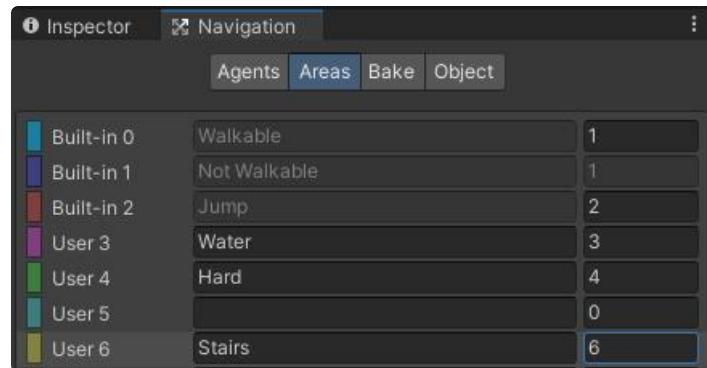


# O primeiro Nav Mesh

## 0.- As Navigation áreas

Agréganse 3 novas áreas ás de *default*: Water, Hard e Stairs que penalizan 3 4 e 6 respectivamente.



	Agents	Areas	Bake
Built-in 0	Walkable	1	
Built-in 1	Not Walkable	1	
Built-in 2	Jump	2	
User 3	Water	3	
User 4	Hard	4	
User 5		0	
User 6	Stairs	6	

Non vin coma mudar a cor das áreas de navegación

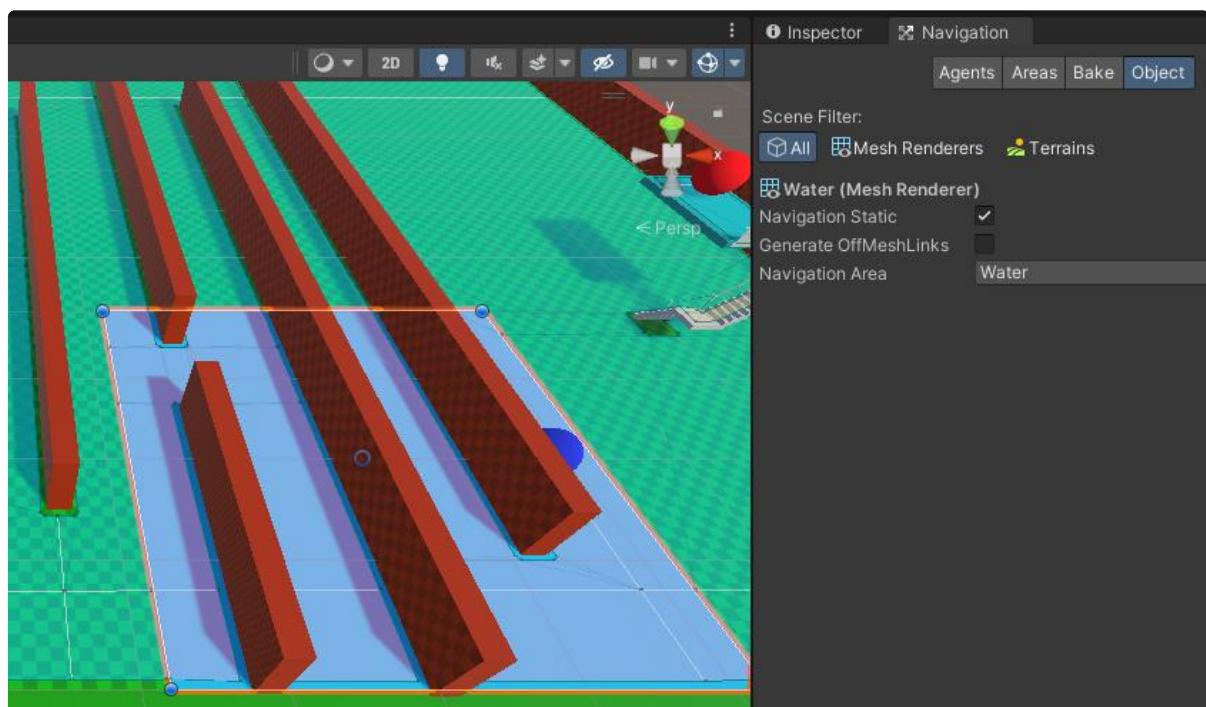
// TODO, preguntar se se pode mudar a cor das áreas do NavMesh e cómo.

Móstrase a asignación das áreas a certos obxectos da escea:

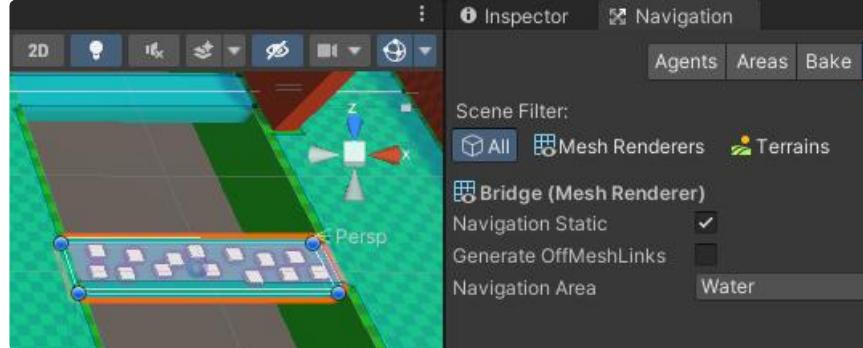
### New area Water

Dificultade 3

Créase un charco cuadrado ao que se lle asigna este nivel de dificultade:



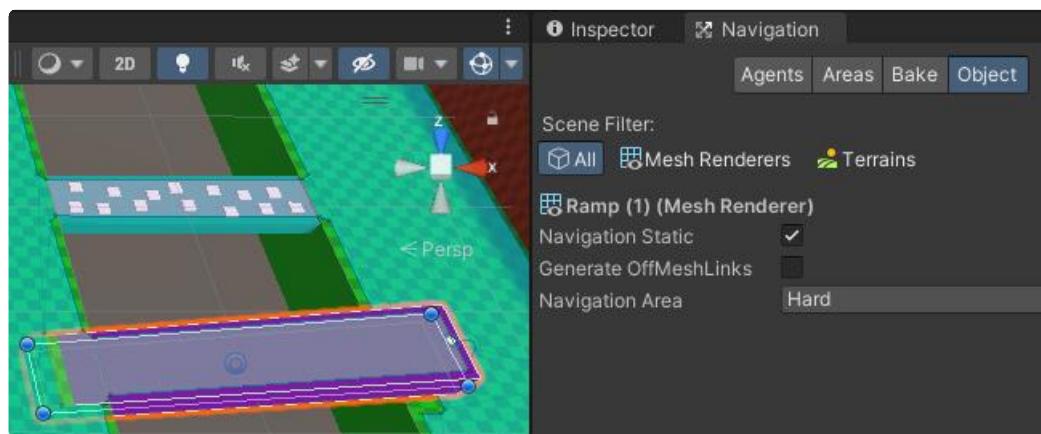
Tamén se lle asigna este nivel de dificultade ao Bridge por interpretar os cuadrados como cristais, p.ex.



### New area Hard

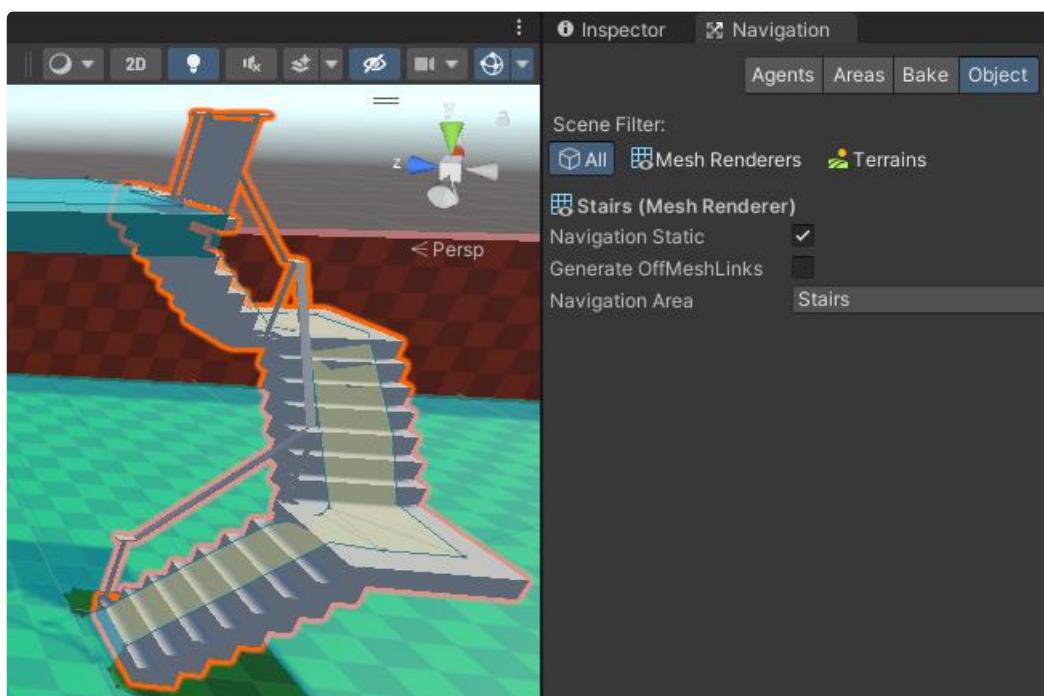
Dificultade 4

Aplícase á Ramp(1) porque atravesa un algo e remata nun salto



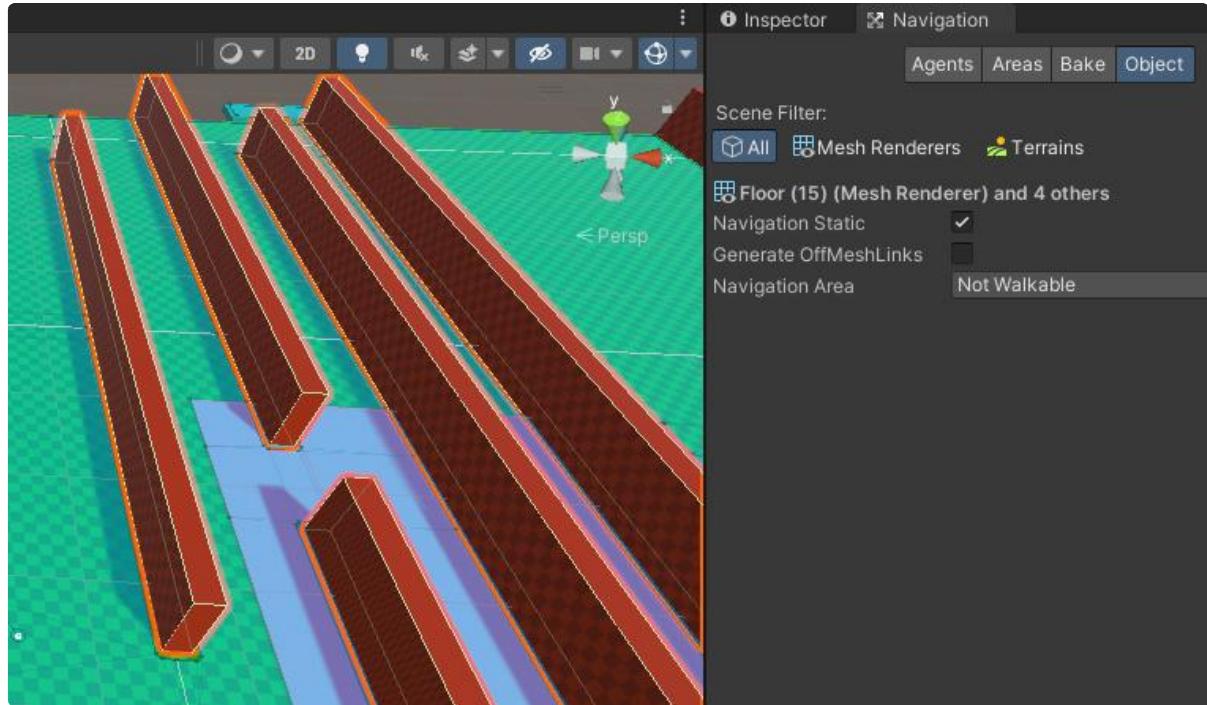
### New area Stairs

Dificultade 6



**Not Walkable areas**

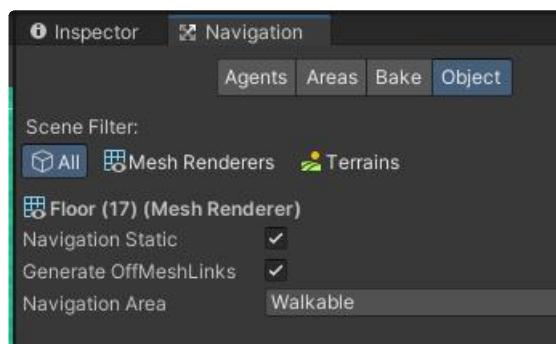
Eliminase a NavMesh da parte superior dos muros

**1.- Offmesh-link**

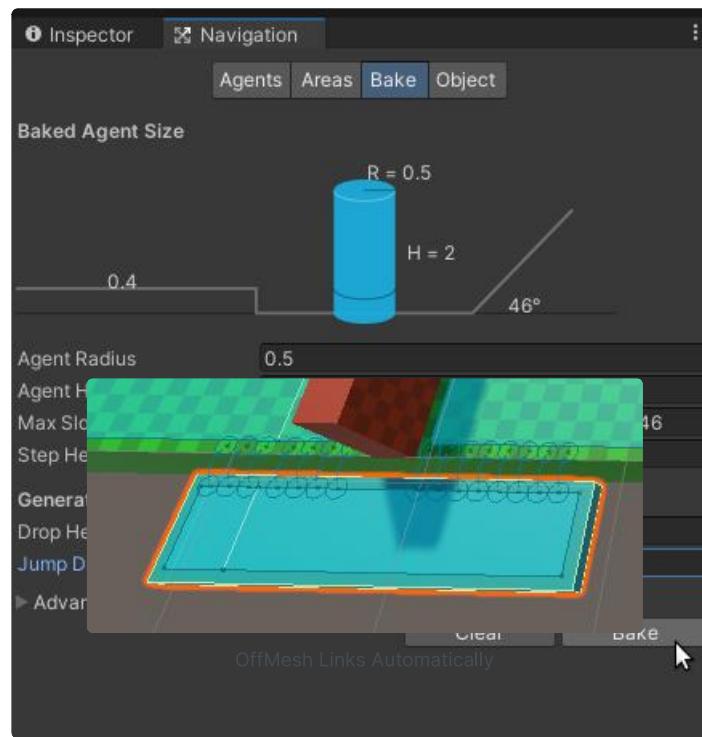
Referencia: [Creating an OffMesh Link](#)

**OffMesh Links Automatically**

Primeiro xeramos os Offmesh-link automáticamente



Entre zonas walkable que estean separadas un máximo de 2 metros



### Manual OffMesh Links

Créase un EmtyObject para anidar 3 OffMesh Links. Sítúanse fora do --LEVEL--

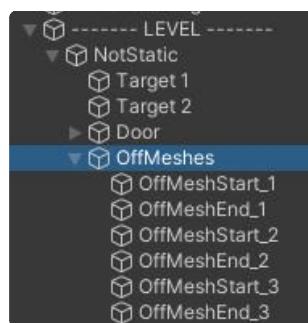
Ordear o elementos na xerarquía

Créase un grupo OffMesh

Os objects OffMesh Links non son estáticos así que, para facilitar no inspector a asignación en masa de elementos estáticos dos que non créase un grupo NotStatic baixo o paquete do prefab (previamente desempaquetado) que chameramos, p.ex. OffMeshes. Todo este grupo e anidados desmárcase de Static



Dentro de este grupo crearemos os 4 saltos de malla que no se xeraron automáticamente

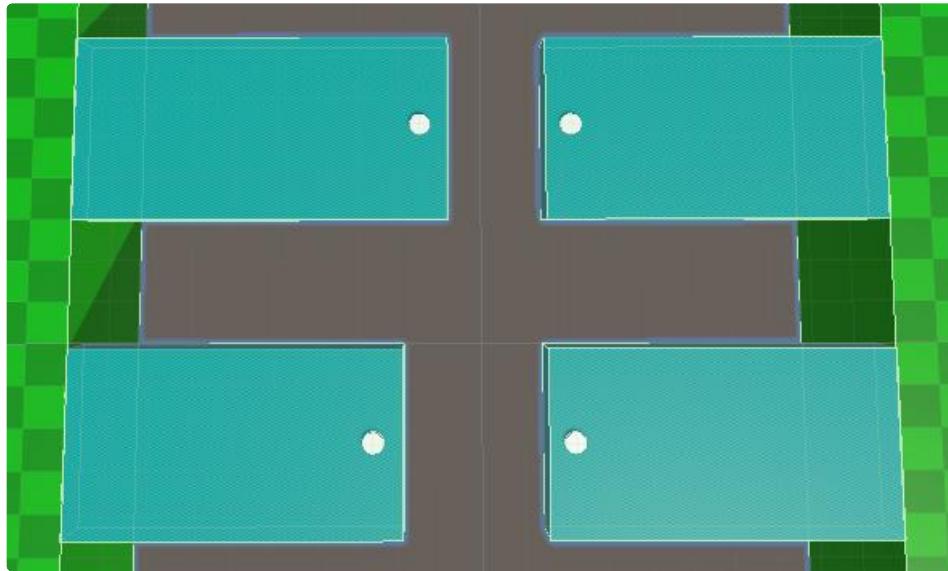


O procedimento

Cada link de salto ten 2 elementos, inicio e fin, e poden ser unidireccionais ou en ambos sentidos:

Seguindo o tutorial créase un cilindro de radio o de un axente standard e de pouca altura. O propósito entendo que é facilitar a localización dos puntos de salto e que se distingan na escea dos OffMesh links automáticos.

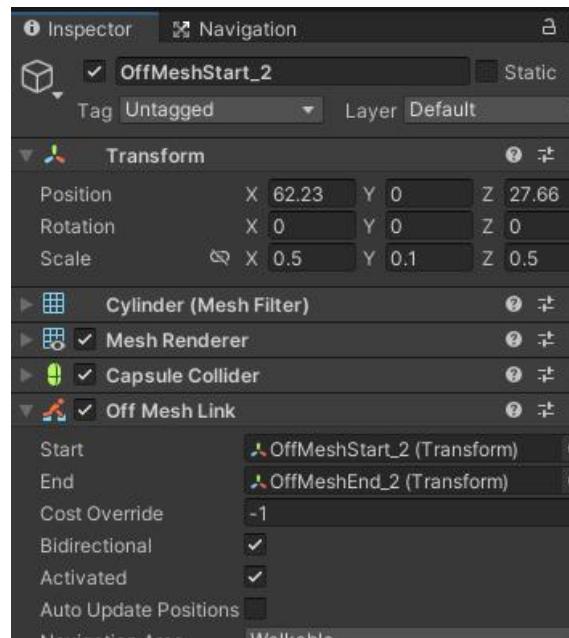
Asegurámonos que este obxecto no estea marcado como Static e duplicámolo. Situamos ambos obxectos coma puntos de A e punto B do salto na escea.

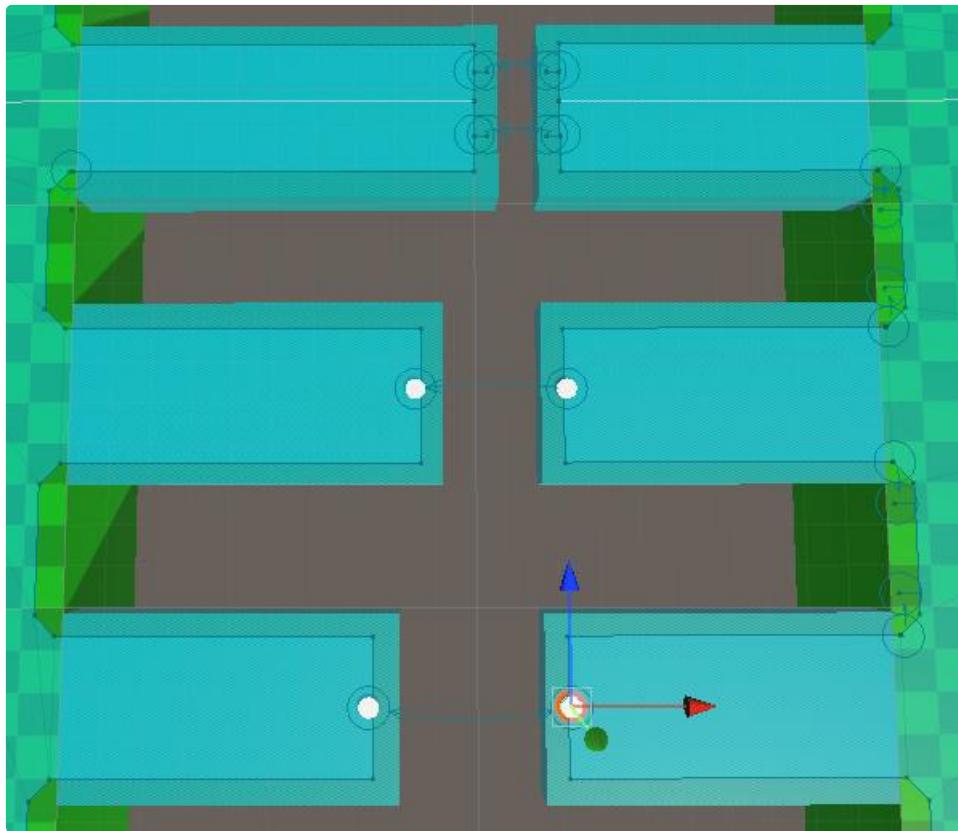


Os OffMesh links traballan en parella. Un deles terá un compoñente OffMeshLink no que se establecen os parámetros do salto na malla, o outro simplemente contén o transform da súa situación na escea.

Os parámetros de cada par son:

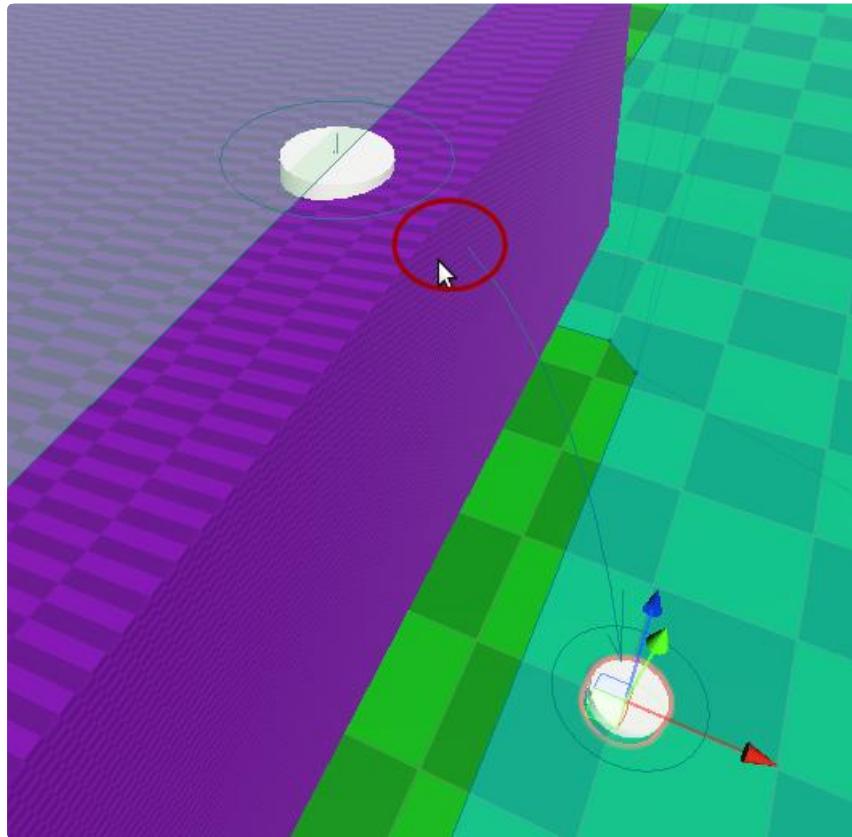
- Os puntos de inicio e de fin
- O custo á hora de calcular a ruta. Por defecto -1, que é o da área a que pertence
- A bidireccionalidade true/false
- Activate true/false
- Actualización se movemos o obxecto
- Un selector de asignacion de Navigation area



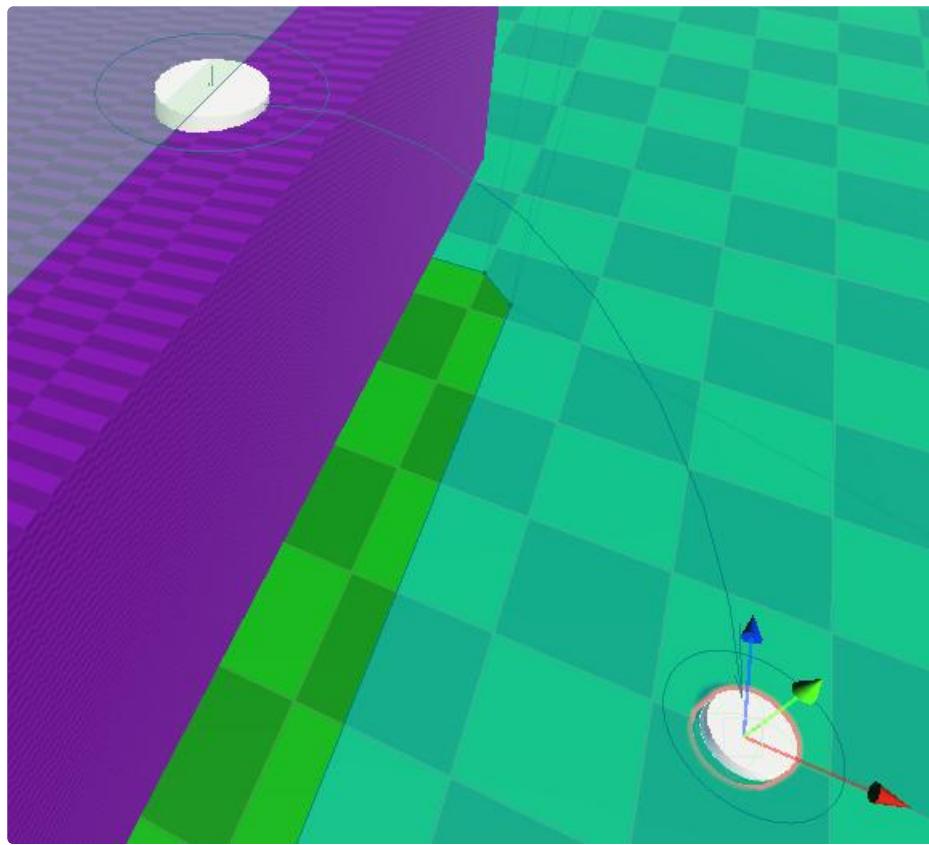


OffMeshLinks bidireccionais

Se o enlace atravesa obxectos tamén o farán os axentes



O punto End está a escasa distancia do punto Start para este salto. O axente atravesará o material



Listo!

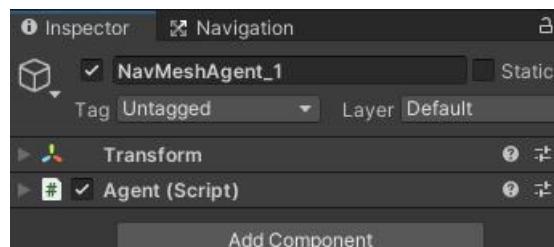
OffMeshLink unidireccional

## 2 - Nav Mesh Agent 1

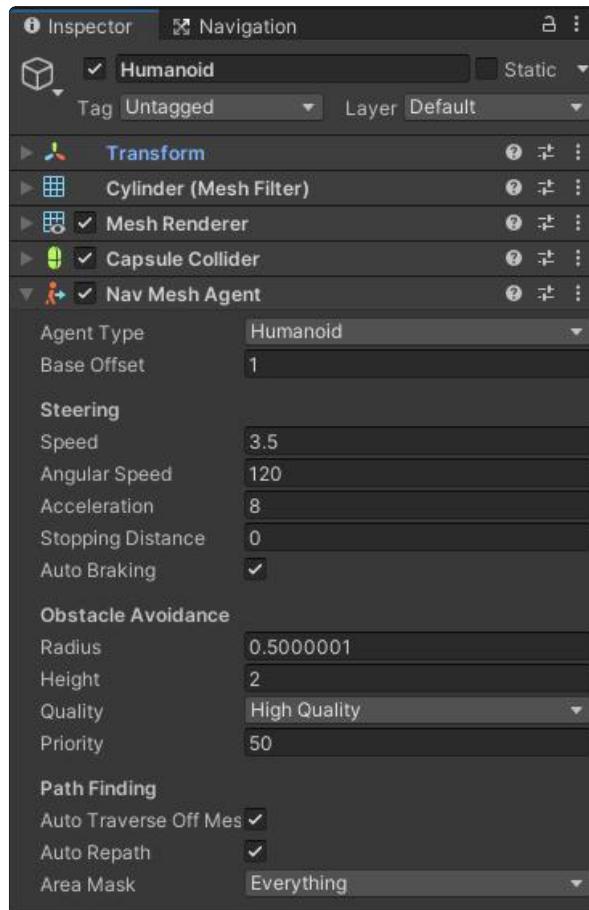
Referencia: [Creating a NavMesh Agent](#)

### O GameObject

Agrégase un Empty Object á escena (Non static) que levará o compoñente script



Créase o Agent 1 con un cilindro e un cubo, e será este cilindro quen leve o compoñente **Nav Mesh Agent**

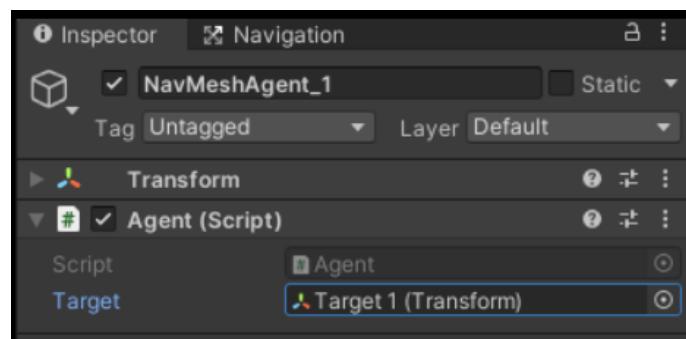


Non é que sexa unha configuración moi complicada, pero vai ser a de tódolos axentes, así que a gardamos coma prefab



### Agent.cs (script)

O proxecto base xa trae un script para asignar aos axentes. Ten unha variable pública para asignar o obxectivo de cada axente.

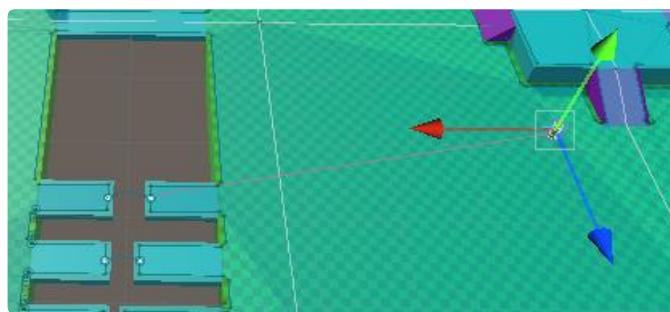


Debo modificalo dada a xerarquía na escea arriba detallada (na que o cilindro humanoide ten o compñénte NavMeshAgent pero o script asígnaselle ao seu parent)

Agent.cs

```
void Start()
{
    // O primeiro fillo ten un componente NavMeshAgent
    agent = GetComponentInChildren<NavMeshAgent>();
    agent.destination = target.position;
}
```

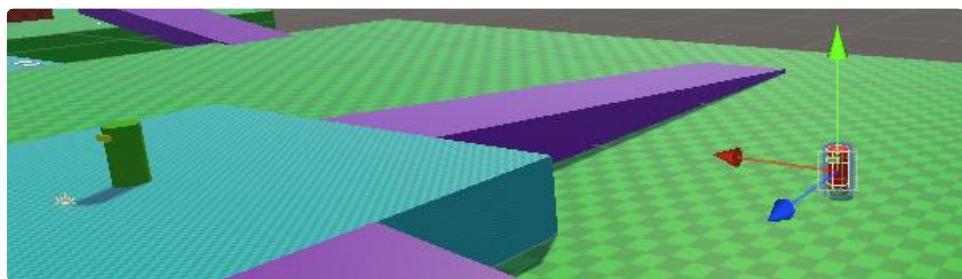
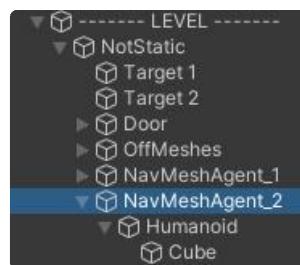
Funciona. O Axente vai seguindo a ruta calcula cara o seu target, Un RaycastDraw vai mostrando un raio vermello na línea que vai seguir



Listo!

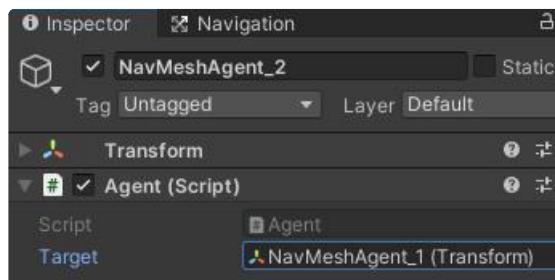
### 3 - O Axente 2 persegue ao Axente 1

Arrastramos o prefab ao grupo NotStatic, situámolo na escea e mudámoslle a cor



2 axentes na escea

Na public Transform Target do seu propio script, asígnaselle coma obxectivo o Axente 1

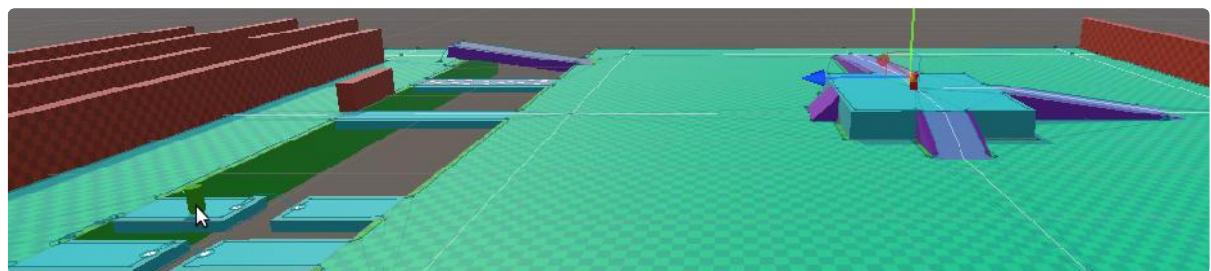


Axente 2 con Target ao Axente 1

## Error

### Pero non funciona.

O Axente 2 entende coma Target a posición inicia do Axente 1 e, ao chegar ahí, para. Nen sequera pinta a RaycastDraw ao chegares a destino:

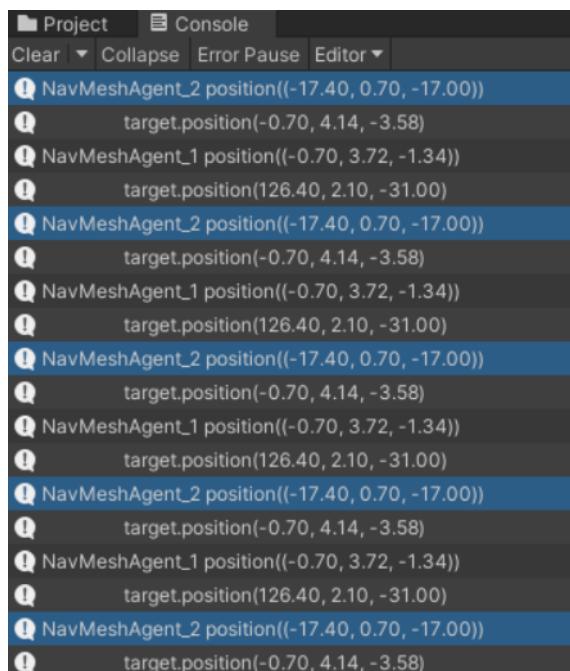


## Debug Console

### Agent.cs

```
void Update()
{
    agent.destination = target.position;

    Debug.Log($"{gameObject.name} position({transform.position})");
    Debug.Log($"    target.position{target.position}");
}
```



Obsérvese que as `positions` de ambos axentes, que ben se ven movéndose na escea, imprimíense fixas en cada frame, `Updates()`.

Vale, entendo que esto é debido a que, seguindo a lóxica de Trojan, creei os axentes coma Empty Objects que anidan o cilindro humanoide que fai as veces de axente. O cilindro Humanoid é o que ten o compoñente NavMeshAgent pero o seu pai é o que ten o script Agent.cs

Na miña implementación, a valiosa pasantía de Trojan resúltame apropiada en parte, a saber, sí que considero máis correcto que un GameObject NavMeshAgent anide o seu cilindro axente, pois o GameObject parent pode ter outras funcionalidades alleas ao NavMesh.

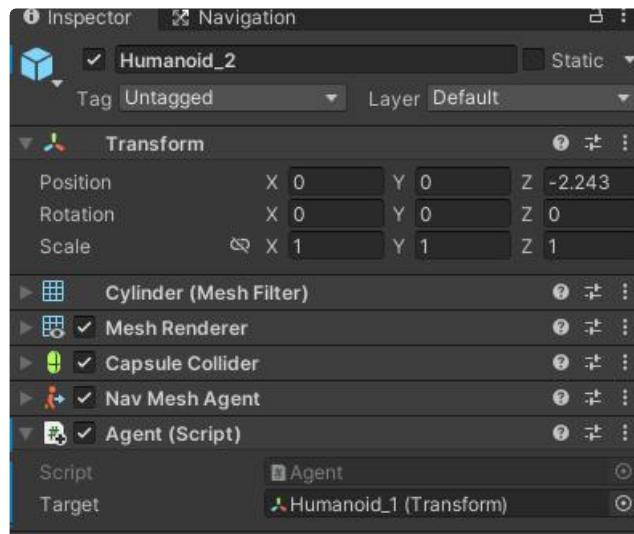
Paréceme a forma correcta de programar GameObjects dinámicos.

O meu fallo xurde de aplicarlle o script Agent.cs ao parent no lugar do cilindro Humanoid dado que, polo que se ve por Console, as propiedades de `using UnityEngine.AI`, aplícanse a elementos NavMesh

## Corrección

Esto amáñase fácilmente:

1. Aplicar coma target ao Humanoid anidado que ten o compoñente NavMeshAgent
2. Aplicar o script ao compoñente que ten o NavMesh dado que Agent.cs é exclusivo para NavMeshAgent's



Así que, despois de experimentar, rematamos aplicando o exemplo de Unity, a saber, agregando tanto o NavMeshAgent coma o script ao cilindro.

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

public class Agent : MonoBehaviour
{
    public Transform target;
    NavMeshAgent agent;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        agent.destination = target.position;
    }

    void Update()
    {
        if (agent.remainingDistance < 1f)
        {
            agent.isStopped = true;
        }
        else
        {
            agent.isStopped = false;
        }
    }
}
```

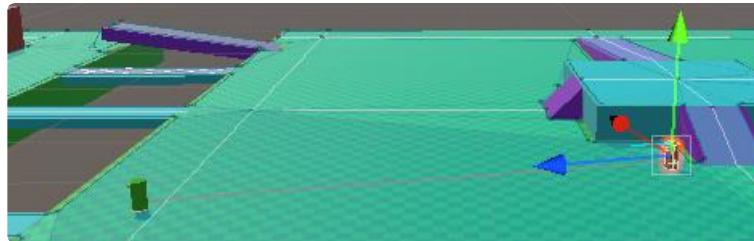
```

{
    agent = GetComponent<NavMeshAgent>();
    agent.destination = target.position;
}

void Update()
{
    agent.destination = target.position;

    Debug.Log($"{gameObject.name} position({transform.position})");
    Debug.Log($"\\t target.position{target.position}");
}

```



Listo!

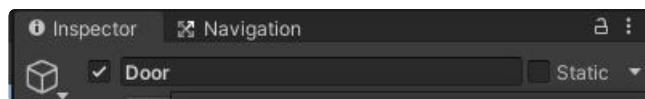
## 4 - NavMesh Obstacle en movimiento

Referencia: [Creating a NavMesh Obstacle](#)

No proxecto base xa hai un GameObject Door con un script con movimento SmoothStep con PingPong para o vector Z.

Hai que configurarlo para que o NavMesh o entenda coma Obstacle cando crea a malla, para eso

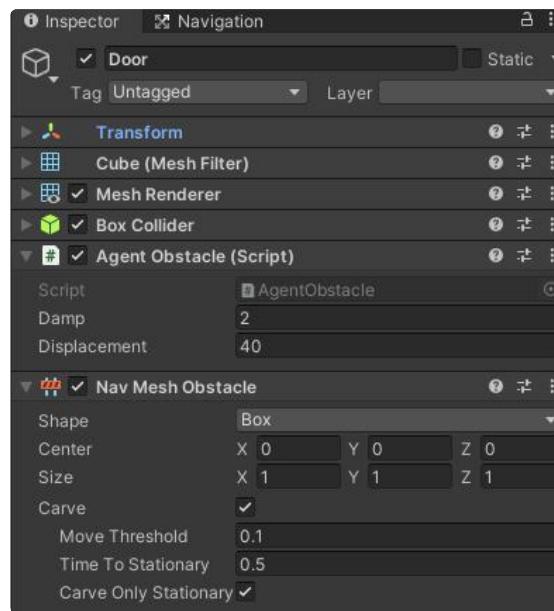
### 1.- O GameObject non pode ser estático



### 2.- Hai que agregarlle un **compoñente Nav Mesh Obstacle**,

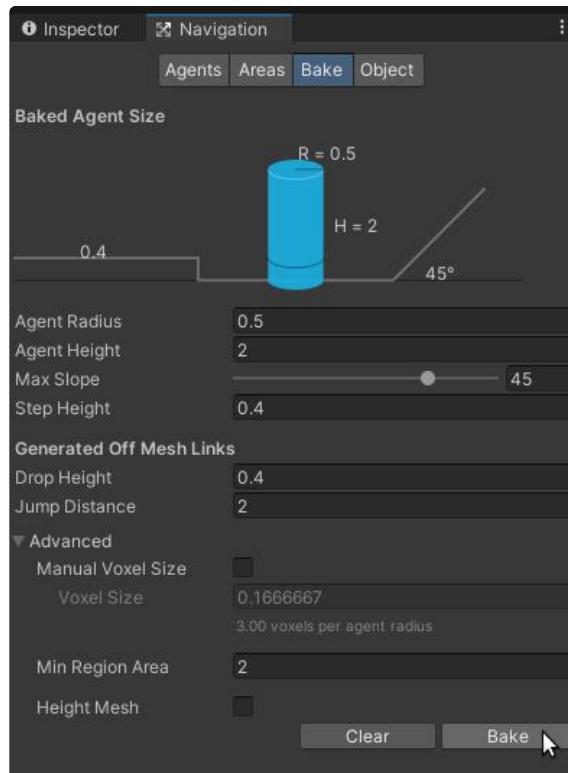
Deixamos o GameObjet Door parent soamente co seu transform na escea (polas mesmas razóns arriba detalladas) e agregamos tando co compoñente NavMeshObstacle coma o script AgentObstacle á Door filla



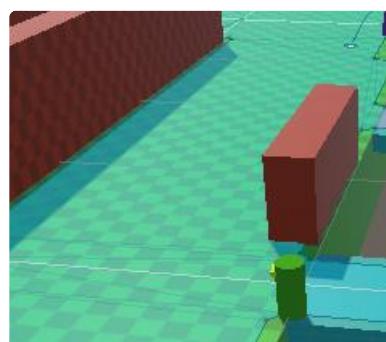


### 3.- Actualizar a nova malla Nav Mesh

Navigation > Bake > Bake



Bake Update



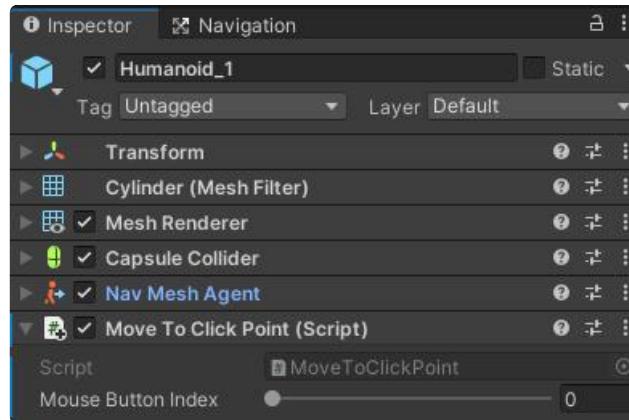
Listo!

## 5 - Target OnClick

### Método 2

Referencia: Moving an Agent to a Position Clicked by the Mouse

Sustituímos no Humanoid do NavMeshAgent\_1 o script Agent por este:



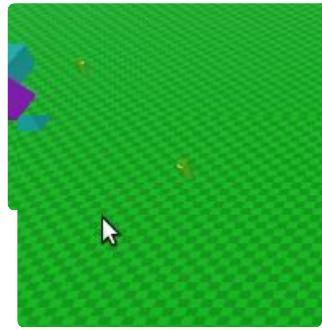
```
using UnityEngine;
using UnityEngine.AI;

public class MoveToClickPoint : MonoBehaviour
{
    [Range(0, 1)]
    public int mouseButtonIndex = 0;
    NavMeshAgent agent;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(mouseButtonIndex))
        {
            RaycastHit hitData;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hitData, 1000))
            {
                agent.destination = hitData.point;
            }
        }
    }
}
```

Funciona perfectamente, mareamos ao pobre NavMeshAgent\_1 a clicks de rato:



## Método 1

Tal cual están configuradas as 2 pelotas Target\_1 e Target\_2. Estas xa teñen agregado o seguinte script Target.cs

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

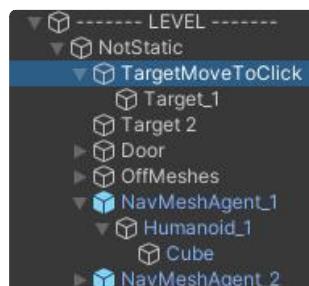
public class Target : MonoBehaviour
{
    [Range(0, 1)]
    public int mouseButtonIndex = 0;

    void Start() { }

    void Update()
    {
        if (Input.GetMouseButtonDown(mouseButtonIndex))
        {
            RaycastHit hitData;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hitData, 1000))
            {
                transform.position = hitData.point;
            }
        }
    }
}
```

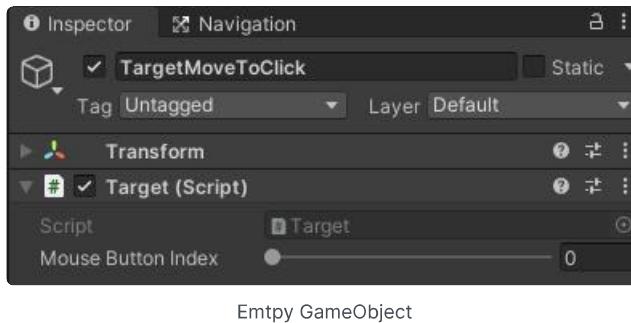
Moi simiar ao anterior pero para aplicar a un GameObject para que faga as veces de Goal

Simplemente asignando este script a calquer GameObject da escea i éste coma coma destino de un axente



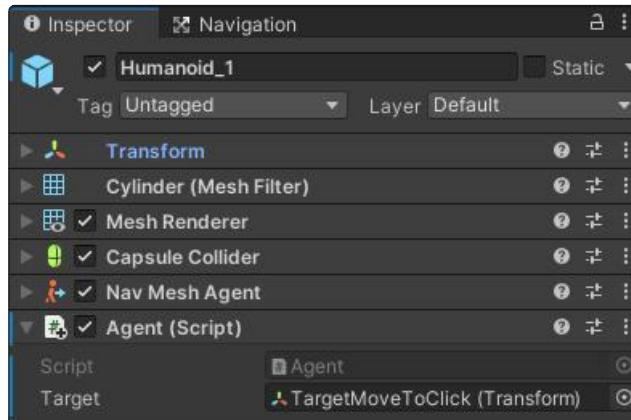
Se a TargetMoveToClick muda de transform.position a click de rato no mapa, tamén muda o transform.position de agent.destination.

Paréceme más ordenado este último, responde a click de rato pero implementando a funcionalidade en un GameObject independiente, que ben podería ser diminuto ou invisible (simplemente un transform)

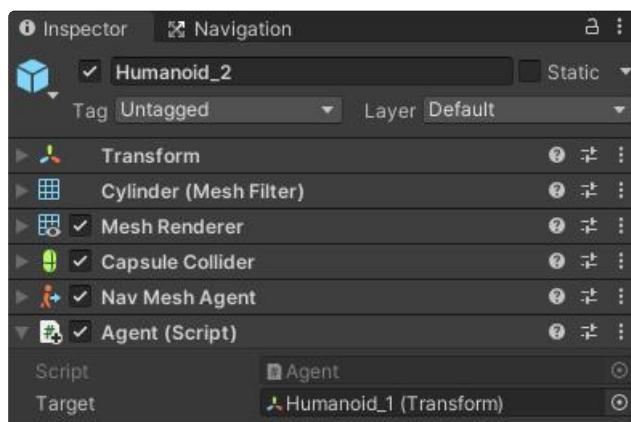


Empty GameObject

mantendo o script para os axentes independientes de esa funcionalidade. Poden poñer coma destination a un GameObject que se move a click de rato ou outro que estea quieto, ou outro móbil coma fixemos no punto 3. Sempre o mesmo script



O Axente 2 ten coma obxectivo un GameObject que se pode mover no mapa a click do xogador



O Axente 2 ten coma obxectivo o Axente 1 que se move tras o Axente 1

Listo!

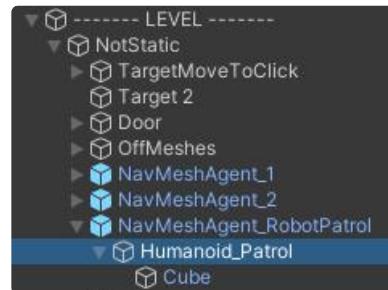
## 6 - Axente patrullador

Referencia: [Making an Agent Patrol Between a Set of Points](#)

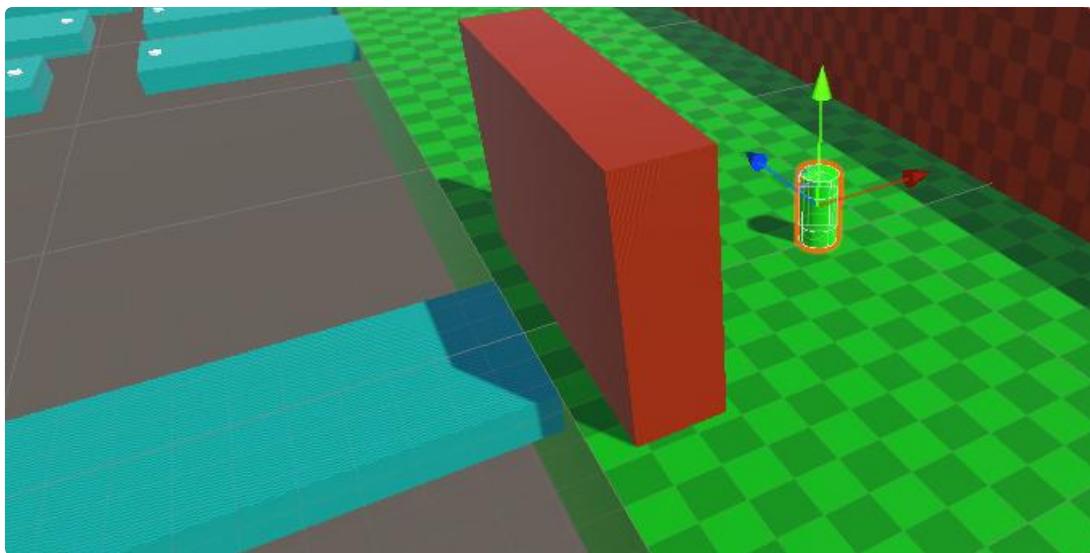
Un terceiro axente que estea patrullando entre, polo menos, 3 puntos.

## New NavMeshAgent

1.- Arrastamos á xerarquía, no grupo NotStatic, un prefab NavMeshAgent e poñémoslle unha cor de tono verde para camuflar pero que destaque (lime)



Situámolo na escea, p.ex., tras a NavMeshObstacle Door



## Agent Patrol (script)

2.- A este Humanoid\_Patrol mudámoslle o Agent.cs por Patrol.cs

```
using UnityEngine;
using UnityEngine.AI;
using System.Collections;

public class Patrol : MonoBehaviour
{
    public Transform[] points;
    private int destPoint = 0;
    private NavMeshAgent agent;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();

        // Disabling auto-braking allows for continuous movement
        // between points (ie, the agent doesn't slow down as it
        // approaches a destination point).
        agent.autoBraking = false;

        GotoNextPoint();
    }
}
```

```

void Update()
{
    // Choose the next destination point when the agent gets
    // close to the current one.
    if (!agent.pathPending && agent.remainingDistance < 0.5f)
        GotoNextPoint();
}

void GotoNextPoint()
{
    // Returns if no points have been set up
    if (points.Length == 0)
        return;

    // Set the agent to go to the currently selected destination.
    agent.destination = points[destPoint].position;

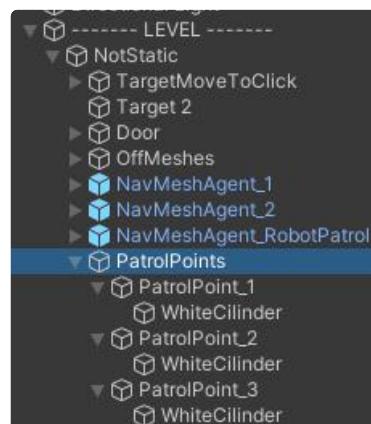
    // Choose the next point in the array as the destination,
    // cycling to the start if necessary.
    destPoint = (destPoint + 1) % points.Length;
}
}
}

```

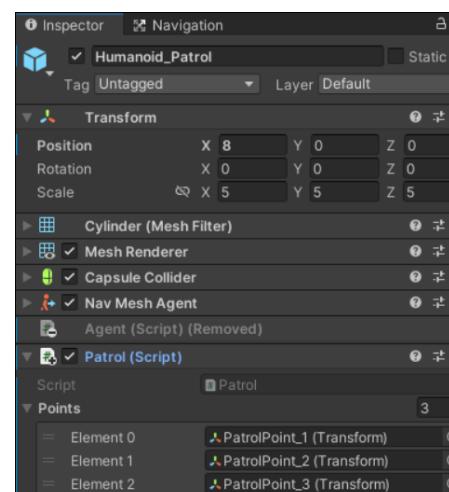
## Set of Patrol Points

3.- Creamos os 3 puntos mínimos de patrulla

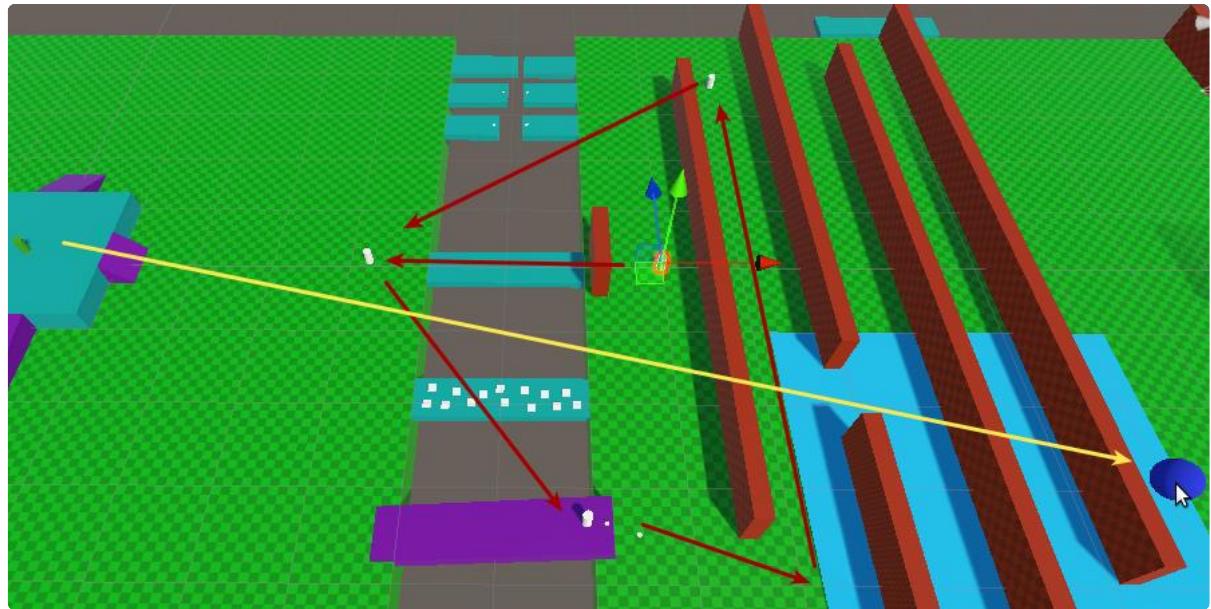
Dev: Anídanse sendos cilindros brancos en trigger para que se vexan no escenario



4.- Asígnaselle ao script os 3 puntos de patrulla



Estes puntos pódense ampliar ao gusto, modificar a velocidade dos NPCs, así coma desatar algunha *action* cando o AgentPatrol alcance ao NavMeshAgent\_1, pero o exercicio funciona:



Listo!