
FIAsarum
Progetto FIA 2025/2026

CHI SIAMO?

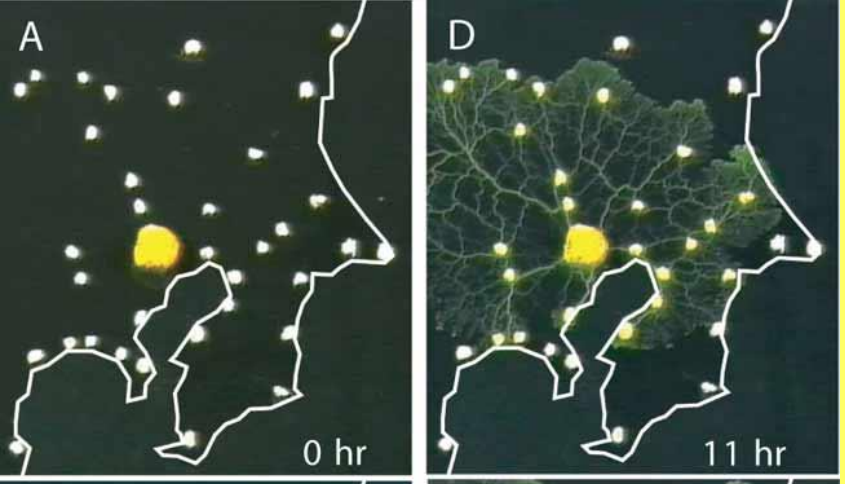
Setola Angela

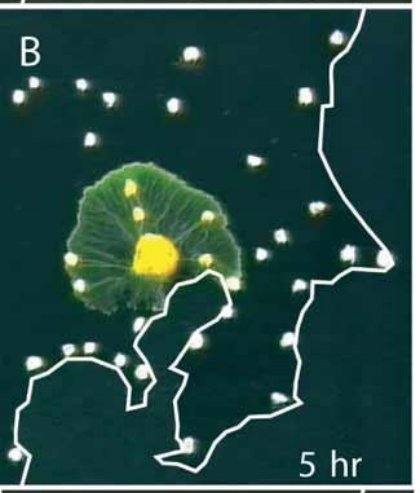
Farace Mirko

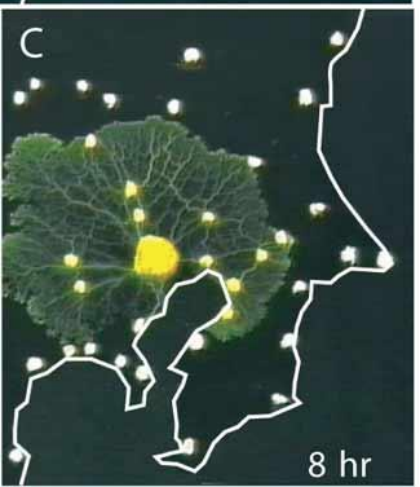
INTRODUZIONE:

Da questo video siamo rimasti affascinati dal comportamento peculiare di questo organismo e ci siamo posti la domanda: «e se la rendessimo digitale?»

Video da Barbascura X: Lo strano organismo che risolve labirinti e prende decisioni.
(Link: https://youtu.be/xieWiuPv7U0?si=BCSLNf-0M9Xqi_YV)







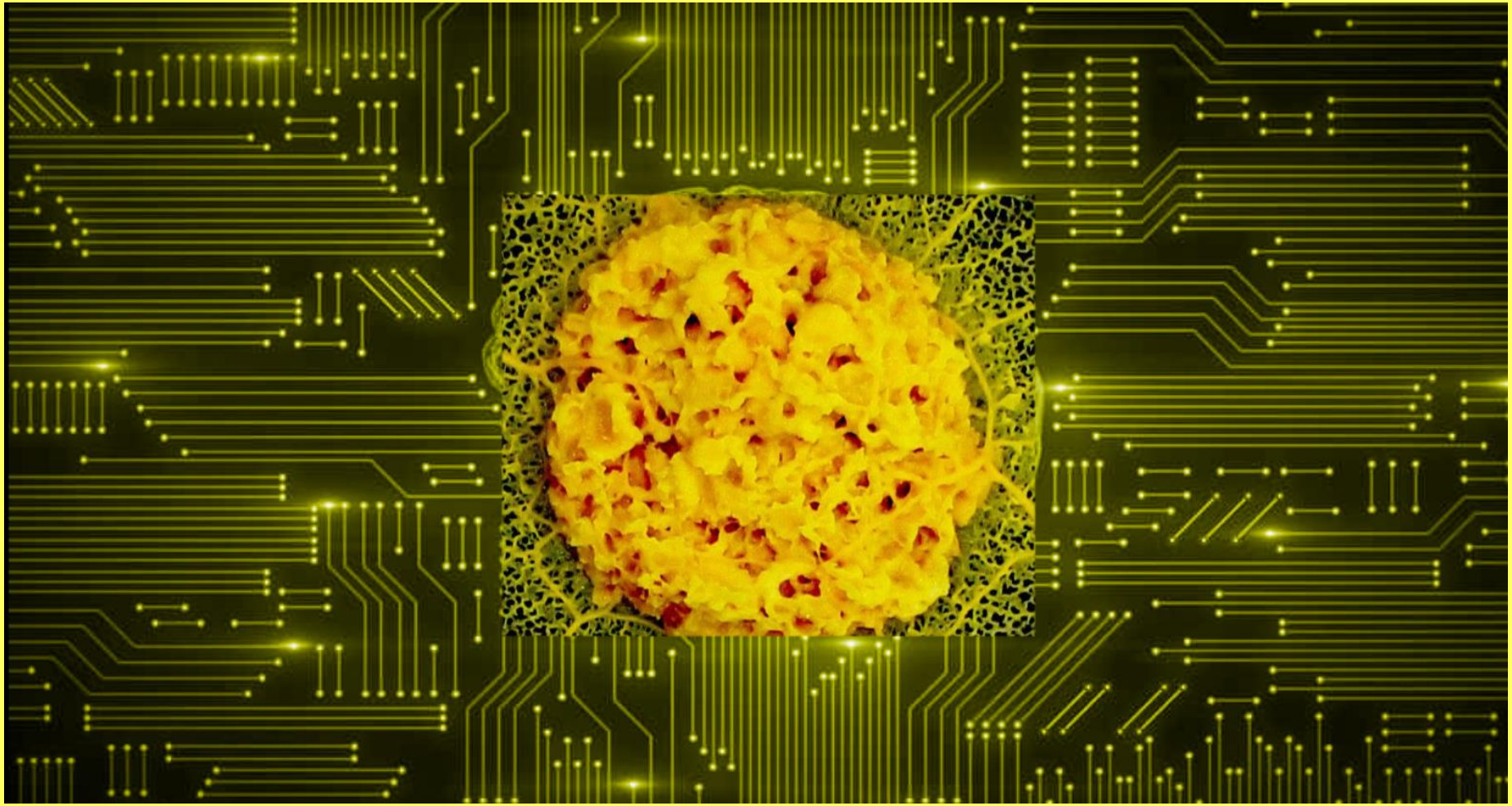
MA NON SOLO!

Questa melma è stata utilizzata anche per ricreare la rete metropolitana di tokyo. L'opera è stata possibile dagli studenti dell'università di Hokkaido e hanno dato del cibo in punti specifici, che corrispondono al luogo in cui si trovano le varie stazioni. Il cibo è servito quindi come una «meta» da raggiungere e, dopo quasi 30 ore, la melma è riuscita a ricreare una possibile rete metropolitana di Tokyo. Anzi, non è una possibile rete, ma ha ricreato alla perfezione la rete metropolitana della metropoli Giappone! Il che è impressionante.

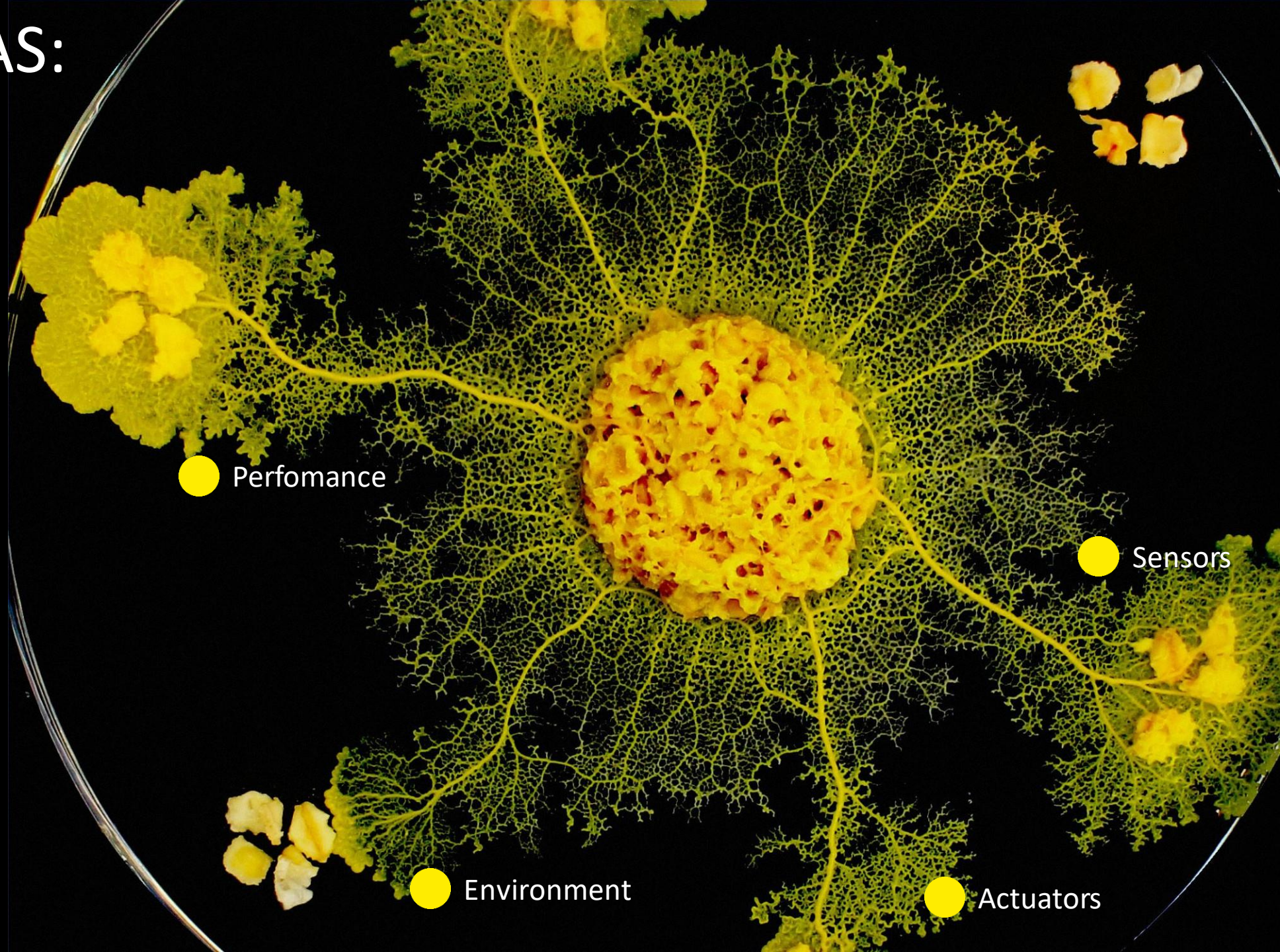
Possiamo quindi dire che alla fine hanno dato un punto di partenza alla melma (dove l'hanno posata) e hanno dato un punto di arrivo. Noi faremo una cosa simile: dato un punto di partenza, cercheremo di raggiungere la meta presente nella griglia, nel modo più efficiente possibile!

OBIETTIVO:

Creare un’IA in grado di simulare il comportamento di ricerca della muffa policefala, utilizzando gli algoritmi visti a lezione



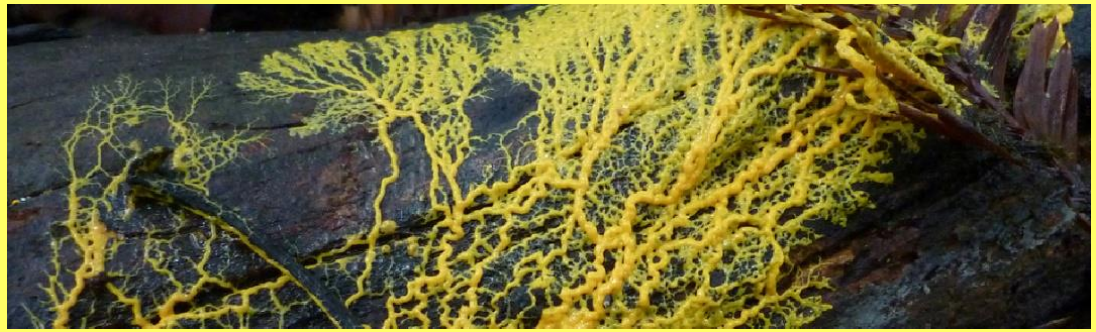
SPECIFICA PEAS:



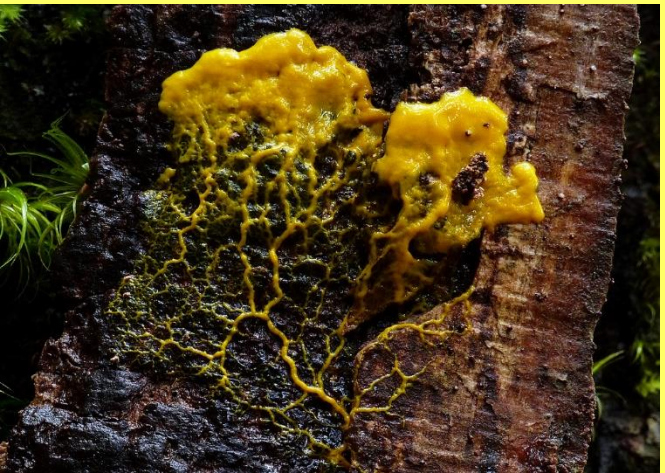
SPECIFICHE DELL'AMBIENTE

SINGOLO AGENTE

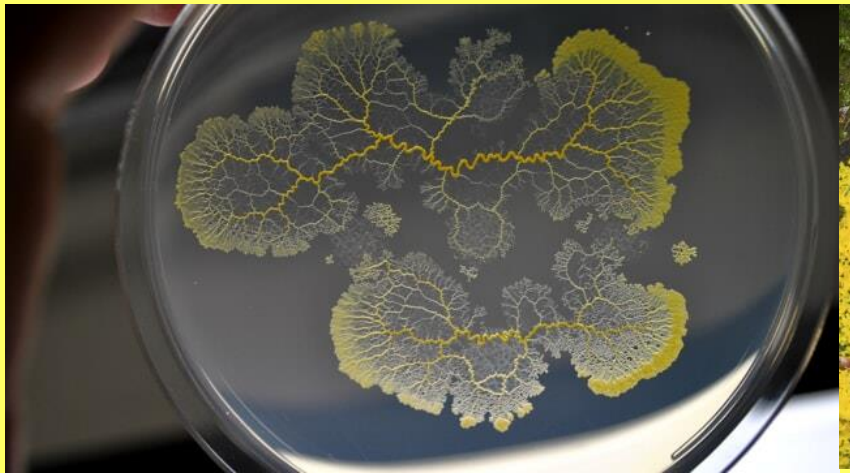
PARAIZIALMENTE OSSERVABILE



STATICO



EPISODICO



ANALISI DEL PROBLEMA

Stato iniziale: il punto di partenza e il punto d'arrivo sono scelti casualmente in una griglia

Descrizione azioni possibili: L'agente può muoversi in 8 direzioni. In alto, in basso, a destra, a sinistra, in basso a destra, in basso a sinistra, in alto a destra, in alto a sinistra.

Test Obiettivo: L'obiettivo della muffa è quello di raggiungere il cibo con il miglior percorso possibile

Modello di transizione: Si deve controllare ad ogni azione se si è raggiunto o meno il «traguardo»

Costo cammino: ogni passo nell'ambiente costa 1

COME ABBIAMO AFFRONTATO IL PROBLEMA?

Abbiamo affrontato il problema usando due algoritmi di ricerca, uno a ricerca informata e uno a ricerca non informata:

Algoritmo di Ricerca A*

Algoritmo di ricerca in Ampiezza

RICERCA A*

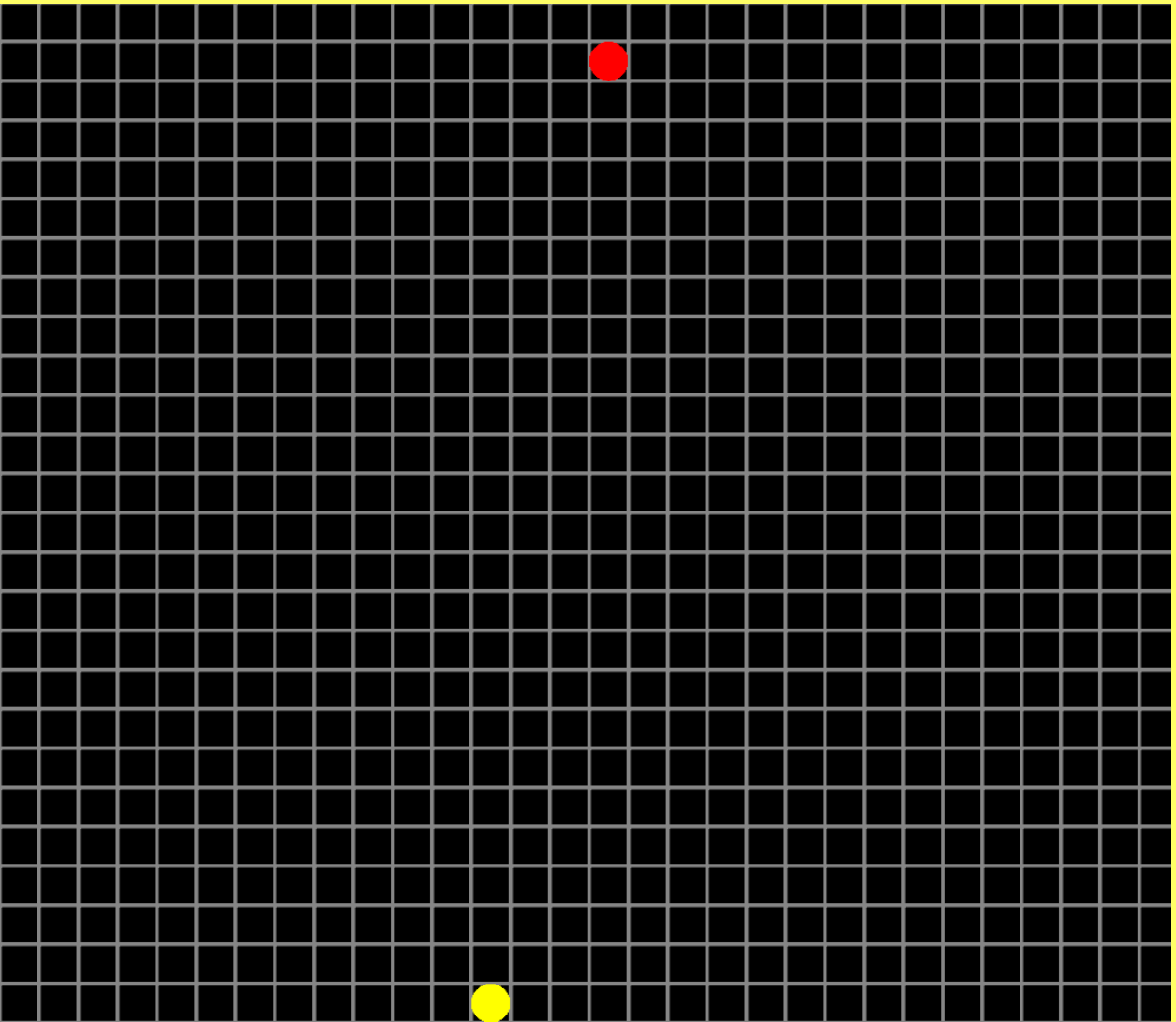
L'algoritmo usa tre funzioni: g(), che corrisponde al costo reale per raggiungere il nodo obiettivo a partire da quello iniziale, h(), che corrisponde al costo stimato per raggiungere il nodo obiettivo a partire dal nodo attuale, in fine abbiamo f() che è la somma dei due e corrisponde alla stima del costo del percorso più adatto.

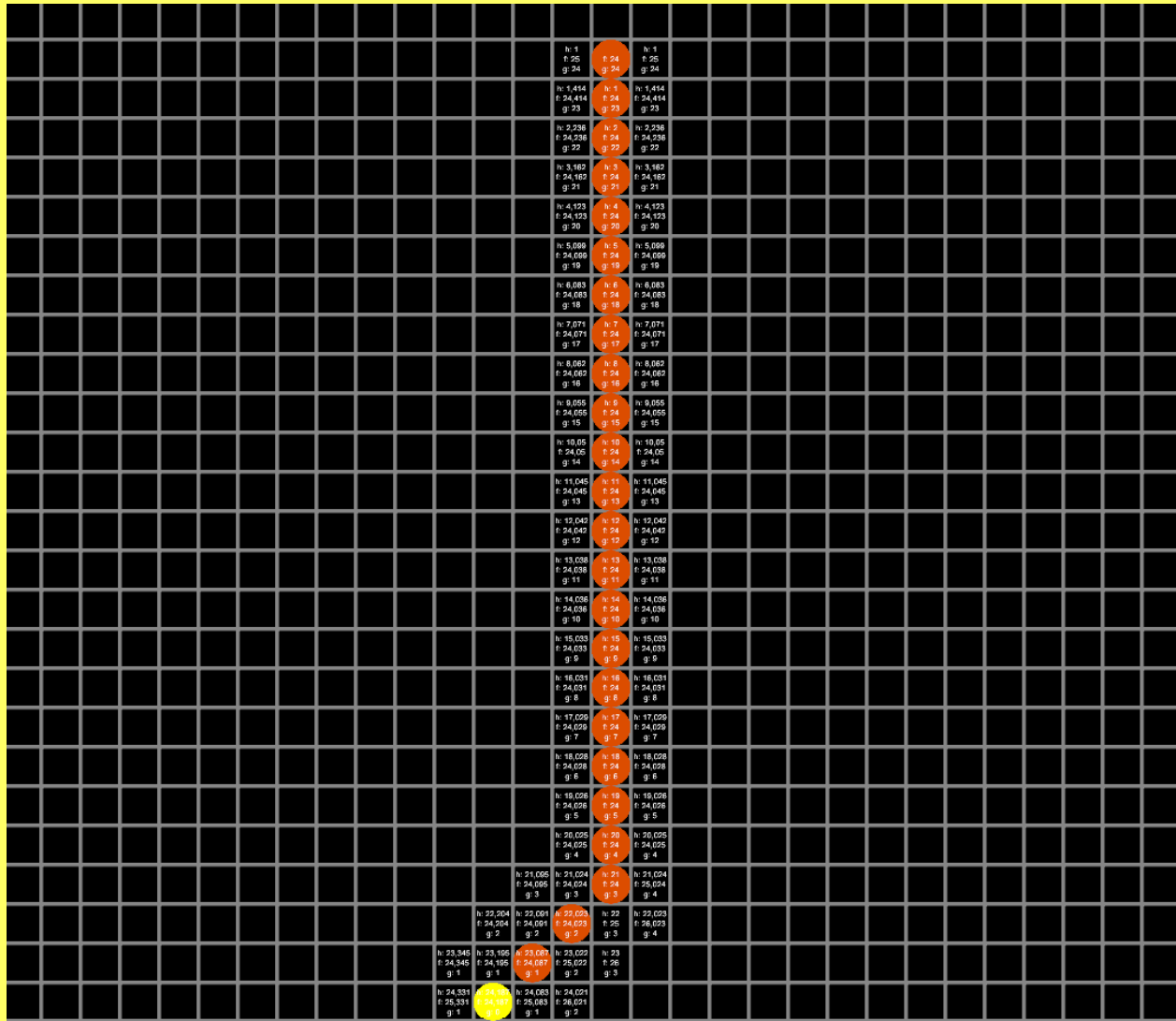
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AStar : MonoBehaviour
{
    public static List<Node> FindPathToGoal(Node start, Node goal)
    {
        PriorityQueue<Node> openList = new PriorityQueue<Node>();        // frontiera
        HashSet<Node> closedList = new HashSet<Node>();     // nodi esplorati

        // inizializziamo i valori del nodo di partenza
        start.g = 0f;
        start.h = EuclideanDistance(start, goal);
        start.f = start.h;
        start.parent = null;

        openList.Enqueue(start, start.f);  // inizializziamo la frontiera col nodo di partenza

        while (openList.Count > 0)
        {
            Node current = openList.Dequeue();
            if (current == goal)
                return ReconstructPath(current);

            closedList.Add(current);

            // controlliamo i vicini e calcoliamo le funzioni f, g, h di ognuno di loro
            foreach (Node n in current.neighbors)
            {
                if (!n.isObstacle && !closedList.Contains(n))
                {
                    float tentativeG = current.g + 1f;

                    if (tentativeG < n.g)
                    {
                        n.g = tentativeG;
                        n.h = EuclideanDistance(n, goal);
                        n.f = n.g + n.h;
                        n.parent = current;
                        openList.Enqueue(n, n.f);
                    }
                }
            }
        }

        return null;    // nessun percorso trovato
    }

    /* ricostruiamo tutto il percorso dal nodo iniziale al nodo goal */
    public static List<Node> ReconstructPath(Node current)
    {
        List<Node> newPath = new List<Node>();

        while (current != null)
        {
            newPath.Add(current);
            current = current.parent;
        }
        newPath.Reverse();

        return newPath;
    }

    /* la nostra euristica; è stata scelta la distanza Euclidea perché è la scelta migliore in quanto la muffa si sposta in 8 direzioni */
    public static float EuclideanDistance(Node a, Node b)
    {
        return Mathf.Sqrt(Mathf.Pow(a.transform.position.x - b.transform.position.x, 2) + Mathf.Pow(a.transform.position.y - b.transform.position.y, 2));
    }

}
```

RICERCA A*

Ecco un esempio di applicazione dell'algoritmo

Vi sono anche descritti tutti i valori delle funzioni calcolate durante la ricerca





RICERCA IN AMPIEZZA

Iniziando dal punto di partenza, andiamo a esplorare le celle vicine. Queste celle visitate vengono salvate in una coda e se una cella corrisponde alla cella di arrivo, allora si creerà il percorso per raggiungerla. Se invece non lo dovesse trovare allora continuerà la ricerca finché non lo troverà.

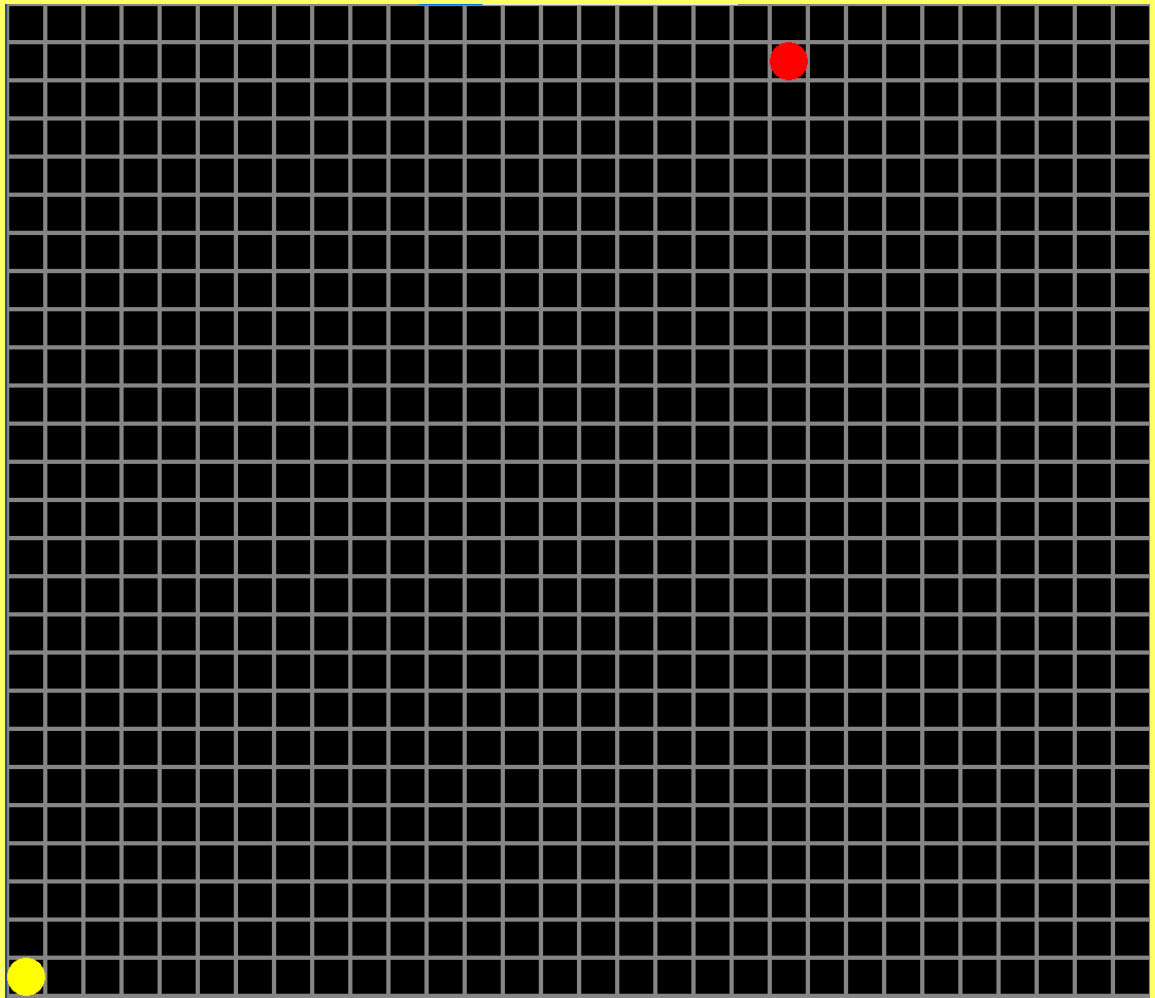
```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class BFS : MonoBehaviour
{
    public static List<Node> exploredNodes;
    public static List<Node> FindPathToGoal(Node root)
    {
        Queue<Node> openList = new Queue<Node>();     // frontiera
        HashSet<Node> closedList = new HashSet<Node>();    // nodi esplorati

        // inizializziamo la frontiera col nodo di partenza
        root.parent = null;
        openList.Enqueue(root);

        while(openList.Count > 0)
        {
            Node current = openList.Dequeue();
            if (TestGoal(current))
            {
                exploredNodes = closedList.ToList<Node>();
                return ReconstructPath(current);
            }

            closedList.Add(current);

            foreach (Node n in current.neighbors)
            {
                if (!n.isObstacle && !closedList.Contains(n) && n != null)
                {
                    if (!openList.Contains(n))
                    {
                        openList.Enqueue(n);
                    }

                    n.parent = current;
                    n.previouslyExploredByBFS = true;
                }
```

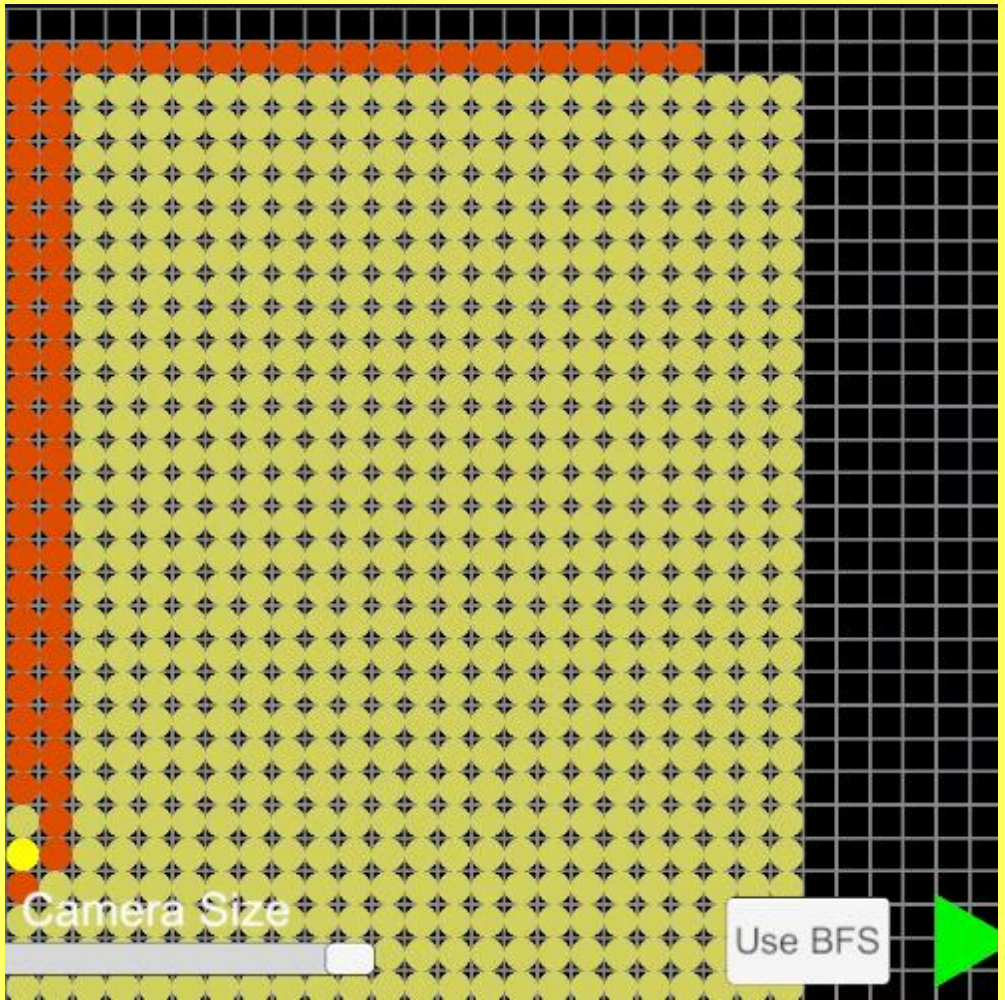
```
            }
        }

        return null;   // nessun percorso trovato
    }

    /* ricostruiamo tutto il percorso dal nodo iniziale al nodo goal */
    public static List<Node> ReconstructPath(Node current)
    {
        List<Node> newPath = new List<Node>();

        while (current != null)
        {
        newPath.Add(current);
        current = current.parent;
        }
        newPath.Reverse();

        return newPath;
    }

    public static bool TestGoal(Node n)
    {
        if(GameObject.Find("Food(Clone)").transform.position == n.transform.position)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

RICERCA IN AMPIEZZA

Ecco un esempio di applicazione dell'algoritmo

Abbiamo anche aggiunto un effetto grafico che ci permette di vedere la ricerca in tempo reale!





CONCLUSIONE

L’esperienza di progetto per IA è stata divertente, non solo perché ci ha permesso di approfondire sugli argomenti trattati a lezione, ma di farlo a modo nostro con un argomento «curioso».

GRAZIE PER L’ATTENZIONE