Harvest Saga

You have no enemies...

Chi Siamo

Luigi Consiglio (0522501894)

Eljon Hida (0522501890)

Idea

L'ispirazione per il progetto Harvest Saga nasce dal manga storico **Vinland Saga**, che racconta la storia di **Thorfinn Karlsefni,** un giovane vichingo costretto a diventare un guerriero sin dall'infanzia. Cresciuto in un mondo dominato da conflitti e vendetta, Thorfinn trascorre anni sul campo di battaglia.

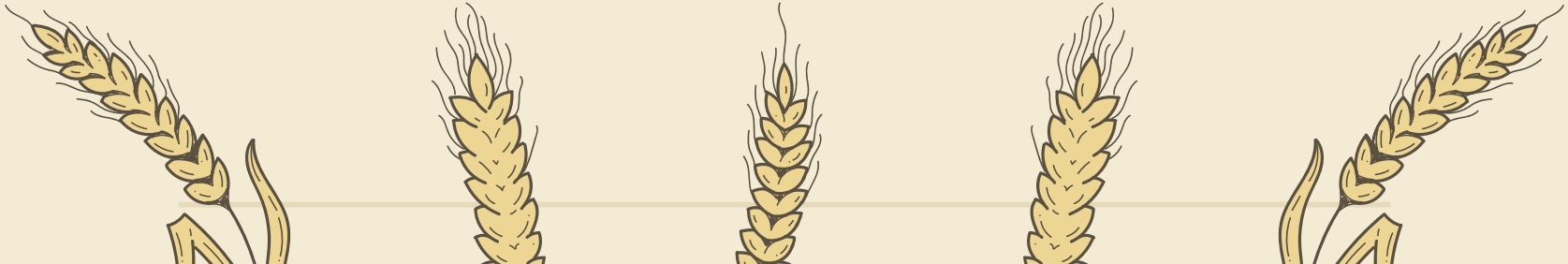
Tuttavia, il corso della sua vita cambia radicalmente quando, devastato dai sensi di colpa e dalla consapevolezza della brutalità della guerra, sceglie di abbandonare la violenza e dedicarsi alla costruzione di un **mondo di pace** .

Idea

Harvest Saga è un progetto che simula un'esperienza di **Smart Agriculture** un concetto che punta a migliorare le pratiche agricole attraverso l'uso di tecnologie avanzate come l'intelligenza artificiale, il machine learning e l'automazione. In cui un agente virtuale, addestrato tramite **Unity ML-Agents** , impara a mietere fasci di grano in modo efficiente.

L'idea di Smart Agriculture si riflette nel comportamento dell'intelligenza artificiale, che simula un approccio ottimizzato alla coltivazione: l'agente non solo miete il grano, ma calcola le **traiettorie migliori** per **raggiungere gli obiettivi** ed evitare gli ostacoli.

Sommario



01

Obiettivi

Obiettivi

L'obiettivo è riprodurre un ambiente agricolo realistico e dinamiche di raccolta basate su strategie di apprendimento automatico, più precisamente:

1. Simulare un ambiente agricolo : si punta all'avere piante di grano, strumenti di mietitura e un ambiente che possa ricordare quello di un campo coltivato.

2. Implementare un AI competente : utilizzando Unity ML Agents per addestrare l'agente virtuale a mietere il grano e trasportarlo sull'apposita piattaforma, ad esempio un carretto.

02

Reinforcement Learning

Reinforcement Learning - Definizione

Il Reinforcement Learning è un tipo di **apprendimento automatico**
che si concentra sull'addestramento di agenti che interagiscono con l'ambiente.

In RL, l'agente osserva l'ambiente attraverso una serie di **osservazioni** , poi prende **decisioni** su quali azioni intraprendere e infine esegue quell' **azione** associando ad essa una ricompensa.

L'obiettivo dell'apprendimento per rinforzo è apprendere una **policy** , che è essenzialmente una mappatura dalle osservazioni alle azioni.

- **Observation** : l'agente riceve informazioni dall'**ambiente** attraverso le osservazioni.

- **Decision** : l'agente utilizza le osservazioni per decidere quale azione intraprendere.

- **Action** : una volta presa una decisione, l'agente esegue l'azione scelta nell'ambiente. Le azioni possono essere **continue** o **discrete** a seconda della complessità dell'ambiente e degli agenti.

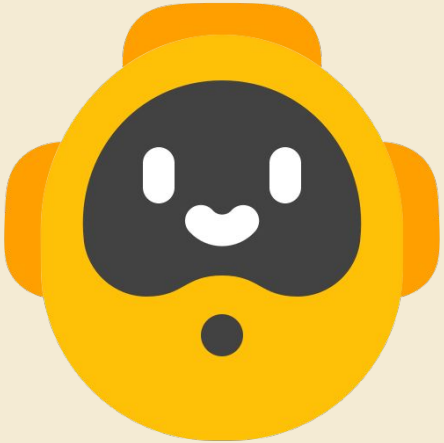
- **Reward** : **l'agente** riceve una **ricompensa** dell'ambiente, che valuta quanto bene l'azione ha performato. L'agente utilizza questa informazione per adattare la sua politica e migliorare le future decisioni.

Reinforcement Learning - ML-Agents

Il **ML-Agents Toolkit** fornisce gli strumenti necessari per usare Unity come motore di simulazione per l'addestramento degli agenti AI. Comprende cinque componenti principali:

- **Learning Environment** : è la scena di Unity, che include il campo e i personaggi. Qui l'agente osserva, agisce e impara. La stessa scena può essere usata sia per l'addestramento che per i test.
- **Python Low-Level API** : è un'interfaccia Python che permette di controllare l'ambiente di apprendimento. Questa API vive fuori da Unity e comunica con il gioco tramite un componente dedicato.
- **External Communicator** : collega l'ambiente di Unity con l'API Python, permettendo la comunicazione tra i due.
- **Python Trainers** : contiene gli algoritmi di machine learning che permettono di addestrare gli agenti.

L'abbiamo scelto perché:



Permette all'**agente** di **imparare** attraverso tentativi ed errori, adattando la propria strategia in base alle ricompense ricevute per la raccolta efficiente del grano o alle punizioni per percorsi meno ottimali.

Inoltre, il Reinforcement Learning è particolarmente efficace nel gestire situazioni incerte e non lineari, come la disposizione casuale degli ostacoli nel campo, rendendolo una **soluzione ideale** per l'addestramento dell'agente virtuale.

03

Tecnologie Usate

Tecnologie Usate

Tutte le tecnologie utilizzate per lo sviluppo di questo progetto

Unity	È un motore grafico 3D utilizzato per creare ambienti simulativi interattivi. Nel progetto Harvest Saga permette di progettare il campo agricolo e gestire il comportamento dell'agente.
C#	È il linguaggio di programmazione principale utilizzato in Unity per implementare logiche di gioco e comportamenti dell'agente.
PyTorch	È una libreria di Machine Learning utilizzata per costruire e addestrare modelli AI. In Harvest Saga, è sfruttata per il reinforcement learning dell'agente tramite Unity ML-Agents.
Tensorboard	È uno strumento di visualizzazione che permette di monitorare l'addestramento dell'agente. Consente di analizzare i progressi e le performance in tempo reale durante il training.
Visual Studio Code	È un editor di codice utilizzato per scrivere e modificare codice in vari linguaggi. Offre integrazioni con Unity e strumenti utili per il debugging.

04

Risultati & Demo Progetto

Codice: FarmerAgent

```
public class FarmerAgent : Agent
    public override void OnActionReceived(ActionBuffers actions)
    {
        if (frozen) return;
        // Ottieni le azioni continue
        var continuousActions = actions.ContinuousActions;
        var discreteActions = actions.DiscreteActions;

        // Calcola il vettore di movimento in avanti rispetto alla rotazione attuale
        Vector3 move = transform.forward * continuousActions[1];
        // Applica il movimento al CharacterController
        characterController.Move(move * speed * Time.deltaTime);

        // Attiva animazione di camminata
        float distanceToWheat = nearestWheat != null
        ? Vector3.Distance(transform.position, nearestWheat.WheatCenterPosition)
        : float.MaxValue;

        // Gestione della camminata
        bool isMoving = continuousActions[1] != 0 && distanceToWheat > 0.5f;
        SetWalkingAnimation(isMoving);

        // Movimento fisico
        if (isMoving)
        {
            characterController.Move(move * speed * Time.deltaTime);
        }
```

La funzione OnActionReceived() viene chiamata quando è assegnata un azione che sia dall'utente o dalla rete neurale.

```
public override void CollectObservations(VectorSensor sensor)
{
    // Se nearestWheat è nullo, osserva un array vuoto e ritorna in anticipo
    if (nearestWheat == null)
    {
        sensor.AddObservation(new float[9]);
        return;
    }

    // Osserva la rotazione locale dell'agente (4 osservazioni)
    sensor.AddObservation(transform.localRotation.normalized);

    // Ottieni un vettore dall'agente al grano più vicino
    Vector3 toWheat = nearestWheat.WheatCenterPosition - transform.position;

    // Osserva un vettore normalizzato che punta al grano più vicino (3 osservazioni)
    sensor.AddObservation(toWheat.normalized);

    // Osserva un prodotto scalare che indica se l'agente è rivolto verso il grano (1 osservazione)
    // (+1 significa che l'agente punta direttamente al grano, -1 significa direttamente lontano)
    sensor.AddObservation(Vector3.Dot(transform.forward.normalized, -nearestWheat.WheatUpVector.normalized));

    // Osservare la distanza relativa dall'agente al grano (1 osservazione)
    sensor.AddObservation(toWheat.magnitude / HarvestArea.AreaDiameter);
}
```

La funzione CollectObservation() raccoglie informazioni sull'agente, come la sua rotazione, la direzione verso il grano e la distanza. Questi dati vengono usati dall'AI per capire come muoversi e mietere efficacemente.

Codice: FarmerAgent

```
public class FarmerAgent : Agent
    private void TriggerEnterOrStay(Collider collider)
            // Guarda al grano con questo WheatCollider
            Wheat wheat = harvestArea.GetWheatFromCollider(collider);

            // Ottieni l'istanza di SickleCollision
            SickleCollision sickle = GetComponentInChildren<SickleCollision>();

            if (trainingMode)
            {
                // Calcola il reward per mietere il grano
                float alignmentBonus = .02f * Mathf.Clamp01(Vector3.Dot(transform.forward.normalized,
                AddReward(0.02f + alignmentBonus);
            }

            // Se il grano è nullo passa al prossimo
            if (!wheat.IsWheatActive())
            {
                UpdateNearestWheat();
            }
        }
    }
```

```
public class FarmerAgent : Agent
    private void OnControllerColliderHit(ControllerColliderHit hit)
    {
        if (trainingMode && hit.collider.CompareTag("Boundary"))
        {
            Debug.Log("Collisione con Boundary!");
            AddReward(-0.5f);
        }
        else if (trainingMode && hit.collider.CompareTag("Ostacolo"))
        {
            Debug.Log("Collisione con Ostacolo!");
            AddReward(-0.3f);
        }
    }
```

La funzione TriggerEnterOrStay() gestisce tutte le collisioni con un Trigger ed un Collider, assegnando una **reward positiva** se triggera il grano.

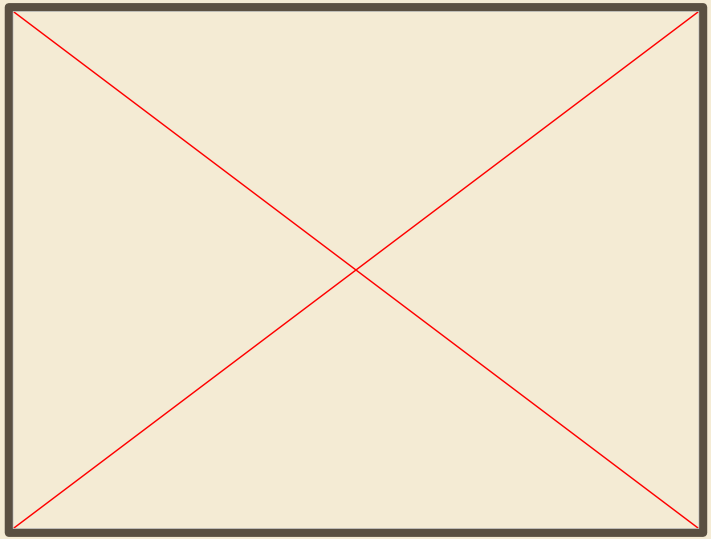
La funzione OnControllerColliderHit() viene chiamata quando l'agente collide con un oggetto solido diverso dal grano, assegnando una **reward negativa** .

Oggetti di Scena: Personaggio

È l'agente del nostro sistema che dovrà apprendere come riuscire a **mietere** un numero di fasci di **grano** pari a 12 in un **tempo limite** di 150 secondi.

È costituito da un body e da un **sickle** (falcetto) ed il suo comportamento è definito in FarmerAgent.

L'agente per permettere l'apprendimento, avrà dei sensori definiti con la componente di **Unity Ray Perception Sensor 3D** .



Il modello, per fare in modo che fosse il più simile possibile (pur sempre restando nei limiti di uno stile LowPoly) al Thorfinn del manga, è stato gentilmente creato e offerto da un caro amico:

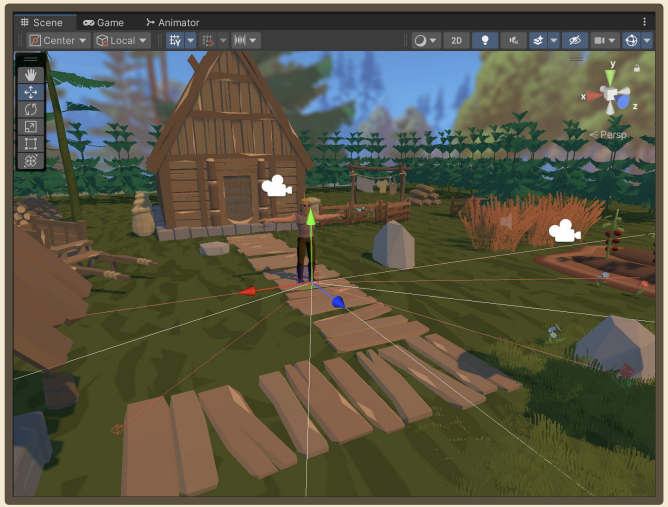
Andrea(3D Artist) -> Link IG

Oggetti di Scena: Personaggio

Di seguito è mostrata una foto con i vari Raycast associati all'agente. Avrà due di questi oggetti, il primo per controllare lateralmente e di fronte a sé con un totale di 9 raggi.

Il secondo per avere visione di eventuali ostacoli dietro di sé con un solo raggio.

I sensori sono una componente di Unity, denominata Ray Perception Sensor 3D, che permette all'agente di avere una visione dell'ambiente circostante.

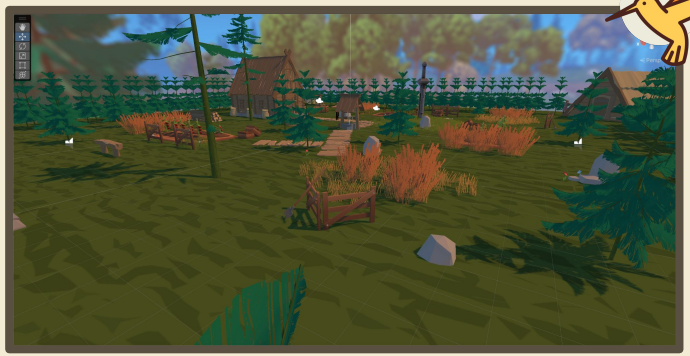


Oggetti di Scena: Campo

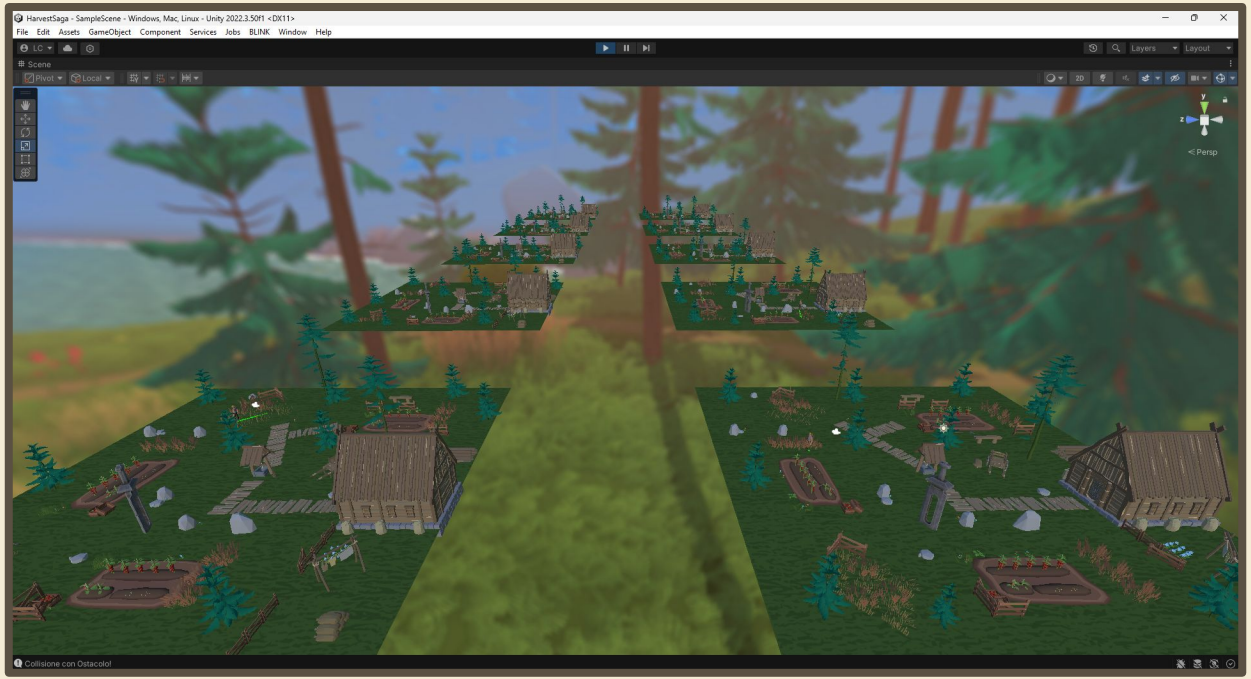
Di seguito abbiamo due foto della scena da due inquadrature diverse in modo da mostrare, non solo l'agente ma anche i vari ostacoli (come alberi, rocce e staccionate) e obiettivi (i vari fasci di grano).

La scena presenta inoltre altre finezze estetiche per rendere il tutto molto più gradevole alla vista e cercare di immedesimare il più possibile coloro che assistono alla simulazione, tra queste:

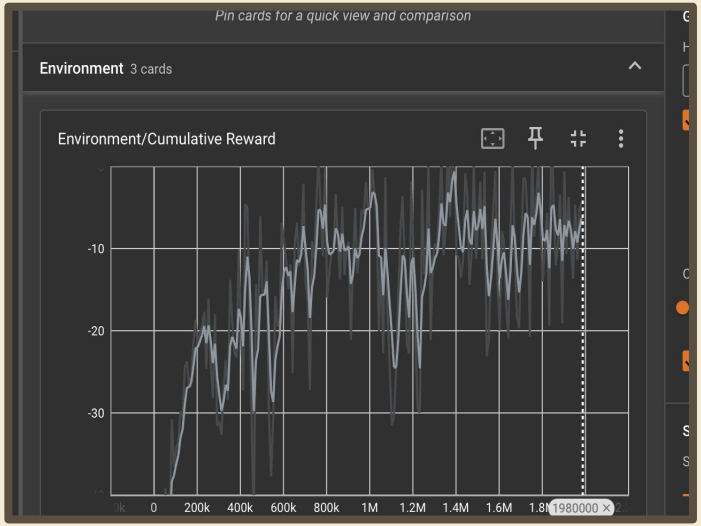
- Case in legno;
- Fiori;
- Campi coltivati con ortaggi;
- Assi di legno;

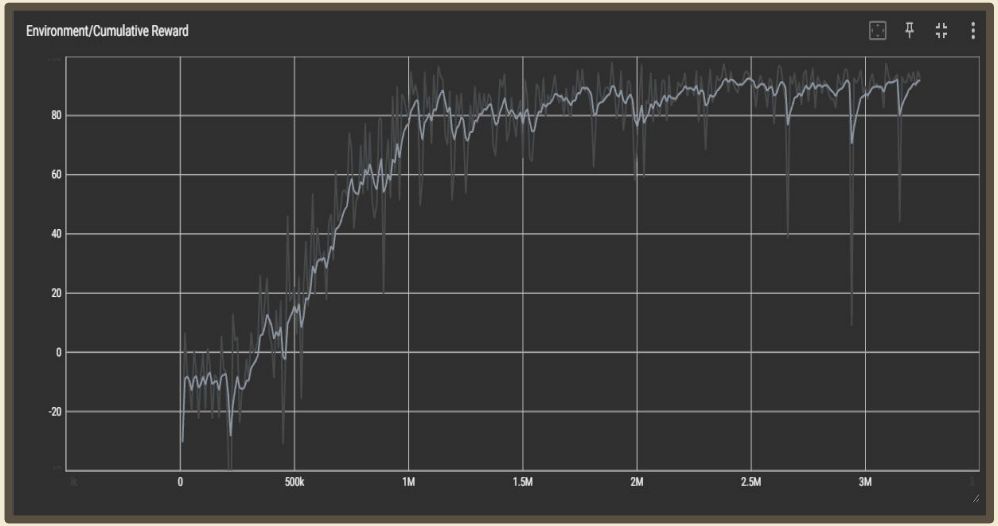


Training e Risultati



Training e Risultati





Il reward cumulativo rappresenta la **somma delle ricompense ottenute** dall'agente durante una serie di azioni, dunque l'agente tiene conto di tutte le ricompense ottenute fino a quel momento. Nel RL, il reward cumulativo è spesso utilizzato per ottimizzare l'agente a lungo termine poiché descrive quanto bene sta facendo in generale.

Grazie per l'attenzione