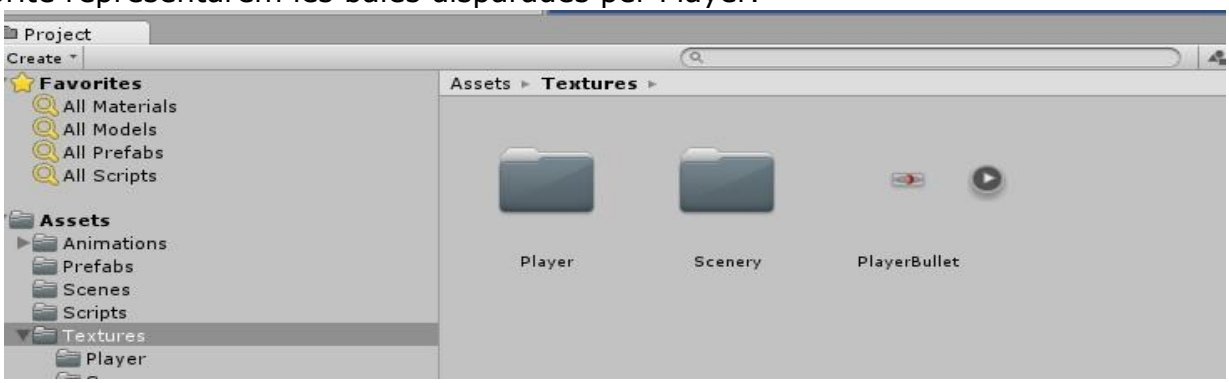
SESSIÓ 2. (4)

Ampliar les capacitats del protagonista

8. Disparar.

8.1 Crear els projectils

Primer de tot importarem el fitxer **PlayerBullet.png** a la carpeta Textures. Amb aquest sprite representarem les bales disparades per Player.



Els següents passos ja els hem vist en tractar altres objectes del joc:

- Crearem l'**objecte del joc** anomenat **PlayerBullet**, amb la imatge anterior associada.
- Li assignarem un component **BoxCollider2D**, queha de ser **trigger**.
- Li assignem un component **RigidBody2D. Saps dir com hauria de ser Gravity Scale**, per evitar que caigui a terra.
- Creem un nou **tag** anomenat **PlayerBullet** i l'assignem com a tag de l'objecte PlayerBullet.

Un cop definit l'objecte, amb les propietats bàsiques, anem a gestionar el seu comportament per mitjà de l'script **PlayerBulletController**. (Recordar assignar-lo a l'objecte PlayerBullet !)

```
using UnityEngine; using
System.Collections;

public class PlayerBulletController : MonoBehaviour
{
      //Se li assignara un objecte automaticament, en crear una bala a
PlayerStateListener public GameObject
      playerObject = null; public float
      bulletSpeed = 15.0f;

      public void launchBullet() { // Volem que el Player dispari
      cap al costat al que mira.
            // Aixo ens ho indica el component "local scale"
```

```
            float mainXScale = playerObject.transform.localScale.x;
            Vector2 bulletForce;
            if(mainXScale< 0.0f)
            {
                  // Disparar cap a l'esquerra
                  bulletForce = new Vector2(bulletSpeed * -1.0f,0.0f);
            }
            else
            {
                  // Disparar cap a la dreta
                  bulletForce = new Vector2(bulletSpeed,0.0f);
            }
            rigidbody2D.velocity = bulletForce;
      }
}
```

Comentaris al codi:

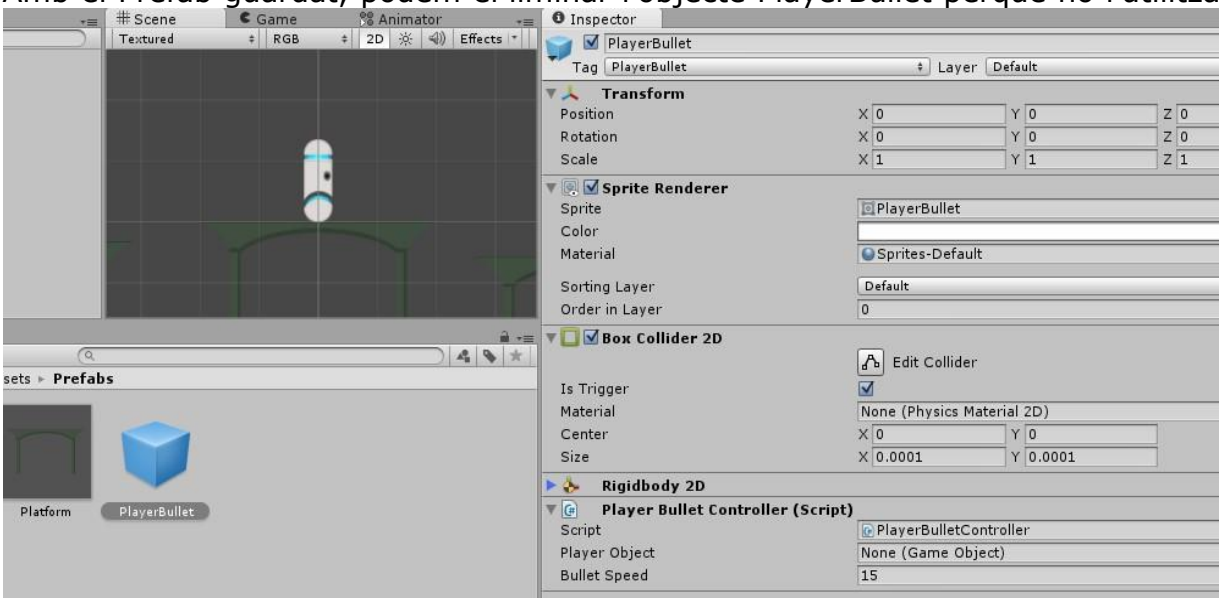
Aconseguim el moviment del projectil assignant un valor a l'atribut **Velocity** del **RigidBody2D** de la bala. (Anteriorment, en la implementació de l'estat **jump** hem modificat la velocitat d'un RigidBody aplicant-li una força. Fer-ho d'una manera o altra depèn de l'efecte que volguem obtenir).

Volem que el Player dispari cap al costat al que mira. Per saber-ho, es pot gestionar una variable que ho indiqui en cada moment. Però, en el nostre cas, podem aprofitar la manera en que canviem l'orientació del Player cap a la dreta o l'esquerra.

Recordem que els sprites originals del Player només presenten la figura "mirant cap a la dreta". Per fer-lo mirar cap a l'esquerra, li hem aplicat una "local scale" horitzontal negativa. **(Revisar el mètode onStateCycle() a l'script PlayerStateListener)**

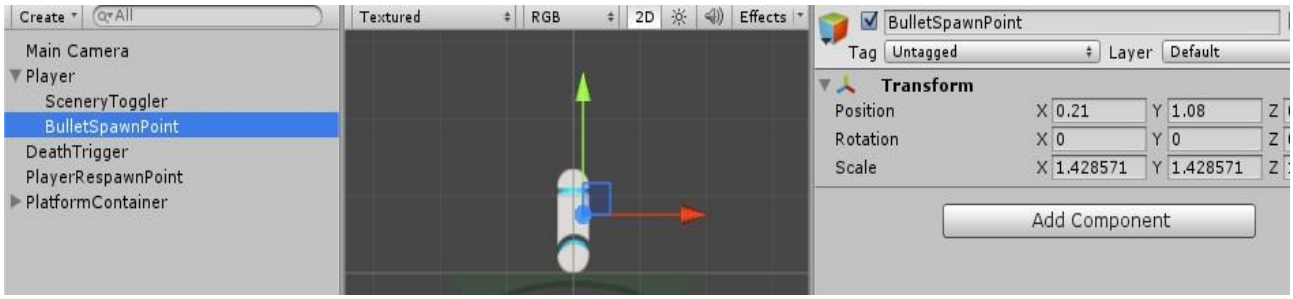
Per això sabem que, si playerObject.transform.localScale.x és positiva el player va cap a la dreta i, si és negativa, que va cap a l'esquerra.

Una vegada tenim el projectil completament definit, creem un **Prefab** per poder-ne fabricar tants com en necessitem.
Amb el Prefab guardat, podem el·liminar l'objecte PlayerBullet perquè no l'utilitzarem.



8.2 Definir en el Player l'orígen dels projectils

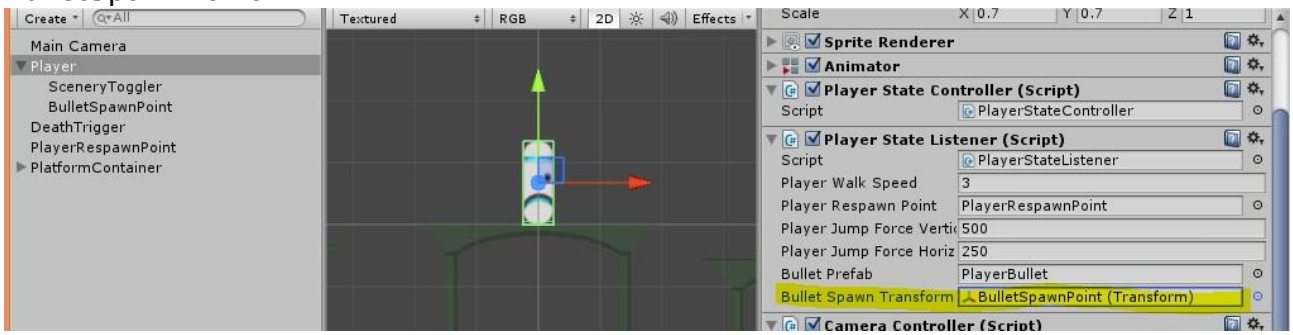
Anem a definir un nou objecte que serà el punt de creació de projectils: **BulletSpawnPoint**. Serà un altre "**Empty Object**" del joc, sense imatge associada i **fill de l'objecte Player**. Li assignem la posició X:0,21 Y:1,08, de manera que quedi situat aproximadament on el Player té dibuixat el punt negre.



A l'script PlayerStateListener hi afegim el següent atribut públic:

```
public Transform bulletSpawnTransform;
```

Des de la pestanya de l'Object Inspector li assignem el nou objecte **BulletSpawnPoint**. Amb aquest atribut tindrem disponible, a dins de l'script, el component Transform de BulletSpawnPoint.



8.3 Implementar l'estat "disparar" del Player

A continuació, considerarem un **nou estat** del Player: **firingWeapon**.

El Player passarà a aquest quan dispari. Immediatament després de disparar, retornarem el Player a l'estat anterior, sigui el que sigui.

Per gestionar un nou estat hem de fer les ampliacions en el codi que són habituals:

1. Ampliar l'enumeració d'estats definida a PlayerStateController

```
//Definicio dels diferents estats del player
public enum playerStates
{ idle = 0, left,
      right, jump,
      landing,
      falling,
      kill,
      resurrect,
      firingWeapon,
      _stateCount
}
```

Comentaris al codi:
A més d'incloure el nou estat **firingWeapon**, també hem afegit el nou element **_stateCount,** que deixarem sempre al final. El compilador assigna internament a cada element de l'enumeració un codi enter consecutiu, a partir de 0. Per això, _stateCount sempre indicarà quants estats hi ha a l'enumeració: si tenim n estats, numerats de 0 a (n – 1), _stateCount valdrà n. La utilitat d'aquest element s'aclarirà més endevant.

---

2. Tractar l'entrada de l'usuari a l'script PlayerStateCotroller

Unity ofereix el axis Fire1, Fire2 i Fire3 per recollir l'ordre de disparar.
Utilitzarem l'axe **Fire1** (que té associat per defecte la tecla **Control Esquerre**), afegint el següent codi en el mètode **LateUpdate** de l'script **PlayerStateCotroller**:

```
//Disparar
float firing = Input.GetAxis("Fire1");

if(firing > 0.0f)
{ if(onStateChange != null)
      onStateChange(PlayerStateController.playerStates.firingWeapon);
}
```

Comentaris al codi:
Quan es detecta que l'usuari polsa **Fire1**, passem el Player a estat **firingWeapon** i el codi de tractament d'aquest estat s'encarregarà de disparar.

---

3. Adaptar l'script PlayerStateListener

3.1: Nous camps de **dades** (de fet, alguns ja els tenim, però fins ara no eren necessaris):

Per gestionar el retorn a l'estat anterior necessitem els camps

```
private PlayerStateController.playerStates previousState;
private PlayerStateController.playerStates currentState;
```

que s'actualitzen al final del mètode onStateChange:

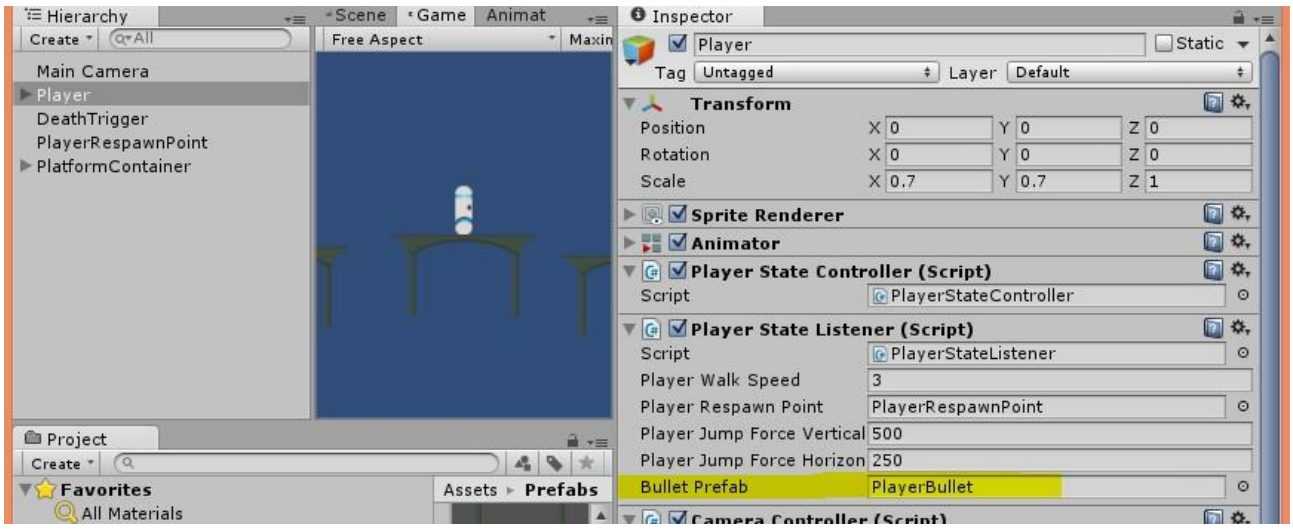
```
// Guardar estat actual com a estat previ previousState =
currentState;

// Assignar el nou estat com a estat actual del player
currentState = newState;
```

Per crear les bales necessitem un nou atribut públic de la classe:

```
public GameObject bulletPrefab = null;
```

Comprovem que, quan seleccionem el Player, a l'Object Inspector apareix el nou atribut públic BulletPrefab en el component PlayerStateListener. Llavors li assignem el **Prefab PlayerBullet** que hem generat abans.



3.2: Afegir el "case" corresponent al nou estat **firingWeapon** on és habitual:

3.2.1: **mètode onStateChange** (s'invoca una vegada en canviar el Player a estat **firingWeapon**). Afegim el tractament

```
case PlayerStateController.playerStates.firingWeapon:

                    // Construir l'objecte bala a partir del Prefab
                GameObject newBullet = (GameObject)Instantiate(bulletPrefab);

                    // Establir la posicio inicial de la bala creada
                    //(la posicio de BulletSpawnTransform)
            newBullet.transform.position = bulletSpawnTransform.position;
```

```
                        // Agafar el component PlayerBulletController de la bala
      //que s'ha creat
PlayerBulletController bullCon =
            newBullet.GetComponent<PlayerBulletController>();

      // Assignar a l'atribut playerObject de l'script
      //PlayerBulletController el Player bullCon.playerObject = gameObject;

      // Invocar metode que dispara la bala bullCon.launchBullet();

      // Despres de disparar, tornar a l'estat previ.
onStateChange(currentState); break;
```

Comentaris al codi:
Es crea una bala i es situa a la mateixa posició que l'objecte BulletSpawnPoint (que hem assignat des de l'Inspector a l'atribut públic **bulletSpawnTransform**). Això s'aconsegueix igualant els components **Transform.position**.

Des d'un objecte es pot accedir als components d'un altre objecte del joc per mitjà del mètode **GetComponent**. En aquest script, accedim al component PlayerBulletController (un script) de PlayerBullet. Ho fem per assignar a l'atribut públic **playerObject** l'objecte Player. En aquest context **gameObject** fa referència a l'objecte que està executant l'script (es a dir, Player).
Veure http://docs.unity3d.com/ScriptReference/Component-gameObject.htm

---

3.2.2 **mètode checkForValidStatePair** (s'invoca dins de **onStateChange** per comprovar que es pot passar al nou estat des de l'actual). Afegim el tractament del nou estat:

```
      case PlayerStateController.playerStates.firingWeapon:
            returnVal = true;
            break;
```

per permetre anar a qualsevol estat des de **firingWeapon**.

D'altra banda, s'ha de permetre disparar (passar a estat **firingWeapon**) des de tots els estats, excepte **kill** i **resurrect**. Els estats **Idle**, **Left** i **Right** ja permeten anar a qualsevol altre. Però en el "**case**" dels estats **jump**, **landing** i **falling**, en el "**if**" que comprova si la transició es acceptable, afegirem la condició

```
|| newState == PlayerStateController.playerStates.firingWeapon
```

3.2.3 **mètode checkIfAbortOnStateCondition** (s'invoca dins de **onStateChange** per comprovar si s'ha d'abortar el canvi d'estat.

Aquest estat no té tractament especial, però és aconsellable tenir previst el tractament de tots els estats. Per això hi afegim:

```
            case PlayerStateController.playerStates.firingWeapon:
                  break;
```

EXPERIMENT

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

Comprovar que el Player pot disparar en els estats previstos i de manera coherent amb el sentit de desplaçament.