Máster en Desarrollo de Videojuegos

(Especialidad de Programación)

Motores: Unity

Construyendo un Thirds Person Platformer. I Parte.

[15 - 10 - 2019]

Ismael Sagredo: isagredo@ucm.es

Universidad Complutense de Madrid

Contenido

Contenido 1
Preparativos 2
Third Person Platformer 3
Banjo - Kazooie 4
Donkey Kong 64 4
Rayman 2 5
Evolución del Género: 6
Tomb Raider: 6
Uncharted 7
Creación de la escena inicial y el personaje 9
Carga de Niveles 10
Game Manager 10
Creamos el primer Nivel: Verde. 12
Recogemos items. 14
Mostrar información en el GUI 16

Preparativos

En las anteriores clases conseguimos construir un prototipo de FPS. Ahora vamos a crear un prototipo de juego de plataformas en 3d en tercera persona.

Durante estas sesiones vamos a aprender a utilizar las siguientes características:

- Gestión de animaciones
- Creación de obstáculos de diferentes tipos.
- Diseño de niveles de variada dificultad.
- Configuración y uso de sistemas de partículas.
- Creación de cutscenes
- Carga/descarga de niveles
- Uso del GUI de Unity

Todas las sesiones se realizarán sobre un **mismo proyecto**, de forma que vaya creciendo poco a poco hasta llegar a ser algo parecido a un juego.

Debido a esto último, se aconseja que os guardéis en un **pendrive** o algo similar los proyectos que vayáis acabando después de cada clase.

Si por alguna razón **no podéis** hacer una copia de seguridad de vuestro trabajo, y cuando volváis al laboratorio han borrado los proyectos no os preocupéis, en el **La Wiki** se irán colgando todas las **versiones intermedias**, de forma que para cada práctica estará disponible ya **resuelta**, la anterior.

Third Person Platformer

Los **Third Person Platformers** son un género que surge a partir de **Super Mario 64**. Y que tuvo su punto álgido a finales de la década de los 90. Especialmente proliferaron en la consola Nintendo 64 donde aparecieron el anteriormente nombrado padre del género: Super Mario 64, una evolución 3d de los juegos clásicos de Mario 2D que habían aparecido de NES y Super NES. Super mario 64 también es considerado como el precursor de los Sandbox o juegos de mundo abierto.

Su mecánica principal consiste en mundos más o menos abiertos y de recorrido libre, con un camino para superar, normalmente un enemigo final o un acceso a una puerta secreta que te permita pasar de nivel. En la fase se encuentran diversos enemigos que hay que superar, bien saltando sobre ellos o disparándolos o golpeándolos de algún modo. Normalmente existen partes móviles a donde hay que saltar para acceder. Es importante saltar en el momento preciso y coordinarse correctamente con las plataformas para no caer sobre un enemigo o precipitarse al vacío.

Junto con Mario 64 en aquella época podemos mencionar juegos como:

Banjo - Kazooie

Manejabas a un oso (Banjo) y a un pájaro (Kazooie) y ambos se compenetraban para resolver las plataformas y puzles que iban surgiendo por el camino. Cada uno tiene unas habilidades diferentes. El sentido del humor del juego es una de sus señas de identidad. A Banjo Kazooie le continuó su secuela **Banjo Tooie**.

Donkey Kong 64

Heredero de los Donkey Kong 2D de Super Nintendo, se adaptó a la nueva moda de los plataformas 3D de finales de los 90. Fue la propia Rare la que creo el juego al igual que Bajo y sus predecesores Donkey Kong Country en Super Nintendo.

Rayman 2

Otro plataformas clásico que se pasó a la moda de los plataformas 3D.

Otros plataformas que derivan de estos son **Crash Bandicoot**, conservando en mayor medida la perspectiva bidimensional en gran parte de sus niveles**.**

O **Ratchet and Clank** más orientado a la acción:

Últimas entregas de Super Mario 3D: **Super Mario Galaxy** que juega con la gravedad y recorrer planetoides, **Super Mario 3D World** con cooperativo, o **Super Mario Oddysey** con controles por sensor de movimiento**.**

Evolución del Género:

No pueden ser considerados juegos del género pero sí que se basan en ellos.

Tomb Raider:

Plataformas, gotas de acción, y sesudos puzles. Tomb Raider es una evolución de los plataformas 3D de los 90 que ha llegado hasta nuestros días con el reboot de **Tomb Raider** y **Rise of The Tomb Raider**. Se mezclan elementos narrativos, se prescinde un tanto de la complejidad de los saltos donde se guía mucho más al usuario… Pero aún recuerdan a sus origenes.

Uncharted:

Plataformas, muy en la línea de **Tomb Raider**, con mucho más peso de mecánicas shooter y coberturas (sobre todo a partir del 2) tras el éxito de **Gears of War**. Pero aún así tiene reminiscencias de un plataformas 3D.

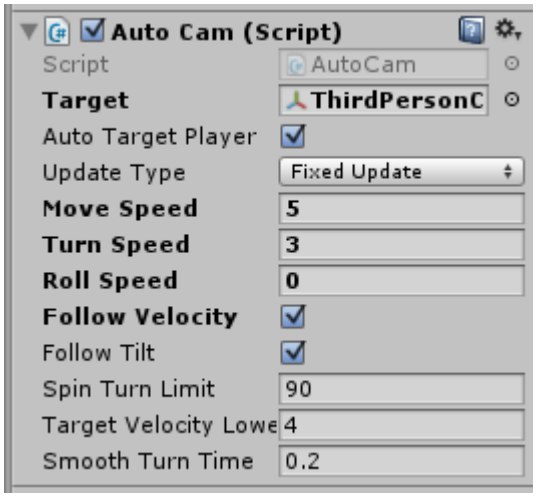
No es el único caso, podemos establecer una analogía entre **Prince of Persia 2D** y su salto a las 3D y su posterior evolución hacia **Assassins Creed**.

Creación de la escena inicial y el personaje

Para empezar, debemos **Crear un proyecto nuevo**. De momento no importaremos ningún paquete por defecto, así que el proyecto tendrá que estar vacío. Importamos los **Standard Assets** de Unity.

Arrastramos a la escena el Third Person Controller. Que está en **Standard Assets>> Characters >> ThirdPersonCharacter >>Prefabs >> ThirdPersonController**

- Le cambiamos el **Tag** a Player. Y colocamos el player en la (0,0.5,0)
- Añadimos un suelo en el (0,0,0) creando un cubo y le cambiamos la escala a (10,1,10)
- Eliminamos el **Main Camera**.
- Instanciamos **Standard Assets >>camera>> Prefabs >> MultipurposeCameraRig.**
- Le arrastramos el **GameObject ThirdPersonController** al atributo Target si no está establecido ya. La configuración del **Auto Cam** la podemos ver a continuación.



Target es el game object al que sigue.

- **Move speed** es a la velocidad a la que se mueve.
- **Turn Speed** velocidad a la que se gira.
- **Roll Speed** velocidad a la que el personaje rueda. (No aplicable en nuestro caso)
- **Follow Velocity**: sirve para que la cámara siga al personaje, si éste gira en la dirección en la que se mueve.

- **Follow Flit**: Si el personaje se inclina al andar, entonces la cámara lo tiene en cuenta.
- **SpinTurnLimit**: El umbral más allá del cual la cámara se detiene tras rotación del objetivo.
- **Target Velocity Lower Limit**: La velocidad mínima de giro de la cámara.
- **Smooth turn Time**: suavizado del giro de la cámara.

Carga de Niveles

Vamos a realizar nuestro juego de plataformas 3D. Cada clase se centrará en un color y se creará una parte del nivel que aplique los conceptos creados. Así pues, habrá una escena Start multicolor que interconecta una escena Verde, Azul, Amarilla y Roja.

Cada escena contiene una llave para abrir la siguiente, pero para llegar a la llave el jugador tendrá que sortear una serie de obstáculos.

Guardamos en un prefab el Third Person Controller y la Cámara que llamaremos MainCamaraPlatformer y Player. Para guardarlo, por ejemplo crearos una carpeta Prefabs y los colocamos debajo de un gameobject común (por ejemplo lo podemos llamar **TPPlatformer**) para tener ambos en un mismo prefab.

- Instanciamos el prefab del Nivel0.
- Arrastramos los prefab de **TPPlatformer** y los colocamos sobre el escenario
- Añadimos una luz direccional
- Agrupamos todas las partes del escenario como hijo de un nuevo GameObject que llamaremos **Escenario**.

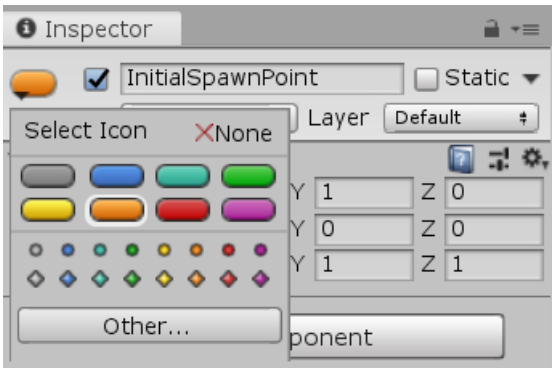
Game Manager

Necesitamos un GameManager que gestione las cosas del juego a nivel global. Entre otras cosas, controlaremos en él la carga de niveles.

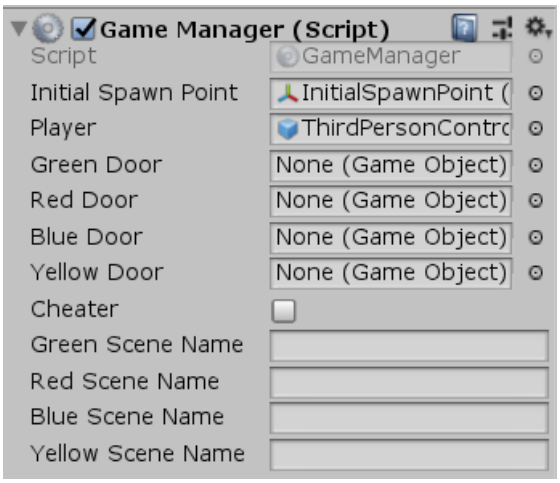
- Creamos un **GameObject** vacío llamado **GameManager**
- Creamos un **Tag** (*Edit> Project Settings>Tags o desde el Tag del GameObject*) **GameManager.**
- Añadimos el componente **GameManager**.

Ahora vamos a crear un punto de Spawn. Este punto nos servirá para instanciar al player cuando muera.

- Creamos un GameObject vacío, y lo llamamos **InitialSpawnPoint.** En la posición 0,1,0.
- Le asignamos un icono para poder verlo en el editor

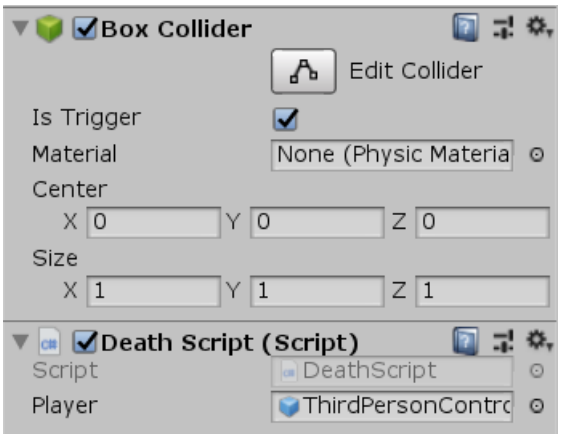


- Lo arrastramos al GameManager en el atributo **Initial Spawn Point**.
- Arrastramos al atributo Player el transform the **Thirds Person Controller (Ojo al controller. Es quien tiene la física)**.



Ahora crearemos un plano de muerte que haga que el player muera al contactar con él y por tanto respawnee. (Mejor que un plano una caja ☺). Quitar el MeshRender y el Mesh Filter de la caja de muerte y cambiarle el nombre a **DeathPlane**.

- Configurar como **trigger**.
- Añadir el componente **DeathScript**
- Configurar sus atributos públicos.



Creamos el primer Nivel: Verde.

En este apartado crearemos el primer nivel de nuestro plataformas. Dicho nivel será una nueva escena dentro de nuestro proyecto. Esta escena se cargará de manera aditiva a la escena principal ¿Qué significa esto?

A grandes rasgos, dentro de Unity una escena puede cargarse de dos formas con un mismo método:

SceneManager.LoadScene(**string** sceneName**, LoadSceneMode** mode)

Donde **LoadSceneMode** es un enumerado que puede tomar dos valores: **Single** (por defecto) que carga la escena indicada, o **Additive** que carga una escena dentro de la escena de juego.

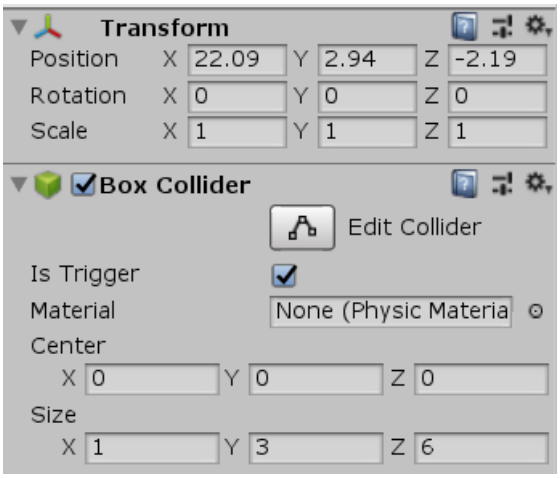
A parte de la carga normal hay una carga Asíncrona. La carga asíncrona no bloquea el juego mientras carga, lo que permite cargar objetos de forma mucho más transparente para el juego y no afecta tanto al rendimiento. Se puede utilizar para construir pantallas de carga o bien para cargar una subescena sin que el jugador note el tirón de la carga.

- Guardamos la escena como una nueva escena que llamaremos **green_world**.
- Hacemos un prefab con el GameObject Scenario.
- Instanciamos el Escenario
- Instanciamos el PasilloVerde
- Nos aseguramos de que ambos estén contiguos conectados por la puerta verde.
- Hay que borrar la luz, la cámara y el escenario.
- Crear un GameObject green_world vacío en el (0,0,0) y colocar toda la escena debajo. (PlataformaVerde y Pasillo Verde).

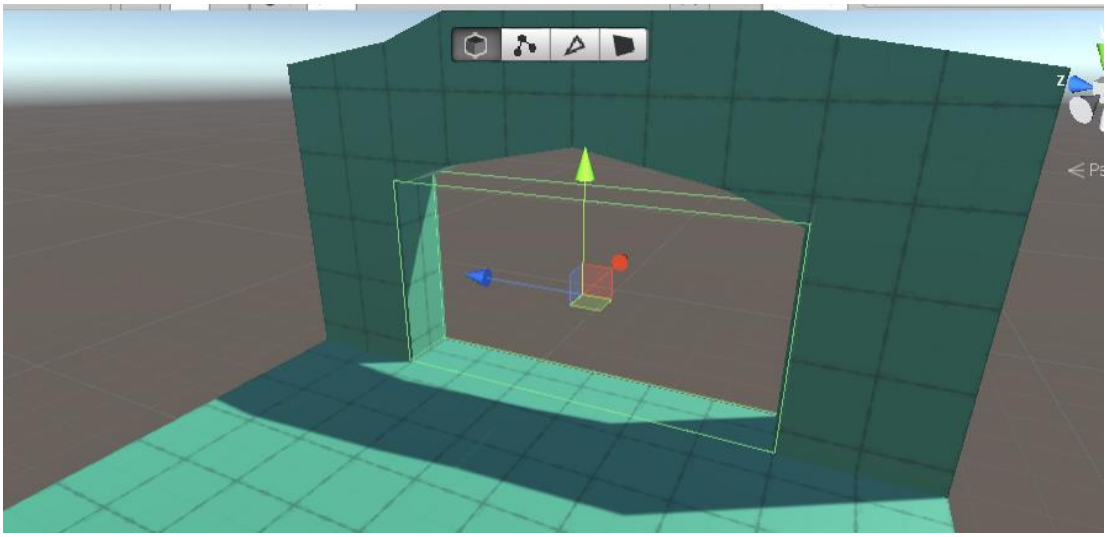
- Guardamos la escena.

Ahora vamos a configurar la puerta para que active la carga de la escena. Al principio del juego, únicamente estará activado el portal **Verde**, y conforme vayamos completando niveles, podremos ir activando los demás.

- Abrir la escena Start
- Crear un GameObject vacío y llamarlo **GreenTrigger**. Añadirle un componente **Box Collider**, y hacerlo hijo de **GreenDoor.**



- Marcar el BoxCollider como **trigger**.
- Ajustar el BoxColider para que ocupe la pared completa como aparece en la figura.



- Añadir el componente **TriggerLoadAdditive** al **GreenTrigger**. Y completar los TODOS.

```
public class TriggerLoadAdditive : MonoBehaviour
{
    // TODO 1 - Añadir string público que será el nombre del nivel a cargar
    public GameManager.DoorColor m_LevelToLoad;


    // TODO 2 - Hacer función que cargue un nivel de forma aditiva.
    // Será necesario pasarle el gameObject que ha colisionado con el trigger
    // Tras cargar el nivel, desactivamos el trigger
    void LoadLevelAdditive(GameObject go)
    {
        if (go.tag == m_PlayerTag)
        {
            //Application.LoadLevelAdditive(m_LevelToLoad);
            //GetComponent<Collider>().enabled = false;
            GameObject gogm =
GameObject.FindGameObjectWithTag("GameManager");
            GameManager gm = gogm.GetComponent<GameManager>();
            gm.TriggerLoadAdditive(m_LevelToLoad);
        }
    }

}
```

- Configurar los atributos públicos.
- Añadir tanto **Start** como **green_world** al **Build Setting**.

Recogemos items.

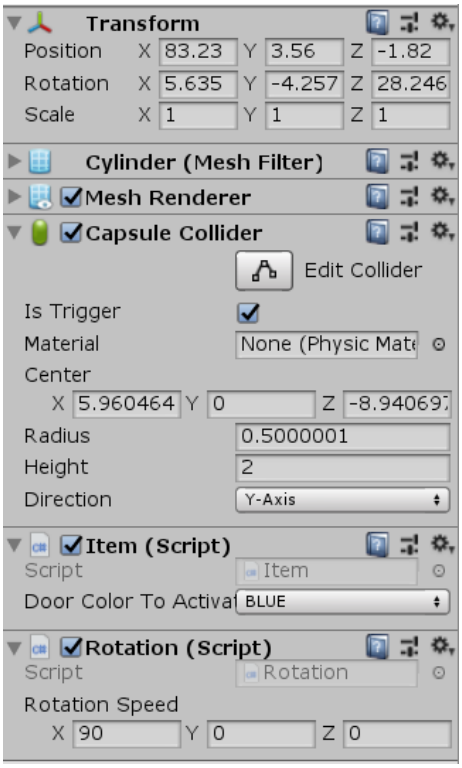
Como ya se ha mencionado en apartados anteriores, cuando comencemos a jugar no estarán accesibles todos los niveles del juego, sino que únicamente estará el nivel Verde.

Para poder ir abriendo nuevos niveles, necesitamos conseguir una serie de ítems que nos "activen" los triggers que dispararán la carga aditiva del nivel.

- Abrimos la escena Start.
- Asignamos al GameManager todas las puertas.
- Probamos los *cheaters*.
- Creamos el GameObject **GenericItem**
- Añadimos el componente **Item**
- Completamos los Todos.

```
// TODO 1 - Obtener componente GameManager del GameObject m_GameManager y
guardarlo
            // en una variable local
            GameManager gmComp = m_GameManager.GetComponent<GameManager>();

            // TODO 2 - Si el componente existe..
            if (gmComp)
            {
                // TODO 3 - llamar directamente a la función ActivateDoor
sobre el componente GameManager
                // Como parámetro, habrá que pasarle el
m_DoorColorToActivate, y un true, indicando que SI quieres activar la puerta
                gmComp.ActivateDoor(m_DoorColorToActivate, true);

                // TODO 4 - Crear un nuevo GameObject (indicar que es
necesario para crear un nuevo transform)
                // Dicho GameObject será el nuevo spawnPoint del player.
Habrá que sumarle (0,4,0) a su posición para
                // que al hacer el relocate del player, éste no atraviese el
suelo
                GameObject spawn = new GameObject();
                Vector3 playerPos = other.transform.position;
                spawn.transform.position = new Vector3(playerPos.x,
playerPos.y + 4, playerPos.z);

                // TODO 5 - Llamar directamente a la función
SetCurrentSpawnPoint sobre el componente GameManager
                // pasando como parámetro la matriz de transformación de
nuestro nuevo GameObject
                gmComp.SetCurrentSpawnPoint(spawn.transform);

                // TODO 6 - Autodestruirse
                //Desaparece el ítem de la escena
                // Pista: Destroy(...);
                Destroy(this.gameObject);

            }
            else
            {
                Debug.LogWarning("No existe componente GameManager dentro del
Game Manager");
            }
```


- Creamos un prefab a partir del item.
- Creamos una instancia al final de green_world con el material azul.
- Configuramos Item como se muestra en la figura



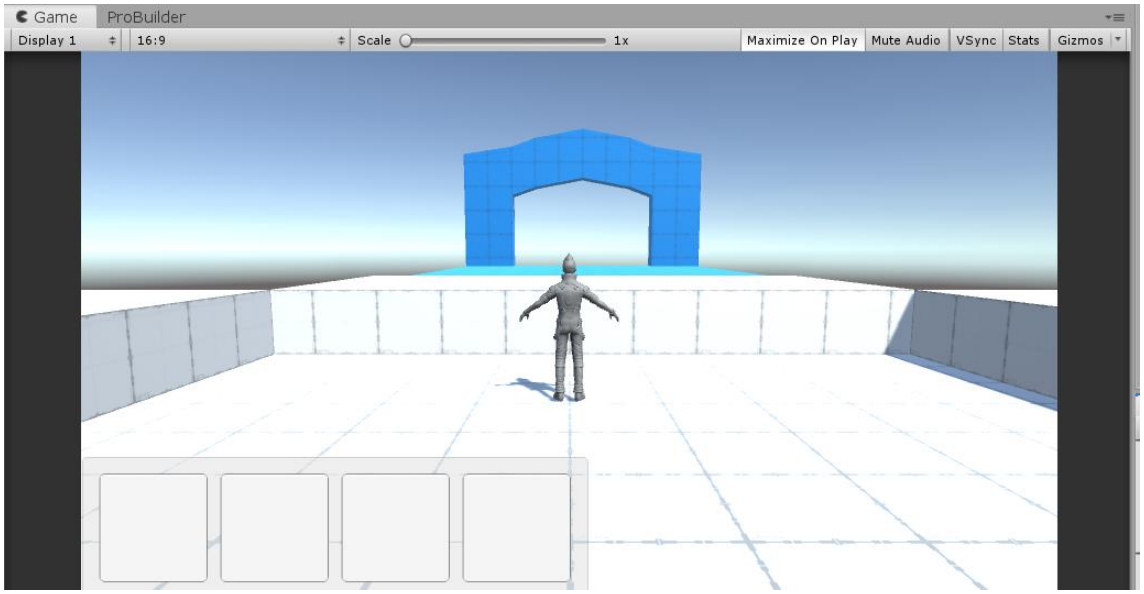
Mostrar información en el GUI

Ahora vamos a probar a crear un GUI para nuestro personaje. Mostraremos cuatro cuadrados de color blanco que cambiarán de color cada vez que el usuario coja una nueva llave.

- Creamos un **canvas**
- Añadimos un panel
- Dentro del panel añadimos una imagen
- Modificamos **source imagen** y añadimos el **UISprite** que viene en Unity**.**
- Creamos un prefab con la imagen.
- Instanciamos 4 objetos.
- Añadimos al panel un **horizontal layout group** para que las coloque. El espacio entre cajas será de 10.

- En el atributo **Child Alignment** => **Middle Center**.
- Desactivamos los dos ticks: **Child Force Spand**, **Height**.

Añadimos **Layout Element** seleccionando todos los imagenDoor y marcando ***Preferred Width*** y **Preferred Height** y estableciendo los valores a 55.



- Finalmente añadimos el element GUIITem a los imageDoor y completamos los TODOs.
- Configuramos cada uno en su color
- Probamos el nivel.

```
void Update () {
        // TODO 1. En cada tick tenemos que comprobar si la puerta está activa
        if (m_doorGameObject == null)
            return;
        if (!m_doorGameObject.activeInHierarchy) return;

        // TODO 2. Si es así, tenemos que pintar la imagen m_Image del mismo
color que la puerta
        switch (m_DoorColor)
        {
            case GameManager.DoorColor.BLUE:
                m_Image.color = Color.blue;
                break;
            case GameManager.DoorColor.GREEN:
                m_Image.color = Color.green;
                break;
            case GameManager.DoorColor.RED:
                m_Image.color = Color.red;
                break;
            case GameManager.DoorColor.YELLOW:
                m_Image.color = Color.yellow;
                break;
        }
```


```
        // TODO 3. Por último, para que el componente no haga chequeos inútiles
        // lo desactivamos (sólo si la puerta está activa)
        this.enabled = false;
    }
```