

## SESSIÓ 2. (1)

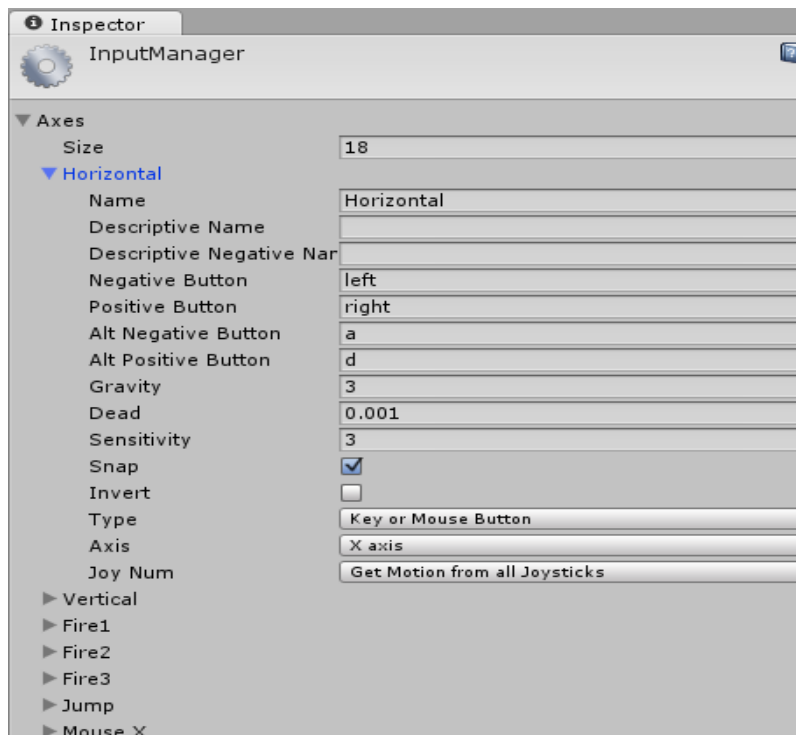
### Ampliar les capacitats del Player.

#### 1. Interactuar amb el protagonista.

Ara donarem al Player la capacitat de desplaçar-se per l'escena.

Si anem a **Edit > Project Settings > Input** podem veure a l'Object Inspector les associacions de tecles amb accions que proporciona Unity per defecte. Unity emmagatzema les associacions de tecles en diferents "**axes**" (eixos), cadascun dels quals pot tenir un valor en coma flotant.

Per exemple veiem que l'axe Horizontal, per defecte té associades les tecles fletxa dreta i fletxa esquerra. El valor disminueix amb la fletxa esquerra i al contrari.



Aquesta configuració es pot modificar perquè el joc suporti input de diferents dispositius com ara joysticks.

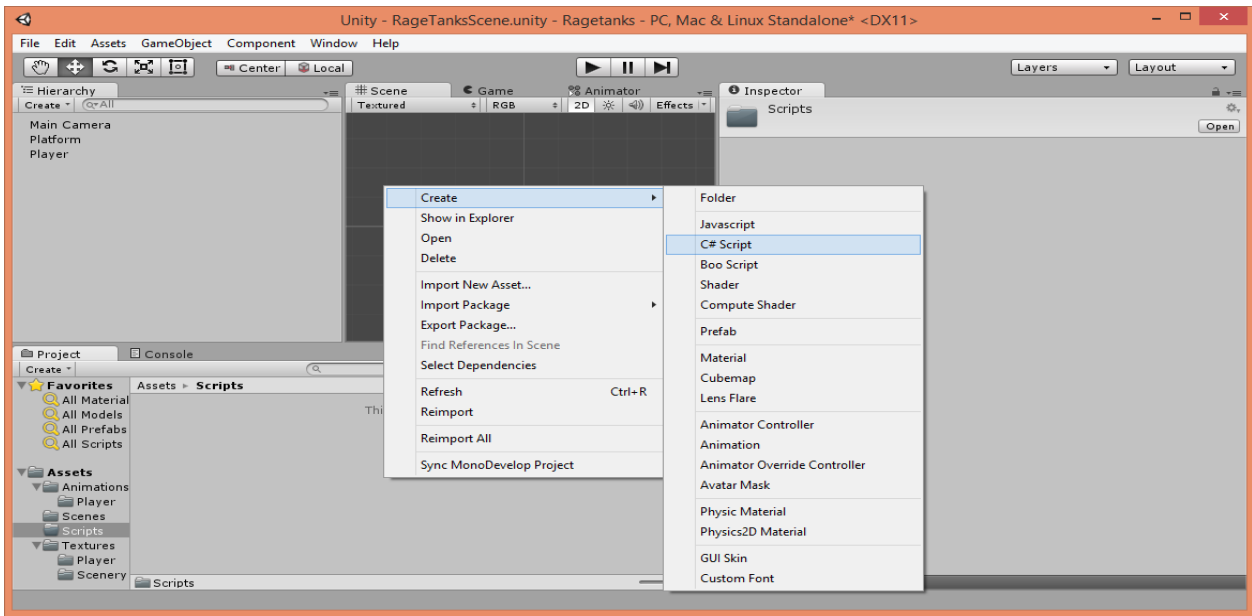
Unity permet afegir codi i programar certs comportaments del joc. Això afegeix molta potència a l'eina. Els llenguatges de programació admesos són C#, JavaScript i Boo, però C# és el més habitual, amb molta diferència, i és el que utilitzarem en aquests documents.

El codi es pot escriure amb qualsevol editor senzill com Notepad++ o amb l'editor d'un IDE com VisualStudio, però Unity porta incorporat un IDE anomenat

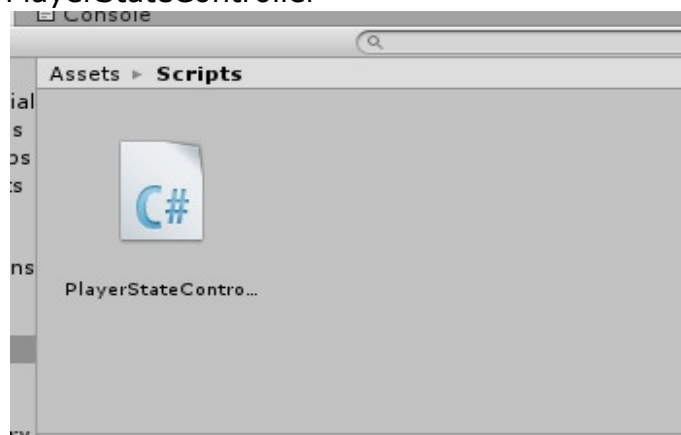
**MonoDevelop**, que està associat per defecte als fitxers font C#. Per obrir-lo només cal donar doble clic sobre el fitxer font en el Object Panel.

Anem a crear un Script per poder escoltar l'input que ve de l'axe horitzontal. Si és negatiu, desplaçem cap a l'esquerra el Player, si és positiu, cap a la dreta.

Creem una nova carpeta en el projecte, que anomenem Scripts i creem un nou script en C# (botó dret, Create, C# Script).



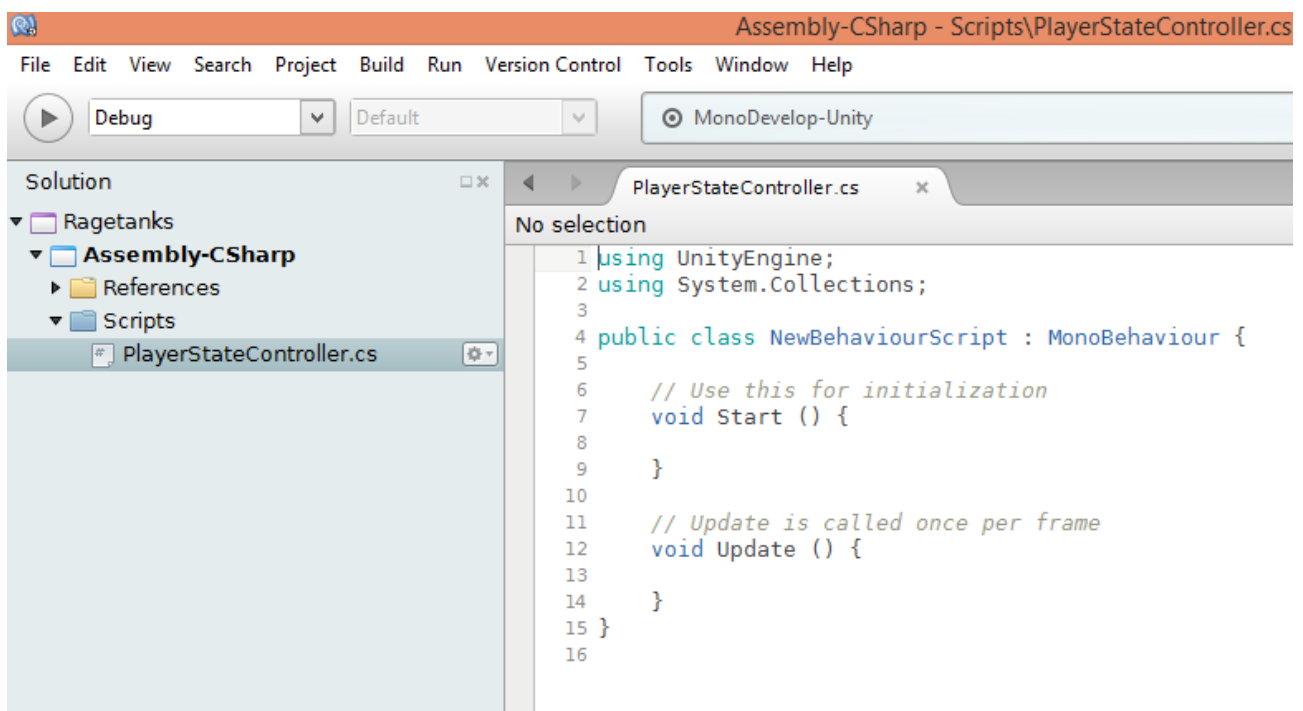
Anomenem l'script PlayerStateController



Fem doble click a l'script (o, amb l'script seleccionat, botó "Open" des de l'Inspector) per obrir l'entorn MonoDevelop,



en el qual programarem els scripts de Unity.



Quan anem a escriure un nou script en C#, es genera una plantilla buida amb l'estructura més habitual per aquest tipus de codi. Normalment es defineix una classe que hereda de **MonoBehaviour**, que és una classe predefinida en la API de Unity. El resultat de compilar un script d'aquest tipus és un **component** que pot ser arrossegat a/des del Project Panel i formar part d'un objecte.

Normalment, els scripts s'associen a un objecte del joc i llavors es poden veure a l'Object Inspector, com un component més de l'objecte.

**(Nota: l'equivalent en C# de la paraula clau "extends" (i també de "implements") de Java són els dos punts)**

Els components d'un objecte poden rebre i manejar els events de l'objecte. De la classe MonoBehaviour s'hereten unes quantes funcions (mètodes) que poden ser

sobreescrites en les classes derivades, per tal d'adaptar la resposta a aquests events.

Les funcions que apareixen en la plantilla són **Start()** i **Update()**.

- La funció **Start()** es crida una vegada quan apareix en escena l'objecte que té associat l'script. En el cas d'objectes creats en temps de disseny i que sempre són presents, **Start()** s'executa en començar l'escena.
- La funció **Update()** es crida una vegada **per frame**, és a dir, unes quantes vegades per segon. Per això, aquesta funció és apropiada per programar el moviment o les característiques que canvien en cada objecte al llarg del temps.

N'hi ha més, documentades a <http://docs.unity3d.com/Manual/EventFunctions.html>.

La API de Unity és una biblioteca de classes a disposició dels desenvolupadors per facilitar-los la feina. La referència oficial és a <http://docs.unity3d.com/ScriptReference/> i, per exemple, a <http://docs.unity3d.com/ScriptReference/MonoBehaviour.html> trobem la documentació de la classe **MonoBehaviour**.

Seguint amb l'activitat, anomenem la nova classe **PlayerStateController**. Afegim el codi següent:

```
using UnityEngine;
using System.Collections;
public class PlayerStateController : MonoBehaviour{

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

//Definició del delegate playerStateHandler
    public delegate void playerStateHandler(PlayerStateController.playerStates newState);

//Definició d'event onStateChange i assignació de onStateChange com a EventHandler
    public static event playerStateHandler onStateChange;

// Aquest metode es crida despres de Update() a cada frame.
    void LateUpdate (){

        // Recollir l'input actual en el Horizontal axis (eix horitzontal)
        float horizontal = Input.GetAxis("Horizontal");

        //Tractar segons el valor de l'input recollit
        if(horizontal != 0f)
        {
            //Hi ha algun moviment: canviar l'estat del protagonista a left o right
            if(horizontal < 0f)
            {
                if(onStateChange != null) onStateChange(PlayerStateController.
                    playerStates.left);
            }
            else
```

```

        {
            if(onStateChange != null) onStateChange(PlayerStateController.
                                                playerStates.right);
        }
    }
    else
        //No hi ha cap moviment: canviar l'estat del protagonista a idle
    {
        if(onStateChange != null) onStateChange(PlayerStateController.
                                                playerStates.idle);
    }
}
}
}

```

## Comentaris al codi:

Hi apareixen un quants dels estats que està previst que tingui el player, encara que de moment només se n'utilitzen tres. S'implementa amb una **enumeració**.

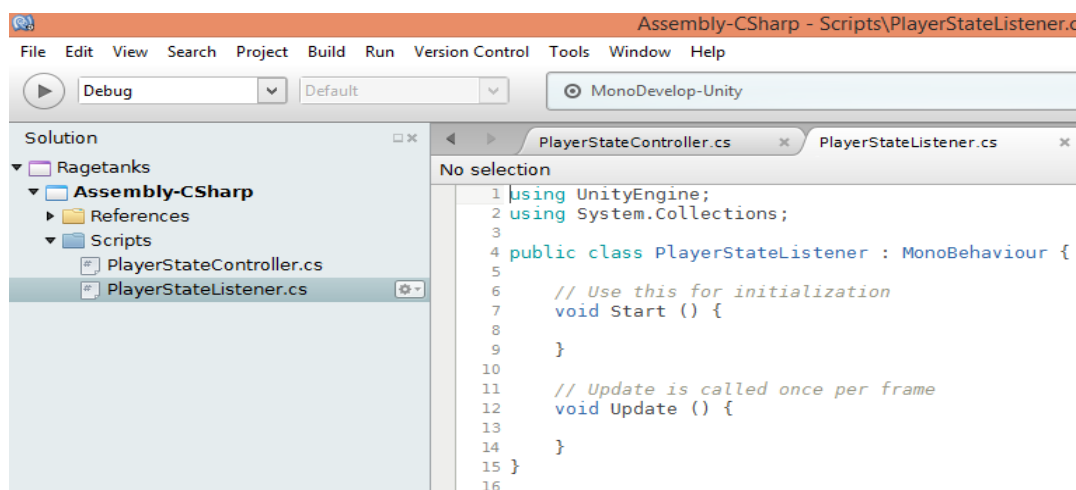
Es **llegeix l'input d'un dels eixos predefinit** a Unity amb Input.GetAxis. Es **genera un event onStateChange** amb l'estat al qual s'ha de passar com a paràmetre.

Es fa servir el mecanisme Event + Delegate que servirà perquè qualsevol objecte del joc pugui ser informat dels canvis d'estat del player. És un mecanisme molt potent i que facilita la programació, sobre tot quan hi ha molts objectes que han de saber com evoluciona la situació del joc.

Aquí es generen els events i en el següent script es programa el codi necessari per escoltar-los i tractarlos.

Compilem (Build) en MonoDevelop i tornem a Unity.

Ara s'ha de programar un altre script que escolti per detectar els canvis d'estat i sàpiga què fer en cada cas. L'anomenem **PlayerStateListener**.



Hi carreguem el codi següent:

```

using UnityEngine;
using System.Collections;

[RequireComponent(typeof(Animator))]
public class PlayerStateListener : MonoBehaviour
{
    public float playerWalkSpeed = 3f;

    private Animator playerAnimator = null;
    private PlayerStateController.playerStates previousState =
PlayerStateController.playerStates.idle;
    private PlayerStateController.playerStates currentState =
PlayerStateController.playerStates.idle;

    //Aquest mètode de MonoBehaviour s'executa cada vegada que s'activa l'objecte
associat a l'script.
    //L'objecte s'apunta a escoltar l'event onStateChange: afegeix la funcio
onStateChange a la llista de
    //handlers (manegadors) de l'event PlayerStateController.onStateChange. Amb això,
cada vegada que
    //es generi un event PlayerStateController.onStateChange, el sistema passara el
control a la funcio
    //onStateChange (i, seqüencialment, a totes les funcions que s'hagin afegit a la
llista de handlers
    //d'aquest event)

    void OnEnable()
    {
        PlayerStateController.onStateChange += onStateChange;
    }

    //Aquest mètode de MonoBehaviour s'executa cada vegada que es desactiva l'objecte
associat a l'script.
    //Es deixa d'escoltar l'event onStateChange
    void OnDisable()
    {
        PlayerStateController.onStateChange -= onStateChange;
    }

    void Start()
    {
        playerAnimator = GetComponent<Animator>();
    }

    void LateUpdate()
    {
        onStateCycle();
    }

    // Processar l'estat en cada cicle
    void onStateCycle()
    {
        // Guardar l'actual localScale de l'objecte (és al component Transform de
l'objecte)
        Vector3 localScale = transform.localScale;
        transform.localEulerAngles = Vector3.zero;

        switch(currentState)
        {
            {
                case PlayerStateController.playerStates.idle:
                    break;

                case PlayerStateController.playerStates.left:
                    //moure cap a l'esquerra modificant la posició
                    transform.Translate(new Vector3((playerWalkSpeed * -1.0f) *
Time.deltaTime, 0.0f, 0.0f));

```

```

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

        break;

    case PlayerStateController.playerStates.right:
        //moure cap a la dreta modificant la posició
        transform.Translate(new Vector3(playerWalkSpeed * Time.deltaTime,
0.0f, 0.0f));

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

        break;

    case PlayerStateController.playerStates.jump:
        break;

    case PlayerStateController.playerStates.landing:
        break;

    case PlayerStateController.playerStates.falling:
        break;

    case PlayerStateController.playerStates.kill:
        break;

    case PlayerStateController.playerStates.resurrect:
        break;
    }
}

// onStateChange es crida sempre que canvia l'estat del player
public void onStateChange(PlayerStateController.playerStates newState)
{
    // Si l'estat actual i el nou són el mateix, no cal fer res
    if(newState == currentState)
        return;

    // Comprovar que no hi hagi condicions per abortar l'estat
    if(checkIfAbortOnStateCondition(newState))
        return;

    // Comprovar que el pas de l'estat actual al nou estat està permès. Si no ho
està, no es continua.
    if(!checkForValidStatePair(newState))
        return;

    // Realitzar les accions necessàries en cada cas per canviar l'estat.
    // De moment només es gestionen els estats idle, right i left
    switch(newState)
    {
    case PlayerStateController.playerStates.idle:
        playerAnimator.SetBool("Walking", false);
        break;

    case PlayerStateController.playerStates.left:
        playerAnimator.SetBool("Walking", true);
        break;

    case PlayerStateController.playerStates.right:

```

```

        playerAnimator.SetBool("Walking", true);
        break;

    case PlayerStateController.playerStates.jump:
        break;

    case PlayerStateController.playerStates.landing:
        break;

    case PlayerStateController.playerStates.falling:
        break;

    case PlayerStateController.playerStates.kill:
        break;

    case PlayerStateController.playerStates.resurrect:
        break;
}

// Guardar estat actual com a estat previ
previousState = currentState;
// Assignar el nou estat com a estat actual del player
currentState = newState;
}

// Comprovar si es pot passar al nou estat des de l'actual.
// Es tracten diversos estats que encara no estan implementats, perquè el
// codi sigui més il·lustratiu
bool checkForValidStatePair(PlayerStateController.playerStates newState)
{
    bool returnVal = false;

    // Comparar estat actual amb el candidat a nou estat.
    switch(currentState)
    {
    case PlayerStateController.playerStates.idle:
        // Des de idle es pot passar a qualsevol altre estat
        returnVal = true;
        break;

    case PlayerStateController.playerStates.left:
        // Des de moving left es pot passar a qualsevol altre estat
        returnVal = true;
        break;

    case PlayerStateController.playerStates.right:
        // Des de moving right es pot passar a qualsevol altre estat
        returnVal = true;
        break;

    case PlayerStateController.playerStates.jump:
        // Des de Jump només es pot passar a landing o a kill.
        if(
            newState == PlayerStateController.playerStates.landing
            || newState == PlayerStateController.playerStates.kill
        )
            returnVal = true;
        else
            returnVal = false;
        break;

    case PlayerStateController.playerStates.landing:
        // Des de landing només es pot passar a idle, left o right.
        if(
            newState == PlayerStateController.playerStates.left
            || newState == PlayerStateController.playerStates.right
            || newState == PlayerStateController.playerStates.idle
        )

```



```

        returnVal = true;
    else
        returnVal = false;
    break;

case PlayerStateController.playerStates.falling:
    // Des de falling només es pot passar a landing o a kill.
    if(
        newState == PlayerStateController.playerStates.landing
        || newState == PlayerStateController.playerStates.kill
    )
        returnVal = true;
    else
        returnVal = false;
    break;

case PlayerStateController.playerStates.kill:
    // Des de kill només es pot passar resurrect
    if(newState == PlayerStateController.playerStates.resurrect)
        returnVal = true;
    else
        returnVal = false;
    break;

case PlayerStateController.playerStates.resurrect :
    // Des de resurrect només es pot passar Idle
    if(newState == PlayerStateController.playerStates.idle)
        returnVal = true;
    else
        returnVal = false;
    break;
}
return returnVal;
}

// Aquesta funció comprova si hi ha algun motiu que impedeixi passar al nou estat.
// De moment no hi ha cap motiu per cancel·lar cap estat.

bool checkIfAbortOnStateCondition(PlayerStateController.playerStates newState)
{
    bool returnVal = false;

    switch(newState)
    {
    case PlayerStateController.playerStates.idle:
        break;
    case PlayerStateController.playerStates.left:
        break;
    case PlayerStateController.playerStates.right:
        break;
    case PlayerStateController.playerStates.jump:
        break;
    case PlayerStateController.playerStates.landing:
        break;
    case PlayerStateController.playerStates.falling:
        break;
    case PlayerStateController.playerStates.kill:
        break;
    case PlayerStateController.playerStates.resurrect:
        break;
    }
    // Retornar True vol dir 'Abort'. Retornar False vol dir 'Continue'.
    return returnVal;
}
}

```

## Comentaris al codi:

Tot aquest codi implementa un mecanisme d'estats, amb el qual es **gestiona l'estat del protagonista** en funció dels events que es van generant.

El moviment es fa per mitjà de la funció **Translate** de la classe **transform**, que conté la posició, rotació i escala de l'objecte. També es té en compte la propietat **localScale**, per gestionar correctament el canvi d'escala que pugui haver-se aplicat a l'objecte. Veure <http://docs.unity3d.com/ScriptReference/Transform.html>

Quan el jugador utilitza les tecles de moviment, es genera un event que provoca que s'acabi cridant el mètode **onStateChange(newState)**, on es comprova que la transició d'un estat a un altre té sentit i, si és correcte, es canvia l'animació i l'estat del protagonista.

Per canviar l'animació es canvia el valor de la variable booleana **Walking**, que hem creat en definir les animacions.

En el moviment de l'sprite, que es fa amb el mètode **transform.Translate** s'indica, per mitjà d'un objecte **Vector3**, el canvi de posició en cadascun dels eixos de coordenades.

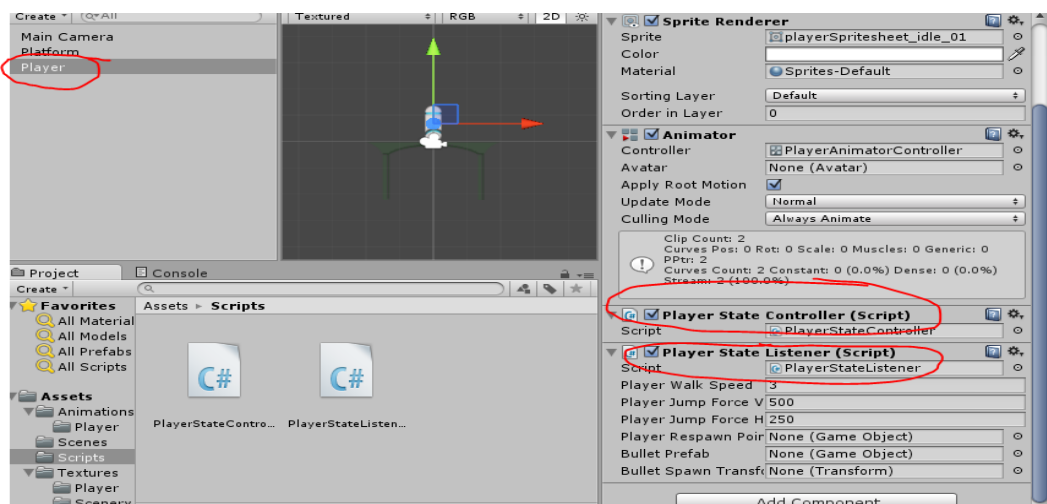
Si ens fixem en el SpriteSheet del protagonista, veiem que els dibuixos originals presenten la figura mirant sempre cap a la dreta. Aquesta orientació va bé pel desplaçament cap a la dreta, però no pel desplaçament contrari.

Per canviar l'orientació, juguem amb l'**escala de l'objecte**: aplicant una escala negativa, aconseguim la imatge especular de la figura original.

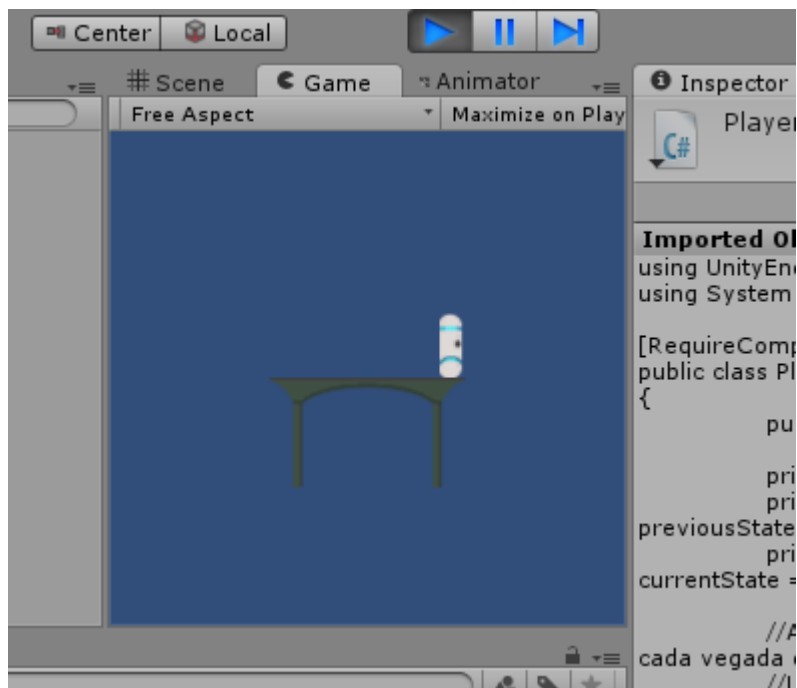
Primer de tot s'obté l'escala que ja s'està aplicant (**transform.localScale**) i s'emmagatzema en un objecte **Vector3**. Llavors, quan el canvi d'estat impliqui canvi de direcció en el desplaçament, la **localScale** es multiplica per -1, si convé.

**Val la pena revisar el codi, llegir els comentaris i seguir les crides que es fan.**

Finalment s'han d'**associar els scripts** **PlayerStateController** i **PlayerStateListener** a l'objecte **Player**.



Ara ja podem executar el joc i comprovar que el protagonista es mou cap a la dreta o cap a l'esquerra, segons si es polsa la tecla fletxa\_dreta (o tecla A) o la tecla fletxa\_esquerra (o tecla D).



## Activitats

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

### 2. Experiment

Incorporar el moviment cap amunt del Player. Utilitzar l'estat **jump**. Convé entendre (de moment, a grans trets) com funciona el codi proposat.

No cal que el resultat sigui perfecte (ni tan sols que funcioni...més endavant ja es veurà com fer-ho). L'objectiu és provar de modificar el codi i prendre contacte amb MonoDevelop + Unity.