SESSIÓ 3.

Afegir enemics al joc

2. Donar a l'enemic capacitat de destrucció.

Donarem a l'Enemy la capacitat de destruir el Player quan entrin en contacte (quan es produexi una col·lisió).

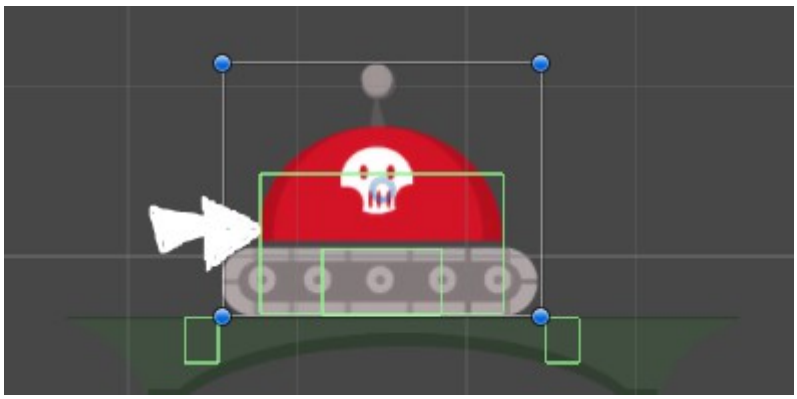
2.1 Crear objecte que detecti col·lisions

Per fer això afegirem un **nou objecte buit**, **fill de l'Enemy**, amb un component **BoxCollider2D**.

És interessant fer notar que, per tractar colisions amb característiques diferents, el millor és tenir diferents objectes fill amb el collider adequat. En jocs de l'estil de l'exemple que s'està desenvolupant, no és estrany que un personatge tingui tres o més àrees de col·lisió (BoxCollider2D pels peus, BoxCollider2D per defensa, BoxCollider2D per atac etc).

Creem l'objecte **DefenseCollider** i el fem **fill de** l'objecte **Enemy**, en la posició X: 0 Y: 0.29. (coordenades relatives a l'objecte principal).

Li afegim un component **BoxCollider2D** i fem que ocupi aproximadament **un quart de la superfície de l'Enemy**. Amb això volem aconseguir que l'Enemy pugui matar el Player però també volem que el Player tingui un marge per poder escapar-se.

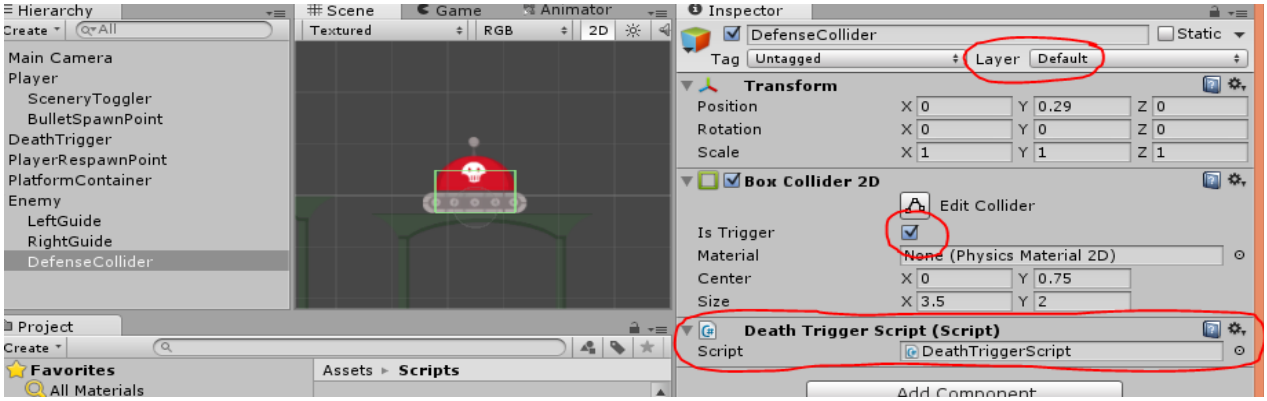


2.2 Fer reaccionar el Player en cas de col·lisió amb l'Enemy

Ara volem que la col·lisió del **Player** amb l'objecte **DefenseCollider** provoqui la mort del Player.

Recordem que ja hem programat un script que feia morir el Player en col·lisionar amb un objecte: l'script **DeathTriggerScript**. L'haviem utilitzat en la primera versió de l'objecte **DeathTrigger** (revisar l'apartat corresponent). Quan hem creat l'objecte **SceneryToggler**, hem passat a gestionar la mort del player per caiguda en l'script **PlayerColliderListener** i **DeathTriggerScript** s'ha deixat d'utilitzar. Ara és un bon moment per a recuperar-lo.

Només hem d'**activar la propietat IsTrigger** del component **BoxCollider2D** de **DefenseCollider** i associar-li l'script **DeathTriggerScript**.



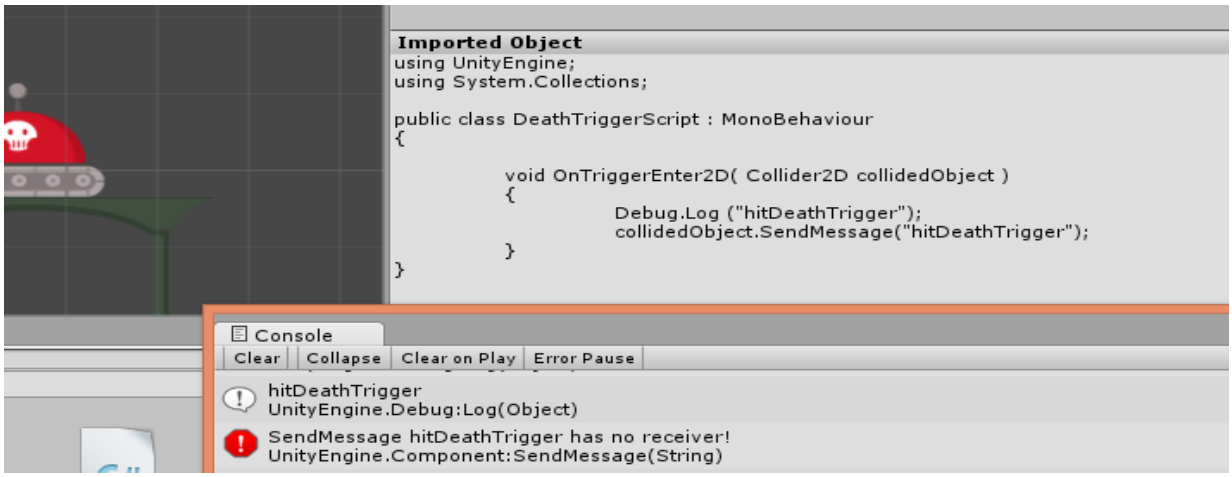
També ens em d'assegurar que **DefenseCollider** estigui en la capa per defecte, **Default layer**, i no en la capa **Enemy**. D'aquesta manera, el Player podrà col·lisionar amb el component DefenseCollider de l'Enemy, però no amb la resta de components.

EXPERIMENT

Executar el joc tal com està en aquest moment, amb els últims canvis.
Comprovar que el Player mor quan col·lisiona amb l'objecte DefenseCollider.
Observar els missatges de la consola.

2.3 Evitar missatges d'error innecessaris

A la imatge es mostra el codi de l'script **DeathTriggerScript** i uns missatges visualitzats a la consola (**Window > Console**)



Veiem que apareix un missatge de traça de l'execució, generat amb Debug.Log, que diu "hitDeathTrigger". Just després apareix un error que diu que el missatge hitDeathTrigger no té receptor.

Això passa perquè el missatge "hitDeathTrigger" es rebut per Player, que el gestiona amb el mètode hitDeathTrigger, i també pels **objectes fills de Player** que, **en no tenir implementat el mètode hitDeathTrigger**, provoquen l'error.

Com que hem trobat el motiu del missatge d'error i hem comprovat que no està passant res greu, evitarem que es generi aquest missatge. Així mantindrem la consola neta i preparada per a detectar de seguida els errors que puguin sortir.

Per evitar el missatge, a l'script **DeathTriggerScript** afegim un paràmetre en la crida al mètode SendMessage:

```
collidedObject.SendMessage("hitDeathTrigger",
                           SendMessageOptions.DontRequireReceiver);
```

Amb això fem que no es comprovi si algú tracta el missatge o no.

3. Fer l'Enemy vulnerable als trets del Player.

Per detectar que l'Enemy ha estat tocat per un projectil utilitzarem també el component **DefenseCollider**. Aquest serà un exemple de collider amb més d'una funció.

3.1 Detectar els impactes de projectil

Utilitzarem un sistema d'events similar al que utilitzem amb el Player: en resposta a un cert esdeveniment (col·lisió amb una bala), es genera un event (hitByBullet) que haurà de ser tractat per un mètode, que escolta aquest tipus d'event.

Crearem un nou script **TakeDamageFromPlayerBullet**, que associarem a l'objecte **DefenseCollider** (en aquest moment tindrà dos components script), i que contindrà el següent codi:

```
using UnityEngine;
using System.Collections;

public class TakeDamageFromPlayerBullet : MonoBehaviour
{
      //Definicio de delegat i event per tractar impacte de bala
   public delegate void hitByPlayerBullet();   (A)
   public event hitByPlayerBullet hitByBullet;  (B)

      //Tractament de col.lisio amb DefenseCollider: si
      //es amb una bala, generar event.
   void OnTriggerEnter2D (Collider2D collidedObject)  (C)
   {
      if(collidedObject.tag == "PlayerBullet")
      {
            if(hitByBullet != null)
                  hitByBullet();
      }
   }
}
```

Comentaris al codi:

(A) Definim un **delegate** anomenat **hitByPlayerBullet**, que representa una funció sense paràmetres i amb retorn void.
(B) Definim un **event** anomenat **hitByBullet**, que serà tractat per una funció amb la signatura definida pel **delegate hitByPlayerBullet**.
(C) El mètode **OnTriggerEnter2D** tracta el fet que un objecte (especificat pel paràmetre **collidedObject**) hagi entrat en col·lisió amb l'objecte que executa l'script (en aquest cas, l'objecte **DefenseCollider**). Com a tractament, **si s'ha col·lisionat amb una bala**, es genera un **event hitByBullet**.

Ara falta programar la funció, amb signatura igual a la de hitByPlayerBullet, que tracti l'event hitByBullet.

EXPERIMENT

Afegir al codi la generació de missatges de depuració que es mostra marcada en groc.

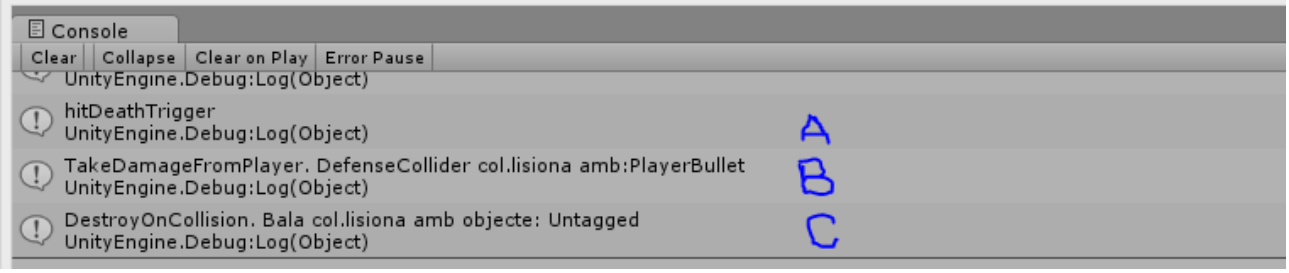
```
using UnityEngine;
using System.Collections;

public class DeathTriggerScript : MonoBehaviour
{

    void OnTriggerEnter2D( Collider2D collidedObject )
    {
        Debug.Log ("hitDeathTrigger");                    (A)
        collidedObject.SendMessage("hitDeathTrigger",
                                   SendMessageOptions.DontRequireReceiver);
    }
}
```

```
using UnityEngine;
using System.Collections;

public class TakeDamageFromPlayerBullet : MonoBehaviour
{
    //Definicio de delegat i event per tractar impacte de bala
    public delegate void hitByPlayerBullet();
    public event hitByPlayerBullet hitByBullet;

    //Tractament de col.lisio amb DefenseCollider: si
    //es amb una bala, generar event.
    void OnTriggerEnter2D (Collider2D collidedObject)
    {
        if(collidedObject.tag == "PlayerBullet")
        {
            Debug.Log ("TakeDamageFromPlayer. DefenseCollider col.lisiona amb:"   (B)
                       + collidedObject.tag);
            if(hitByBullet != null)
            {
                Debug.Log ("TakeDamageFromPlayer. Enemic Tocat!!!!!");      ?
                hitByBullet();
            }
        }
    }
}
```

```
using UnityEngine;
using System.Collections;

public class DestroyOnCollision : MonoBehaviour
{
    void OnTriggerEnter2D(Collider2D hitObj)
    {
        Debug.Log ("DestroyOnCollision. Bala col.lisiona amb objecte: " + hitObj.tag);   (C)
        DestroyObject(gameObject);
    }
}
```

Executar el joc tal com està en aquest moment, amb els últims canvis.

Quan disparem sobre l'enemic, obtenim aquest missatges en la finestra de Consola:



Veiem que el missatge "Enemic Tocat !!!!!", marcat amb un interrogant, no hi apareix. Això és perqué ningú s'ha subscrit encara per escoltar l'event **hitByBullet**, i la condició `(hitByBullet != null)` és falsa.

3.2 Tractar l'event hitByBullet

Ara farem que l'script **EnemyControllerScript** escolti i tracti l'event **hitByBullet**. Per això hi afegim el codi descrit a continuació.

Primer de tot, al començament d'**EnemyControllerScript**, afegim una referència a la classe **TakeDamageFromPlayerBullet**:

```
public TakeDamageFromPlayerBullet bulletColliderListener = null;
```

Des de l'Object Inspector hem d'assignar l'objecte **DefenseCollider** a aquest camp **bulletColliderListener**.

Després, en el mètode **OnEnable**, subscribim com a "escoltador" de l'event **hitByBullet** el mètode **hitByPlayerBullet** (que implementarem tot seguit).

```
void OnEnable()
{
      // Suscripció a l'event hitByBullet.
      bulletColliderListener.hitByBullet += hitByPlayerBullet;
}
```

També, en el mètode **OnDisable**, anul·lem la subscripció com a "escoltador" de l'event **hitByBullet**.

```
void OnDisable()
{
      // cancel.lar la suscripció a l'event hitByBullet.
      bulletColliderListener.hitByBullet -= hitByPlayerBullet;
}
```

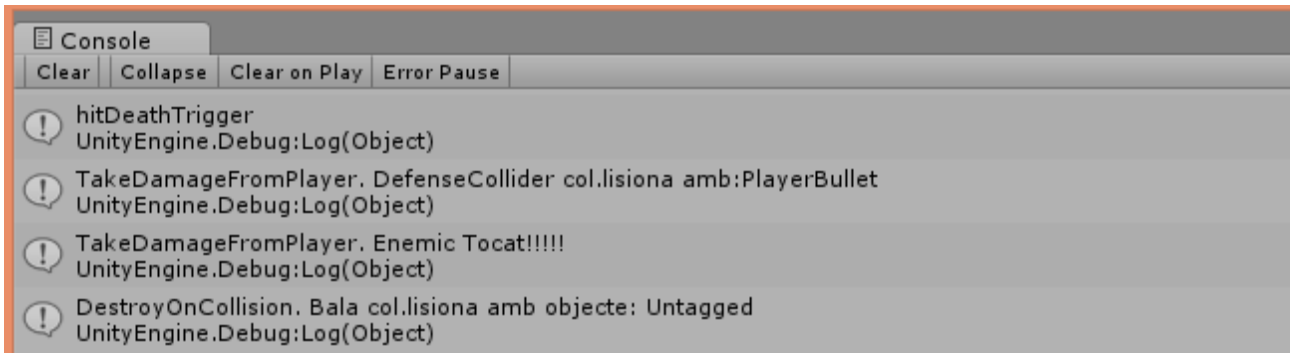
Finalment, implementem el mètode **hitByPlayerBullet** que s'executarà com a tractament de l'event **hitByBullet**.

```
public void hitByPlayerBullet()
{
      // Esperar un moment i destruir l'objecte Enemy
      Destroy(gameObject,0.1f);
}
```

Utilitzem el mètode Destroy indicant una espera d'un segon abans de realitzar la destrucció. Veure **http://docs.unity3d.com/ScriptReference/Object.Destroy.html**

EXPERIMENT

Executar el joc tal com està en aquest moment, amb els últims canvis.
Comprovar que l'enemic queda destruit quan rep un tret i que, ara sí, apareix en la Consola el missatge "Enemic Tocat!!!!".



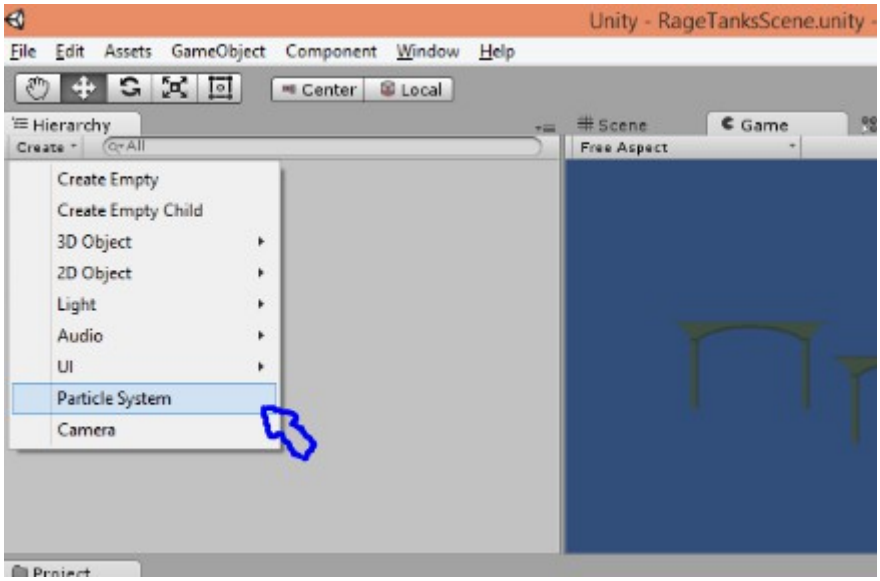
4. Simular l'explosió de l'Enemy. Sistemes de partícules.

Hem aconseguit que els projectils del Player destruixin l'enemic, però l'efecte visual és una mica estrany: l'enemic simplement desapareix.

Afegirem un efecte d'explosió utilitzant un **sistema de partícules** (particle system). En Unity , els sistemes de partícules s'utilitzen per crear fum, foc, pols i efectes especials que requereixen un nombre gran d'objectes similars.

La documentació oficial **http://docs.unity3d.com/es/current/Manual/ParticleSystems.html**

Creem un sistema de partícules i l'anomenem **EnemyDeathFX**.



Convertim aquest emissor de partícules en un **Prefab** que també anomenem **EnemyDeathFX**.

Ara ampliem l'script **EnemyControllerScript** per afegir l'efecte visual a la destrucció de **l'Enemy**:

Primer de tot afegim un atribut públic:

```
public ParticleSystem deathFxParticlePrefab = null;
```

i completem el mètode **hitByPlayerBullet**:

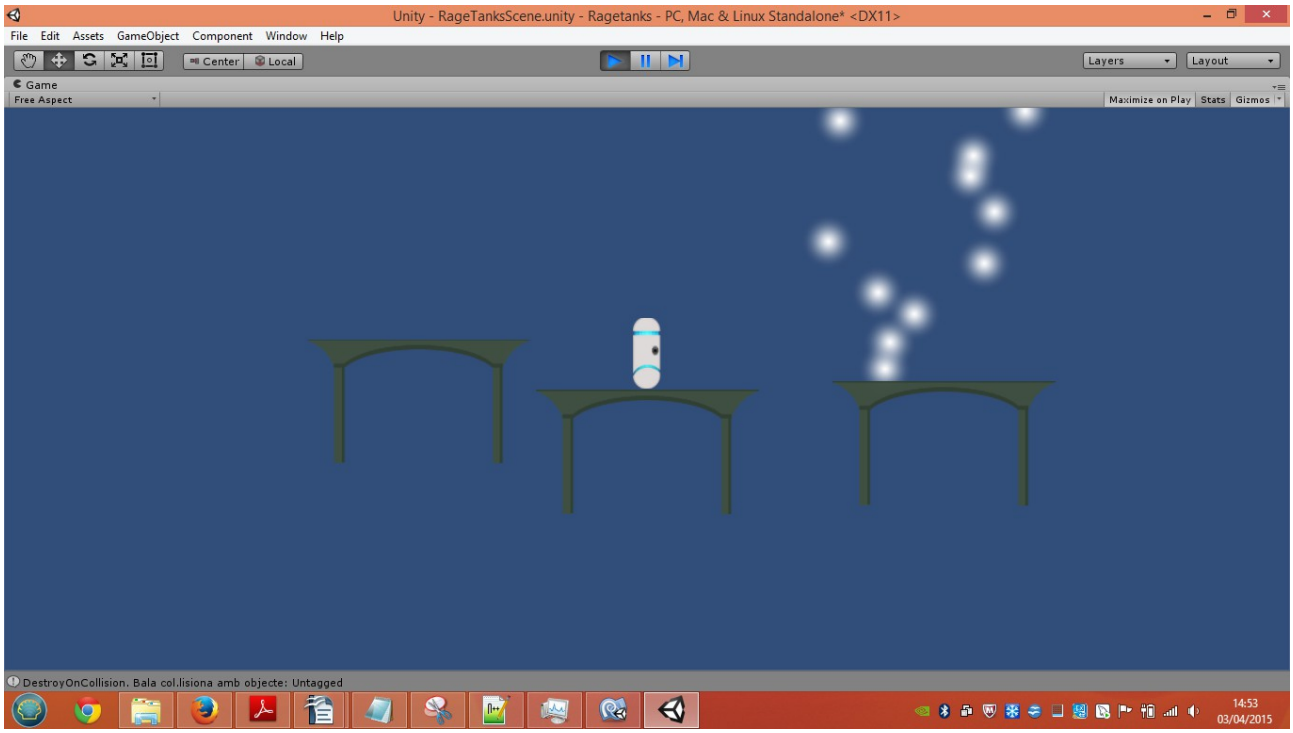
```
public void hitByPlayerBullet()
{
      // Crear l'objecte emissor de partícules
      ParticleSystem  deathFxParticle =
                  (ParticleSystem) Instantiate(deathFxParticlePrefab);
```

```
        // Obtenir la posició de l'enemic
        Vector3 enemyPos = transform.position;

        // Crear un nou vector davant de l'enemic (incrementar component z)
        Vector3 particlePosition =
                        new Vector3(enemyPos.x,enemyPos.y,enemyPos.z + 1.0f);
        // Posicionar l'emissor de partícules en aquesta nova posició
        deathFxParticle.transform.position = particlePosition;


        // Esperar un moment i destruir l'objecte Enemy
        Destroy(gameObject,0.1f);
}
```

Des de l'Object Inspector, hem d'assignar el prefab **EnemyDeathFX** a l'atribut **deathFxParticlePrefab** de l'script **EnemyControllerScript**, associat a l'objecte **Enemy**.

EXPERIMENT

Executar el joc tal com està en aquest moment, amb els últims canvis.
Comprovar que l'enemic queda destruit quan rep un tret i es produeix una emissió de partícules.

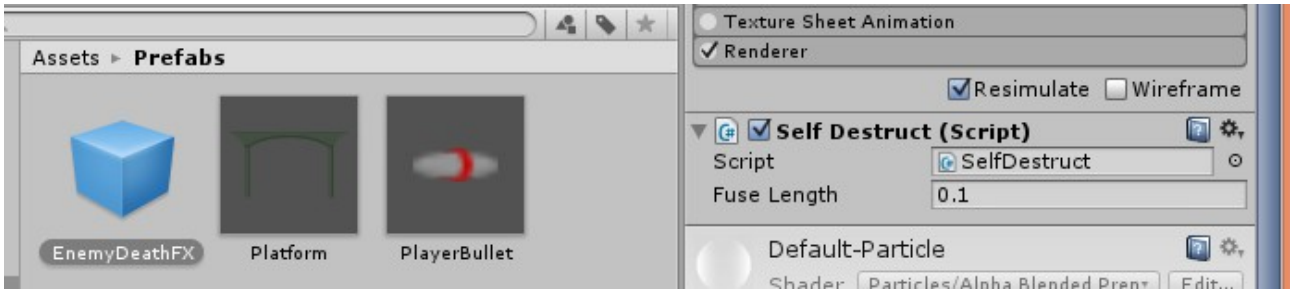


El defecte més evident d'aquesta solució és que el núvol de partícules no desapareix.

Anem a gestionar-ho amb un nou script que anomenem **SelfDestruct**

```
using UnityEngine;
using System.Collections;

public class SelfDestruct : MonoBehaviour
{
	public float fuseLength = 0.1f; // llargada de la metxa
	private float destructTime = 0.0f;

	void Start()
	{
		//Quan es crea l'objecte, establir moment de la destrucció
		destructTime = Time.time + fuseLength;
	}

	void Update()
	{
		//A cada frame, mirar si ha arribat el moment de la destrucció.
		if(destructTime<Time.time)
			Destroy(gameObject);
	}
}
```

i que associarem al prefab **EnemyDeathFX**

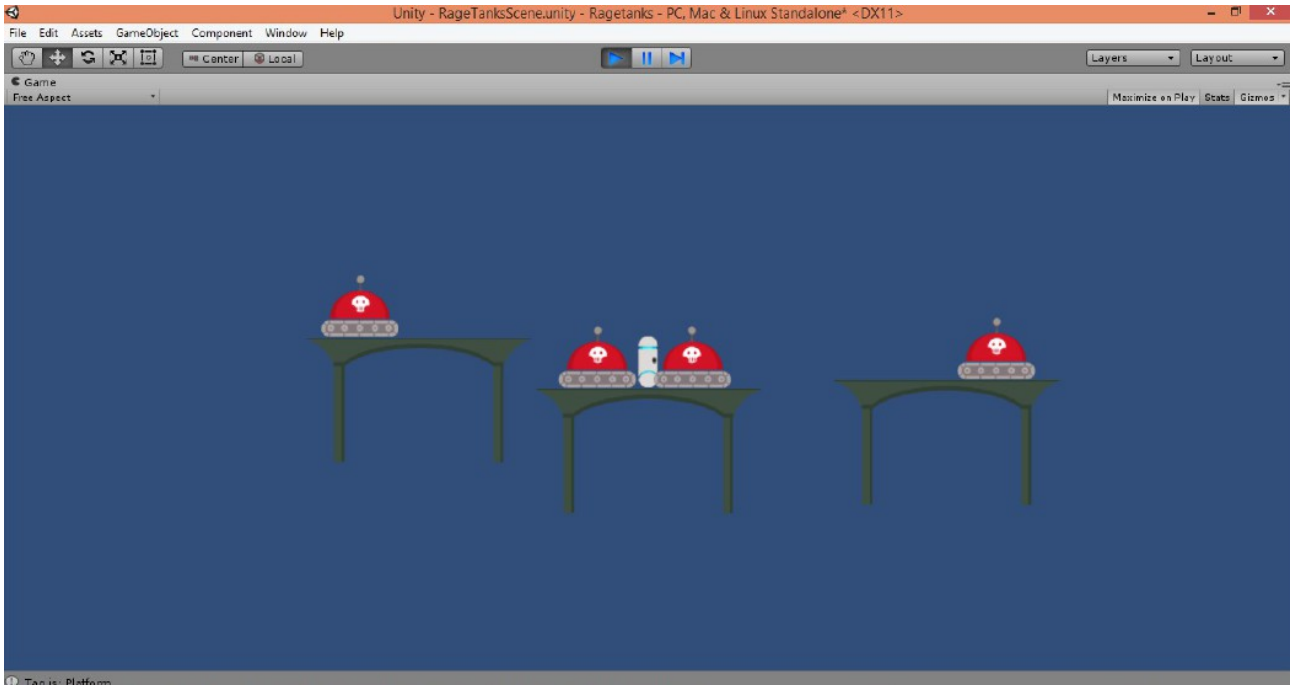


Des de l'Object Inspector es pot **donar un valor adequat a Fuse Lenght** (el temps de vida de les partícules). Es pot provar amb el valor 1,25.

5. Guardar com plantilla la definició de l'Enemy.

Quan tinguem el personatge de l'enemic completament definit, el guardem com a Prefab. (Abans poder fer uns últims ajustos, com reduir una mica la seva mida...)

Ara podem instal·lar un enemic en cada plataforma i provar el joc.



Per mantenir-ho tot endreçat, creem un objecte buit que anomenem **EnemyContainer** i hi situem com a fills tots els objectes Enemy, de la mateixa manera que hem guardat totes les plataformes dins de l'objecte **PlatformContainer**.

EXPERIMENT

Instal·lar dos tancs en la mateixa plataforma i veure què passa. Pensar com fer-ho perquè els tancs no col·lisionin entre ells.

EXPERIMENT

En l'exemple hem utilitzat les partícules que Unity proporciona per defecte quan es crea un sistema de partícules.

Es proposa seguir atentament el video
http://singletechgames.com/2014/01/16/tutorial-en-espanol-de-unity-2d-6-version-4-3/

i millorar l'efecte d'explossió del joc. Es proporcionen sprites per poder fer-ho.