Sistema de Generación de Mazmorras en Unity

Triana Casal Rafael Alberto

Introducción:

Este es un sistema de generador de mazmorras hecho en Unity3D, con un código hecho en Visual Studio (Cabe aclarar que la versión que yo ocupo es Unity 2021.3.10f1 y *VisualStudio* 2022). En este documento tendrás una breve explicación acerca del código, *"assests"* utilizados y cualquier movimiento que haya hecho dentro del mismo programa.

Utilidad:

Es útil para juegos que involucren la exploración de distintos tipos de zonas, ya sean salas de alguna mansión, mazmorras, laberinto, entre otras cosas; esto, ya que puede generar automáticamente una gran variedad de diseños de habitaciones que los jugadores pueden explorar. Esto puede ahorrar tiempo y esfuerzo al diseñador del juego al crear mazmorras de forma manual y también puede agregar una mayor variedad al mismo juego, ya que cada nueva partida puede tener un diseño de mazmorra distinto al que se vio anteriormente.

- *Numero 1: Elementos de Escena*

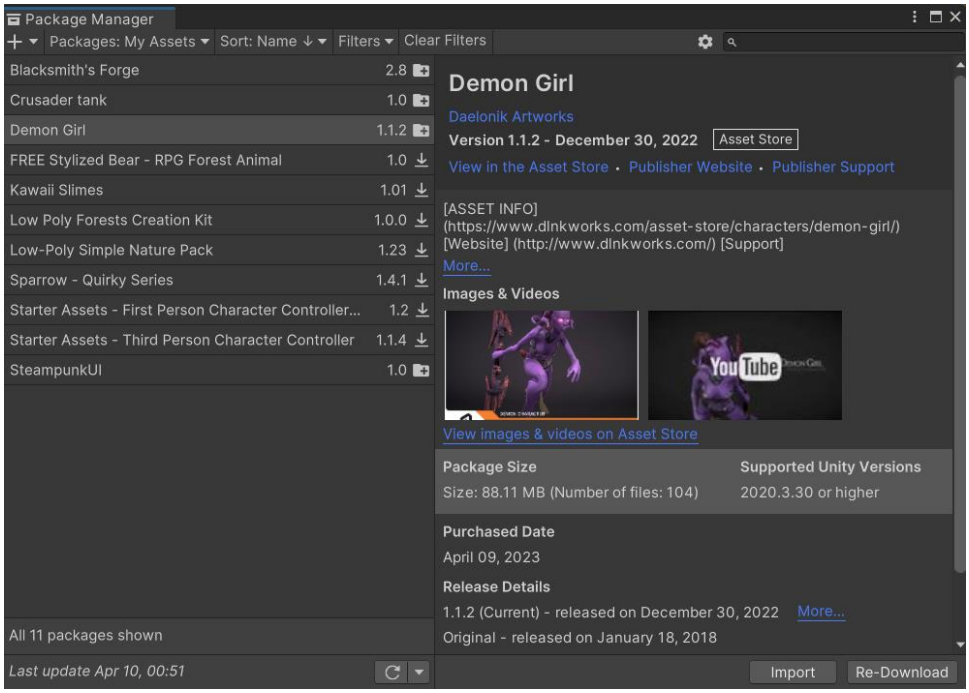
Aquí mostraré todo lo utilizado dentro de la escena y de donde ha sido obtenido.

Personaje:

Para el personaje he utilizado un paquete gratuito de Unity que se puede descargar directamente desde el apartado de *window* ----> *Asset Store* y te dará lo opción de descargarlo para poder utilizar algunas figuras o

materiales que Unity no cuenta de base. Sin embargo, se podrá trabajar con todas las opciones que Unity ya nos da de sus objetos bases.

Este es el *asset* descargado para el personaje que yo he utilizado.



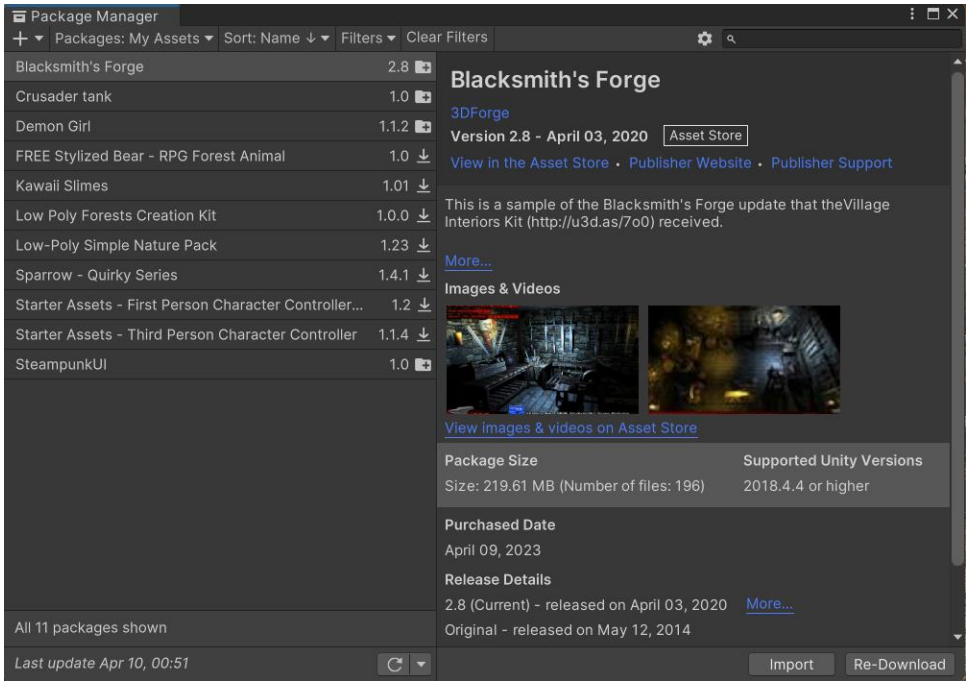
Propiedades del personaje (DemonGirl):

Al personaje que ha sido introducido a nuestra escena, se tendrán que agregar los siguientes componentes.

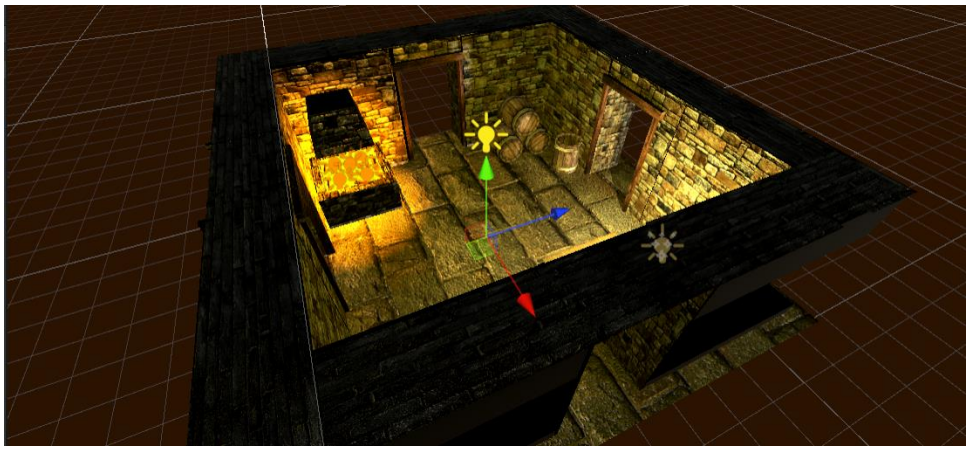
En este caso ya contamos con un código de control de movimiento de personaje con cámara aérea que se encuentra ya dentro de los scripts llamado *"Player Movement"*. Si se llegase a contar con alguna duda acerca de este código se podrá revisar el contenido de este para cualquier aclaración ya que cada línea de código cuenta con su respectiva explicación como un comentario añadido o también buscar la documentación de este.

Mapa:

Para el mapa he utilizado un paquete gratuito de Unity que se puede descargar directamente desde el apartado de *window* ----> *Asset Store* y te dará lo opción de descargarlo para poder utilizar algunas figuras o materiales que Unity no cuenta de base. Sin embargo, se podrá trabajar con todas las opciones que Unity ya nos da de sus objetos bases.



Este es el paquete en específico utilizado, donde es nada meramente visual para la construcción del mapa.



Todo elemento del mapa esta con una *Box Collider* donde he modificado el mismo *collider* a mi gusto. Esto es para poder darles propiedades físicas a los objetos y nuestro personaje no pueda traspasarlos.

Prefabs:

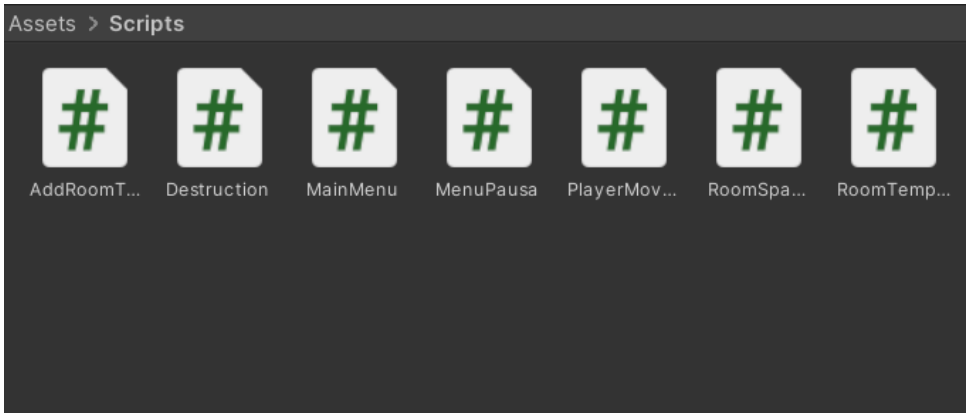
Para este código he utilizado recámaras con distintas salidas y sus respectivos *"spawn points"* (puntos de generación) y la he convertido en *"Prefab"*, esto será necesario para la realización del código y el juego.

Cada habitación tiene un nombre que especifica las salidas que contiene cada una respectivamente.

- *Número 2: Códigos (Scripts):*

Para este sistema de generación de mazmorras se han ocupado tres códigos distintos, los cuales cada uno cuenta con una función en específico. Dentro del código tendrás agregado comentarios explicando las líneas de código y su funcionalidad. Habrá más códigos que tendrán otras funciones dentro del juego, pero no afectan a la generación de mazmorras, principalmente lo que se conoce como la interfaz de usuario.

Como recomendación haz una carpeta específica para cada elemento que utilizarás, una para código, otra para materiales, prefaps, etc.



RoomTemplates:

Se utiliza para almacenar y actualizar una lista de habitaciones generadas aleatoriamente en el juego. La clase contiene "*arrays*" de GameObjects para cada tipo de habitación posible, una lista de GameObjects para almacenar las habitaciones creadas, un objeto GameObject para la habitación cerrada, un contador para el número de habitaciones generadas y un objeto Text para mostrar el contador en la interfaz de usuario.

El código también define métodos para incrementar y actualizar el contador de habitaciones en la interfaz de usuario.

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

public class RoomTemplates : MonoBehaviour
{
    // Se declaran arrays de GameObjects para cada tipo de habitación posible
    public GameObject[] bottomRooms;
    public GameObject[] topRooms;
    public GameObject[] rightRooms;
    public GameObject[] leftRooms;

    public GameObject closedRoom;
    // Se declara un GameObject para la habitación cerrada

    public List<GameObject> rooms;
    // Se declara una lista de GameObjects para almacenar las habitaciones creadas

    System.Random rnd = new System.Random();
    // Se declara una instancia del generador de números aleatorios de la clase Random

    private int roomCount = 0;
    // Se declara una variable para el contador de habitaciones generadas
    public Text roomCountText;
    // Se declara un objeto Text para mostrar el contador de habitaciones generadas en la interfaz de usuario
```

```
    public void IncrementRoomCount()
        // Método para incrementar el contador de habitaciones generadas
    {
        roomCount++;
        UpdateRoomCount();
        // Se llama al método para actualizar el contador en la interfaz de usuario
    }

    private void UpdateRoomCount()
        // Método para actualizar el contador de habitaciones en la interfaz de usuario
    {
        roomCountText.text = "Cuartos Generados: " + roomCount.ToString();
        // Se actualiza el objeto Text con el valor actual del contador de habitaciones
    }
```

RoomSpawner:

El código se encarga de generar una sala dentro del nivel, conectándola con otras salas ya existentes. El script utiliza un conjunto de plantillas de salas

(los RoomTemplates) para seleccionar y generar la sala correspondiente según el tipo de puerta que se necesite (top, bottom, left o right), y se asegura de que cada sala generada se contabilice en la cuenta total de salas generadas (roomCount). Además, el código detecta cuando dos spawners de salas colisionan, lo que significa que una sala ya ha sido generada en esa dirección, y en su lugar crea una sala cerrada para bloquear el camino y evitar bucles infinitos en la generación de niveles.

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

public class RoomSpawner : MonoBehaviour
{
    // La cantidad de lados abiertos que tiene la habitación en la que se encuentra este spawner.
    public int openSide;

    // Referencia al script RoomTemplates que contiene las plantillas de habitaciones disponibles.
    private RoomTemplates templates;

    // El número aleatorio utilizado para elegir una plantilla de habitación.
    private int rand;

    // Indica si la habitación ya ha sido generada por este spawner.
    private bool spawned = false;

    private void Start()
    {
        // Busca el objeto con la etiqueta "Rooms" y obtiene una referencia al script RoomTemplates.
        templates = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();

        // Invoca el método Spawn después de 0.1 segundos.
        Invoke(nameof(Spawn), 0.1f);
    }
```

```
private void Spawn()
{
    // Si la habitación ya ha sido generada, no hace nada.
    if (spawned) return;

    // Seleciona una plantilla de habitación según el lado abierto de la habitación en la que se encuentra este spawner.
    switch (openSide)
    {
        case 1: // Necesita puerta inferior.
            rand = Random.Range(0, templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand], transform.position, templates.bottomRooms[rand].transform.rotation);
            break;

        case 2: // Necesita puerta superior.
            rand = Random.Range(0, templates.topRooms.Length);
            Instantiate(templates.topRooms[rand], transform.position, templates.topRooms[rand].transform.rotation);
            break;

        case 3: // Necesita puerta izquierda.
            rand = Random.Range(0, templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand], transform.position, templates.leftRooms[rand].transform.rotation);
            break;

        case 4: // Necesita puerta derecha.
            rand = Random.Range(0, templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand], transform.position, templates.rightRooms[rand].transform.rotation);
            break;
    }
```

```
    // Incrementa la cuenta de habitaciones generadas en el script RoomTemplates.
    templates.IncrementRoomCount();

    // Marca la habitación como generada.
    spawned = true;
}

private void OnTriggerEnter(Collider other)
{
    // Si el collider no tiene la etiqueta "SpawnPoint", no hace nada.
    if (!other.CompareTag("SpawnPoint")) return;

    // Obtiene una referencia al RoomSpawner del otro collider.
    var otherSpawner = other.GetComponent<RoomSpawner>();

    // Si el otro spawner aún no ha generado una habitación y esta habitación
    // tampoco ha sido generada, se genera una habitación cerrada y se destruye este spawner.
    if (otherSpawner != null && !otherSpawner.spawned && !spawned)
    {
        Instantiate(templates.closedRoom, transform.position, Quaternion.identity);
        Destroy(gameObject);
    }

    // Marca la habitación como generada.
    spawned = true;
}
```

Destruction:

Este código implementa la funcionalidad de destruir cualquier objeto que entre en contacto con el objeto al que se agrega este script, utilizando el método "*OnTriggerEnter"*. En otras palabras, si un objeto con un "*Collider"* entra en contacto con el objeto que tenga este script, ese objeto se destruirá inmediatamente.

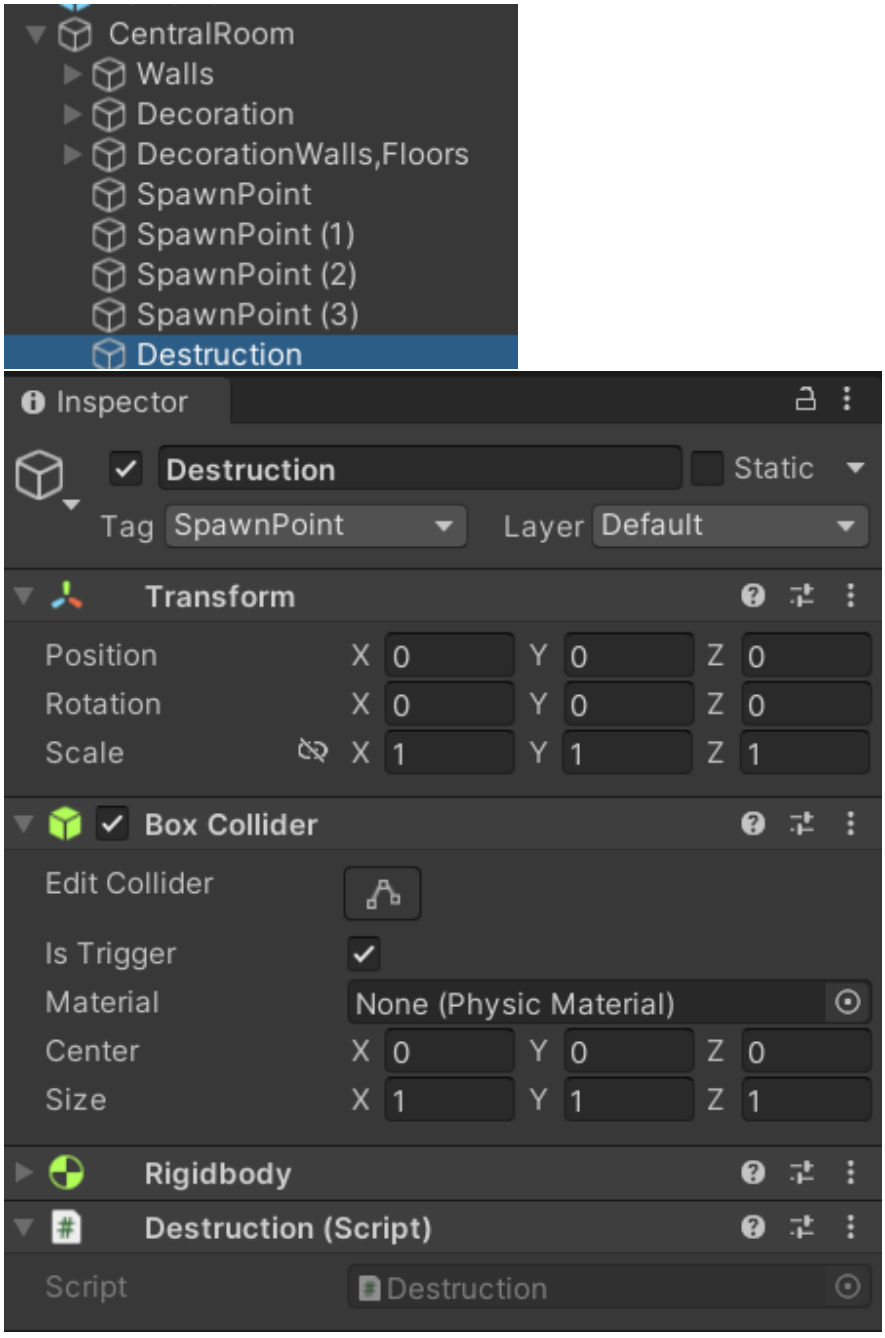
```
// Importa los namespaces System.Collections y System.Collections.Generic
// y UnityEngine para utilizar las clases y funciones de estas bibliotecas.
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Define la clase Destruction que hereda de MonoBehaviour.
public class Destruction : MonoBehaviour
{
    // Define una función que se ejecutará cuando un collider ingrese en contacto con este objeto.
    private void OnTriggerEnter(Collider other)
    {
        // Destruye el objeto asociado al collider que ha entrado en contacto con este objeto.
        Destroy(other.gameObject);
    }
}
```

- *Número 3: Asignación:*

Destruction:

Este Script será asignado a un *"Empty Object"* el cual estará asignado en la *"Central Room"*, esto para evitar que se creen habitaciones encimas de

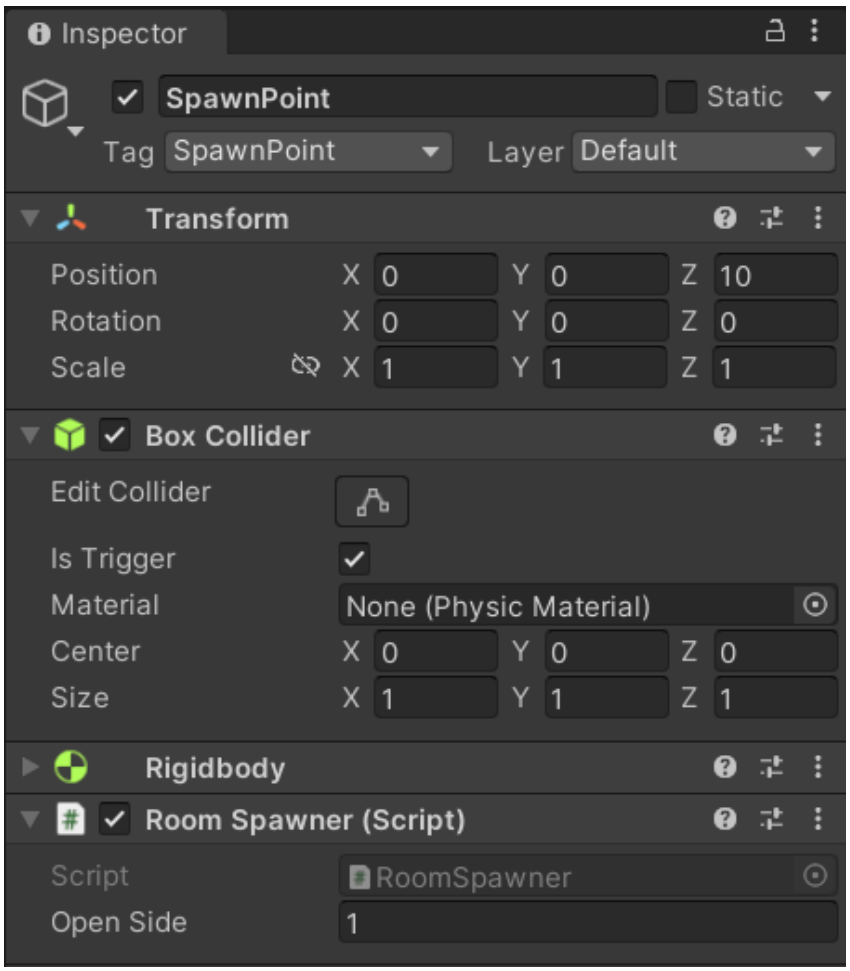
esta. También tomar en cuenta que este *"Empty Object"* tendrá la etiqueta de *"Spawn Point"*.



Room Spawner:

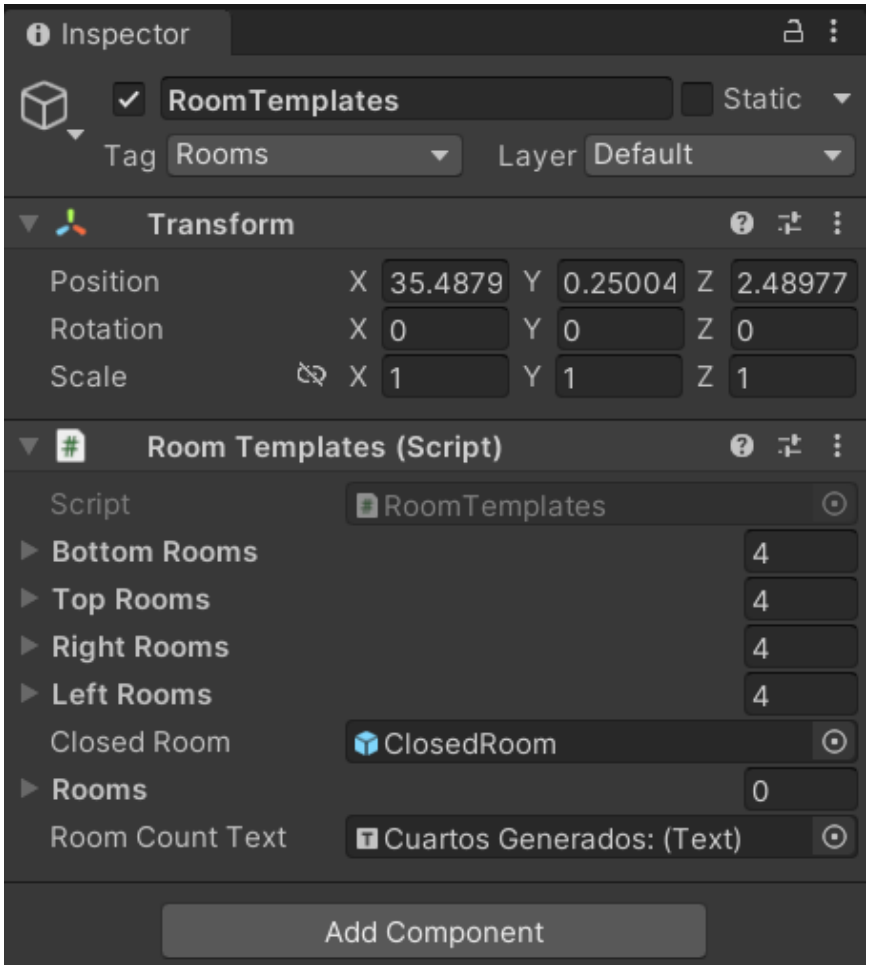
Este Script será asiginado a todos los *"Spawn Points"* hecho en cada una de las habitaciones, donde darás a cada salida (*"Spawn Point"*) un número

de identificación, en este caso es "*Botton"* (Abajo)= 1, "*Top" (*Arriba) = 2, "*Left"*(Izquierda) = 3 y "*Right"*(Derecha) = 4.



RoomTemplates:

Este Script será asignado a un "*Empty Object*" que se encontrará en la misma posición que nuestra habitación central y donde se le colocarán sus respectivas funciones.


Inspector
RoomTemplates
Static
Tag
Rooms
Layer
Default
Transform
Position
X 35.4879
Y 0.25004
Z 2.48977
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Room Templates (Script)
Script
RoomTemplates
Bottom Rooms
4
Top Rooms
4
Right Rooms
4
Left Rooms
4
Closed Room
ClosedRoom
Rooms
0
Room Count Text
Cuartos Generados: (Text)
Add Component