Bubble Bobble, Taito, 1986

Plataformas 2D

Índice

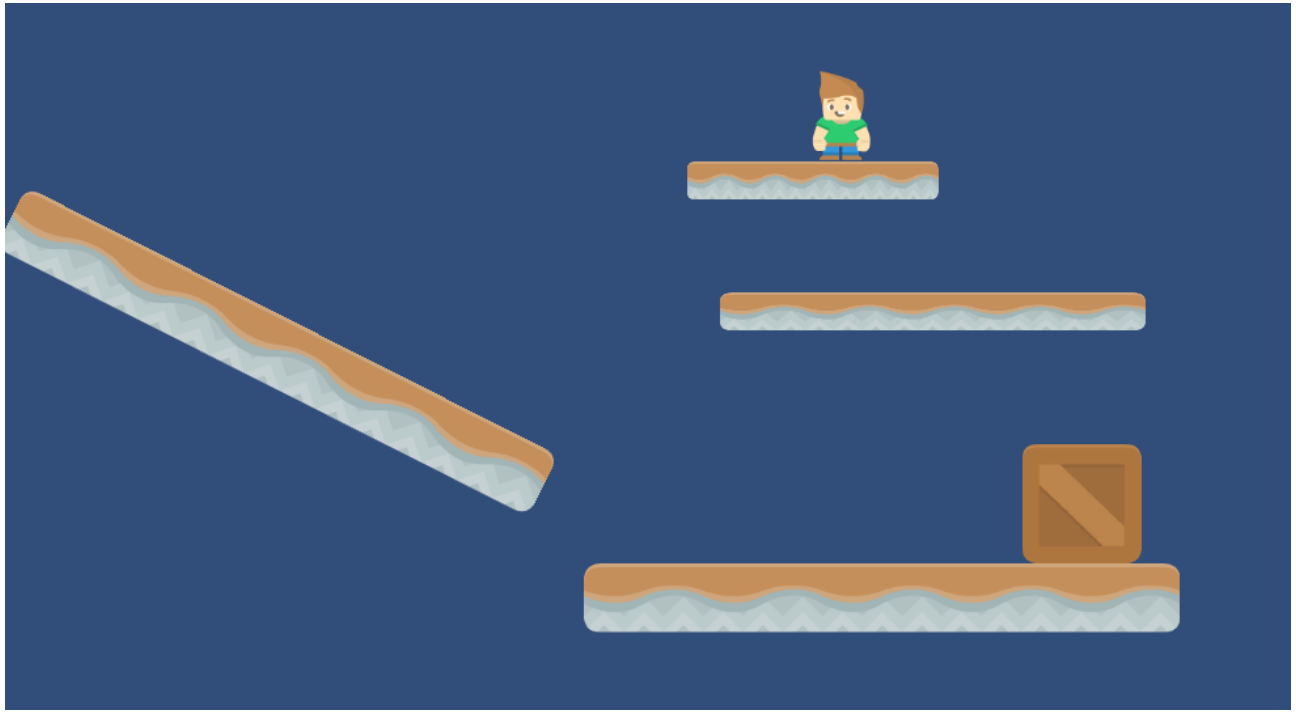
- Introducción
- Creando el entorno
- Creando el personaje
- Plataformas inclinadas
- Plataformas unidireccionales
- Plataformas móviles
- Control de la cámara
- Trampolín
- Parallax scrolling

Introducción

- En esta unidad vamos a implementar un juego de plataformas 2D básico
- Los juegos de plataformas 2D tienen varias mecánicas en común:
  - El protagonista se mueve lateralmente en un escenario. Normalmente puede andar y saltar
  - La cámara sigue al protagonista, con un scroll
  - El escenario está formado por plataformas
  - El objetivo del juego suele ser llegar a un lugar concreto, recolectar objetos, destruir a los enemigos, etc.
  - Puede haber varios tipos de plataformas: inclinadas, móviles, de una dirección, cintas transportadoras, escaleras, destruibles, trampolín…

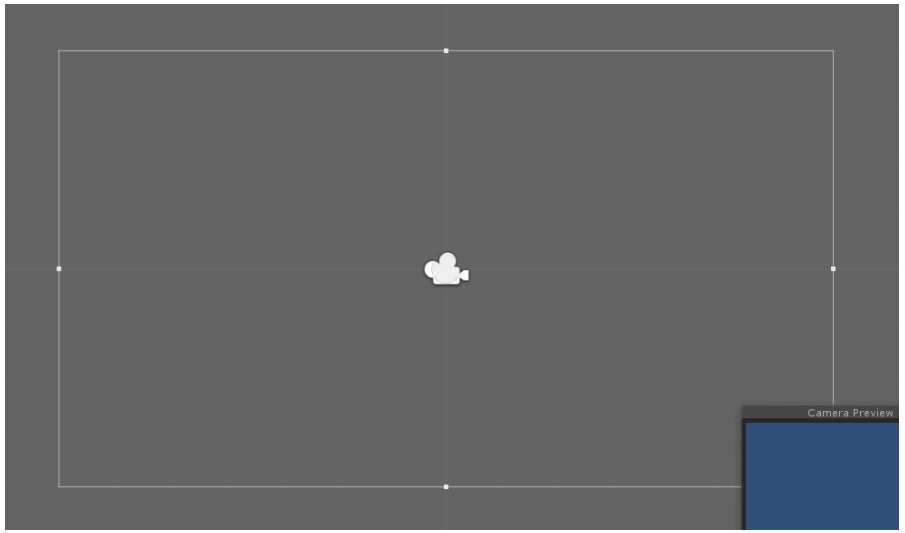
Introducción

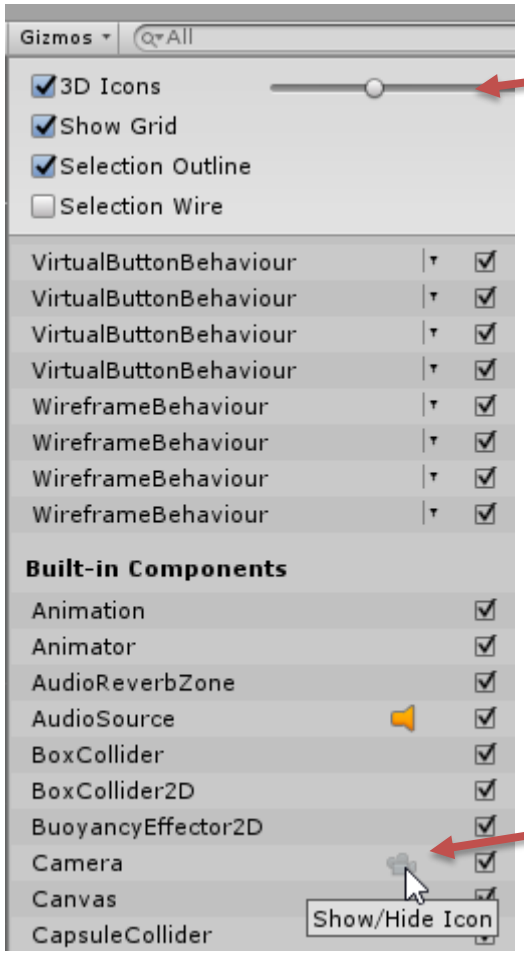
- Vamos a hacer una escena sencilla con varias plataformas
- El jugador podrá mover al personaje lateralmente y saltar



Empezando

- Crea un proyecto de tipo 2D
- Crea los directorios típicos (Scripts, Sprites…)
- Reduce el tamaño de la cámara a 4
- Normalmente el icono de la cámara aparece en el centro de la ventana y puede molestar mientras que trabajamos en la escena:



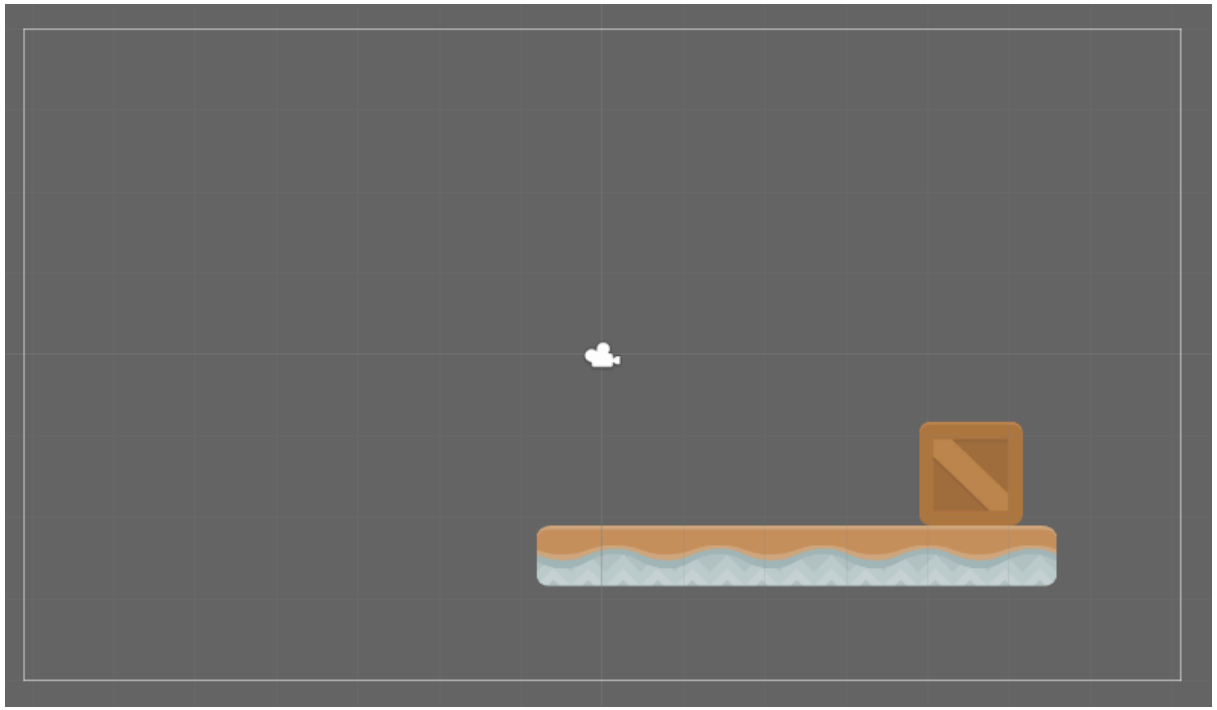


Cambiar el tamaño

Ocultar el icono

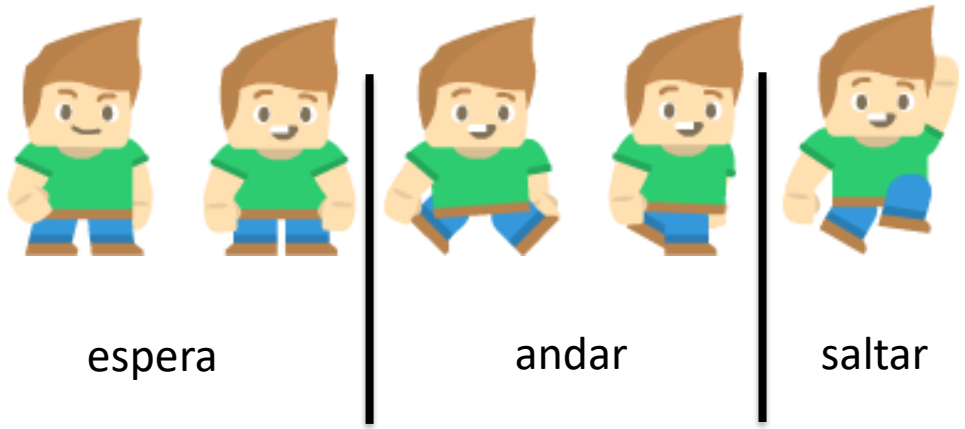
Creando el entorno

- Importa los sprites del entorno y construye una plataforma y una caja
  - Recuerda: inicialmente lo único que nos interesa es la jugabilidad del nivel. Usa primitivas sencillas o assets existentes



Creando el personaje

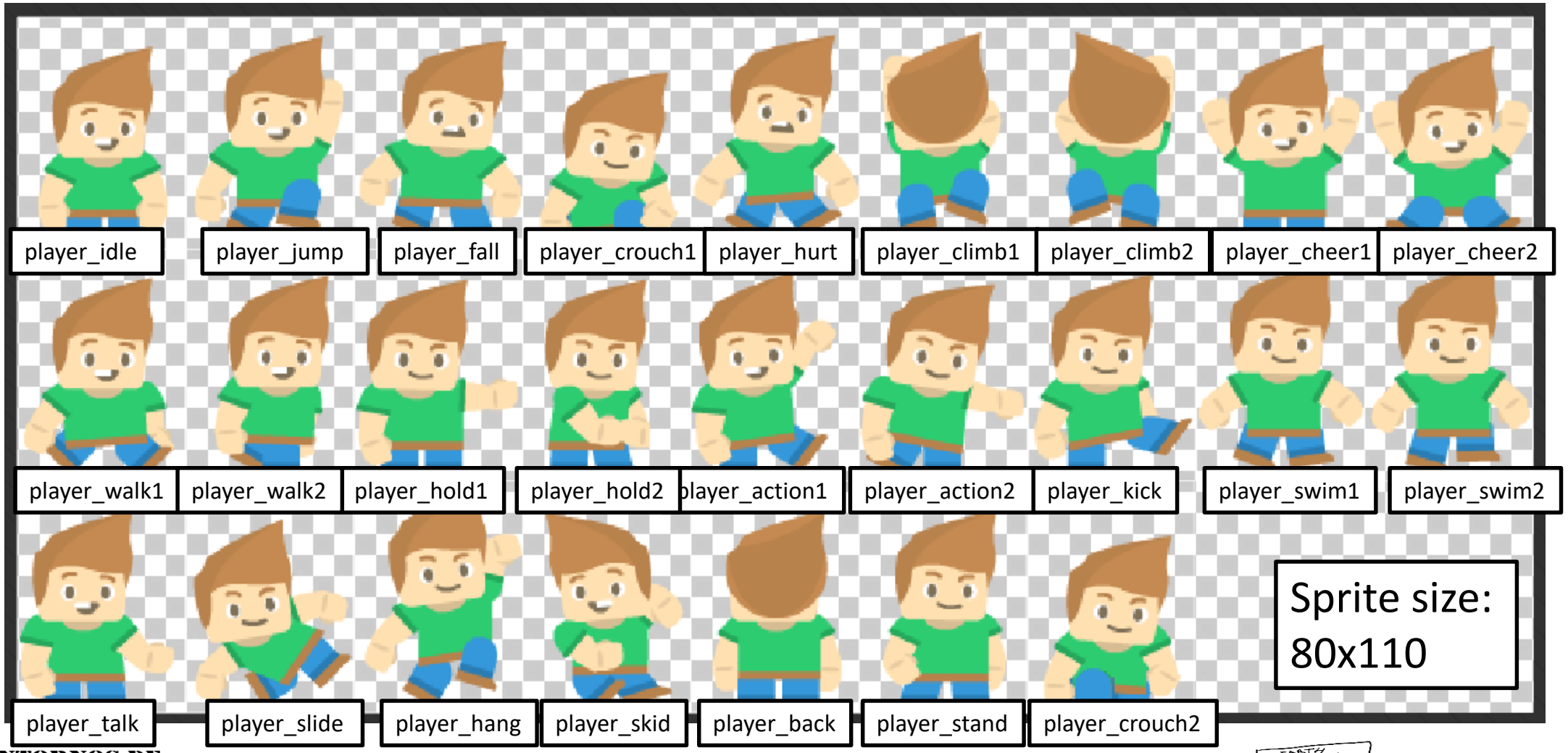
- La animación de los personajes puede venir en ficheros separados, o en un solo fichero (*sprite sheet*)
- Importa el sprite sheet del jugador, de la que usaremos las animaciones para los estados de espera (idle), andar y saltar



Creando el personaje

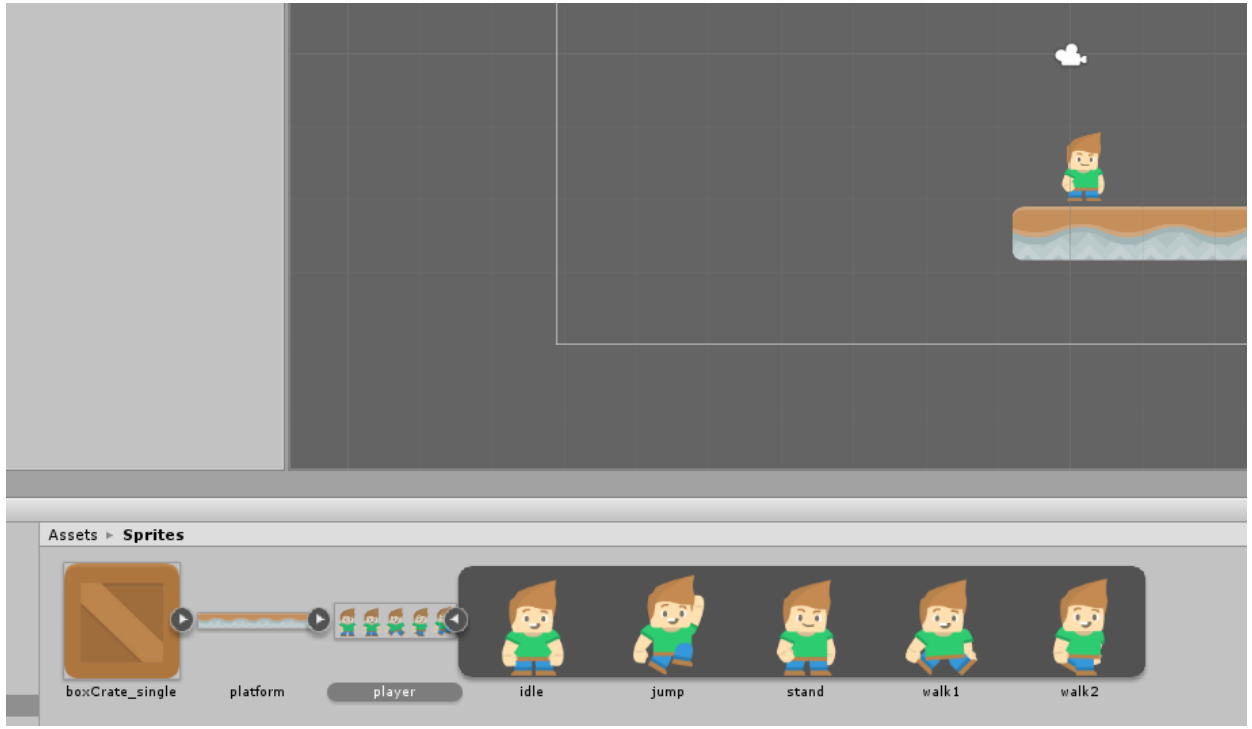
- Para extraer los distintos frames de las animaciones, tenemos que seleccionar el fichero de la carpeta Sprites, y en el Inspector, cambia el Sprite Mode de Single a Multiple
- Luego, abre el Sprite Editor:

Creando el personaje



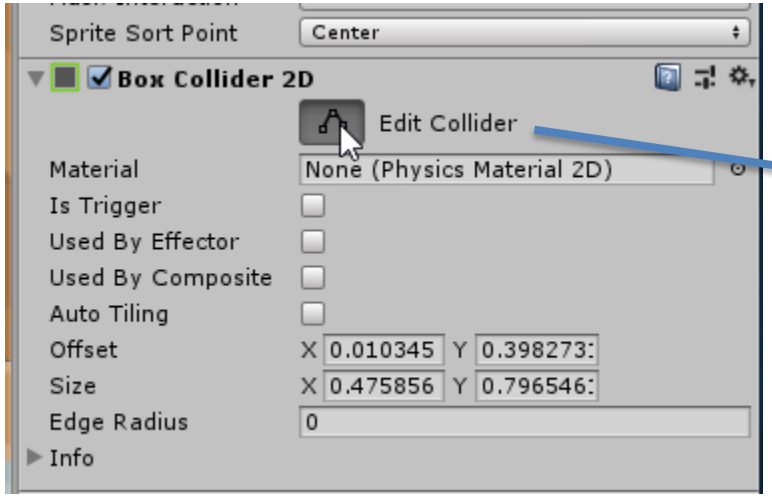
Creando el personaje

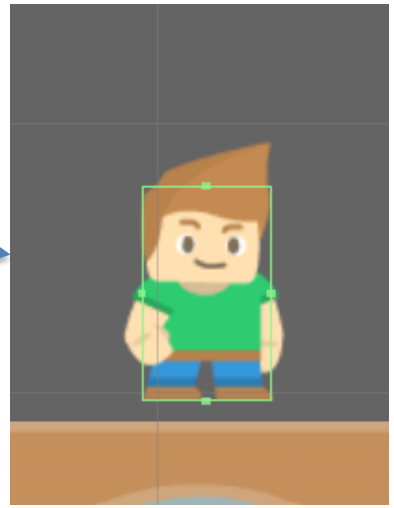
- Una vez que se han extraído los frames, ya se pueden seleccionar en el panel Project, y se puede arrastrar a la escena
- Arrastra el fotograma “player-stand” a la escena, y llama al nuevo objeto Player



Creando el personaje

- El siguiente paso es hacer que el personaje esté controlado (parcialmente) por el motor de física. Añade al GameObject Player:
  - Un RigidBody2D (esto hace que le afecte, por ejemplo, la gravedad)
  - Un Box Collider 2D (esto define la forma que ve el motor de física, y la que se usa para el cálculo de colisiones)

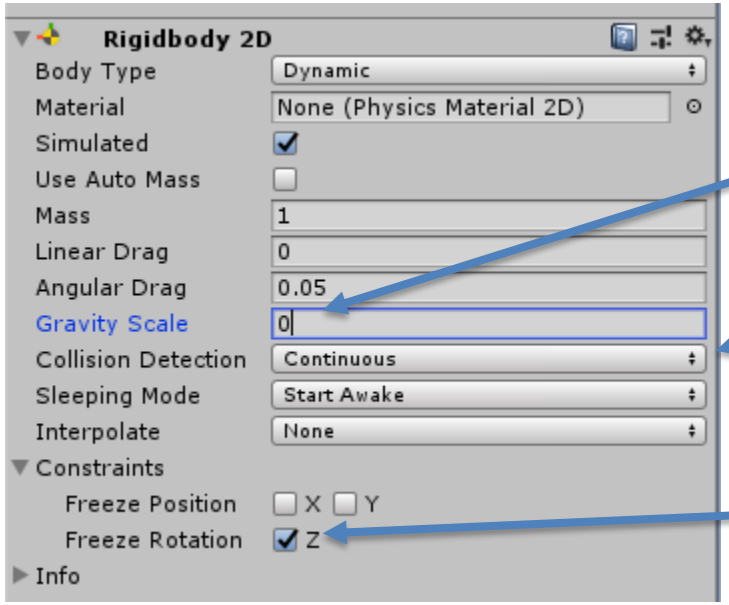




Puedes ajustar el collider a la forma del personaje

Creando el personaje

- Vamos a ajustar varios parámetros del Rigidbody 2D



Sin gravedad (por el momento)

Cálculo continuo de detección de colisiones

No rotar el personaje

Creando el personaje

- Script para el control del personaje:

```
public class PlaformerPlayer : MonoBehaviour
{
    public float speed = 4.5f;
    private Rigidbody2D _body;

    void Start() {
        _body = GetComponent<Rigidbody2D>();
    }

    void Update() {
        float deltaX = Input.GetAxis("Horizontal") * speed;
        Vector2 movement = new Vector2(deltaX, _body.velocity.y);
        _body.velocity = movement;
    }
}
```

Creando el personaje

- Ejecuta el juego y prueba el movimiento
- Comprobarás que la respuesta del personaje no es muy rápida
- Unity añade algo de aceleración a la entrada de teclado por defecto para hacer los movimientos más suaves. La puedes configurar en el panel:
  - `Edit\Project Settings\Input\Horizontal\`
  - Prueba a cambiar `Gravity` y `Sensitivity` a 6
- O usa la versión Raw:
  - `Input.GetAxisRaw("Horizontal")`

Creando el personaje

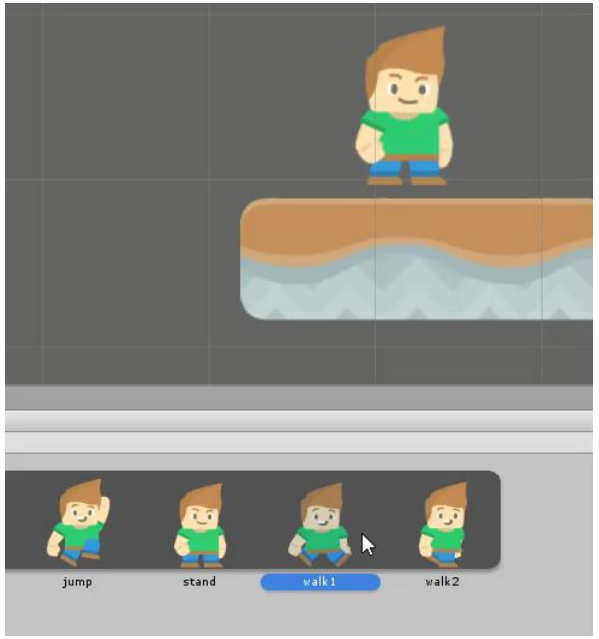
- Ahora mismo el jugador puede atravesar la caja
- Para evitarlo, tenemos que añadir un Collider 2D a cada elemento de la escena
  - Además, si el objeto tiene que entrar en la simulación de la física, habría que añadirle un RigidBody 2D
- Como estamos aplicando el movimiento al jugador a través del RigidBody, el motor de física calcula colisiones y ya no puede atravesar la caja
  - Si hubiéramos movido el objeto cambiando su `transform.position`, no se aplicaría el cálculo de colisiones

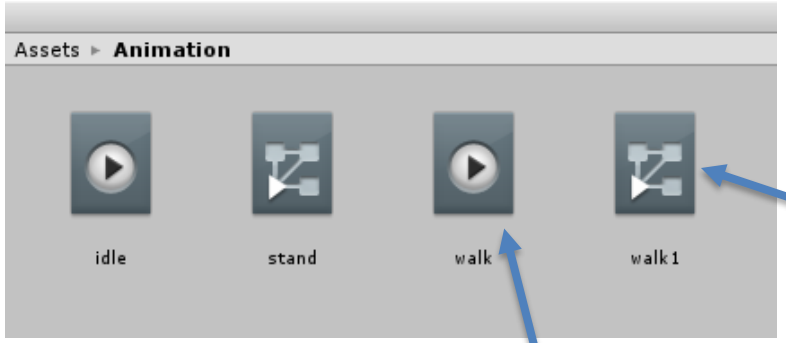
Creando el personaje

- El siguiente paso consiste en animar el personaje
- Para ello vamos a utilizar el motor de animación de Unity
- En el centro del sistema de animación se encuentra una máquina de estados, donde:
  - cada estado es una acción (andar, saltar, disparar…), y
  - las transiciones entre los estados se disparan por condiciones internas (se ha completado una animación) o externas (el jugador ha pulsado el botón de disparo)

Creando el personaje

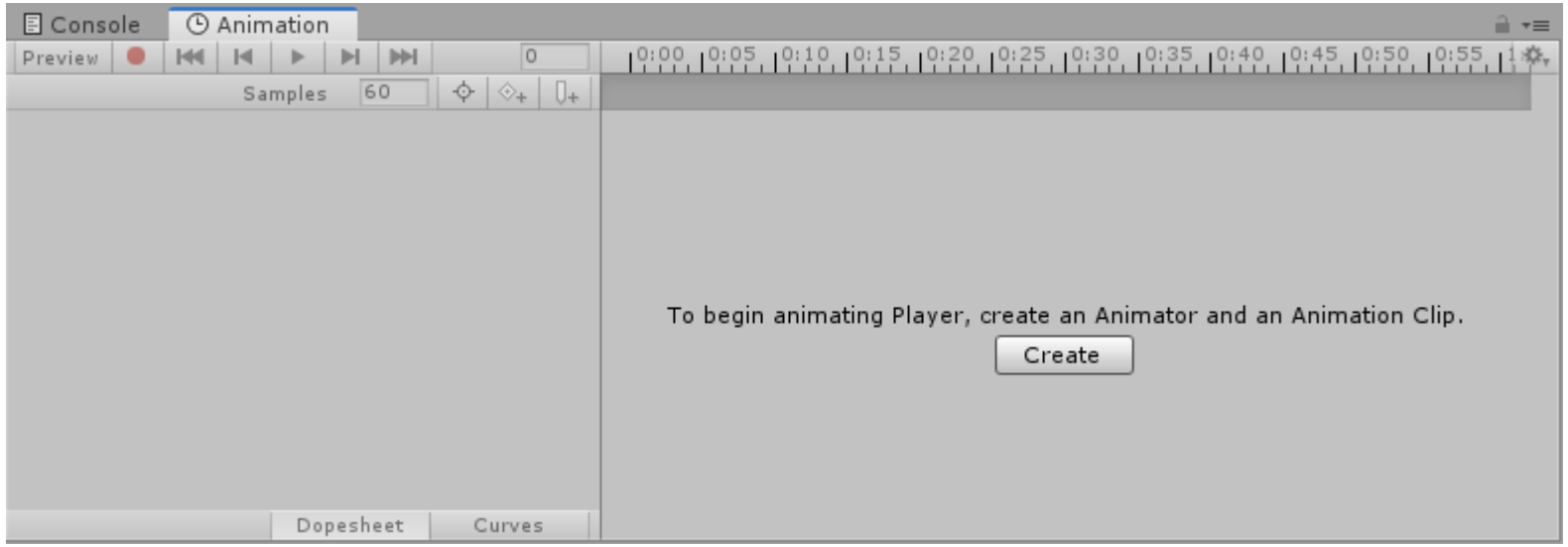
- Para crear un estado (una animación, o Animation Clip en Unity) en un juego con sprites hay varias opciones:
  - Si tiene varios frames, seleccionarlos y arrastrarlos a la escena
  - Unity pedirá el nombre del clip de animación (walk, idle)
  - Por defecto, junto a cada clip de animación, también se crea una máquina de estados





Creando el personaje

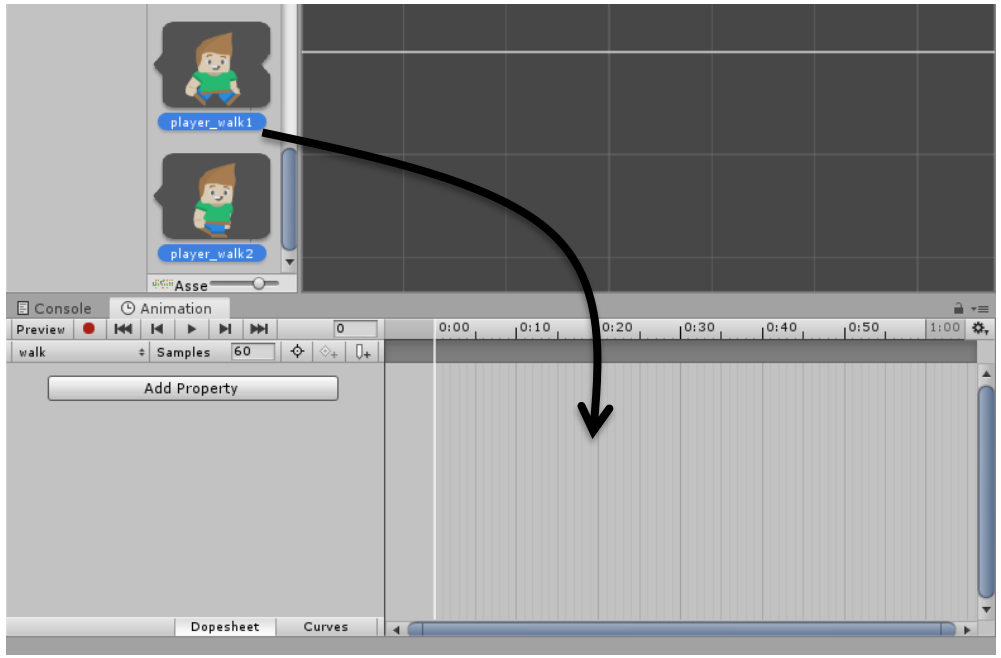
- También puedes seleccionar el objeto inicial (no animado), y abrir la pestaña Animation:



- Pulsa Create para crear un animation clip y un animator controller

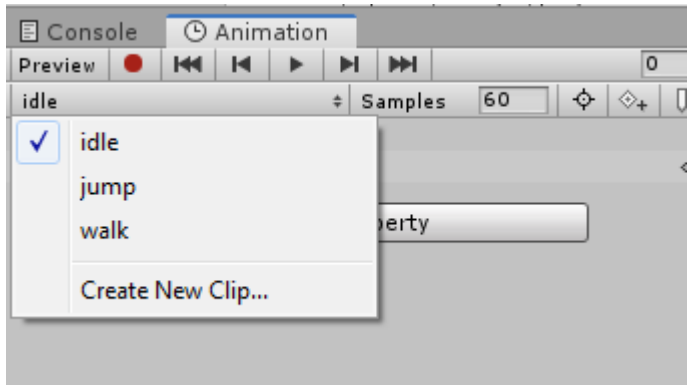
Creando el personaje

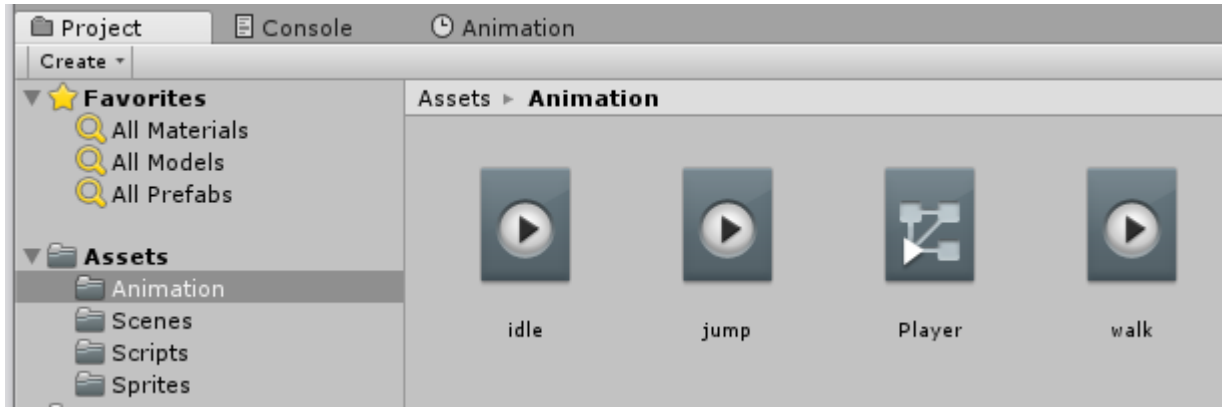
- Arrastra los frames de la animación a la línea de tiempos y ajusta la velocidad de la animación en el campo Samples



Creando el personaje

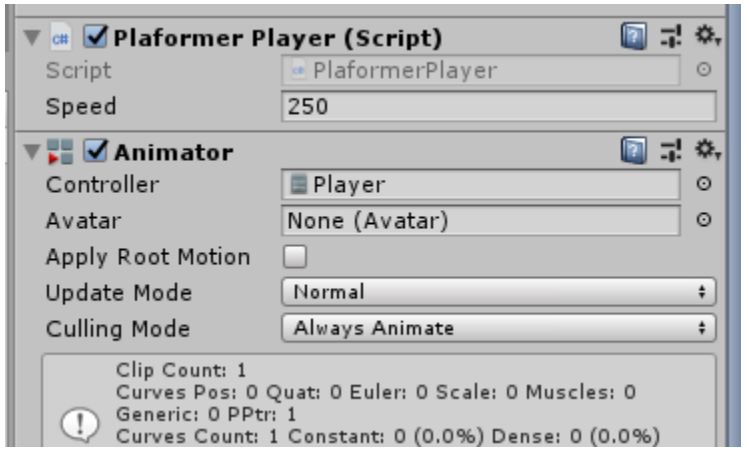
- Prepara las animaciones walk, jump e idle
- Organiza todos los clips de animación y el controlador en una carpeta

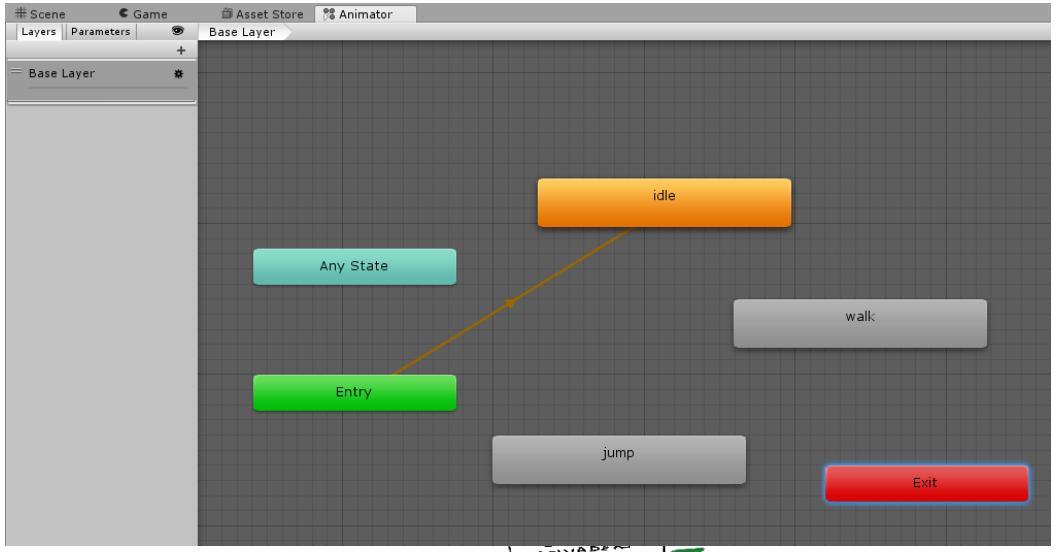




Creando el personaje

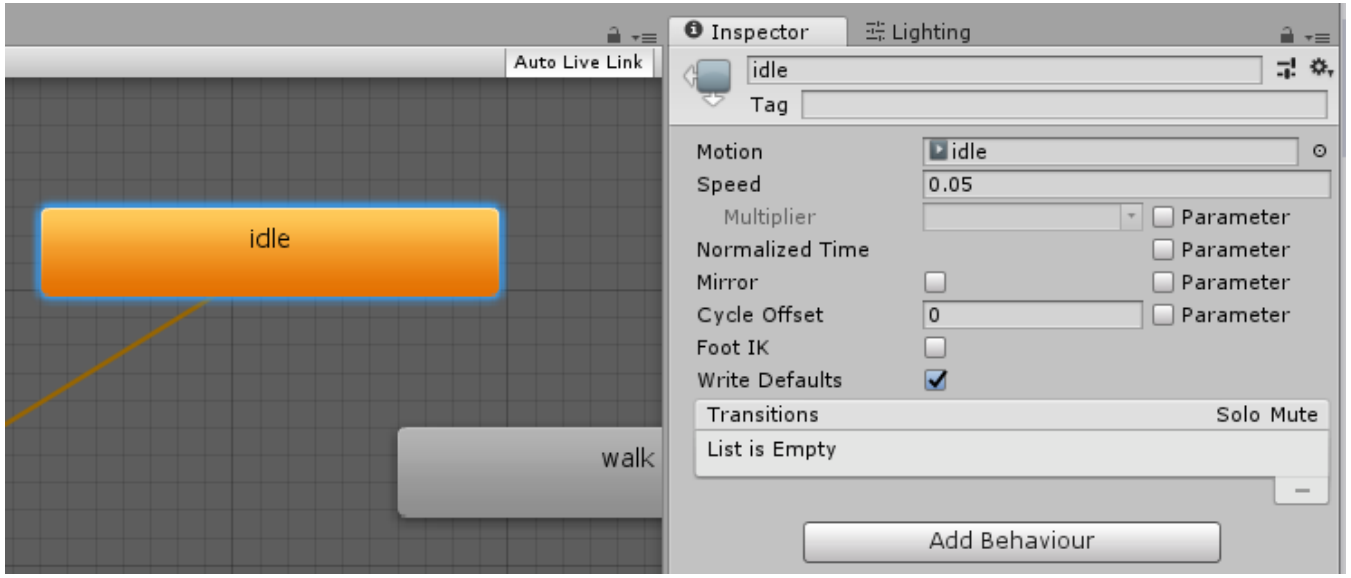
- Selecciona el GameObject del personaje
- Si no lo tiene, añade un componente `Animator`
- Arrastra la máquina de estados al campo `Controller`
- Con el personaje seleccionado, abre la ventana `Animator` (`Window\Animation\Animator`)
- Arrastra las animaciones que falten a la nueva ventana
- Al ejecutar el programa podemos ver ya la animación marcada en naranja





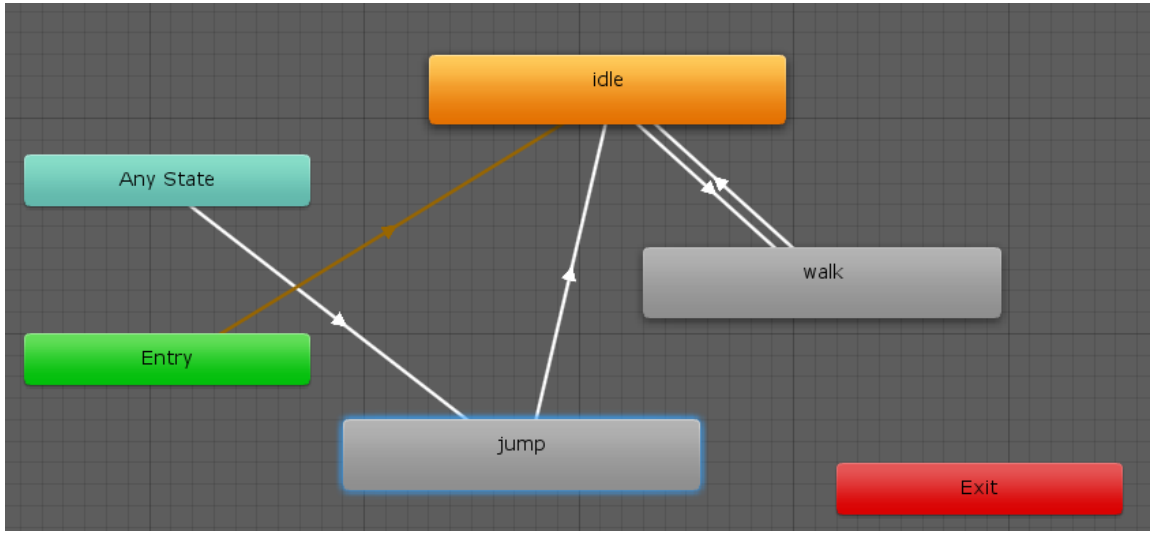
Creando el personaje

- Para cambiar el estado inicial, pulsa con el botón derecho sobre el estado deseado y selecciona `Set as Layer Default State`
- Al seleccionar un estado podemos modificar algunos parámetros como, por ejemplo, la velocidad de reproducción.



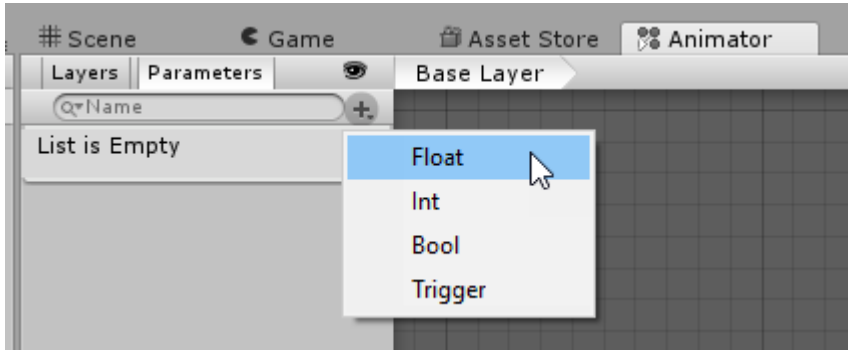
Creando el personaje

- Vamos a crear las transiciones entre los estados:
  - idle <-> walk
  - Any State -> jump
  - Jump -> idle
- Haz clic con el botón derecho sobre el estado inicial, selecciona Make Transition y arrastra la flecha hasta el estado final



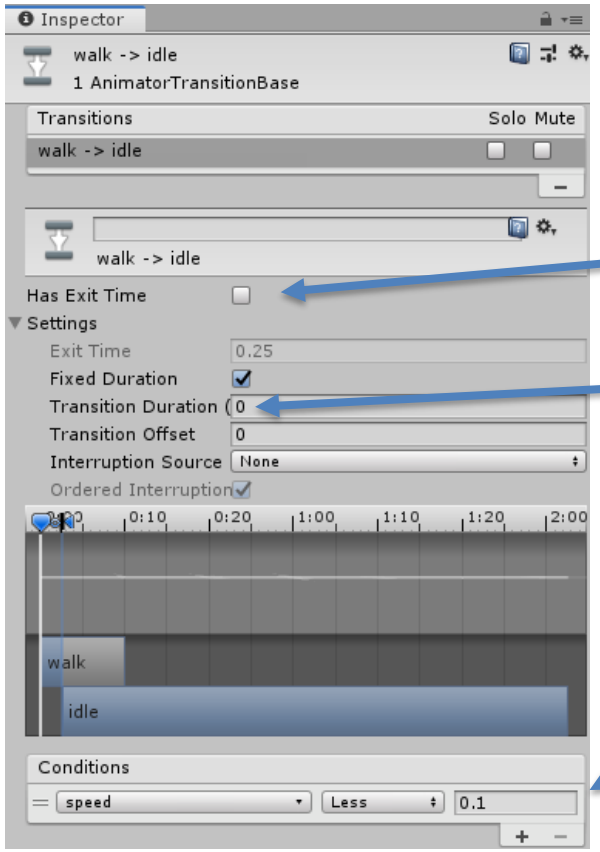
Creando el personaje

- Las transiciones se pueden disparar por condiciones externas, controladas por un parámetro de la máquina de estados
- Crea un parámetro de tipo float, llamado speed:



Creando el personaje

- Ahora vamos a configurar las transiciones entre idle y walk:
  - Idle -> walk: cuando speed > 0.1
  - Walk -> idle: cuando speed < 0.1



Desactiva “Has Exit Time”

Transición inmediata

Condición/es de disparo

Creando el personaje

- Ahora el script PlatformerPlayer debe establecer el valor del parámetro speed:

PlatformerPlayer.cs

```
private Rigidbody2D _body;
private Animator _anim;

void Start() {
  _body = GetComponent<Rigidbody2D>();
  _anim = GetComponent<Animator>();
}

void Update() {
  float deltaX = Input.GetAxis("Horizontal") * speed;
  _anim.SetFloat("speed", Mathf.Abs(deltaX));
  if (!Mathf.Approximately(deltaX, 0f)) {
            transform.localScale = new Vector3(Mathf.Sign(deltaX), 1f, 1f);
  }
  Vector2 movement = new Vector2(deltaX, _body.velocity.y);
```

Creando el personaje

- El siguiente paso es dar al personaje la capacidad de saltar
- Vuelve a poner a 1 el parámetro Gravity Scale en el RigidBody del personaje
- Comprueba cómo afecta la gravedad al personaje
  - Puedes incrementar o decrementar la gravedad globalmente en Edit\Project Settings\Physics 2D o únicamente para el personaje con un Gravity Scale mayor a uno

Creando el personaje

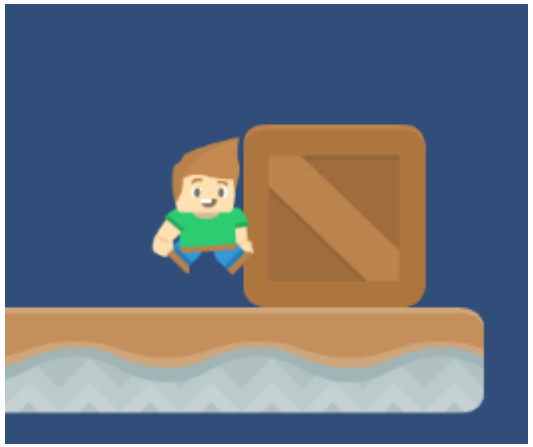
- La acción de saltar se consigue añadiendo un impulso vertical al RigidBody del personaje
  - La velocidad vertical en cada momento la calculará el motor de física
  - Ajusta la fuerza de salto en el editor

```
public float jumpForce = 10.0f;

void Update() {
  […]
  Vector2 movement = new Vector2(deltaX, _body.velocity.y);
  _body.velocity = movement;

  if (Input.GetKeyDown(KeyCode.Space)) {
     _body.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
  }
}
```

PlatformerPlayer.cs

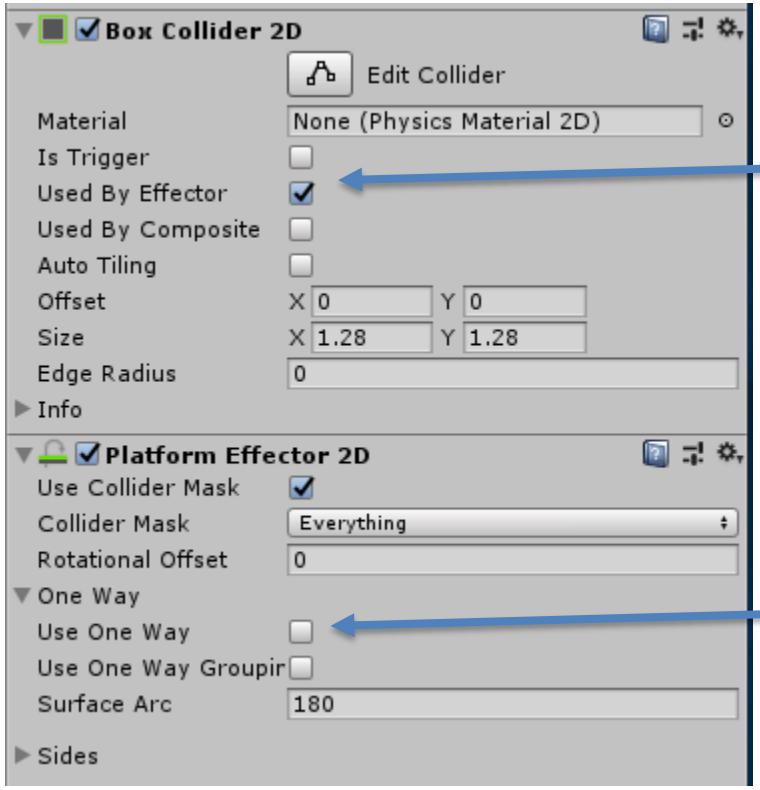
Creando el personaje

- Hay un problema: al caer al lado de una plataforma, el personaje se puede quedar pegado al lado si se avanza en esa dirección:



Creando el personaje

- Para resolver el problema, añade un componente `Platform Effector 2D` a todas las plataformas



Activa esta opción

Desactiva “Use One Way”

Creando el personaje

- El problema del código anterior es que el jugador puede saltar en el aire
- Para arreglarlo, hay que saber cuándo el personaje está sobre una plataforma

PlatformerPlayer.cs

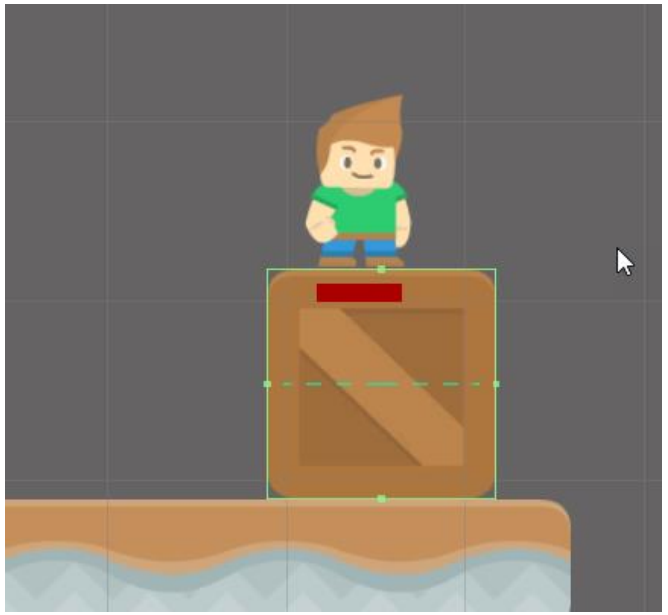
```
private BoxCollider2D _box;

void Start() {
  […]
  _box = GetComponent<BoxCollider2D>();
}
void Update() {
  […]
  Vector2 movement = new Vector2(deltaX, _body.velocity.y);
  _body.velocity = movement;
  Vector3 max = _box.bounds.max;
  Vector3 min = _box.bounds.min;
  Vector2 corner1 = new Vector2(max.x, min.y - .1f);
  Vector2 corner2 = new Vector2(min.x, min.y - .2f);
  Collider2D hit = Physics2D.OverlapArea(corner1, corner2);
  bool grounded = (hit != null);
  if (grounded &&
    Input.GetKeyDown(KeyCode.Space)) {
    _body.AddForce(Vector2.up * jumpForce,
    ForceMode2D.Impulse);
  }
}
```

Depurando en la ventana Escena

- Unity ofrece una forma fácil de añadir elementos propios a la vista de la escena
- Un Gizmo es un elemento que ayuda durante la construcción la escena (los hay predefinidos, pero también podemos añadir los nuestros)

```
private void OnDrawGizmos() {
  if (_box != null) {
    Gizmos.color = Color.red;
    Vector3 max = _box.bounds.max;
    Vector3 min = _box.bounds.min;
    Vector3 corner1 = new Vector3(max.x, min.y - .1f, 0);
    Vector3 corner2 = new Vector3(min.x, min.y - .2f, 0);
    Gizmos.DrawCube((corner1 + corner2) * 0.5f, corner2 - corner1);
  }
}
```

PlatformerPlayer.cs



Depurando en la ventana Escena

Gizmos

class in UnityEngine / Implemented in:UnityEngine.CoreModule

Description

Gizmos are used to give visual debugging or setup aids in the scene view.

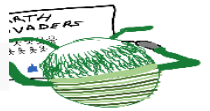
All gizmo drawing has to be done in either OnDrawGizmos or OnDrawGizmosSelected functions of the script.
OnDrawGizmos is called every frame. All gizmos rendered within OnDrawGizmos are pickable. OnDrawGizmosSelected is called only if the object the script is attached to is selected.

Static Properties

color	Sets the color for the gizmos that will be drawn next.
matrix	Set the gizmo matrix used to draw all gizmos.

Static Methods

DrawCube	Draw a solid box with center and size.
DrawFrustum	Draw a camera frustum using the currently set Gizmos.matrix for it's location and rotation.
DrawGUITexture	Draw a texture in the scene.
DrawIcon	Draw an icon at a position in the scene view.
DrawLine	Draws a line starting at from towards to.
DrawMesh	Draws a mesh.
DrawRay	Draws a ray starting at from to from + direction.
DrawSphere	Draws a solid sphere with center and radius.
DrawWireCube	Draw a wireframe box with center and size.
DrawWireMesh	Draws a wireframe mesh.
DrawWireSphere	Draws a wireframe sphere with center and radius.



Añadiendo una plataforma inclinada

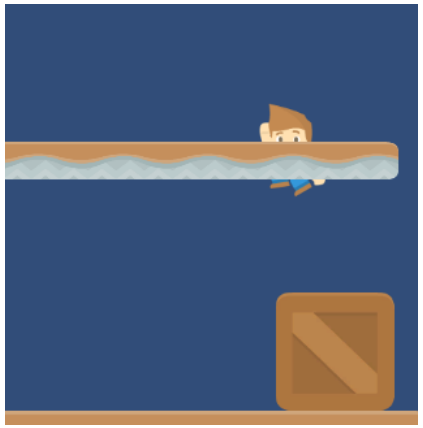
- Duplica el suelo y añade una inclinación al nuevo objeto
- El personaje puede saltar sobre la rampa y andar hacia arriba y hacia abajo, pero al quedarse quieto la gravedad hace que caiga poco a poco
- Para solucionarlo, sólo hay que desactivar la gravedad sobre el personaje cuando esté sobre el suelo y cuando esté quieto:

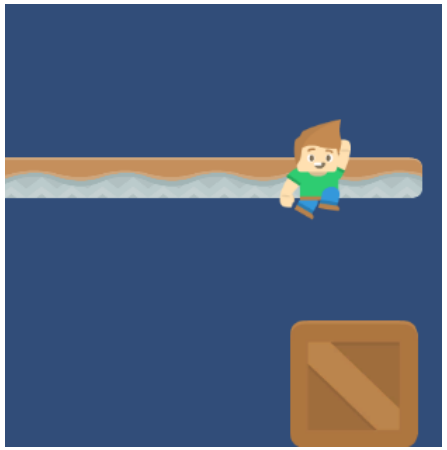
```
[…]
bool grounded = (hit != null);
_body.gravityScale = grounded &&  Mathf.Approximately(deltaX, 0.0f)? 0.0f : 1.0f;
if (grounded && Input.GetKeyDown(KeyCode.Space)) {
  _body.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
```

PlatformerPlayer.cs

Plataformas unidireccionales

- Las plataformas unidireccionales permiten atravesarlas desde abajo, pero no desde arriba
- Duplica la plataforma suelo y activa la opción de `Use One Way` del componente `Platform Effector 2D`
- Ahora el personaje puede atravesar la plataforma desde abajo
- Puede aparecer un problema en el orden de dibujado:
  - Establece el valor de `Order in Layer` del `Sprite Renderer` del personaje a `1`





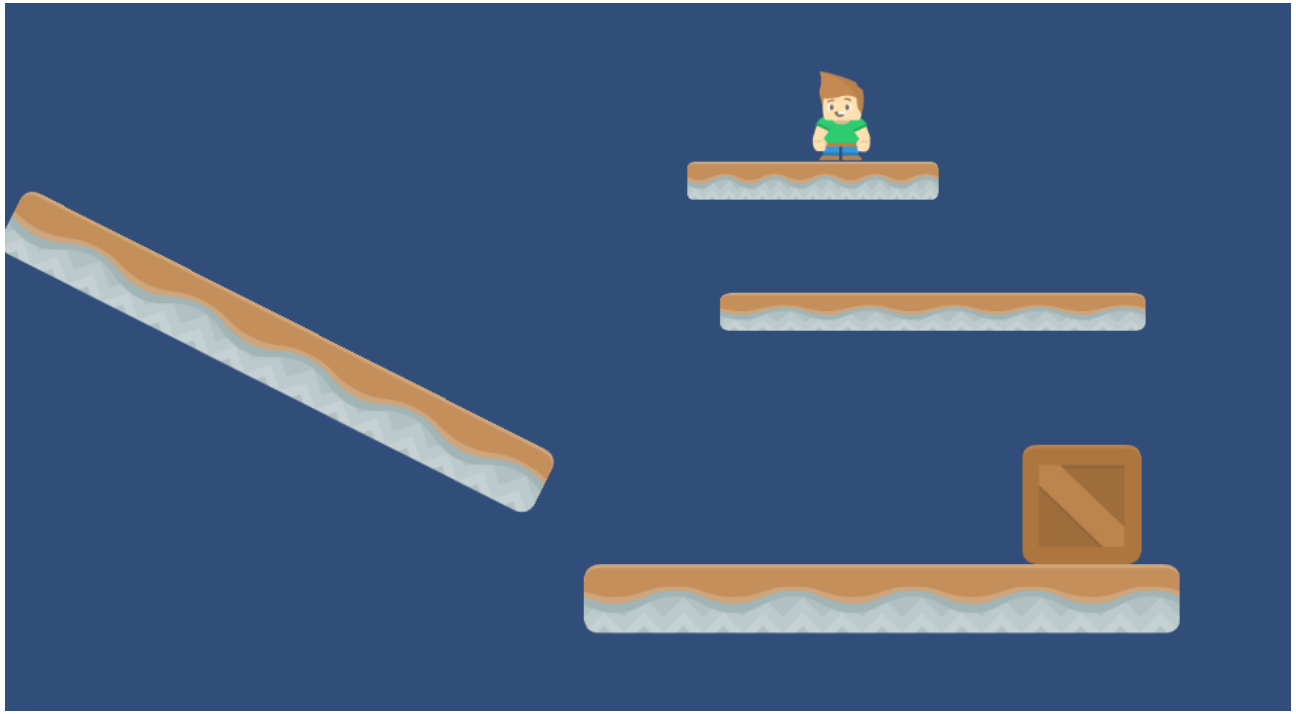
Plataformas móviles

- Otro tipo de plataforma muy común son las plataformas móviles
- Primero, vamos a crear un script para mover la plataforma:

```
public class MovingPlatform : MonoBehaviour {
  public GameObject finalPosition;
  public float speed = 0.5f;
  private Vector3 _startPos;
  private float _trackPercent = 0.0f;
  private int _direction = 1;

  void Start() {
    _startPos = transform.position;
  }
  void Update() {
    if (finalPosition == null) return;
    _trackPercent += _direction * speed * Time.deltaTime;
    Vector3 pos = (finalPosition.transform.position - _startPos) * _trackPercent + _startPos;
    pos.z = _startPos.z;
    transform.position = pos;

    if ((_direction == 1 && _trackPercent > 0.9f) ||
          (_direction == -1 && _trackPercent < 0.1f))
          {
              _direction = -_direction;
          }
      }
  }
```

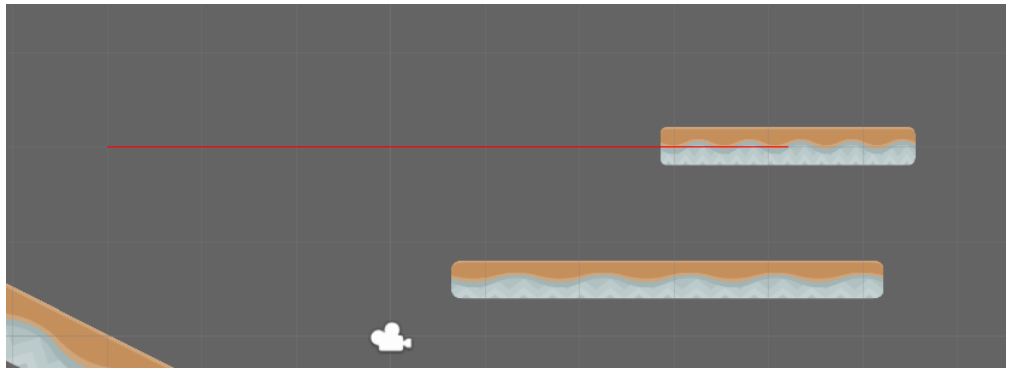

Plataformas móviles

- Duplica una plataforma, asígnale el script y configura los parámetros para que se mueva por la escena
- Intenta subir a la plataforma



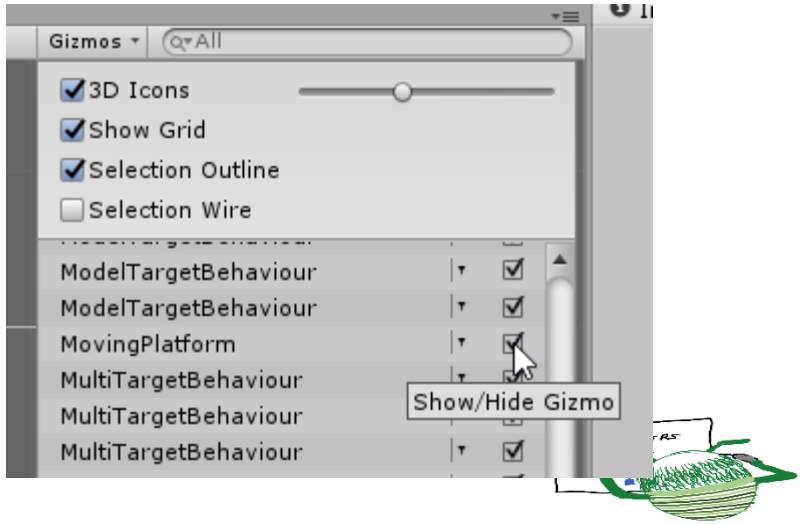
Plataformas móviles

- Podemos añadir un Gizmo al script para visualizar el camino que seguirá la plataforma:



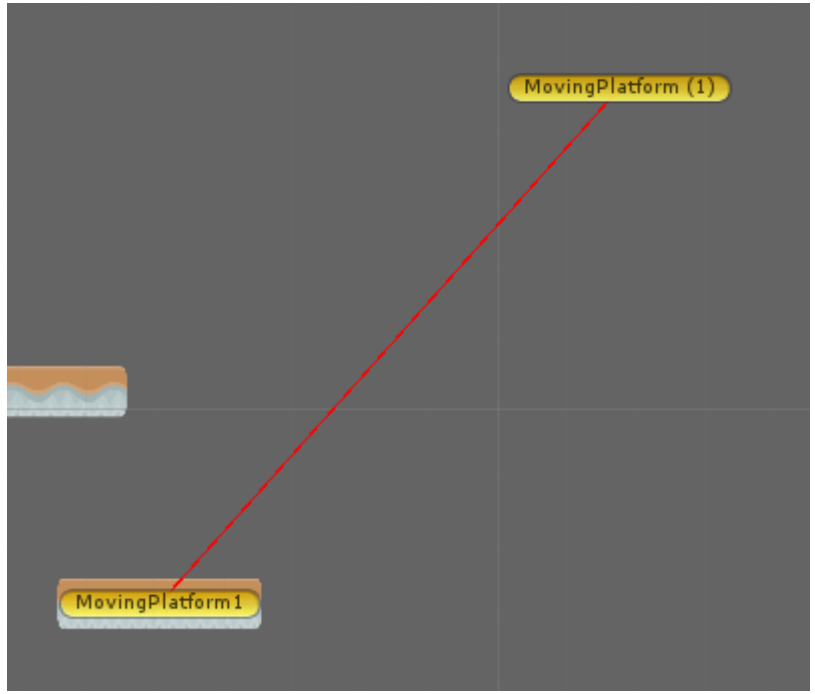
```
private void OnDrawGizmos() {
  Gizmos.color = Color.red;
  if (Application.isPlaying)
    Gizmos.DrawLine(_startPos, finalPosition.transform.position);
  else
    Gizmos.DrawLine(transform.position, finalPosition.transform.position);
}
```

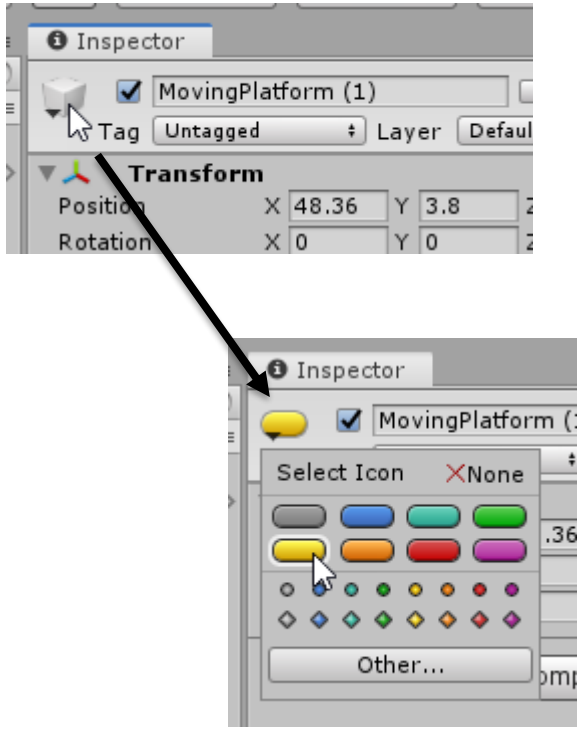
- En el editor también se puede deshabilitar, en el menú Gizmos:



Plataformas móviles

- También se pueden asociar iconos o etiquetas a los gameobjects de una escena (incluyendo los empties) para hacerlos más visibles:





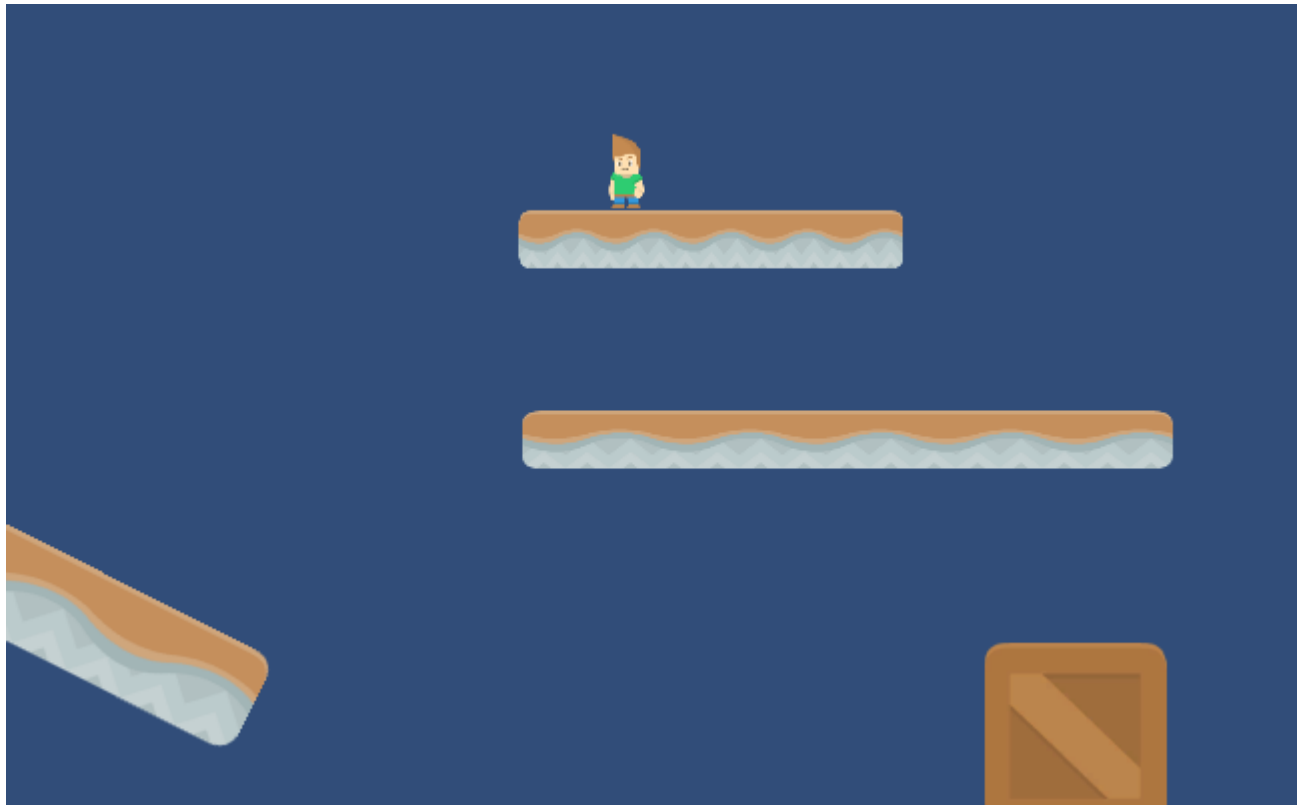
Plataformas móviles

- Para hacer que el personaje se quede fijo sobre una plataforma móvil:

```
if (grounded && Input.GetKeyDown(KeyCode.Space)) {
  _body.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}

MovingPlatform platform = null;
if (hit != null) {
  platform = hit.GetComponent<MovingPlatform>();
}
if (platform != null) {
  transform.parent = platform.transform;
} else {
  transform.parent = null;
}
```

PlatformerPlayer.cs

Plataformas móviles



Plataformas móviles

- Para arreglar el problema de la escala:

PlatformerPlayer.cs

```
MovingPlatform platform = null;
Vector3 pScale = Vector3.one;
if (hit != null) {
  platform = hit.GetComponent<MovingPlatform>();
}
if (platform != null) {
  transform.parent = platform.transform;
  pScale = platform.transform.localScale;
} else {
  transform.parent = null;
}

_anim.SetFloat("speed", Mathf.Abs(deltaX));
if (!Mathf.Approximately(deltaX, 0)) {
  transform.localScale = new Vector3(Mathf.Sign(deltaX) / pScale.x, 1 / pScale.y, 1);
}
```


Control de la cámara

- El siguiente paso de esta demo va a ser hacer que cámara siga al personaje
  - Asigna el script a la cámara y asigna el personaje al campo target

```
public class FollowCam : MonoBehaviour {
  public Transform target;

  void LateUpdate () {
    transform.position = new Vector3(
        target.position.x, target.position.y, transform.position.z);
  }
}
```

Control de la cámara

- El movimiento de la cámara es un poco brusco
- La mayoría de los juegos de plataformas incorporan movimientos de cámara bastante complejos:
  - https://gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php

```
public float smoothTime = 0.2f;
private Vector3 _velocity = Vector3.zero;

void LateUpdate () {
  Vector3 targetPosition = new Vector3(target.position.x, target.position.y, transform.position.z);
  transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref _velocity, smoothTime);
}
```

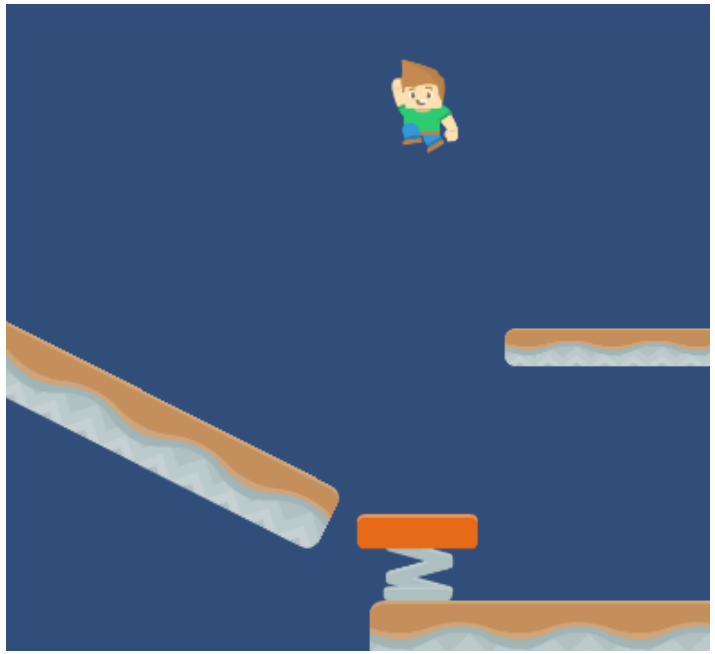
FollowCam.cs

Ejercicio

- Haz que se muestre la animación de salto cuando el personaje esté en el aire (consejo: añade un parámetro booleano a la máquina de estados)

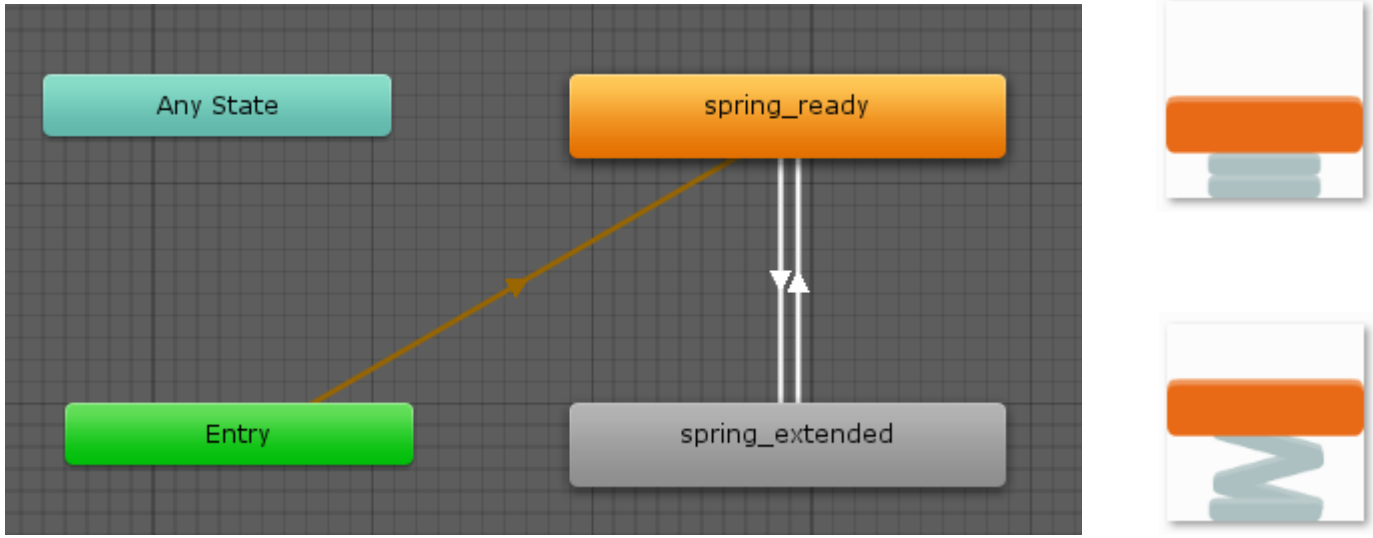
Trampolín

- Vamos a implementar una plataforma trampolín, que cada vez que el personaje caiga encima, le lance verticalmente hacia arriba con más fuerza que un salto normal



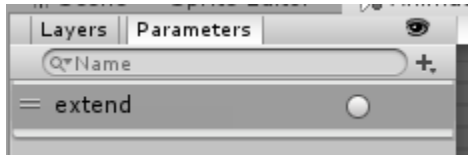
Trampolín

- Importa los sprites al proyecto y arrastra el sprite que usarás para representar el trampolín en la escena
- Selecciona el game object en la escena y crea dos animaciones, con un único frame:



Trampolín

- En este caso, la animación queremos que se inicie, y luego por tiempo, vuelva al estado inicial
- Para lanzar la animación definiremos un parámetro en el Animation Controller de tipo Trigger:



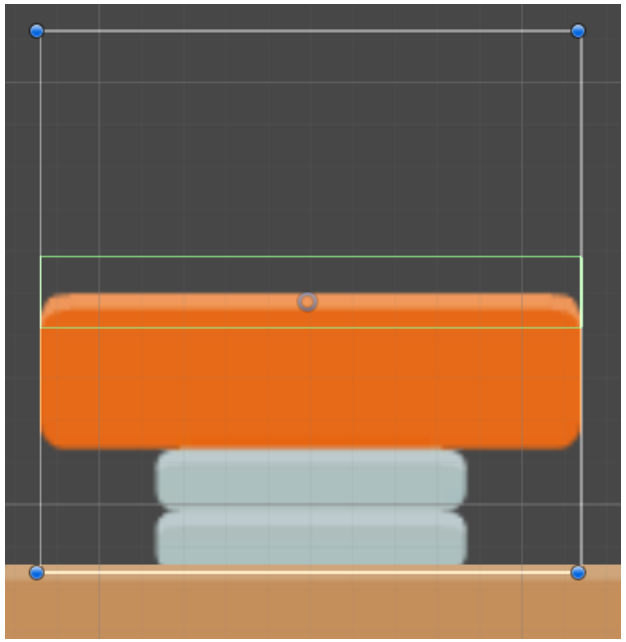
  - Un parámetro de tipo Trigger es como uno booleano, pero que cuando se pone a true, cambia automáticamente a false en el siguiente frame

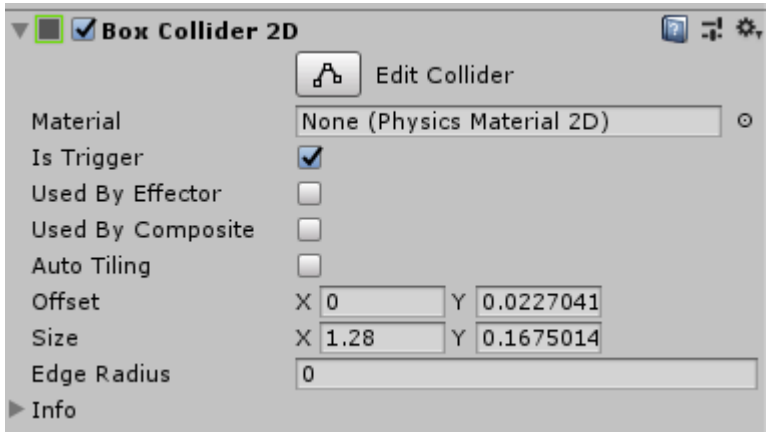
Trampolín

- Establece las propiedades de las transiciones:
  - De *spring_ready* a *spring_extended*:
    - Has Exit Time = false
    - Transition Duration = 0
    - Conditions: extend
  - De *spring_extended* a *spring_ready*:
    - Has Exit Time = true
    - Fixed Duration = true
    - Transition Duration = <cuánto tiempo debe permanecer extendido>

Trampolín

- Para activar el trigger, vamos a añadir un Box Collider 2D al trampolín, para detectar al jugador:





Trampolín

- Preparamos el script del jugador para que otros elementos de la escena puedan lanzarlo hacia arriba:

PlatformerPlayer.cs

```
void Update() {
  […]
  bool grounded = (hit != null);
  if (grounded && Input.GetKeyDown(KeyCode.Space)) {
    _body.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
    Jump(jumpForce);
  }
 […]
}

public void Jump(float force) {
  _body.AddForce(Vector2.up * force, ForceMode2D.Impulse);
}
```

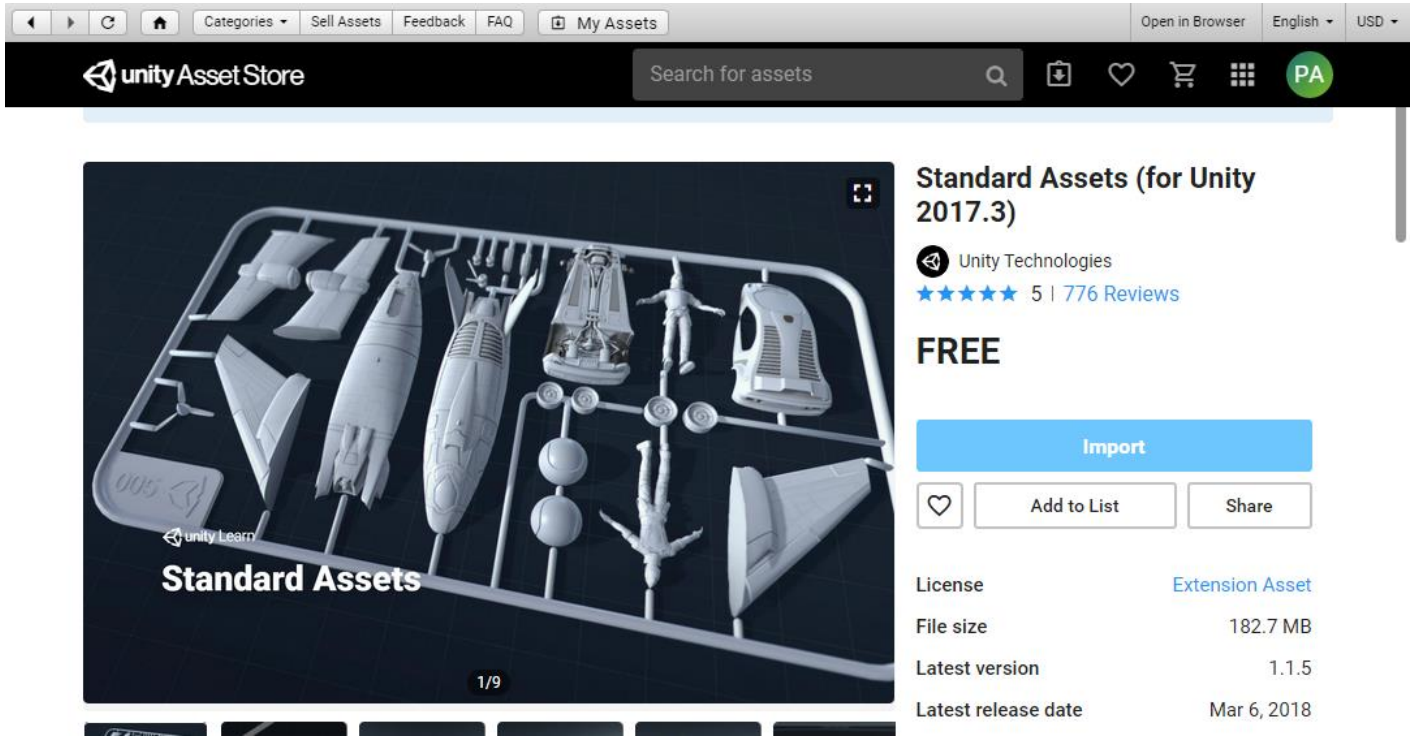

Trampolín

- Añadimos el siguiente script al trampolín:

```
public class Eject : MonoBehaviour {
  public float springForce = 30.0f;
  private Animator _anim;

  private void Awake() {
    _anim = GetComponent<Animator>();
  }
  private void OnTriggerEnter2D(Collider2D other) {
    if (other.CompareTag("Player")) {
      PlatformerPlayer player = other.GetComponent<PlatformerPlayer>();
      player.Jump(springForce);
      _anim.SetTrigger("extend");
    }
  }
}
```

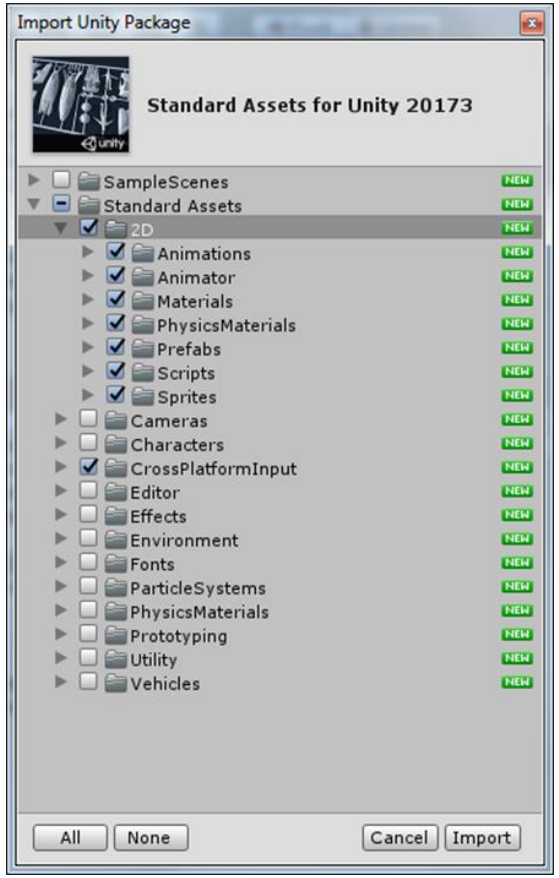
El manejador de Unity

- El paquete de recursos estándar de Unity trae la implementación de un personaje 2D que puedes utilizar
- Baja el paquete Estándar Assets de la Asset Store



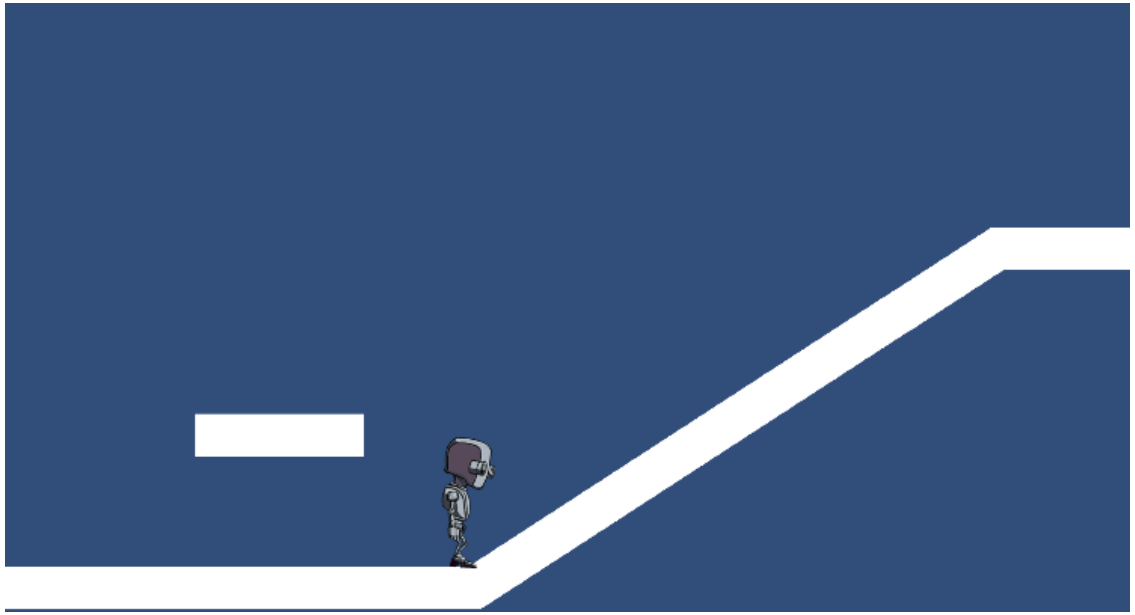
El manejador de Unity

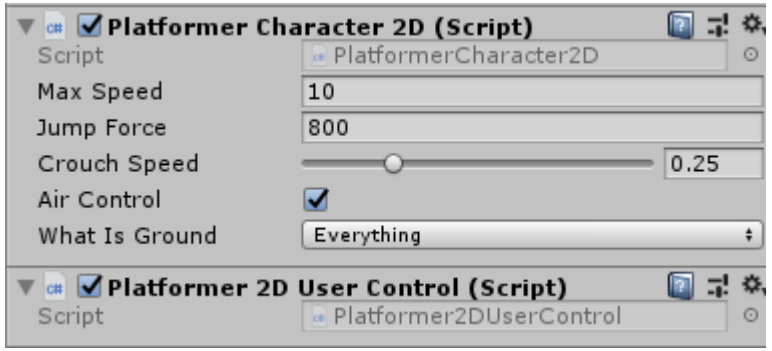
- Importa el apartado de 2D y CrossPlatformInput:



El manejador de Unity

- Crea una nueva escena y arrastra alguna plataforma desde la carpeta de prefabs y al personaje CharacterRobotBoy





El manejador de Unity

- Brackeys ha adaptado el manejador de Unity y lo explica en el siguiente vídeo:

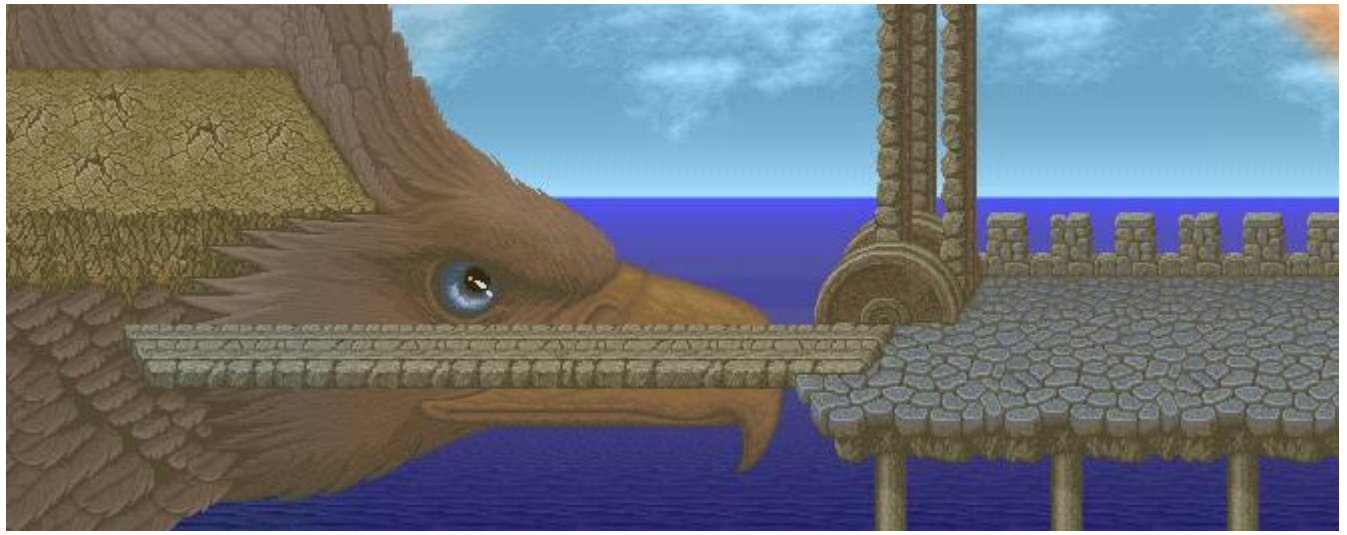
2D Movement in Unity
(Tutorial)

https://www.youtube.com/watch?v=dwcT-Dch0bA

https://github.com/Brackeys/2D-Character-Controller

El fondo

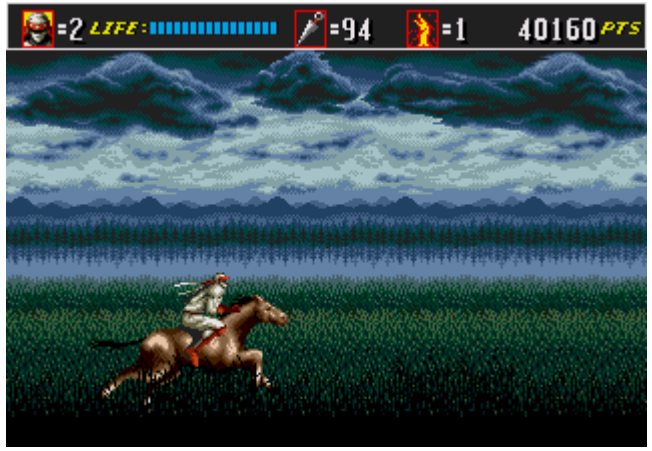
- El fondo del juego proporciona la escena donde se desarrolla la acción
- Normalmente no es interactivo
- Puede ser estático o dinámico
- Normalmente aparece por detrás de los personajes



El fondo

Parallax scrolling

- Los juegos 2D usan normalmente el paralaje y múltiples capas para simular movimiento y añadir profundidad a la escena



Shinobi III

Shadow of the Beast

El fondo

Parallax scrolling

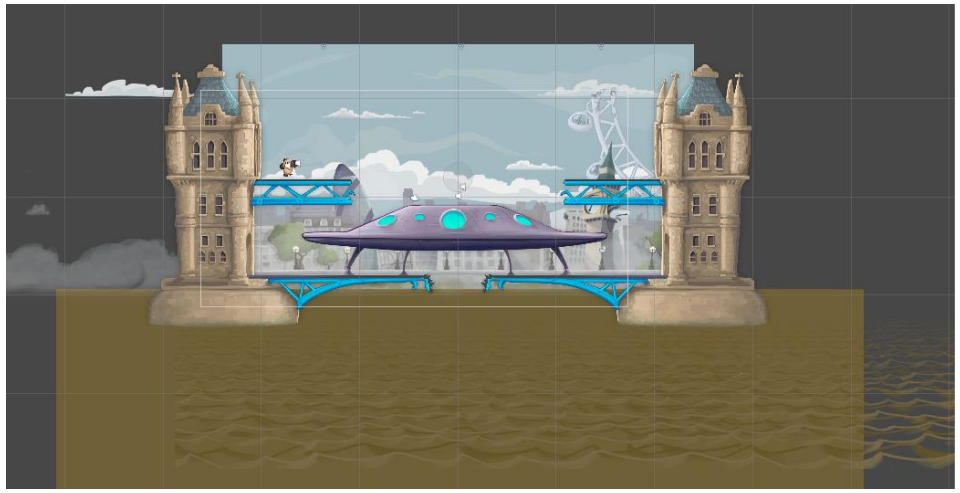
- Los juegos 2D usan normalmente el paralaje y múltiples capas para simular movimiento y añadir profundidad a la escena

Muramasa The Demon Blade

Street Fighter 2

El fondo

- Los fondos se implementan en Unity como:
  - Una o más imágenes superpuestas
  - Tiles

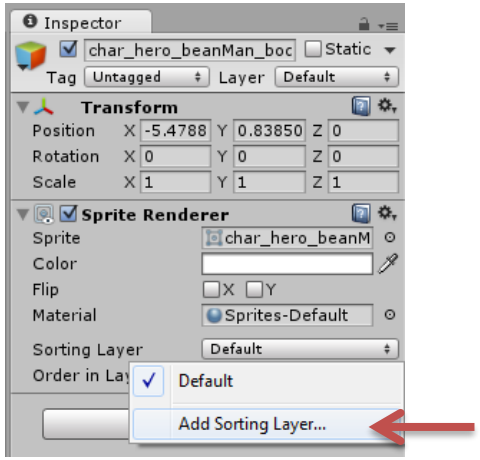


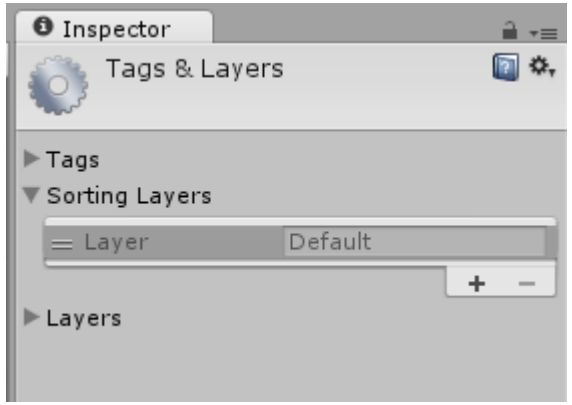
2D Platformer (Unity demo)

The Legend of Zelda

Sorting Layers

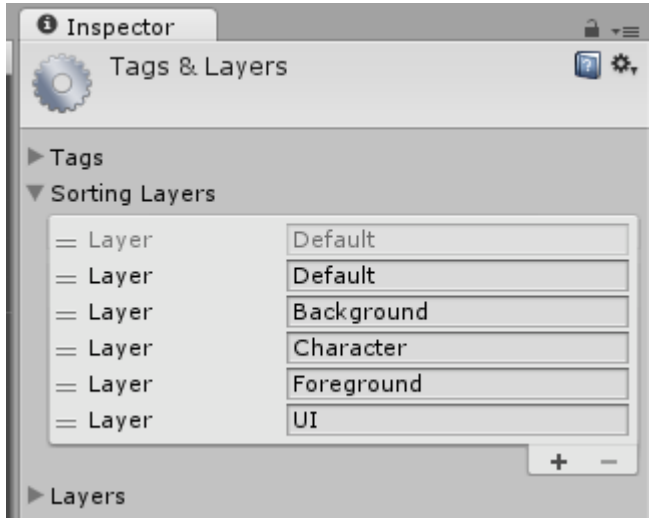
- Unity proporciona una herramienta para definir la ordenación relativa entre los elementos de la escena
- Sorting layers
  - El sprite renderer lo usa para determinar el orden de las capas independientemente de su distancia a la cámara
  - Se pueden definir las sorting layers desde el componente Sprite Renderer o desde Edit\Project Settings\Tags & Layers





Sorting Layers

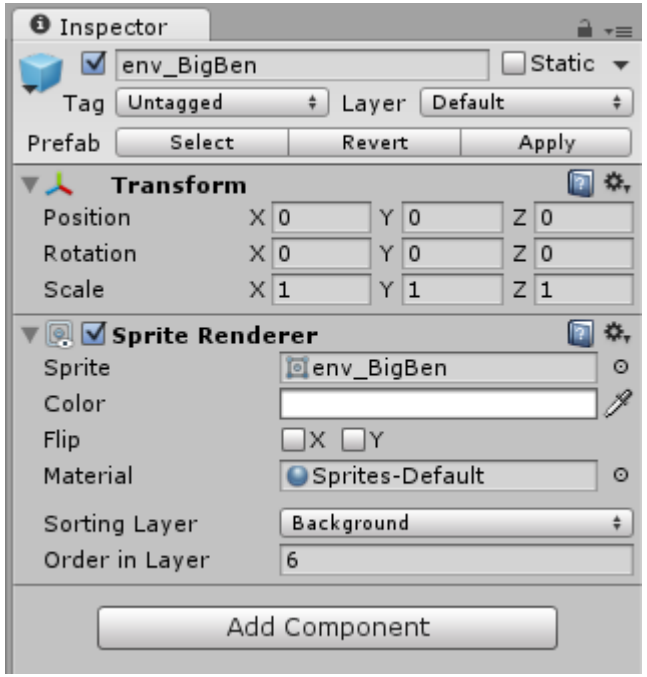
- Es habitual tener varias sorting layers para organizar los sprites de una escena:



- Los elementos del ejemplo se dibujaran en el orden: Default, Background, Character, Foreground, etc. Los elementos de una capa aparecerán por encima de los de las capas anteriores

Sorting Layers

- Una sorting layer normalmente tiene varios sprites. El orden de dibujado de dichos sprites se puede definir en el campo “Order in Layer”
  - Se dibujan en orden creciente



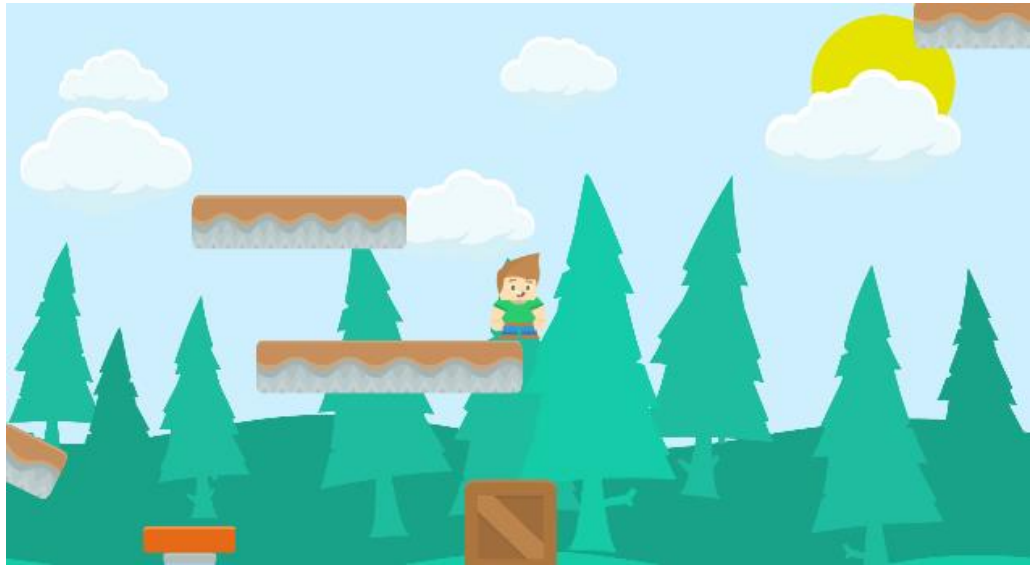
Sorting Layers

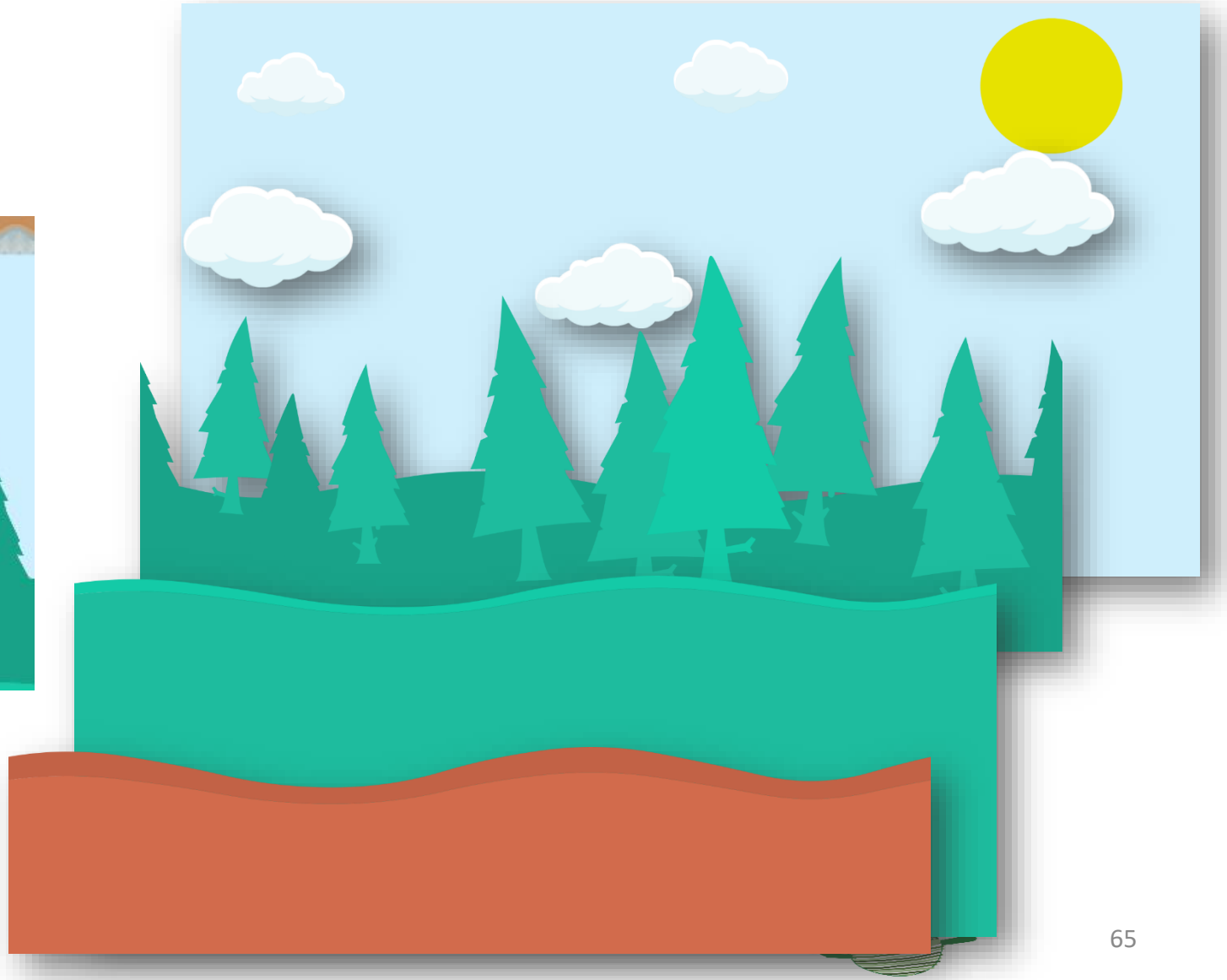
- Una vez que hemos completado una capa (por ejemplo, el fondo), se puede hacer que sus elementos no se puedan seleccionar desde el desplegable Layers del Editor:

Los elementos aún se pueden seleccionar en el panel de la jerarquía

- Resumiendo, el orden relativo entre sprites viene determinado por la siguiente precedencia:
  1. Sorting Layer
  2. Order in Layer
  3. Coordenada Z

Parallax scrolling



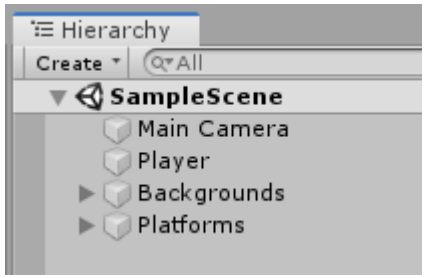


Parallax scrolling

- Importa las imágenes al proyecto
- Define un valor de PPU común, de tal forma que b0 ocupe toda la ventana
- Crea las siguientes sorting layers:
  - Default
  - Background
  - Platforms
  - Player
  - Foreground
- Añade las capas del fondo, definiendo sus sorting layers y el orden dentro de la capa (b0 a b3 en Background, b4 en Foreground).
  - Asigna al jugador y las plataformas a sus respectivas capas

Parallax scrolling

- El parallax scrolling funciona haciendo que las distintas capas se muevan a distinta velocidad
  - Las capas más cercanas a la cámara se mueven más rápido que las más lejanas
  - Vamos a hacer que b0 se mueva a la misma velocidad que la cámara, y que b5 no se mueva
    - Para el jugador, b0 no se mueve y b5 se mueve a la misma velocidad que la cámara
  - b1 a b3 se moverán a una fracción de la velocidad de la cámara
- Sitúa todas las capas en la escena
- Añade una plataforma larga en la base de la escena, para que el personaje pueda andar
- Puedes organizar la escena para que quede más clara:



Parallax scrolling

- Añade el siguiente script a b0 .. b5

```
public class ParallaxHorizontal : MonoBehaviour {
  private float startPosition;
  private Transform cam;
  public float parallaxFraction;

  void Start() {
    startPosition = transform.position.x;
    cam = Camera.main.transform;
  }

  void LateUpdate() {
    float offset = (cam.position.x * parallaxFraction);
    transform.position = new Vector3(startPosition + offset,
      transform.position.y,
      transform.position.z);
  }
}
```

Parallax scrolling

- Define las siguientes fracciones de paralaje (o ajústalas a tu gusto):
  - b0: 1, b1: 0.8, b2: 0.6, b3: 0.2, b4: 0

Parallax scrolling

- El siguiente paso es conseguir un scroll infinito:
  - Triplica cada fondo, poniendo una copia de cada imagen a izquierda y derecha de la original
  - Quita el script a las copias, y hazlas hijas del original

Parallax scrolling

```
public class ParallaxHorizontal : MonoBehaviour {
  private float width, startPosition;
  private Transform cam;
  public float parallaxFraction;
  void Start() {
    startPosition = transform.position.x;
    width = GetComponent<SpriteRenderer>().bounds.size.x;
    cam = Camera.main.transform;
  }
  void LateUpdate() {
    float offset = (cam.position.x * parallaxFraction);
    float moved = cam.position.x - offset;
    if (moved > startPosition + width) startPosition += width;
    else if (moved < startPosition - width) startPosition -= width;
    transform.position = new Vector3(startPosition + offset,
      transform.position.y,
      transform.position.z);
  }
}
```

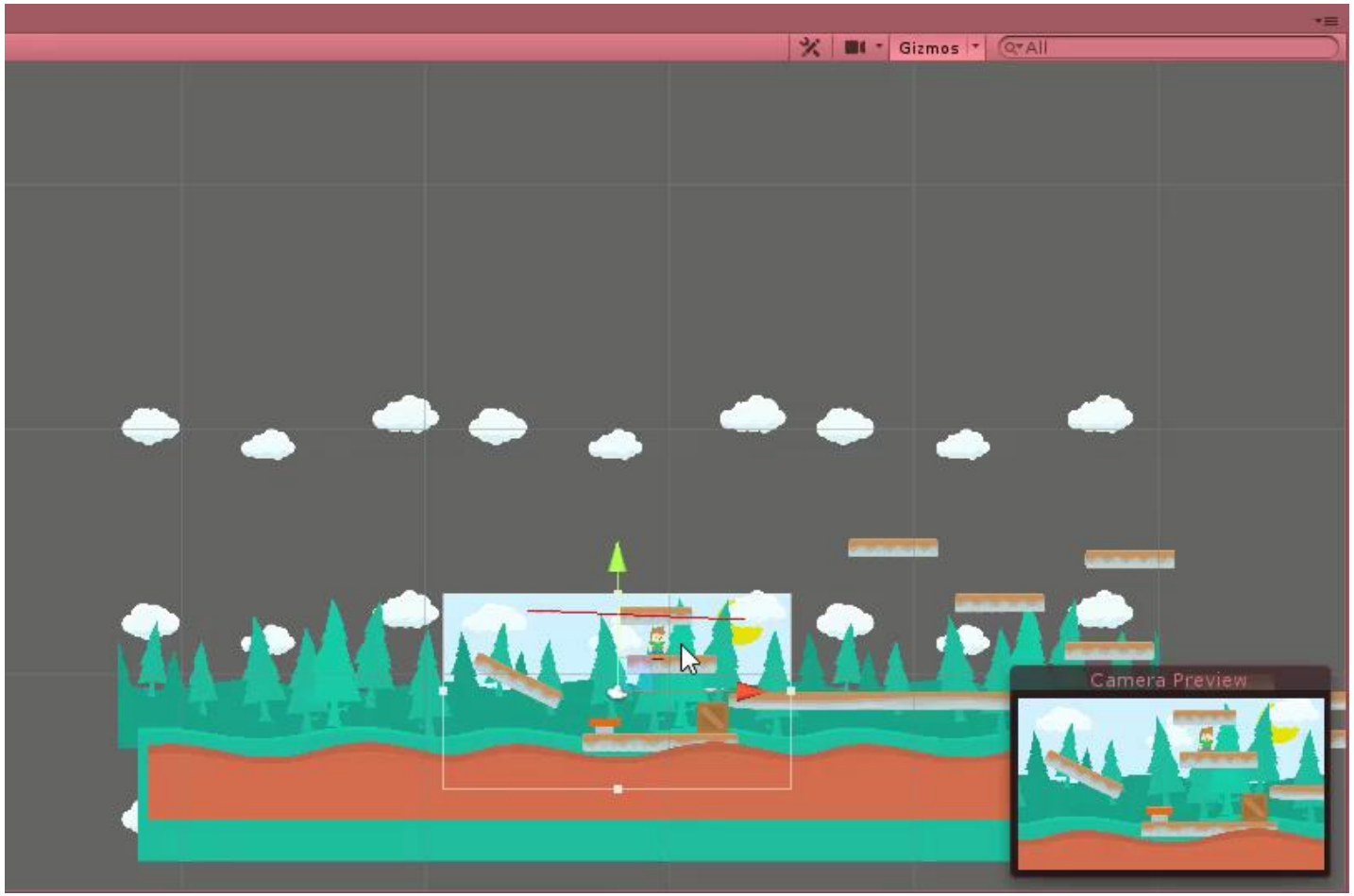
Parallax scrolling

```
public class ParallaxScroll : MonoBehaviour {
  private Vector2 size, startPosition;
  private Transform cam;
  public Vector2 parallaxFraction;
  void Start() {
    startPosition = transform.position;
    size = GetComponent<SpriteRenderer>().bounds.size;
    cam = Camera.main.transform;
  }
  void LateUpdate() {
    Vector2 offset = (cam.position * parallaxFraction);
    Vector2 temp = new Vector2(cam.position.x, cam.position.y) - offset;
    if (temp.x > startPosition.x + size.x) startPosition.x += size.x;
    else if (temp.x < startPosition.x - size.x) startPosition.x -= size.x;
    if (parallaxFraction.y > 0f && temp.y > startPosition.y + size.y)
      startPosition.y += size.y;
    else if (parallaxFraction.y > 0f && temp.y < startPosition.y - size.y) startPosition.y -= size.y;
    transform.position = new Vector3(startPosition.x + offset.x, startPosition.y + offset.y,
            transform.position.z);
    }
}
```

- Por último, vamos a dar la posibilidad de hacer scroll vertical:

Parallax scrolling

- Vamos a hacer que la capa b2 se repita también verticalmente
  - Añade copias por encima y por debajo hasta tener 9
  - Todas las copias son hijas de la central, y la central es la única que tiene el script



Ejercicio

- Completa tu nivel con nuevas plataformas
- Añade un destino, que lance un mensaje a la consola cuando el jugador lo alcance
- Implementa plataformas que se destruyen al transcurrir un tiempo después de que el jugador las pise

Bibliografía

- Joseph Hocking. Unity in Action. 2nd edition. Manning, 2018.
  - Capítulo 6

- Assets descargados de: www.kenney.nl