Inteligencia Artificial para videojuegos

PAC 3 - Aprendizaje en videojuegos

Jon Gómez Palomar

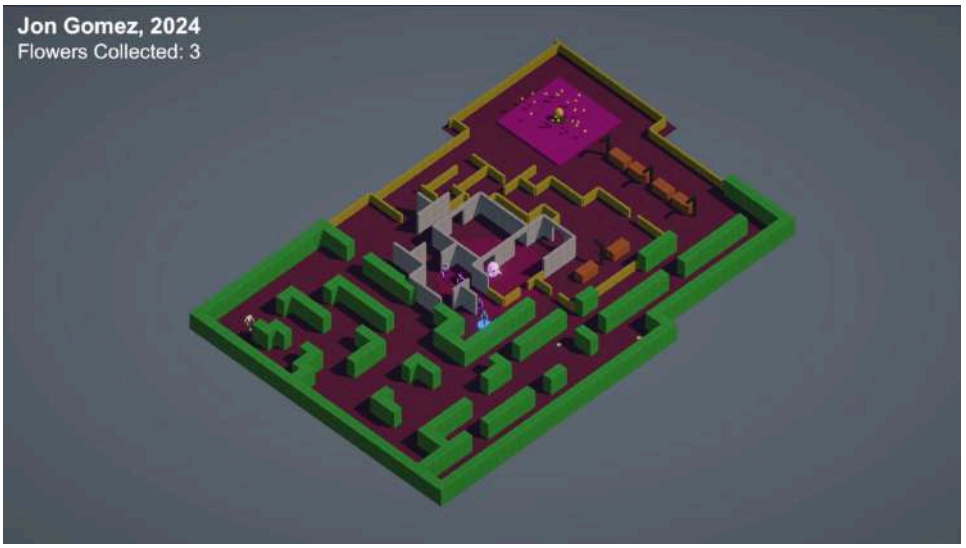
Universitat Oberta de Catalunya

Índice

Presentación general

El propósito de esta actividad es hacer una puesta de contacto con los agentes de aprendizaje autónomo (ml-agents) de Unity. Esta actividad ha sido la más compleja con diferencia, y ya voy adelantando que los resultados no son los mejores, pero por limitaciones de tiempo y hardware no he podido dar más de mí con este proyecto. Aun así, estoy orgulloso de lo aprendido y espero que este documento lo refleje correctamente.

Puntualizar que, respecto a la ACT3, algunos de los otros agentes funcionan un poco peor debido al cambio de versión que he tenido que hacer para usar ml-agents (de 2022 a 2023).



https://youtu.be/Hu58-CBT_jo
Repositorio GitLab: https://gitlab.com/jongompal/artificial-intelligence-unity

Estructura del repositorio

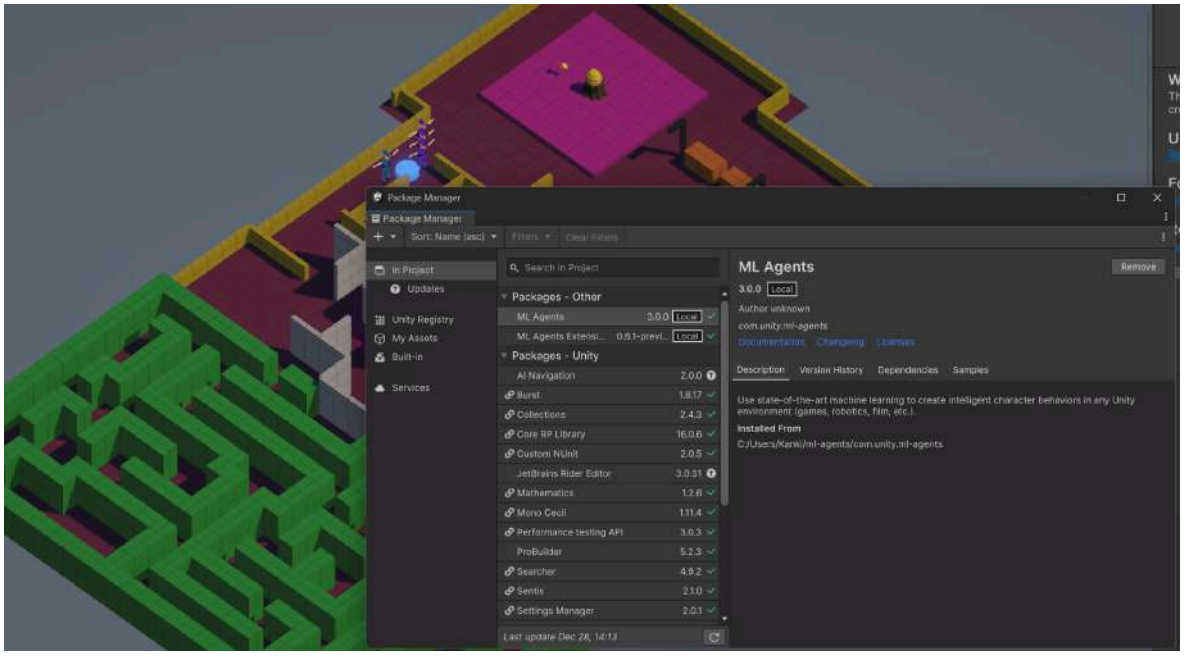
- **ACT2:** proyecto Unity de la actividad 2.
- **Artificial Intelligence:** proyecto Unity con la actividad 3 y la PAC1 y 2.
  - La carpeta assets incluye todo el contenido de las actividades anteriores. Para la actividad 3 se han añadido archivos en Scripts, scenes, prefabs y creado la carpeta ml-agents con todo lo relacionado con este tema (incluido los archivos .onnx que merece la pena utilizar)
  - La escena “final” es la que se llama Main V3-ML
- **Build:** intento de build de servidor para entrenamiento en paralelo
- **ml-agents:** repositorio de Unity para ejecutar los entornos de entrenamiento, con todas las configuraciones (.yaml) y los resultados de los múltiples (muchísimos) intentos. Disponibles en las carpetas config y results respectivamente.

ACT 1: instalación de ML-Agents

Antes que nada he instalado Python 3.10.12 del repositorio PythonWindows porque oficialmente solo hay disponibles instaladores compilados para la 3.10.11 y la guía de Unity recomienda la .12. Después, he instalado el entorno Anaconda y creado una máquina virtual con la versión de Python con el código `conda create -n mlagents python=3.10.12 && conda activate mlagents` en el terminal Anaconda Prompt.

En este punto me he puesto a seguir la guia de instalación del profesor (Aprenentatge per reforç amb ML-Agents | gameAIUnity) para instalar torch y el entorno de Unity.

Por último, he instalado los paquetes del repositorio clonado a mi proyecto de Unity.



ACT 2: Roller Ball tutorial

La actividad originalmente la creé en una escena aparte en el proyecto principal de las PEC, pero posteriormente lo he añadido también como proyecto de Unity independiente en la ruta del repositorio.

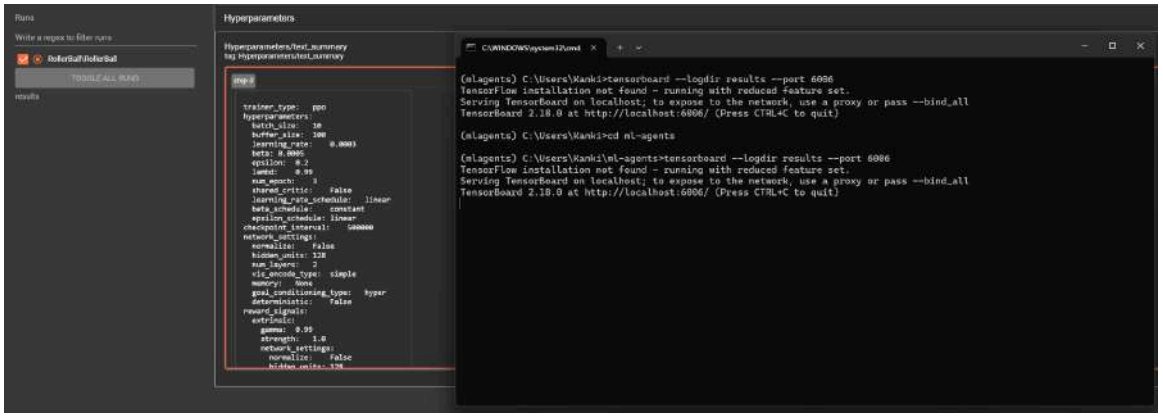
He hecho el setup de la actividad 2 siguiendo la guía de: [Making a New Learning Environment - Unity ML-Agents Toolkit]

Añadiendo el archivo `rollerball_config.yaml` y marcando la ruta correspondiente en el Anaconda Prompt (cd ml-agents), el servidor se ejecuta correctamente.

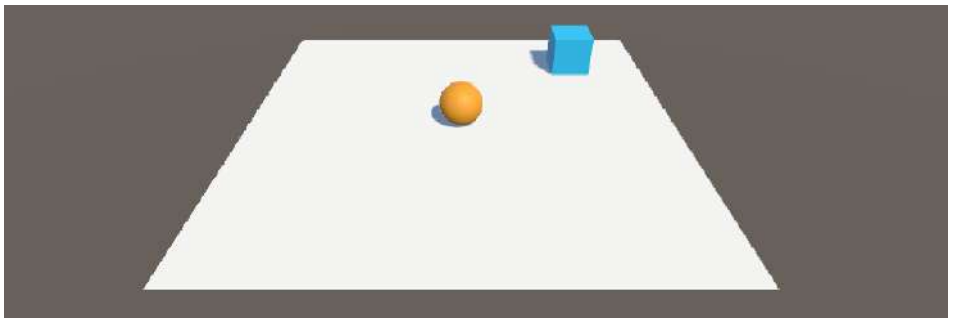
En mi caso estoy guardando los archivos (lo que sería el “cerebro”) en ml-agents/learn. Para reiniciar el entrenamiento utilizo:

```
mlagents-learn config/rollerball_config.yaml --run-id=RollerBall --force
```

Ahora quería ver las estadísticas de entrenamiento con TensoBoard utilizando `tensorboard --logdir results --port 6006`, pero por alguna razón lo que me mostraba el servidor era la siguiente captura:

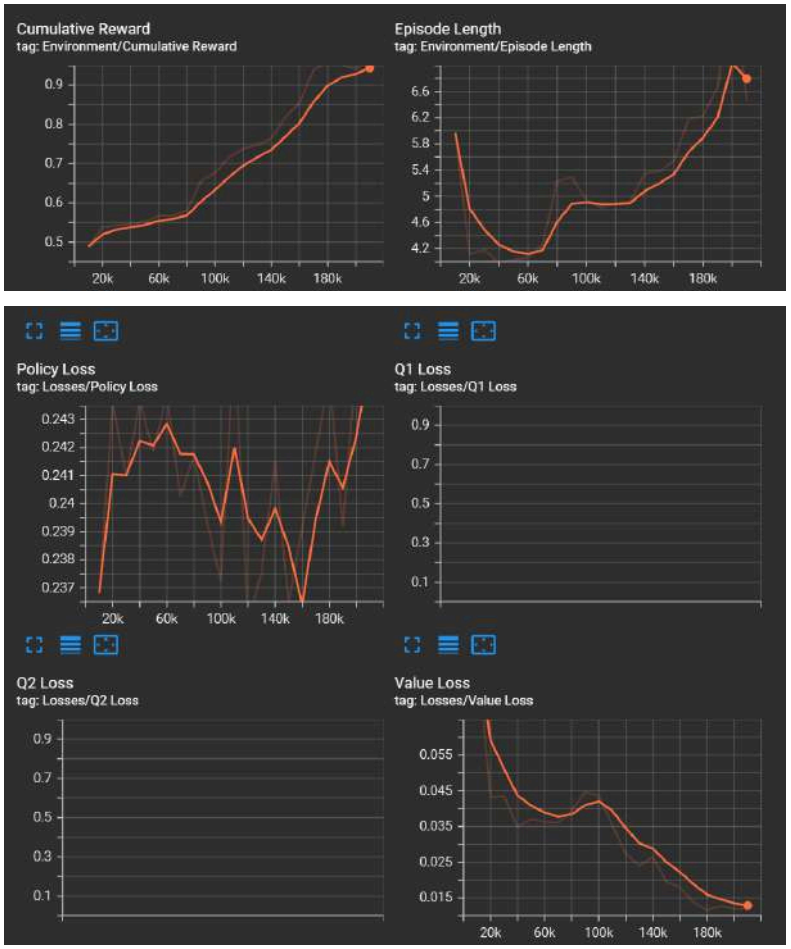


En este punto decidí reinstalar todo el entorno siguiendo los pasos de gebakx.github.io/gameAIUnity/ml/mlagentsGuide.txt, ya que por lo visto había incompatibilidades entre las versiones de los distintos elementos y no había manera de finalizar el entrenamiento.



En realidad no había ningún problema con este apartado, pero ha sido una buena idea reinstalar el entorno para asegurar la compatibilidad entre versiones.

Resultados



Para probar el modelo entrenado he localizado la carpeta de results/RollerBall y he arrastrado a los assets del proyecto el archivo .onnx con el mismo nombre. Este se debe asignar en el apartado “model” del script Behavior Parameters del editor.

En general el modelo actúa correctamente, pero hay veces donde se queda “atascado” en una esquina evitando caer pero sin llegar a tocar el cubo (reward).

Código *RollerAgent*

```
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;

public class RollerAgent : Agent
{
    Rigidbody rBody;
    void Start () {
        rBody = GetComponent<Rigidbody>();
    }

    public Transform Target;
    public override void OnEpisodeBegin()
    {
       // If the Agent fell, zero its momentum
        if (this.transform.localPosition.y < 0)
        {
            this.rBody.angularVelocity = Vector3.zero;
            this.rBody.velocity = Vector3.zero;
            this.transform.localPosition = new Vector3( 0, 0.5f, 0);
        }

        // Move the target to a new spot
        Target.localPosition = new Vector3(Random.value * 8 - 4,
                                           0.5f,
                                           Random.value * 8 - 4);
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        // Target and Agent positions
        sensor.AddObservation(Target.localPosition);
        sensor.AddObservation(this.transform.localPosition);

        // Agent velocity
        sensor.AddObservation(rBody.velocity.x);
        sensor.AddObservation(rBody.velocity.z);
    }
```

```
    public float forceMultiplier = 10;
    public override void OnActionReceived(ActionBuffers actionBuffers)
    {
        // Actions, size = 2
        Vector3 controlSignal = Vector3.zero;
        controlSignal.x = actionBuffers.ContinuousActions[0];
        controlSignal.z = actionBuffers.ContinuousActions[1];
        rBody.AddForce(controlSignal * forceMultiplier);

        // Rewards
        float distanceToTarget = Vector3.Distance(this.transform.localPosition,
Target.localPosition);

        // Reached target
        if (distanceToTarget < 1.42f)
        {
            SetReward(1.0f);
            EndEpisode();
        }

        // Fell off platform
        else if (this.transform.localPosition.y < 0)
        {
            EndEpisode();
        }
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var continuousActionsOut = actionsOut.ContinuousActions;
        continuousActionsOut[0] = Input.GetAxis("Horizontal");
        continuousActionsOut[1] = Input.GetAxis("Vertical");
    }

}
```

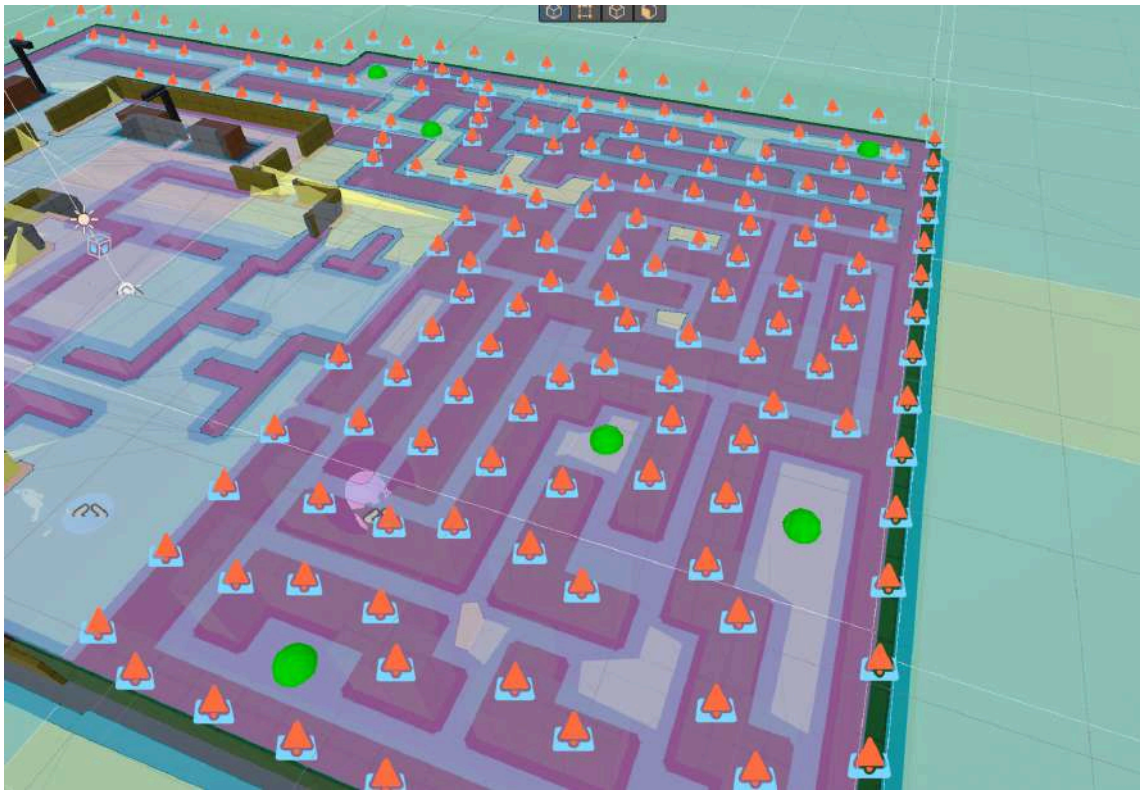
ACT 3: Mi propio ML-Agent

Esta actividad ha sido la responsable de un descenso progresivo hacia la locura. Por ese motivo se podrá ver a lo largo de este apartado que muchos intentos no acaban llegando a buen puerto. Del mismo modo, podréis apreciar que los archivos del proyecto no son los más intuitivos, pero espero que con este informe y el vídeo adjunto se pueda entender mejor.

Desarrollo

Primeramente, le he pedido a ChatGPT que me diera una lista de ideas de agentes ML que podría implementar en mi escenario actual. Finalmente, me he decidido por un personaje que recolectará flores por el mapa evitando a los demás agentes. Creo que es el que encaja mejor con la ambientación del juego y el sistema de recompensas es intuitivo.

He aprovechado la parte del mapa con el laberinto de setos para colocar spawners con flores:



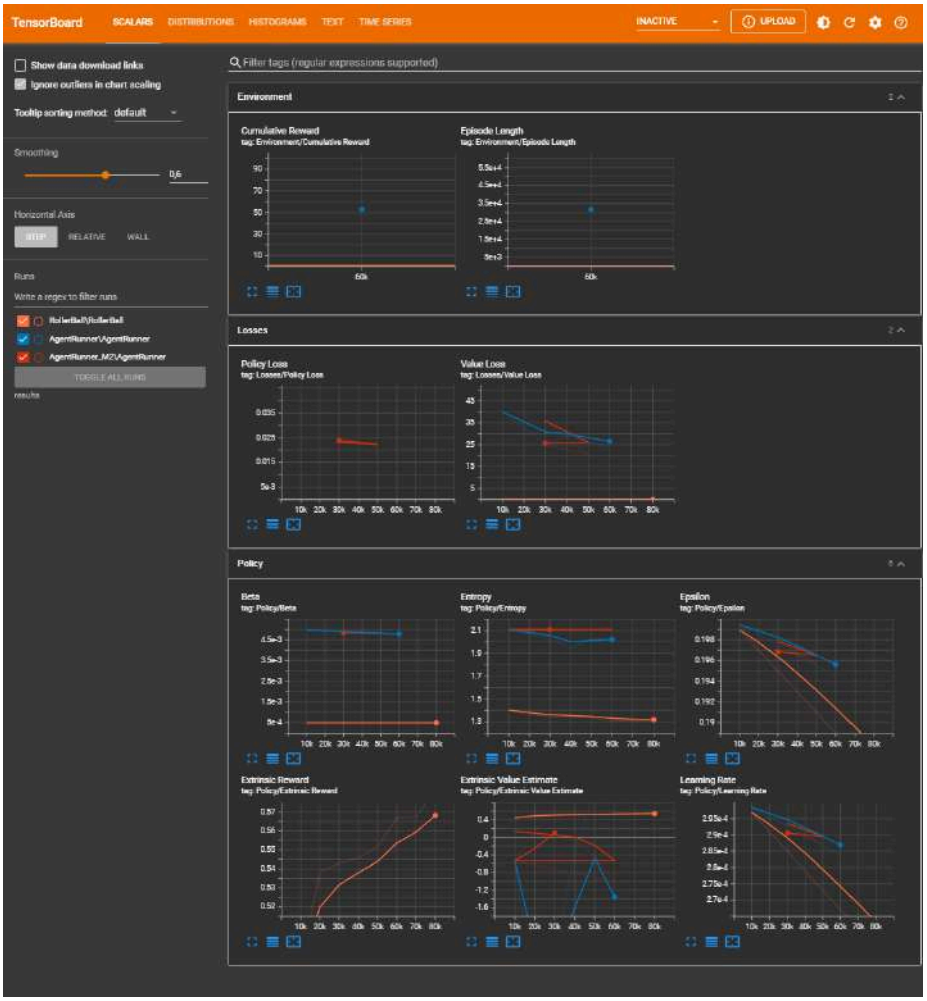
He utilizado un tag SpawnPoint para indicar los empties donde deben aparecer los prefabs de las flores y la layer Flowers para que el ML-agent sepa los objetos que debe ir a buscar. También he metido los “bushes” en una capa con el mismo nombre para que sepa las paredes que debe evitar.

He preparado el método *Heuristic* para poder mover al agente con las teclas WASD y así comprobar que todo funciona correctamente en el primer stage del aprendizaje.

Al crear al agente es importante añadir los constraints necesarios para que no se vuelque y la física actúe correctamente. Además, mi personaje estaba haciendo “clipping” porque tenía activado un maskObject para verlo a través de las paredes con un sphere collider.

Tras escribir la lógica para que recogiera las flores, he comenzado a entrenar el modelo. Le costaba mucho entrar en el laberinto y moverse por él. Le aumenté las variables de movimiento para que tuviera más libertad y añadí rewards más complejas para que el agente tuviera más clara la dirección a la que ir (las flores).

Para hacer el entrenamiento más rápido he copiado múltiples veces la zona de entrenamiento. En este punto no estaba obteniendo buenos resultados, muy posiblemente porque la tarea era demasiado compleja para empezar desde cero.

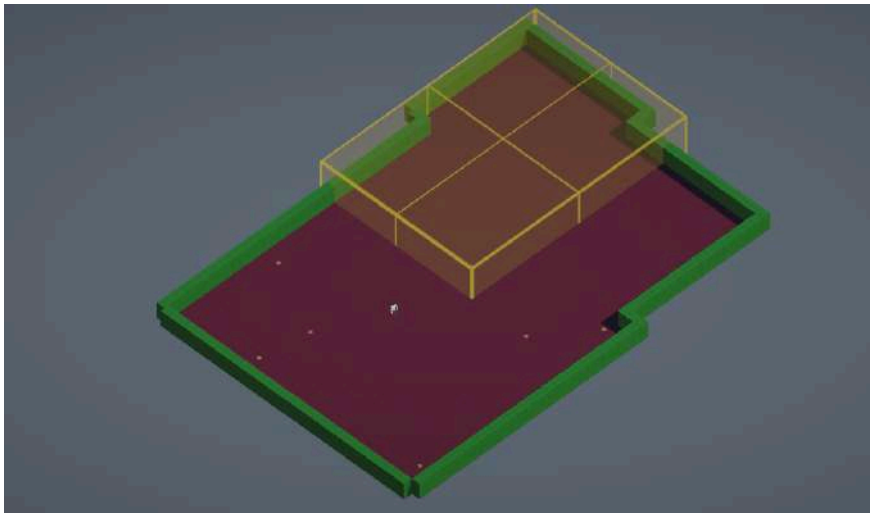


Por este motivo decidí seguir los pasos de curriculum learning, aumentando la dificultad del escenario progresivamente.

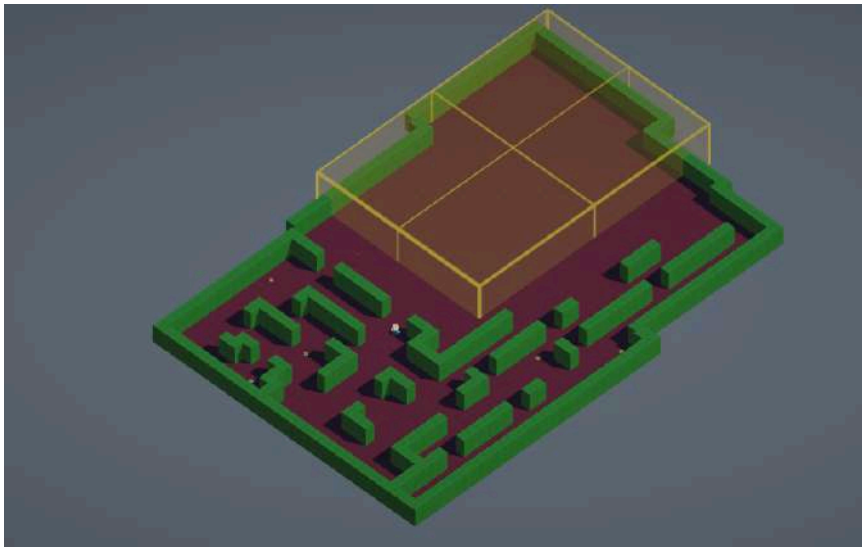
Curriculum Learning

Para hacer un entrenamiento progresivo he incorporado un CurriculumManager en todas las escenas por las que va a pasar el agente en el entrenamiento. Las escenas son las siguientes (en el proyecto se pueden encontrar en la carpeta de escenas):

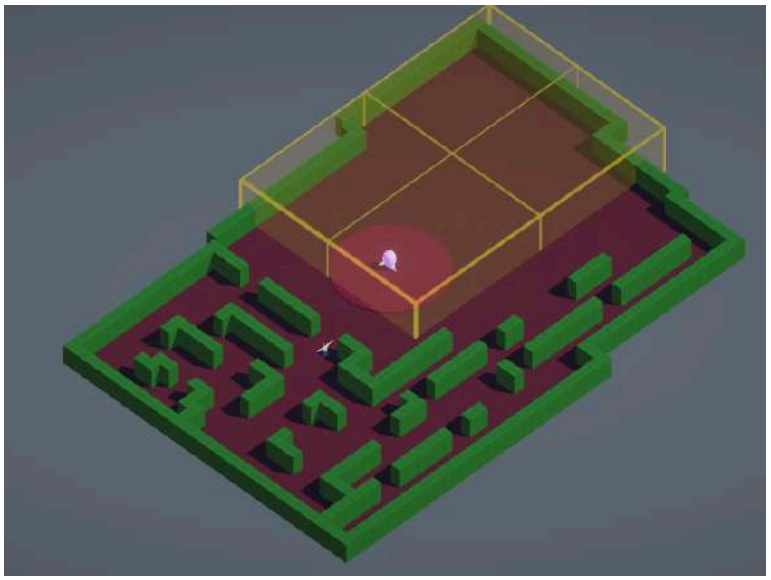
CL0 - Únicamente hay flores



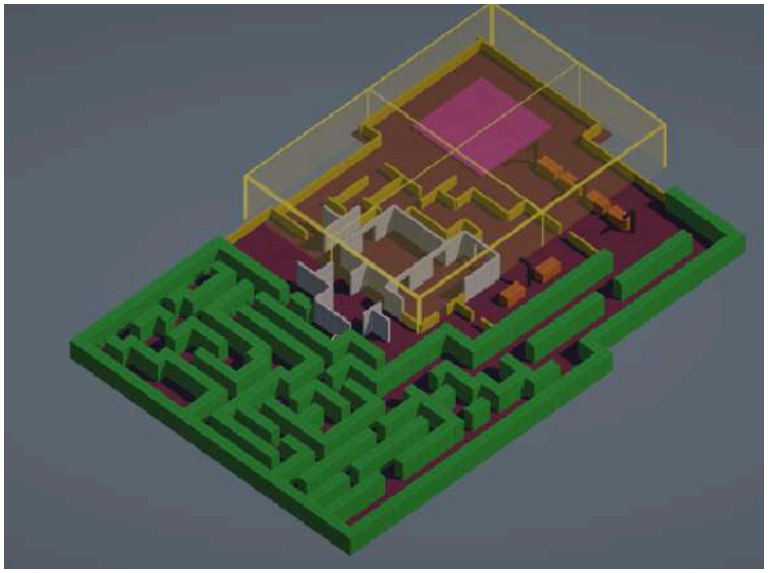
CL1 - Flores y laberinto simple



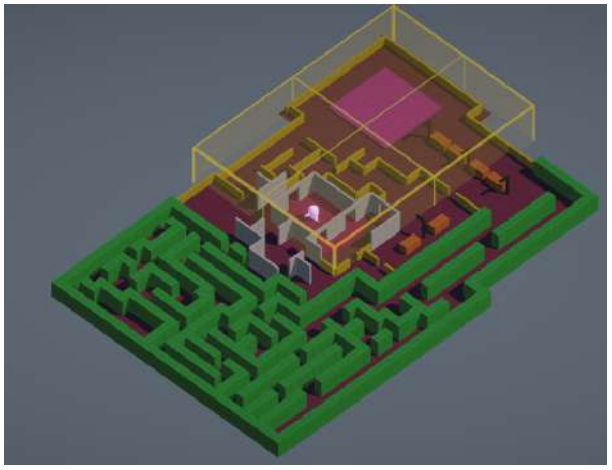
CL2 - Flores, laberinto simple y ghostAgent



CL3 - Flores y laberinto complejo



CL4 - Flores, laberinto y ghostAgent (es decir, todos los elementos que interactuarían con el ml-agent)



Código CurriculumManager

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.IO;

public class CurriculumManager : MonoBehaviour
{
    [Header("Scene Thresholds")]
    public SceneThreshold[] sceneThresholds; // Array of scenes and their thresholds

    private int currentPhase = 0; // Current phase index
    public string resultsPath = "results"; // Carpeta de resultados
    public string runId = "AgentRunner"; // ID del entrenamiento

    void Awake()
    {
        // Ensure only one instance of CurriculumManager exists
        if (FindObjectsByType<CurriculumManager>(FindObjectsSortMode.None).Length > 1)
        {
            Destroy(this.gameObject);
        }
        else
        {
            DontDestroyOnLoad(this.gameObject);
        }
    }

    public void CheckAndAdvance(int collectedFlowers)
    {
        if (currentPhase >= sceneThresholds.Length)
        {
            Debug.Log("All phases completed. Training is complete.");
            return;
        }

        SceneThreshold currentScene = sceneThresholds[currentPhase];

                            Debug.Log($"Collected    Flowers:    {collectedFlowers},    Threshold:
{sceneThresholds[currentPhase].flowerThreshold}");
        if (collectedFlowers >= currentScene.flowerThreshold)
        {
            AdvanceToNextPhase();
        }
    }

    public void AdvanceToNextPhase()
    {
        SaveModelSnapshot();

        if (currentPhase < sceneThresholds.Length - 1)
```

```
        {
            currentPhase++;
            LoadScene(currentPhase);
        }
        else
        {
            Debug.Log("Final phase reached. No more scenes to load.");
            // Optionally, implement end-of-training behavior here
        }
    }

    private void LoadScene(int phaseIndex)
    {
        if (phaseIndex >= 0 && phaseIndex < sceneThresholds.Length)
        {
            string sceneToLoad = sceneThresholds[phaseIndex].sceneName;
            Debug.Log($"Loading scene: {sceneToLoad}");
            SceneManager.LoadScene(sceneToLoad);
        }
        else
        {
            Debug.LogError($"Phase index {phaseIndex} is out of bounds!");
        }
    }

        private void SaveModelSnapshot()
    {
        string sourcePath = Path.Combine(resultsPath, runId, $"{runId}.onnx");
                            string    destinationPath    =    Path.Combine(resultsPath,    runId,
$"Phase_{currentPhase}_{runId}.onnx");

        if (File.Exists(sourcePath))
        {
            File.Copy(sourcePath, destinationPath, overwrite: true);
            Debug.Log($"Model snapshot saved: {destinationPath}");
        }
        else
        {
            Debug.LogWarning($"Model file not found: {sourcePath}");
        }
    }
}
```

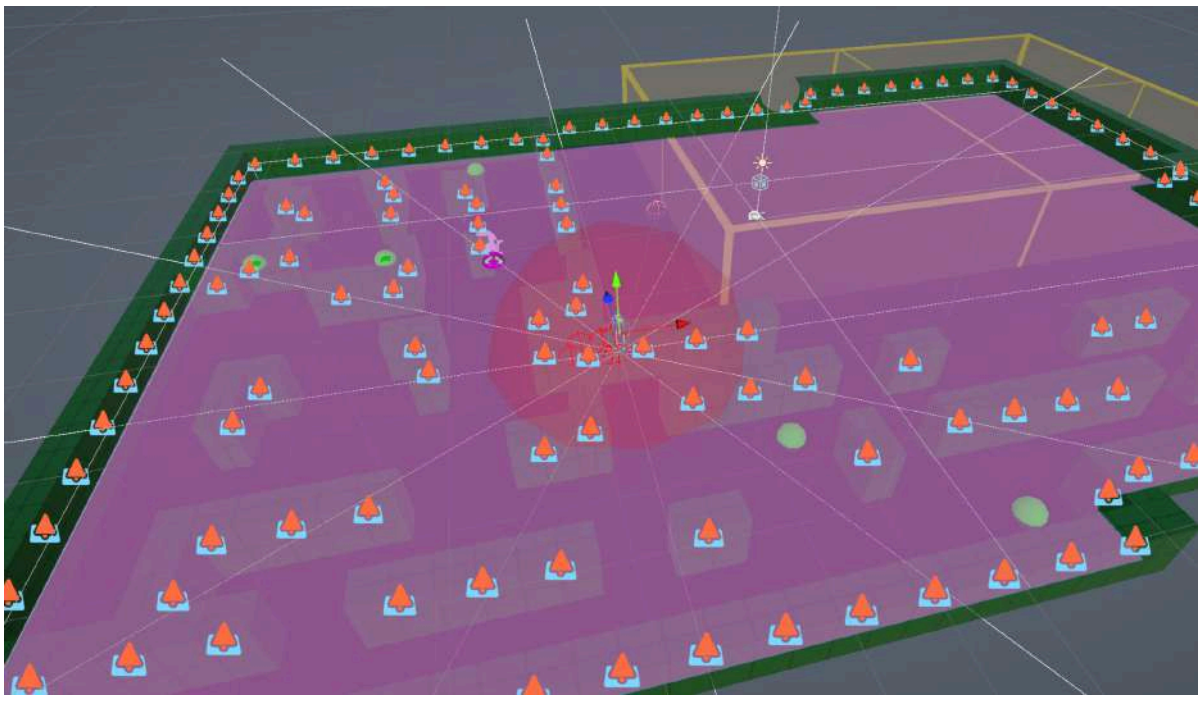
Intento de multi-thread

En este punto me doy cuenta de que el coste computacional de una tarea como esta es muy elevado, y más utilizando un único entorno renderizado. Por ese motivo intento (durante más tiempo del que debería) arrancar el entorno de entrenamiento con varios *environments headless*. Para ello debía hacer una build para servidor y exportarla, lo que me dio múltiples errores de compilación. Y una vez hecho, por algún motivo, el entorno de anaconda no captaba la IP de los servidores en local que había ejecutado con la builds exportadas.

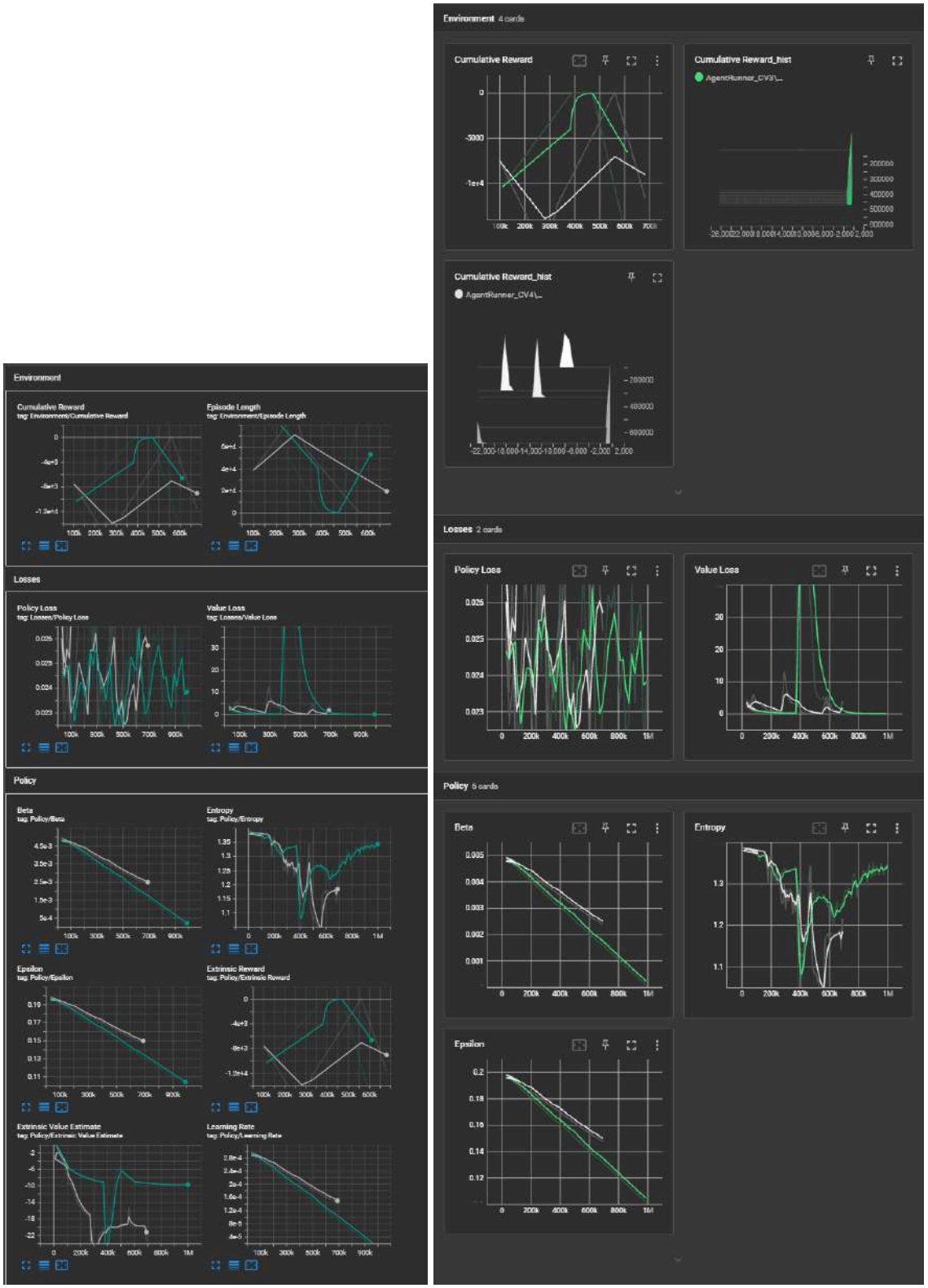
Se puede apreciar la aventura en el siguiente chat con ChatGPT:
https://chatgpt.com/share/6782acc3-e690-800b-a809-c4f4800b56c7

Tras rendirme con el tema de hacer entrenamiento en paralelo (porque la opción de copiar múltiples veces el entorno de entrenamiento como hace Sebastian Schuchmann en su vídeo sobre el tema era imposible con la complejidad de mi escenario y mi tarjeta gráfica) decido buscar otras maneras de mejorar la lógica del agente. Mencionar que pese a esto intento hacer todos los entrenamientos subiendo el time.scale a 10 o 15, pero si lo subía demasiado el editor de Unity dejaba de responder.

Añado un segundo raycast que atraviesa paredes en busca de las flores, lo cual implica una mejora substancial en el comportamiento, pero sigue siendo muy difícil que explore el laberinto. Además, reiniciar el agente al tocar el fantasma atascó el entrenamiento y se quedó varias horas avanzando prácticamente nada en el mean reward.



Aquí podemos apreciar la mejora utilizando el segundo raycast.



Comparación CV3 (verde) y CV4 (blanco)

Probando el algoritmo SAC

Aun así, en este punto no logro mejorar el comportamiento del agente tan siquiera en el entorno del laberinto básico, de modo que pido ayuda en el foro de clase y el profesor me recomienda dos cosas: utilizar SAC en lugar de PPO y simplificar todavía más el setup.

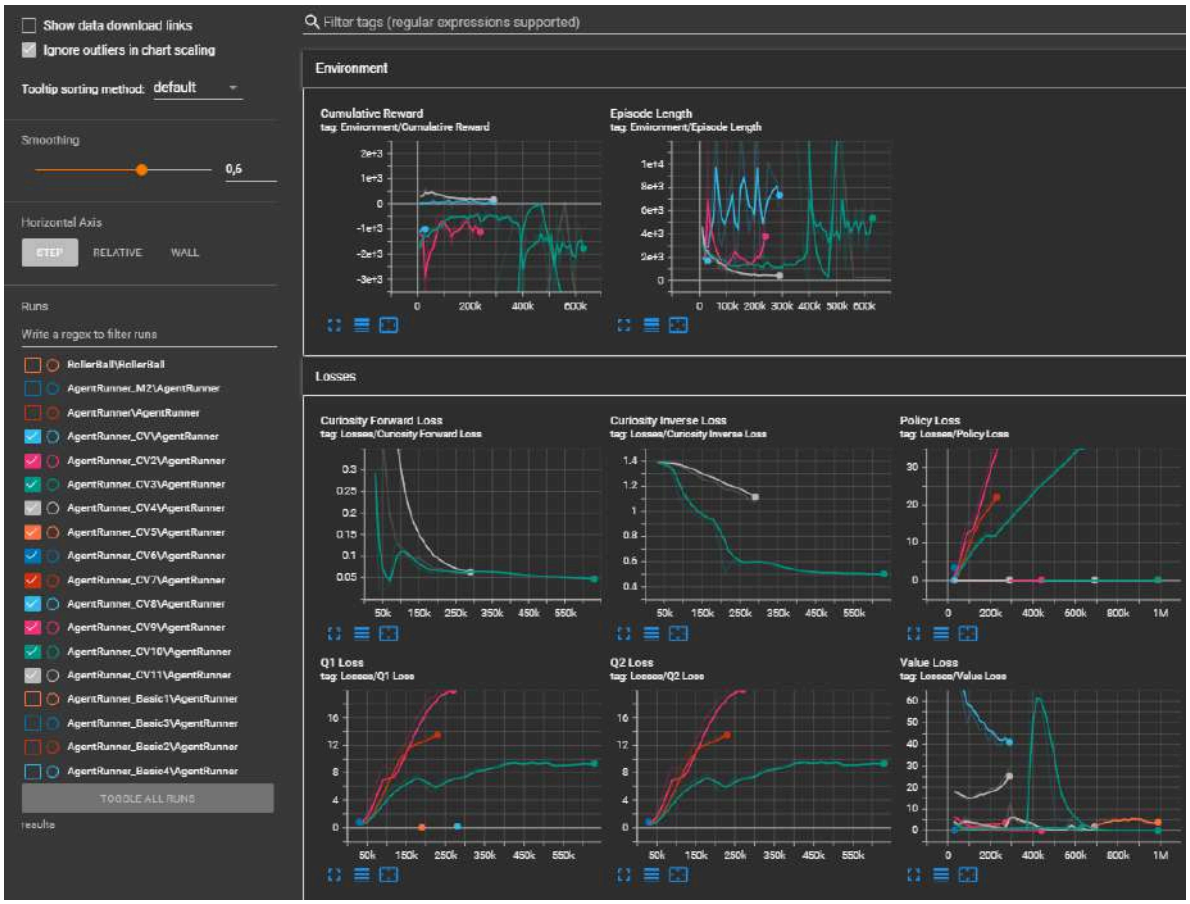
Cada vez veo más claro que había puesto las expectativas muy altas con los ml-agents (sobre todo teniendo en cuenta mis recursos). De este modo comienzo a probar el algoritmo Soft Actor-Critic (SAC) con mi agente y cambiando de acciones discretas a continuas

Aquí es donde sucede otro de mis errores durante el proceso de este proyecto: construir sobre lo ya construido en lugar de hacer versiones distintas. Precisamente por esto el código del agente (AgentRunner) se acaba convirtiendo en un engendro intratable.

Conversación con ChatGPT adaptando el código:
https://chatgpt.com/share/6782b031-f7bc-800b-92d8-423488b142c3

El código de las configuraciones de entrenamiento (archivos .yaml) se pueden encontrar en la carpeta de configuración del repositorio ml-agents.

Resultados con Curriculum learning



Código *AgentRunner* hasta el momento

```
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine.UI;
using System.Collections.Generic;
using System.Linq;

[RequireComponent(typeof(Rigidbody))]
public class AgentRunner : Agent
{
    [Header("Environment Settings")]
    public GameObject area;
    public LayerMask flowerLayer; // Layer to identify flowers

    [Header("Agent Settings")]
    public float turnSpeed = 300f;  // Rotational speed
    public float moveSpeed = 2f;    // Movement speed
    private Vector3 initialPosition;
    private Rigidbody agentRb;

    [Header("Goal Settings")]
    private Transform[] flowers;     // Active flowers in the map
    private Transform goal;          // Current goal of the agent
    public float maxDistance = 20f;
    private float previousDistanceToGoal;

    [Header("Flower Collection UI")]
    private int flowersCollected = 0;
    public Text flowersCollectedText; // UI Text to display flower count

    [Header("Ghost Settings")]
    public GameObject ghost;                    // Reference to the ghost GameObject
    public float ghostThresholdDistance = 5f;  // Distance threshold for negative reward

    [Header("Flower Spawner Settings")]
    public FlowerSpawner flowerSpawner; // Reference to the FlowerSpawner

    [Header("Curriculum Settings")]
    public float[] rewardThresholds; // Array of thresholds for each curriculum phase
    public CurriculumManager curriculumManager; // Reference to CurriculumManager
    private int currentPhase = 0; // Current phase index

    // Colors for Gizmos (optional for debugging)
    private Color goalColor = Color.green;
    private Color ghostColor = Color.magenta;
    private Color dangerZoneColor = new Color(1f, 0f, 0f, 0.3f); // Semi-transparent red

    private float cumulativeReward = 0f; // Cumulative reward in the episode
```

```
// References to TWO Ray Perception Sensors
private RayPerceptionSensorComponent3D wallsRaySensor;
private RayPerceptionSensorComponent3D flowersRaySensor;

public override void Initialize()
{
    initialPosition = transform.position;
    agentRb = GetComponent<Rigidbody>();
    agentRb.useGravity = true;
    agentRb.isKinematic = false;
    agentRb.collisionDetectionMode = CollisionDetectionMode.Continuous;

    // Initialize flower collection UI
    UpdateFlowersCollectedText();

    // Automatically find FlowerSpawner if not assigned
    if (flowerSpawner == null)
    {
        flowerSpawner = FindFirstObjectByType<FlowerSpawner>();
        if (flowerSpawner == null)
        {
            Debug.LogError("FlowerSpawner not found in the scene!");
        }
    }

    // Automatically find CurriculumManager if not assigned
    if (curriculumManager == null)
    {
        curriculumManager = FindFirstObjectByType<CurriculumManager>();
        if (curriculumManager == null)
        {
            Debug.LogError("CurriculumManager not found in the scene!");
        }
    }

    // ---------------------------------------------------------------------------
    // 1) CREATE AND CONFIGURE THE FIRST SENSOR FOR WALLS (and possibly Ghost)
    // ---------------------------------------------------------------------------
    wallsRaySensor = gameObject.AddComponent<RayPerceptionSensorComponent3D>();
    wallsRaySensor.SensorName = "WallsRaySensor"; // Unique name
    wallsRaySensor.RayLength = 10f;
    wallsRaySensor.RaysPerDirection = 3; // Increase for better coverage
    wallsRaySensor.MaxRayDegrees = 70f;
    wallsRaySensor.DetectableTags = new List<string>() { "Wall", "Ghost" };
    wallsRaySensor.RayLayerMask = LayerMask.GetMask("Walls", "Ghost", "Default");
    // Ensure "Walls", "Ghost", and "Default" layers exist and are correctly assigned

    // ---------------------------------------------------------------------------
    // 2) CREATE AND CONFIGURE THE SECOND SENSOR FOR FLOWERS (ignores walls)
    // ---------------------------------------------------------------------------
    flowersRaySensor = gameObject.AddComponent<RayPerceptionSensorComponent3D>();
```

```
        flowersRaySensor.SensorName = "FlowersRaySensor"; // Unique name
           flowersRaySensor.RayLength = 30f; // Longer rays to detect flowers from a greater
distance
        flowersRaySensor.RaysPerDirection = 6; // More rays for better detection
        flowersRaySensor.MaxRayDegrees = 180f; // 360-degree perception
        flowersRaySensor.DetectableTags = new List<string>() { "Flower" };
        flowersRaySensor.RayLayerMask = LayerMask.GetMask("Flowers");
        // Ensure "Flowers" layer exists and is correctly assigned

        goal = GetNearestFlower();
    }

    private void Update() {
        goal = GetNearestFlower();
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        goal = GetNearestFlower();

        // 1) Relative position to the goal (nearest flower)
        if (goal != null)
        {
            Vector3 directionToGoal = (goal.position - transform.position).normalized;
                  float distanceToGoal = Vector3.Distance(transform.position, goal.position) /
maxDistance; // Normalized
            sensor.AddObservation(directionToGoal.x);
            sensor.AddObservation(directionToGoal.z);
            sensor.AddObservation(distanceToGoal);
        }
        else
        {
            Debug.LogWarning("No goal found for the agent!");
            sensor.AddObservation(0f); // Direction X
            sensor.AddObservation(0f); // Direction Z
            sensor.AddObservation(1f); // Max distance
        }

        // 2) Relative position to the ghost
        if (ghost != null)
        {
                                    Vector3 directionToGhost = (ghost.transform.position -
transform.position).normalized;
                                float distanceToGhost = Vector3.Distance(transform.position,
ghost.transform.position) / ghostThresholdDistance; // Normalized
            sensor.AddObservation(directionToGhost.x);
            sensor.AddObservation(directionToGhost.z);
            sensor.AddObservation(distanceToGhost);
        }
        else
        {
            sensor.AddObservation(0f); // Direction X
```

```
            sensor.AddObservation(0f); // Direction Z
            sensor.AddObservation(1f); // Max distance
        }

        // RayPerceptionSensors automatically add their own observations
    }

    public override void OnActionReceived(ActionBuffers actionBuffers)
    {
        MoveAgent(actionBuffers);
    }

    /// <summary>
    /// Interprets discrete actions and applies them to the Agent's movement/rotation.
    /// </summary>
    public void MoveAgent(ActionBuffers actionBuffers)
    {
        // Discrete actions: [0] forward/back, [1] turn
        var discreteActions = actionBuffers.DiscreteActions;
        int forwardAction = discreteActions[0]; // {0=No movement, 1=Forward, 2=Backward}
        int turnAction = discreteActions[1];    // {0=No turn, 1=Right, 2=Left}

        float moveInput = 0f;
        float turnInput = 0f;

        // Forward/Backward
        if (forwardAction == 1) moveInput = 1f;  // Move forward
        else if (forwardAction == 2) moveInput = -1f; // Move backward

        // Left/Right
        if (turnAction == 1) turnInput = 1f;   // Turn right
        else if (turnAction == 2) turnInput = -1f; // Turn left

        // Apply movement
        Vector3 move = transform.forward * moveInput * moveSpeed;
        agentRb.AddForce(move, ForceMode.VelocityChange);

        // Apply rotation
        float turn = turnInput * turnSpeed * Time.fixedDeltaTime;
        transform.Rotate(0, turn, 0);

        // Limit velocity to prevent excessive speed
        if (agentRb.velocity.sqrMagnitude > 25f)
        {
            agentRb.velocity = agentRb.velocity.normalized * 5f;
        }

        // Reward Structure
        if (goal != null)
        {
            float currentDistanceToGoal = Vector3.Distance(transform.position, goal.position);
            float distanceChange = previousDistanceToGoal - currentDistanceToGoal;
```

```
            if (distanceChange > 0)
            {
                AddReward(0.1f); // Increased reward for getting closer
            }
            else
            {
                AddReward(-0.05f); // Reduced penalty for moving away
            }
            previousDistanceToGoal = currentDistanceToGoal;
        }
        else
        {
            AddReward(-0.01f); // Small penalty for each step without a goal
        }

        // Negative Reward Logic: Proximity to Ghost
        if (ghost != null)
        {
                                float distanceToGhost = Vector3.Distance(transform.position,
ghost.transform.position);
            if (distanceToGhost < ghostThresholdDistance)
            {
                AddReward(-0.05f); // Increased penalty for being too close
                //Debug.Log("Agent is near the ghost!");
            }
        }
    }

    private void FixedUpdate()
    {
        // Check if the current phase has a valid threshold
                    if (currentPhase < rewardThresholds.Length && cumulativeReward >=
rewardThresholds[currentPhase])
        {
              Debug.Log($"Threshold {rewardThresholds[currentPhase]} reached! Advancing to next
phase.");
            curriculumManager.AdvanceToNextPhase(); // Notify CurriculumManager
            currentPhase++;
            EndEpisode(); // End current episode to load new scene
        }
    }

    /// <summary>
    /// Provide discrete actions for testing with keyboard (Heuristic).
    /// This will only be used when Behavior Type is set to "Heuristic Only."
    /// </summary>
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        // Discrete action branch sizes must match the Behavior Parameters in the Inspector:
        // Branch 0: 3 (No movement, Forward, Backward)
        // Branch 1: 3 (No turn, Right, Left)
```

```
        var discreteActionsOut = actionsOut.DiscreteActions;

        // Default is "do nothing":
        discreteActionsOut[0] = 0; // forwardAction
        discreteActionsOut[1] = 0; // turnAction

        // Forward/back
        if (Input.GetKey(KeyCode.W))
            discreteActionsOut[0] = 1; // forward
        else if (Input.GetKey(KeyCode.S))
            discreteActionsOut[0] = 2; // backward

        // Left/right
        if (Input.GetKey(KeyCode.D))
            discreteActionsOut[1] = 1; // right
        else if (Input.GetKey(KeyCode.A))
            discreteActionsOut[1] = 2; // left
    }

    public override void OnEpisodeBegin()
    {
        cumulativeReward = 0f;
        flowersCollected = 0;
        UpdateFlowersCollectedText();

        agentRb.velocity = Vector3.zero;
        transform.position = initialPosition;
        transform.rotation = Quaternion.Euler(new Vector3(0f, Random.Range(0, 360), 0f));

        ReactivateFlowers();
        FindFlowers();
        goal = GetNearestFlower();

        if (goal != null)
        {
            previousDistanceToGoal = Vector3.Distance(transform.position, goal.position);
        }
        else
        {
            Debug.LogWarning("No active flowers found at the start of the episode.");
        }
    }

    void OnCollisionEnter(Collision collision)
    {
        // Check if we collided with a flower layer object
        if (((1 << collision.gameObject.layer) & flowerLayer.value) != 0)
        {
            AddReward(5.0f); // Reward for collecting a flower

            flowersCollected++;
```

```
            UpdateFlowersCollectedText();

            if (flowerSpawner != null)
            {
                flowerSpawner.FlowerCollected(collision.gameObject);
            }
            else
            {
                Debug.LogWarning("FlowerSpawner reference is not assigned in AgentRunner!");
            }

            if (goal != null)
            {
                previousDistanceToGoal = Vector3.Distance(transform.position, goal.position);
            }
            else
            {
                Debug.LogWarning("No active flowers found after collecting one.");
                EndEpisode(); // Optionally end the episode when all flowers are collected
            }

            if (curriculumManager != null)
            {
                curriculumManager.CheckAndAdvance(flowersCollected);
            }
            else
            {
                            Debug.LogWarning("CurriculumManager reference is not assigned in
AgentRunner!");
            }
        }

        goal = GetNearestFlower();
    }

    private void UpdateFlowersCollectedText()
    {
        if (flowersCollectedText != null)
        {
            flowersCollectedText.text = "Flowers Collected: " + flowersCollected;
        }
    }

    private void FindFlowers()
    {
                Collider[] flowerColliders = Physics.OverlapSphere(transform.position, 100f,
flowerLayer);
        flowers = new Transform[flowerColliders.Length];
        for (int i = 0; i < flowerColliders.Length; i++)
        {
            flowers[i] = flowerColliders[i].transform;
        }
```

```
    }
    private Transform GetNearestFlower()
    {
        FindFlowers();
        if (flowers == null || flowers.Length == 0) return null;

        Transform nearest = null;
        float minDistance = float.MaxValue;

        foreach (var point in flowers)
        {
            if (point == null || !point.gameObject.activeSelf) continue;

            float dist = Vector3.Distance(transform.position, point.position);
            if (dist < minDistance)
            {
                nearest = point;
                minDistance = dist;
            }
        }
        return nearest;
    }

    private void ReactivateFlowers()
    {
        foreach (Transform flower in area.transform)
        {
            flower.gameObject.SetActive(true);
        }
    }

    private void OnDrawGizmos()
    {
        // Only draw Gizmos when the game is playing
        if (!Application.isPlaying) return;

        if (goal != null)
        {
            Gizmos.color = goalColor;
            Gizmos.DrawSphere(goal.position, 0.5f);
        }

        if (ghost != null)
        {
            Gizmos.color = dangerZoneColor;
            Gizmos.DrawSphere(transform.position, ghostThresholdDistance);

            Gizmos.color = ghostColor;
            Gizmos.DrawSphere(ghost.transform.position, 0.5f);
        }
    }
}
```

Archivos .yaml

```yaml
behaviors:
  AgentRunner:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 1300000
    time_horizon: 64
    summary_freq: 10000
    threaded: false
```

```yaml
behaviors:
  AgentRunner:
    trainer_type: sac
    hyperparameters:
      learning_rate: 0.0005
      learning_rate_schedule: constant
      batch_size: 512
      buffer_size: 2000000
      buffer_init_steps: 20000
      tau: 0.005
      steps_per_update: 20
      save_replay_buffer: false
      init_entcoef: 0.1
      reward_signal_steps_per_update: 20
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
      curiosity:
        strength: 0.05
        gamma: 0.99
        encoding_size: 256
        learning_rate: 3.0e-4
    keep_checkpoints: 5
    max_steps: 1200000
    time_horizon: 64
    summary_freq: 10000
    threaded: false
```

Trying to “keep it simple”

La recta final con este proyecto ha sido un intento de conseguir algo medianamente decente aunque fuera con el escenario y código más simples posibles.

Uno de los problemas de que el agente fuera tan perdido era que el goal no se definía antes de buscarlo. Es un error absurdo, pero debido a que no conocía el funcionamiento de los ml-agents no he podido identificarlo hasta pasadas horas de entrenamiento. Con un *debugging* más escrupuloso lo hubiera percibido antes, pero no ha sido hasta que me he puesto a revisarlo punto por punto que lo he descubierto. Pero bueno, de todo se aprende; poniendo en un *update* que defina el *goal* recurrentemente se ha arreglado. También he añadido gizmos para visualizar claramente el destino del agente y cerciorarme de que conoce la posición del agente Ghost.

Otro de mis errores (que documentándome mejor seguro que no hubiera sucedido) era que no utilizaba el modo Heuristic correctamente; para entrar en modo heurístico y hacer inferencia al modelo se tiene que hacer con el entorno de entrenamiento activado y conectado a Unity y posteriormente continuar el entrenamiento (--resume) en default.

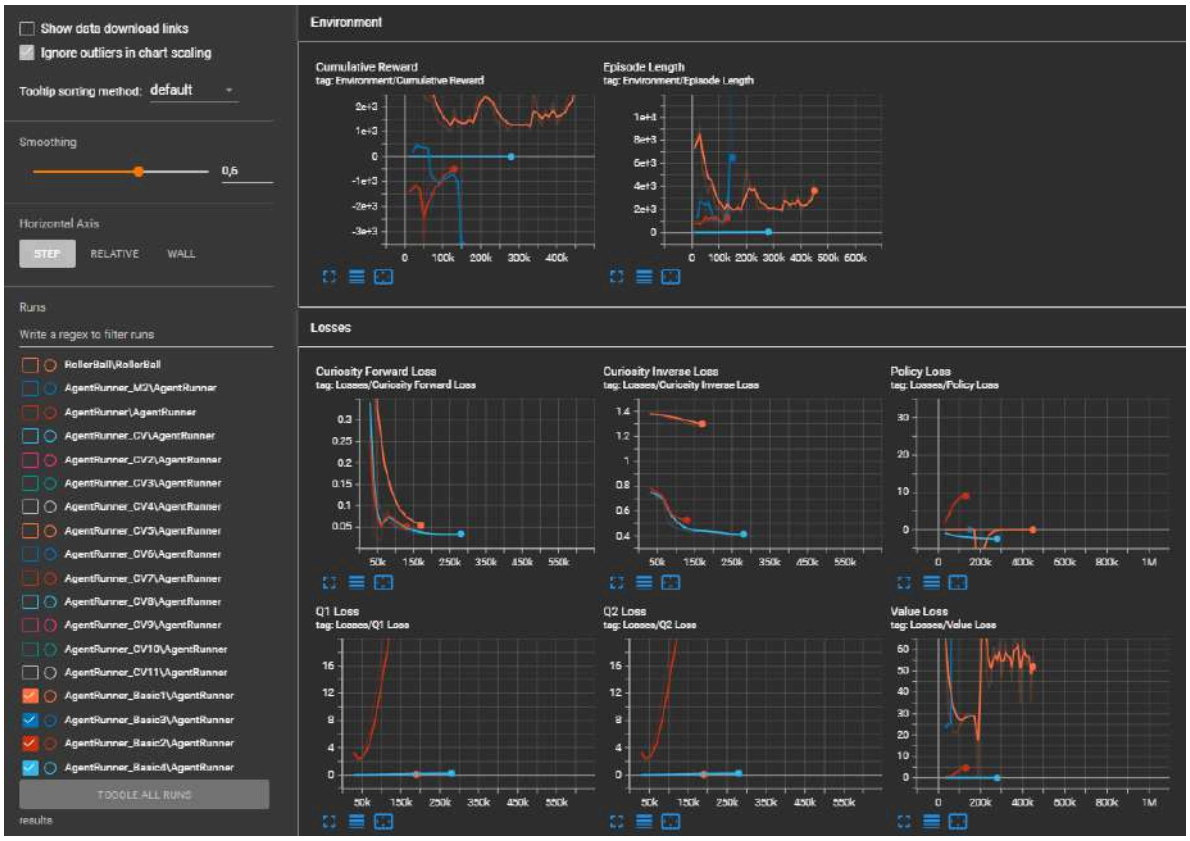
Para este entrenamiento también dejo de lado todo el sistema que había hecho de entrenamiento progresivo con currículum learning y pongo el agente en un escenario adaptado (hasta ahora había estado utilizando Main V2-ML y los CL0-4, pero este es Main V3-ML).

Otra de las decisiones que se me ocurre tomar es la de hacer que un episodio de entrenamiento finalice también al tocar una pared y al acercarse al fantasma (hasta ahora solamente terminaba cuando recogía las 6 flores o tocaba al agente). Esta quizás es una buena vía de desarrollo, pero aumenta enormemente el tiempo de entrenamiento.

Sé que en el enunciado de la actividad especifica que debemos jugar muchas iteraciones (hasta un millón), pero debido a mis restricciones de hardware (principalmente por no poder dejarlo encendido múltiples horas) y que no he podido paralelizar el entrenamiento lo más lejos que he llegado ha sido al step 630k.

Paradójicamente, los entrenamientos con el código simplificado iban más lentos, pero dificilmente llegaban a puntos de “atascarse”, que era la principal causa de detención de los entrenamientos del currículum learning.

Resultados



Código *AgentRunner* simplificado

```
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;

[RequireComponent(typeof(Rigidbody))]
public class AgentRunnerBasic : Agent
{
    [Header("Environment Settings")]
    public Transform ghost;                    // Ghost position
    public LayerMask detectableLayers;         // Layers detectable by wall ray sensors
    public LayerMask detectableLayersFlowers;  // Layers detectable by flower ray sensors

    [Header("Movement Settings")]
    public float moveForceMultiplier = 5f;     // Multiplier for movement force
    public float turnSpeed = 150f;              // Adjusted from 200f
```

```
    [Header("Reward Settings")]
    public float rewardForFlower = 1.0f;
    public float penaltyForGhostProximity = -1.0f;
    public float proximityRewardMultiplier = 0.1f;   // Multiplier for proximity-based reward
    public float penaltyForWallCollision = -0.5f;   // New penalty for wall collision

    public float distanceThreshold = 1.0f;          // Ghost
     public int flowersToCollect = 6;                    // Number of flowers to collect before
episode reset

    [Header("Ray Perception Settings")]
    public float rayLength = 10f;
    public float[] rayAngles = { -45f, 0f, 45f };    // Angles for ray directions
    public float rayDegrees = 30f;                   // Field of view for each ray

    [Header("Debug Settings")]
     public bool enableDebugLogs = true;                // Toggle for enabling/disabling debug
logs

    private Rigidbody rBody;
    private Transform nearestFlower;
    private int flowersCollected = 0;
    private Vector3 initialPosition;

    // Ray Perception Sensors
    private RayPerceptionSensorComponent3D wallRaySensor;
    private RayPerceptionSensorComponent3D flowerRaySensor;

    // Reference to FlowerSpawner
    public FlowerSpawner flowerSpawner;

    public override void Initialize()
    {
        base.Initialize();
        initialPosition = transform.position; // Store the initial position

        rBody = GetComponent<Rigidbody>();

        // **Find FlowerSpawner in the scene**
        flowerSpawner = FindFirstObjectByType<FlowerSpawner>();
        if (flowerSpawner == null)
        {
            Debug.LogError("FlowerSpawner not found in the scene!");
        }

        // Initialize Ray Perception Sensors
        InitializeRaySensors();

        // **Do not call FindNearestFlower() here to avoid "No active flowers found."**
        // It will be called in OnEpisodeBegin() after flowers are spawned
    }
```

```
    private void InitializeRaySensors()
    {
        // Initialize Wall Ray Sensor
        wallRaySensor = gameObject.AddComponent<RayPerceptionSensorComponent3D>();
        wallRaySensor.SensorName = "WallRaySensor";
        wallRaySensor.RayLength = rayLength;
        wallRaySensor.RaysPerDirection = rayAngles.Length;
        wallRaySensor.DetectableTags = new List<string>() { "Wall", "Ghost" };
        wallRaySensor.RayLayerMask = detectableLayers;

        // Initialize Flower Ray Sensor
        flowerRaySensor = gameObject.AddComponent<RayPerceptionSensorComponent3D>();
        flowerRaySensor.SensorName = "FlowerRaySensor";
        flowerRaySensor.RayLength = rayLength;
        flowerRaySensor.RaysPerDirection = rayAngles.Length;
        flowerRaySensor.DetectableTags = new List<string>() { "Flower" };
        flowerRaySensor.RayLayerMask = detectableLayersFlowers;
    }

    public override void OnEpisodeBegin()
    {
        // Reset Agent's position and velocity
        rBody.velocity = Vector3.zero;
        rBody.angularVelocity = Vector3.zero;

        // **Attempt to find a valid random position**
        Vector3 randomPosition;
        int maxAttempts = 10;
        int attempts = 0;
        do
        {
            float randomOffsetX = Random.Range(-5f, 5f);
            float randomOffsetZ = Random.Range(-5f, 5f);
            randomPosition = initialPosition + new Vector3(randomOffsetX, 0, randomOffsetZ);
            attempts++;
                    } while (Physics.CheckBox(randomPosition, transform.localScale / 2,
Quaternion.identity, detectableLayers) && attempts < maxAttempts);

        transform.position = randomPosition;
        transform.rotation = Quaternion.Euler(0, Random.Range(0, 360), 0);

        // **Reset flowers via FlowerSpawner**
        flowerSpawner.SpawnAllFlowers();

        // **Reset counters**
        flowersCollected = 0;
        FindNearestFlower();

        if (enableDebugLogs)
        {
            Debug.Log("Episode Began: Agent reset and flowers repositioned.");
        }
```

```
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        // **Relative position to the nearest flower**
        if (nearestFlower != null)
        {
                                    Vector3   directionToFlower   =   (nearestFlower.position   -
transform.position).normalized;
                               float   distanceToFlower   =   Vector3.Distance(transform.position,
nearestFlower.position);
            sensor.AddObservation(directionToFlower.x);
            sensor.AddObservation(directionToFlower.z);
            sensor.AddObservation(distanceToFlower / rayLength); // Normalize
        }
        else
        {
            Debug.LogWarning("No active flowers found.");

            sensor.AddObservation(0f);
            sensor.AddObservation(0f);
            sensor.AddObservation(1f);
        }

        // **Relative position to the ghost**
        if (ghost != null)
        {
            Vector3 directionToGhost = (ghost.position - transform.position).normalized;
            float distanceToGhost = Vector3.Distance(transform.position, ghost.position);
            sensor.AddObservation(directionToGhost.x);
            sensor.AddObservation(directionToGhost.z);
            sensor.AddObservation(distanceToGhost / rayLength); // Normalize
        }
        else
        {
            sensor.AddObservation(0f);
            sensor.AddObservation(0f);
            sensor.AddObservation(1f);
        }

        // **Ray Perception Sensors' Observations:**
        // These are automatically added by the RayPerceptionSensorComponents
        // No need to manually collect them here
    }

    public override void OnActionReceived(ActionBuffers actionBuffers)
    {
        // Continuous Actions: [0] Move Forward/Backward, [1] Turn Left/Right
        float moveInput = actionBuffers.ContinuousActions[0]; // Typically in range [-1, 1]
        float turnInput = actionBuffers.ContinuousActions[1]; // Typically in range [-1, 1]

        // Apply movement
```

```
        Vector3 force = transform.forward * moveInput * moveForceMultiplier;
        Quaternion turn = Quaternion.Euler(0, turnInput * turnSpeed * Time.fixedDeltaTime, 0);
        rBody.AddForce(force, ForceMode.Force);
        rBody.MoveRotation(transform.rotation * turn);

        // **Reward for moving closer to the flower**
        if (nearestFlower != null)
        {
                            float distanceToFlower = Vector3.Distance(transform.position,
nearestFlower.position);
                float normalizedDistance = Mathf.Clamp01(distanceToFlower / rayLength); //
Normalize between 0 and 1
            float proximityReward = proximityRewardMultiplier * (1 - normalizedDistance); //
Higher reward as closer
            proximityReward = Mathf.Min(proximityReward, 0.1f); // Cap the reward to prevent
excessive rewards
            AddReward(proximityReward);
            if (enableDebugLogs)
            {
                // Optionally enable for debugging
                        // Debug.Log($"Proximity Reward Added: {proximityReward:F4} for being
{distanceToFlower:F2} units from the nearest flower.");
            }
        }

        // **Movement Penalty to Encourage Efficiency**
        AddReward(-0.001f * Time.fixedDeltaTime); // Small time-based penalty

        // **Penalty for being close to the ghost**
        if (ghost != null)
        {
            float distanceToGhost = Vector3.Distance(transform.position, ghost.position);
            if (distanceToGhost < distanceThreshold)
            {
                AddReward(penaltyForGhostProximity);
                if (enableDebugLogs)
                {
                    Debug.Log($"Penalty Applied: {penaltyForGhostProximity} for being too close
to the ghost ({distanceToGhost:F2} units).");
                }
                EndEpisode(); // Terminate the episode upon penalty
            }
        }

        // **Check if agent has reached the flower**
        if (nearestFlower != null)
        {
                            float distanceToFlower = Vector3.Distance(transform.position,
nearestFlower.position);
            if (distanceToFlower < distanceThreshold)
            {
                AddReward(rewardForFlower);
```

```
                if (enableDebugLogs)
                {
                    Debug.Log($"Flower Collected: {rewardForFlower} reward added.");
                }

                // **Deactivate the flower**
                nearestFlower.gameObject.SetActive(false);
                flowersCollected++;
                if (enableDebugLogs)
                {
                                                    Debug.Log($"Total   Flowers   Collected:
{flowersCollected}/{flowersToCollect}");
                }

                // **Notify FlowerSpawner of collection**
                flowerSpawner.FlowerCollected(nearestFlower.gameObject);

                FindNearestFlower();

                // **Check if required number of flowers are collected**
                if (flowersCollected >= flowersToCollect)
                {
                    if (enableDebugLogs)
                    {
                        Debug.Log($"Collected {flowersToCollect} flowers. Ending episode.");
                    }
                    EndEpisode();
                }
            }
        }
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var continuousActionsOut = actionsOut.ContinuousActions;

        // Heuristic: Manual control for testing
        continuousActionsOut[0] = Input.GetAxis("Vertical");   // Forward/Backward
        continuousActionsOut[1] = Input.GetAxis("Horizontal"); // Left/Right
    }

    private void FindNearestFlower()
    {
        if (flowerSpawner == null)
        {
            Debug.LogError("FlowerSpawner reference not set!");
            return;
        }

        float minDistance = Mathf.Infinity;
        nearestFlower = null;
```

```
        foreach (GameObject flower in flowerSpawner.GetActiveFlowers())
        {
            if (flower == null || !flower.activeSelf) continue;

            float distance = Vector3.Distance(transform.position, flower.transform.position);
            if (distance < minDistance)
            {
                minDistance = distance;
                nearestFlower = flower.transform;
            }
        }

        if (nearestFlower != null)
        {
            if (enableDebugLogs)
            {
                Debug.Log($"Nearest Flower Found at distance: {minDistance:F2} units.");
            }
        }
        else
        {
            if (enableDebugLogs)
            {
                Debug.LogWarning("No active flowers found.");
            }
        }
    }

    private void ResetFlowers()
    {
        flowerSpawner.SpawnAllFlowers();
        if (enableDebugLogs)
        {
            Debug.Log("All flowers have been reset and repositioned.");
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        // If agent collides with a flower
        if (collision.gameObject.CompareTag("Flower"))
        {
            AddReward(rewardForFlower);
            if (enableDebugLogs)
            {
                Debug.Log($"Collision with Flower: {rewardForFlower} reward added.");
            }

            collision.gameObject.SetActive(false);
            flowersCollected++;
            if (enableDebugLogs)
            {
```

```
                Debug.Log($"Total Flowers Collected: {flowersCollected}/{flowersToCollect}");
            }

            // **Notify FlowerSpawner of collection**
            flowerSpawner.FlowerCollected(collision.gameObject);

            FindNearestFlower();

            // **Check if required number of flowers are collected**
            if (flowersCollected >= flowersToCollect)
            {
                if (enableDebugLogs)
                {
                        Debug.Log($"Collected {flowersToCollect} flowers via collision. Ending
episode.");
                }
                EndEpisode();
            }
        }

        // If agent collides with the ghost
        if (collision.gameObject.CompareTag("Ghost"))
        {
            AddReward(penaltyForGhostProximity);
            if (enableDebugLogs)
            {
                        Debug.Log($"Collision with Ghost: {penaltyForGhostProximity} penalty
applied.");
            }
            EndEpisode(); // Ensure episode termination upon collision
        }

        // Handle Wall Collision
        if (collision.gameObject.CompareTag("Wall"))
        {
            AddReward(penaltyForWallCollision);
            if (enableDebugLogs)
            {
                Debug.Log($"Collision with Wall: {penaltyForWallCollision} penalty applied.");
            }
            EndEpisode(); // Terminate the episode upon wall collision
        }

    }

    private void OnDrawGizmos()
    {
        // Only draw Gizmos when the game is playing
        if (!Application.isPlaying) return;

        // Draw line to the nearest flower
        if (nearestFlower != null)
```

```
        {
            Gizmos.color = Color.green;
            Gizmos.DrawLine(transform.position, nearestFlower.position);
            Gizmos.DrawSphere(nearestFlower.position, 0.5f);
        }

        // Draw line to the ghost
        if (ghost != null)
        {
            Gizmos.color = Color.red;
            Gizmos.DrawLine(transform.position, ghost.position);
            Gizmos.DrawSphere(ghost.position, 0.5f);
        }

        // Visualize Wall Ray Sensor
        if (wallRaySensor != null)
        {
            foreach (var angle in rayAngles)
            {
                Vector3 direction = Quaternion.Euler(0, angle, 0) * transform.forward;
                Gizmos.color = Color.blue;
                          Gizmos.DrawLine(transform.position, transform.position + direction *
rayLength);
            }
        }

        // Visualize Flower Ray Sensor
        if (flowerRaySensor != null)
        {
            foreach (var angle in rayAngles)
            {
                Vector3 direction = Quaternion.Euler(0, angle, 0) * transform.forward;
                Gizmos.color = Color.magenta;
                          Gizmos.DrawLine(transform.position, transform.position + direction *
rayLength);
            }
        }
    }
}
```

Tablas de recompensas y acciones del agente

Los valores de esta tabla de acciones y recompensas es aproximada, puesto que he ido probando entre múltipes opciones para ver cómo se comportaba el agente

Acciones
Simplemente las necesarias para moverse: avanzar, retroceder, rotar izquierda y rotar derecha

Recompensas	
Avanzar hacia objetivo (acercarse a *flower*)	0.1
Distanciarse del objetivo (alejarde de *flower)*	-0.05
Moverse sin objetivo	-0.01
Penalización por estar en el área del fantasma	-0.5
Recompensa por recoger una flor	2.0

Lista de las distintas sesiones de entrenamiento

Iniciales
mlagents-learn config/AgentRunner_config.yaml --run-id=AgentRunner

mlagents-learn config/AgentRunner_config.yaml --run-id=AgentRunner_M2 --force

Intento de paralelizar
mlagents-learn config/AgentRunner_config.yaml --env "D:\Desktop D\0- Máster UOC\2-Inteligencia Artificial\Unity\Build\Artificial_Intelligence.exe" --run-id=AgentRunner_M2 --num-envs=4 --force

Uso de Currículum Learning (CL)
mlagents-learn config/AgentRunner_config.yaml --run-id=AgentRunner_CV2 --force

mlagents-learn config/AgentRunner_config_V2.yaml --run-id=AgentRunner_CV3 --force

mlagents-learn config/AgentRunner_config_V2.yaml --run-id=AgentRunner_CV4 --resume

mlagents-learn config/AgentRunner_config_V3.yaml --run-id=AgentRunner_CV6 --force

mlagents-learn config/AgentRunner_config_V2.yaml --run-id=AgentRunner_CV8 --resume

mlagents-learn config/AgentRunner_config_V3.yaml --run-id=AgentRunner_CV9 --force

mlagents-learn config/AgentRunner_config_V3.yaml --run-id=AgentRunner_CV10 --resume

mlagents-learn config/AgentRunner_config_V4.yaml --run-id=AgentRunner_CV11 --force

Simplificando entorno:
ppo
mlagents-learn config/AgentRunner_config_V4.yaml --run-id=AgentRunner_Basic1 --resume

mlagents-learn config/AgentRunner_config_V4.yaml --run-id=AgentRunner_Basic3 --resume

sac
mlagents-learn config/AgentRunner_config_V3.yaml --run-id=AgentRunner_Basic2 --resume

mlagents-learn config/AgentRunner_config_V3.yaml --run-id=AgentRunner_Basic4 --resume

Conclusiones

Si tuviera que definir con una palabra esta actividad sería *frustración.* Y no es por culpa del temario, siquiera por mis resultados; es porque personalmente me apasiona la inteligencia artificial (la tecnología de las redes neuronales más correctamente) y de verdad deseaba hacer esta actividad por este ejercicio. Lo que verdaderamente me ha frustrado ha sido quemarme en el proceso y no estar satisfecho con los resultados.

Definitivamente, desarrollar modelos ya es de por sí una tarea compleja, pero hacerlo en un entorno interactivo como el de los videojuegos lo es aún más. Mis respetos a todos aquellos que lo dominan y se dedican a ello. Y perdón por la longitud que ha adquirido este documento, pero sentía que debía plasmar toda mi experiencia con este trabajo.

Webgrafia

1. Python Release Python 3.10.11. (n.d.). Python.org. https://www.python.org/downloads/release/python-31011/
2. Adang. (n.d.). PythonWindows/3.10.12/python-3.10.12-amd64-full.exe at master · adang1345/PythonWindows · GitHub. https://github.com/adang1345/PythonWindows/blob/master/3.10.12/python-3.10.12-amd64-full.exe
3. Anaconda. (2024, October 24). Download Now | Anaconda. https://www.anaconda.com/download/success
4. Omar Santiago. (2021, July 21). mlagent start, tensorboard start, trainer config file [Video]. YouTube. https://www.youtube.com/watch?v=i3LSP2kDb7Q
5. ML-agents/docs/Training-Curriculum-Learning.md at master · gzrjzcx/ML-agents. (n.d.). GitHub. https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Curriculum-Learning.md
6. Technologies, U. (n.d.). Designing a Learning Environment - Unity ML-Agents Toolkit. https://unity-technologies.github.io/ml-agents/Learning-Environment-Design/
7. ml-agents/docs/Training-ML-Agents.md at develop · Unity-Technologies/ml-agents. (n.d.). GitHub. https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-ML-Agents.md
8. OpenAI. (2025). ChatGPT (versión 10 de enero) [Large language model]. https://chat.openai.com/chat
9. Conversación de ChatGPT para solución de errores de compilación, proporcionar ideas y limpiar el código: https://chatgpt.com/share/6782bfcc-ed58-800b-89a7-e737d9cae2c2 y https://chatgpt.com/share/6783a2f6-a56c-800b-9317-69378af4c94b

Assets

- Low Res Flora v1 | 3D Vegetation | Unity Asset Store. (n.d.). Unity Asset Store. https://assetstore.unity.com/packages/3d/vegetation/low-res-flora-v1-191170