

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LA LAGUNA



DOCUMENTACIÓN

UNIDAD 01: PROYECTO

DOCENTE: LAMIA HAMDAN M.

NUM DE CONTROL	NOMBRE
19130984	Ángel Dario Vidaña Vargas
19130894	Eder Fernando Campa Saucedo
19130932	Carlos Daniel López Romo
19130959	Fernando Pérez Romero
20130046	Francisco Torres Hernández

FECHA DE ENTREGA: 29/09/2023

¿Que es el algoritmo A*?	3
Descripción	4
Características Principales:	4
Elementos del Paquete:	5
Interacción Principal:	5
Componente AstarPath:	5
Scripts de Movimiento:	6
Desarrollo de Aplicación Adicional:	6
Funcionalidades Adicionales:	6
Implementación Técnica:	6
Clase AstarPath:	7
Movement Scripts:	7
AIPath:	7
RichAI:	8
AILerp:	8
Clase llamada AIDestinationSetter:	9
Uso de Modificadores:	9
Clase Seeker:	10
Clase AILerp:	10
Variables Públicas:	11
Métodos:	11
Variables:	12
Características Obsoletas:	12
Clase AIPath:	12
Variables publicas:	13
Métodos Públicos:	14
Métodos Privados/Protegidos:	14
Clase RichAI:	15
Métodos Públicos:	15
Variables Públicas:	15
Miembros Heredados Públicos:	16
Escenas	17
Aleatorios:	17
Main Camera:	18
Clase Generador:	18
Objetos en el mapa (picos, hielo, monstruos, fuego, etc.):	21
Grid:	22
Propiedades:	22
Suelo, Obstaculos y Obstaculos Suaves:	23
Propiedades:	24
Jugador, Monedas, Picos:	26
Propiedades:	29
Objetivo:	31
Conclusion	33
Referencias	34

¿Que es el algoritmo A*?

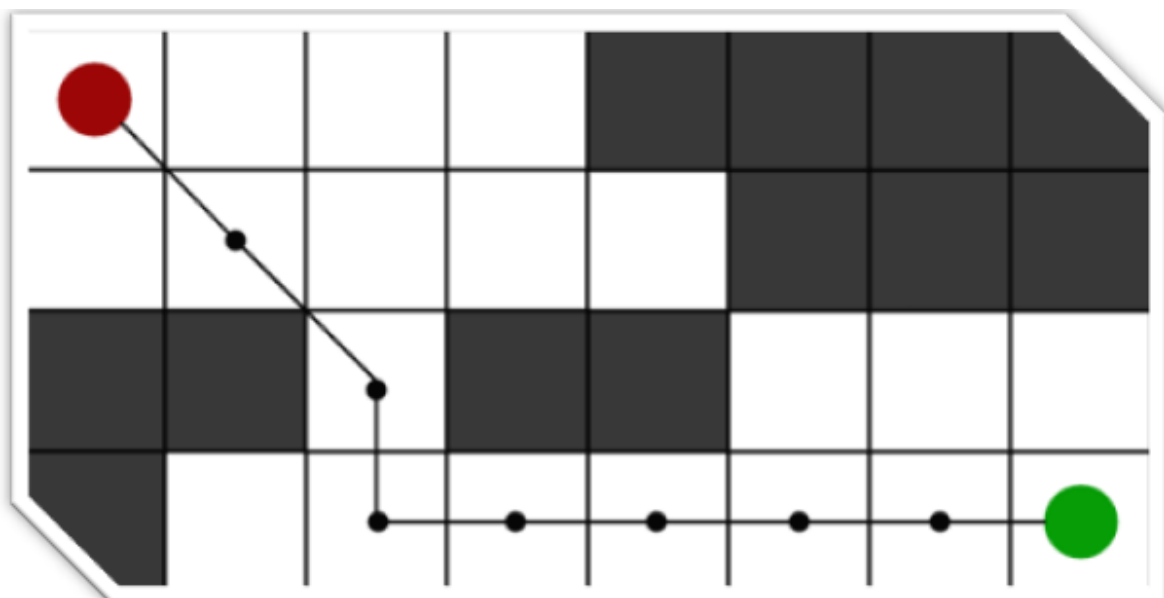
Algoritmo A*: El algoritmo A* Es un método eficaz y ampliamente utilizado en el campo de la inteligencia artificial y la informática, específicamente en la resolución de problemas de búsqueda de rutas óptimas en grafos o espacios de estado. Este algoritmo se basa en la combinación de la búsqueda primero el mejor y la búsqueda con costo uniforme.

La principal idea detrás del algoritmo A* es encontrar la ruta óptima desde un punto de inicio hacia un punto de destino en un grafo ponderado, minimizando el costo total que implica llegar desde el inicio al destino.

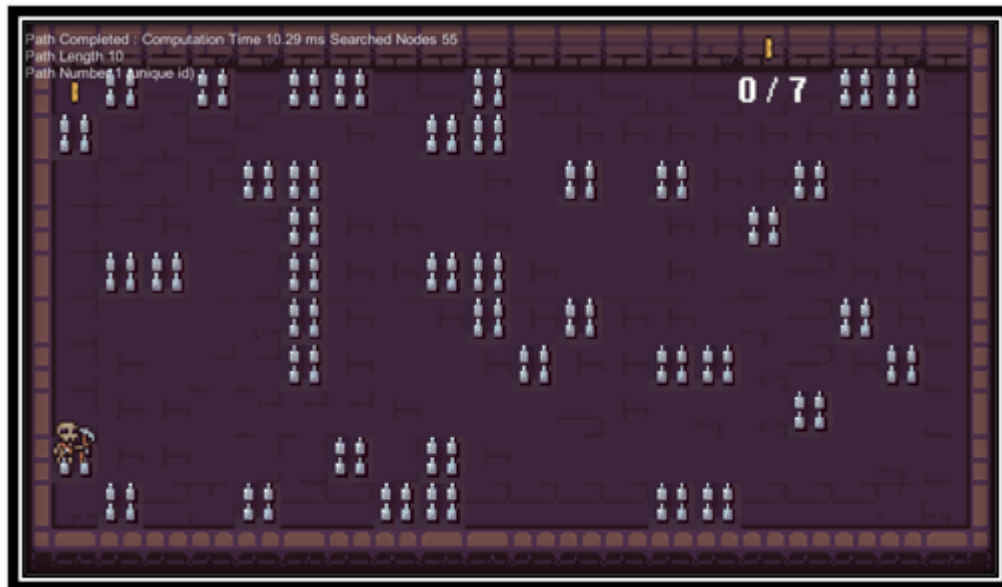
El algoritmo A* utiliza dos heurísticas cruciales para tomar decisiones informadas durante la búsqueda. Primero, utiliza una función de costo $g(n)$ que representa el costo real acumulado desde el nodo de inicio hasta el nodo actual n . Segundo, utiliza una función heurística $h(n)$ que estima el costo desde el nodo actual n hasta el nodo de destino. La función de costo total $f(n)$ se define como la suma de $g(n)$ y $h(n)$: $f(n) = g(n) + h(n)$.

Durante la ejecución del algoritmo, se exploran los nodos del grafo siguiendo una estrategia de expansión preferencial, priorizando aquellos nodos con un valor de $f(n)$ más bajo. En cada iteración, el algoritmo selecciona el nodo con el menor valor de $f(n)$ y explora sus nodos vecinos. Luego, calcula y actualiza los valores de $g(n)$ y $f(n)$ para cada nodo vecino, eligiendo la mejor ruta hasta el momento.

El algoritmo A* garantiza que la solución encontrada es óptima siempre que la heurística utilizada ($h(n)$) sea admisible, es decir, que nunca sobreestima el costo real hasta el destino. Si la heurística es admisible y consistente, A* encontrará la ruta más corta desde el nodo de inicio hasta el nodo de destino.



Descripción



El proyecto es un juego desarrollado en Unity en el que un pequeño personaje debe recoger una moneda que aparece de forma aleatoria en un mapa. El personaje utiliza el algoritmo A* para encontrar la ruta óptima hacia la moneda.

Características Principales:

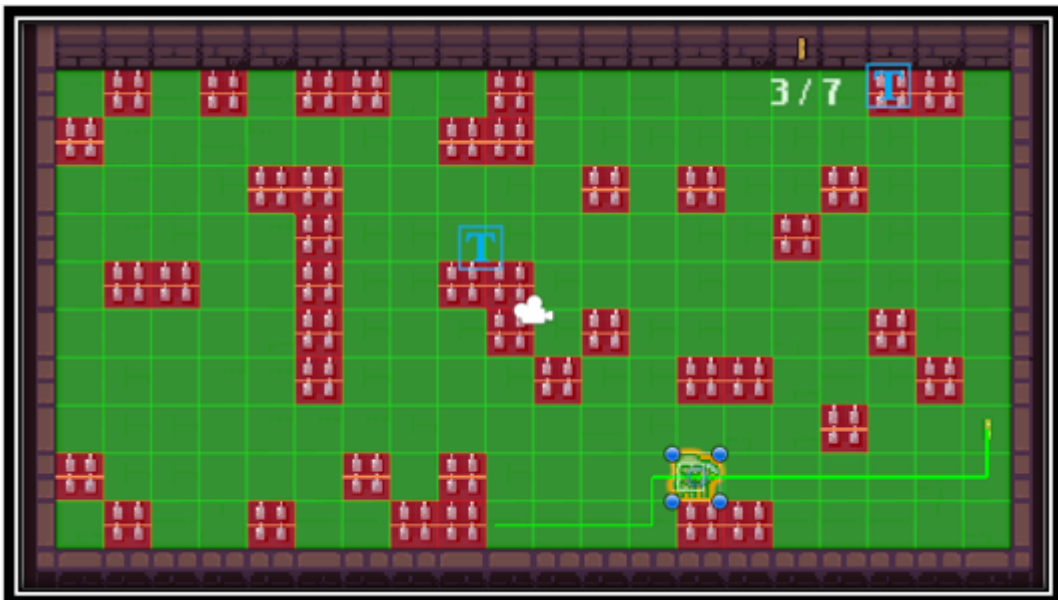
- 1.- Personaje: Un personaje que se mueve por el mapa automáticamente.
- 2.- Moneda: Una moneda que aparece en una posición aleatoria del mapa.
- 3.- Mapa: Un entorno en el que el personaje se desplaza y busca la moneda.



Algoritmo A*: Utilizado por el personaje para determinar la ruta óptima hacia la moneda

Elementos del Paquete:

- 1.-Guiones de Movimiento: Indican al agente cómo moverse y hacia dónde debe moverse.
- 2.-Gráficos: Describen dónde puede moverse un agente.
- 3.-Obstáculos Temporales: Cortan agujeros en la malla de navegación o la actualizan de otras maneras.
- 4.-Enlaces fuera de la malla: Permiten a un agente moverse o saltar entre partes de la malla de navegación.
- 5.-Modificadores de Ruta: Post Procesan rutas para suavizarlas u otros ajustes.



Interacción Principal:

El jugador interactúa principalmente con el guión de movimiento y el componente Buscador.

El guión de movimiento controla el movimiento del agente, su velocidad, rotación, etc., así como también su destino y cuándo recalcular la ruta.

El componente Buscador es controlado por el guión de movimiento y calcula la ruta utilizando el algoritmo A*.

Componente AstarPath:

Contiene todos los datos del gráfico en una escena permitiendo un solo componente de este tipo.

Puede contener uno o varios gráficos, del mismo o diferente tipo, y cada gráfico administra sus nodos.

Scripts de Movimiento

Hay varios scripts de movimiento incluidos en el paquete que controlan el movimiento del agente.

Desarrollo de Aplicación Adicional

Esqueleto (jugador) y Monedas

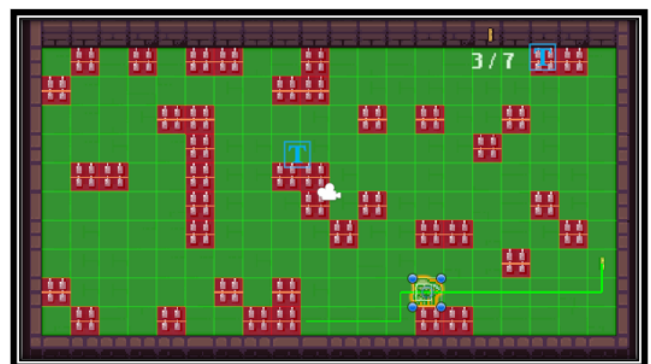
Se implementa una aplicación adicional que involucra un Esqueleto (jugador) y monedas en un mundo discreto con obstáculos y diferentes tipos de terreno.

El mundo tiene un objetivo que el agente debe alcanzar.

Este agente es inteligente y realiza cálculos en cada paso para tomar decisiones correctas y llegar al objetivo.

Funcionalidades Adicionales:

- 1.-Mundo Discreto: Un entorno con obstáculos y diferentes tipos de terreno que afectan el movimiento del agente.
- 2.-Cambio de Ubicación del Objetivo: Cada vez que se inicia el sistema, el objetivo y, si es posible, el agente cambia de ubicación en el mundo discreto.
- 3.-Utilización de Sentidos: El agente utiliza sentidos simulados para detectar obstáculos y tomar decisiones informadas.



Implementación Técnica:

- 1.-Librerías para Gráficos: Se utilizan librerías para graficar que permiten representar visualmente el mundo discreto y la posición de los elementos.

Algoritmo A*: Se implementa el algoritmo A* con heurística y costo para permitir que el agente encuentre la ruta óptima hacia el objetivo en el mundo discreto.

Estructura del Proyecto (Clases, Métodos, Librerías, Etc)

Clase AstarPath

La clase AstarPath es el componente principal del sistema de búsqueda de rutas A*. Maneja todo el sistema de búsqueda de rutas, calcula todas las rutas y almacena la información. Esta clase es una clase singleton, lo que significa que debería existir como máximo una instancia activa de ella en la escena. Usualmente, la interacción con el sistema de búsqueda de rutas se realiza a través de la clase Seeker.

Esta clase es el padre de todas sus subclases lo que hace que recopile su información y así calcule el algoritmo estrella.

Movement Scripts:

Estos scripts facilitan el movimiento de objetos dentro de la escena, que a menudo representan personajes o entidades. Su función principal es manejar la búsqueda de rutas y guiar al objeto a lo largo de la ruta especificada.

Dentro de este paquete, los scripts de movimiento son opcionales. Se puede utilizar la búsqueda de rutas sin estos scripts, y los desarrolladores tienen la flexibilidad de crear sus propios scripts de movimiento personalizados. Sin embargo, estos scripts predefinidos ofrecen una sólida base para el comportamiento del personaje en muchos juegos. Pueden ser utilizados como punto de partida y, más adelante en el proceso de desarrollo, personalizados o reemplazados con scripts adaptados al juego específico.

En el paquete se incluyen tres scripts principales de movimiento: AIPath (El utilizado), RichAI y AILerp. A pesar de algunas consideraciones en cuanto a la nomenclatura, estos scripts tienen propósitos distintos y se adaptan a diferentes tipos de gráficos y estilos de movimiento.

AIPath

- Un script de movimiento versátil compatible con varios tipos de gráficos.
- Sigue las rutas de forma suave y responde a la física.
- Adecuado para juegos en 3D y 2D.
- Compatible con la evasión local.

RichAI

- Diseñado específicamente para gráficos de navmesh/recast y no es compatible con otros tipos de gráficos.
- Excelente para seguir rutas en gráficos basados en navmesh, puede lidiar mejor con las desviaciones del camino y generalmente sigue el camino de forma más suave.
- Ofrece un sólido soporte para enlaces fuera de la malla en comparación con AIPath.
- Compatible con la evasión local.
- Ideal para juegos en 3D (movimiento en el plano XZ), pero no está diseñado para juegos en 2D.

AILerp

- Utiliza interpolación lineal ('lerp') para moverse a lo largo de la ruta (de ahí 'lerp', que significa interpolación lineal), sin utilizar física de ninguna manera.
- Sigue la ruta exactamente, sin desviaciones.
- Debido a los puntos anteriores, no tiene sentido usarlo con evasión local y, por lo tanto, no la admite.
- Es el script de movimiento más rápido, ya que el movimiento en sí es mucho más simple. Sin embargo, ten en cuenta que si necesitas algún tipo de realismo físico en el juego, generalmente deberías usar uno de los otros scripts de movimiento.

Todos los scripts de movimiento incluidos en el paquete implementan la interfaz Pathfinding.IAStarAI. Esta interfaz permite una interacción fluida con varios scripts de movimiento, lo que la convierte en una herramienta valiosa para los desarrolladores.

Por ejemplo, algunas propiedades que pueden resultar útiles (estas propiedades existen en todos los scripts de movimiento) son:

- destination: La posición en el mundo a la que este agente debe moverse.
- reachedDestination: Verdadero si el agente ha alcanzado la destino.
- velocity: La velocidad real a la que se está moviendo el agente.
- desiredVelocity: La velocidad que este agente desea tener.

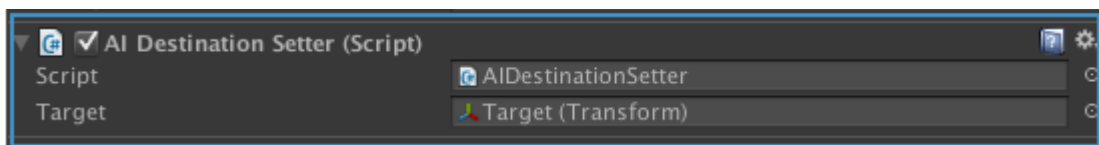
Se utilizó y modificó el guión de movimiento AIPath ya que solo se requiere recorrer un mundo simple en 2d.

Clase llamada AIDestinationSetter:

Esta clase, llamada AIDestinationSetter, se encarga de establecer la posición de destino para un agente de inteligencia artificial (AI). A continuación, describo las partes clave de la clase y lo que hacen.

Este objeto obtiene la información y propiedades del objetivo (Target) con el que se establecerá la ruta (Origen y destino) la cual será utilizada por el guión de movimiento.

- Target: Es la variable que representa el objeto al cual el AI debe moverse.
- ai: Es de tipo IAStarAI y parece ser una referencia al componente que maneja la lógica de movimiento del AI.



Uso de Modificadores

Los modificadores son pequeños scripts que se pueden conectar al Seeker y pre o post procesar la ruta antes de devolverla al llamante.

Para usar modificadores, simplemente adjunta los scripts al mismo GameObject al que se adjunta el componente Seeker. Estos funcionarán automáticamente sin necesidad de modificar código.

Los scripts de movimiento AIPath y AILerp utilizan modificadores.

Modificador de Suavizado Simple

- El modificador de suavizado simple es un modificador para suavizar la ruta. Puede hacerlo subdividiéndola y acercando los vértices entre sí o utilizando splines, en particular, la curva de Bezier.

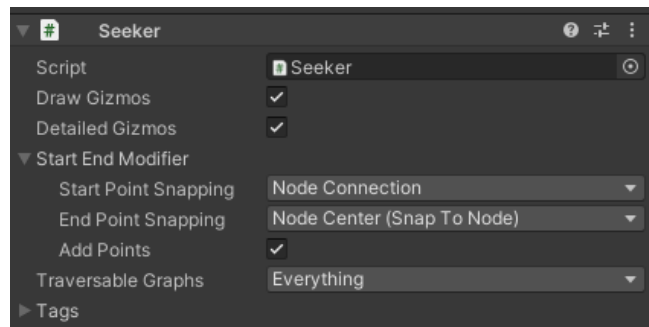
Modificador de Embudo

- El modificador de embudo es un modificador para simplificar rutas en mallas de navegación o gráficos de cuadrícula de una manera rápida y precisa.
- Como su nombre indica, aplica el algoritmo de embudo a la ruta. Esto devolverá la ruta más corta dentro del corredor de la ruta que calculó el buscador de caminos.

En este proyecto se utilizó un modificador para que no fuera posible hacer rutas en diagonal.

Clase Seeker

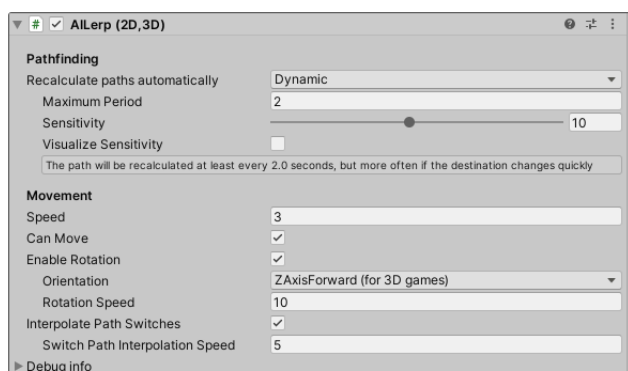
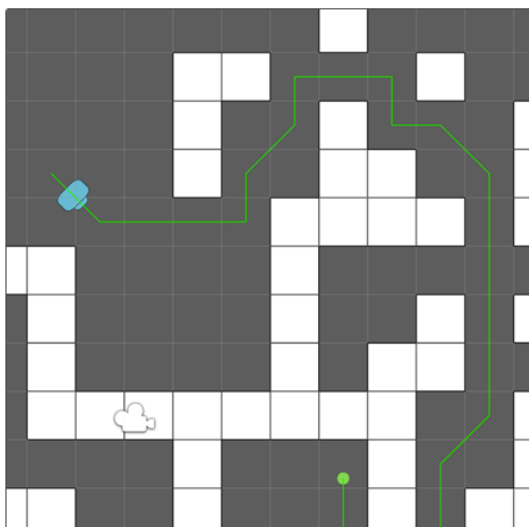
La clase Seeker maneja las llamadas de ruta para una sola unidad. Este componente está diseñado para adjuntarse a una sola unidad (IA, robot, jugador, lo que sea) para manejar sus llamadas de búsqueda de ruta. También se encarga del post procesamiento de rutas utilizando modificadore



Este script es el encargado de recopilar la información y propiedades del objeto origen (Jugador) y de qué manera se moverá este.

Clase AILerp

La clase AILerp es un script de movimiento con interpolación lineal diseñado para facilitar un movimiento suave y preciso para agentes que siguen un camino especificado. Este script permite que un agente atravesase puntos de referencia en un camino utilizando interpolación lineal. Es particularmente adecuado para ciertos tipos de juegos y puede funcionar tanto en entornos 2D como en 3D.



Variables Públicas

- **autoRepath:** Especifica cómo el agente recalcula automáticamente su camino.
- **canMove:** Habilita o deshabilita completamente el movimiento.
- **destinationPublic:** Propiedad pública que representa la posición de destino.
- **enableRotation:** Determina si el AI debe rotar para enfrentar la dirección del movimiento.
- **hasPath:** Indica si este agente sigue actualmente un camino.
- **interpolatePathSwitches:** Controla la interpolación cuando se calcula un nuevo camino.
- **isStopped:** Obtiene o establece si el agente debe detenerse.
- **onSearchPath:** Se desencadena cuando el agente recalcula su camino.
- **orientation:** Especifica la dirección en la que se mueve el agente.
- **pathPending:** Indica si actualmente se está calculando un camino.
- **position:** Posición del agente.
- **reachedDestination:** Indica si el AI ha alcanzado el destino.
- **reachedEndOfPath:** Indica si se ha alcanzado el final del camino actual.
- **remainingDistance:** Distancia aproximada restante a lo largo del camino actual hasta el final del camino.
- **rotation:** Rotación del agente.
- **rotationSpeed:** Controla la velocidad de rotación del agente.
- **speed:** Velocidad del agente en unidades del mundo.
- **switchPathInterpolationSpeed:** Velocidad a la que se interpola al nuevo camino.
- **updatePosition:** Determina si la posición del personaje debe estar acoplada a la posición del Transform.
- **updateRotation:** Determina si la rotación del personaje debe estar acoplada a la rotación del Transform.
- **velocity:** Velocidad real a la que se mueve el agente.

Características Privadas/Protegidas

Métodos

- **AILerp():** Constructor para la clase AILerp.
- **Awake():** Inicializa las variables de referencia.
- **CalculateNextPosition(direction, deltaTime):** Calcula la próxima posición del AI (un fotograma en el futuro).
- **ClearPath():** Borra el camino actual del agente.
- **ConfigureNewPath():** Encuentra el punto más cercano en el camino actual y configura el interpolador.

- `ConfigurePathSwitchInterpolation()`: Configura la interpolación al cambiar de camino.
- `Init()`: Inicializa la instancia de `AILerp`.
- `Move(deltaPosition)`: Mueve al agente en función de la posición delta proporcionada.
- `OnEnable()`: Invocado cuando se habilita el componente.
- `OnPathComplete(_p)`: Llamado cuando se ha completado el cálculo de un camino solicitado.
- `OnUpgradeSerializedData(version, unityThread)`: Maneja la compatibilidad hacia atrás de la serialización.
- `Reset()`: Maneja la compatibilidad hacia atrás de la serialización.
- `SimulateRotationTowards(direction, deltaTime)`: Simula la rotación hacia una dirección dada.
- `Start()`: Comienza la búsqueda de caminos.
- `Update()`: Actualiza el componente `AILerp`.

Variables

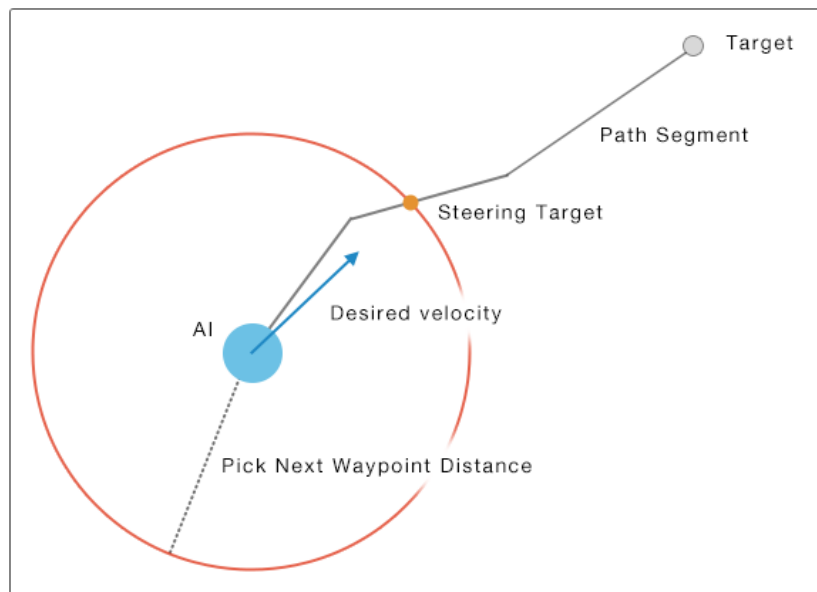
- Variables privadas y protegidas para administrar el movimiento, el camino, la dirección, la serialización, y más.

Características Obsoletas

- `ForceSearchPath()`: Solicita un camino hacia el objetivo.
- `canSearch`: Habilita o deshabilita el recálculo del camino en intervalos regulares.
- `repathRate`: Determina con qué frecuencia el agente busca nuevos caminos.
- `rotationIn2D`: Especifica si el eje hacia adelante del personaje estará a lo largo del eje Y en lugar del eje Z.
- `target`: Objetivo al que moverse.

Clase `AIPath`

La clase `AIPath` representa un script de movimiento predeterminado diseñado para permitir a los agentes seguir caminos en un entorno de juego. Este script es parte del Proyecto de Ruta A* y proporciona una funcionalidad básica de movimiento. Aunque no es esencial para el funcionamiento del sistema, puede facilitar la configuración del movimiento para los personajes en el juego. Es adecuado para muchos tipos de unidades, aunque si se busca el mayor rendimiento (por ejemplo, si se mueven cientos de personajes), puede ser necesario personalizar este script o escribir uno personalizado para optimizarlo específicamente para el juego.



Variables publicas

- RepathRate: Determina con qué frecuencia se buscarán nuevos caminos.
- Destination: Es el punto al que el AI intentará moverse, puede ser un punto en el suelo donde el jugador hizo clic en un RTS, o el objeto del jugador en un juego de zombies.
- MaxSpeed: Controla la velocidad máxima del AI.
- RotationSpeed: Velocidad de rotación en grados por segundo.
- SlowdownDistance: Es la distancia aproximada desde el objetivo donde el AI empezará a reducir la velocidad.
- PickNextWaypointDist: Determina la distancia a la que el AI se moverá hacia el siguiente punto en el camino.
- EndReachedDistance: Distancia al punto final para considerar que se ha alcanzado el final del camino.
- HasPath: Indica si este agente tiene actualmente un camino que sigue.
- ReachedDestination: Indica si el AI ha alcanzado el destino.
- ReachedEndOfPath: Indica si el agente ha llegado al final del camino actual.
- RemainingDistance: Distancia aproximada restante a lo largo del camino actual hasta el final del camino.
- SteeringTarget: Punto en el camino hacia el cual se está moviendo el agente.
- RotationSpeed: Velocidad de rotación en grados por segundo.
- SlowWhenNotFacingTarget: Reduce la velocidad cuando no está mirando en la dirección del objetivo.
- SlowdownDistance: Distancia desde el final del camino donde el AI comenzará a reducir la velocidad.
- WhenCloseToDestination: Qué hacer cuando está dentro de la distancia de "endReachedDistance" desde el destino.

Métodos Públicos

- `GetRemainingPath(buffer, stale)`: Llena un buffer con el camino restante.
- `OnTargetReached()`: Se llama cuando se alcanza el final del camino.
- `Teleport(newPosition, clearPath=true)`: Mueve instantáneamente al agente a una nueva posición.

Métodos Privados/Protegidos

- `ApplyGravity(deltaTime)`: Acelera al agente hacia abajo para simular gravedad.
- `Awake()`: Inicialización de la clase.
- `CalculateDeltaToMoveThisFrame(position, distanceToEndOfPath, deltaTime)`: Calcula cuánto moverse durante un solo frame.
- `CalculateNextRotation(slowdown, nextRotation)`: Calcula la siguiente rotación.
- `CalculatePathRequestEndpoints(start, end)`: Devuelve el punto de inicio y el punto final de la próxima solicitud automática de ruta.
- `CancelCurrentPathRequest()`: Cancela la solicitud de ruta actual.
- `ClampToNavmesh(position, positionChanged)`: Convierte la posición del personaje para que esté en la malla de navegación.
- `ClearPath()`: Borra el camino actual del agente.
- `FixedUpdate()`: Llamado en cada actualización de física.
- `MovementUpdateInternal(deltaTime, nextPosition, nextRotation)`: Llamado durante `Update` o `FixedUpdate`, dependiendo de si se usan `Rigidbody`s para el movimiento o no.
- `OnDisable()`: Llamado cuando el componente está deshabilitado.
- `OnDrawGizmos()`: Llamado para dibujar gizmos.
- `OnDrawGizmosSelected()`: Llamado para dibujar gizmos seleccionados.
- `OnEnable()`: Llamado cuando el componente está habilitado.
- `OnPathComplete(newPath)`: Llamado cuando se ha calculado una ruta solicitada.
- `OnUpgradeSerializedData(version, unityThread)`: Maneja la compatibilidad con versiones anteriores de serialización.
- `RaycastPosition(position, lastElevation)`: Comprueba si el personaje está en el suelo y evita la penetración en el suelo.
- `Reset()`: Maneja la compatibilidad con versiones anteriores de serialización.
- `SimulateRotationTowards(direction, maxDegrees)`: Simula la rotación del agente hacia la dirección especificada y devuelve la nueva rotación.
- `Start()`: Comienza la búsqueda de rutas.
- `Update()`: Llamado en cada frame.
- `UpdateVelocity()`: Actualiza la velocidad.

Clase RichAI:

La clase RichAI es una implementación avanzada de un agente de inteligencia artificial (AI) diseñada para trabajar en entornos con gráficos de navegación basados en navmesh. Esta clase extiende de las clases AIBase e IAStarAI proporcionando funcionalidades de movimiento y navegación especializadas. A continuación, se describen sus componentes y funcionalidades principales.

Métodos Públicos

GetRemainingPath(buffer, stale)

- Este método llena un búfer con el camino restante del agente.

SearchPath()

- Permite recalcular la ruta actual del agente.

Teleport(newPosition, clearPath=true)

- Mueve instantáneamente al agente a una nueva posición especificada.

Variables Públicas

acceleration

- Determina la máxima aceleración del agente.

approachingPartEndpoint

- Indica si el agente se acerca al último punto de la ruta en la parte actual de la ruta.

approachingPathEndpoint

- Indica si el agente se acerca al último punto de todas las partes de la ruta en la ruta actual.

endReachedDistance

- Establece la distancia máxima al punto final para considerarlo alcanzado.

funnelSimplification

- Controla si se debe utilizar la simplificación de embudo en el camino.

hasPath

- Indica si este agente tiene actualmente una ruta que sigue.

onTraverseOffMeshLink

- Es llamado cuando el agente comienza a atravesar un enlace fuera de la malla de navegación.

pathPending

- Indica si actualmente se está calculando una ruta.

reachedDestination

- Indica si el agente ha alcanzado su destino.

reachedEndOfPath

- Indica si el agente ha alcanzado el final de la ruta actual.

remainingDistance

- Proporciona una estimación de la distancia restante a lo largo de la ruta actual hasta el final de la misma.

repeatedlySearchPaths

- Controla si se deben buscar rutas de manera repetida.

rotationSpeed

- Establece la velocidad máxima de rotación del agente.

slowWhenNotFacingTarget

- Permite ralentizar al agente cuando no está orientado hacia la dirección del objetivo.

slowdownTime

- Determina cuánto tiempo antes de llegar al final de la ruta se debe empezar a reducir la velocidad.

steeringTarget

- Representa el punto en la ruta hacia el cual el agente se está moviendo actualmente.

traversingOffMeshLink

wallDist

- Determina la distancia a la que las paredes se utilizarán para evitar colisiones.

wallForce

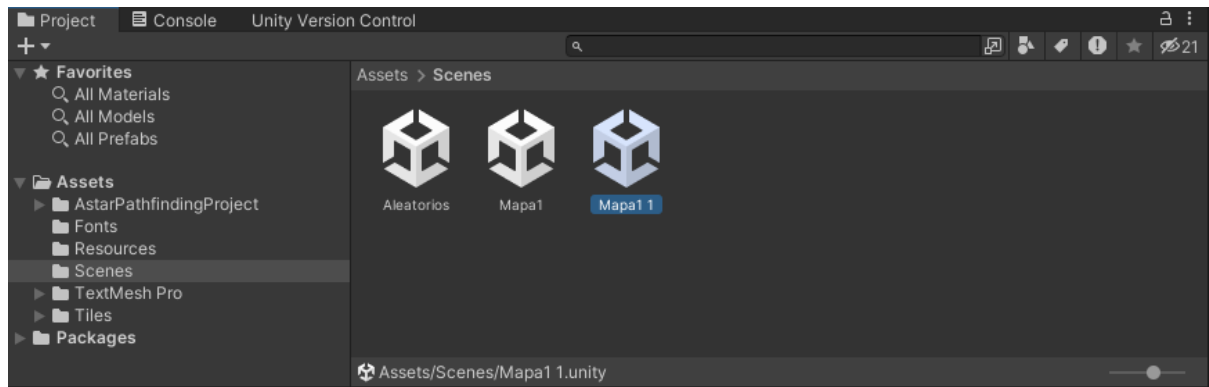
- Establece la fuerza para evitar colisiones con las paredes.

Miembros Heredados Públicos

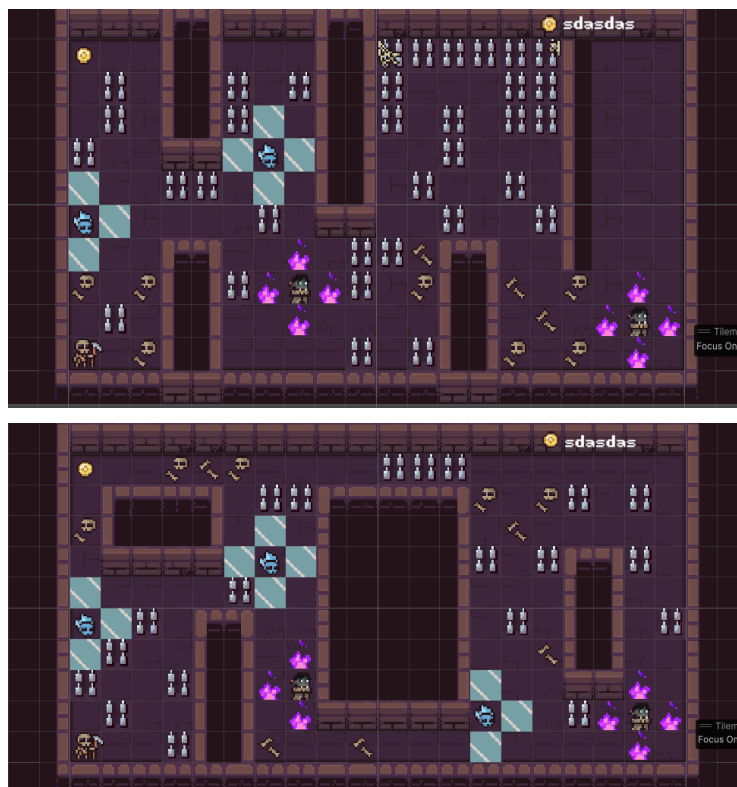
La clase RichAI hereda miembros de las clases AIBase e IAStarAI. Algunos de estos miembros son:

- FinalizeMovement: Mueve al agente a una posición específica.
- FindComponents: Busca y recupera componentes adjuntos necesarios para el movimiento.
- Move: Permite mover al agente.
- SetPath: Hace que el agente siga una ruta específica.
- Y otros métodos y variables que controlan la posición, rotación, velocidad, y otros aspectos del movimiento del agente.

Escenas

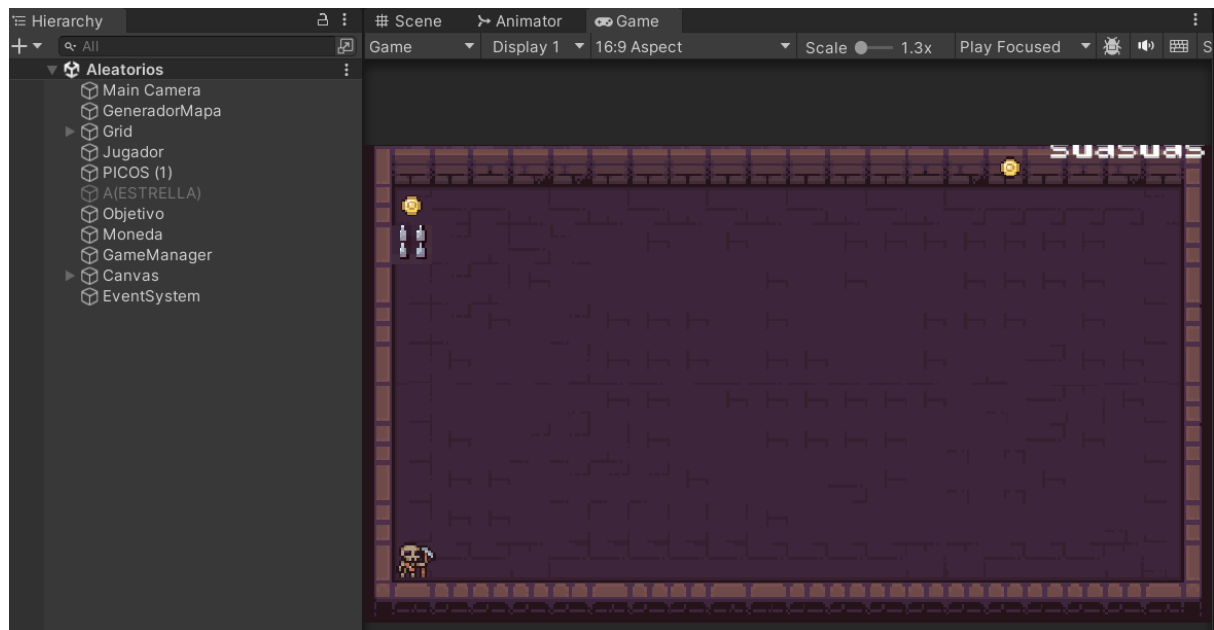


Las escenas “Mapa1” y “Mapa1 1” son mapas ya establecidos con multiples objetos para pruebas



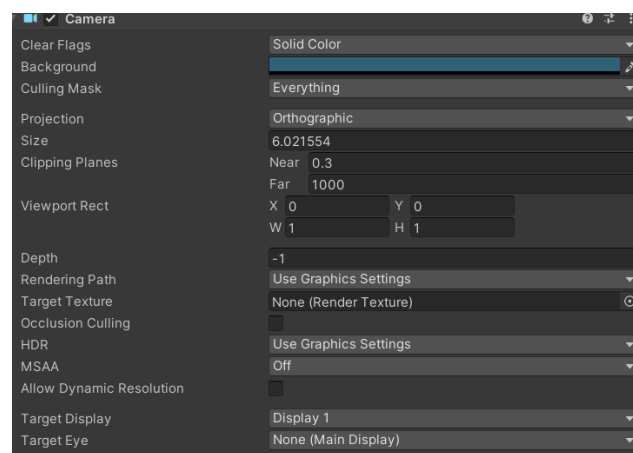
Aleatorios

Esta escena se encarga de generar mapas aleatorios mediante la clase generador, teniendo en cuenta algunos aspectos, como el número de picos que deberá tener el mapa, las monedas, y el jugador .



Main Camera

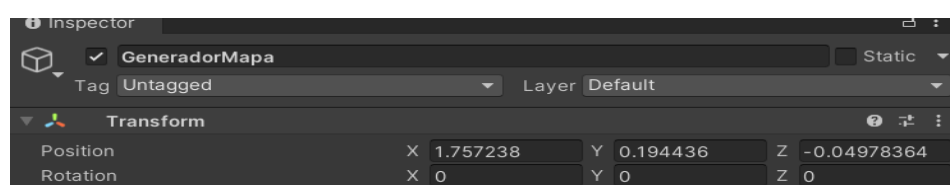
El objeto Camera son los dispositivos que capturan y muestran el mundo al jugador.

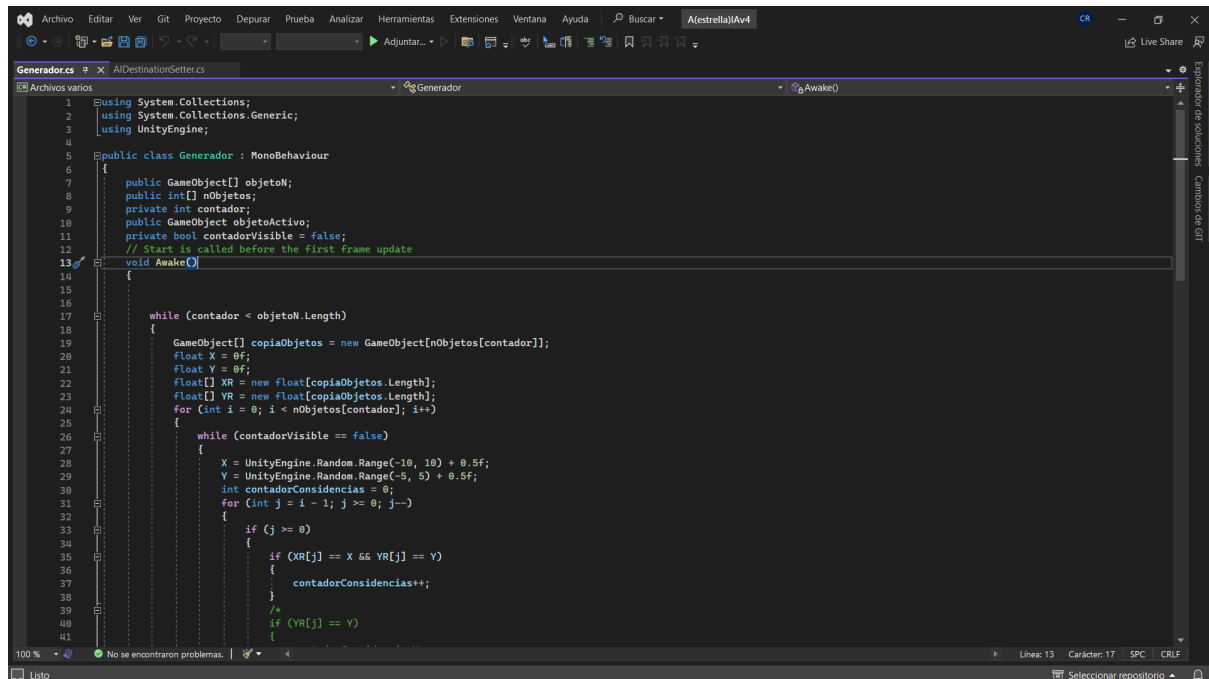


Clase Generador

Este script se encarga de generar una serie de objetos con la propiedad GraphUpdateScene en posiciones aleatorias dentro de un rango específico y evita que se superpongan en la misma posición. Se adjunta a un GameObject en el escenario de Unity y se configura desde el inspector para definir qué objetos generar y cuántos de cada tipo.

Una vez que esta clase termine de generar los objetos aleatorios se activará el objeto que traza las rutas (A(estrella)) y lo hará según el mapa que se generó.





```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Generador : MonoBehaviour
6 {
7     public GameObject[] objetoN;
8     public int[] nObjetos;
9     public GameObject objetoActivo;
10    private int contador;
11    private bool contadorVisible = false;
12    // Start is called before the first frame update
13    void Awake()
14    {
15
16
17        while (contador < objetoN.Length)
18        {
19            GameObject[] copiaObjetos = new GameObject[nObjetos[contador]];
20            float X = 0f;
21            float Y = 0f;
22            float[] XR = new float[copiaObjetos.Length];
23            float[] VR = new float[copiaObjetos.Length];
24            for (int i = 0; i < nObjetos[contador]; i++)
25            {
26                while (contadorVisible == false)
27                {
28                    X = UnityEngine.Random.Range(-10, 10) + 0.5f;
29                    Y = UnityEngine.Random.Range(-5, 5) + 0.5f;
30                    int contadorConsidencias = 0;
31                    for (int j = i - 1; j >= 0; j--)
32                    {
33                        if (j >= 0)
34                        {
35                            if (XR[j] == X && VR[j] == Y)
36                            {
37                                contadorConsidencias++;
38                            }
39                            /*
40                             * if (VR[j] == Y)
41                             {
```

Se importan las siguientes bibliotecas:

- System.Collections: Proporciona clases para trabajar con colecciones de datos.
- System.Collections.Generic: Proporciona tipos genéricos para trabajar con colecciones de datos.
- UnityEngine: Proporciona acceso a las clases y funciones de Unity.

Se define la clase Generador, que se hereda de MonoBehaviour, lo que significa que es un componente de Unity que puede ser adjuntado a un objeto en el escenario.

Se declaran las siguientes variables públicas:

- objetoN: Un arreglo de GameObjects que contiene los objetos que serán generados.
- nObjetos: Un arreglo de enteros que determina cuántos objetos de cada tipo se generarán.
- contador: Un entero que se utiliza para iterar a través del arreglo objetoN.
- objetoActivo: Un GameObject que puede ser configurado desde el inspector de Unity y que se activa al final de la generación de objetos.
- contadorVisible: Un booleano que se utiliza para controlar si un objeto se ha generado en una posición válida.

```
42         contadorConsidencias++;
43     }
44     /*
45     */
46     if (contadorConsidencias == 0)
47     {
48         Debug.Log("Posicion X: " + X + " Contador: " + contadorConsidencias);
49         Debug.Log("Posicion Y: " + Y + " Contador: " + contadorConsidencias);
50         VR[i] = Y;
51         XR[i] = X;
52     }
53     /*
54     */
55     if (i == 0)
56     {
57         XR[i] = X;
58         VR[i] = Y;
59         Debug.Log("Posicion X: " + X);
60         Debug.Log("Posicion Y: " + Y);
61         copiaObjetos[i] = Instantiate(objetoN[contador], new Vector3(XR[i], VR[i], -0.08f), Quaternion.identity);
62         contadorVisible = true;
63     }
64     /*
65     */
66     if (contadorConsidencias == 0)
67     {
68         contadorVisible = true;
69         copiaObjetos[i] = Instantiate(objetoN[contador], new Vector3(XR[i], VR[i], -0.08f), Quaternion.identity);
70     }
71     }
72     contadorVisible = false;
73     contador++;
74     objetoActivo.SetActive(true);
75 }
76
77 // Update is called once per frame
78 void Update()
79 {
80 }
81
82
```

En el método Awake(), que se ejecuta al inicio del juego:

- Se entra en un bucle while que itera a través de los elementos en el arreglo objetoN.
- Dentro del bucle, se crea un nuevo arreglo copiaObjetos de GameObjects con una longitud igual al valor correspondiente en nObjetos.
- Se declaran variables X e Y para almacenar las coordenadas de posición de los objetos generados.
- Se crean arreglos XR e YR para llevar un registro de las posiciones ya ocupadas por objetos generados previamente.

Luego, hay un bucle for que itera sobre el número de objetos que se deben generar (nObjetos[contador]):

- Dentro de este bucle, se inicia otro bucle while que se ejecuta mientras contadorVisible sea igual a false.
- Se generan valores aleatorios X e Y dentro de ciertos rangos.
- Se verifica si estas coordenadas ya han sido utilizadas antes para evitar superposiciones.
- Si las coordenadas son únicas (no se han utilizado previamente), se crea una copia del objeto desde objetoN[contador] en la posición (X, Y, -0.08f) y se almacena en el arreglo copiaObjetos.
- Se establece contadorVisible en true para salir del bucle while.

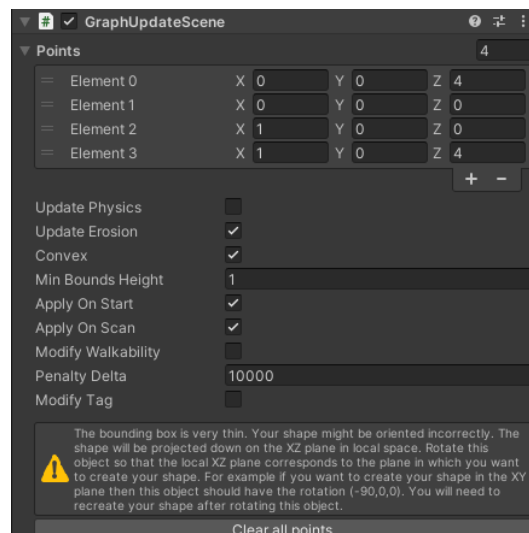
Al final de cada iteración del bucle for, se restablece contadorVisible a false para que el proceso se repita para el siguiente objeto.

El contador se incrementa para pasar al siguiente tipo de objeto.

Finalmente, después de que se hayan generado todos los objetos, se activa el objeto objetoActivo.

Objetos en el mapa (picos, hielo, monstruos, fuego, etc.)

Los objetos modifican la heurística general que aparece en el mapa dependiendo la posición en la que se encuentren con el componente GraphUpdateScene.



Picos: La penalización total por pasar por este objeto son 10000 puntos.



Hielo: La penalización total por pasar por este objeto son 2000 puntos.



Huesos: La penalización total por pasar por este objeto son 1000 puntos.



Fuego: La penalización total por pasar por este objeto son 8000 puntos.

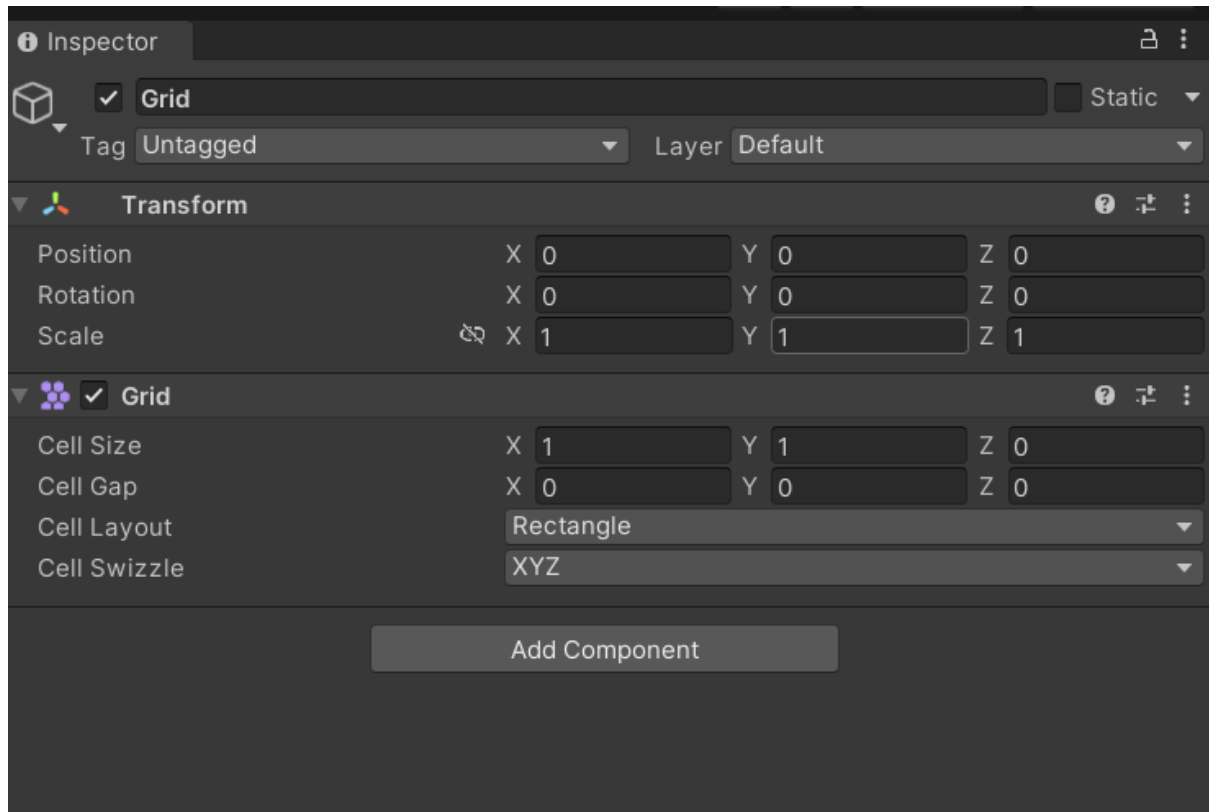


Monstruos: La penalización total por pasar por este objeto son 10000 puntos.



Grid

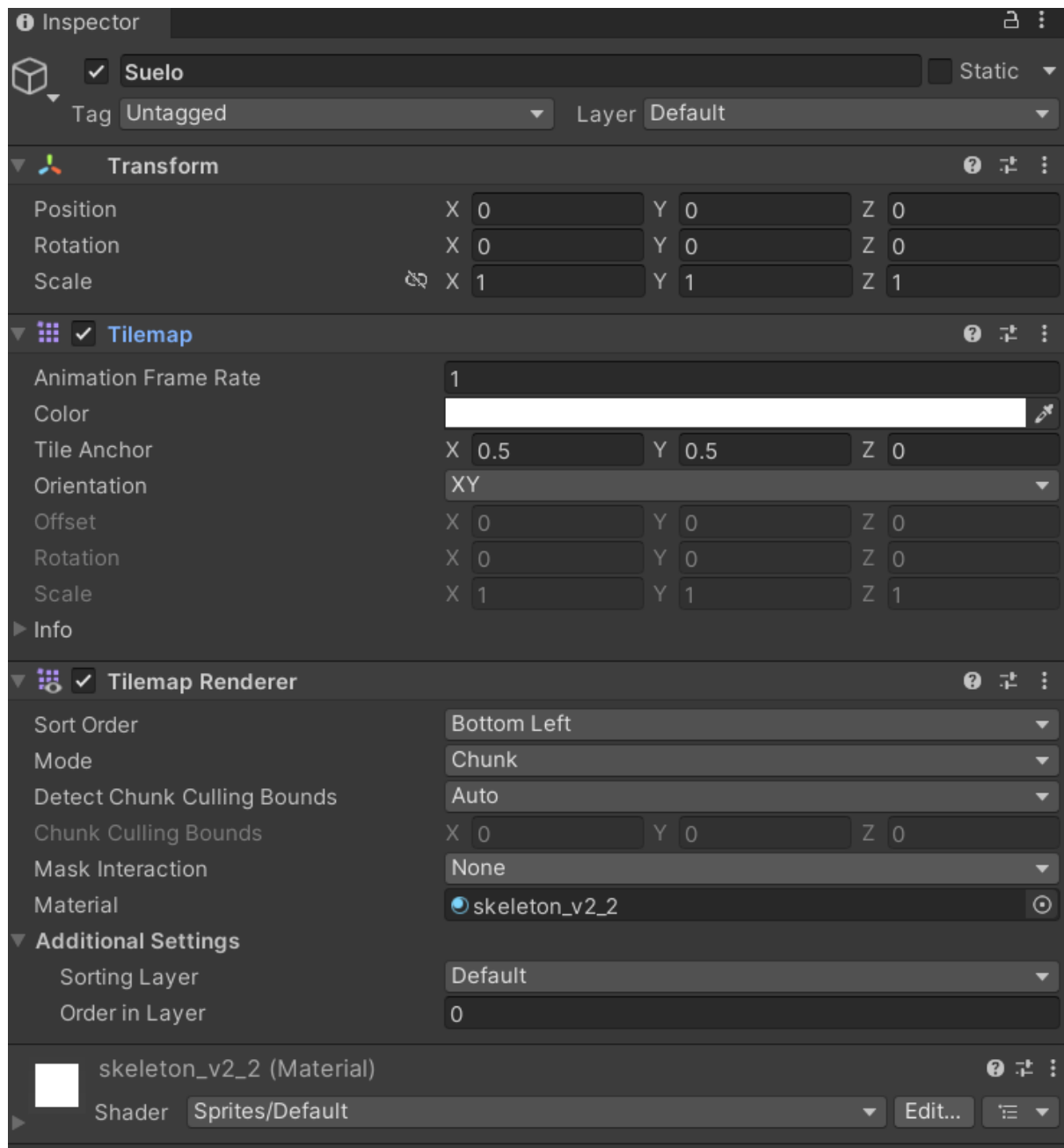
El componente Grid Layout Group coloca a sus layout elements (Elementos de diseño) hijos en una cuadrícula.



Propiedades

Propiedad:	Función:
Padding	El relleno dentro de los bordes del layout group.
Cell Size	El tamaño a ser utilizado para layout element (Elemento de diseño) en el grupo.

Suelo, Obstaculos y Obstaculos Suaves



El componente Tilemap es un sistema que almacena y maneja Tile Assets para crear niveles 2D. Transfiere la información requerida de los mosaicos colocados en él a otros componentes relacionados, como Tilemap Renderer y Tilemap Collider 2D.

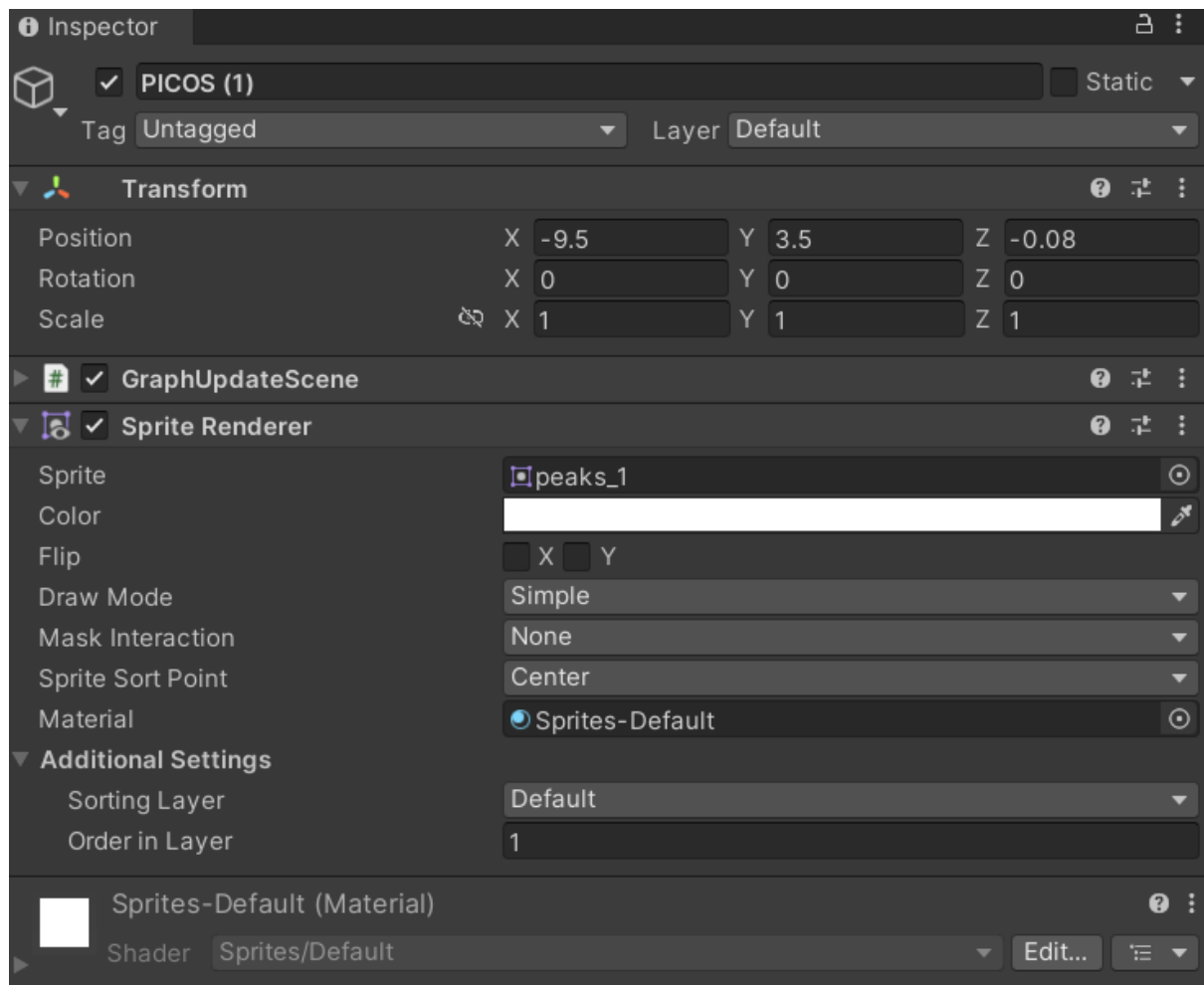
Se utiliza este componente con el componente Grid, o un Grid GameObject primario, para diseñar mosaicos en la cuadrícula asociada.

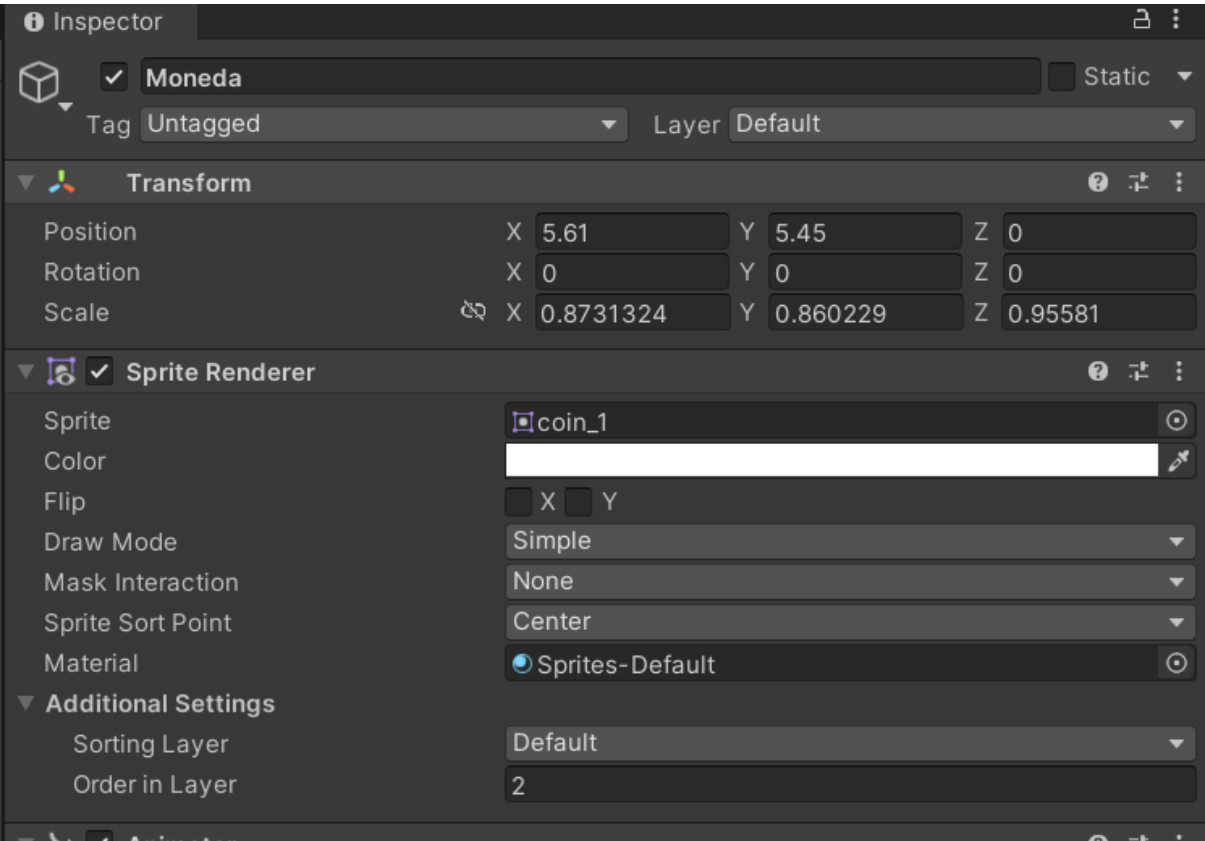
Propiedades

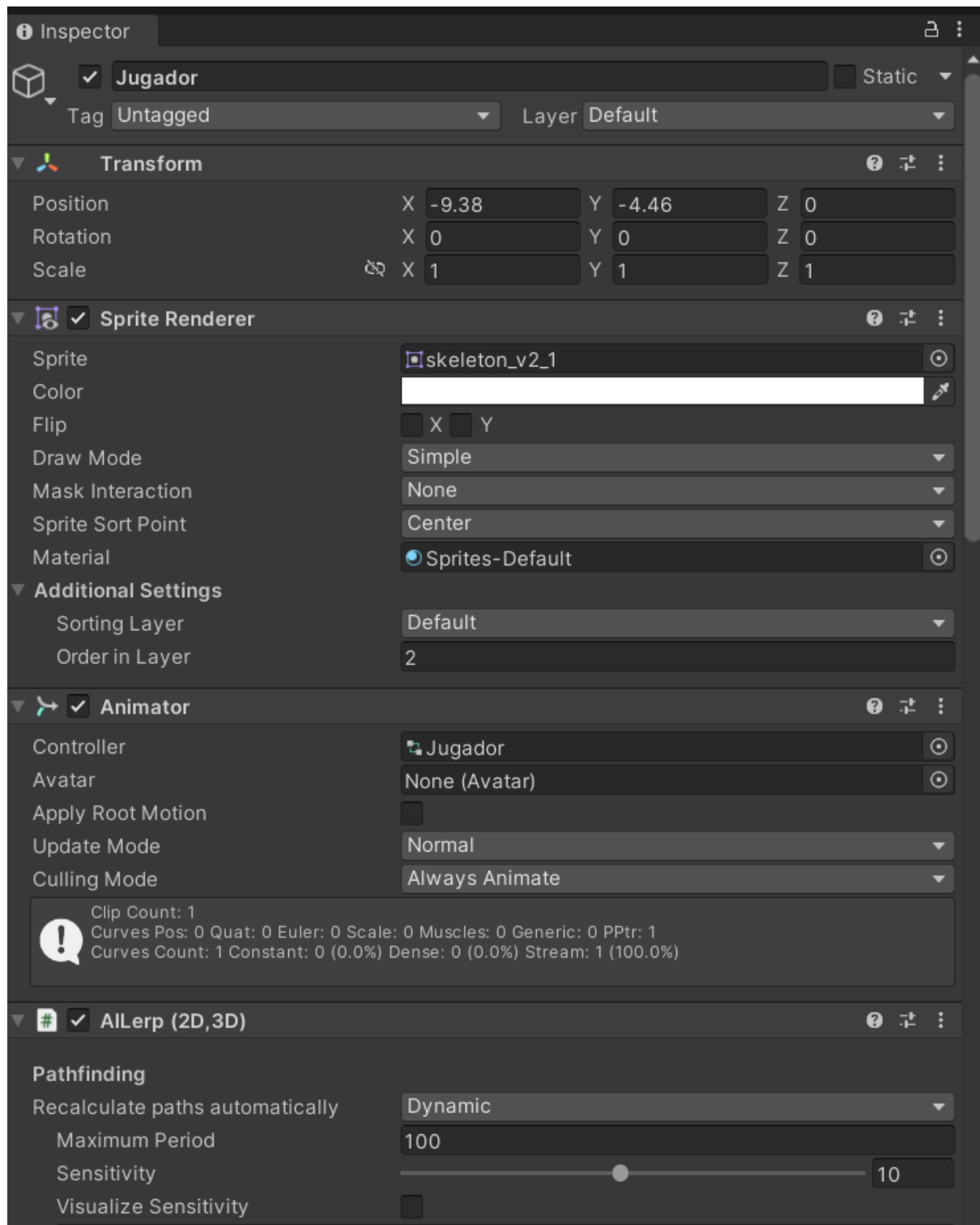
Propiedad	Función
Animation Frame Rate	La velocidad a la que Unity reproduce animaciones de mosaico. Al aumentar o disminuir esto, cambia la velocidad en el factor equivalente (por ejemplo, si establece esto en 2, Unity reproduce animaciones de mosaico al doble de velocidad).
Color	Tiñe los mosaicos de este mapa de teselas con el color seleccionado. Cuando se establece en blanco, Unity renderiza los mosaicos sin tinte.
Tile Anchor	El desplazamiento de anclaje de un mosaico en este mapa de mosaico. Ajuste los valores de posición para desplazar la posición de celda del mosaico en la cuadrícula asociada.
Orientation	La orientación de los mosaicos en el mapa de mosaico. Utilícelo si necesita orientar mosaicos en un plano específico (especialmente en una vista 3D).
XY	Unity orienta los mosaicos en el plano XY.
XZ	Unity orienta los mosaicos en el plano XZ.
YX	Unity orienta los mosaicos en el plano YX.
YZ	Unity orienta las fichas en el plano YZ.
ZX	Unity orienta los mosaicos en el plano ZX.
ZY	Unity orienta los mosaicos en el plano ZY.

Custom	Unity diseña los mosaicos según la configuración de la matriz de orientación personalizada establecida por los parámetros a continuación, que se pueden editar cuando selecciona esta opción.
Position	Muestra el desplazamiento de posición de la matriz de orientación actual. Sólo puede editar esta propiedad cuando la Orientación del mapa de teselas está establecida en Personalizada.
Rotation	Rotación de la matriz de orientación actual. Sólo puede editar esta propiedad cuando la Orientación del mapa de teselas está establecida en Personalizada.
Scale	Escala de la matriz de orientación actual. Sólo puede editar esta propiedad cuando la Orientación del mapa de teselas está establecida en Personalizada.

Jugador, Monedas, Picos







El componente Sprite Renderer representa el Sprite y controla cómo aparece visualmente en una escena para proyectos 2D y 3D.

Cuando creas un Sprite (GameObject > Objeto 2D > Sprite), Unity crea automáticamente un GameObject con el componente Sprite Renderer adjunto.

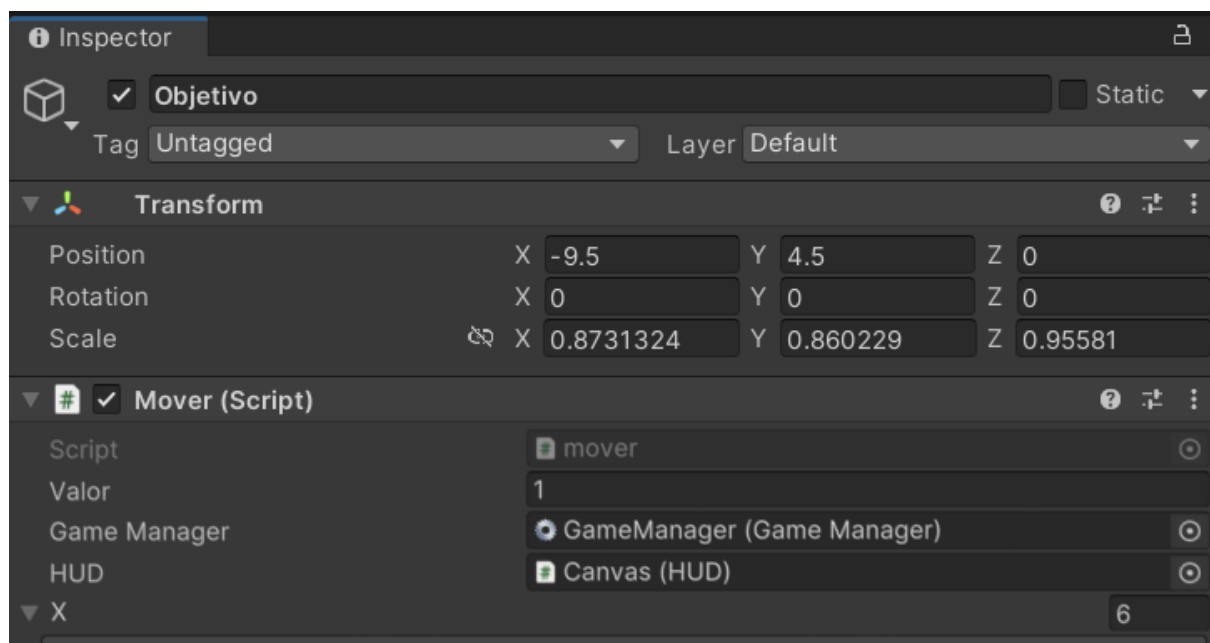
Propiedades

Propiedad	Función
Sprite	Defina qué textura de Sprite debe representar el componente. Haga clic en el pequeño punto de la derecha para abrir la ventana del selector de objetos y seleccione de la lista de activos de Sprite disponibles.
Color	Defina el color del vértice del Sprite, que tiñe o vuelve a colorear la imagen del Sprite. Utilice el selector de color para establecer el color del vértice de la textura de Sprite renderizada. Consulte la sección Color debajo de esta tabla para ver ejemplos.
Flip	Voltea la textura de Sprite a lo largo del eje marcado. Esto no cambia la posición Transformar del objeto GameObject.
Material	Defina el material utilizado para representar la textura Sprite.
Draw Mode	Define how the Sprite scales when its dimensions change. Select one of the following options from the drop-down box.
Sencillo	Toda la imagen se escala cuando cambian sus dimensiones. Esta es la opción predeterminada.
Sliced	Seleccione este modo si el Sprite está cortado en 9 cortes.
Size (<i>'Sliced'</i> <i>'Tiled'</i>) or	Ingrese el nuevo Ancho y Alto del Sprite para escalar el Sprite de 9 rodajas correctamente. También puede utilizar la herramienta Rect Transform para escalar el Sprite mientras aplica propiedades de 9-slicing.
Tiled	De forma predeterminada, este modo hace que el centro del Sprite de 9 cortes se coloque en mosaico en lugar de escalar cuando cambian sus

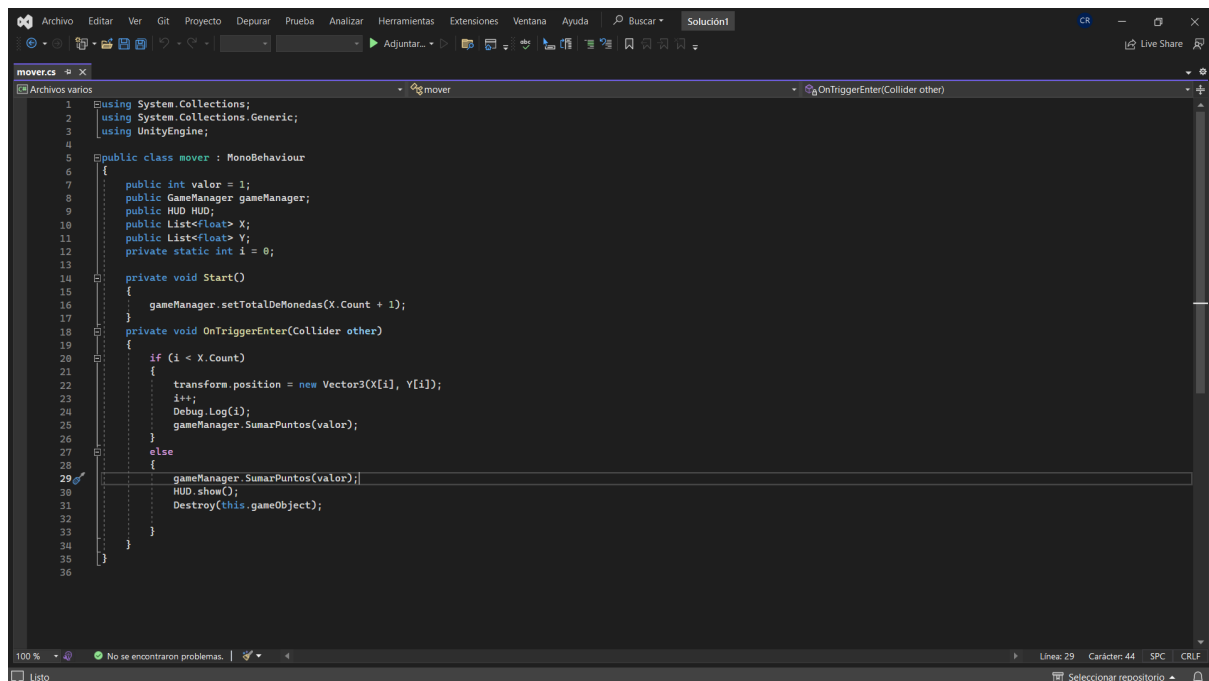
	dimensiones. Utilice el modo de mosaico para controlar el comportamiento de mosaico del Sprite.
Continuous	Este es el modo de mosaico predeterminado. En el modo continuo, los mosaicos de la sección media se equilibran uniformemente cuando cambian las dimensiones del Sprite.
Adaptive	En el modo adaptativo, la textura de Sprite se estira cuando cambian sus dimensiones, de forma similar al modo simple. Cuando la escala de las cotas modificadas cumple con el valor de ampliación, la sección media comienza a ser mosaico.
Stretch Value	Utilice el control deslizante para establecer el valor entre 0 y 1. El valor máximo es 1, que representa el doble de la escala original de Sprite.
Sorting Layer	Establezca la capa de clasificación del Sprite, que controla su prioridad durante el renderizado. Seleccione una capa de ordenación existente en el cuadro desplegable o cree una nueva capa de clasificación.
Order In Layer	Establezca la prioridad de renderizado del Sprite dentro de su capa de clasificación. Los Sprites con números más bajos se representan primero, con Sprites numerados más altos superponiéndose a los de abajo.
Mask Interaction	Establezca cómo se comporta el Renderizador de Sprites cuando interactúa con una máscara de Sprite. Vea ejemplos de las diferentes opciones en la sección Interacción de máscara a continuación.
None	El Renderizador de Sprites no interactúa con ninguna máscara de Sprite en la escena. Esta es la opción predeterminada.
Visible Inside Mask	El Sprite es visible donde la máscara de Sprite lo superpone, pero no fuera de él.

Visible Outside Mask	El Sprite es visible fuera de la máscara de Sprite, pero no dentro de ella. La máscara de Sprite oculta las secciones del Sprite que superpone.
Sprite Point Sort	Elija entre el centro del Sprite o su punto de pivote al calcular la distancia entre el Sprite y la cámara. Consulte la sección sobre Punto de clasificación de sprites para obtener más detalles.

Objetivo



Este script controla el movimiento de un GameObject a través de una serie de posiciones definidas en las listas X e Y, suma puntos al interactuar con otros objetos, y muestra elementos en la interfaz de usuario a medida que avanza en su recorrido. Cuando ha alcanzado todas las posiciones, el objeto se destruye.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class mover : MonoBehaviour
6 {
7     public int valor = 1;
8     public GameManager gameManager;
9     public HUD HUD;
10    public List<Float> X;
11    public List<Float> Y;
12    private static int i = 0;
13
14    private void Start()
15    {
16        gameManager.setTotalDeMonedas(X.Count + 1);
17    }
18    private void OnTriggerEnter(Collider other)
19    {
20        if (i < X.Count)
21        {
22            transform.position = new Vector3(X[i], Y[i]);
23            i++;
24            Debug.Log(i);
25            gameManager.SumarPuntos(valor);
26        }
27        else
28        {
29            gameManager.SumarPuntos(valor);
30            HUD.show();
31            Destroy(this.gameObject);
32        }
33    }
34 }
35
36
```

Se importan las siguientes bibliotecas:

- System.Collections: Proporciona clases para trabajar con colecciones de datos.
- System.Collections.Generic: Proporciona tipos genéricos para trabajar con colecciones de datos.
- UnityEngine: Proporciona acceso a las clases y funciones de Unity.

Se define la clase mover, que hereda de MonoBehaviour, lo que significa que es un componente de Unity que puede ser adjuntado a un objeto en el escenario.

Se declaran las siguientes variables públicas:

- valor: Un entero que representa el valor que se suma a la puntuación del juego cuando el objeto asociado a este script interactúa con otro objeto.
- gameManager: Una referencia a una instancia de la clase GameManager, que probablemente se encarga de gestionar aspectos del juego como la puntuación y otros elementos.
- HUD: Una referencia a una instancia de la clase HUD, que probablemente controla la interfaz de usuario del juego.
- X y Y: Listas de números flotantes que almacenan coordenadas X e Y respectivamente. Estas listas representan las posiciones a las que se moverá el GameObject cuando interactúe con otros objetos.
- i: Una variable estática que se inicializa en 0 y se utilizará para llevar un registro de la posición actual en las listas X e Y.

En el método Start():

- Se llama al método setTotalDeMonedas() del gameManager y se le pasa como argumento la cantidad de elementos en la lista X más 1. Esto sugiere que el gameManager podría estar relacionado con el conteo de monedas en el juego.

En el método OnTriggerEnter(Collider other):

- Este método se ejecuta cuando el GameObject con este script colisiona con otro objeto que tenga un componente Collider.
- Se verifica si el valor de i es menor que la cantidad de elementos en la lista X. Si es cierto, significa que todavía hay posiciones en las listas X e Y para mover el objeto.
- Si es cierto, se cambia la posición del objeto a las coordenadas (X[i], Y[i]), donde i representa la posición actual en las listas X e Y. Luego, i se incrementa en 1.
- Se registra en la consola el valor de i utilizando Debug.Log().
- Se llama al método SumarPuntos(valor) del gameManager para aumentar la puntuación del juego.

Si i es igual o mayor que la cantidad de elementos en las listas X e Y, significa que el objeto ha alcanzado todas las posiciones disponibles y debe ser destruido:

- Se llama al método SumarPuntos(valor) del gameManager nuevamente para asegurarse de sumar puntos antes de destruir el objeto.
- Se llama al método show() de la clase HUD, lo que probablemente muestra algún elemento de la interfaz de usuario.
- Se destruye el objeto asociado a este script utilizando Destroy(this.gameObject).

Conclusion

Como conclusión, este proyecto nos ha enseñado la importancia de la planificación y la implementación cuidadosa en el desarrollo de sistemas de inteligencia artificial. Además, hemos ganado experiencia en el uso de librerías y algoritmos como A* con heurística y costo para resolver problemas complejos.

Este proyecto también ha destacado la necesidad de adaptabilidad, ya que el objetivo cambia constantemente de ubicación. A través de la colaboración, hemos enfrentado desafíos y resuelto problemas de diseño y optimización de manera efectiva.

Referencias

- arongranberg. (29 de 09 de 2023). Obtenido de getstarted:
<https://arongranberg.com/astar/docs/getstarted.html>
- arongranberg. (29 de 09 de 2023). Obtenido de Class Seeker Extends
VersionedMonoBehaviour: <https://arongranberg.com/astar/docs/seeker.html#>
- arongranberg. (29 de 09 de 2023). Obtenido de versionedmonobehaviour:
<https://arongranberg.com/astar/docs/versionedmonobehaviour.html>
- arongranberg. (29 de 09 de 2023). Obtenido de aidestinationsetter:
<https://arongranberg.com/astar/docs/aidestinationsetter.html>
- arongranberg. (29 de 09 de 2023). Obtenido de movementscripts:
<https://arongranberg.com/astar/docs/movementscripts.html>
- arongranberg. (29 de 09 de 2023). Obtenido de modifiers2:
<https://arongranberg.com/astar/docs/modifiers2.html>
- arongranberg. (29 de 09 de 2023). Obtenido de ailerp:
<https://arongranberg.com/astar/docs/ailerp.html#destination>
- arongranberg. (29 de 09 de 2023). *aipath*. Obtenido de
<https://arongranberg.com/astar/docs/aipath.html>
- arongranberg. (29 de 09 de 2023). *astarpath*. Obtenido de
<https://arongranberg.com/astar/docs/astarpath.html>
- arongranberg. (29 de 09 de 2023). *custom_movement_script*. Obtenido de
https://arongranberg.com/astar/docs/custom_movement_script.html
- arongranberg. (29 de 09 de 2023). *iastarai*. Obtenido de
<https://arongranberg.com/astar/docs/iastarai.html#destination>
- arongranberg. (29 de 09 de 2023). *odelink2*. Obtenido de
<https://arongranberg.com/astar/docs/odelink2.html>
- arongranberg. (29 de 09 de 2023). *pathfinding2d*. Obtenido de
<https://arongranberg.com/astar/docs/pathfinding2d.html>
- arongranberg. (29 de 09 de 2023). *richai*. Obtenido de
<https://arongranberg.com/astar/docs/richai.html>
- arongranberg. (29 de 09 de 2023). *Writing Modifiers*. Obtenido de
<https://arongranberg.com/astar/docs/writemodifiers.html>
- unity. (23 de 09 de 2023). Obtenido de class-SpriteRenderer:
<https://docs.unity3d.com/es/2018.4/Manual/class-SpriteRenderer.html>

unity. (29 de 09 de 2023). Obtenido de class-Tilemap:
<https://docs.unity3d.com/es/2018.4/Manual/class-Tilemap.html>

unity. (29 de 09 de 2023). *CreatingScenes*. Obtenido de
<https://docs.unity3d.com/es/530/Manual/CreatingScenes.html>

wikipedia. (29 de 09 de 2023). Obtenido de Singleton_pattern:
https://en.wikipedia.org/wiki/Singleton_pattern