

## Tema 4. Un juego en primera persona

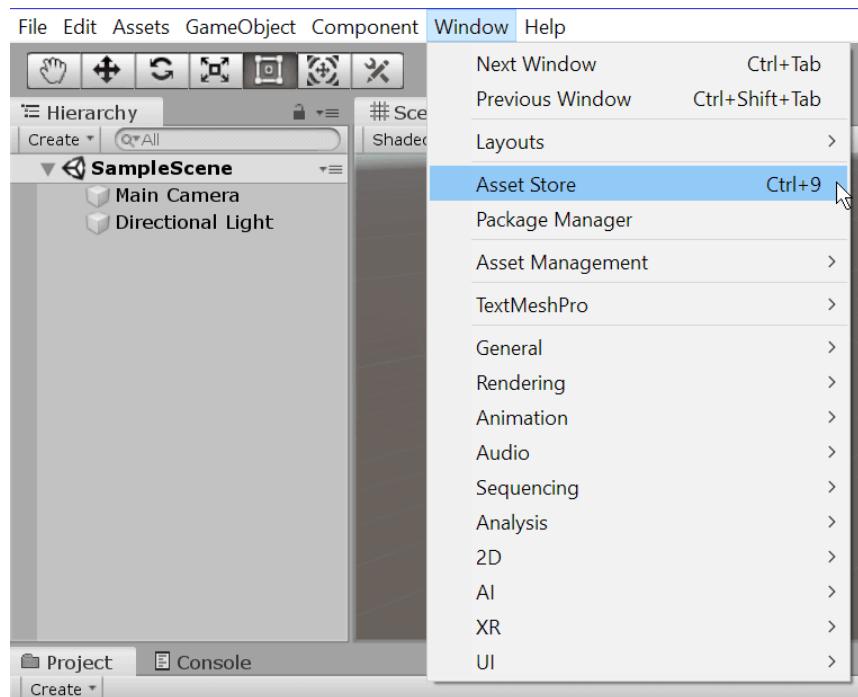
En este tema vamos a profundizar en el mundo de las 3D en Unity, en este caso, empleando también elementos descargados desde la "tienda Unity".

### 4.1. Añadiendo un "FPS controller"

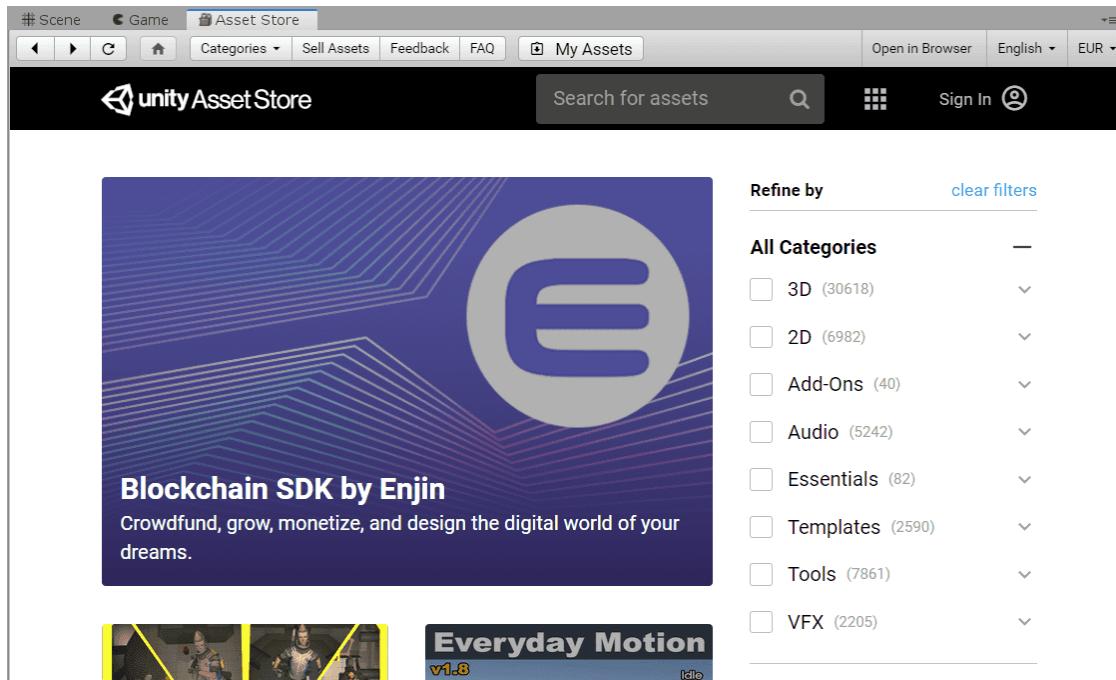
Hemos visto cómo hacer que la cámara siga al personaje, tanto haciendo que la cámara sea un "hijo" del personaje en 3D como usando Cinemachine en 2D. En esta ocasión vamos a usar otra alternativa "prefabricada" y que puede resultar útil para muchos juegos en 3D.

Comenzaremos por crear un nuevo proyecto en 3D, que podemos llamar (por ejemplo) "FPS" (la abreviatura de "First Person Shooter", juego de disparos en primera persona, que es el tipo de juego al que nos vamos a acercar).

A continuación, iremos al menú Window / Asset Store, para acceder a la "tienda Unity":



Dentro de ésta, usaremos la opción de Búsqueda para encontrar los "Standard Assets"



Y es de esperar que el primero que nos aparezca sea uno gratuito, creado por la propia "Unity Technologies":

The screenshot shows the Unity Asset Store search results for "standard assets". The interface is similar to the homepage, with tabs for Scene, Game, and Asset Store, and links for Categories, Sell Assets, Feedback, FAQ, My Assets, Open in Browser, English, and EUR. The search bar contains the query "standard assets". The results list shows "1-10 of 10 results for standard assets". Two assets are listed: "UNITY TECHNOLOGIES Standard Assets (for Unity 2017.)" and "PRODUCTIVE EDGE EDGE Standard Assets". Both assets are marked as "FREE". To the right of the results is a sidebar titled "Refine by" with a "clear filters" link, listing categories: All Categories (3D, Essentials, Tools), Pricing, Unity Versions, Publisher, and Ratings. There are also buttons for "standard assets" and a magnifying glass icon.

Si hacemos clic para ver los detalles, se nos mostrará un resumen de su contenido (entre otras cosas, un "First Person Character Controller") que es lo primero que vamos a usar), su tamaño (182 MB) y un botón para importar a nuestro proyecto ("Import") o, si no estamos identificados, para descargar ("Download"):

UNITY TECHNOLOGIES  
Standard Assets      FREE

**★★★★★** 648 user reviews      Download

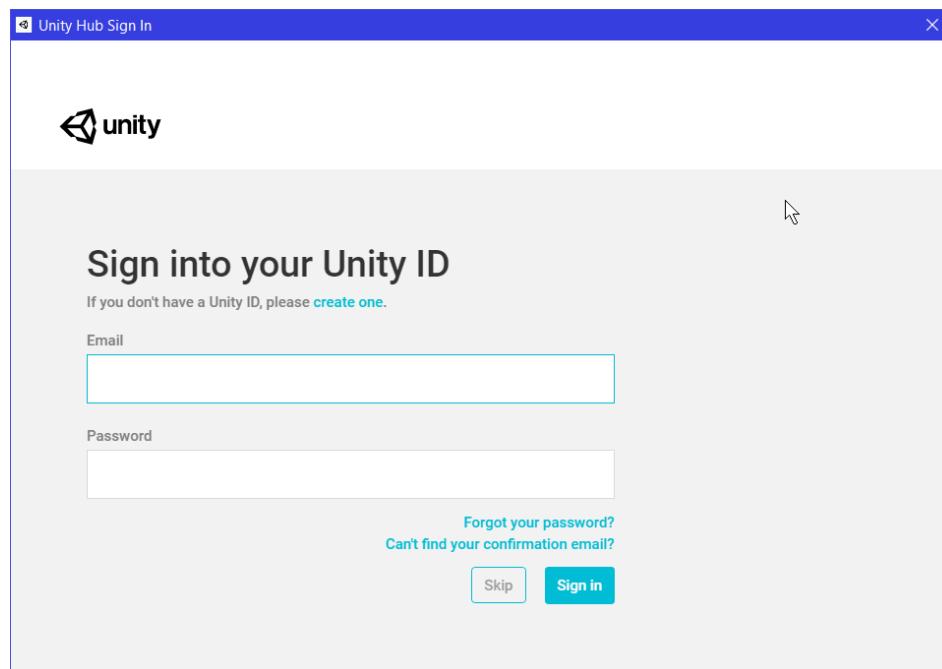
This collection of assets, scripts, and example scenes can be used to kickstart your Unity learning or be used as the basis for your own projects.

The package includes:

- First Person Character Controller
- Third Person Character Controller
- Car Controller
- Aircraft Controller
- Particle Example Scene
- Rollerball Controller
- Sample 2D Platformer Scene
- Camera Rigs

Show More

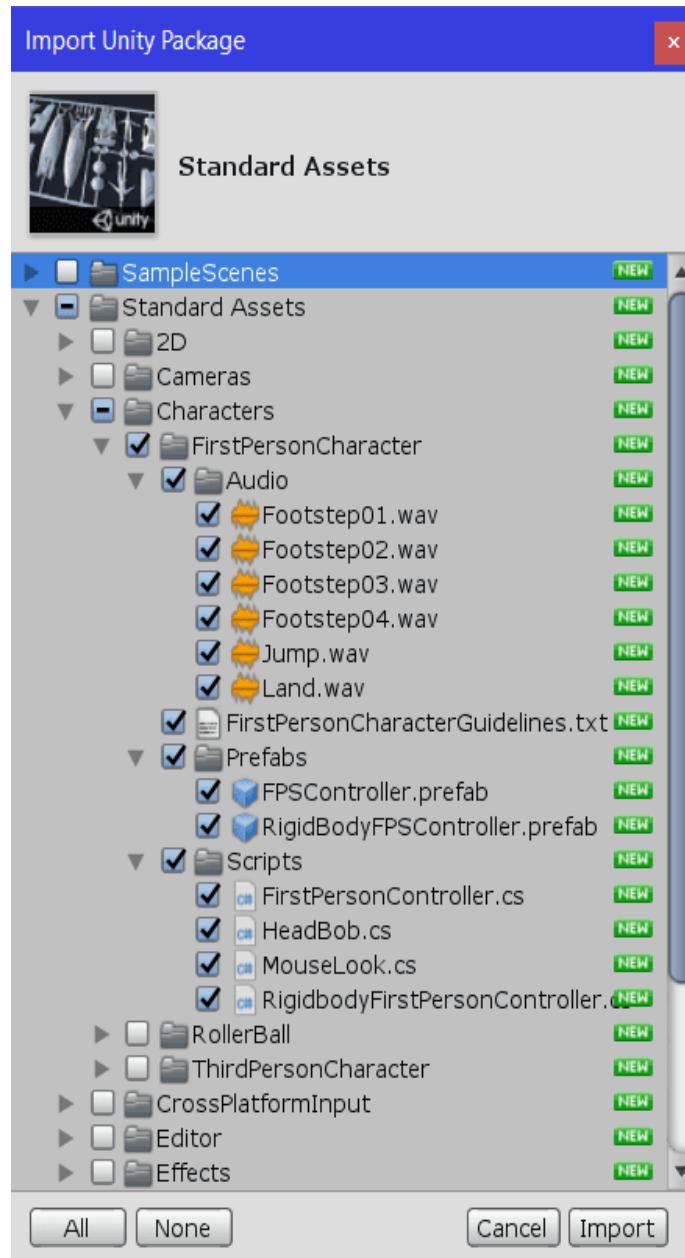
Si no estamos identificados, al hacer clic en "Download", se nos pedirá que nos identifiquemos en este momento:



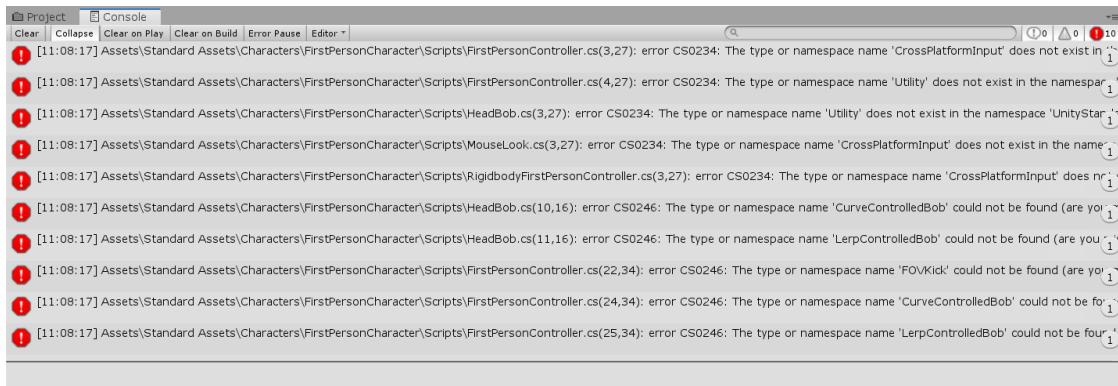
Y entonces ya podremos descargar e importar. Aparecerá una ventana en la que podemos seleccionar qué deseamos importar de los muchos componentes que tiene este paquete.



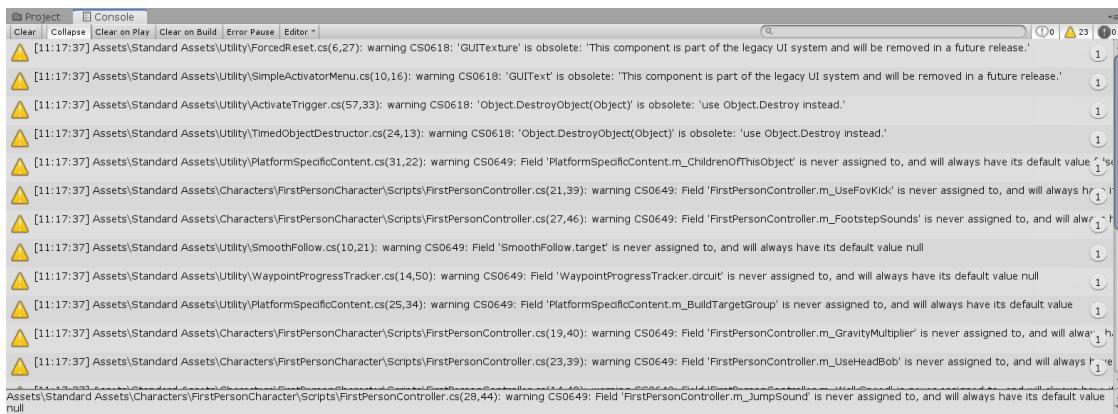
Podemos importar varias veces desde este paquete, por lo que puede ser interesante no incluir los 182 MB de cosas que no vamos a utilizar, y seleccionar exclusivamente los elementos de la categoría "First Person Character" (y más adelante iremos añadiendo elementos adicionales, cuando los necesitemos):



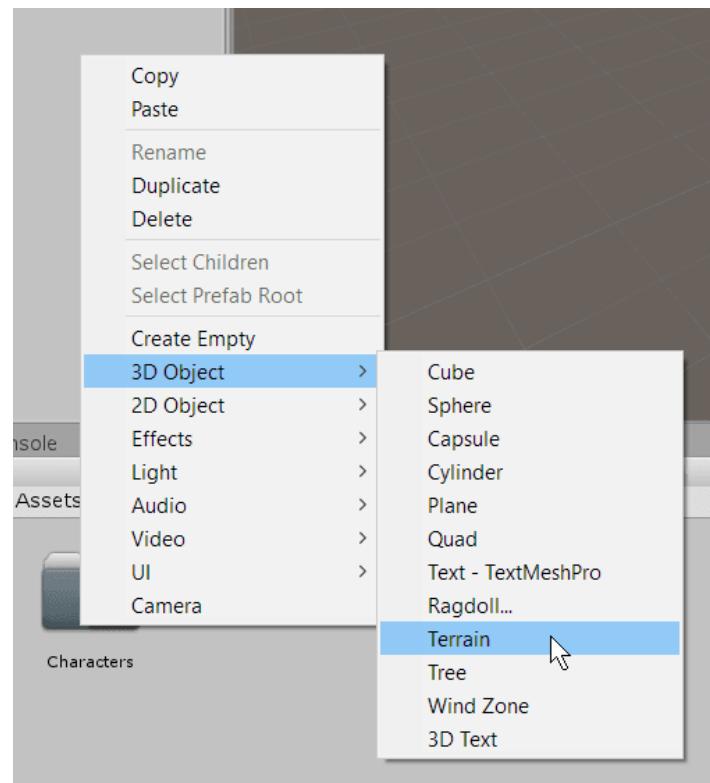
El inconveniente de proceder de esa manera es que puede haber dependencias que nosotros no conoczamos y que luego se conviertan en mensajes de error:



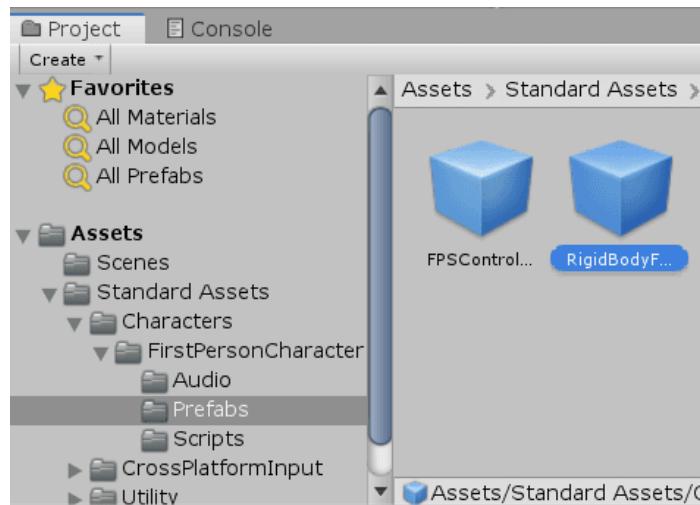
En nuestro caso, deberemos importar también "CrossPlatformInput" y "Utility", y así desaparecerán todos los mensajes de error, aunque quedará algún "warning", tanto por usar componentes "anticuados" como porque algún atributo no tenga (todavía) valores iniciales. Aun así, esos "warnings" no impiden que podamos seguir adelante con nuestro juego:



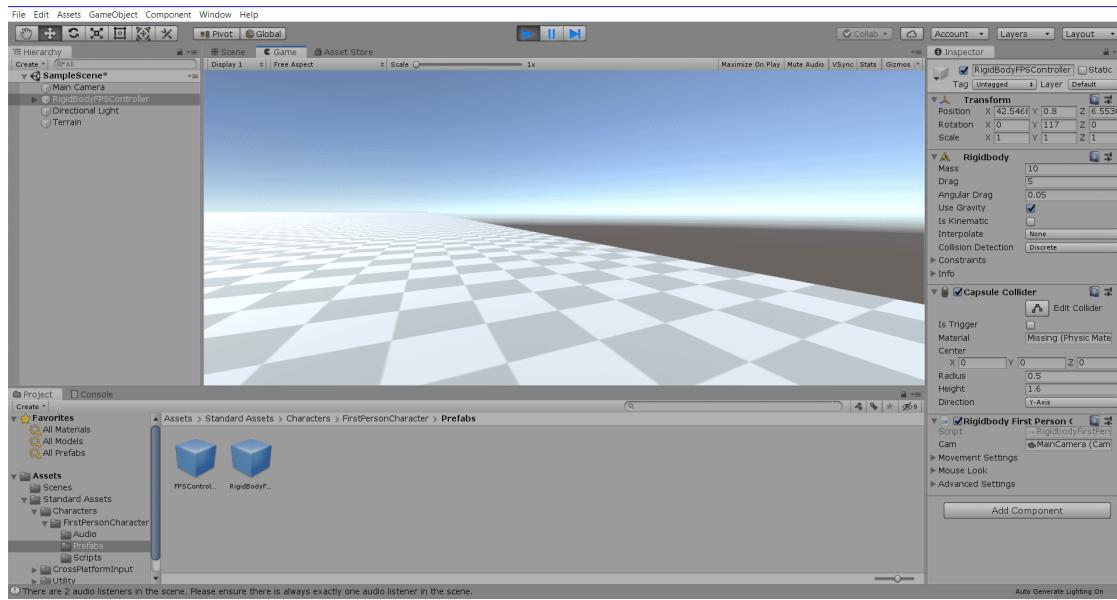
Para poder probar el movimiento del personaje, podemos añadir un "plano" sobre el que movernos, o bien directamente incluir ya un "terreno", cuya superficie alteraremos posteriormente. Lo creamos desde la jerarquía, con el botón derecho, 3D Object / Terrain



Y ahora ya podemos incluir nuestro personaje. Al importar se nos habrá creado una carpeta "Standard Assets", que contendrá otra "Characters", con "FirstPersonCharacter" y a su vez "Prefabs". En esta última carpeta tendremos dos controladores, uno que no tiene un RigidBody asociado y otro que sí. Usaremos este segundo.



Si lanzamos el juego, veremos que podemos movernos con las flechas del teclado o como WASD, saltar con espacio o girar con el ratón:



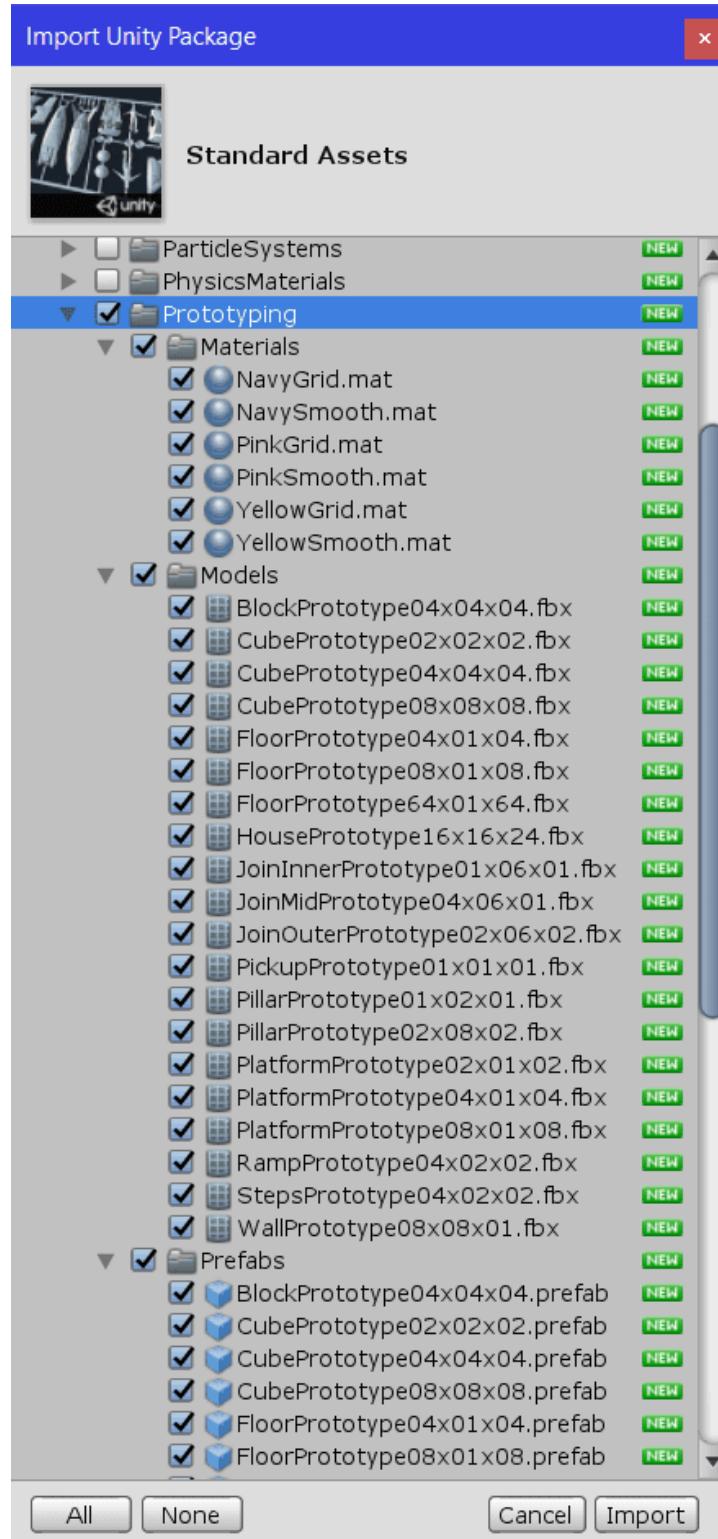
En la parte inferior se nos avisa de que tenemos dos "Audio Listeners". Realmente, lo que tenemos son dos cámaras, porque ha aparecido una nueva cámara

(Para liberar el ratón y poder detener la ejecución, pulsa ESC).

**Ejercicio propuesto 4.1.1:** Crea un proyecto 3D vacío, añade en él un FPS controller, un terreno y comprueba que puedes moverte por ese terreno.

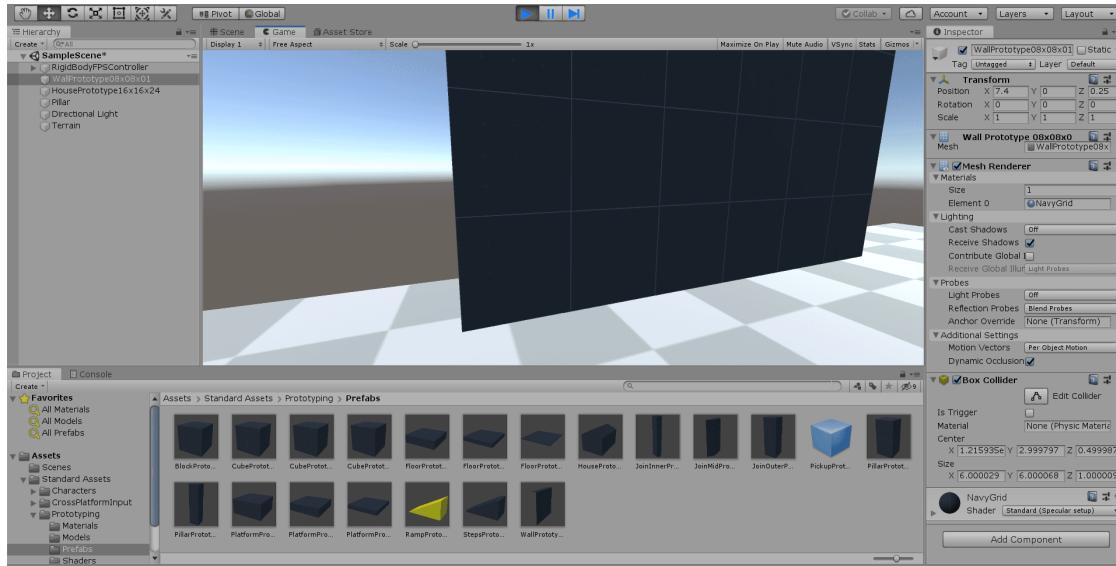
## 4.2. Assets predefinidos para crear un nivel

Podemos crear muchos tipos de elementos a partir de los componentes 3D básicos, como cubos cilindros y esferas. Aun así, dentro de los "standard assets" existe una categoría "Prototyping", que contiene ciertos assets que nos pueden ayudar a hacer un prototipo inicial de un juego como el nuestro.

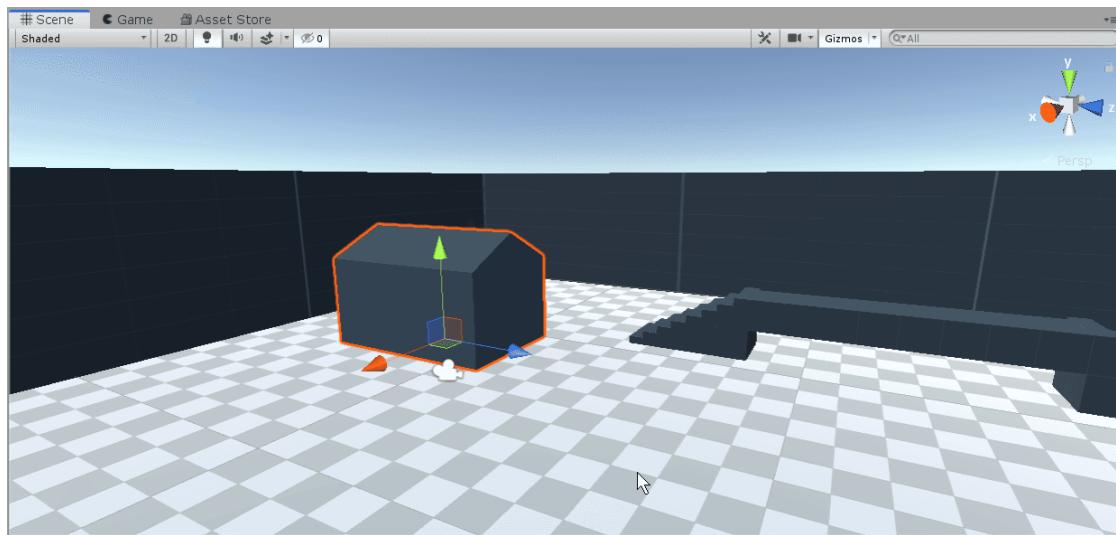


Por ejemplo, tenemos un "prefab" llamado "WallPrototype08x08x01", que es un muro de 8 unidades (metros) de altura, 8 de anchura y 1 de profundidad. Además, si lanzamos el juego y nos

acercamos a él, veremos que su material por defecto incluye una "cuadrícula" (grid) que nos ayuda a calcular mejor las proporciones:



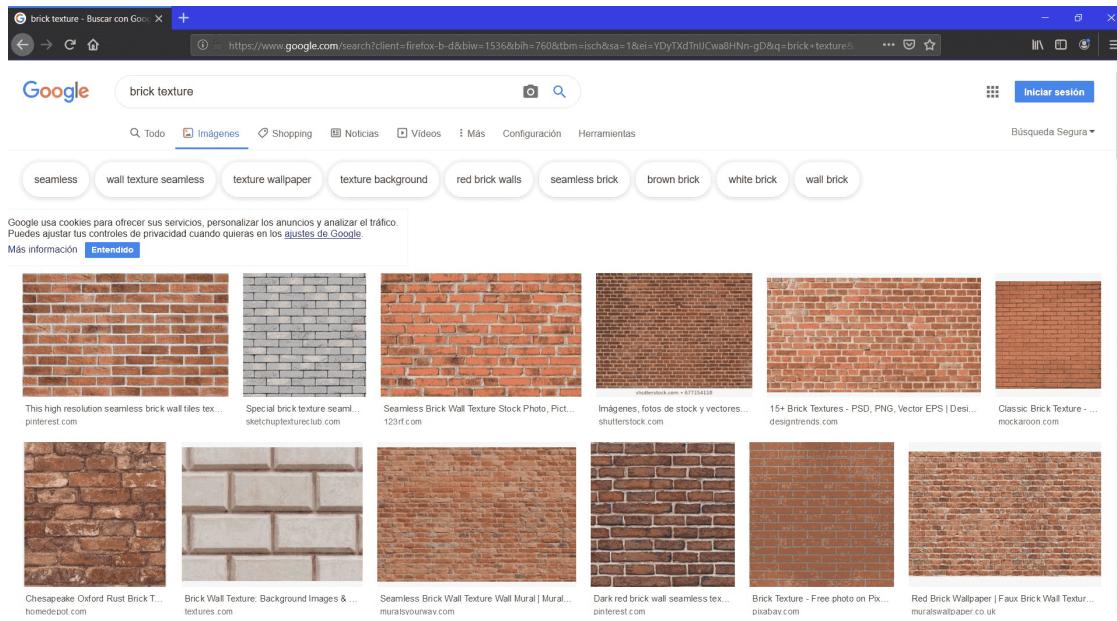
También tenemos casas (muy grandes), rampas, escaleras, columnas...



**Ejercicio propuesto 4.2.1:** Crea un nivel que incluya como mínimo varios muros, una casa (no demasiado grande), alguna escalera y alguna rampa.

### 4.3. Más sobre texturas

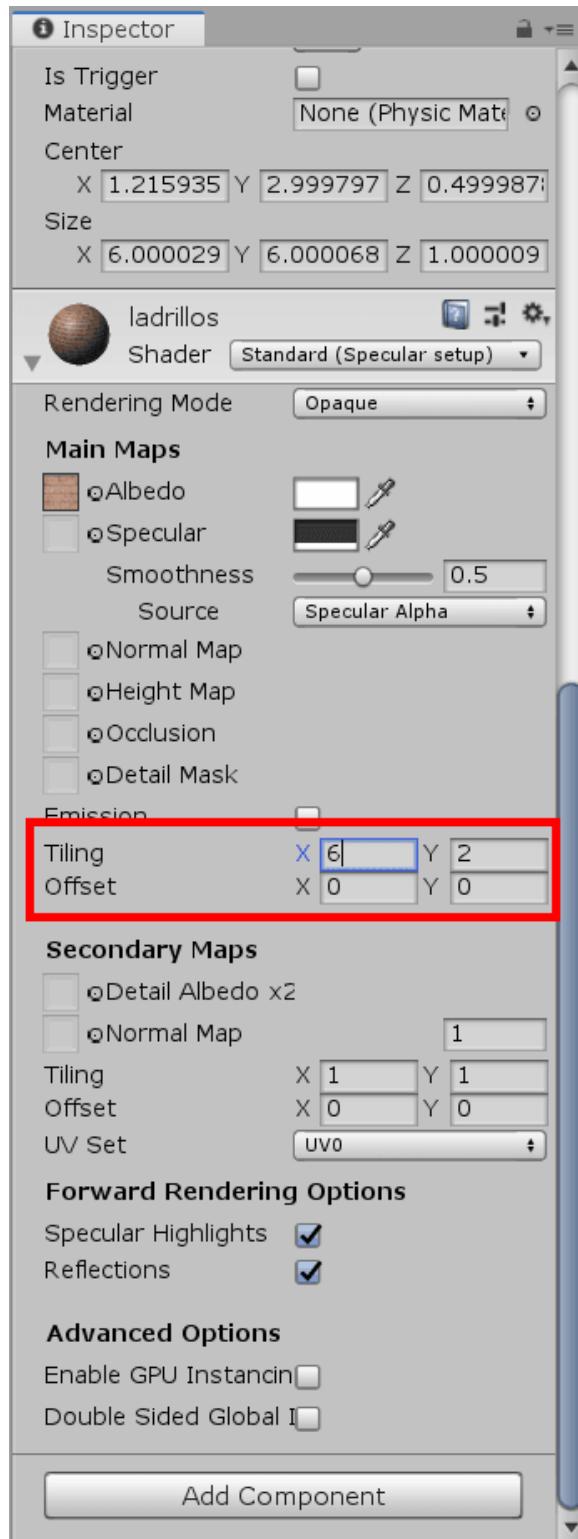
Hemos visto cómo aplicar un color a un material y cómo hacer que tenga un brillo más o menos metálico. Pero también podemos usar una imagen como textura, y eso podría hacer que nuestros muros resultasen más reales. Por ejemplo, podemos localizar una imagen de ladrillos con nuestro buscador de Internet favorito:



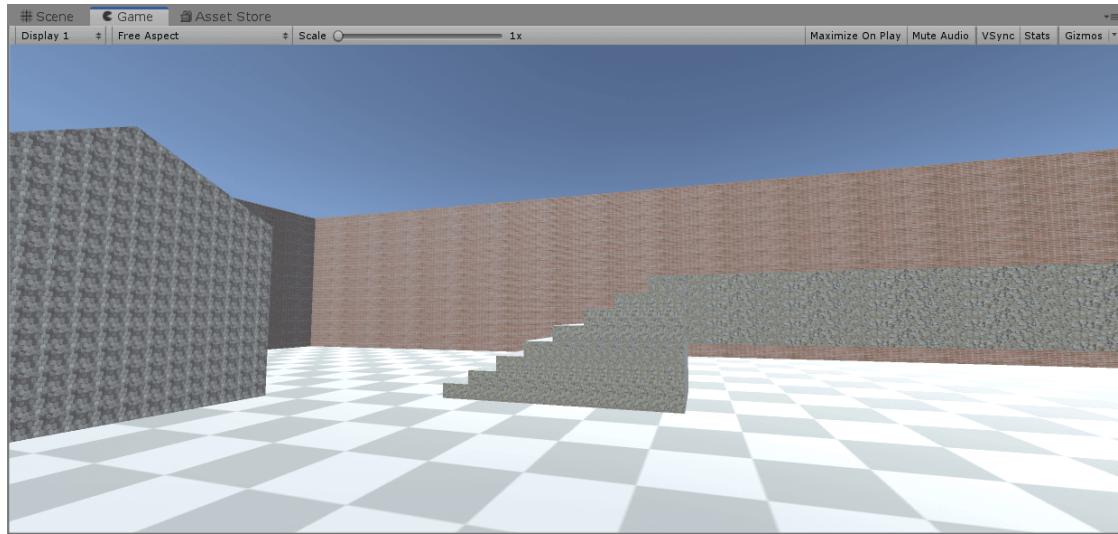
Conviene que sea una imagen "diseñada para ser una textura", de forma que la imagen dé una buena apariencia si se muestra repetidas varias veces, una al lado de otra. En inglés se suele usar la palabra "seamless" para ese tipo de texturas, que te puede ayudar en tu búsqueda en Internet.

Descargaremos la imagen que nos guste, crearemos una carpeta para "Texturas" en nuestro proyecto de Unity y la arrastraremos a dicha carpeta. Posteriormente, para asignar esa textura a un objeto, basta arrastrarla desde el panel inferior hasta el objeto que nos interese.

Eso sí, la textura se "estirará" hasta ocupar toda la superficie del objeto. Generalmente el efecto será mejor si hacemos que se repita varias veces, lo que podemos conseguir con la opción "Tiling" de la textura, en el inspector. Si no queremos que empiece justo en la esquina, podemos desplazarla cambiando su "Offset":



Así, podemos buscar varias texturas y aplicarlas a cada uno de los elementos de nuestro nivel, a nuestro gusto. Podría quedar algo así:



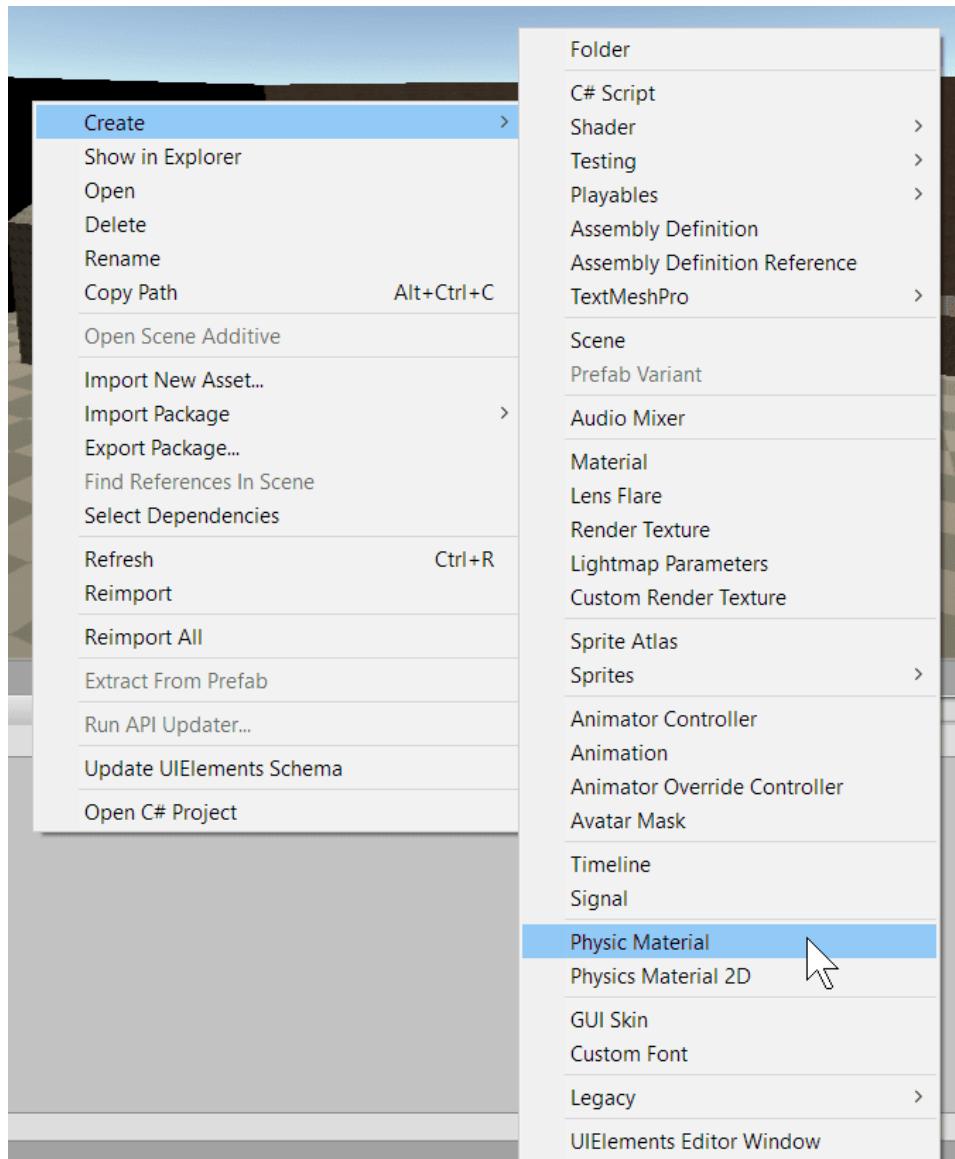
Dentro de poco cambiaremos la textura del suelo, junto con su relieve.

**Ejercicio propuesto 4.3.1:** Mejora la apariencia de los elementos de tu nivel, usando texturas.

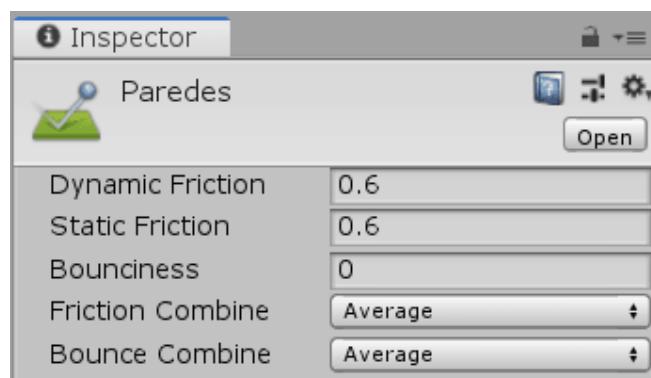
#### 4.4. Fricción

Ahora mismo, si nuestro personaje se acerca a una pared (o un lateral de la casa) y se apoya en ella, no puede avanzar. La fricción es demasiado grande. Pero eso no es lo que suele ocurrir en los juegos FPS, en los que es habitual poder deslizarse junto a la pared.

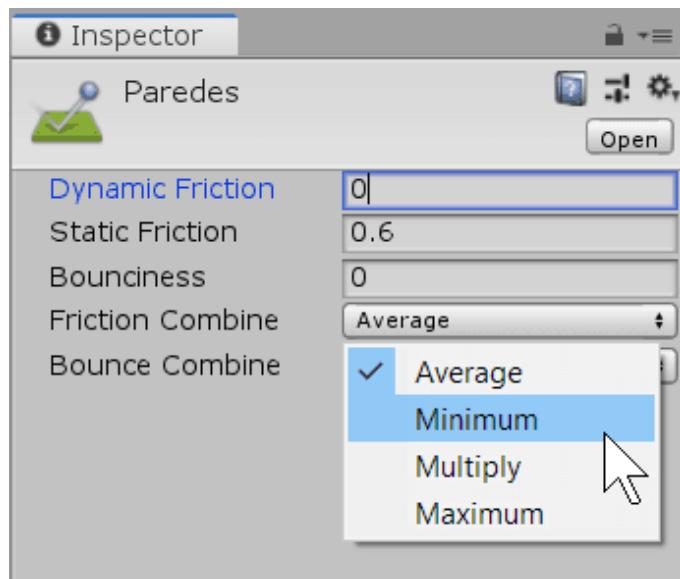
La forma de evitarlo es crear un nuevo "material físico", para asignárselo a esos elementos:



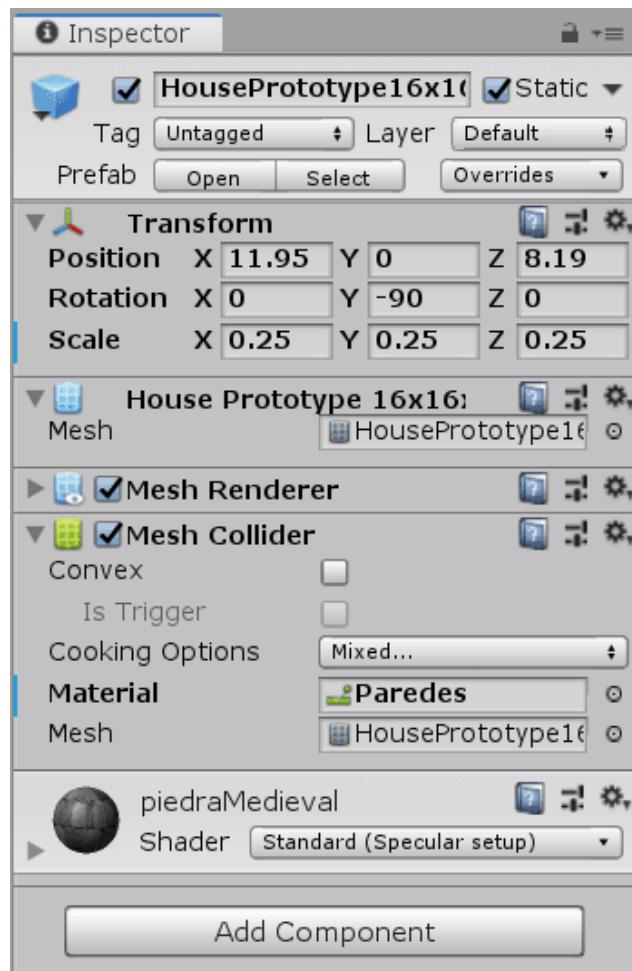
Le podemos dar el nombre "paredes". Veremos que sus valores iniciales son que tenga una fricción (dinámica) de 0.6, y que cuando roza con otro elemento, la fricción calculada sea la media de ambas:



Así que podemos bajar la fricción a 0 y decir, que en caso de rozar otro objeto, se tome ese valor mínimo (que será 0):



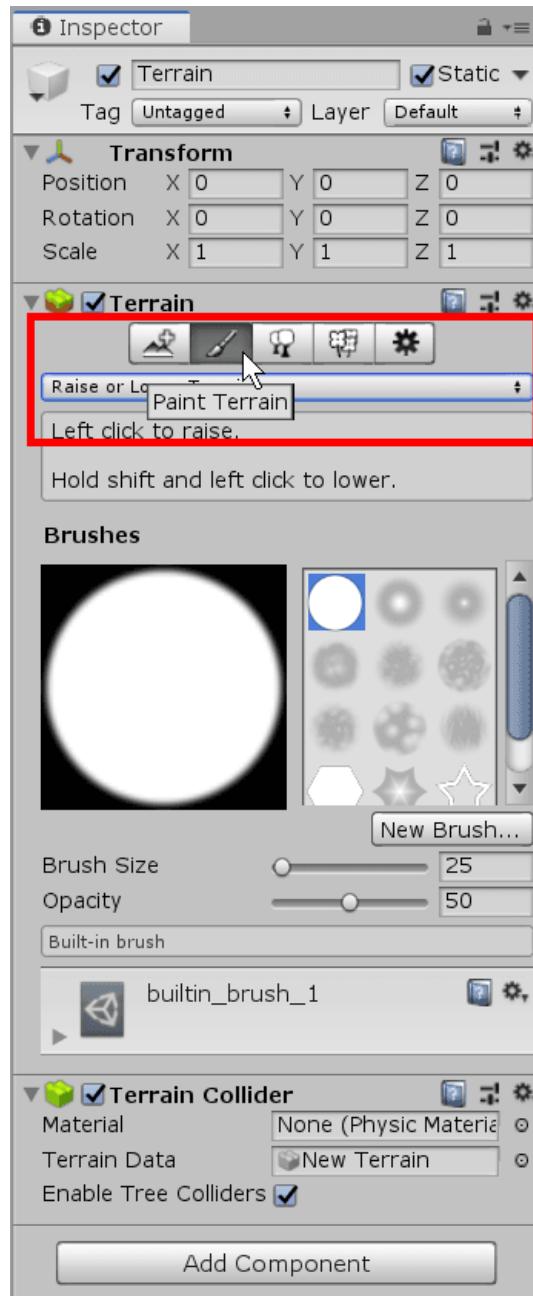
Ya sólo queda arrastrar ese material a las paredes y demás objetos que nos interese, y en el inspector aparecerá como parte de su "collider":



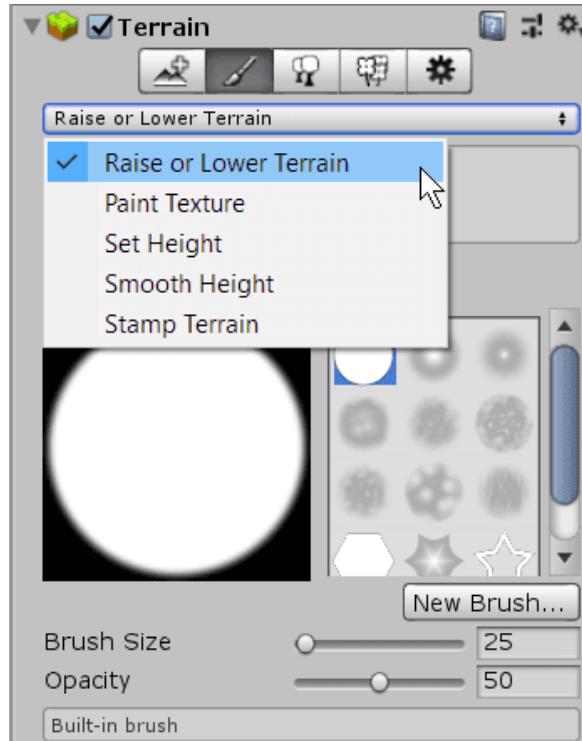
**Ejercicio propuesto 4.4.1:** Haz que no haya tanta fricción entre tu personaje y las paredes.

## 4.5. Un terreno irregular

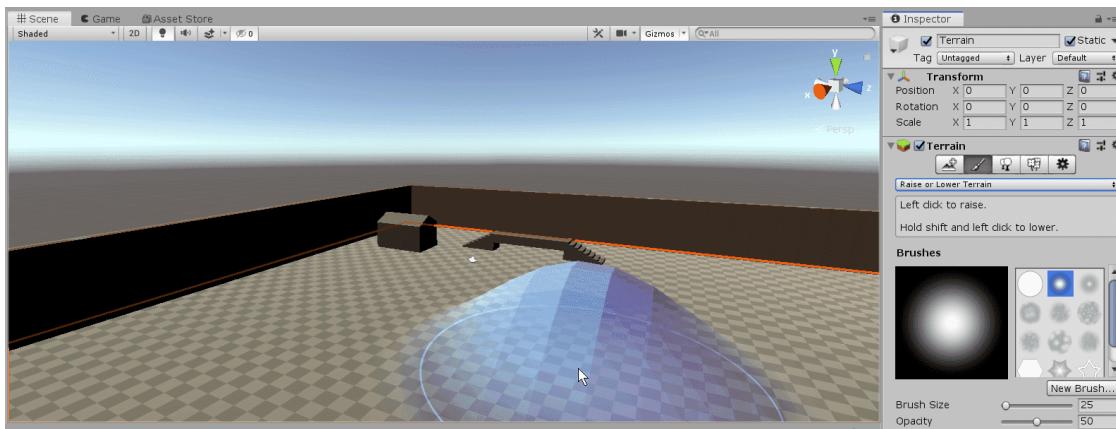
Un "terreno" no es simplemente un plano. Incluye facilidades para indicar relieve y aplicar texturas. Si hacemos clic en el inspector, veremos una barra de herramientas que permite entre otras cosas "pintar el terreno":



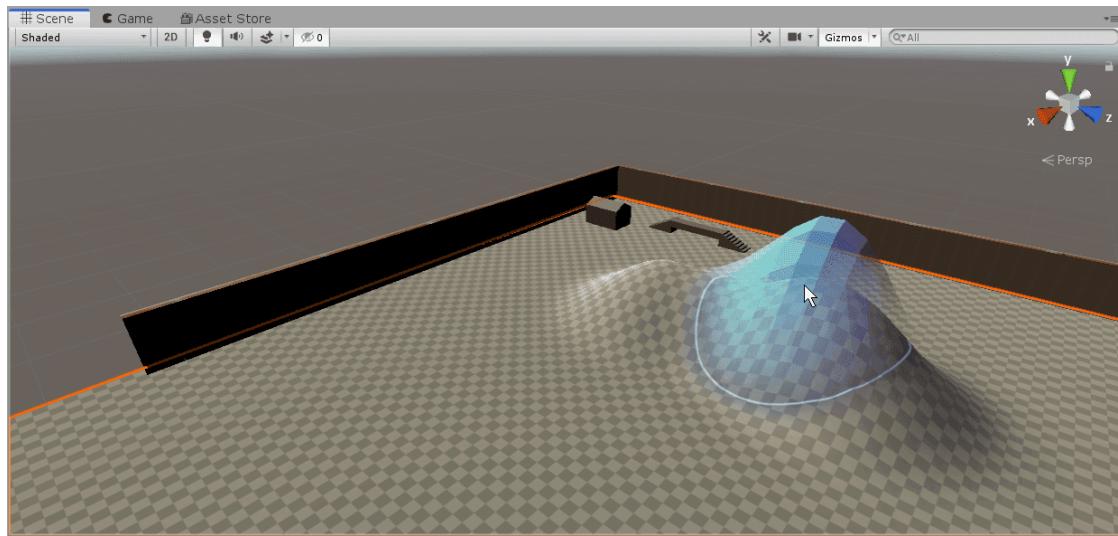
Con esa "pintura" podemos conseguir distintos efectos. Comenzaremos por usar el valor por defecto: elevar o bajar el terreno.



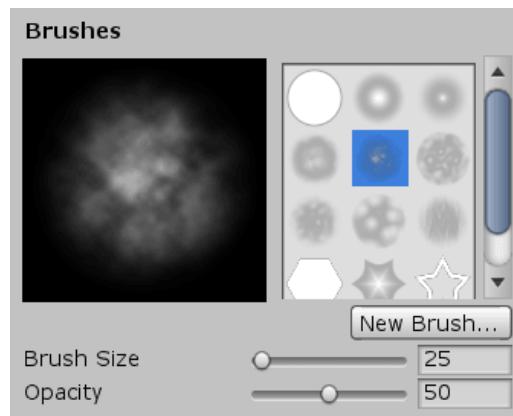
Escogeremos una brocha y veremos que al hacer clic con ella, el terreno se eleva. Si pulsamos Mayús a la vez que hacemos clic, conseguiremos rebajar su altura.



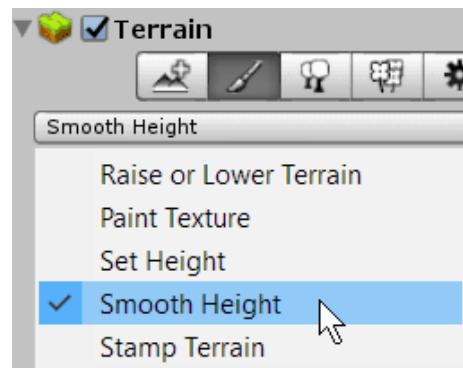
El efecto es acumulativo: si hacemos un segundo clic en el mismo sitio, la elevación será mayor:



Para conseguir distintos efectos, podemos emplear distintas brochas, así como cambiar el tamaño de la brocha o su opacidad:

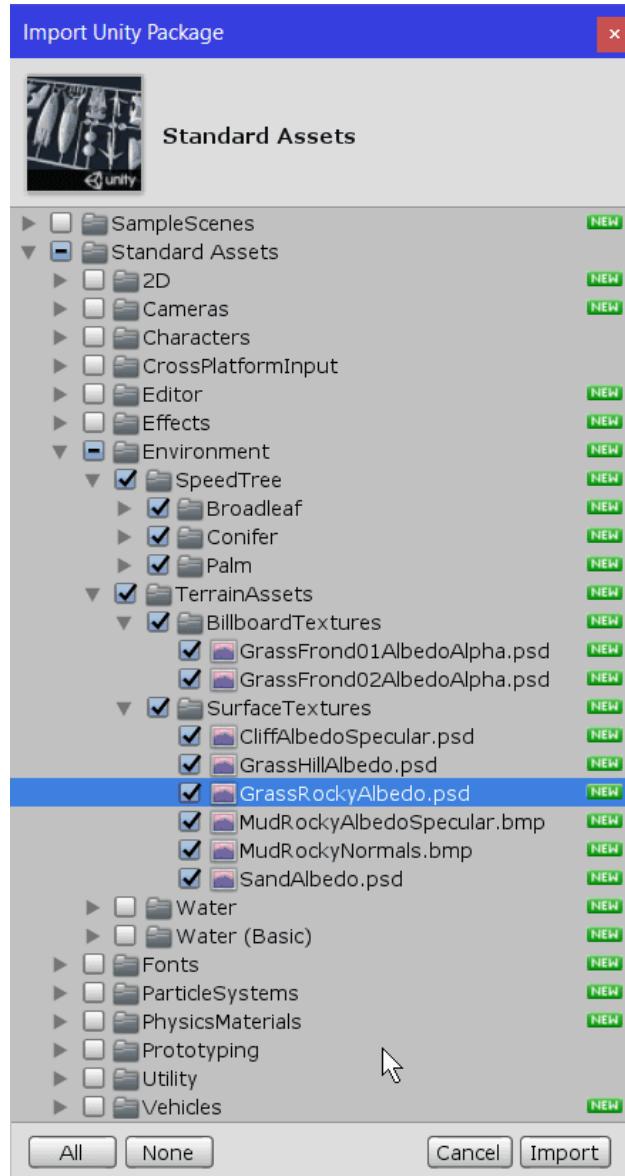


Y si, en algún momento, una zona queda demasiado escarpada, podemos cambiar a la herramienta de "suavizar altura":

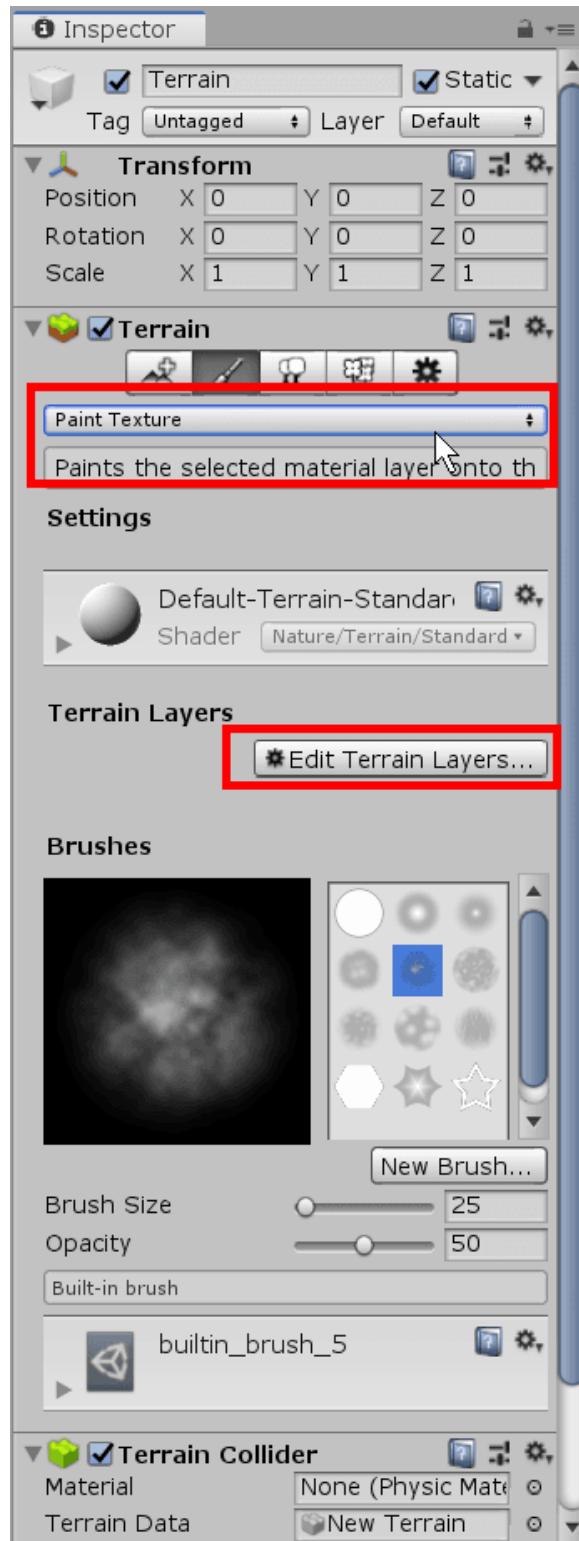


Cuando ya tengamos un relieve que nos guste, llega el momento de aplicar una textura, para que el suelo realmente parezca un suelo. Tenemos varias texturas en el paquete de "Standard Assets"

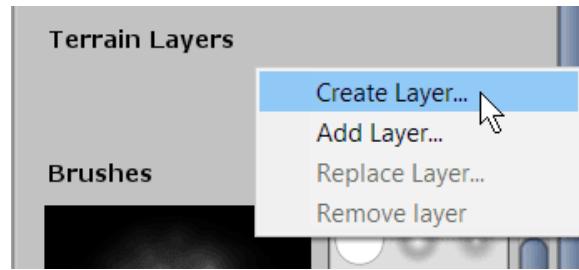
que nos pueden servir, dentro de la categoría "TerrainAssets". De paso, podemos añadir también los árboles, de la categoría "SpeedTree", que usaremos en el siguiente apartado:



Para aplicar una cierta textura a nuestro terreno, que ahora mismo parece un "tablero de ajedrez", deberemos cambiar al modo de "pintar con texturas" ("Paint texture") y luego "editar las capas del terreno" ("Edit terrain layers"):



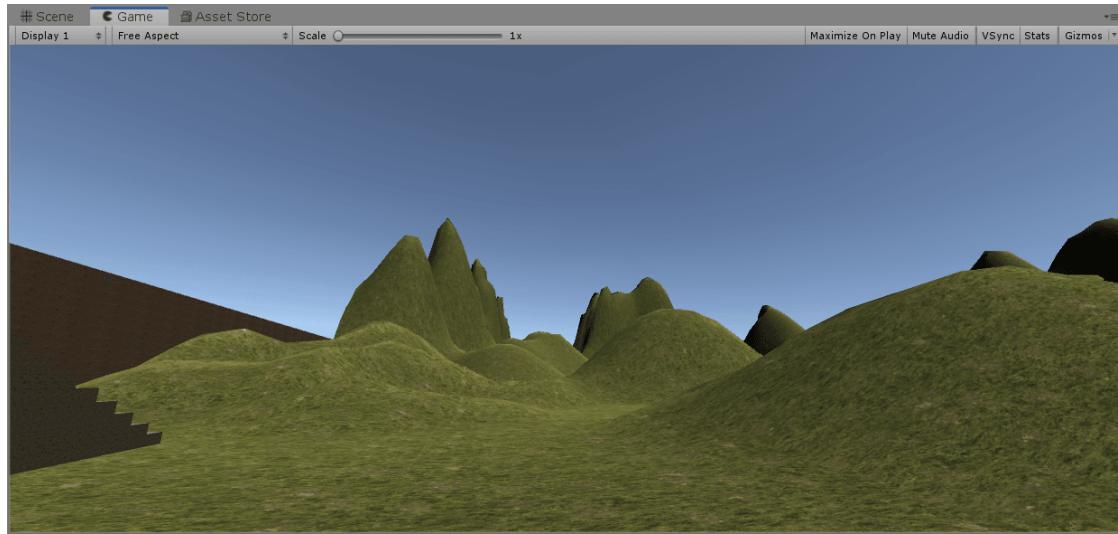
El primer paso será crear una capa:



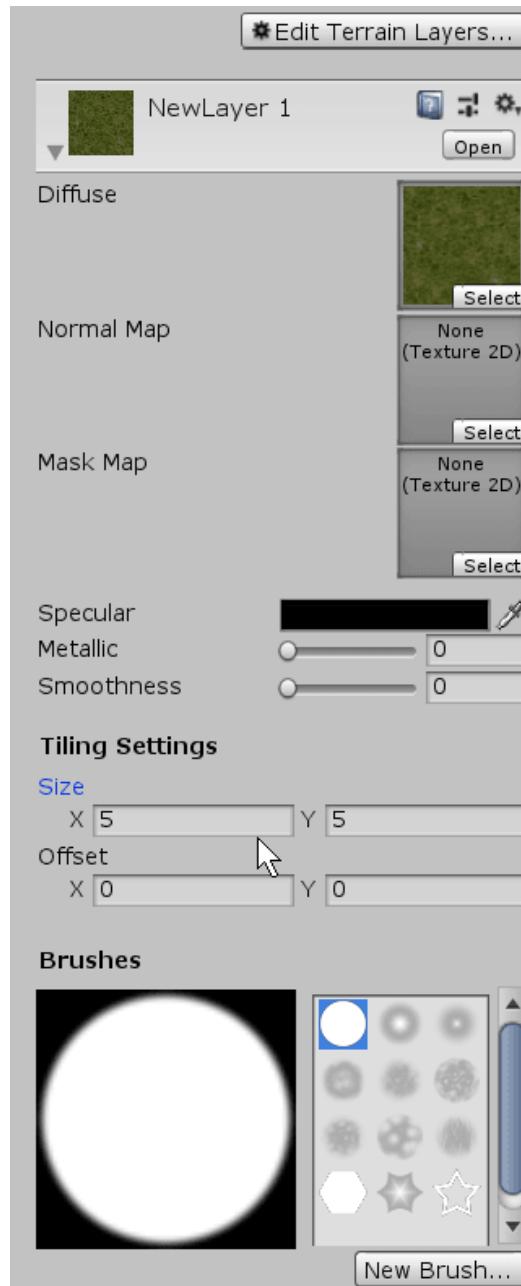
y se nos pedirá que escojamos una textura. Dentro de los "Standard Assets" tenemos varias texturas que nos puedes servir, como una llamada "GrassHillAlbedo":



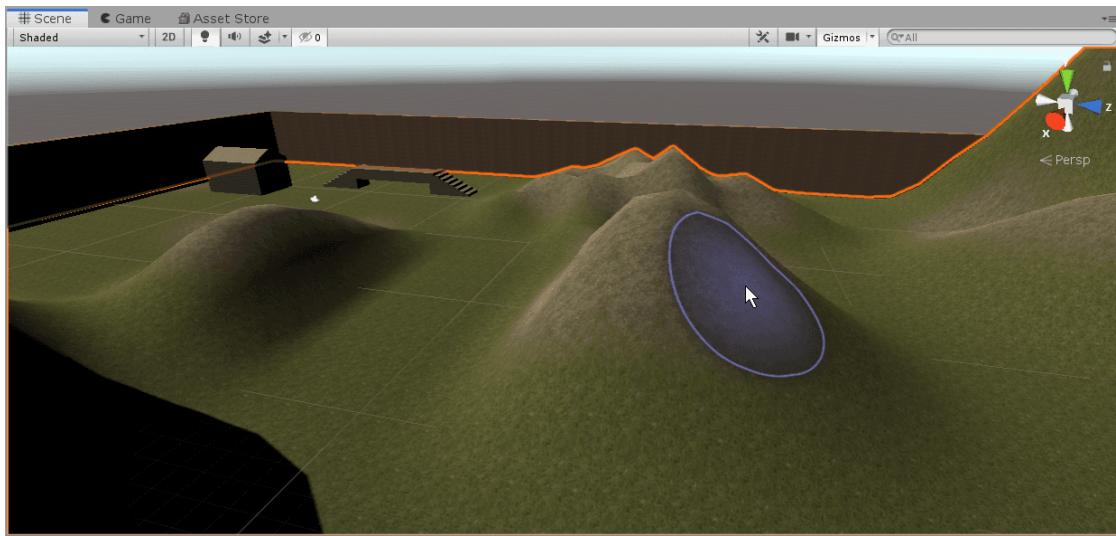
La textura se aplicará automáticamente a todo nuestro terreno. Si no fuera así, podemos escoger una brocha que sea bastante opaca (como la primera), darle un tamaño muy grande y usar para "pintar" el terreno con esa textura. En ambos casos, obtendríamos algo como:



Si nos parece que la textura queda demasiado pequeña (lo ideal es probarlo lanzando el juego), podemos cambiar su tamaño:



Y si queremos que el suelo no sea totalmente uniforme, podemos crear una nueva capa con otra textura. Por ejemplo, esta vez podemos usar otra textura que hay dentro de "Standard Assets", llamada "GrassRockyAlbedo", más grisácea, y aplicarla a las zonas más altas de las montañas o donde queramos marcar "caminos", ajustando el tamaño y la opacidad de la brocha que queramos usar:



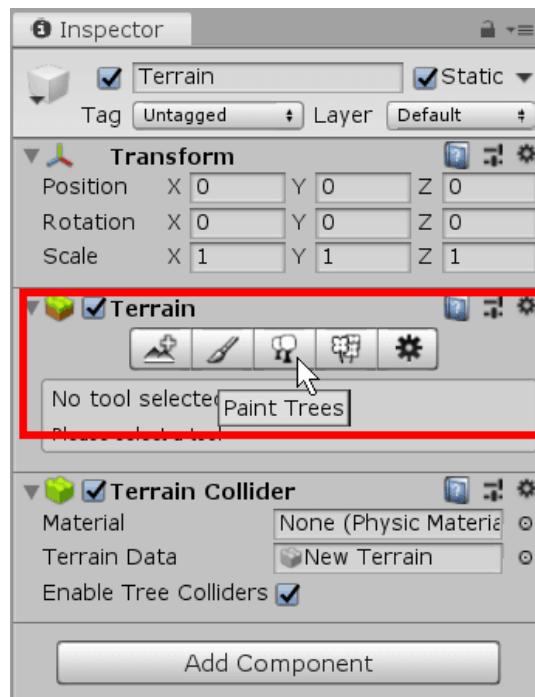
Una tercera textura incluida en "Standard Assets", más oscura, con apariencia de barro, tiene el nombre de "MudRockyAlbedo".

**Ejercicio propuesto 4.5.1:** Añade elevaciones y texturas a tu terreno.

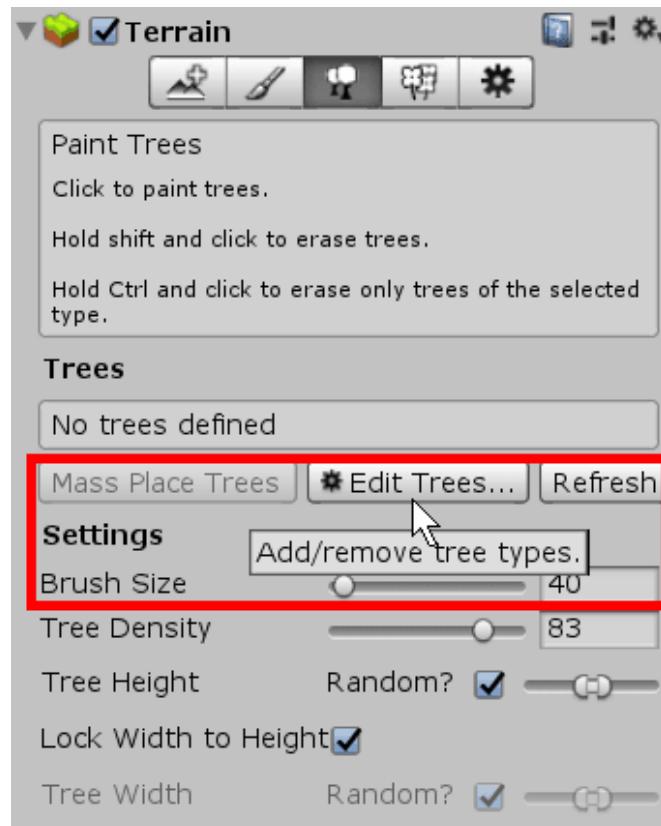
## 4.6. Árboles e hierba

Los "Standard Assets" también incluyen árboles que podríamos añadir a nuestro terreno. Los pasos para incluirlos serán:

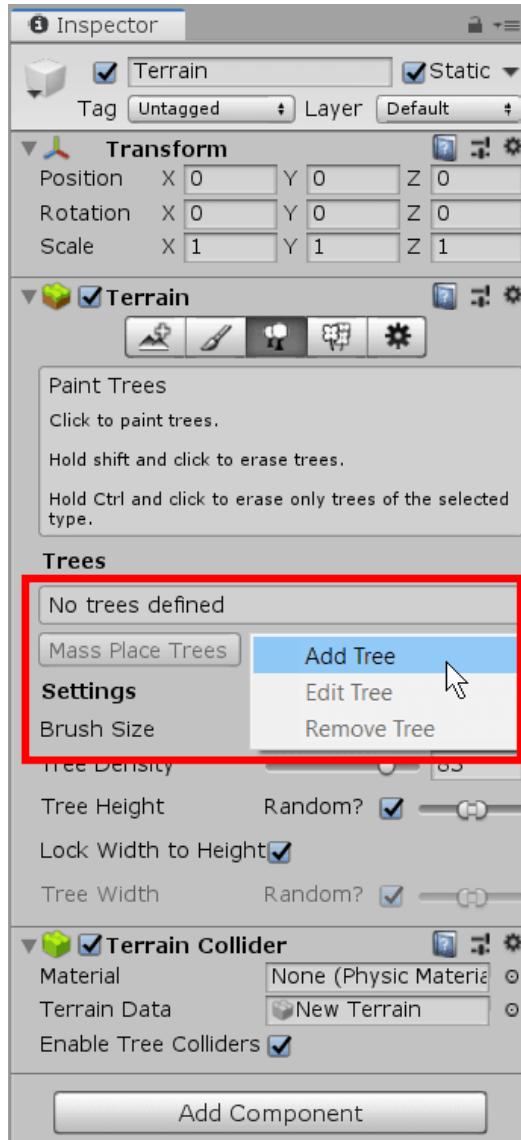
- En primer lugar, seleccionar el terreno y, en el inspector, usar la opción "pintar árboles" ("Paint trees"):



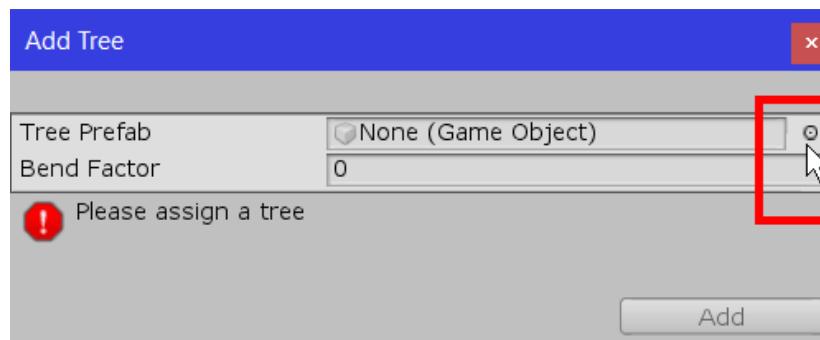
- Aparecerá un botón "Edit tree":



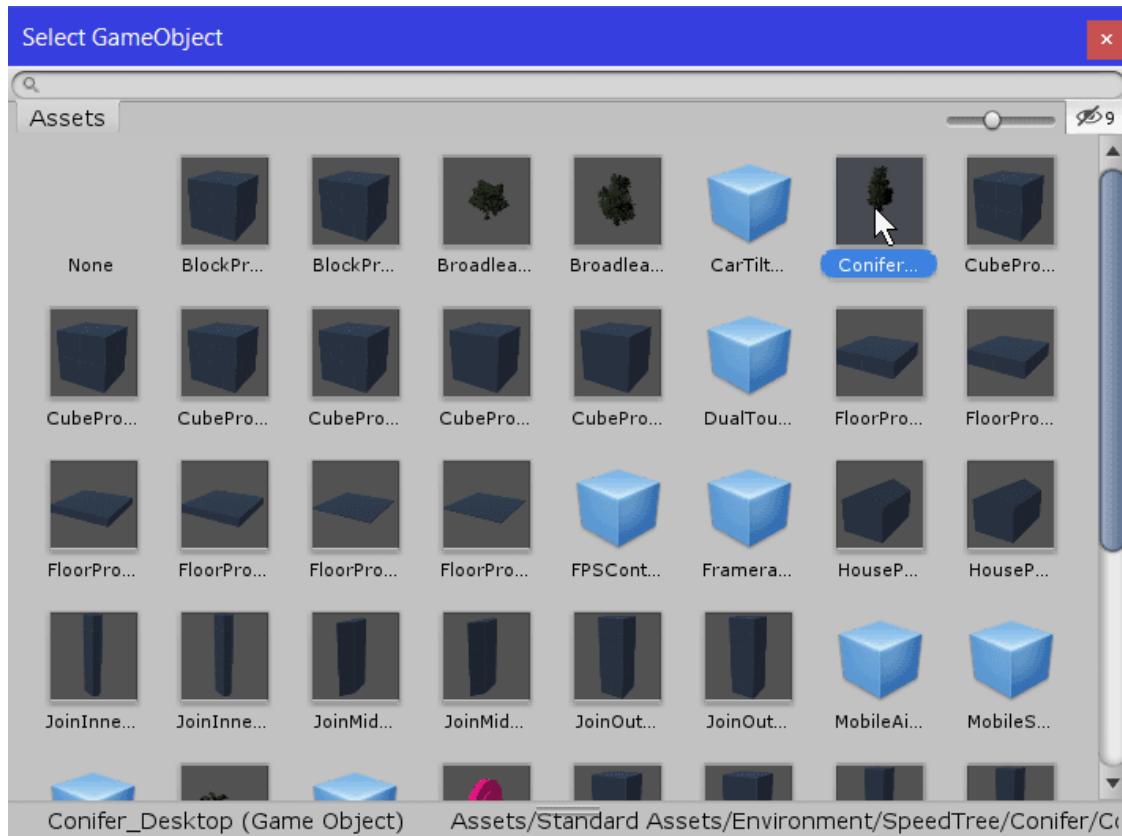
- Que nos dará acceso a la opción de añadir un árbol ("Edit tree"=:



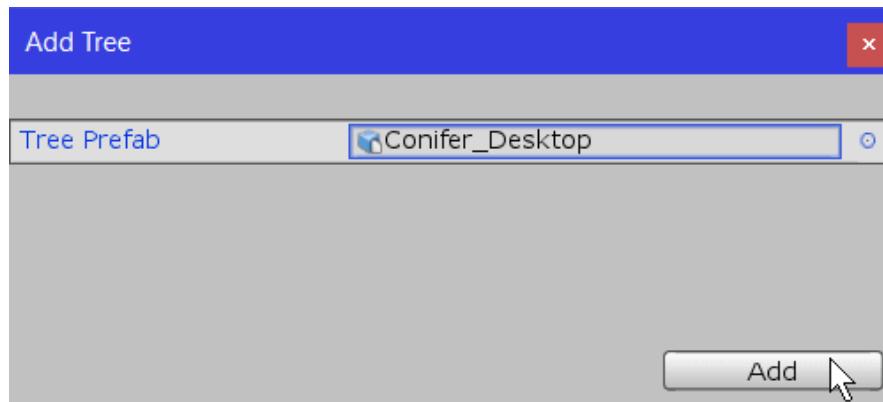
- Pero el selector aparecerá vacío, deberemos hacer clic en su esquina superior izquierda para buscar tipos de árboles entre los objetos que hemos importado:



- Y se nos mostrarán los modelos 3D que tenemos disponibles, y entre ellos debería haber un "Broadleaf/Desktop", un "Conifer/Desktop" y un "Palm/Desktop":



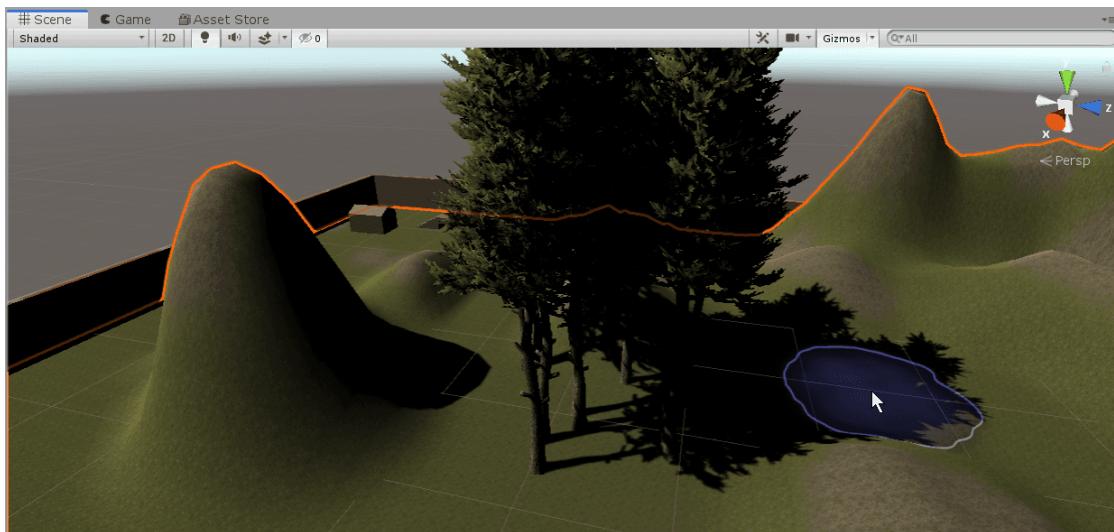
- Deberemos escoger uno de ellos:



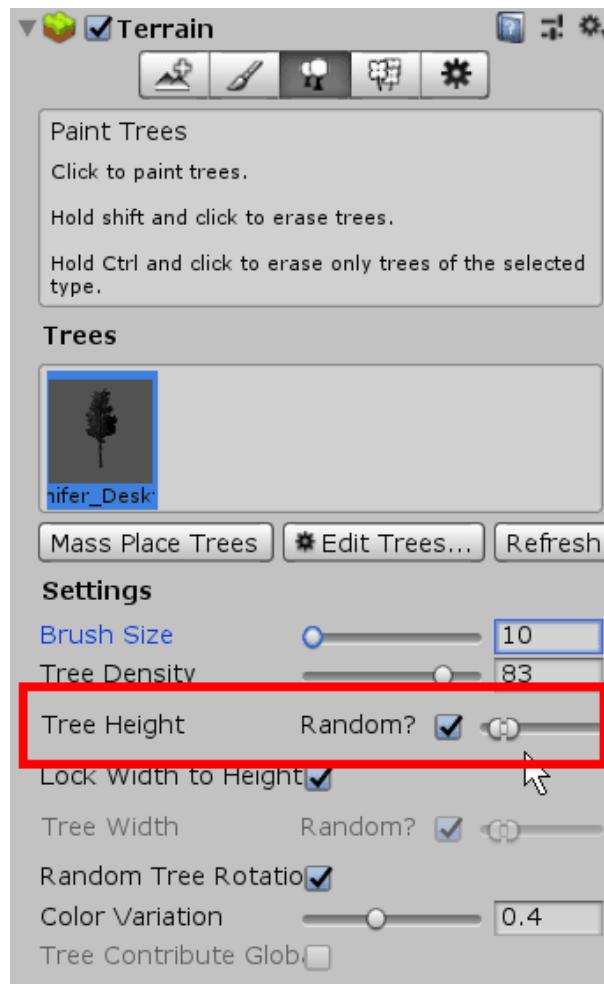
- Y entonces ya aparecerá en el inspector:



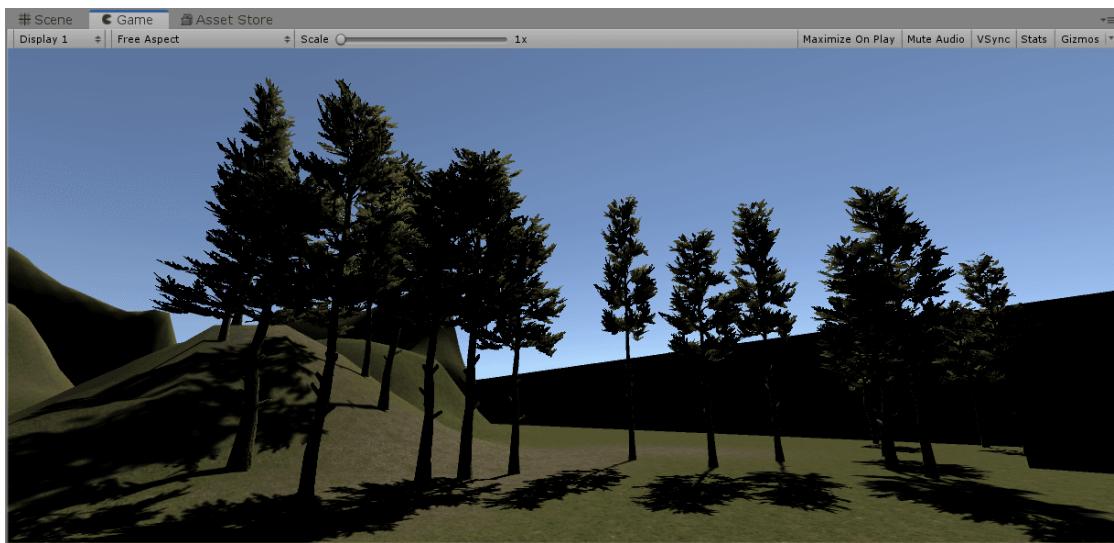
- Y podremos hacer clic en el terreno para dibujar un bloque de árboles tan grande como indique nuestra brocha:



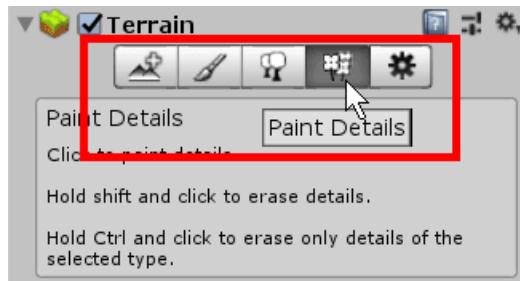
- Es posible que los árboles sean demasiado altos. en ese caso podemos ajustar su altura (que, en principio, será al azar entre un valor máximo y uno mínimo):



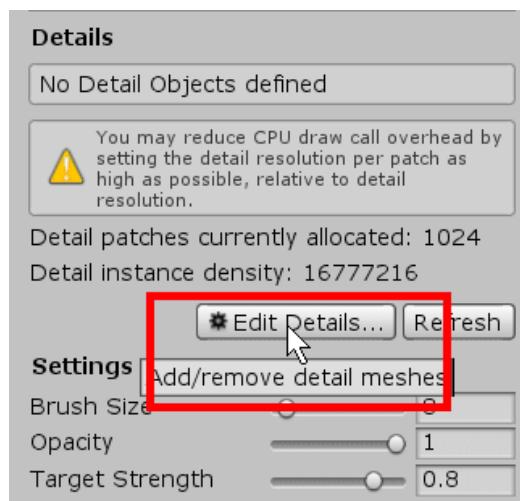
- Y usar May+ clic para borrar los árboles existentes, o clic para añadir árboles nuevos:



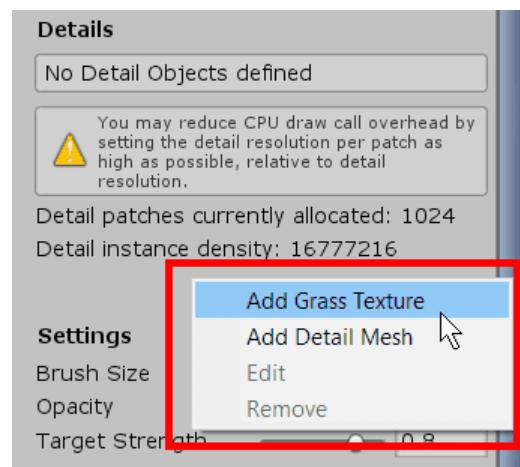
El procedimiento para añadir hierba sería (hasta cierto punto) parecido: comenzamos por ir a la opción de "pintar detalles" ("Paint details") en el terreno:



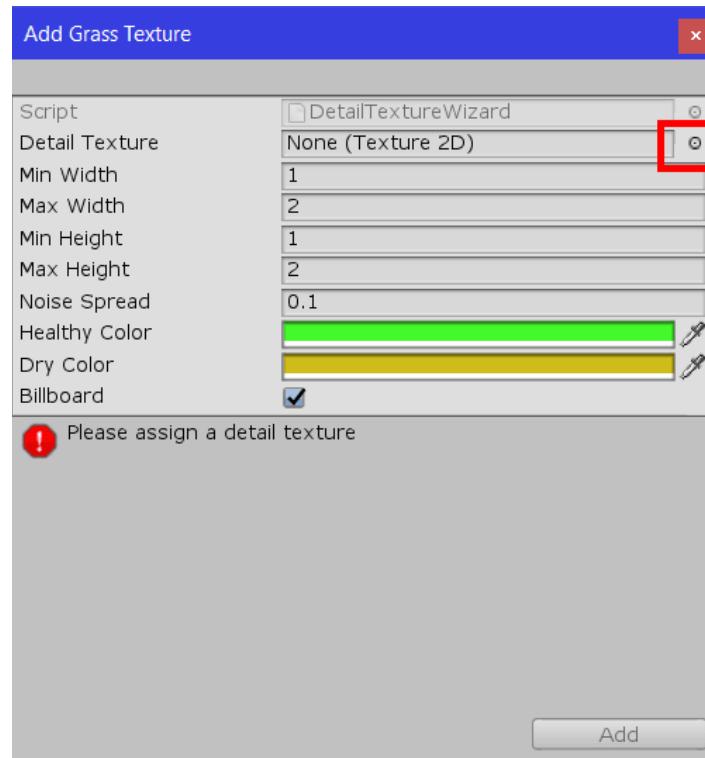
pulsaremos el botón "Add details":



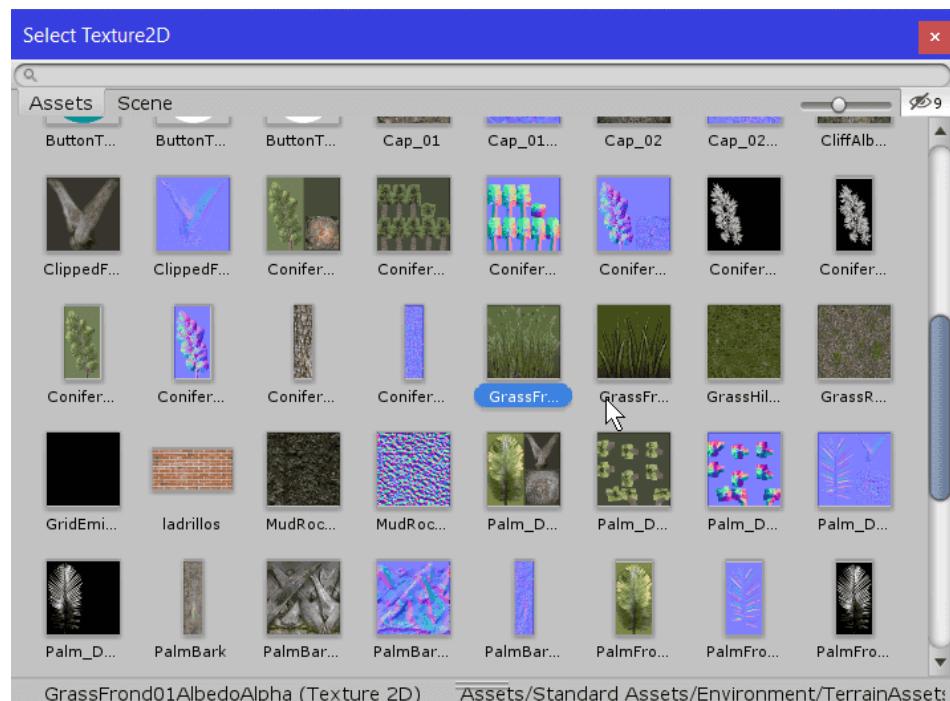
y aparecerá la opción de añadir una textura de hierba ("Add grass texture"):



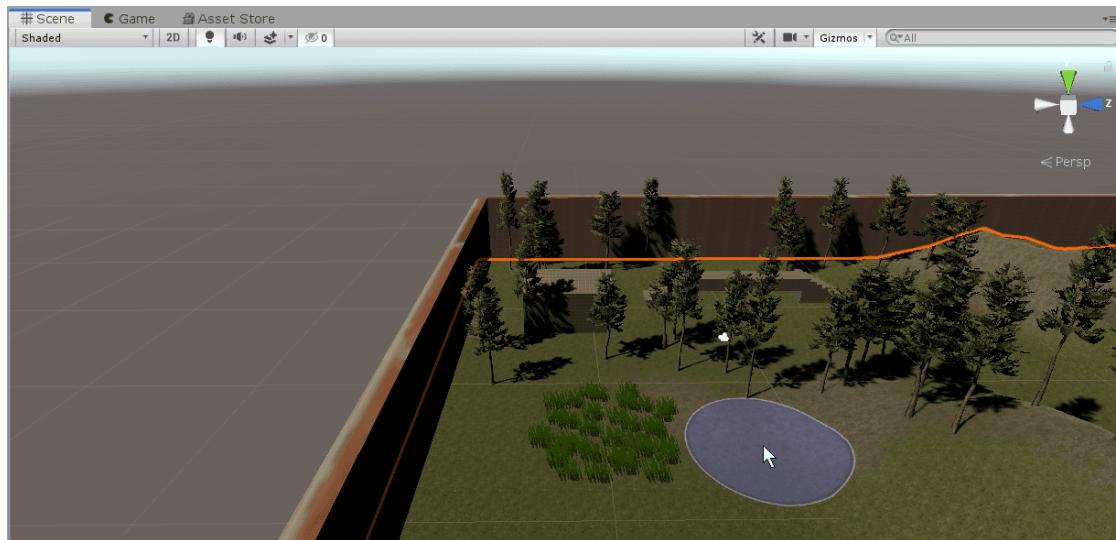
En la siguiente ventana se nos pedirá que escojamos una textura 2d:



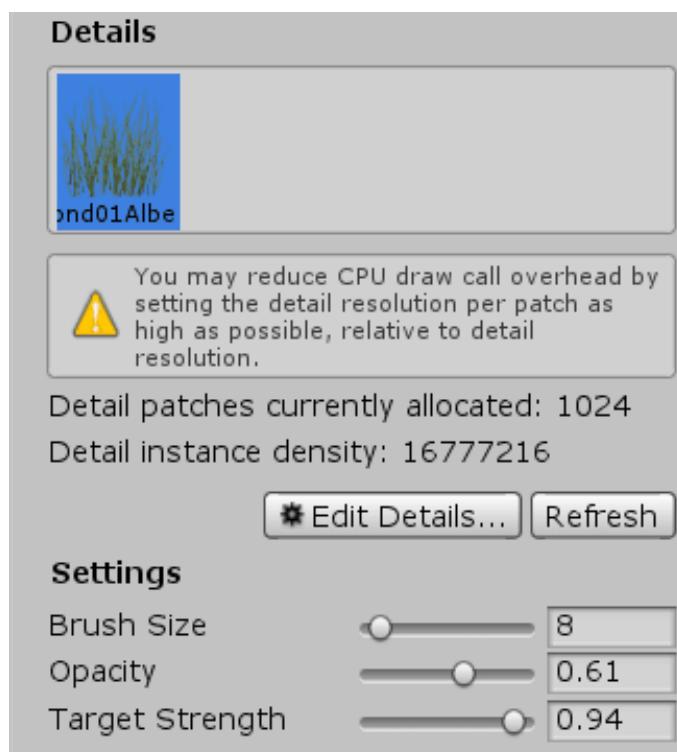
Y, nuevamente, tendremos un par disponibles entre los "Standard Assets"



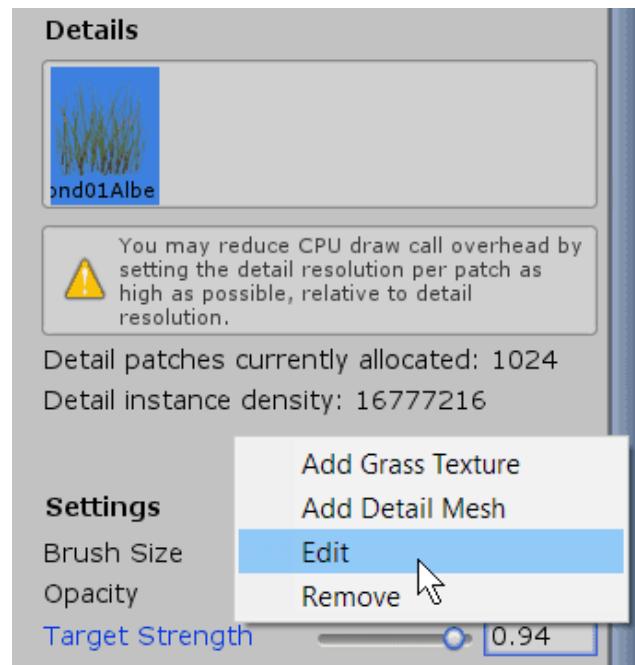
A partir de ese momento, ya podremos hacer clic para dibujar zonas de hierba:



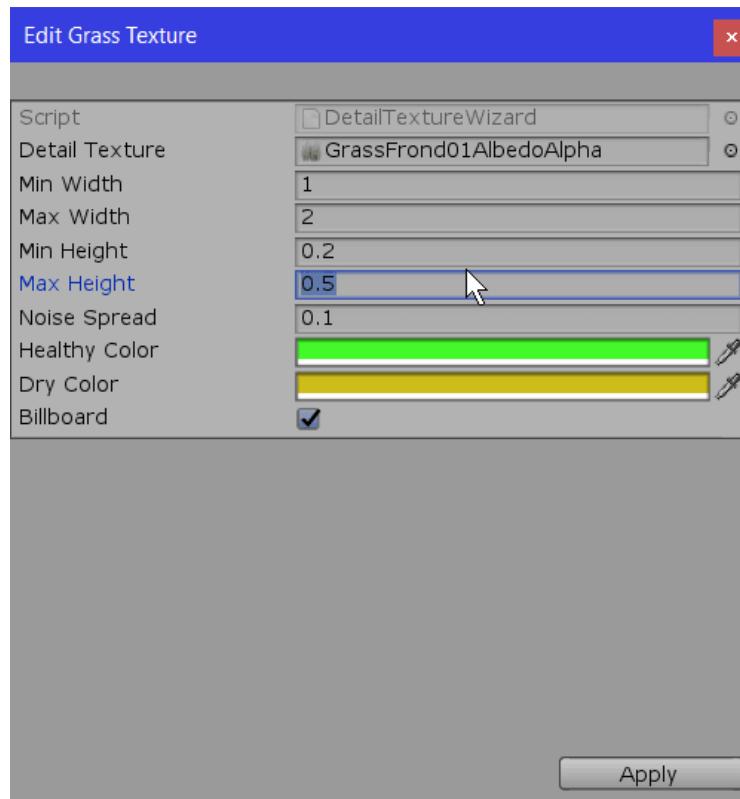
Nuevamente, deberemos probar el juego para asegurarnos de que la hierba no sea demasiado alta (que es posible). Si fuera el caso, pulsaríamos el botón "Edit details":



y escogeríamos la opción "Edit":



para cambiar la altura máxima y mínima (Max Height y Min Height):



**Ejercicio propuesto 4.6.1:** Añade hierba y árboles a tu terreno.

## 4.7. Añadiendo un arma

Hacer un "shooter" implica que nuestro personaje debe disparar, aunque no sean balas sino pintura. Por eso, necesitaremos añadir algún arma, pero no hay ninguna dentro de los "standard assets".

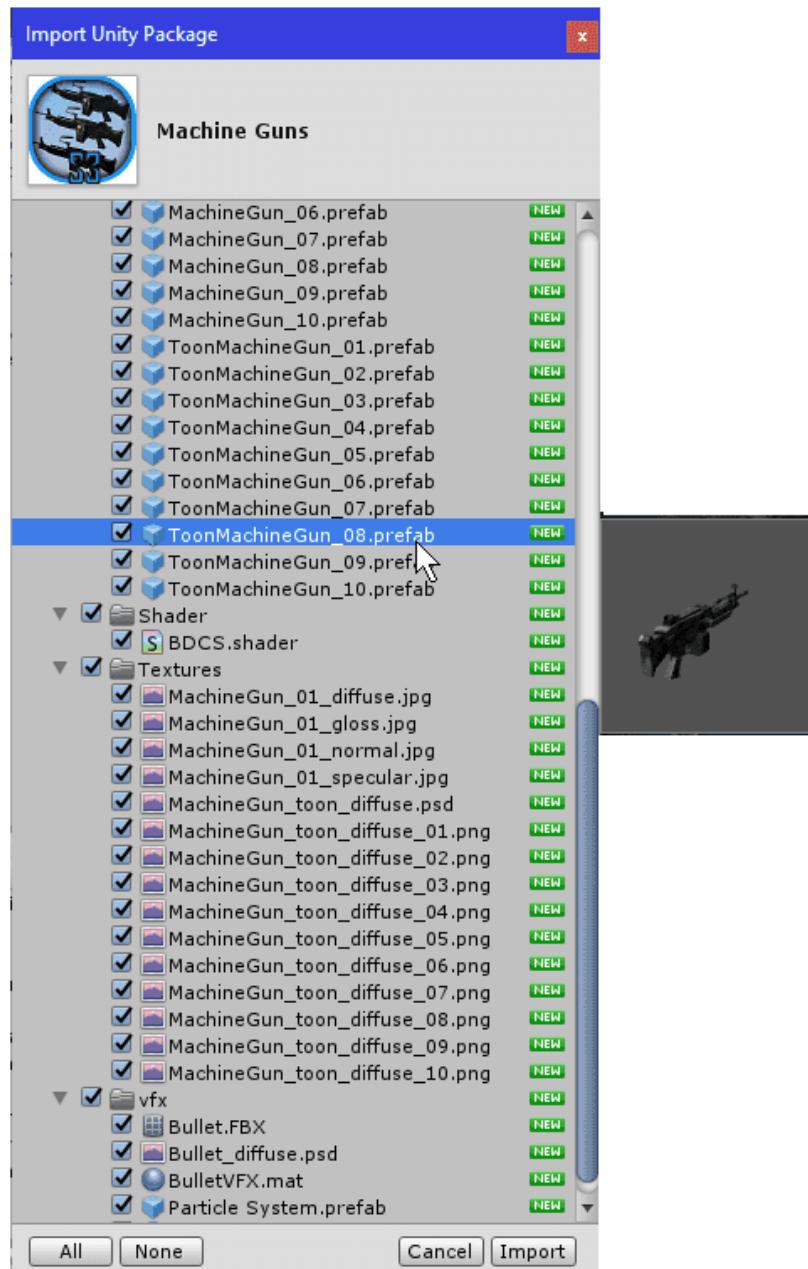
Podemos ir a la Asset Store, buscar por "Gun" y escoger sólo las que sean gratis ("free"). Por ejemplo, hay un kit llamado "Machine guns" recopilado por "Assets3d" que nos puede servir.

The screenshot shows the Unity Asset Store interface with a search bar containing 'gun'. Below the search bar, there are filters: 'Sort by Relevance' and 'View Results 24'. The main area displays 1-24 of 142 results. The first result is 'Machine Guns' by ASSETS3D, which is free. Other results include 'Flare Gun' by ROKAY3D, 'Submachine Gun' by INFERNO, 'Rit Gun' by MSTK STUDIO, 'Paint Gun' by YEDP, and 'GUN' by RKD. On the right side, there are filter panels for 'Refine by' categories like 3D, 2D, Audio, Templates, Tools, and 'Pricing' filters for Free Assets (selected) and Plus / Pro.

Veremos que ocupa cerca de 18 MB:

The screenshot shows the product page for 'Machine Guns' by ASSETS3D. The page features a large image of various machine gun models. Key details include: 'Machine Guns' (FREE), 19 user reviews (4.5 stars), a 'Download' button, and a 'Free Machine Guns' section with a link to a 'Webdemo'. The page also lists 'ASSET3D II Textures' and '3 Animations'. Under 'Features', it highlights: One draw call per weapon, Low poly model: 975 tris, Hand sculpted and baked to low poly, and Animations created in Unity - you can create your own animations. There's also a 'Show More' link. At the bottom, there's a 'Package contents' section showing 18.1 MB total file size and 71 files, with links for 'Share' and 'Add to List'.

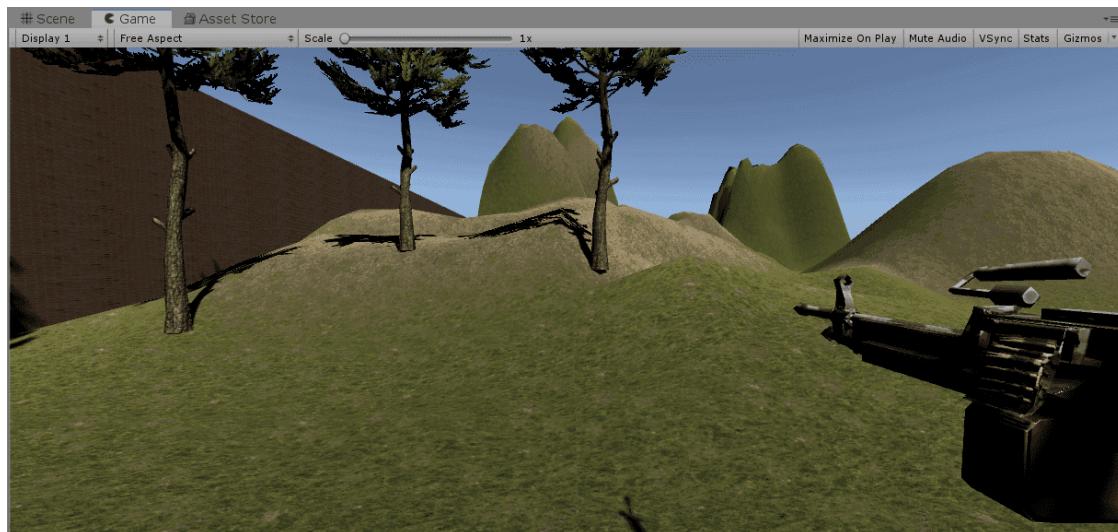
Y que tiene varios "prefab" para distintos tipos de armas automáticas:



Aun así, en esta ocasión quizá no se comporte bien si tratamos de importar sólo el "prefab" que nos interese, y éste no se vea en la escena. Si eso ocurre, tenemos la alternativa de importar todo el paquete, y entonces sí podremos arrastrar un arma a la escena y verla:

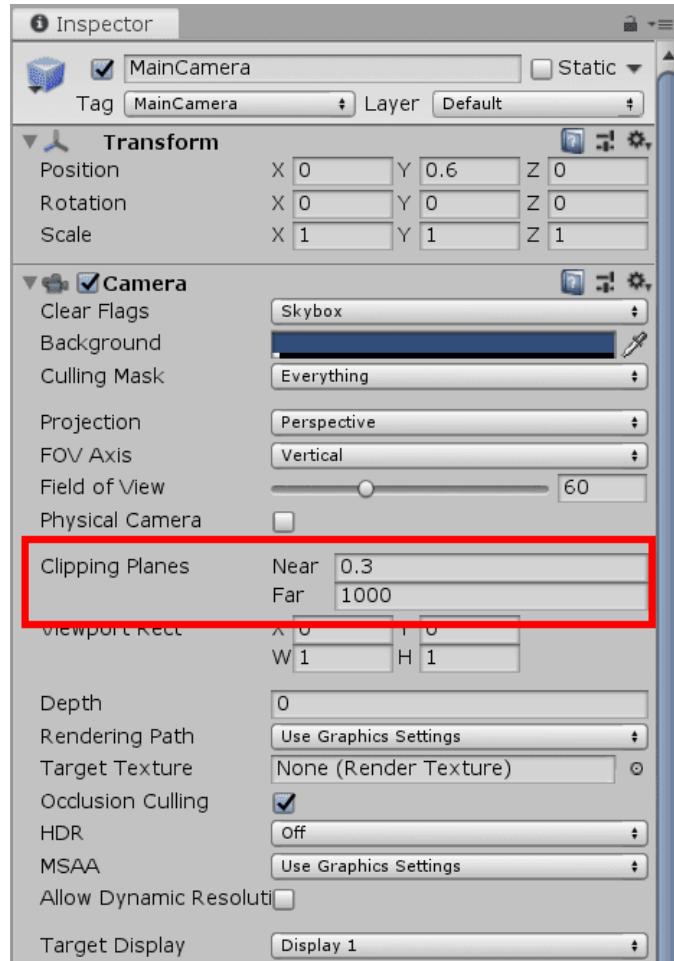


Pero no nos basta con que el arma esté "por ahí". Para que nuestro personaje parezca llevar el arma, deberemos añadir ésta como "hija" del personaje (o incluso de la cámara), y ajustar su rotación (posiblemente 90º en el eje Y) y su posición hasta conseguir un efecto que nos guste.

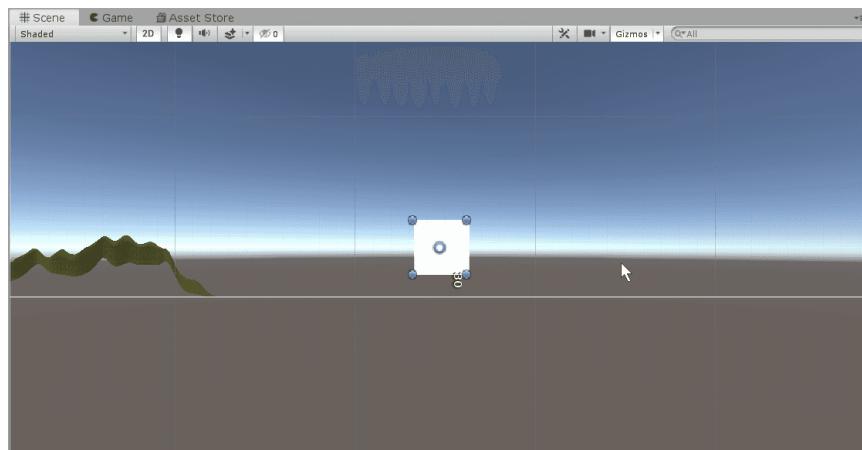


Si la situamos como "hija" del personaje, se moverá a la vez que nos movamos nosotros; si es "hija" de la cámara, además conseguiremos que apunte hacia arriba y hacia abajo cuando utilicemos el ratón para subir y bajar el punto de vista, lo que puede ser un efecto deseable.

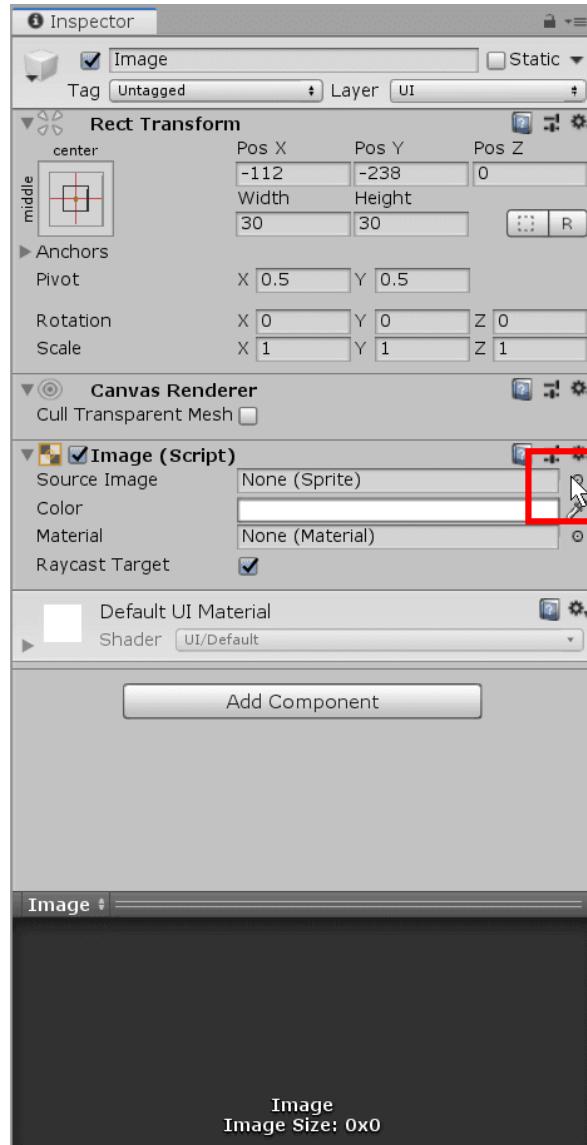
Desde la vista de juego deberemos asegurarnos de que el arma es visible, ya sea cambiando su posición y rotación o incluso afinando los parámetros de la cámara, como los "clipping planes", para que el plano "near" sea más cercano al jugador (por ejemplo, de 0.3 a 0.01).



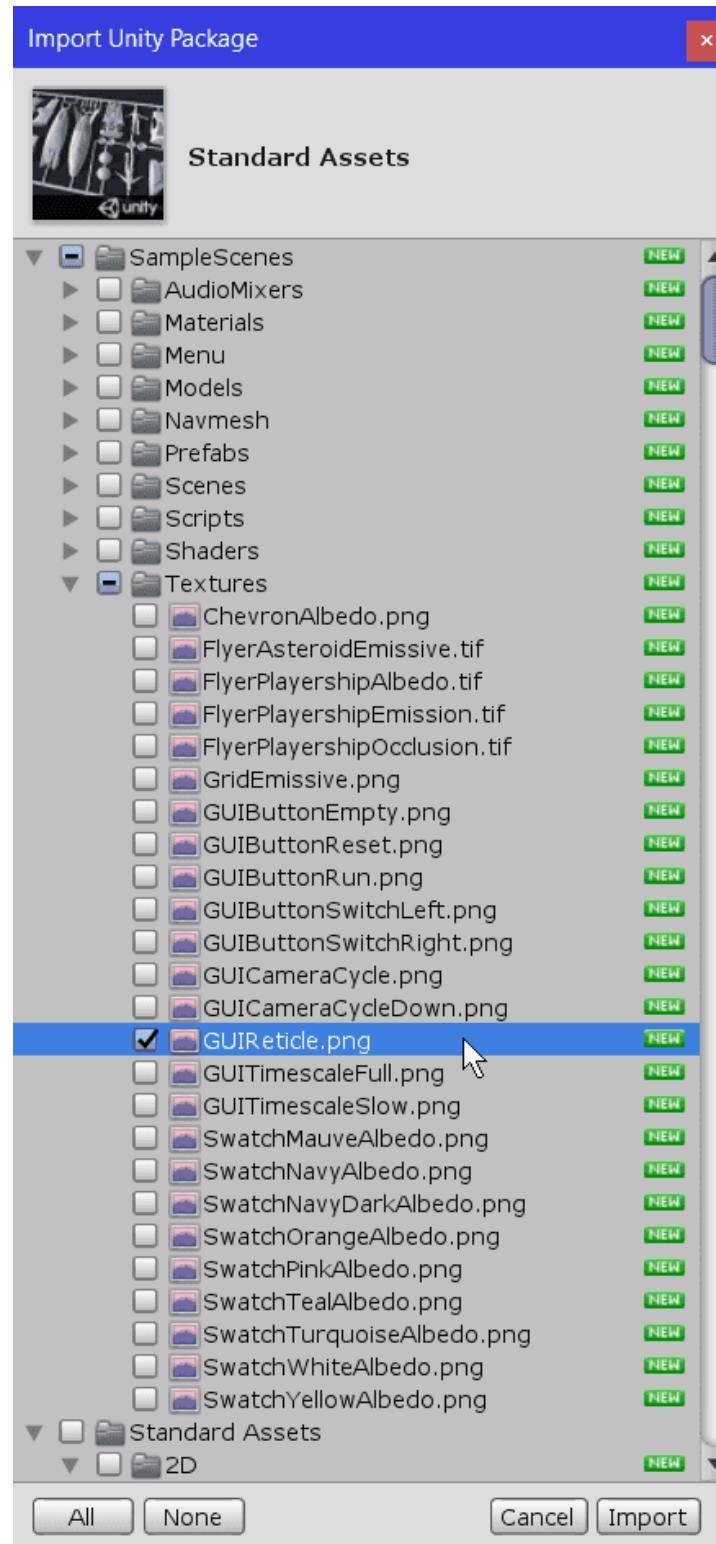
Generalmente el efecto será más vistoso si tenemos un "punto de mira" de la cámara ("crosshair" en inglés) centrado en la escena. Podemos hacerlo añadiendo una imagen, como componente del interfaz de usuario (IU: botón derecho en la jerarquía, UI, Image), ajustando su tamaño (por ejemplo, 30x30) y su alineación en la imagen (centro, centro):



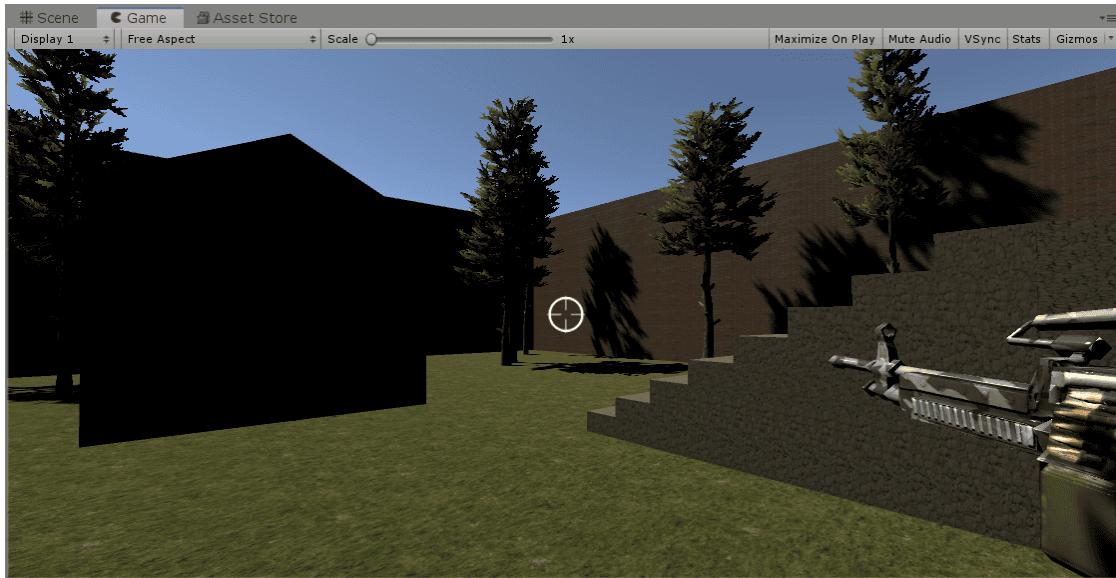
Finalmente, deberíamos cambiar la apariencia de esa imagen (que inicialmente es un cuadrado blanco) por una que sea más apropiada para apuntar.



Por ejemplo, tenemos una en los "standard assets" llamada "GUIReticle", en la categoría "Sample Scenes / Textures":



Y con ella, la pantalla de juego quedaría así:



Si queremos añadir un sonido de disparo, es algo que ya hemos hecho: buscaríamos un sonido de disparo y añadiríamos un Audio Source a nuestra arma.

**Ejercicio propuesto 4.7.1:** Añade un arma con una mirilla para apuntar.

## 4.8. Enemigos móviles que se pueden destruir

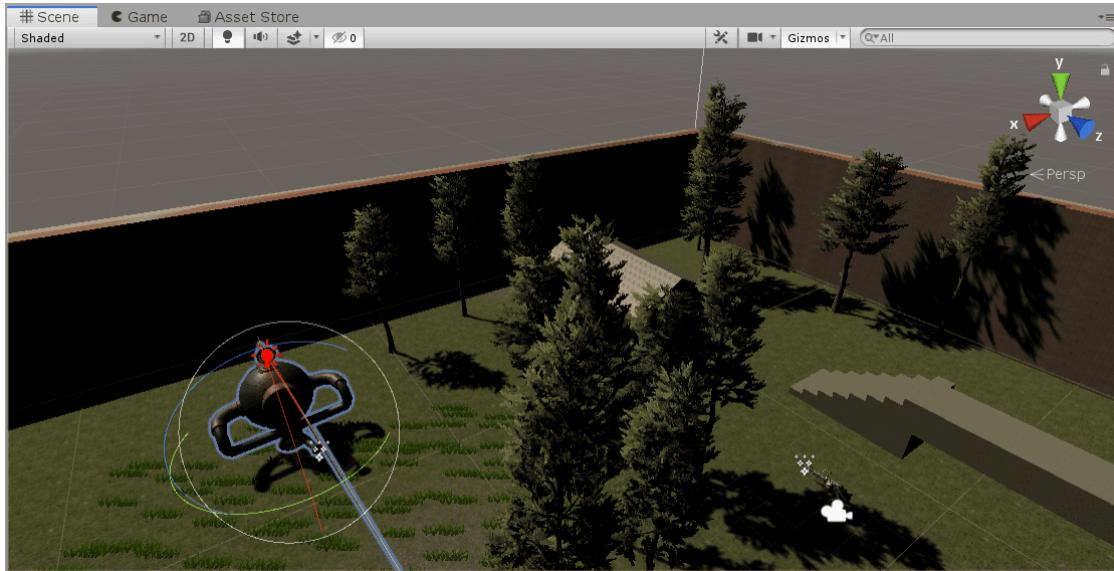
Si queremos que nuestros enemigos sean realistas, podemos recurrir nuevamente a la Asset Store para buscar algunos ya creados y que sean gratuitos. Por ejemplo, si buscamos "enemy" restringiendo a recursos que sean gratis, podemos encontrar packs como éste:

Asset	Creator	Rating	Status
Enemy Robots	WATCHFINDDOMEDIA	★★★★★ (7)	FREE
Enemy Turrets	OLEG JARICOV	★★★★☆ (41)	FREE

No es fácil encontrar assets de calidad, gratuitos y que además incluyan animaciones. Por ejemplo, estos que hemos escogido no contienen animaciones, pero sí podremos moverlos

nosotros por la escena usando "waypoints" como ya habíamos hecho (y más adelante les añadiremos algo de "inteligencia").

En primer lugar, arrastramos un robot a la escena y lo colocamos donde nos interese:



Como punto de partida, podemos añadir a continuación una lógica de waypoints idéntica a la que usamos en el tema 4 (el contacto con 3D en Unity):

```
public class Enemigo : MonoBehaviour
{
    [SerializeField] Transform[] wayPoints;
    Vector3 siguientePosicion;
    byte numeroSiguientePosicion;
    float distanciaCambio = 0.2f;
    float velocidad = 2;

    private void Start()
    {
        siguientePosicion = wayPoints[0].position;
        numeroSiguientePosicion = 0;
    }

    private void Update()
    {
        transform.position = Vector3.MoveTowards(
            transform.position,
            siguientePosicion,
            velocidad * Time.deltaTime);

        if (Vector3.Distance(transform.position,
            siguientePosicion) < distanciaCambio)
        {
            numeroSiguientePosicion++;
            if (numeroSiguientePosicion >= wayPoints.Length)
                numeroSiguientePosicion = 0;
            siguientePosicion = wayPoints[numeroSiguientePosicion].position;
        }
    }
}
```

```

        }
    }
}
```

Como pequeña mejora, podemos hacer que el enemigo no sólo se mueva, sino que además mire en otra dirección. Eso lo podemos conseguir con transform.LookAt:

```

siguientePosicion = ...
transform.LookAt(siguientePosicion);
```

Eso sí, este robot concreto está diseñado de forma que su "laser" apunta hacia "atrás", así que podemos hacer que mire en dirección contraria:

```

transform.LookAt(siguientePosicion);
transform.Rotate(Vector3.up, 180);
```

(Otra alternativa más eficiente sería rotar 180º alrededor del eje Y algunos de los componentes del prefab, de forma visual o con ayuda del inspector).

Para disparar en una cierta dirección, podemos usar "raycasting", y ver si nuestro disparo impacta con el enemigo. En esta ocasión no será un Raycast2D como habíamos usado en el juego de plataformas, sino un RayCast (en 3D). Un ejemplo podría ser:

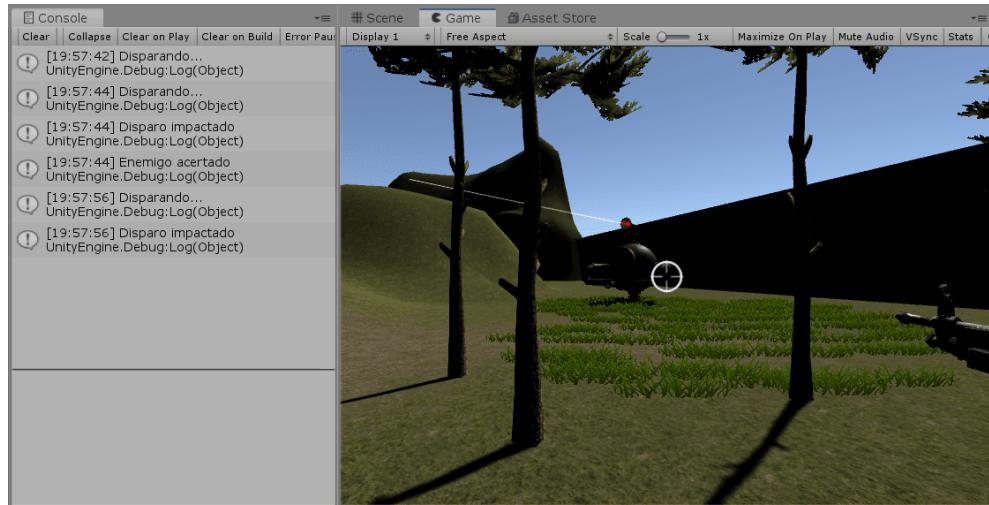
```

public class Jugador : MonoBehaviour
{
    [SerializeField] Camera camara;

    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Debug.Log("Disparando...");
            float distanciaMaxima = 100;
            RaycastHit hit;
            bool impactado = Physics.Raycast(camara.transform.position,
                camara.transform.forward, out hit, distanciaMaxima);

            if (impactado)
            {
                Debug.Log("Disparo impactado");
                if (hit.collider.CompareTag("Enemigo"))
                {
                    Debug.Log("Enemigo acertado");
                }
            }
        }
    }
}
```

Para que lo anterior funcione, es necesario añadir un "collider" al enemigo (por ejemplo, un SphereCollider en la parte principal de su cuerpo) y un "tag" adecuado.

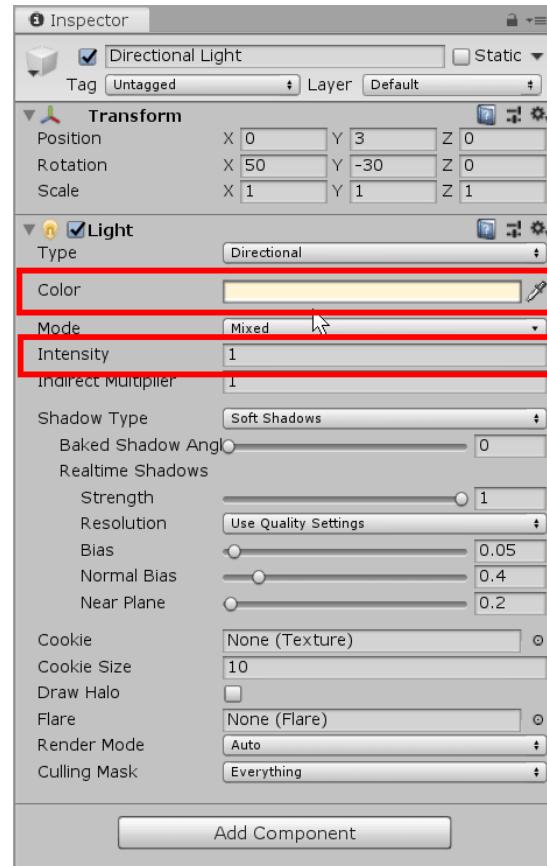


Este esqueleto se podría mejorar para que cada disparo acertado restase algo de "salud" o "energía" al enemigo, hasta que en cierto punto se destruyera o quedase inmovilizado. También se puede añadir, de forma parecida, la posibilidad de que el enemigo nos dispare a nosotros.

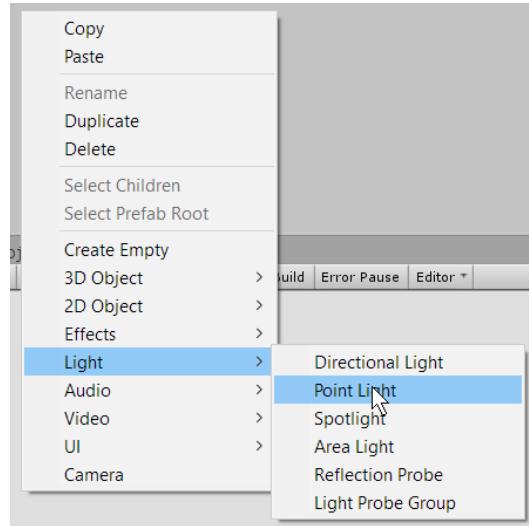
**Ejercicio propuesto 4.8.1:** Añade un enemigo móvil y haz que tus disparos lo puedan destruir.

## 4.9. Iluminación

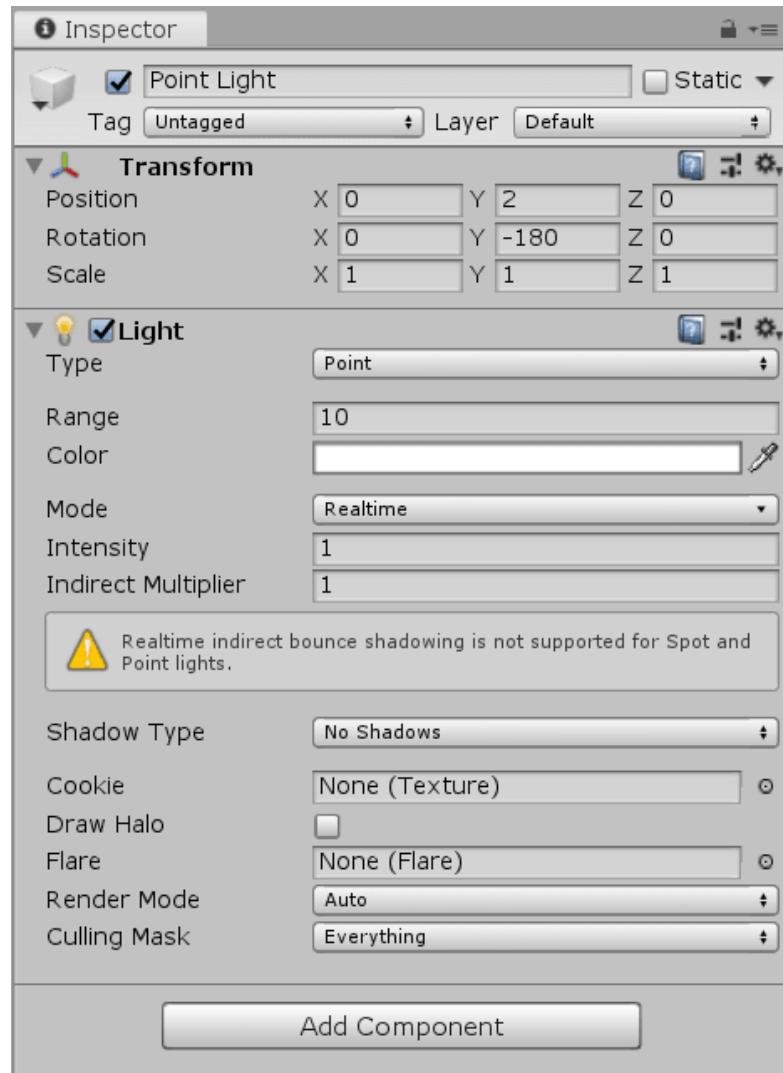
Si queremos dar un aspecto más tétrico al juego, podemos oscurecer la iluminación global. Lo podemos conseguir cambiando el color y/o la intensidad del componente "directional light" que tenemos en la escena.



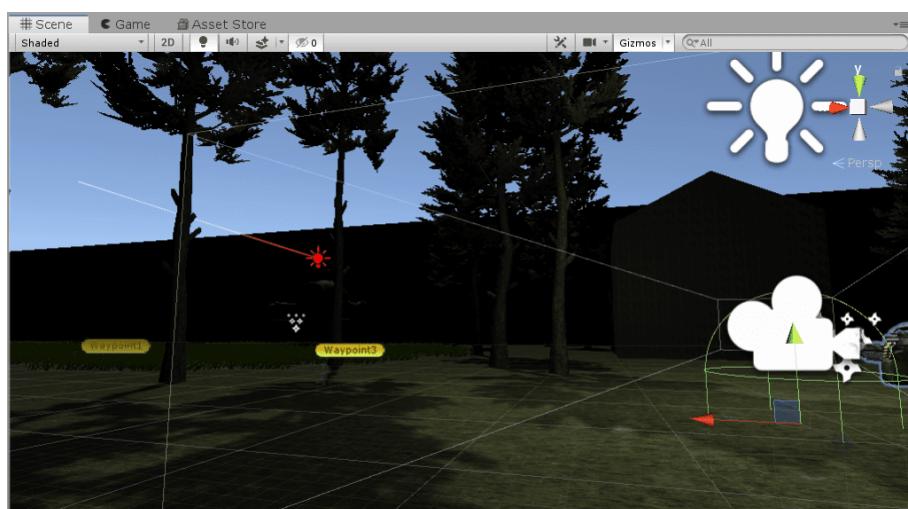
Además, podemos añadir una luz puntual que acompañe a nuestro jugador ("point light"). Lo haremos con el botón derecho en la jerarquía, Create / Light / Point light:



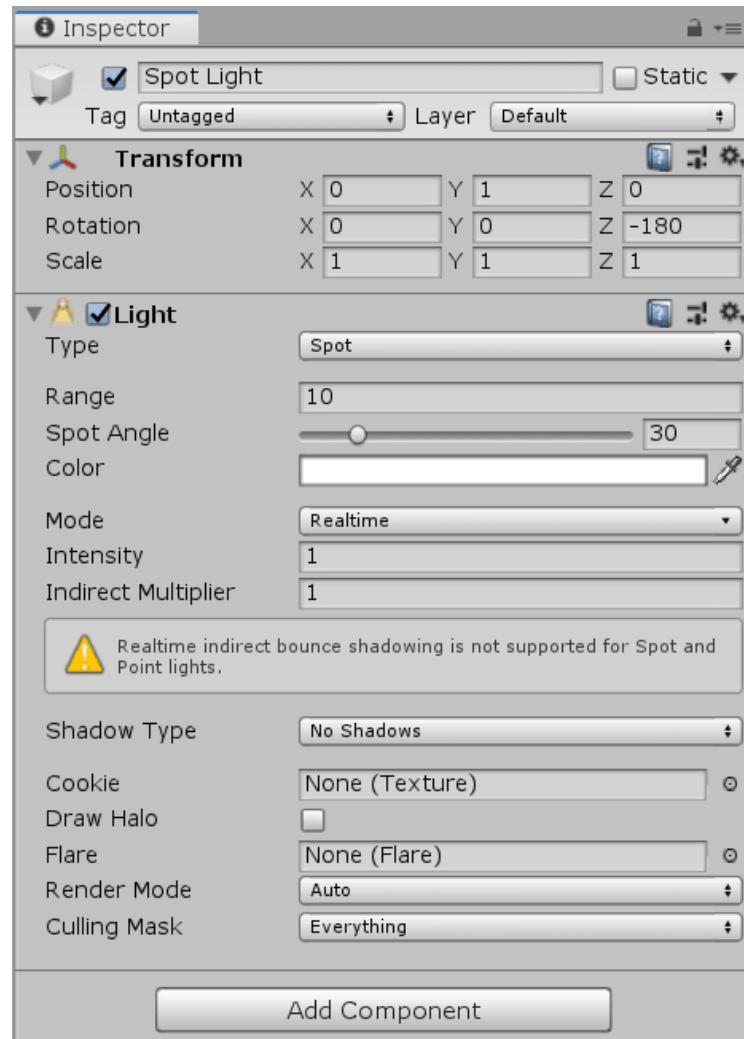
Y la podríamos colocar encima del personaje, como "hija" de éste y con coordenadas como (0,2,0):



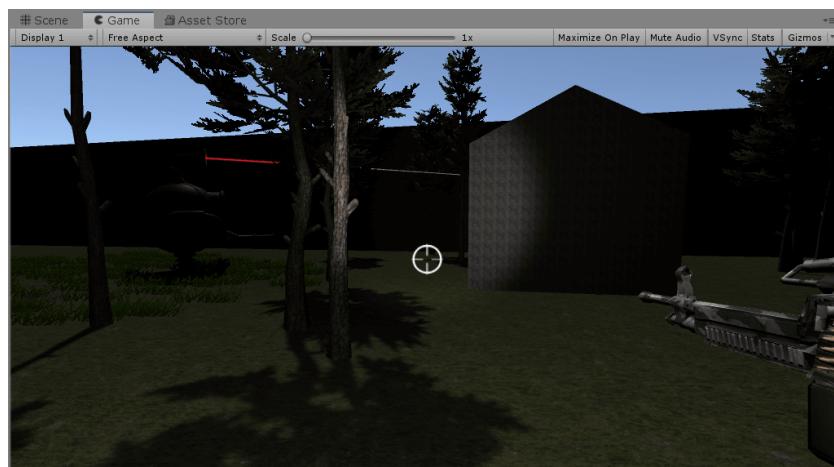
Y el resultado sería algo como:



Como alternativa, si queremos imitar un foco de luz, como una linterna, podemos usar una "spot light", cuyo haz de luz tiene la apariencia de un cono.



Y el resultado ahora sería:



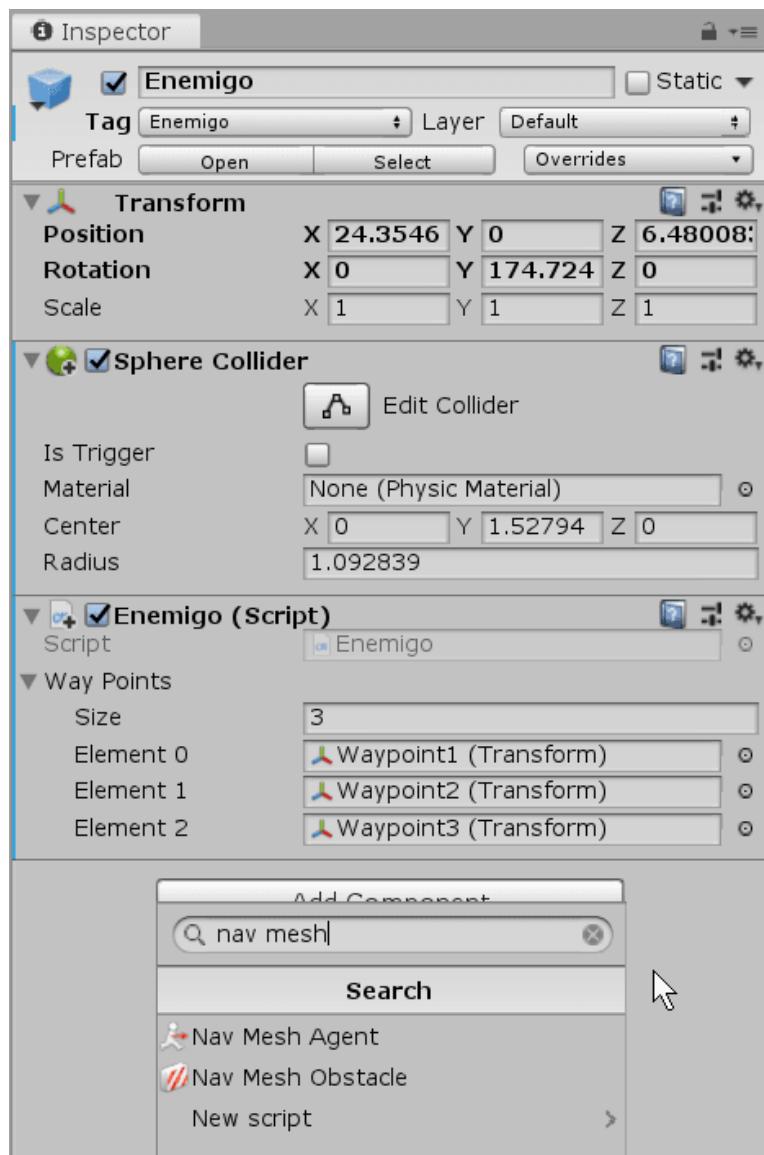
La luz por defecto es una "directional light", se comporta como una fuente lejana, como el sol, y afecta al sentido de todas las sombras del juego.

**Ejercicio propuesto 4.9.1:** Atenúa la luz global del juego y añade una luz puntual o focal.

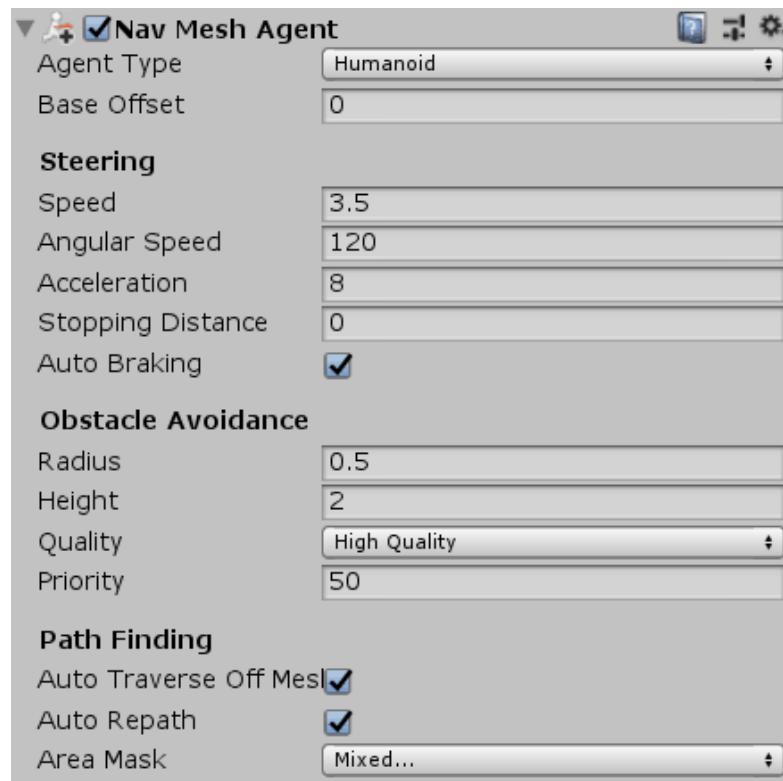
## 4.10. Búsqueda de caminos en Unity

Podemos hacer que el enemigo nos persiga cuando se encuentre a menos de una cierta distancia de nosotros (quizá añadiendo ciertos detalles adicionales, como que esté mirando hacia nosotros, e incluso que se gire si disparamos cerca, como si hubiese oído el disparo). El problema es que, con lo que sabemos hasta ahora, debería moverse en línea recta, y, por lo general, no podemos esperar que no haya obstáculos en el camino.

Pero Unity tiene una alternativa: crear un "agente de malla de navegación" (navigation mesh agent o "Nav Mesh Agent"), que podemos añadir a nuestro enemigo:



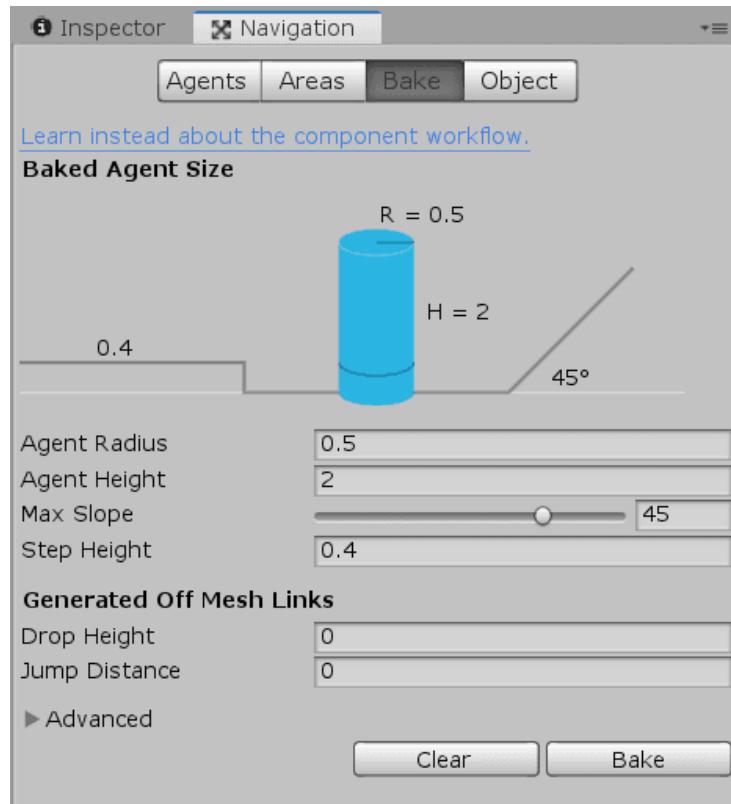
Y tendremos propiedades para afinar su velocidad lineal y angular, la distancia a la que esquivará obstáculos, etc.:



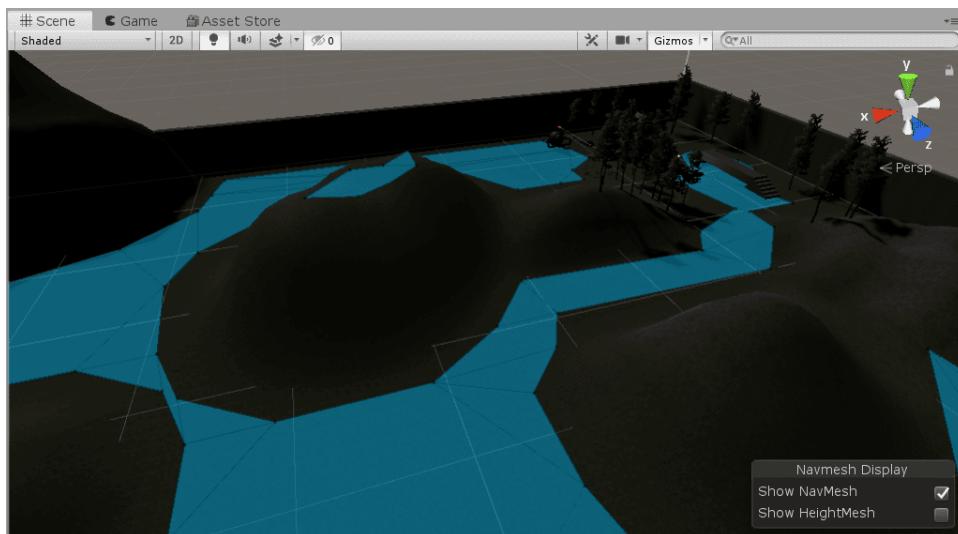
El siguiente paso es crear la "malla" en sí. Deberemos ir al menú Window para que se nos muestre la ventana de navegación (Navigation), dentro de la categoría AI (inteligencia artificial):



Deberemos pulsar el botón "Bake" para "cocinar" una rejilla de navegación para nuestra escena, y se nos preguntarán algunos detalles sobre el tamaño de nuestro enemigo:



Deberemos afinar un poco su radio y altura, la inclinación en grados que queremos que pueda subir y el tamaño de escalones. A continuación, pulsaremos el nuevo botón "Bake" y sobre el terreno se nos mostrará en color azul la zona que se considera "pisable" por nuestro enemigo según esa configuración:



Sólo falta hacer que nos persiga con la ayuda de ese "Nav Mesh Agent". Reescribiremos el script asociado al enemigo para que lo use para acercarse a un objetivo, que seremos nosotros. La forma podría ser simplemente:

```

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

public class Enemigo : MonoBehaviour
{
    [SerializeField] Transform objetivo;
    NavMeshAgent navMeshAgent;

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

    private void Update()
    {
        navMeshAgent.SetDestination(objetivo.position);
    }
}

```

(y desde el "inspector", fijaríamos nuestro jugador como valor para ese "objetivo").

Como mejora posible, podríamos hacer que el enemigo tuviera dos comportamientos distintos: perseguirnos cuando estemos a menos de una cierta distancia, o bien moverse a su aire si estamos lejos.

**Ejercicio propuesto 4.10.1:** Crea una malla de navegación, para que tu enemigo pueda perseguirte.

## 4.11. FixedUpdate

Hasta ahora siempre hemos usado los métodos Start y Update, que son los más importantes, y, en un par de ocasiones, Awake. Pero hay muchas más funciones relacionadas con el ciclo de vida y el orden de ejecución. Se puede ver más detalles en:

<https://docs.unity3d.com/Manual/ExecutionOrder.html>

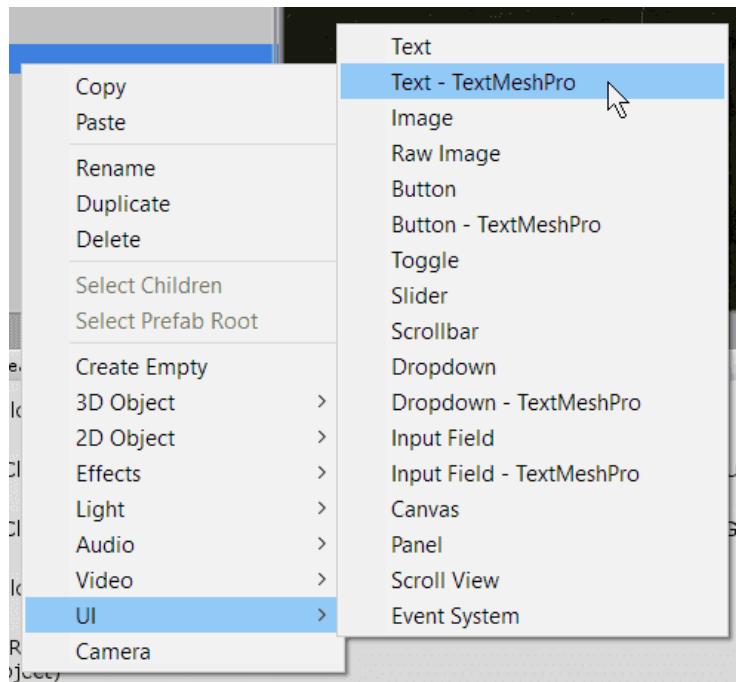
De esta lista, vamos a destacar dos relacionadas con Update:

- LateUpdate se llama una vez por fotograma, después de Update, por lo que es ideal para ajustar la posición de la cámara en un juego en tercera persona en el que ésta siga al personaje, por ejemplo.
- FixedUpdate se llama con un temporizador fijo, al contrario que Update, que podría no llamarse en algún fotograma. Gracias a eso, si se calculan detalles del movimiento usando FixedUpdate, no sería necesario multiplicar por Time.deltaTime.

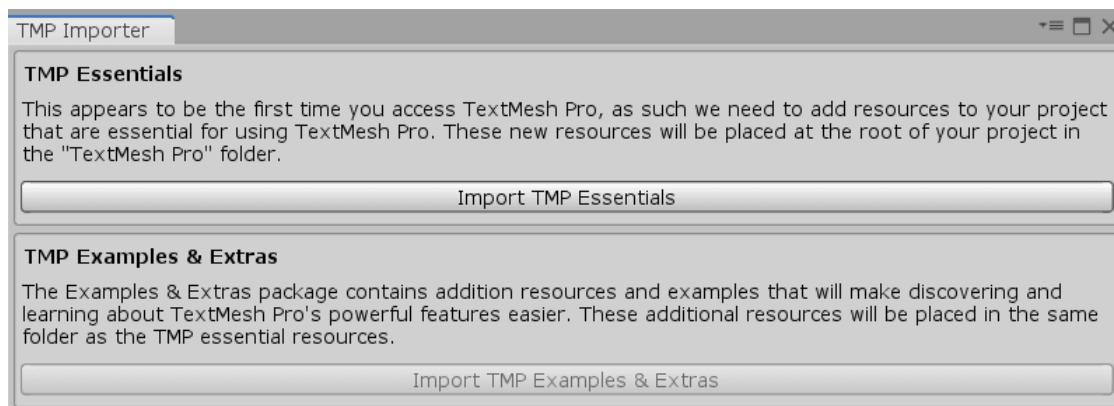
**Ejercicio propuesto 4.11.1:** En algún juego anterior en el que hayas usado Time.deltaTime, prueba a emplear FixedUpdate en vez de Update.

## 4.12. Textos más vistosos: Text Mesh Pro

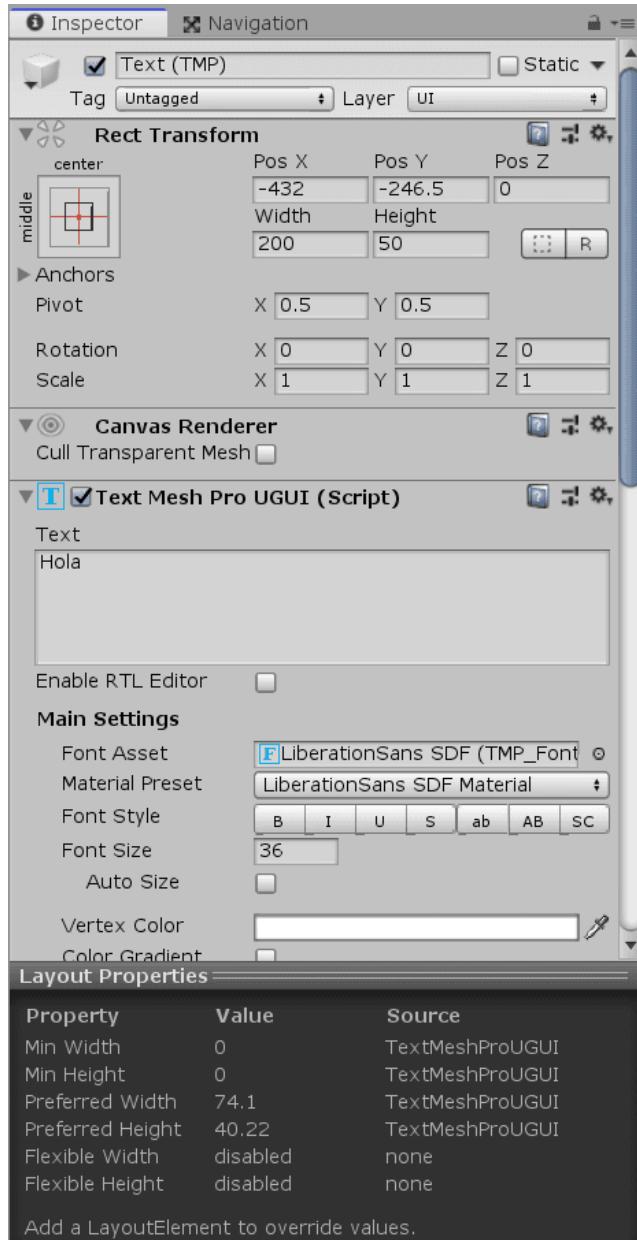
Podemos escribir textos más vistosos que con el componente Text, si usamos el "TextMeshPro", que en versiones anteriores de Unity era un extra instalable con el Package Manager, pero en versiones recientes ya es parte del sistema base. Se accede con el botón derecho, en la opción UI, al igual que los textos normales:



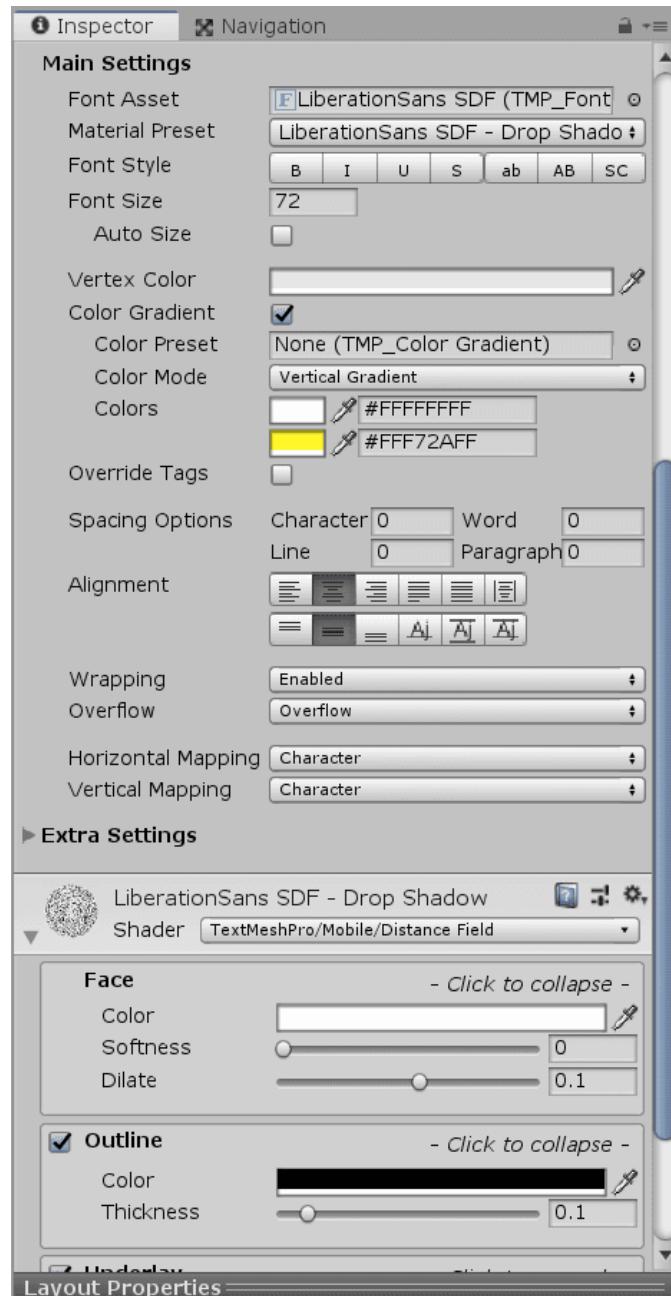
Y nos preguntará si queremos instalar sólo los componentes esenciales o también los ejemplos y extras. Para una toma de contacto básica nos bastará con los esenciales:



Se nos creará en nuestro Canvas (o aparecerá un nuevo canvas, si no tuviéramos uno), y los primeros componentes que veremos recordarán a los de un Text: posición (que normalmente querremos resetear), anclaje, alineación y color...



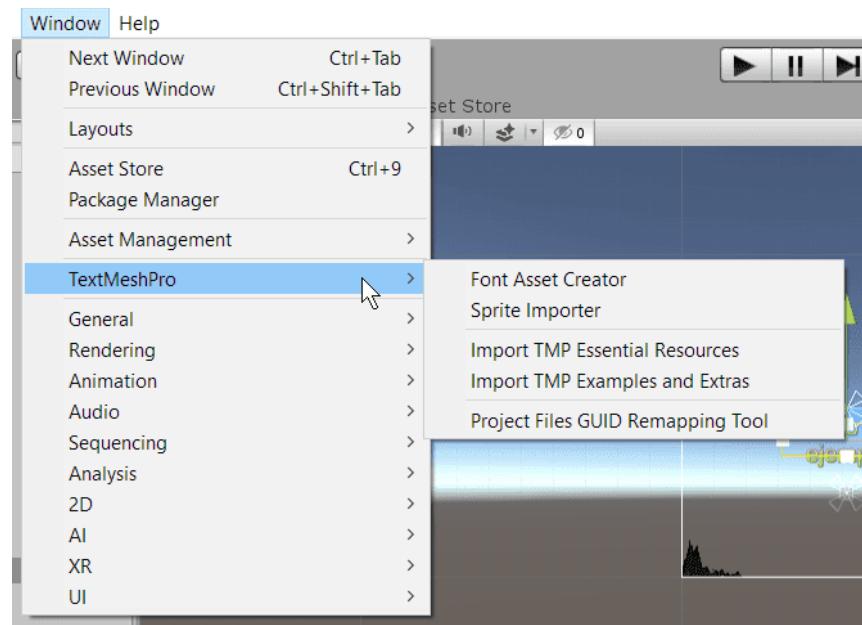
Pero realmente hay muchas más posibilidades. Por ejemplo, podemos colorear usando gradientes horizontales, verticales o de cuatro esquinas, así como añadir borde y sombra:



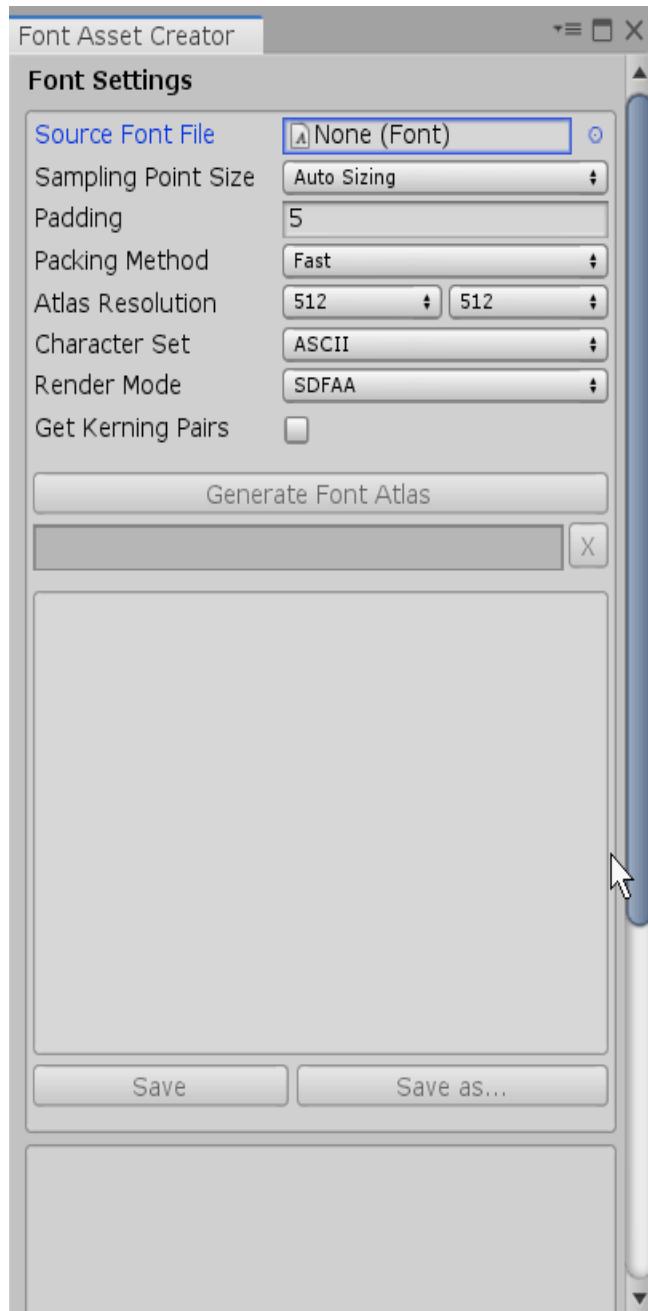
Con muy pocos clics, podemos conseguir resultados mucho más vistosos que con un Text normal:



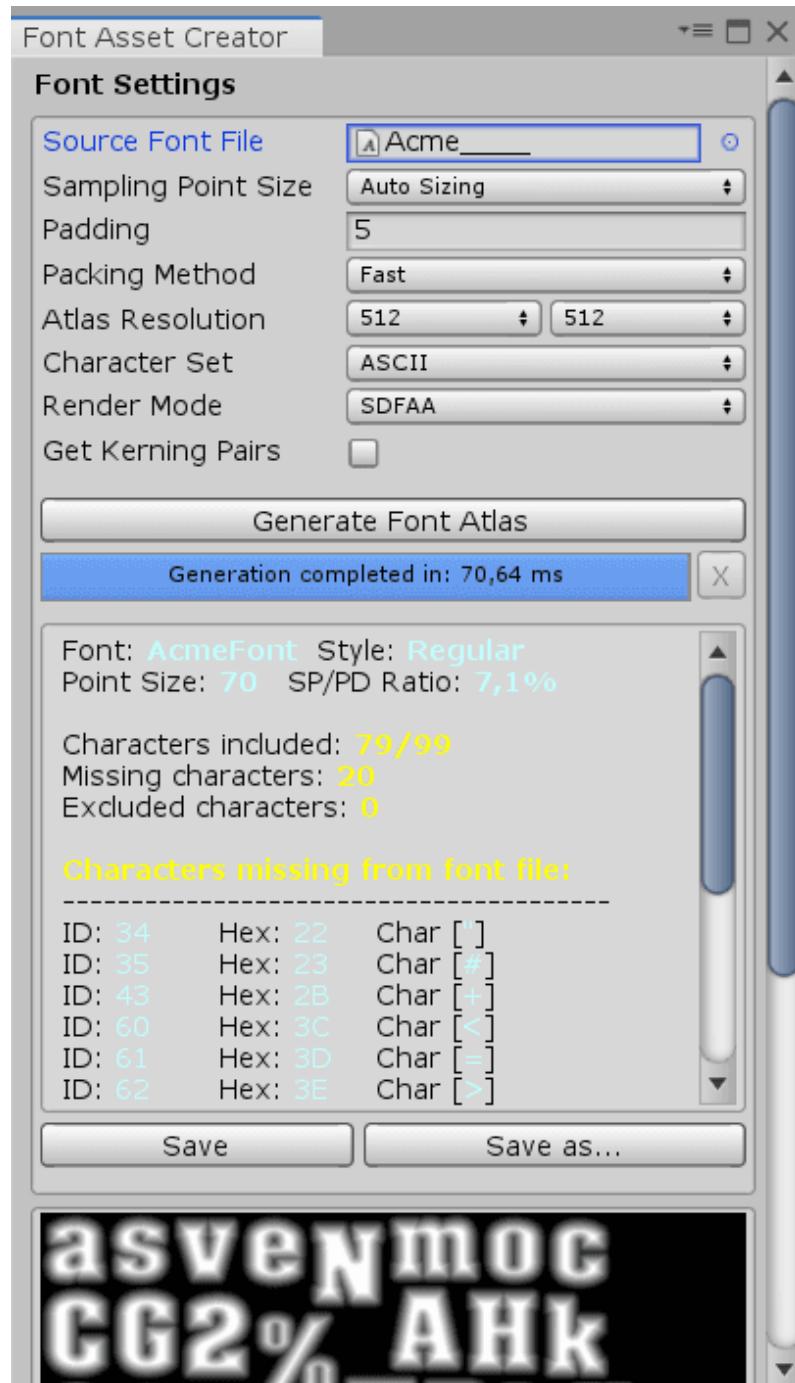
Se incluye un tipo de letra, pero podemos añadir cualquier otro. Para ello, iremos al menú Window, opción TextMeshPro, subopción "Font Asset Creator"



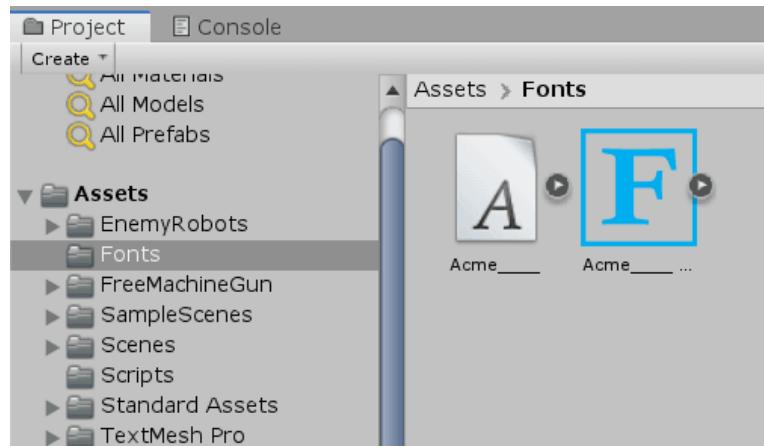
Y se nos pedirá que indiquemos (arrastremos) un fichero fuente ("Source Font File"):



A continuación pulsaremos el botón "Generate Font Atlas" y luego "Save":



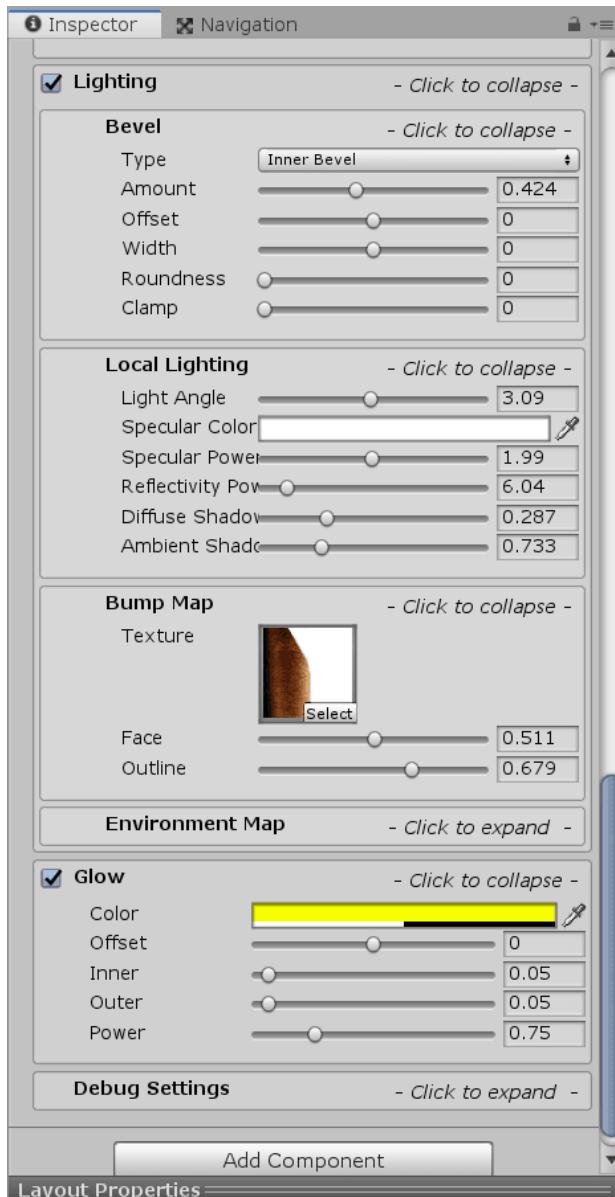
Y aparecerá un nuevo fichero creado a partir de nuestro TTF de partida y que ya podremos usar con TextMeshPro:



Con lo que podemos afinar mucho más el resultado, acercándolo a la apariencia que pretendamos:



Y se puede cambiar también parámetros más avanzados, como la iluminación local, los efectos de brillo, las texturas que hagan aparecer relieve, etc:



**Ejercicio propuesto 4.12.1:** Añade un texto a tu juego usando TextMeshPro. Puede ser un texto fijo durante el juego (como un marcador), o algo que sólo se vea en cierto momento (como un Game Over), o una pantalla de bienvenida, o parte de un botón.

## 4.13. Guardando la configuración

Si queremos guardar cosas como la configuración escogida por el usuario, o una tabla de records, podemos usar ficheros de texto o binarios, como cualquier otro programa en C#. Aun así, en Unity existe una alternativa extremadamente sencilla y portable entre distintos sistemas (aunque menos segura, porque el usuario podría encontrar el fichero y modificarlo a mano).

Se trata de "PlayerPrefs" (abreviatura de "preferencias de jugador". Y su uso es tan sencillo como usar "SetInt" para guardar un valor entero, así:

```
PlayerPrefs.SetInt("Puntos", 50);
```

y luego leerlo posteriormente (al cambiar de escena, o al empezar una nueva partida, según de qué información se trate) con

```
int puntos = PlayerPrefs.GetInt("Puntos", 0);
```

donde el segundo parámetro, opcional, es el valor por defecto en caso de que ese dato no se encontrase en el fichero de preferencias.

Al igual que SetInt, tenemos SetFloat y SetString, y sus correspondientes getters.

Los datos se guardan en disco automáticamente al salir del juego (salvo quizás si se interrumpe por alguna excepción inesperada), pero se puede forzar en cualquier momento el guardado con:

```
PlayerPrefs.Save();
```