SESSIÓ 3. (1)

Afegir enemics al joc

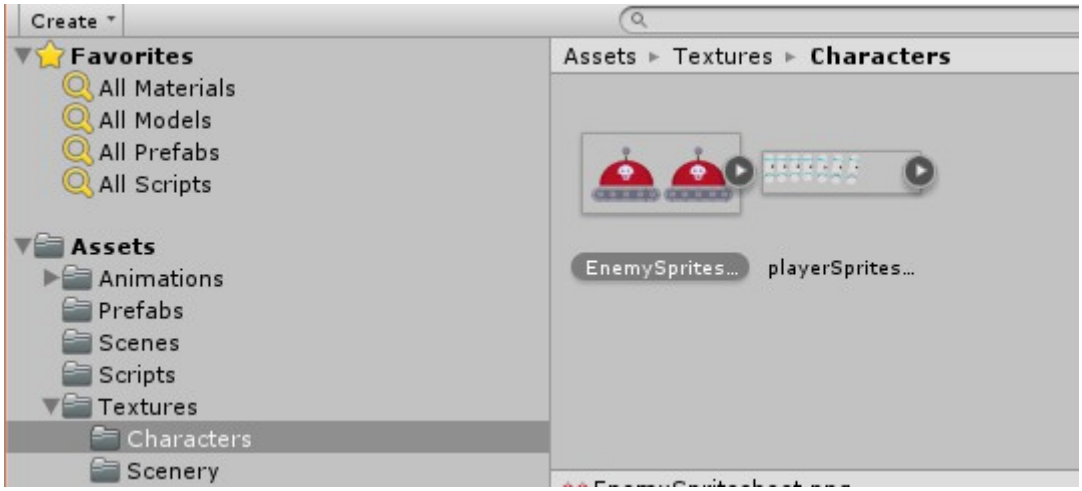
1. Construcció del personatge Enemy.

El que farem en aquest apartat ja ho hem practicat abans: importar sprites, crear objectes, assignar components collider i rigid body, crear una animació i començar la codificació d'un script que donarà el comportament que volem als nous personatges.

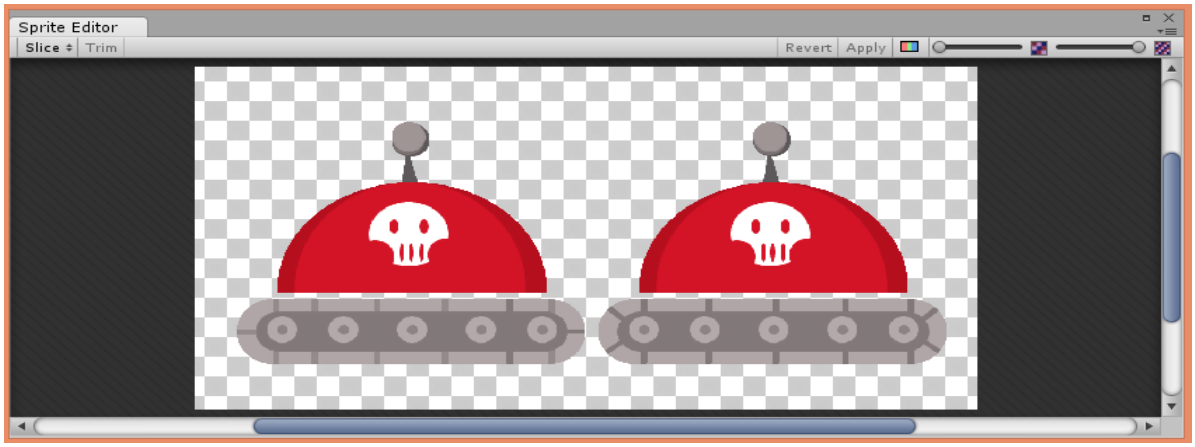
1.1 Importar sprites

A la carpeta **Assets/Textures/Player** li diem **Assets/Textures/Characters**, ja que a partir d'ara contindrà més personatges.

Importem al projecte el fitxer **EnemySpriteSheet.png**, de la mateixa manera que ho vem fer amb **playerSpriteSheet.png**, i el situem a la carpeta **Characters.**



Canviem l'**SpriteMode** de Single a **Multiple** i entrem a l'Sprite Editor.



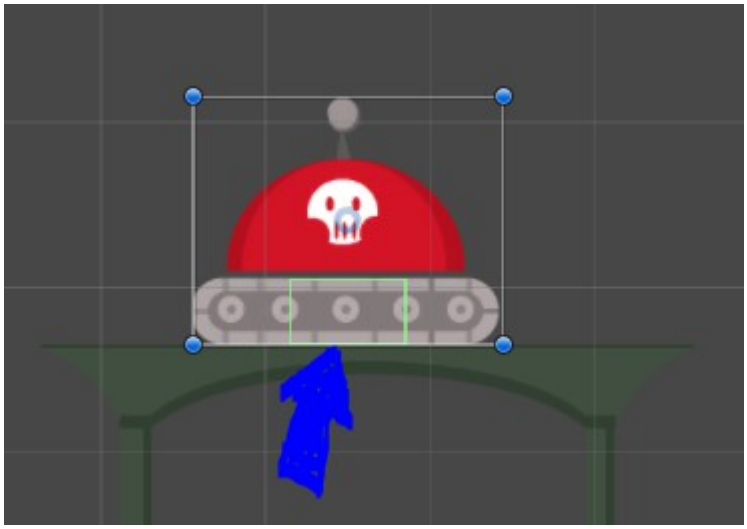
Retallem les dues imatges i els diem **EnemyWalk_1** i **EnemyWalk_2**. Marquem l'opció **Trim** per a eliminar els espais buits al voltant de l'sprite. Situem el **pivot** de les imatges en el **Bottom**.

Recordem polsar **Apply** abans de tancar l'editor d'sprites. Ara podem veure que l'sprite sheet **EnemySpriteSheet.png** té els sprites individuals que hem definit.

1.2 Crear el nou objecte del joc

Arrosseguem **EnemyWalk_1** a l'escena o a la Hierarchy per incloure el personatge al món. Com que sembla una mica massa gran, ajustem l'**scale** a X:0,41 i Y:0,41. Tambél li canviem el nom a **Enemy**.

Associem a **Enemy** un component **BoxCollider2D**. No cal que sigui gaire gran, només cal que cobreixi la part central de les cadenes del tanc. Li assignem **Size** X: 1,71 Y:0,95 i el centrem especificant **Center** a X:0 Y:0,49.



Associem a **Enemy** un component **RigidBody2D**. Establim **Continuous** com a valor de l'atribut **Collision Detection** i posem 30 com a valor de **Gravity Scale**. **Gravity Scale** determina la mesura en què la gravetat afecta a l'objecte. Per tant, com més gran sigui **Gravity Scale**, més pesat serà l'objecte.

Finalment, creem el nou script **EnemyControllerScript**, que associem a **Enemy**. Creem una versió inicial amb aquest contingut:

```
using UnityEngine;
using System.Collections;

public class EnemyControllerScript : MonoBehaviour
{
      public void switchDirections()
      {
      }
}
```

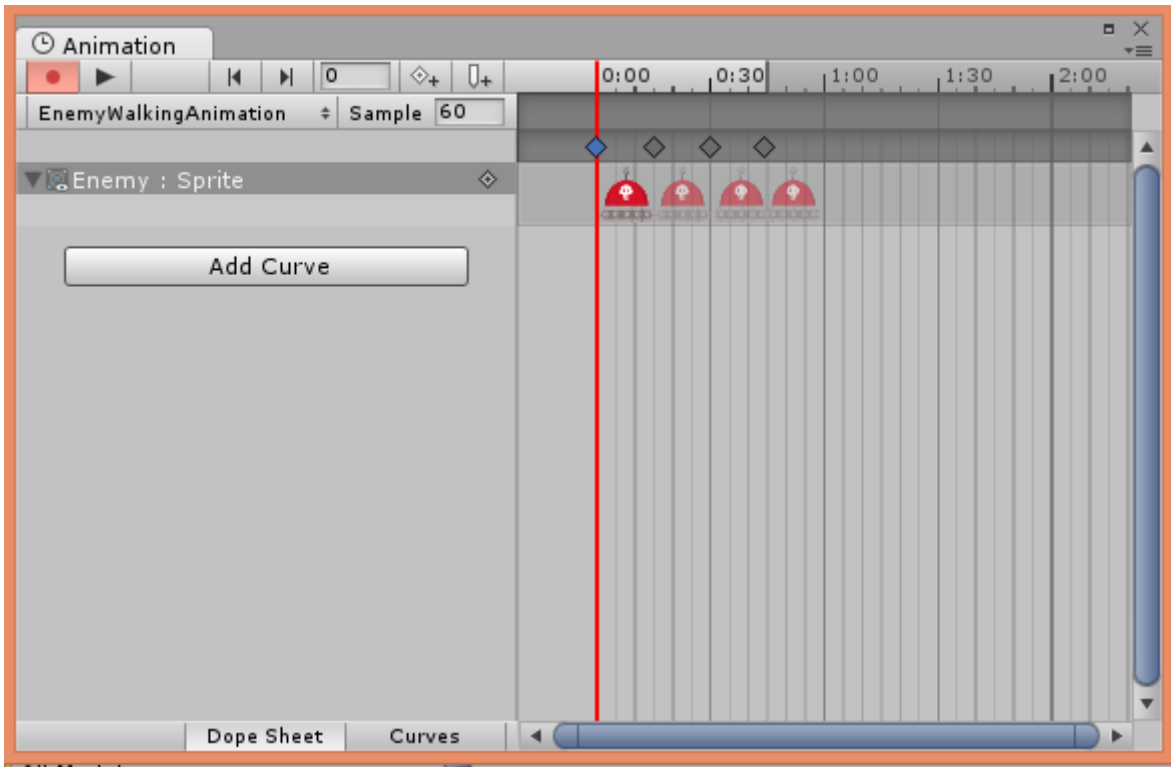
1.3 Crear l'animació del nou personatge

Per crear l'animació de l'enemic **seguirem el mateix procediment emprat quan es van crear les animacions del Player** (es aconsellable tenir a mà la documentació corresponent).

La nova animació l'anomenarem **EnemyWalkingAnimation**, i la guardarem en una nova carpeta **Animations/Enemy**.

El sistema crea una instància de la classe AnimatorController i l'anomena Enemy. Per evitar confusions, li canviem el nom a **EnemyAnimatorController**.

Es pot **experimentar** per trobar la **seqüència d'sprites** més adequada. En aquest document es pren una seqüència EnemyWalk_1, EnemyWalk_1, EnemyWalk_2 i EnemyWalk_2. També es pot experimentar amb el valor de **Sample**.



Comprovem que l'objecte Enemy té un nou **component Animator**. Com que només hi ha una animació, no cal gestionar transicions.

1.4 Moure el nou personatge. (Objectes de notificació)

El moviment que s'ha decidit donar al personatge Enemy és desplaçar-lo sobre una plataforma amunt i avall, d'un extrem a l'altre.

Per detectar que s'ha arribat a un extrem de la plataforma, utilitzem la tècnica de l'**objecte de notificació**: objectes fills d'Enemy (en aquest cas buits), situats als dos

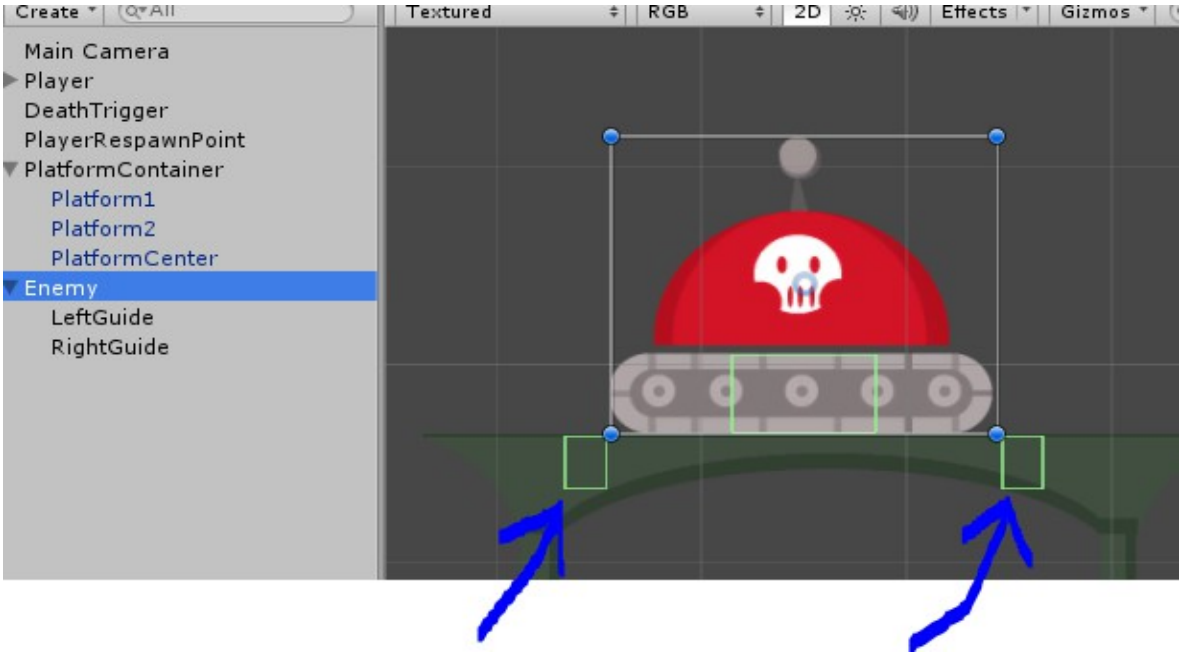
extrems. Quan un d'aquests objectes detecta que deixa de col·lisionar amb la plataforma, és senyal que Enemy ha arribat a l'extrem i és a punt de sortir de la plataforma.

Per implementar-ho:

1. Creem dos objectes buits, fills d'Enemy, que anomenem **LeftGuide** i **RightGuide**, i els situem respectivament a l'esquerra i a la dreta d'Enemy.

2. Assignem a cadascun dels nous objectes un **BoxCollider2D**, amb la propietat **IsTrigger** activada. Una mida adequada pot ser X:0,48 i Y:0.64.

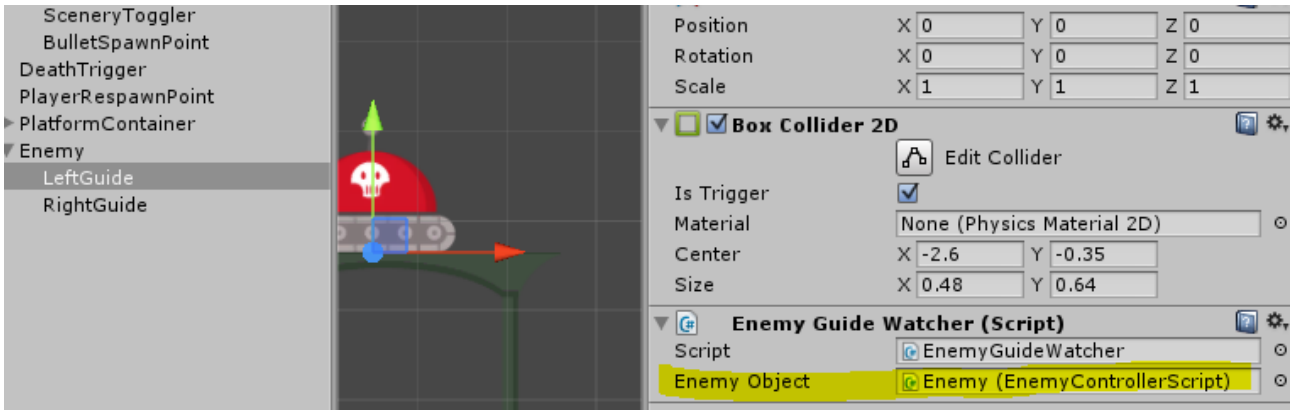
3. La situació adient de LeftGuide i RightGuide pot ser als costats de l'sprite Enemy:



Ara creem un script que anomenem **EnemyGuideWatcher** que gestionarà l'event **onTriggerExit2D** dels objectes LeftGuide i RightGuide.

```
using UnityEngine;
using System.Collections;

public class EnemyGuideWatcher : MonoBehaviour
{

   public EnemyControllerScript enemyObject = null;

   void OnTriggerExit2D( Collider2D otherObj )
   {
      // Aquest event es produeix quan Enemy està a punt
      //de sortir de la Platform.
      //Canviem la direcció del desplaçament

      if(otherObj.tag == "Platform")
            enemyObject.switchDirections();
   }
}
```

Hem d'associar aquest script als objectes **LeftGuide** i **RightGuide** i, des de l'Object Inspector, assignar l'objecte **Enemy** a l'atribut públic **enemyObject**.



Fins ara l'script **EnemyControllerScript** està buit, només conté el mètode **switchDirections()** buit, per poder compilar l'altre script **EnemyGuideWatcher**.

Ara és el moment d'escriure el seu codi:

```
using UnityEngine;
using System.Collections;

public class EnemyControllerScript : MonoBehaviour
{
      public float walkingSpeed = 0.45f;
      private bool walkingLeft = true;

(A)   void Start()
      {
      // Inicialitzar aleatòriament la direcció de desplaçament
            walkingLeft = (Random.Range(0,2) == 1);
            updateVisualWalkOrientation();
      }

(B)   void Update()
      {
      // Moure l'enemic segons la direcció actual de moviment
      // Es modifica la component x.
            if(walkingLeft)
            {
                  transform.Translate(new Vector3(walkingSpeed *
                  Time.deltaTime, 0.0f, 0.0f));
            }
            else
            {
                  transform.Translate(new Vector3((walkingSpeed * -1.0f)
                  * Time.deltaTime, 0.0f, 0.0f));
            }
      }

      public void switchDirections()
      {
      // Canviar la direcció de desplaçament a la contrària de l'actual
            walkingLeft = !walkingLeft;

      // Modificar l'orientació del gràfic associat a Enemy segons
```

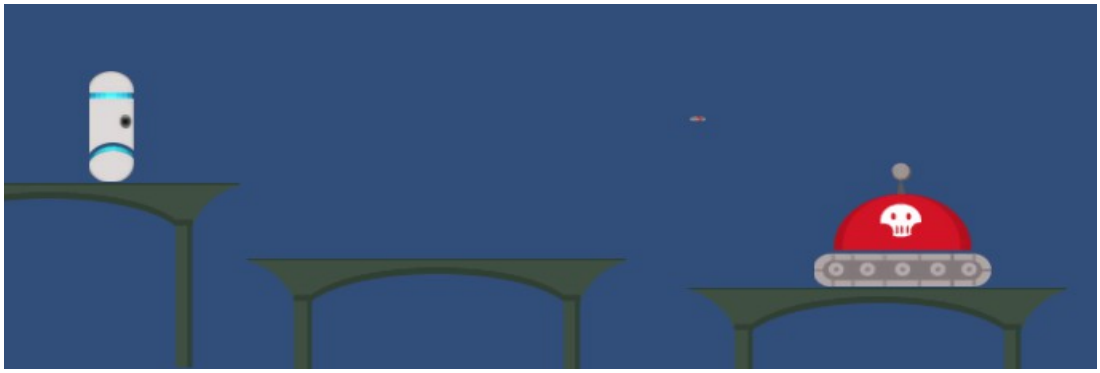
```
      // l'orientació actual (valor de walkingLeft)
            updateVisualWalkOrientation();
      }

(C)   void updateVisualWalkOrientation()
      {
            Vector3 localScale = transform.localScale;

            if(walkingLeft)
            {
                  if(localScale.x> 0.0f)
                  {
                        localScale.x = localScale.x * -1.0f;
                        transform.localScale = localScale;
                  }
            }
            else
            {
                  if(localScale.x< 0.0f)
                  {
                        localScale.x = localScale.x * -1.0f;
                        transform.localScale = localScale;
                  }
            }
      }
}
```

Comentaris al codi:

(A) Quan es crea l'objecte Enemy s'inicialitza aleatòriament el sentit en què es desplaçarà, generant pseudoaleatòriament un 0 o un 1. **Veure http://docs.unity3d.com/ ScriptReference/Random.Range.html.** S'adapta l'orientació de la imatge segons el sentit de desplaçament cridant el mètode **updateVisualWalkOrientation()**.
(B) A cada frame, s'actualitza la posició de l'enemic, segons el sentit en què es desplaça, modificant el component Transform. Es **multiplica la velocitat** de l'objecte (**walkingSpeed**) **pel temps** que ha passat des del frame anterior (**Time.deltaTime**) **i s'obté la posició en l'espai** on li correspon estar (espai = velocitat * temps).
(C) Per canviar l'orientació de la imatge, se li aplica una escala de signe contrari, tal com es va fer amb el Player.

EXPERIMENT

Executar el joc tal com està en aquest moment, amb els últims canvis. Comprovar que l'enemic que s'ha creat es mou com s'espera.

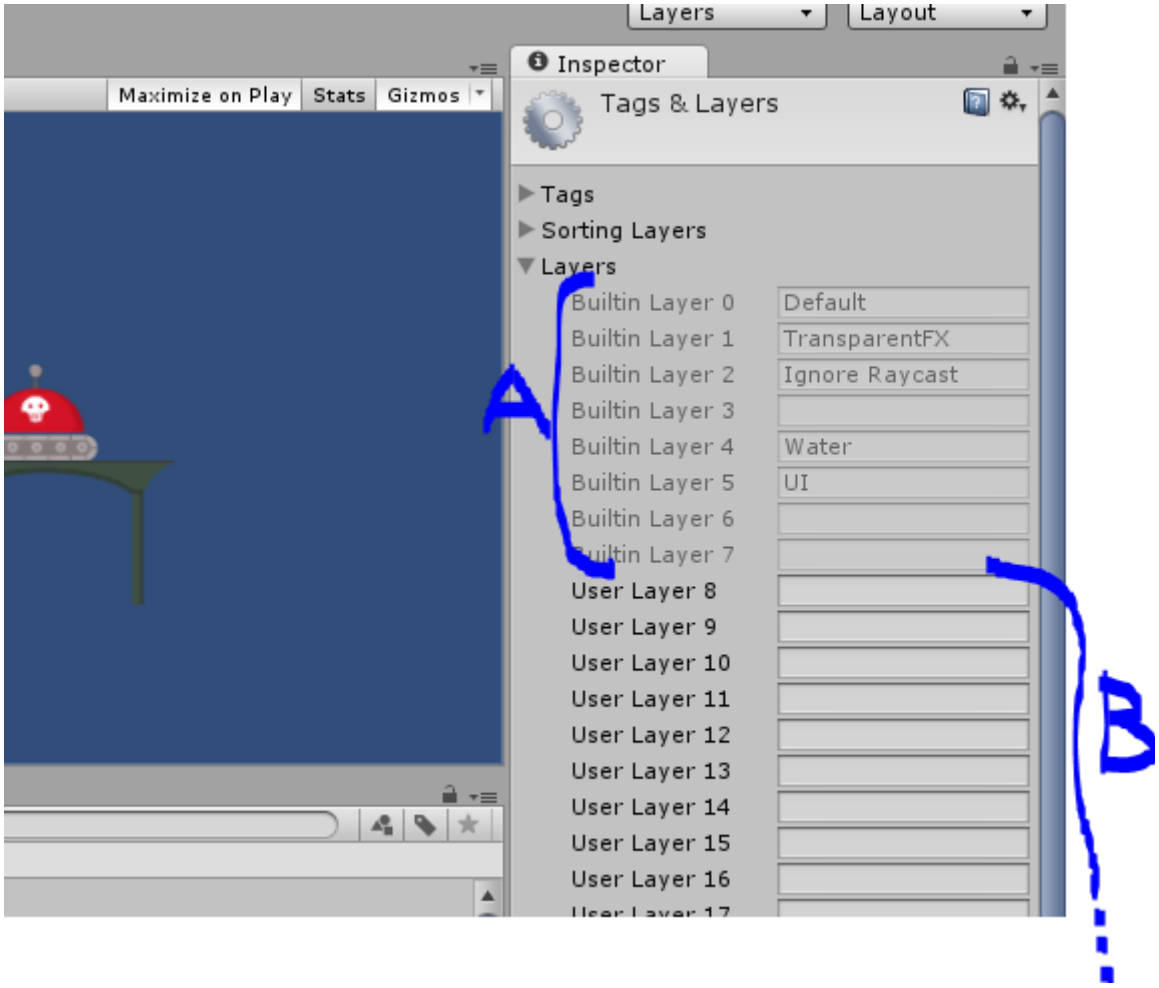


Si fem coincidir el Player i l'Enemy en el mateix lloc es pot comprovar que el sistema de col·lisions fa que els objectes es moguin i reaccionin d'una manera estranya.

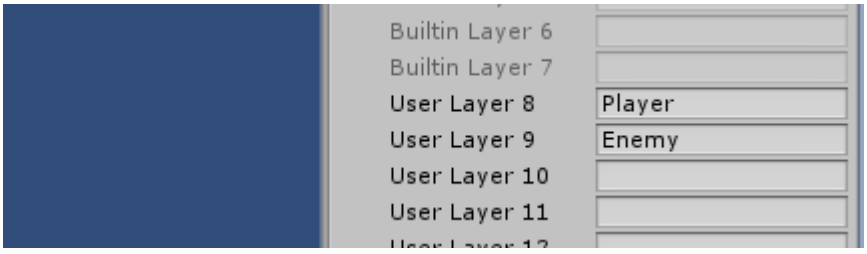
1.5 Evitar col·lisions no desitjades de Player i Enemy. (Layers. Layer Collision Matrix)

Per evitar les col·lisions no desitjades entre el **Player** i l'**Enemy** utilitzarem **layers** (capes) i la **màscara o matriu de col·lisions**.

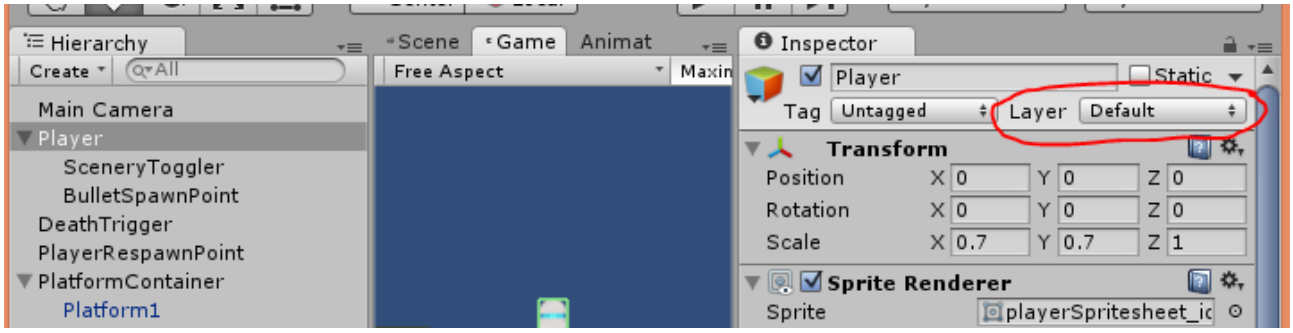
Obrim el panell Tags & Layers (**Edit > Project Settings > Tags and Layers** o des de l'Object Inspector, **Layers > Edit Layers**). Podem veure que hi ha unes capes predefinides per Unity **(A)** i unes altres disponibles per a l'usuari **(B)**.



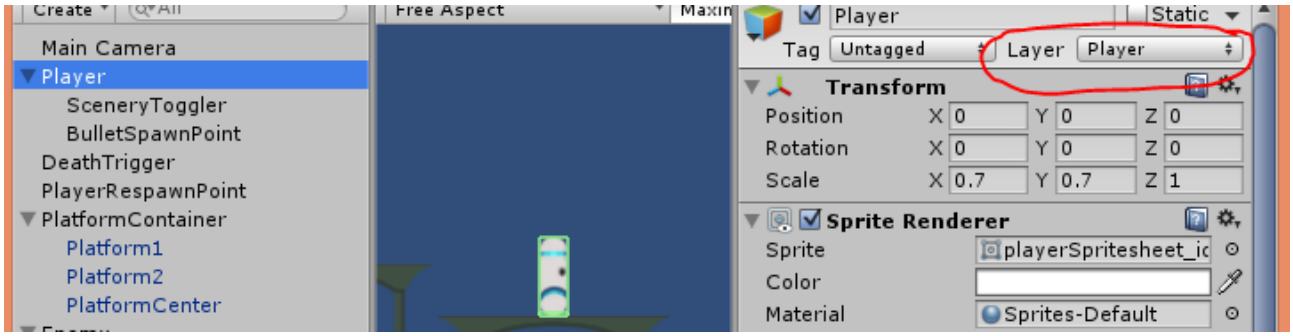
Definim una capa pel Player, que anomenem també "Player", i una altra capa per a l'Enemy, que anomenem "Enemy".



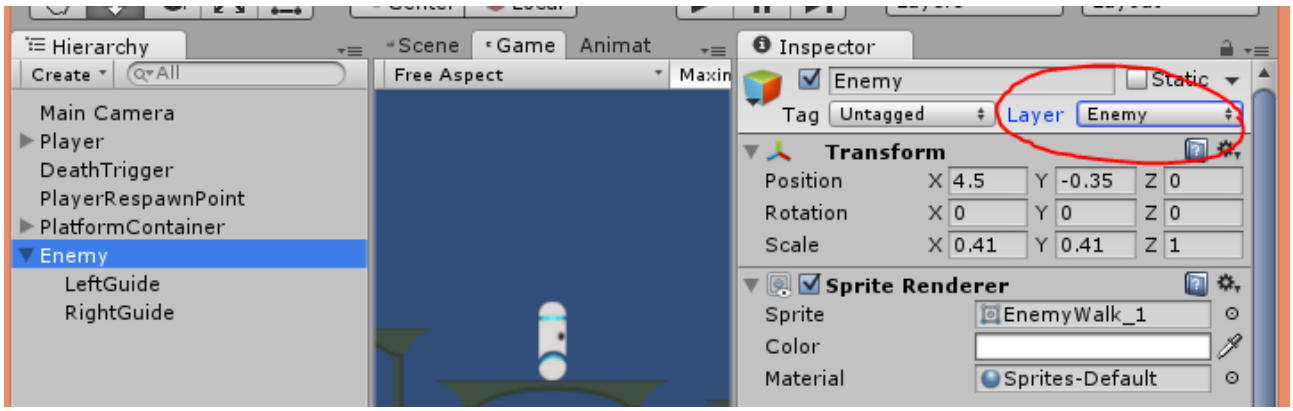
Si seleccionem l'objecte **Player**, podem veure que, de moment, es troba en la capa per defecte (**Default layer**)



Despleguem la llista de capes disponibles i seleccionem la capa "Player". En el moment de canviar el Player de capa, Unity pregunta si també volem canviar de capa els objectes fills de Player. Contestem que si, i ja tenim Player i tots el seus components en una nova capa.

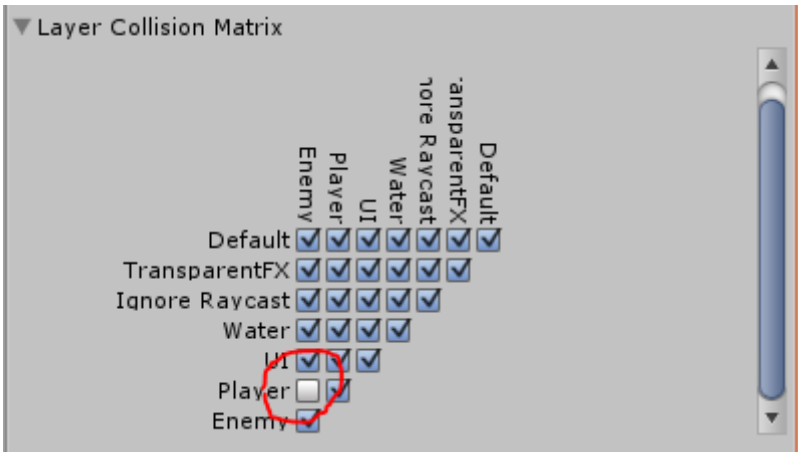


Fem el mateix amb Enemy i els seus objectes fills.



En aquest moment tenim l'objecte Player i els seus fills al layer "Player", l'objecte Enemy i els seus fills al layer "Enemy". La resta d'objectes continuen al layer "Default".

Seleccionem **Edit > ProjectSettings > Physics2D** per obrir la pestanya **Physics2DSettings**. A la part d'abaix hi ha la matriu de **col·lisions.** La modifiquem de manera que els objectes que són a la capa Player no col·lisionin amb els objectes que són a la capa Enemy.



EXPERIMENT

Executar el joc tal com està en aquest moment, amb els últims canvis. Comprovar que quan el Player i l'Enemy coincideixen en el mateix lloc no hi ha desplaçaments inesperats com abans.

Activitats

1. Fer un llistat amb els conceptes introduits en aquest document, indicar la pàgina i explicar què són i per a què s'utilitzen.