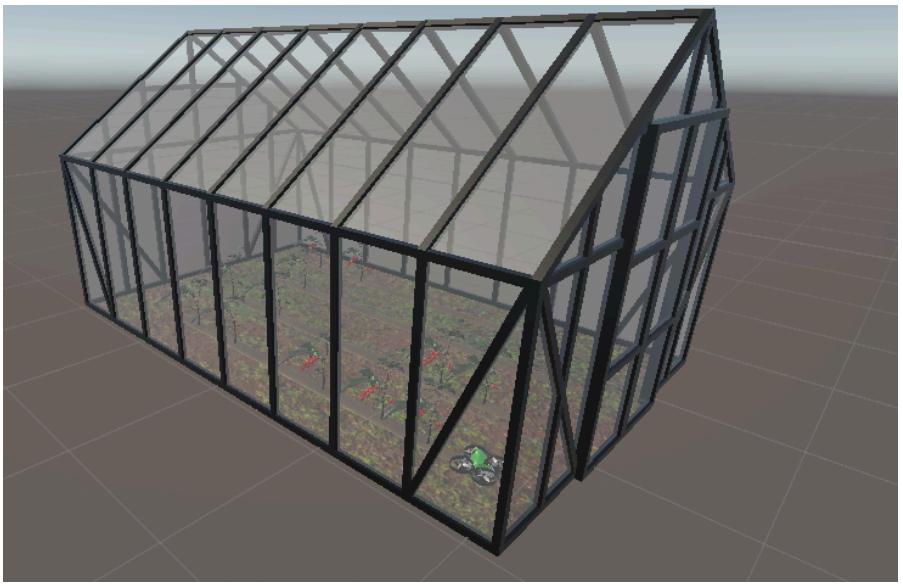
- **Agente Manager (Cerebro/Decisor) y de Interfaz (Visualización):**
  - ***Implementación**:* `AgenteManager.cs`.
  - ***Funcionalidad**:* Centralización de datos (patrón Singleton), procesamiento de reportes recibidos del agente físico y generación de alertas basadas en lógica de negocio (detección de plaga).

```
public class AgenteManager : MonoBehaviour
{
    public static AgenteManager Instance;

    [Header("Referencias UI")]
    public TextMeshProUGUI textoEstado;
    public TextMeshProUGUI textoAlertas;
    private int alertasCount = 0;

    void Awake()
    {
        if (Instance == null) Instance = this;
        else Destroy(gameObject);
    }

    public void MostrarAnalisisUI(PlantaData planta)
    {
        if(textoEstado != null)
        {
            string estado = planta.tienePlaga ? "INFECTADA" : "SANA";
            textoEstado.text = $"MONITOREO EN VIVO\n" +
                               $"Target: {planta.nombreComun}\n" +
                               $"Madurez: {planta.nivelMaduracion:F1}/10\n" +
                               $"Salud: {estado}";
        }
    }

    public void RegistrarAlerta(string nombrePlanta)
    {
        alertasCount++;
        if(textoAlertas != null)
        {
            textoAlertas.text = $"¡ALERTA DE PLAGA!\n" +
                                $"Ubicación: {nombrePlanta}\n" +
                                $"Contador Plagas: {alertasCount}";
            textoAlertas.color = Color.red;
        }
    }
}
```

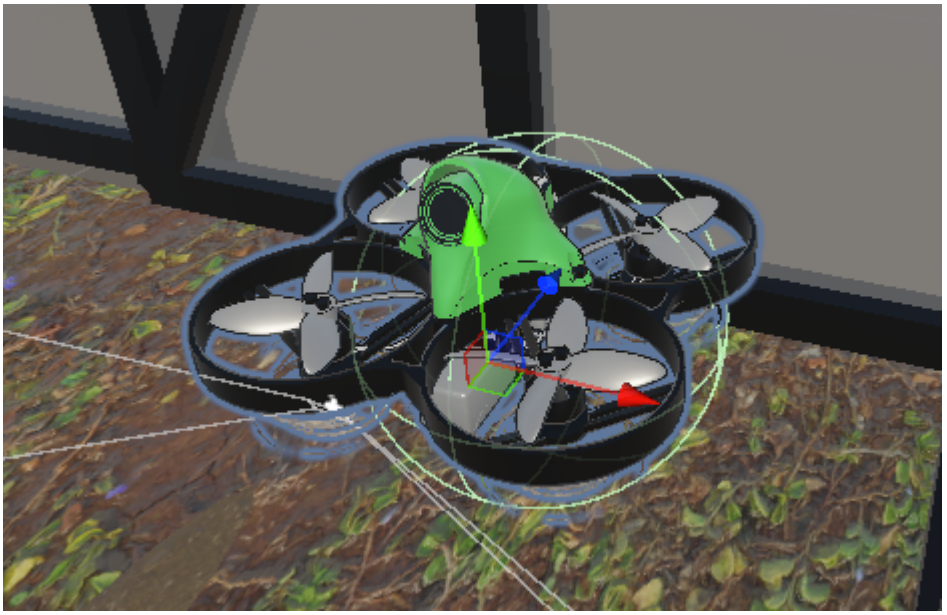
Código y Modelos de la Parte Gráfica

La simulación visual ha pasado de ser un concepto a una ejecución en un motor gráfico (Unity):

- **Entorno Virtual:** Se cuenta con un invernadero modelado con hileras de cultivos y estructura física.



- **Modelos 3D:**
    - ***Dron**:* Modelo `Drone_Cartoon` animado y funcional.



- ***Cultivos**:* Modelos de tomate con capacidad de cambio de estado visual (Color/Textura) mediante el script PlantaData.cs.

```
using UnityEngine;

public class PlantaData : MonoBehaviour
{
    [Header("Datos Identificación")]
    public string nombreComun = "";

    [Header("Datos Biológicos")]
    public bool tienePlaga = false;
    [Range(0, 100)] public float humedad = 65.0f;
    [Range(0, 10)] public float nivelMaduracion = 5.0f;

    [HideInInspector] public bool yaAnalizada = false;

    [Header("Configuración Visual")]
    public Color colorSano = new Color32(255, 4, 0, 255);
    public Color colorEnfermo = new Color32(133, 111, 20, 255);

    private Renderer miRenderer;

    void Start()
    {
        // Generar nombre automático si está vacío
        if (string.IsNullOrEmpty(nombreComun))
        {
            nombreComun = $"Tomate {Random.Range(100, 999)}";
        }

        // Buscar el pintor automáticamente
        miRenderer = GetComponent<Renderer>();

        // Aplicar color inicial
        if (miRenderer != null) ActualizarColor();
    }

    public void MarcarComoEnferma()
    {
        tienePlaga = true;
        ActualizarColor();
    }

    public void MarcarComoSana()
    {
        tienePlaga = false;
        ActualizarColor();
    }

    void ActualizarColor()
    {
        if (miRenderer != null)
        {
            miRenderer.material.color = tienePlaga ? colorEnfermo :
colorSano;
        }
```

```
    }
}
```

- **Interfaz de Usuario (UI):** Sistema de Canvas "World Space" o "Overlay" que muestra métricas en tiempo real (Objetivo, Madurez, Diagnóstico) y alertas visuales.
- **Cámara:** Script `CameraFollow.cs` implementado para seguimiento cinematográfico del agente, mejorando la presentación visual.

```
public class CameraFollow : MonoBehaviour
{
    public Transform target;

    [Header("Ajustes de Cámara")]
    public Vector3 offset = new Vector3(0, 10, -8);
    public float smoothSpeed = 5.0f; // Velocidad de seguimiento
    public float anguloFijo = 50.0f; // Ángulo fijo hacia abajo (Vista
aérea)

    void LateUpdate()
    {
        if (target == null) return;

        // Calcular posición deseada
        Vector3 desiredPosition = target.position + offset;

        // Moverse suavemente
        Vector3 smoothedPosition = Vector3.Lerp(transform.position,
desiredPosition, smoothSpeed * Time.deltaTime);
        transform.position = smoothedPosition;

        // Rotación Fija
```

```
        transform.rotation = Quaternion.Euler(anguloFijo, 0, 0);
    }
}
```

Simulación

Revision3_Simulacion.mp4

- **Plan de trabajo y aprendizaje adquirido.**

Scripts

Tarea	**Rol en el Sistema**	**Asignado**	**Descripción Detallada (Unity C#)**
Script del Agente Manager (Cerebro Central)	Agente Manager (Decisor–Planificador)	**Vini**	Crear el script ManagerAgent.cs (usualmente en un GameObject vacío). **Funciones:** 1. **Recibir Datos:** Leer variables públicas de los robots (humedad detectada, imagen). 2. **Evaluar Riesgo:** Implementar la

			lógica que decide si un valor es "Plaga" o "Normal". 3. **Asignar Tareas:** Indicar a qué coordenadas (Vector3) debe ir el robot. 4. **Coordinación:** Mantener el estado global del cultivo (matriz de datos).
Script del Agente Físico (Controlador del Robot)	Agente Explorador–Ejecutor–Sensor	**Willy**	Crear el script RobotController.cs . **Funciones:** 1. **Movimiento:** Implementar desplazamiento (usando *NavMesh* o movimiento por Grid) hacia el destino dado por el Manager. 2. **Sensado:** Simular sensores usando *Raycasts* o *Triggers*(Colisionadores) para detectar plantas y

			obstáculos. 3. **Ejecución:** Cambiar visualmente la planta (ej. aplicar spray) al recibir la orden. 4. **Evitación:** Detectar otros robots para no chocar (frenado local).

Entorno

Tarea	Modelo 3D / Componente	Asignado	Descripción Detallada (Unity Editor)
Modelado del Invernadero e Iluminación	Escenario Base	**Veilleux**	**Construcción:** Crear el piso, paredes (material transparente/vidrio) y puerta. **Iluminación Global:** Configurar la Directional Light (Sol) para que el ambiente se vea realista. **Iluminación Local:** Implementar luces puntuales (tipo sirena) que puedan ser activadas/desactivadas por código sobre los robots.
Modelado y Animación del Robot	Agente Físico (Visual)	**LeCoquete**	**Prefab del Robot:** Modelar el dron/robot terrestre con sus ruedas/hélices. **Animación:** Crear clips de animación (Idle, Moviéndose, Aplicando

			Tratamiento). **Integración:** Asegurar que el modelo tenga la jerarquía correcta para recibir el script de Willy.
Configuración de Cultivos y Físicas	Elementos Interactivos	**Ali**	**Prefabs de Plantas:** Crear el modelo de la planta de (ji)tomate y el estante/fila. **Instanciación:** Generar el cultivo completo (grid de plantas) al inicio de la escena. **Colliders:** Configurar los *Box Colliders* y *Tags* correctos ("Planta", "Obstaculo") para que los sensores de Willy las detecten.

UI

Tarea	Rol en el Sistema	Asignado	Descripción Detallada (Unity UI)
Interfaz Gráfica de Reporte y Análisis	Agente de Interfaz con el Productor	**LaFlame**	Crear el **Canvas de UI** y el script UIManager.cs. **Funciones:** 1. **Visualización de Alertas:** Cuando Coqueto detecte una plaga, desplegar un icono de alerta sobre la planta afectada en el mundo 3D (World Space UI) o en pantalla. 2. **Panel de Métricas:** Mostrar en tiempo real: Tiempo transcurrido, % de cultivo revisado y Nivel de riesgo actual. 3. **Análisis Final:** Al terminar la simulación, mostrar una pantalla de "Resumen" con los

			datos para la optimización.