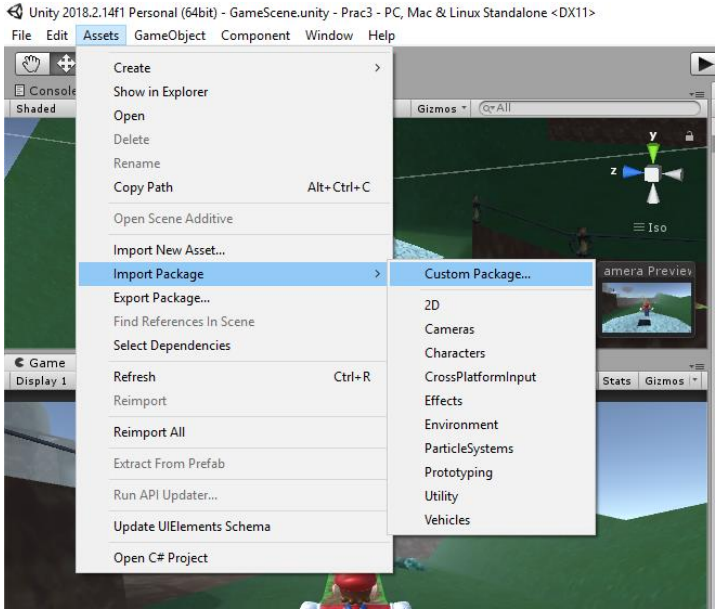
DESENVOLUPAMENT DE JOCS 3D

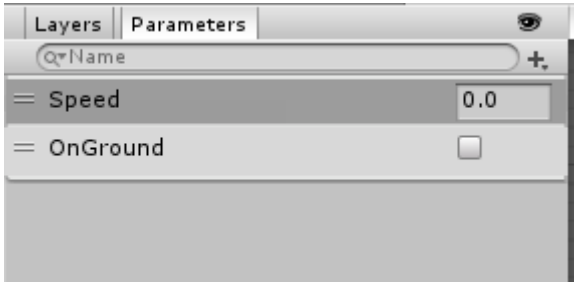
Práctica

Práctica 3 - https://youtu.be/D0PSJy2thnM

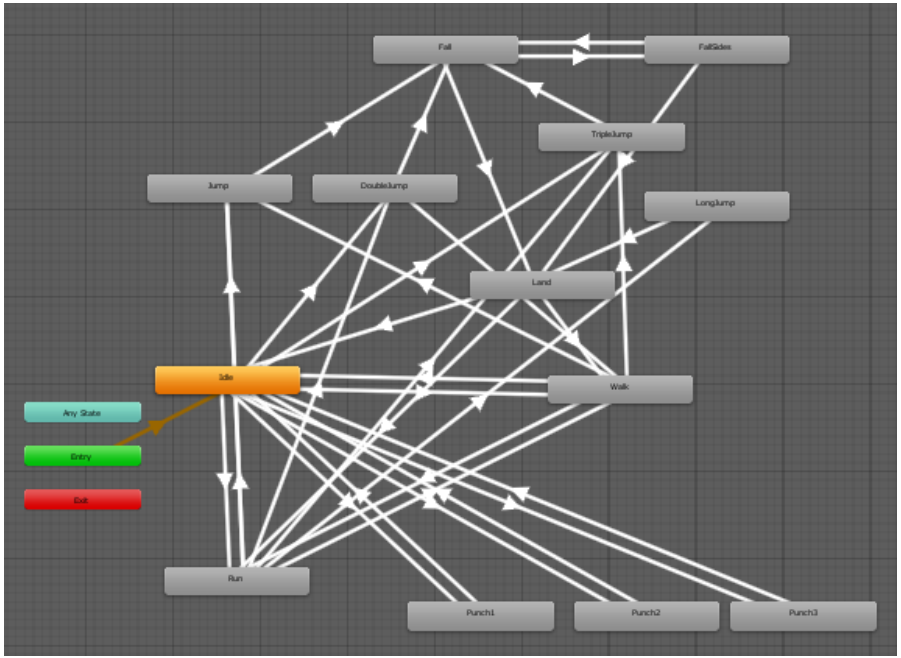
Importando assets



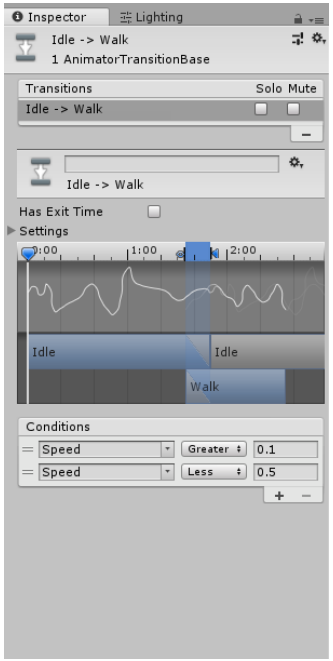
Animator - Parameters



Animator - FSM

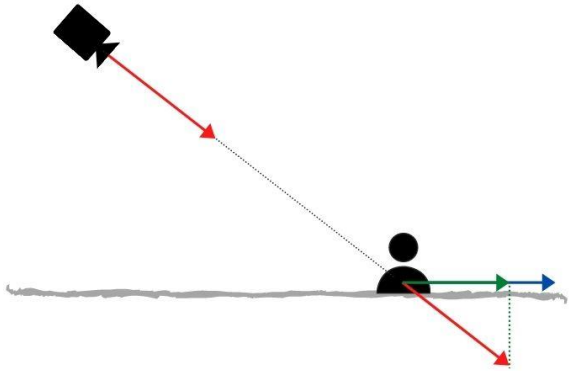


Animator - Transitions



Player Controller – Movement

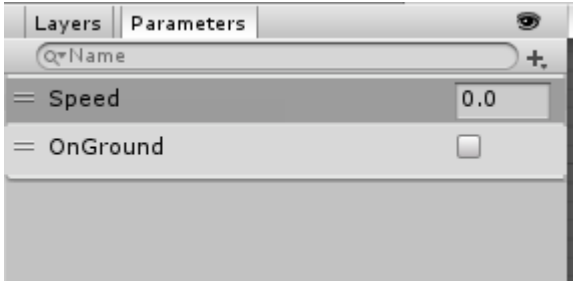
Queremos que el player se desplace en dirección a la orientación de la cámara, por lo que proyectaremos el vector forward y right de la cámara en el plano del suelo y los normalizaremos para obtener los vectores de dirección del player.



Player Controller – Animator Params

Para controlar las animaciones, deberemos modificar los parámetros del Animator desde el Player Controller utilizando:

```
m_Animator.setInteger(“Name”, 3);
m_Animator.setBool(“Name”, true);
m_Animator.setFloat(“Name”, 3.2f);
m_Animator.setTrigger(“Name”);
```

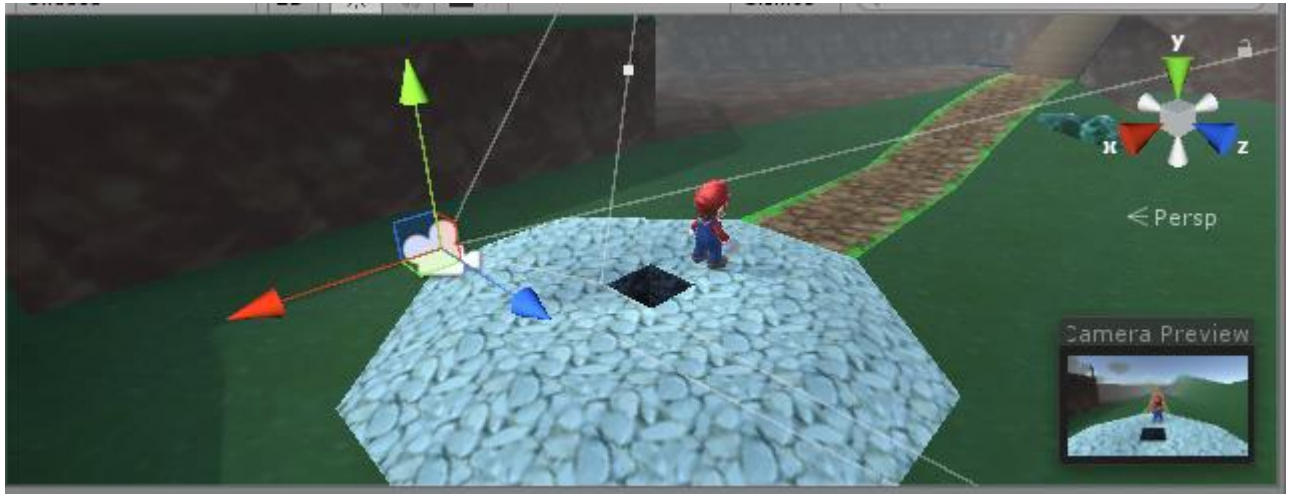


3d Person Camera

50 common game camera mistakes and how to fix them

https://www.youtube.com/watch?v=C7307qRmlMI

3d Person Camera



3d Person Camera

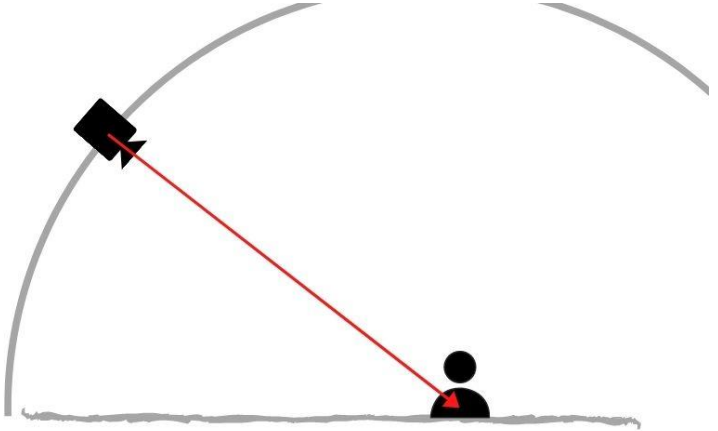
3d Person Camera - Positioning

Tendremos una referencia al transform que perseguimos (m_LookAt).

Del LookAt obtendremos el vector dirección de la cámara (l_Direction) y la distancia actual (l_Distance).

Actualizaremos el m_Yaw y el m_Pitch si se desplaza el mouse.

Calcularemos la posición ideal inicial de la cámara:



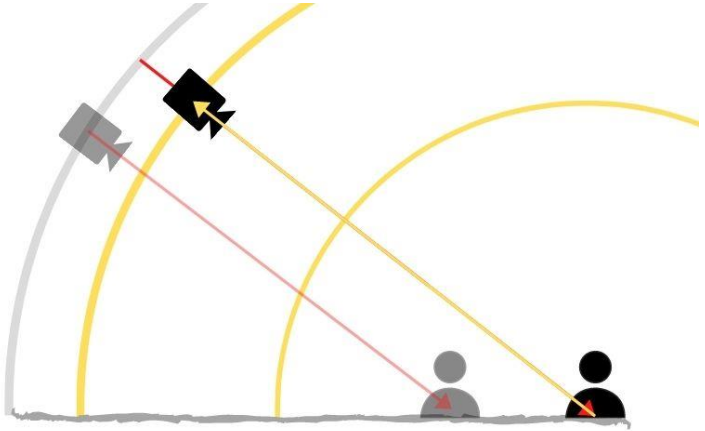
```
l_DesiredPosition = m_LookAt.position + new Vector3(
        Mathf.Sin(l_Yaw)*Mathf.Cos(l_Pitch)*l_Distance,
        Mathf.Sin(l_Pitch)*l_Distance,
        Mathf.Cos(l_Yaw)*Mathf.Cos(l_Pitch)*l_Distance);

l_Direction = m_LookAt.position - l_DesiredPosition;
```

3d Person Camera - Clamping distance

Para que la cámara siga al player, definiremos una distancia máxima (m_MaxDistanceToLookAt) y mínima (m_MinDistanceToLookAt).

Calcularemos la nueva posición corregida de la cámara normalizando el vector dirección anterior y multiplicándolo por la nueva distancia limitada (clamp).

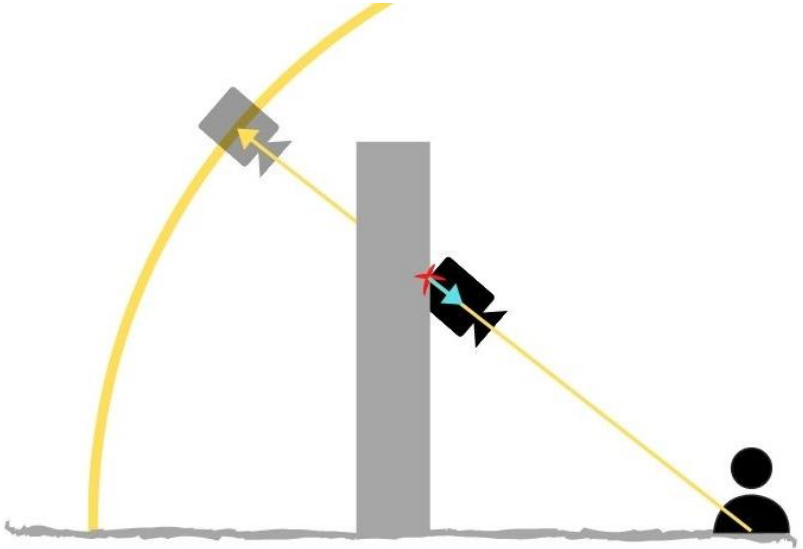


3d Person Camera - Avoid obstacles

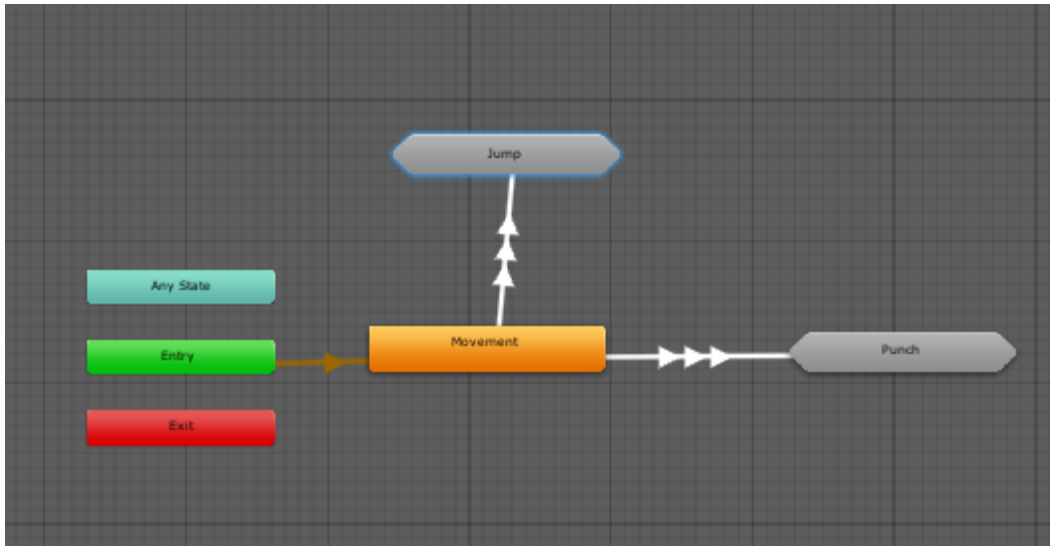
Volveremos a corregir la posición de la cámara para evitar obstáculos entre el player y la cámara.

Lanzaremos un rayo desde el m_LookAt en dirección a la cámara.

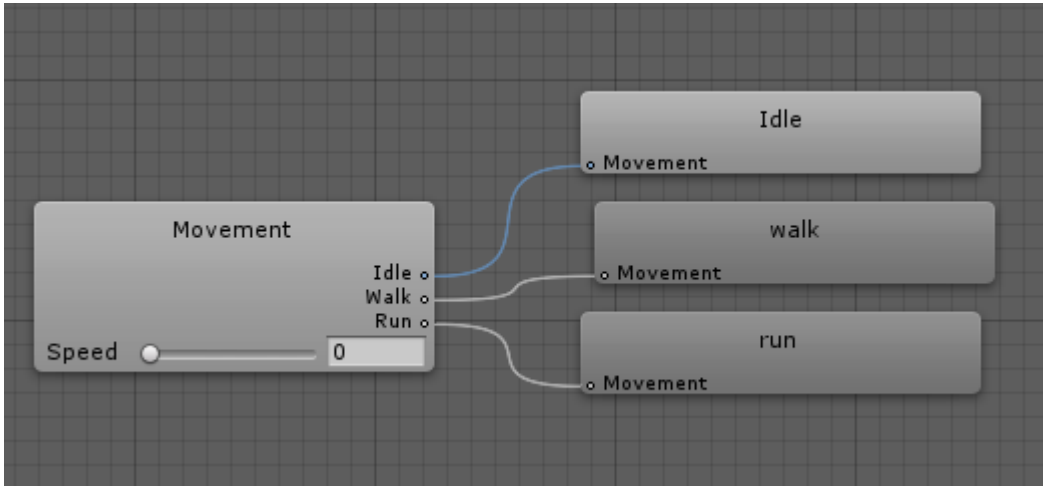
En caso de colisionar con algún objeto, recolocaremos la cámara en la posición del impacto aplicándole un offset en la dirección de la cámara.

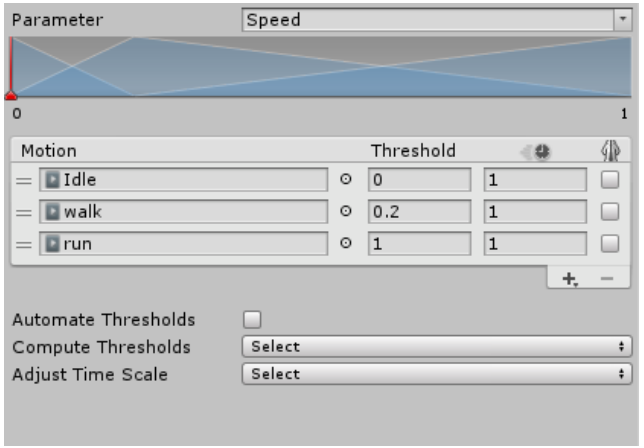


Advanced Animator - Sub-State Machines

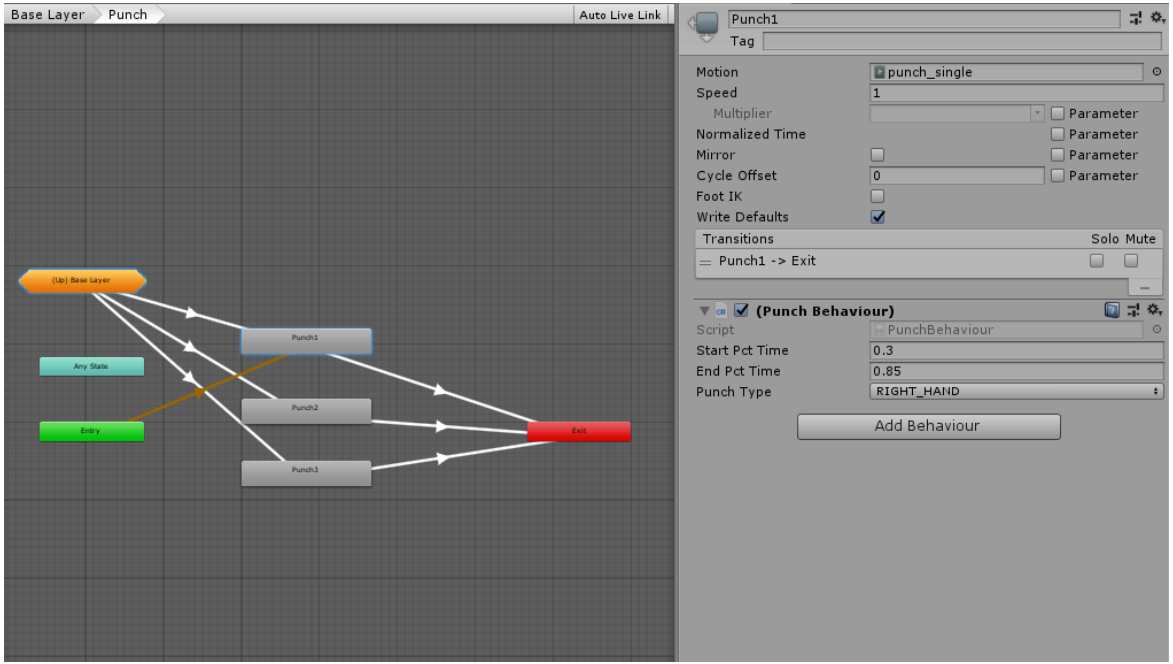


Advanced Animator - BlendTree





Advanced Animator - StateMachineBehaviour



Advanced Animator - StateMachineBehaviour

Podemos utilizar un script del tipo StateMachineBehaviour para añadir lógica a un estado del Animator.

```
override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex){}
override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex){}
```

Crearemos un script PunchBehaviour para implementar la lógica de los ataques de SuperMario. Solo dentro de un rango de tiempo de la animación de Punch, activaremos el ataque en el player.

RestartGame - Observer Pattern

Cuando reiniciemos el juego, queremos que aquellos objetos que deban reinicializarse, reciban una llamada a una función que implemente la lógica necesaria (recolocarse, reiniciar flags…)

Crearemos una interfaz para que todos estos objetos implementen la función RestartGame().

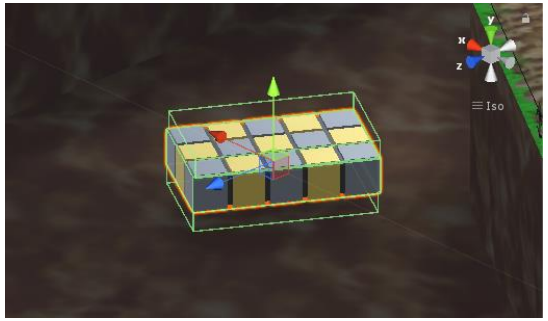
```
public interface IRestartGameElement
{
    void RestartGame();
}
```

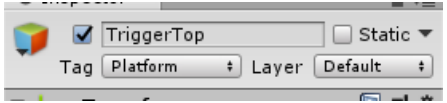
El GameController tendrá una lista de los componentes que implementan esta interfaz, a la cual se habrán registrado llamando a una función:

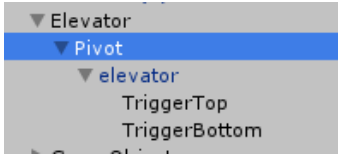
```
public void AddRestartGameElement(IRestartGameElement restartGameElement)
```

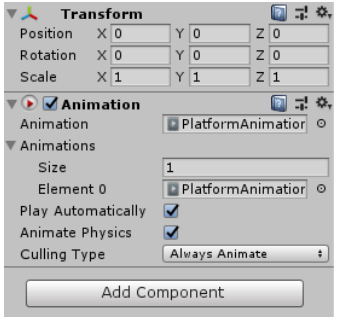
La función RestartGame del GameController, llamará a las funciones RestartGame de todos los componentes registrados al evento.

Platform Attaching







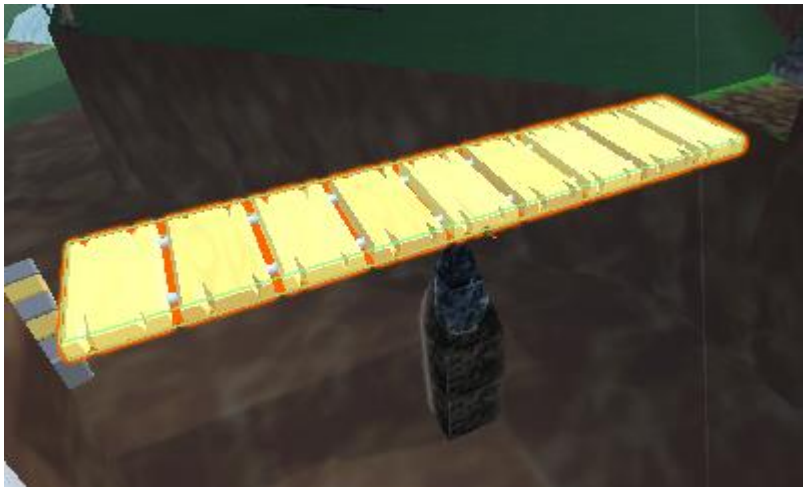


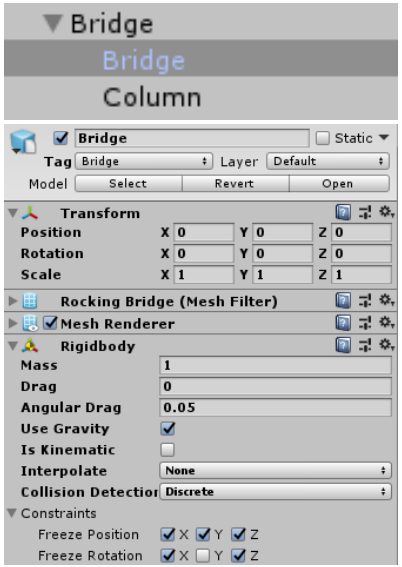
Cuando el player esté encima de la plataforma, queremos que el player herete el movimiento de ésta. Por este motivo, implementaremos la función AttachPlatform que hará al player hijo de la plataforma y la función DetachPlatform, que romperá la jerarquía entre el player y la plataforma.

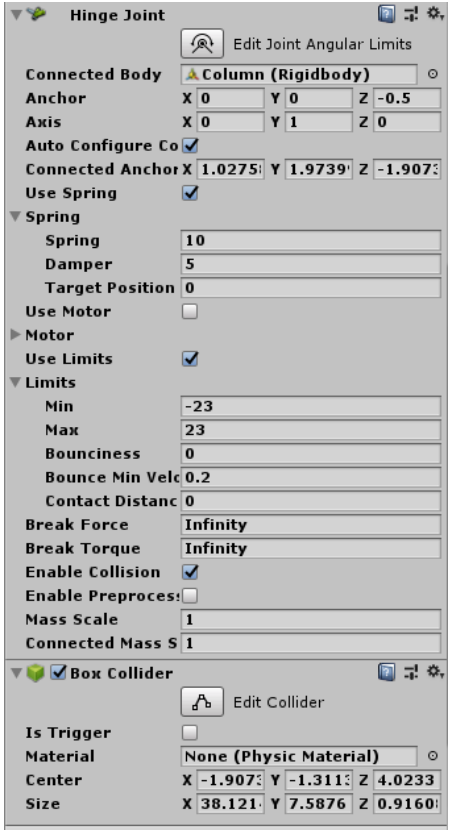
Llamaremos a la función AttachPlatform cuando el player entre en el trigger de la plataforma.

Llamaremos a la función DetachPlatform cuando el player salga del trigger o la plataforma esté inclinada, ya que queremos que el player caiga cuando gire.

Dynamic Platform - Joints







OnControllerColliderHit - AddForce

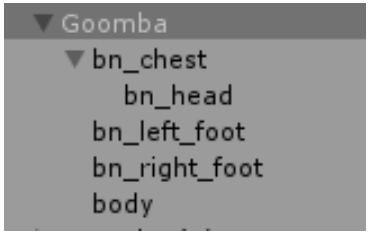
Utilizaremos el callback OnControllerColliderHit(ControllerColliderHit hit) para detectar cuando el CharacterController colisione con el puente.

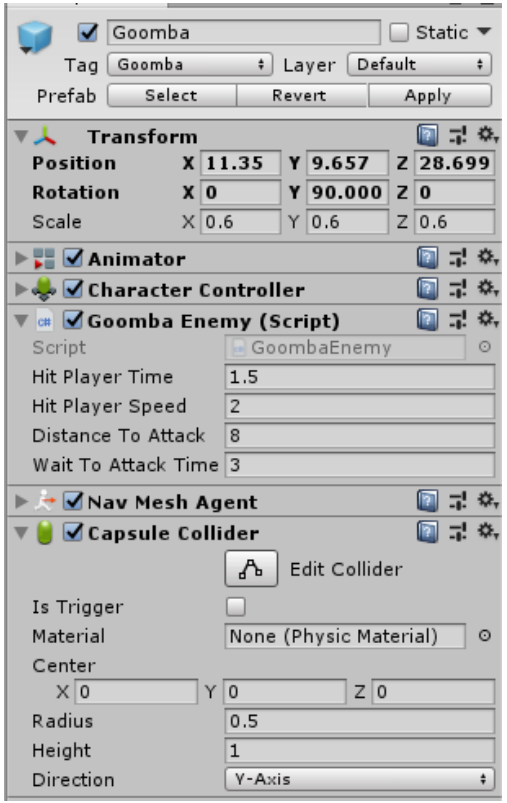
```
public void OnControllerColliderHit(ControllerColliderHit hit){}
```

En este caso aplicaremos una fuerza al puente en el hit.point y en dirección a -hit.normal, con fuerza m_BridgeForce.

```
l_Bridge.AddForceAtPosition(-hit.normal*m_BridgeForce, hit.point);
```

Kill Goomba


Goomba
bn_chest
bn_head
bn_left_foot
bn_right_foot
body


Goomba
Static
Tag Goomba
Layer Default
Prefab Select Revert Apply
Transform
Position X 11.35 Y 9.657 Z 28.699
Rotation X 0 Y 90.000 Z 0
Scale X 0.6 Y 0.6 Z 0.6
Animator
Character Controller
Goomba Enemy (Script)
Script GoombaEnemy
Hit Player Time 1.5
Hit Player Speed 2
Distance To Attack 8
Wait To Attack Time 3
Nav Mesh Agent
Capsule Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center
X 0 Y 0 Z 0
Radius 0.5
Height 1
Direction Y-Axis

Kill Goomba

Utilizaremos el callback OnControllerColliderHit(ControllerColliderHit hit) para detectar cuando el CharacterController colisione con un Goomba

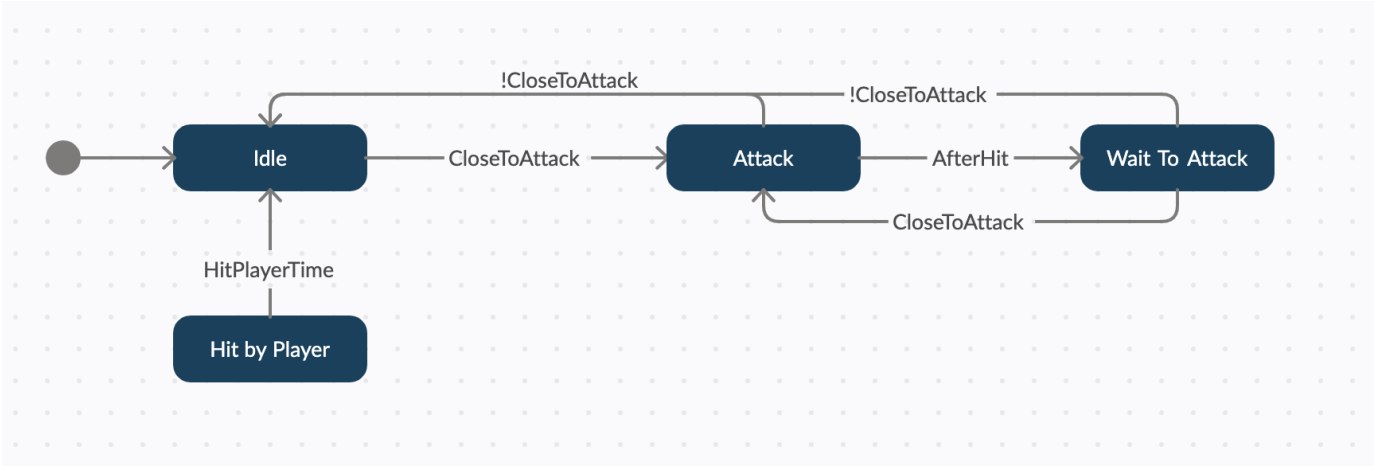
En este caso comprobaremos si el impacto ha sido mientras Mario está cayendo:

```
CanKillWithFeet();
```

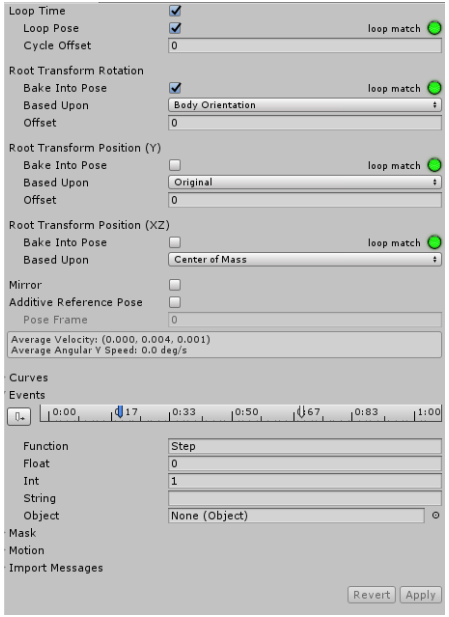
En cuyo caso eliminaremos el Goomba y haremos que el player salte/rebote:

```
JumpOverEnemy();
```

Goomba AI (FSM)



Animation Events

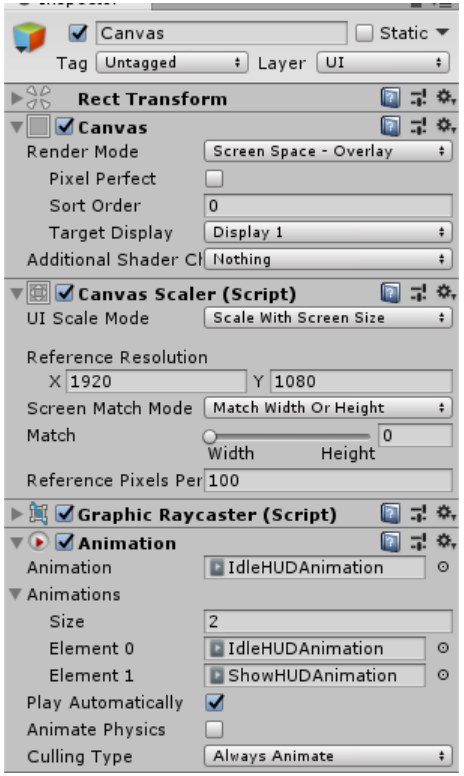



```
public void EventFunction(string stringParameter){}
public void EventFunction(float floatParameter){}
public void EventFunction(int intParameter){}
```

```
public void EventFunction(AnimationEvent animationEvent)
{
    string l_StringParmeter=animationEvent.stringParameter;
    float l_FloatParameter=animationEvent.floatParameter;
    int l_IntParameter=animationEvent.intParameter;
}
```

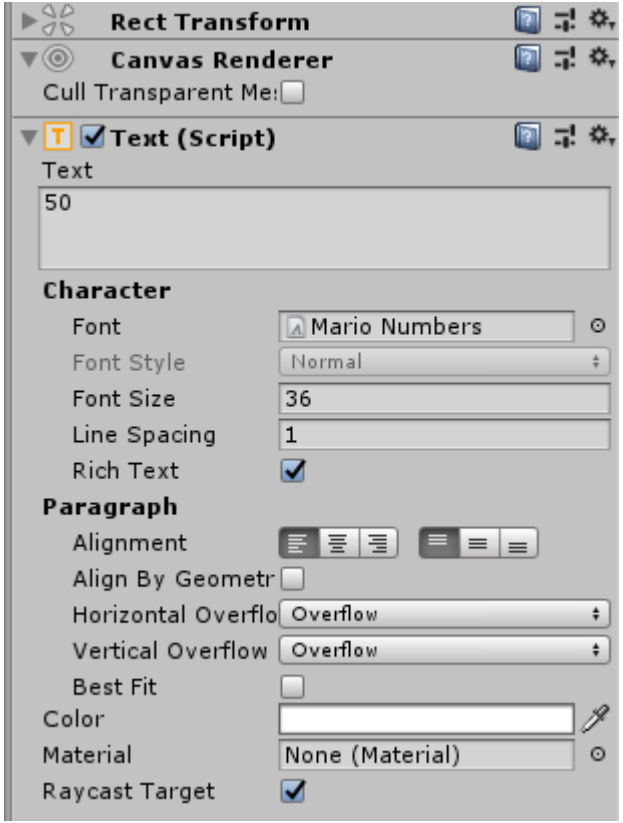
Canvas


Canvas
TopAnchor
CoinsImage
CoinsValue
Life
Background
Over

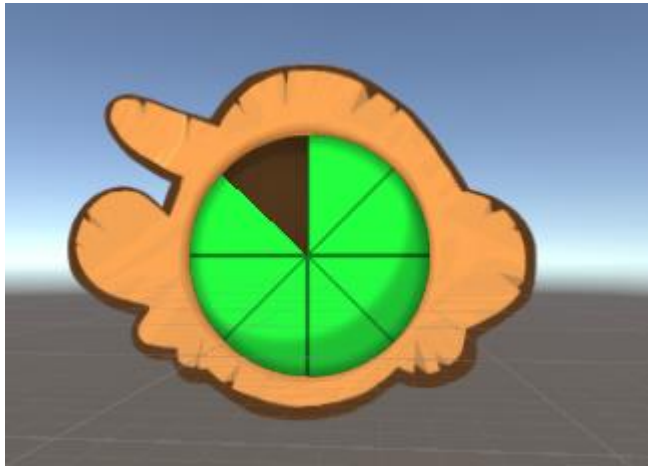

Canvas
Static
Tag
Untagged
Layer
UI
Rect Transform
Canvas
Render Mode
Screen Space - Overlay
Pixel Perfect
Sort Order
0
Target Display
Display 1
Additional Shader Ch
Nothing
Canvas Scaler (Script)
UI Scale Mode
Scale With Screen Size
Reference Resolution
X 1920
Y 1080
Screen Match Mode
Match Width Or Height
Match
0
Width
Height
Reference Pixels Per
100
Graphic Raycaster (Script)
Animation
Animation
IdleHUDAnimation
Animations
Size
2
Element 0
IdleHUDAnimation
Element 1
ShowHUDAnimation
Play Automatically
Animate Physics
Culling Type
Always Animate

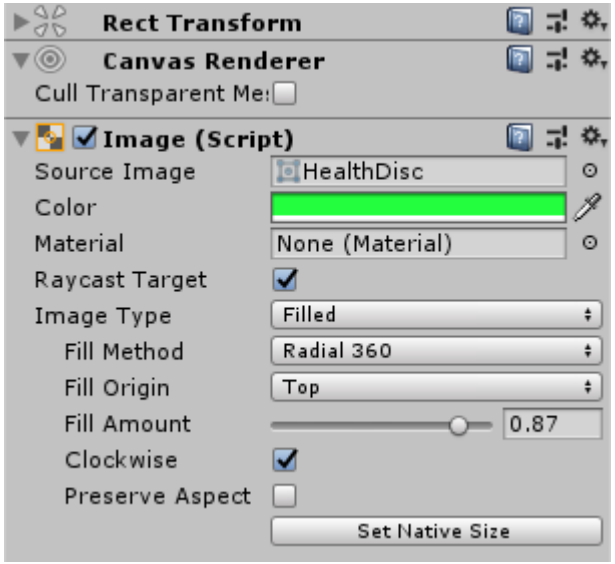
Canvas


50

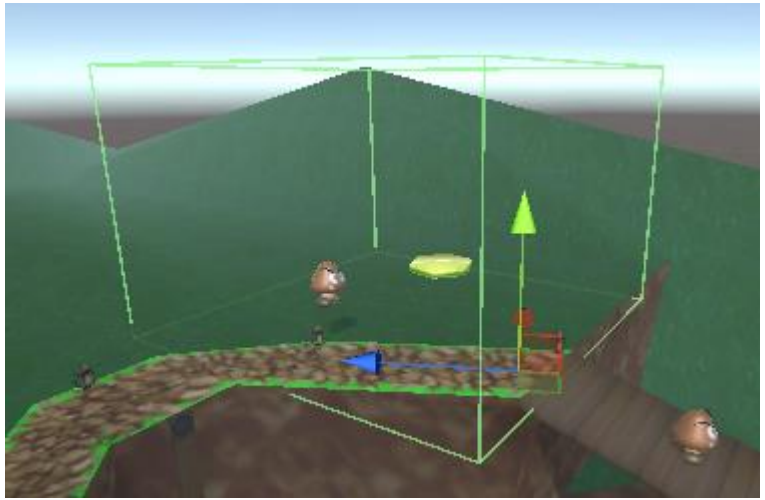

Rect Transform
Canvas Renderer
Cull Transparent Me
Text (Script)
Text
50
Character
Font
Mario Numbers
Font Style
Normal
Font Size
36
Line Spacing
1
Rich Text
Paragraph
Alignment
Align By Geometr
Horizontal Overflo
Overflow
Vertical Overflow
Overflow
Best Fit
Color
Material
None (Material)
Raycast Target

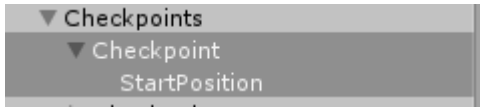
Canvas

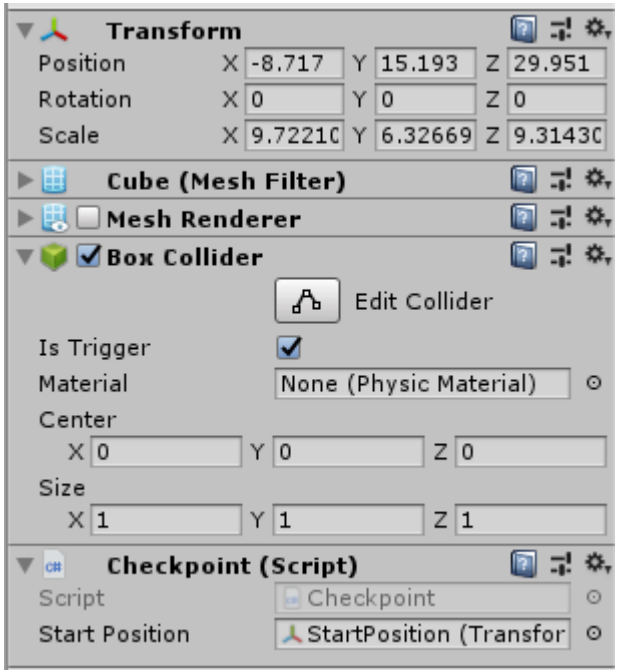



Rect Transform
Canvas Renderer
Cull Transparent Me
Image (Script)
Source Image
HealthDisc
Color
Material
None (Material)
Raycast Target
Image Type
Filled
Fill Method
Radial 360
Fill Origin
Top
Fill Amount
0.87
Clockwise
Preserve Aspect
Set Native Size

Checkpoint




Checkpoints
Checkpoint
StartPosition


Transform
Position X -8.717 Y 15.193 Z 29.951
Rotation X 0 Y 0 Z 0
Scale X 9.72210 Y 6.32669 Z 9.31430
Cube (Mesh Filter)
Mesh Renderer
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center
X 0 Y 0 Z 0
Size
X 1 Y 1 Z 1
Checkpoint (Script)
Script Checkpoint
Start Position StartPosition (Transfor

Game Managers with Dependency Injection

La DI nos permite eliminar las dependencias en las clases de alto nivel de las clases de bajo nivel. También nos permite hacer test unitario ya que podremos inyectar dependencias falsas a un sistema para probarlo de forma aislada.

Unity necesita controlar los constructores por lo que no podremos inyectar estas dependencias en el constructor. Cuando Unity cree una instancia de nuestro Manager, lo registraremos en una clase que gestionará todas las dependencias.

Todos los scripts que requieran de estas dependencias, pedirán una interfaz en esta clase gestora de dependencias. Así **dependerán de una interfaz (abstracción) y no de una implementación**. Esto nos permitirá cambiar la implementación sin alterar todas las clases que sean dependientes de la cambiada.

Game Managers with Dependency Injection

```
public class DependencyInjector
{
    static Dictionary<Type, System.Object> dependencies = new Dictionary<Type, System.Object>();
    public static T GetDependency<T>()
    {
        if (!dependencies.ContainsKey(typeof(T)))
        {
            Debug.LogError("Cannot find: " + typeof(T).ToString()+".");
            return default(T);
        }
        return (T)dependencies[typeof(T)];
    }
    public static void AddDependency<T>(System.Object obj)
    {
        if (dependencies.ContainsKey(typeof(T)))
        {
            Debug.Log("There's already an object of type: " + typeof(T).ToString());
            Debug.Log("Object 1: " + dependencies[typeof(T)].GetType().ToString());
            Debug.Log("Object 2: " + obj.GetType().ToString());
            dependencies.Remove(typeof(T));
        }
        dependencies.Add(typeof(T), obj);
    }
}
```

Game Managers with Dependency Injection

```
public class ScoreManager : MonoBehaviour, IScoreManager
{
    [SerializeField] float points;
    public event ScoreChanged scoreChangedDelegate;

    void Awake()
    {
        DependencyInjector.AddDependency<IScoreManager>(this);
    }
    public void addPoints(float points)
    {
        this.points += points;
        scoreChangedDelegate?.Invoke(this);
    }
    public float getPoints() { return points; }
}
```

```
public interface IScoreManager
{
    void addPoints(float f);
    float getPoints();
    event ScoreChanged scoreChangedDelegate;
}
public delegate void ScoreChanged(IScoreManager scoreManager);
```

Game Managers with Dependency Injection

```
public class HUD : MonoBehaviour
{
    public TextMeshProUGUI score;

    private void Start()
    {
        DependencyInjector.GetDependency<IScoreManager>()
           .scoreChangedDelegate += updateScore;
    }
    private void OnDestroy()
    {
        DependencyInjector.GetDependency<IScoreManager>()
           .scoreChangedDelegate -= updateScore;
    }

    public void updateScore(IScoreManager scoreManager)
    {
        score.text = "Score: " +
           scoreManager.getPoints().ToString("0");
    }
}
```

```
[CreateAssetMenu(menuName="Score")]
public class Score : ScriptableObject
{
    public float points;

    public void score()
    {
        IScoreManager score = DependencyInjector.GetDependency<IScoreManager>();
        score.addPoints(points);
    }
}

public class Coin : MonoBehaviour
{
    [SerializeField]Score score;
    private void OnTriggerEnter(Collider other)
    {
        if(score != null)
        {
            score.score();
        }
        Destroy(gameObject);
    }
}
```