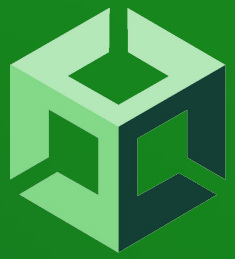
Curso de

Introducción a Unity: Scripting

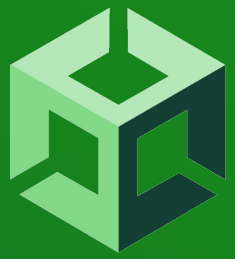
Ruth Margarita García
@pr00thmatic



¡Ahora eres pro con el Scripting!

@pr00thmatic

- **Felicitar** estudiante por acabar
- **recapitulación** ¿qué aprendi? y pa q me sirve
- invitación **Examen** y
- **calificación** curso (no olvides darme 5 estrellas en este curso)
- despedida
- seguir redes



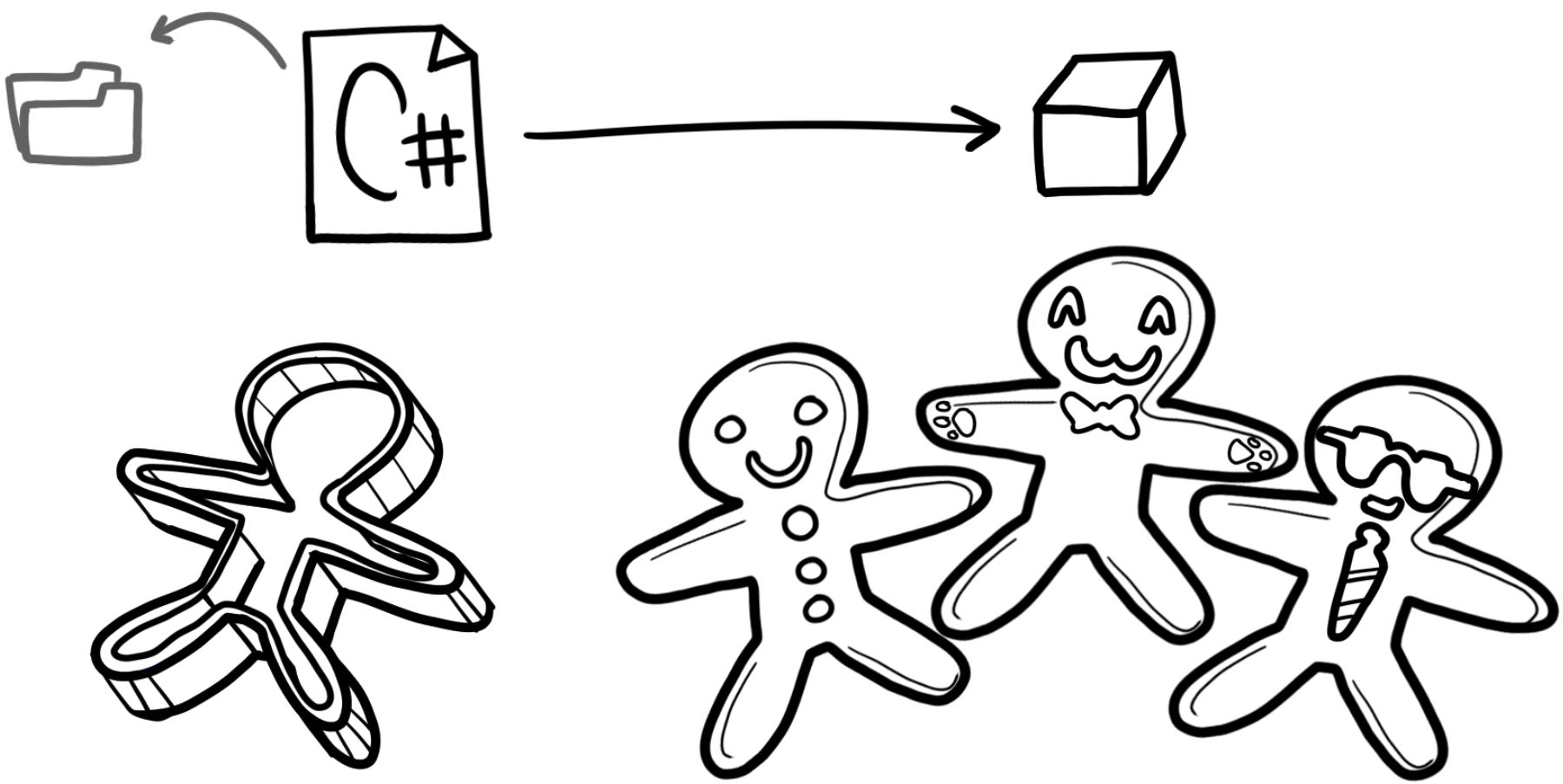
¿Qué es un script?

¿Cuál es su ciclo de vida?

“Scripts” -> MonoBehaviour

- Es un **asset**
- Se añade como **componente**
- Es una **clase** (C#)
- Hereda de **MonoBehaviour**
- **Automatizar**

Script:
¿Asset? ¿Componente?



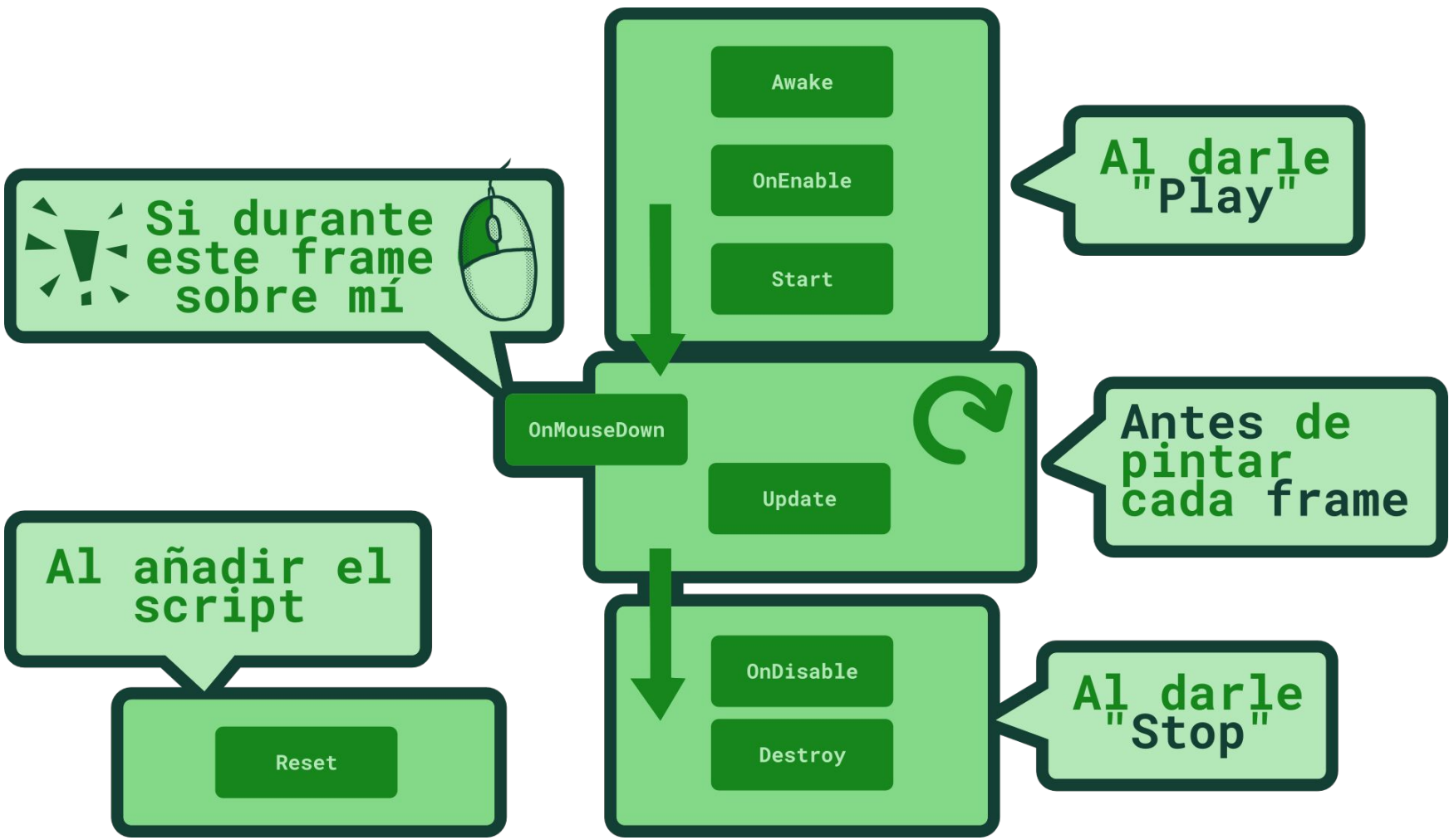
Ciclo de vida de un Script

MonoBehaviour

"Mensajes" comunes de Unity al script

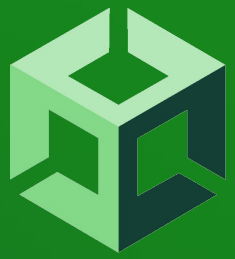
- OnEnable, OnDisable
- Start, Destroy <-
- Awake<-
- Update
- OnMouseDown <-
- Reset <-

¿En qué orden?

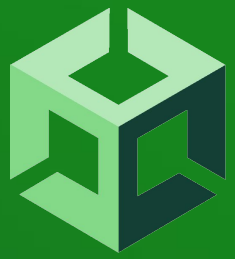

Awake
OnEnable
Start
Al darle "Play"
Si durante este frame sobre mí
OnMouseDown
Update
Antes de pintar cada frame
Al añadir el script
Reset
OnDisable
Destroy
Al darle "Stop"

¡Scripts!

- En asset != en componente
- Mensajes: Start -> Update -> Destroy
- Mensajes: Reset
- Activar y desactivar
- ¡Automatizan!



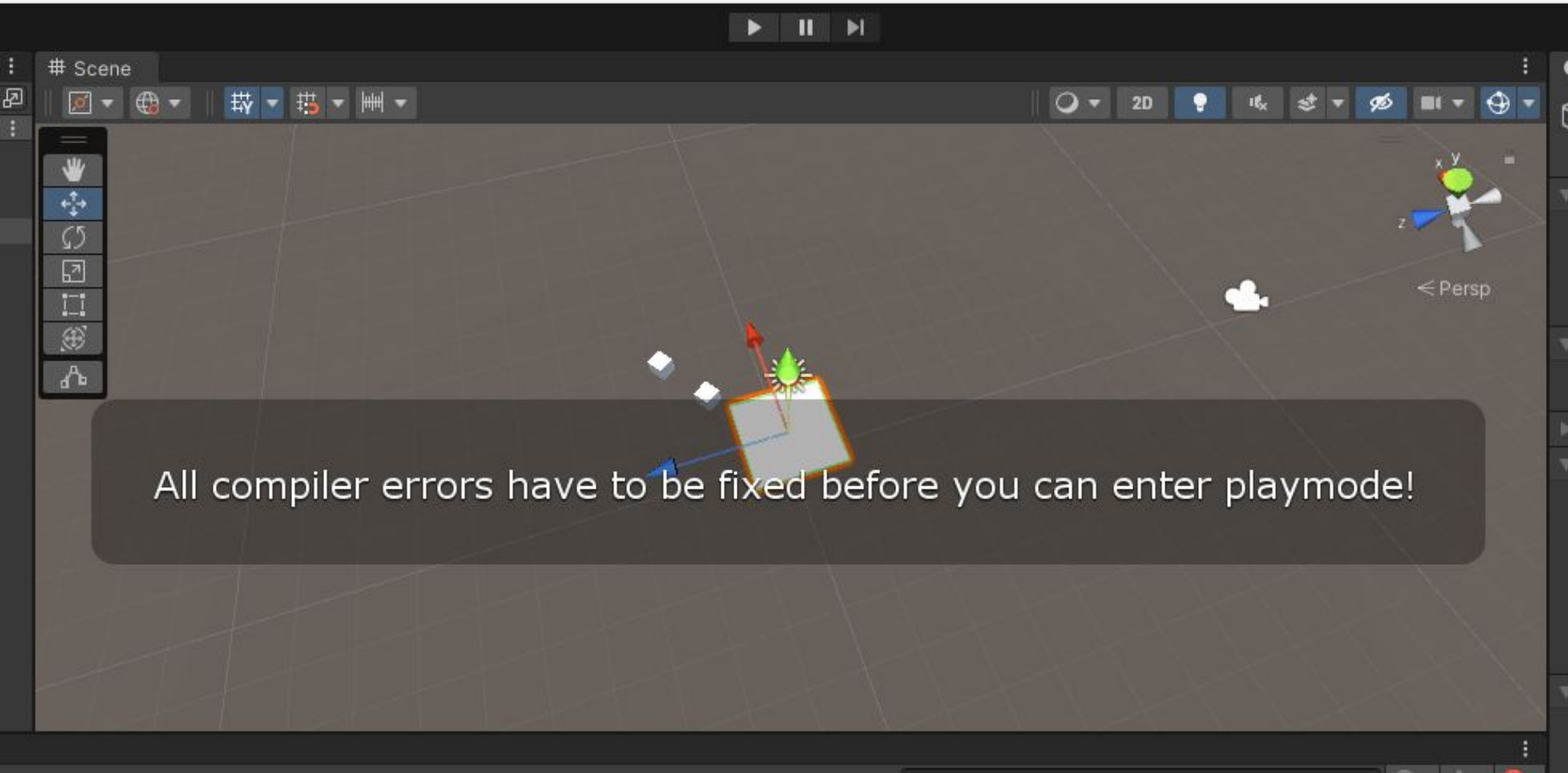
OnDisable, Destroy y OnMouseDown



Usando la consola*

*de una forma muy pro

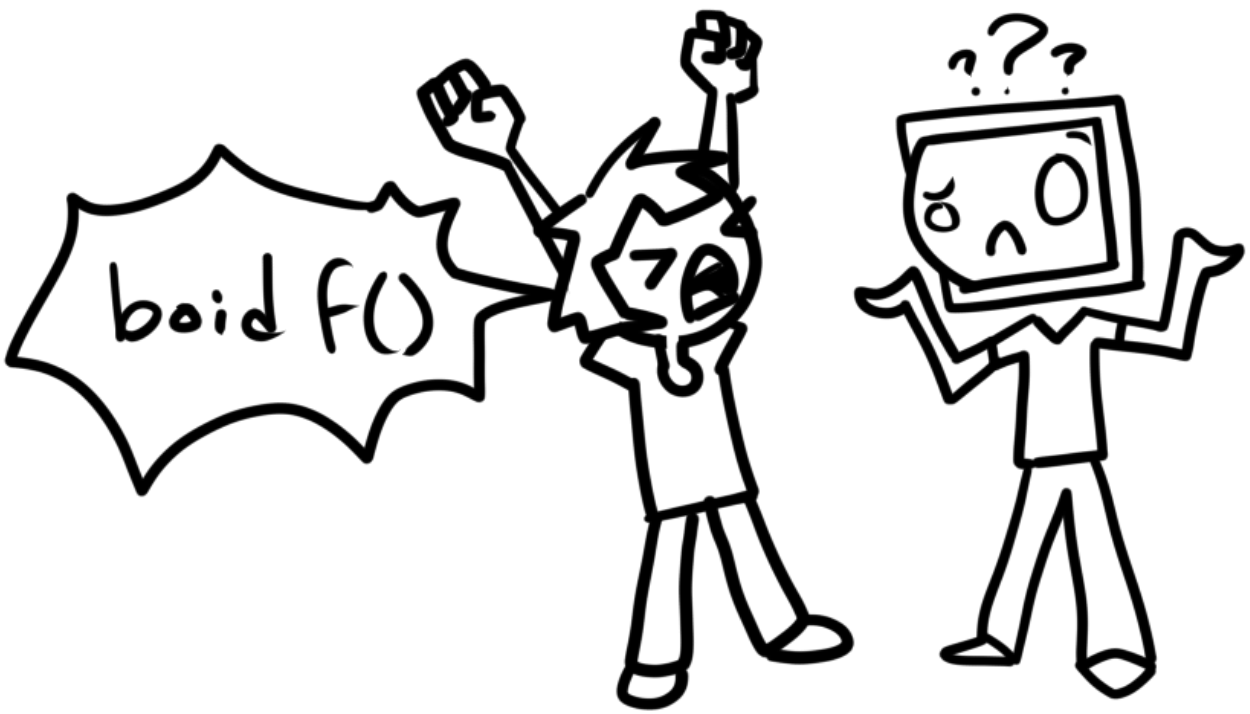
Todos nos equivocamos :(



Errores

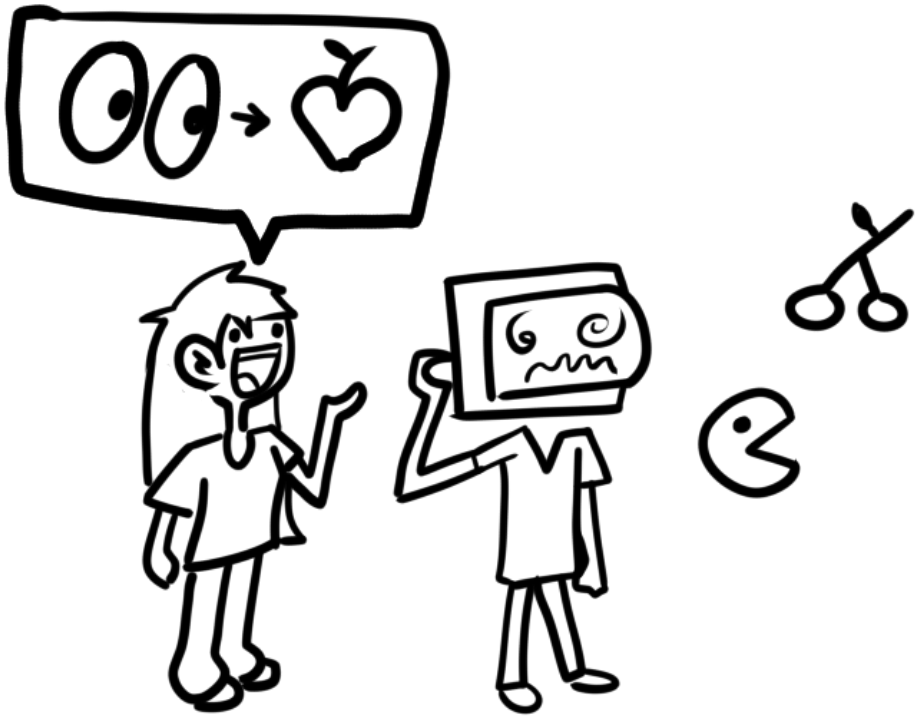
- De compilación
- En tiempo de ejecución
- De lógica

Errores de compilación



Errores en tiempo de ejecución

- No impiden la ejecución del juego
- Imprevistos
- Podrían desestabilizar el juego
- Podrían pausar el juego

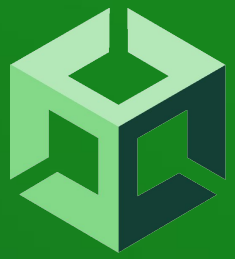


Errores de lógica

- Es culpa del programador :(
- Nos pasa a todos >_<
- ¡Se encuentran! >:)
- ¡Se arreglan! >:D

¡Eres pro con la consola!

- 3 tipos de errores
- Debug.Break
- Debug.Log(mensaje, foco)
- Pausa, Paso a paso, Play
- Pause on error

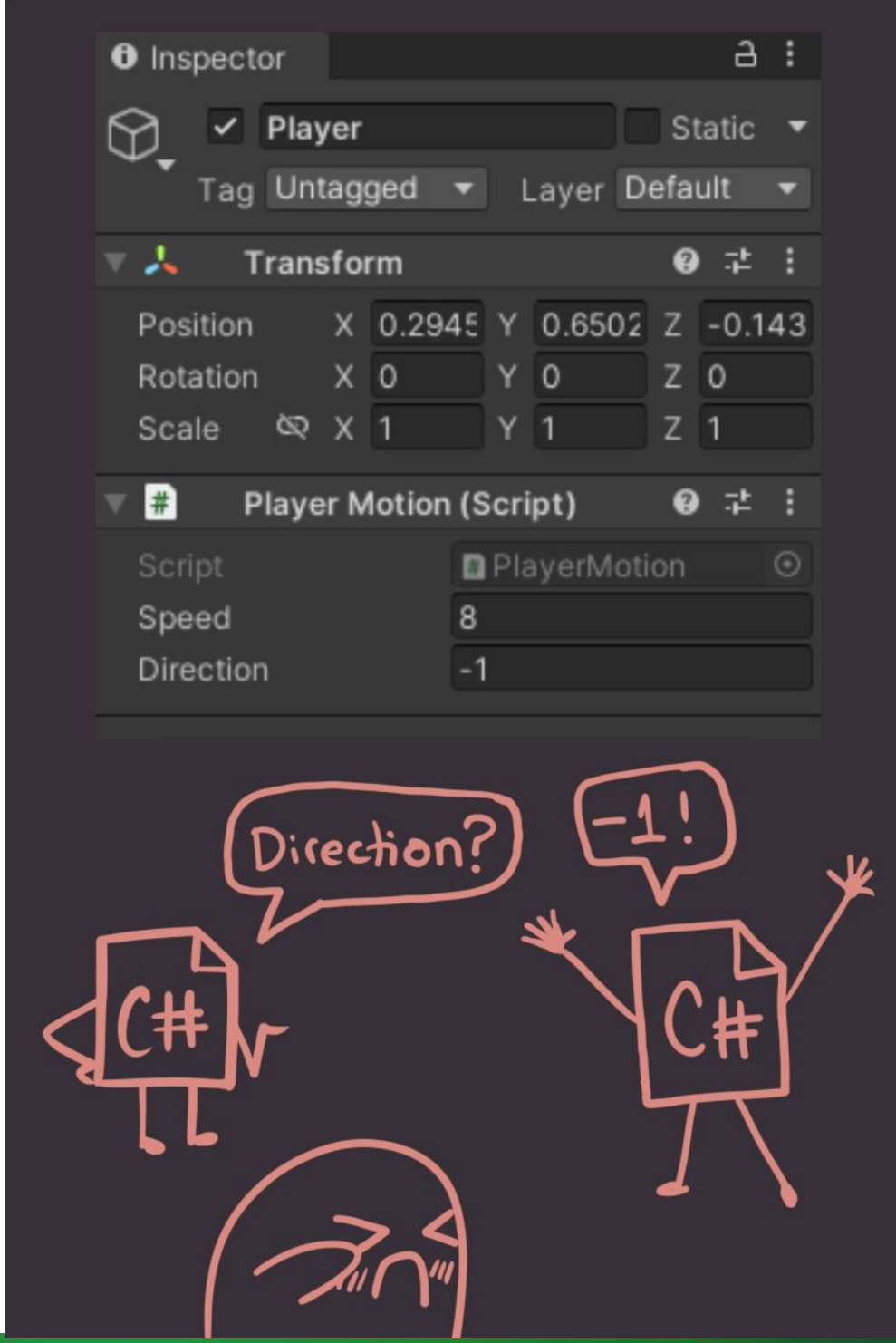


Variables públicas y privadas

[SerializeField], [Header] y [HideInInspector]

Públicas

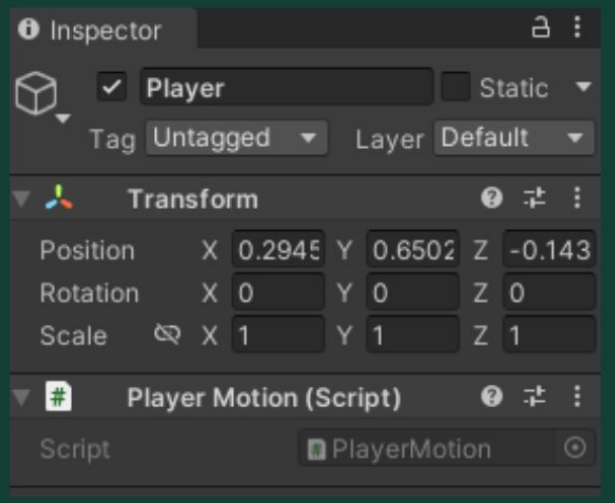
- Se ven en el **Inspector**
- Otros **scripts** pueden **verlas** y **modificarlas**

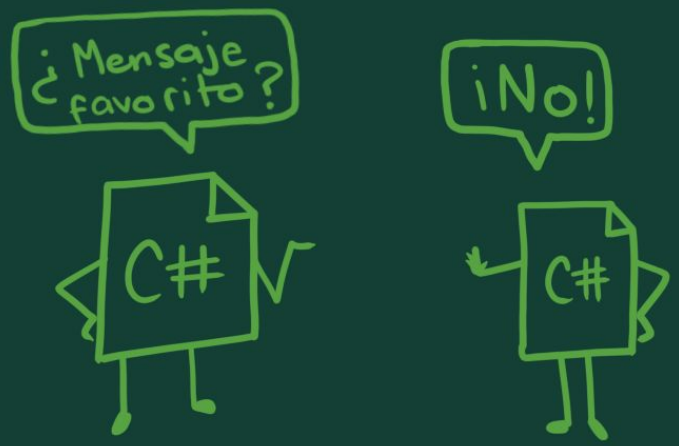


Privadas y protegidas

- No se ven en el inspector*
- Otras clases no pueden verlas*

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class PlayerMotion : MonoBehaviour {
  private int currentMessage = 2;
  string[] messages; // this is privated
  protected int favoriteMessage = 1;
}
```



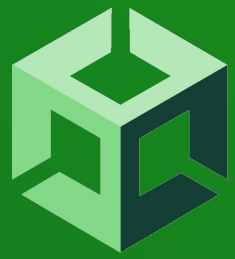


Atributos

- [SerializeField]
- [HideInInspector]
- [Range(n, m)]
- [TextArea]
- [Header(“título”)] y [Space(px)]
- [Tooltip(“texto”)

¡Ahora puedes configurar scripts!

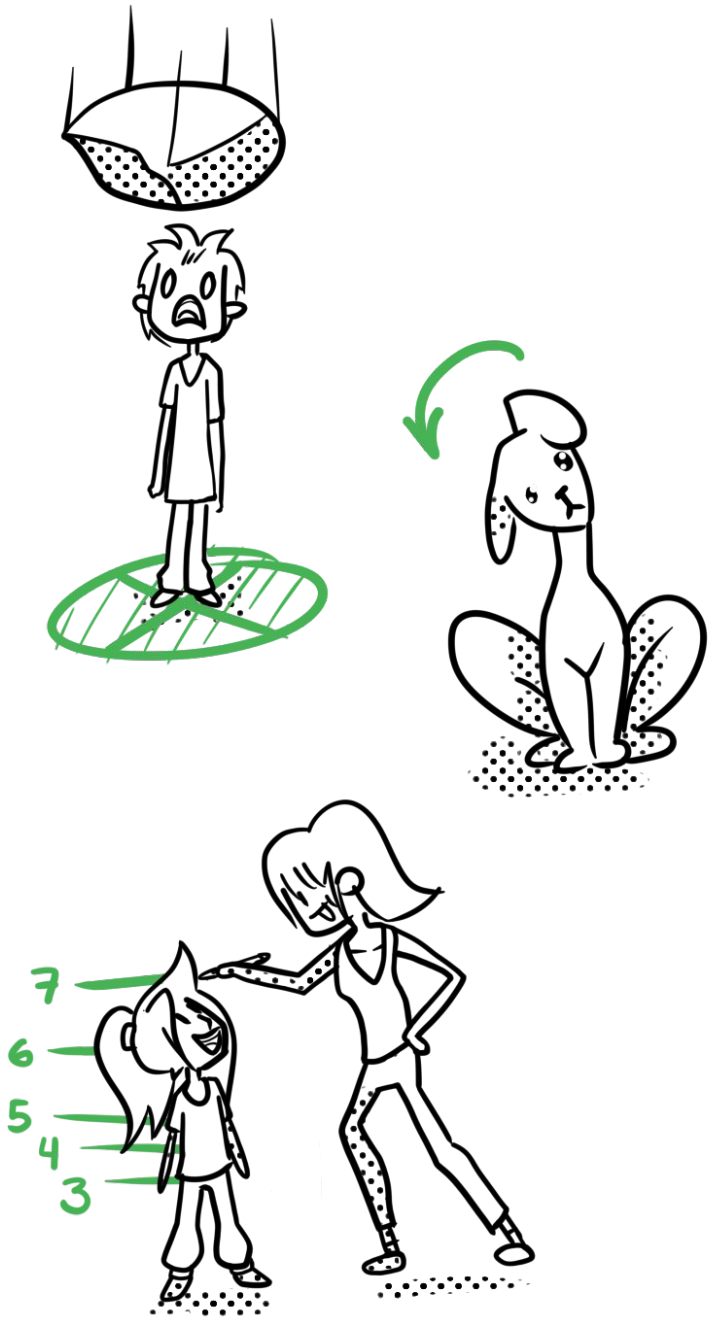
- Variables **públicas**
- Variables **privadas y protegidas**
- Clases **serializables**



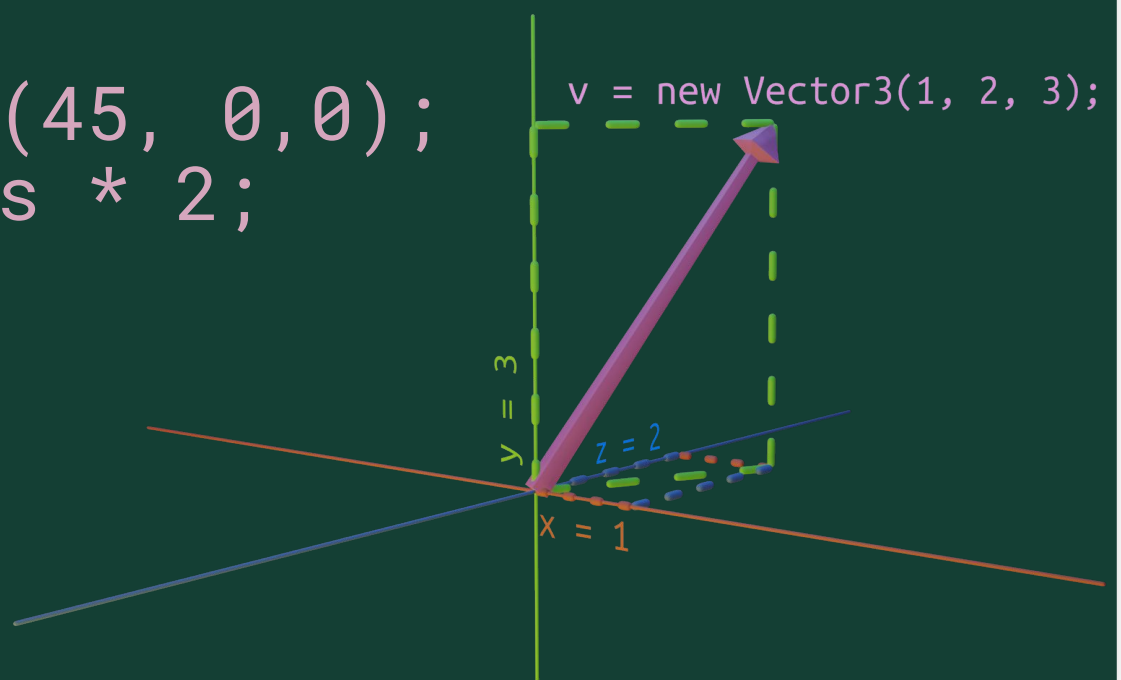
El componente transform y los vectores

Transform

- Posición
- Rotación
- Tamaño

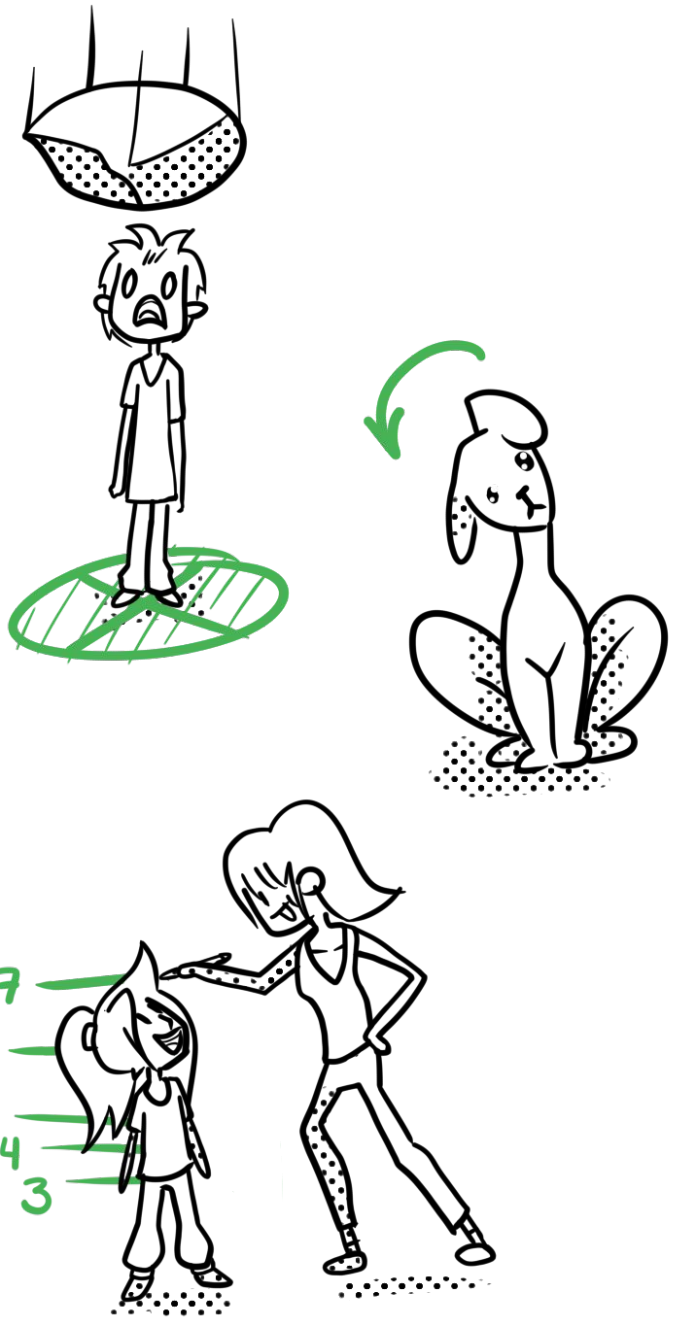


```
// ¿Recuerdas los vectores?
// Vector3 es un Struct

Vector3 v = new Vector3(1, 2, 3);
float x = v.x, y = v.y, z = v.z;
Quaternion q = new Quaternion(1,5,6,2);
q = Quaternion.Euler(45, 90, 10);

q += new Vector3(45, 0,0);
v = q.eulerAngles * 2;
```



Transform

- Posición **(Vector3)**
- Rotación **(Quaternion)**
- Tamaño **(Vector3)**



¿Recuerdas los vectores?

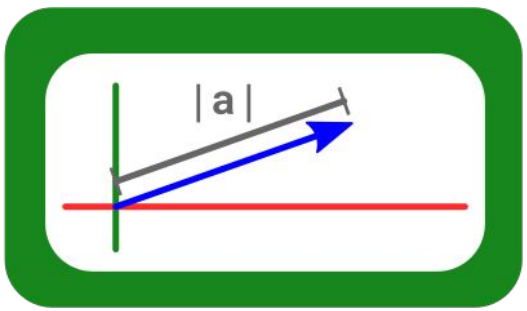
VECTOR != ESCALAR

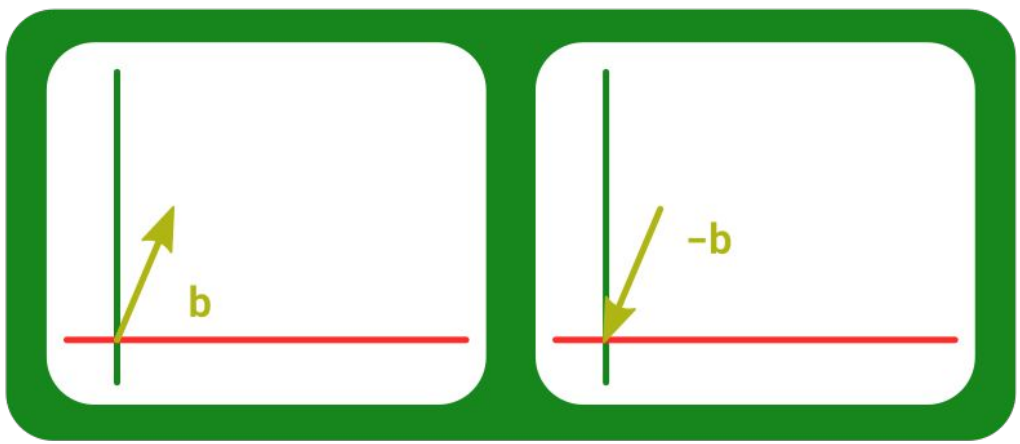
Vector $\in \mathbb{R}^n$ $v=(1,2,3) \in \mathbb{R}^3$

$w=(1,2) \in \mathbb{R}^2$

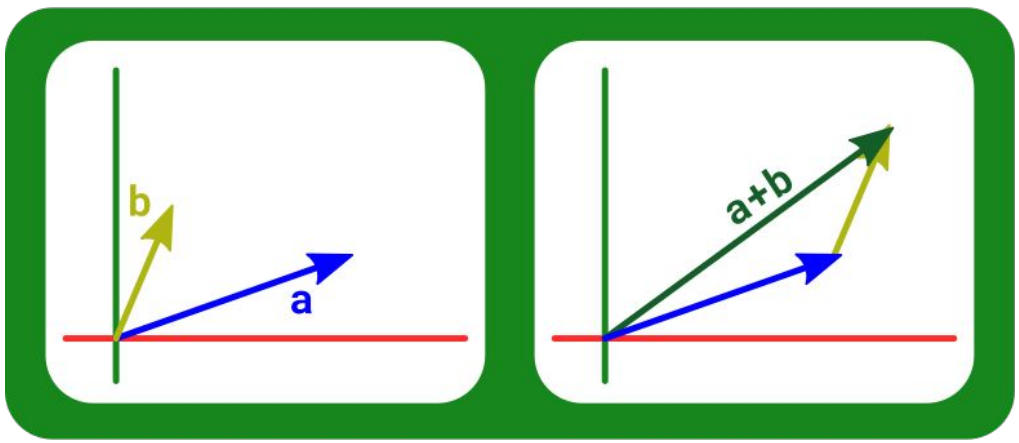
escalar $\in \mathbb{R}$ $x=7.1 \in \mathbb{R}$

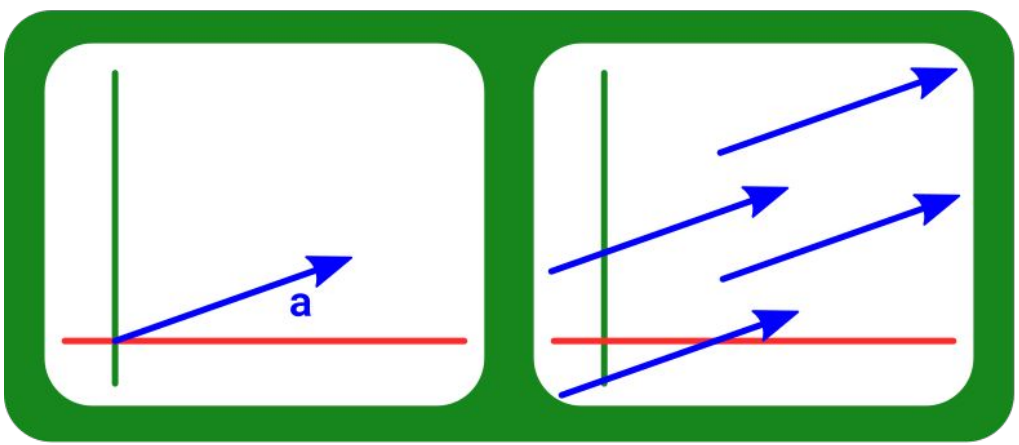
¿Recuerdas los vectores?



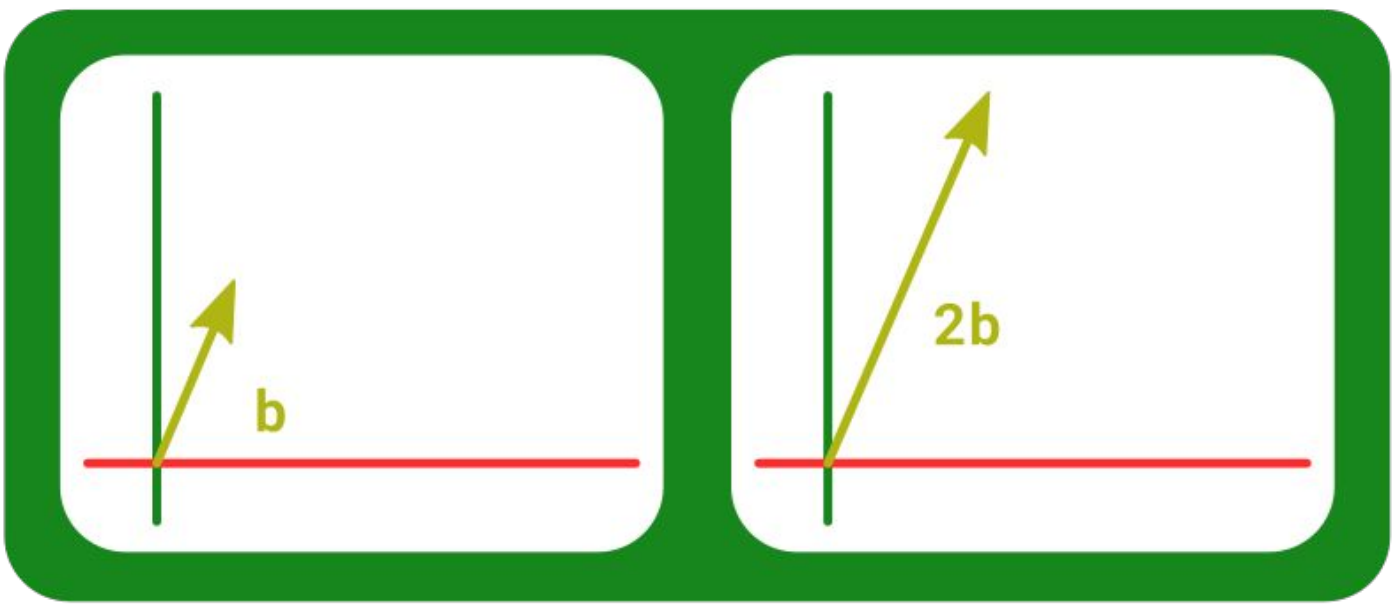


¿Recuerdas los vectores?

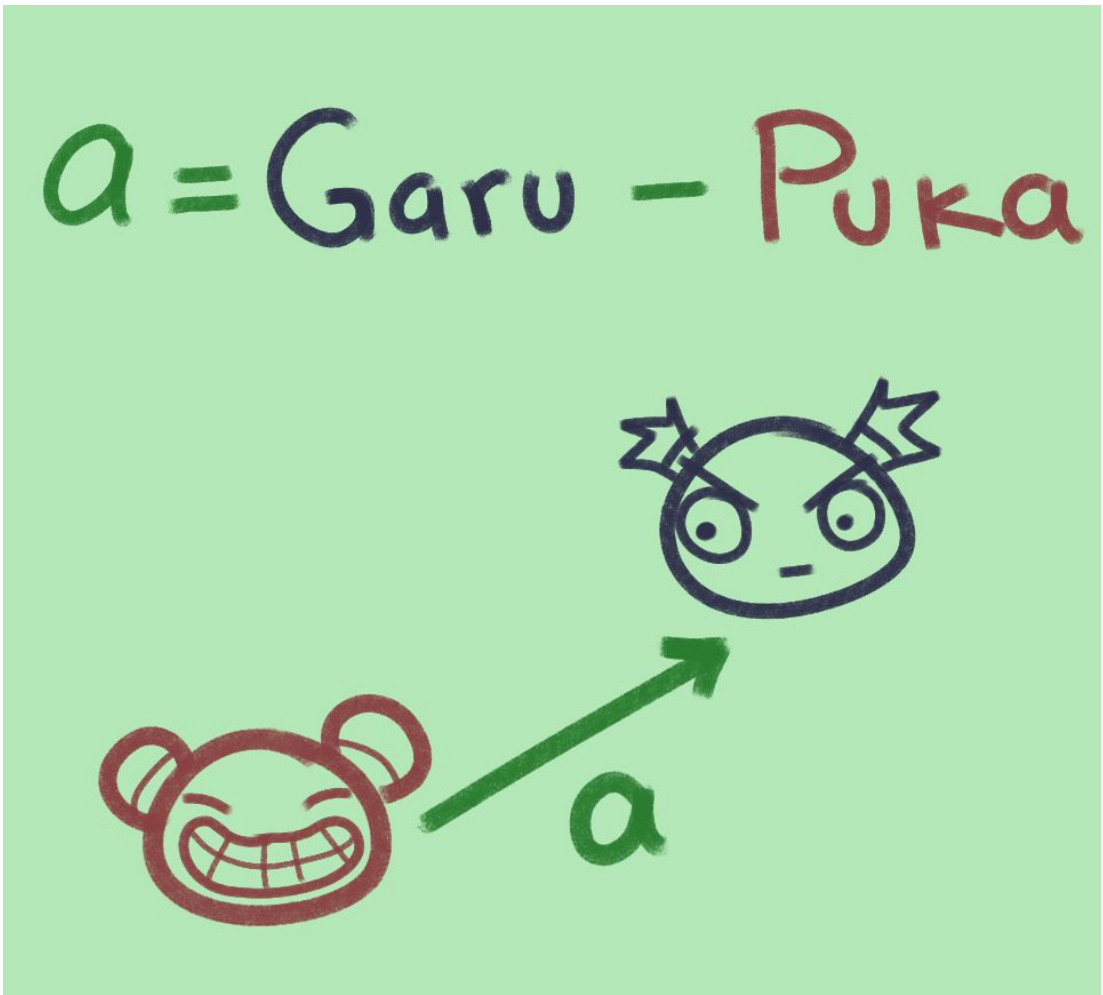




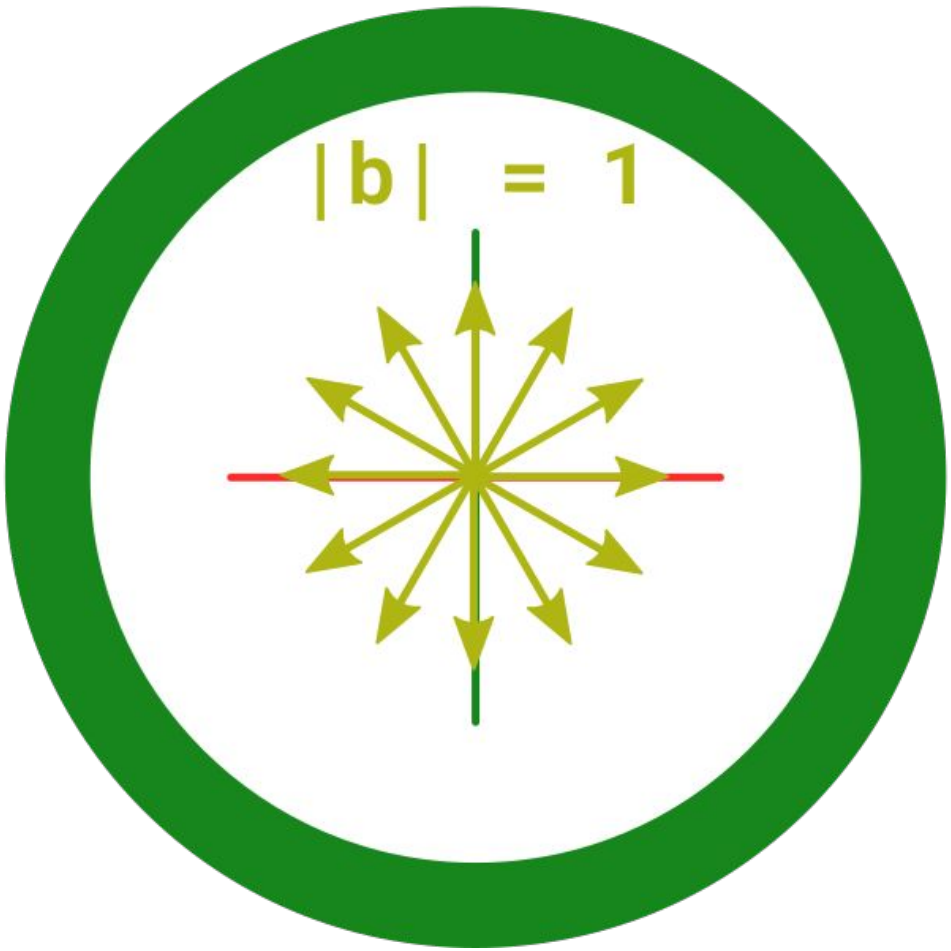
¿Recuerdas los vectores?



¿Recuerdas los vectores?

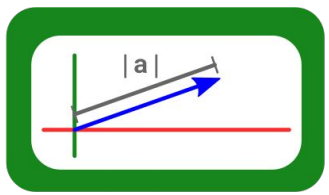


¿Recuerdas los vectores?

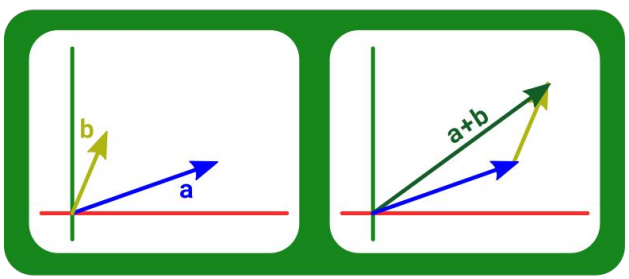


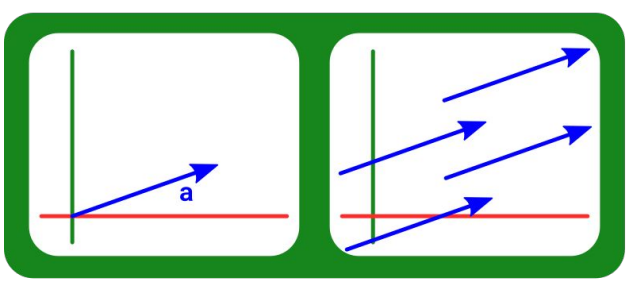
Ya recordamos los vectores

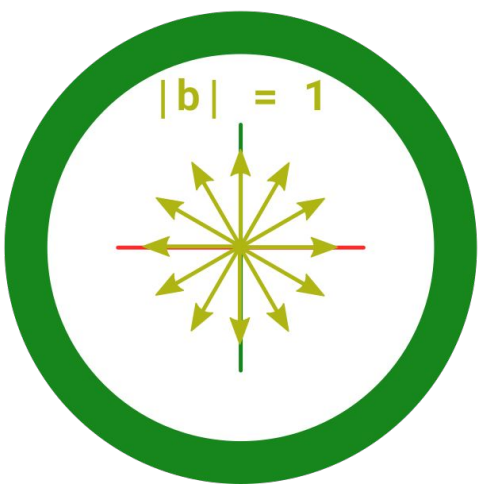
VECTOR != ESCALAR

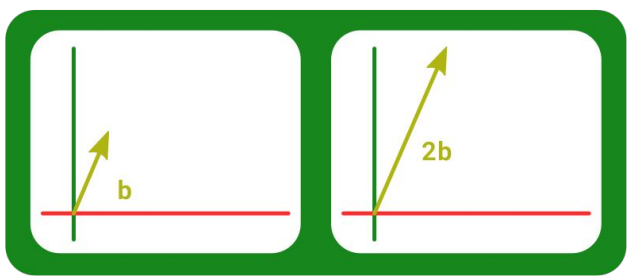


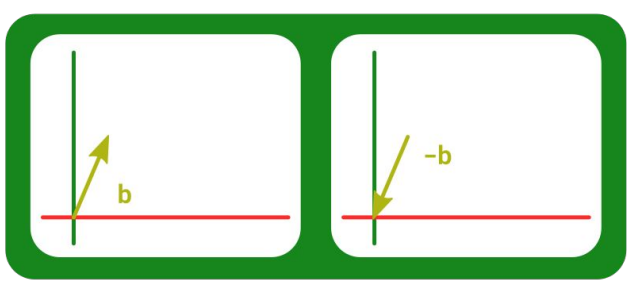
Vector $\in \mathbb{R}^n$ $v=(1,2,3) \in \mathbb{R}^3$
$w=(1,2) \in \mathbb{R}^2$
escalar $\in \mathbb{R}$ $x=7.1 \in \mathbb{R}$

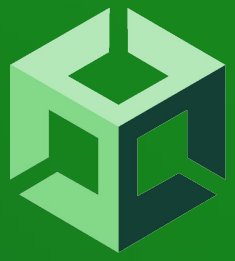






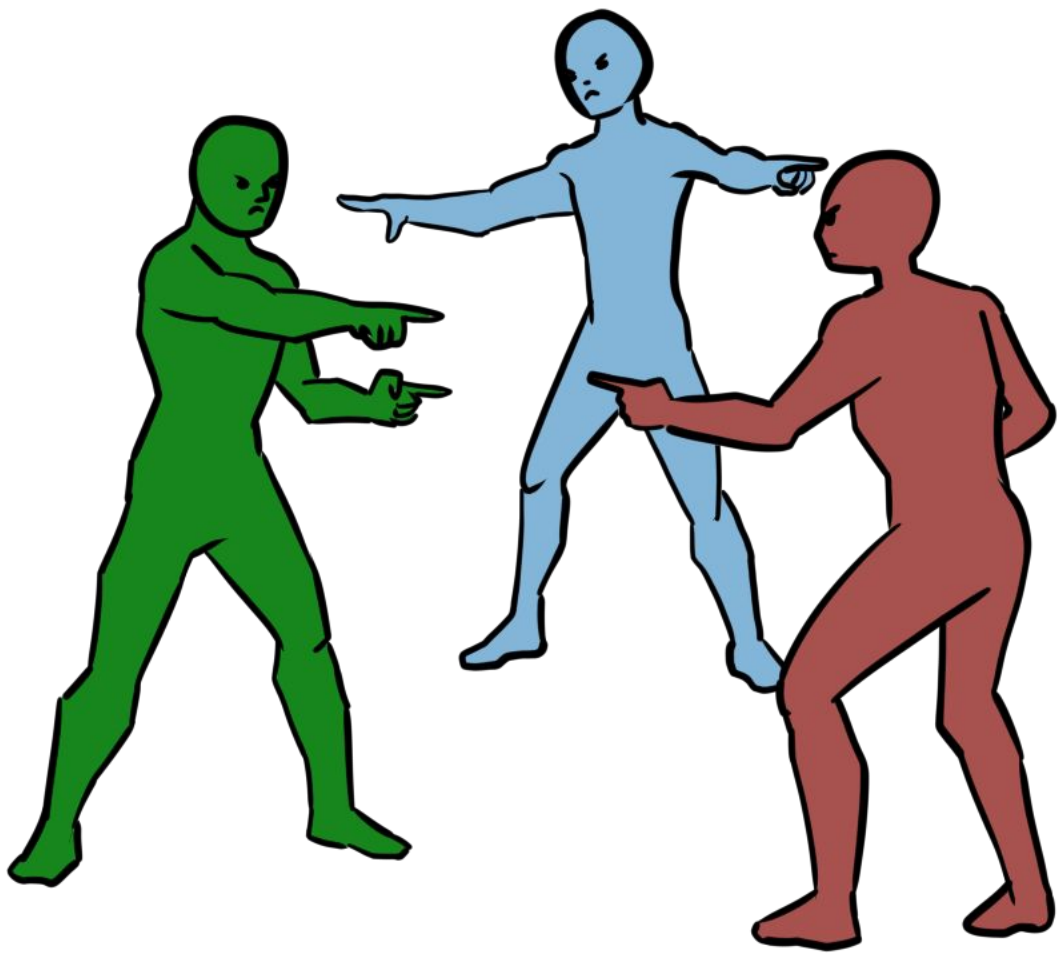






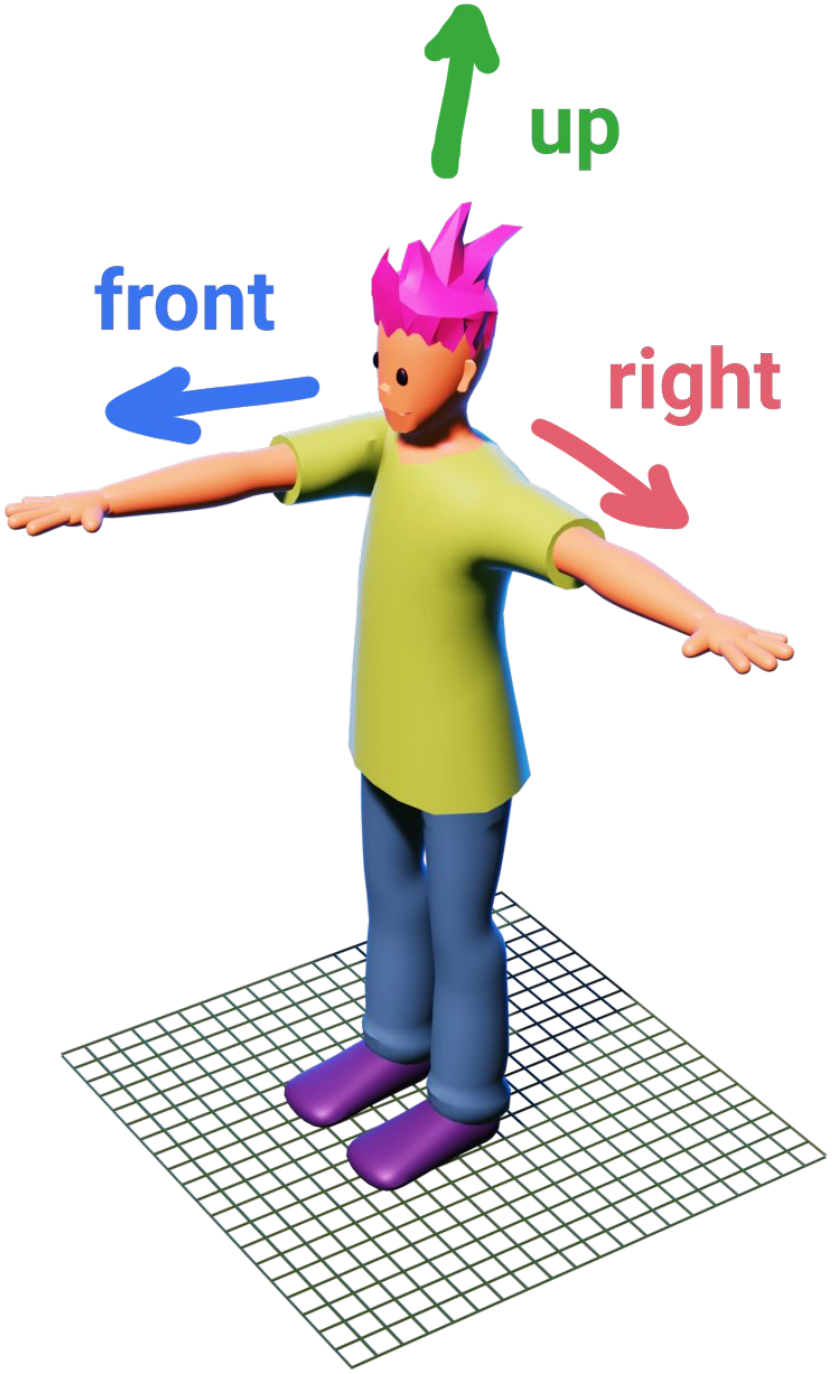
Modificando al componente transform

Local o Global



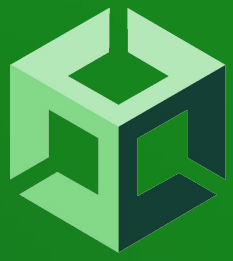
Vectores direccionales

Son vectores **unitarios**



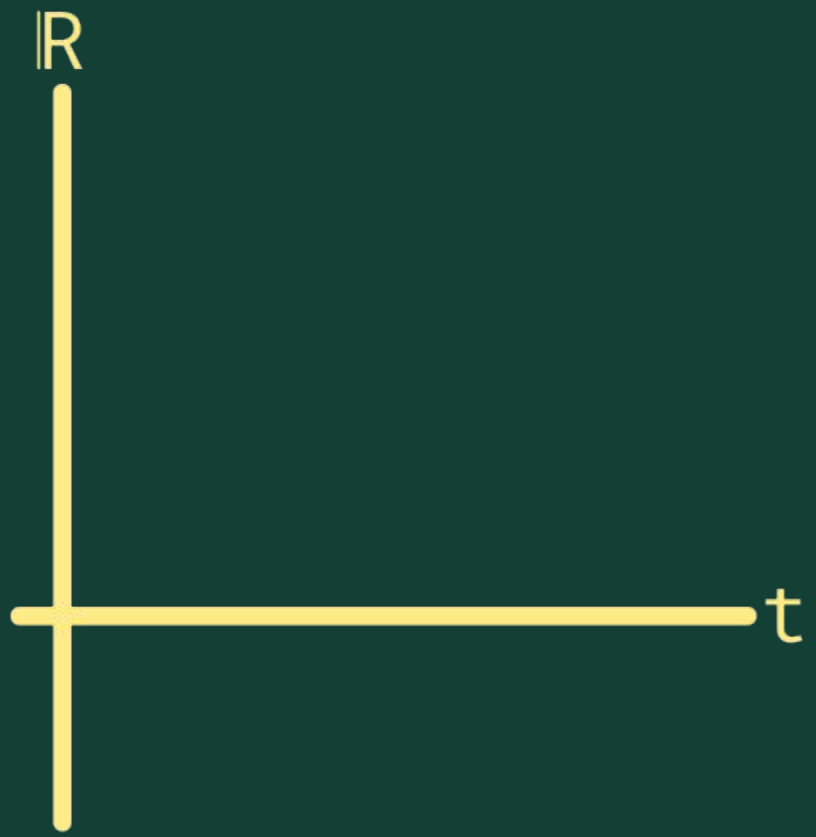
¡Ahora puedes manipular el transform desde script!

- **Posición** *(**Vector3**)*
  - Position y localPosition
  - Translate
- **Rotación** *(**Quaternion**)*
  - Rotation y localRotation, pero con EulerAngles
  - Rotate
- **Tamaño** *(**Vector3**)*
  - LocalScale
  - ¿Escala global? ¡Complicado!
- **Vectores direccionales** *(**Vector3** unitario)*
  - Read & Write

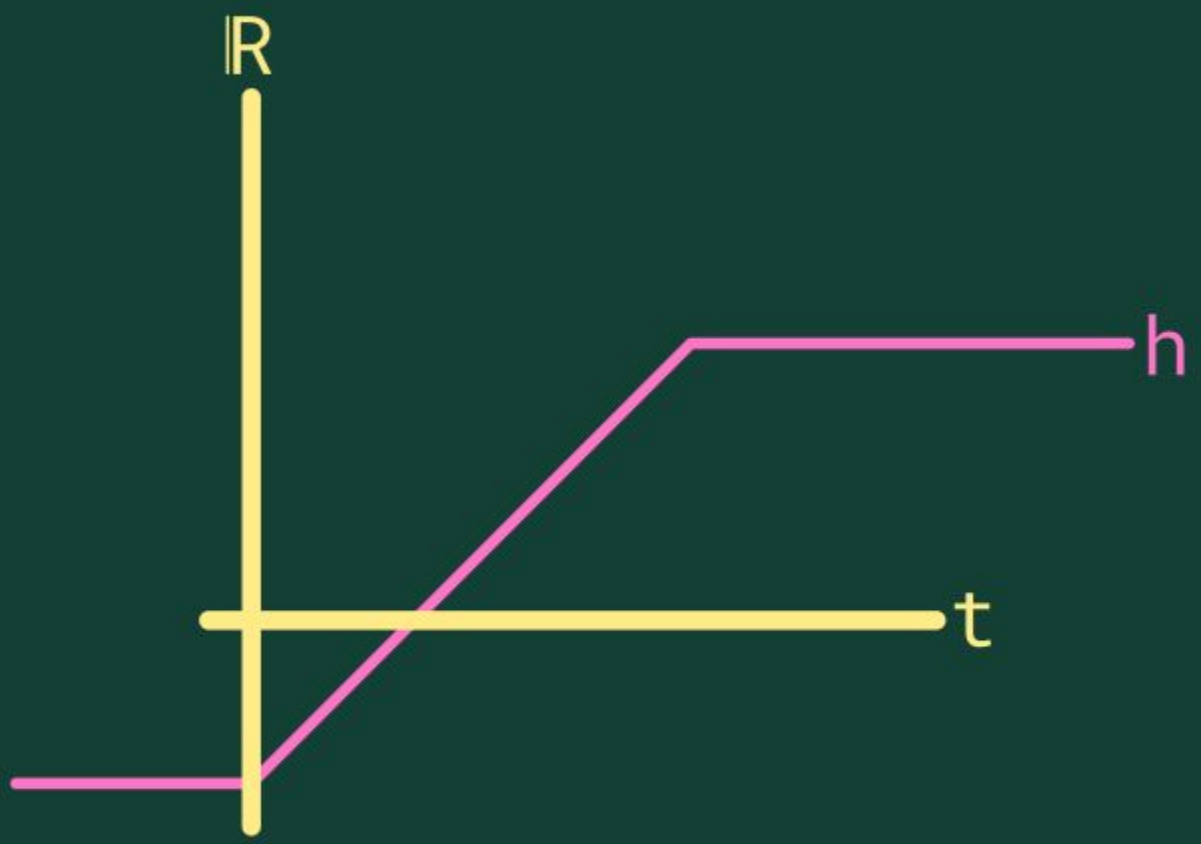


Interpolaciones

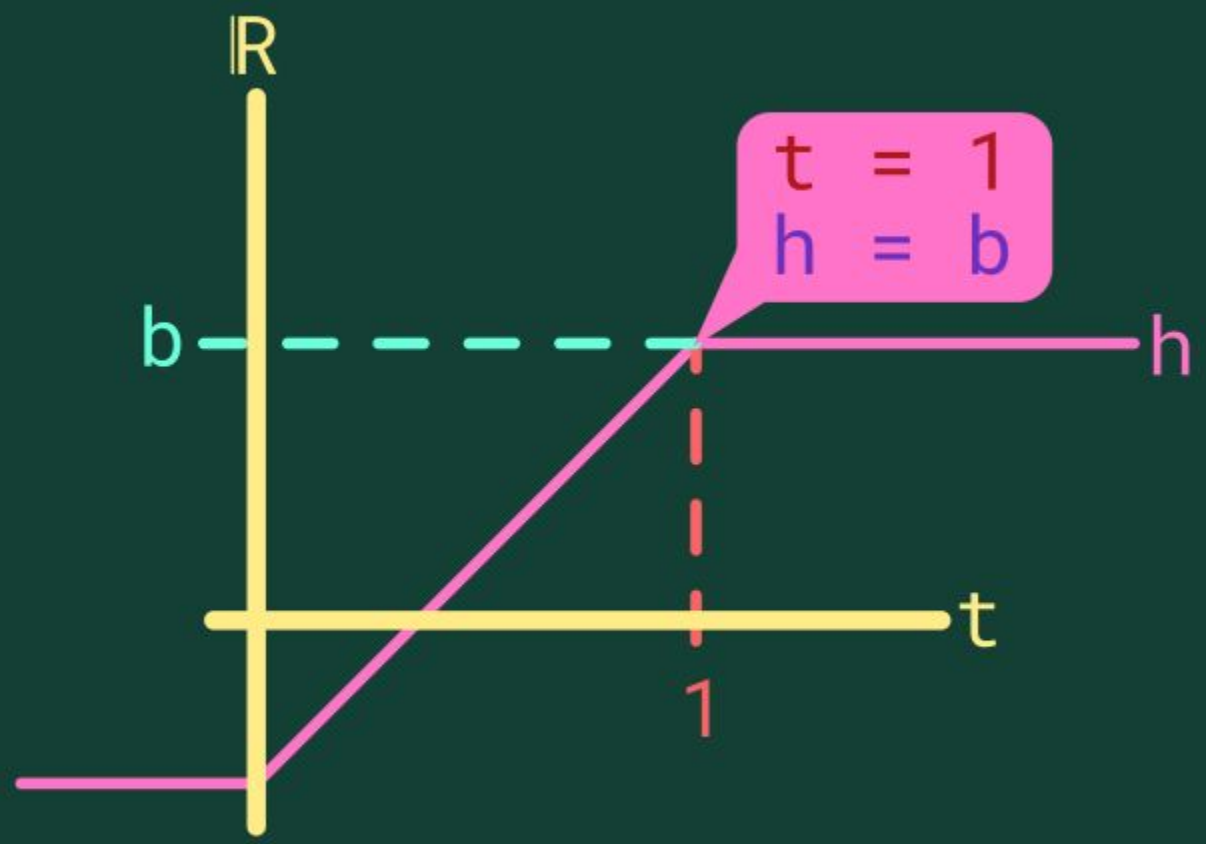
```
float h = Mathf.Lerp(a, b, t)
```



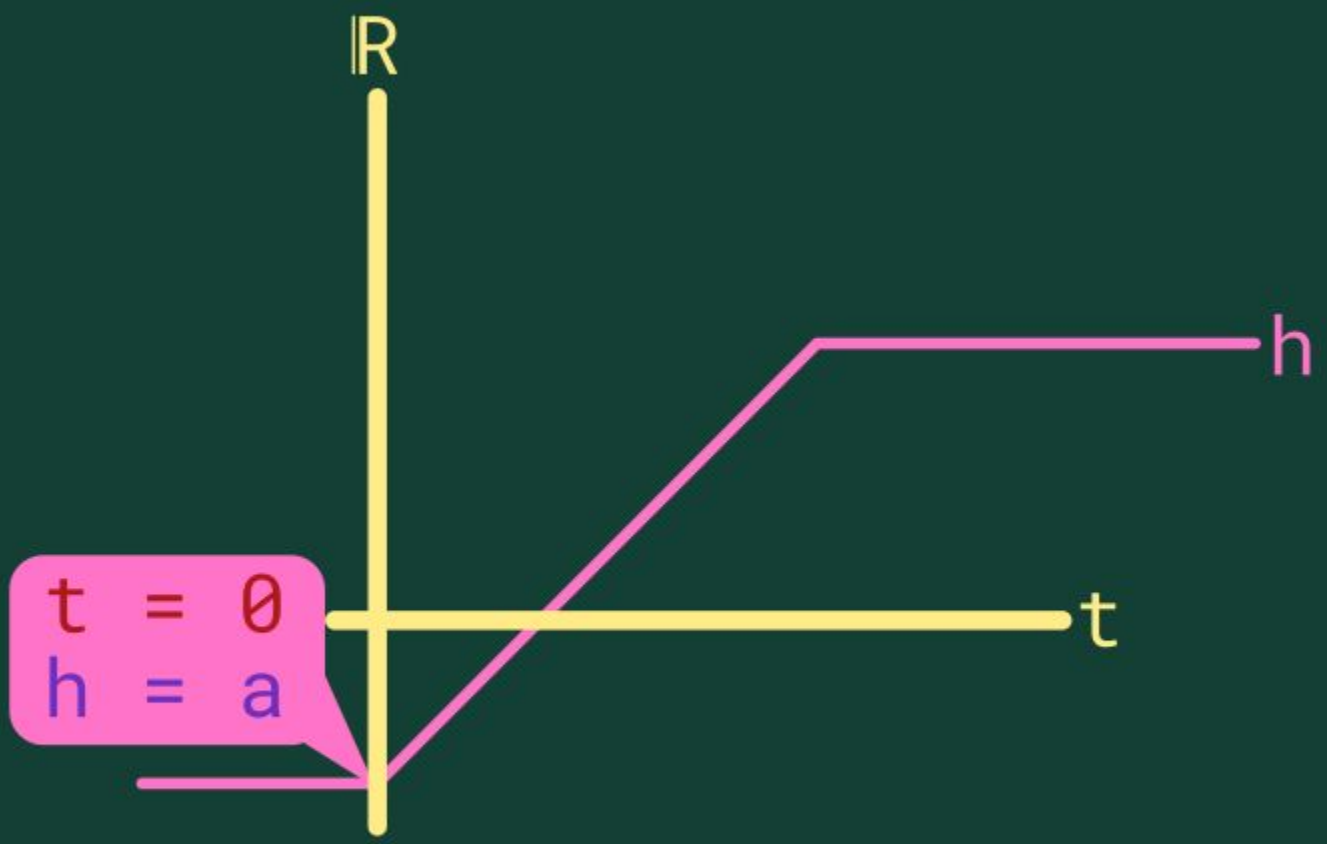
```
float h = Mathf.Lerp(a, b, t)
```



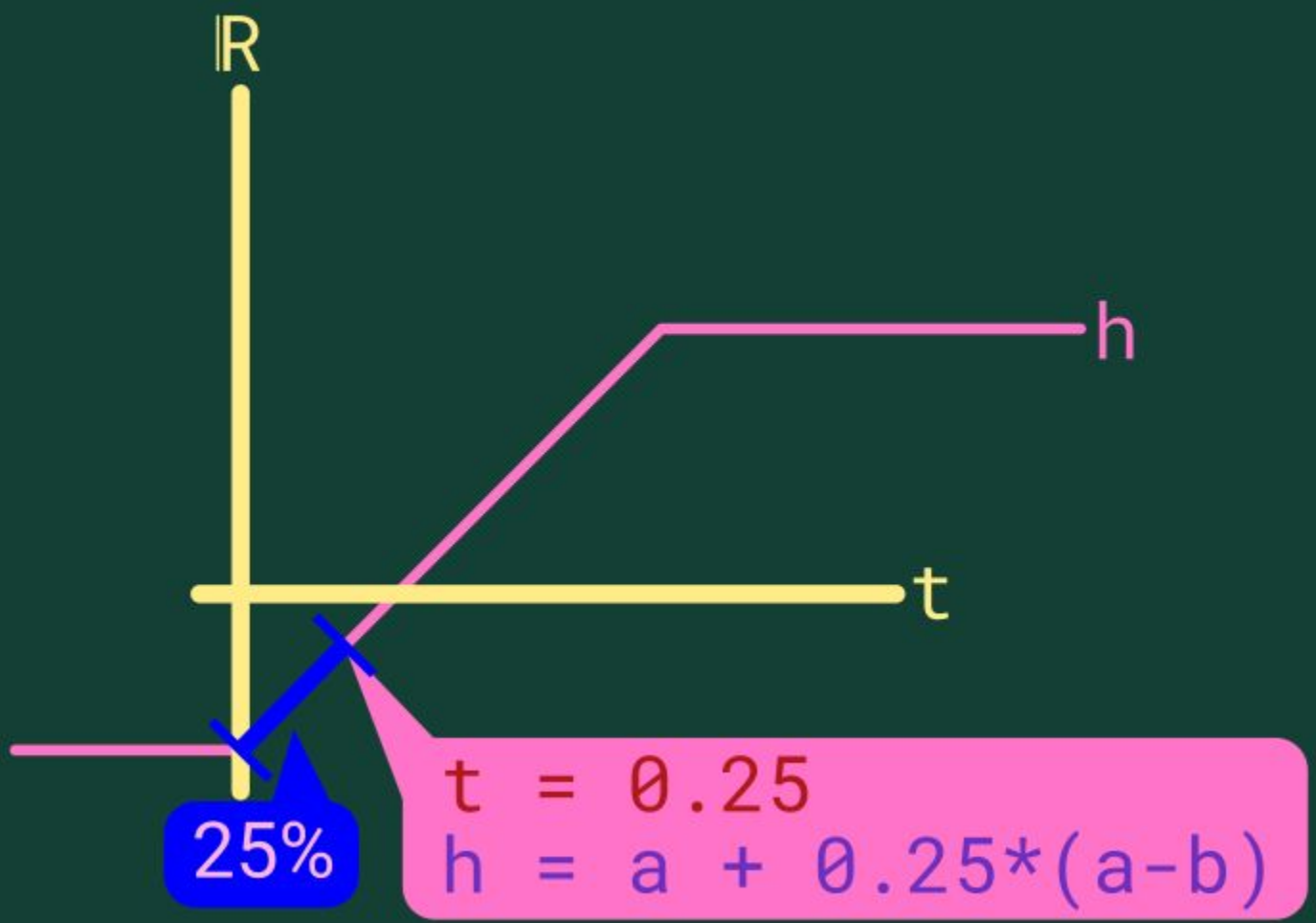
```
float h = Mathf.Lerp(a, b, t)
```



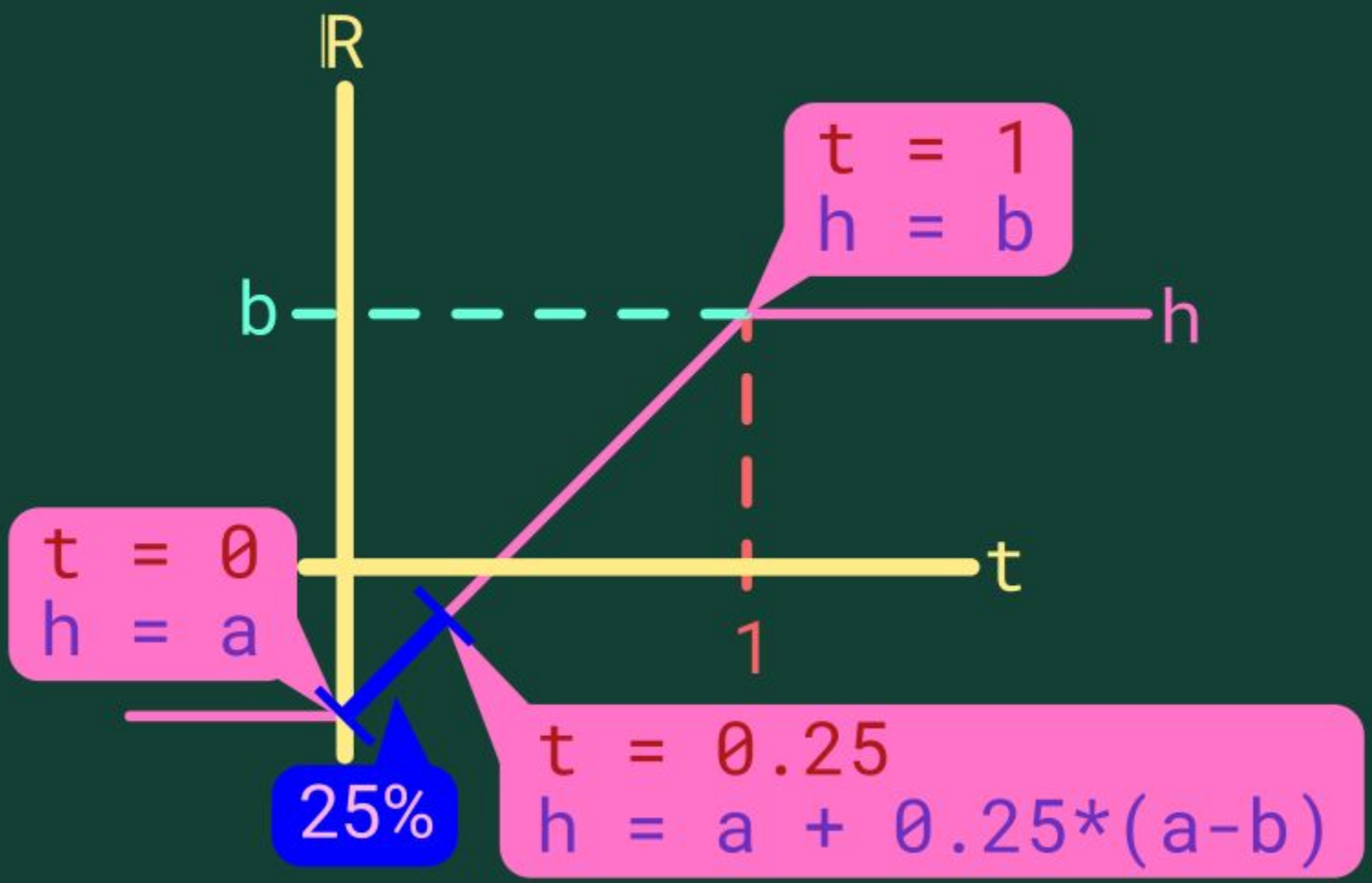
```
float h = Mathf.Lerp(a, b, t)
```



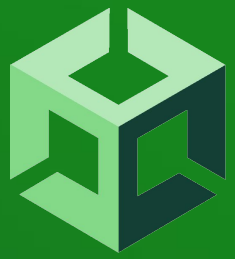
```
float h = Mathf.Lerp(a, b, t)
```



```
float h = Mathf.Lerp(a, b, t)
```



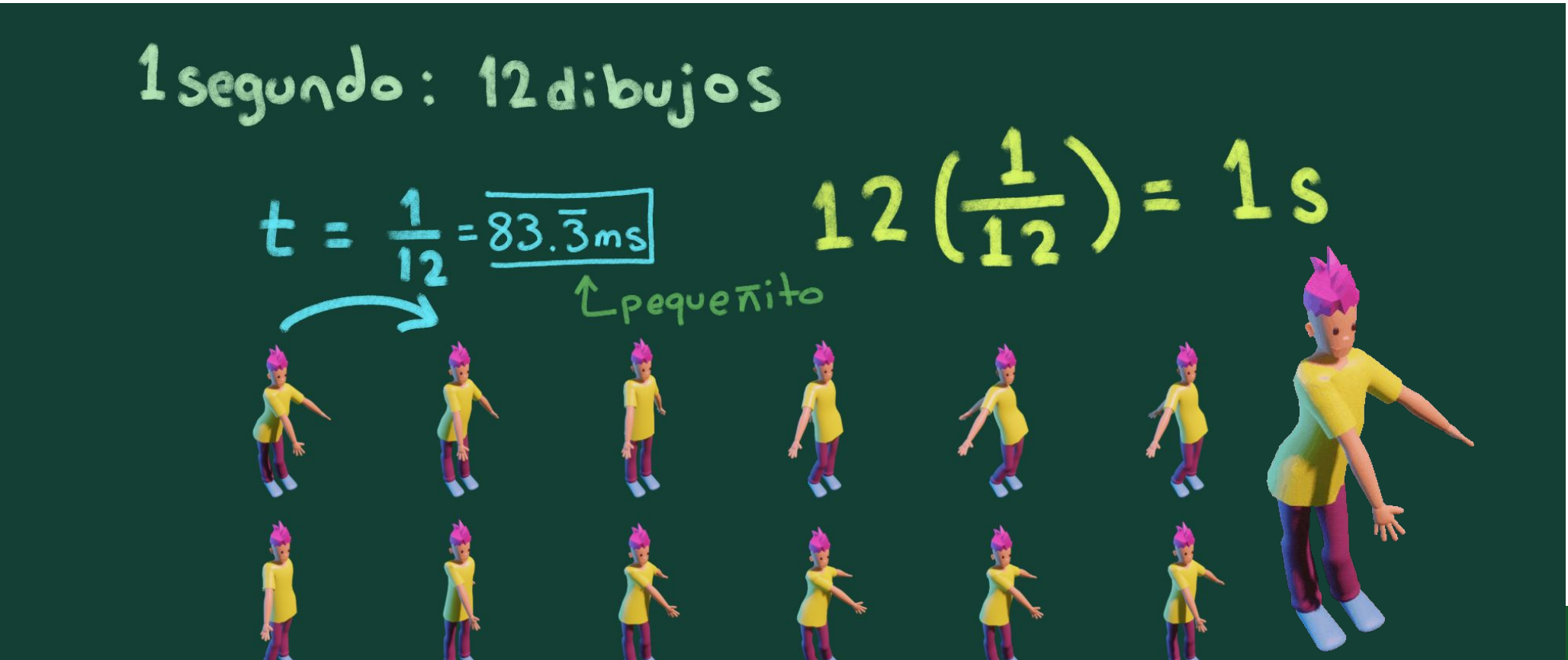
¡Interpolemos variables!



Tiempo en Unity

¿Cómo cuento el tiempo?
(en una animación)

- **12 fps:**
  - En 1 segundo se pintan 12 frames
  - ¿Cuántos segundos pasan entre un frame y el siguiente?



¿Será lo mismo en un juego?

¿A cuántos FPS correría CyberPunk 2077
en cada uno de estos setups?

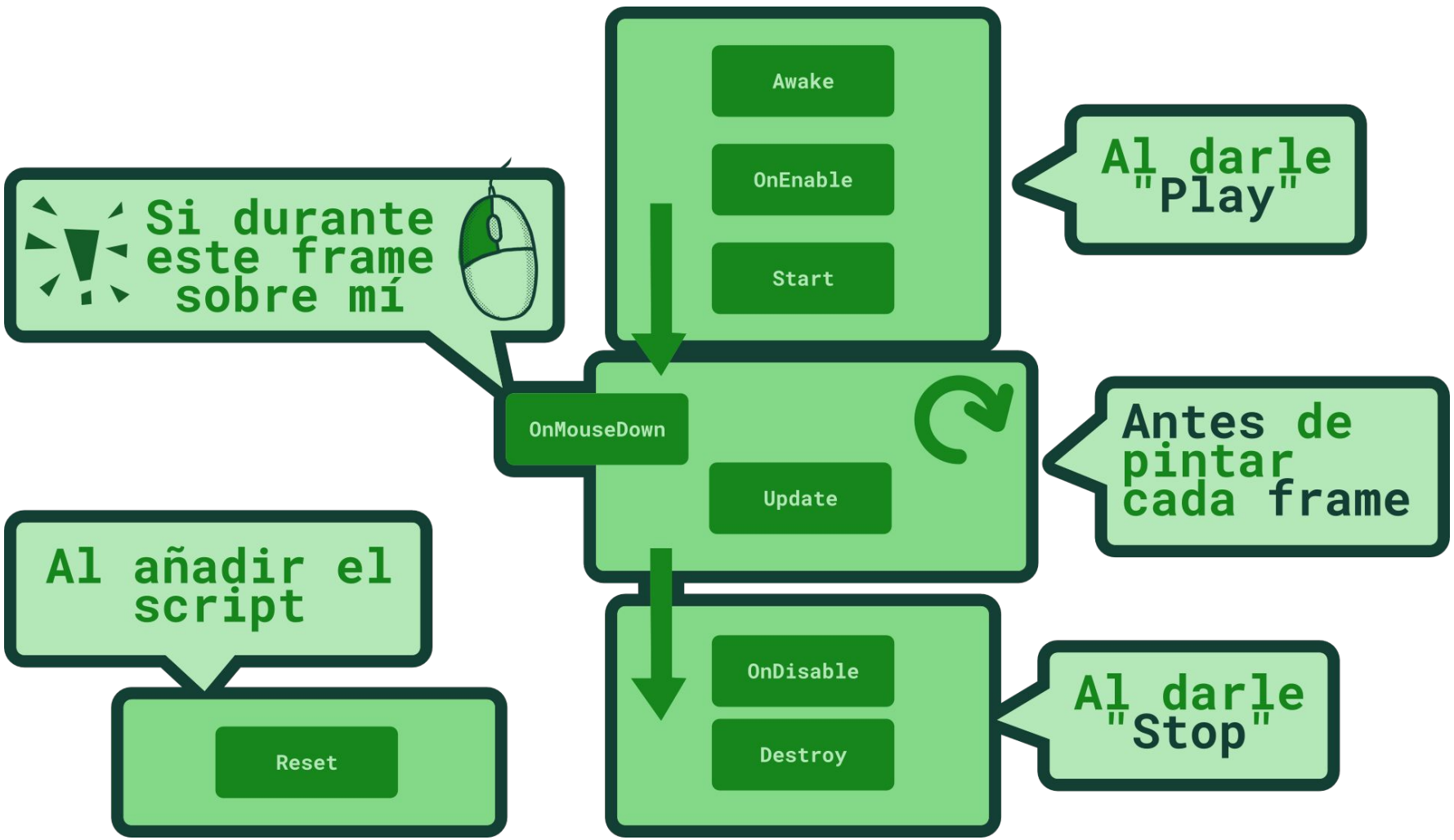
¿Será lo mismo en un juego?

¿En dónde hay menos lag?

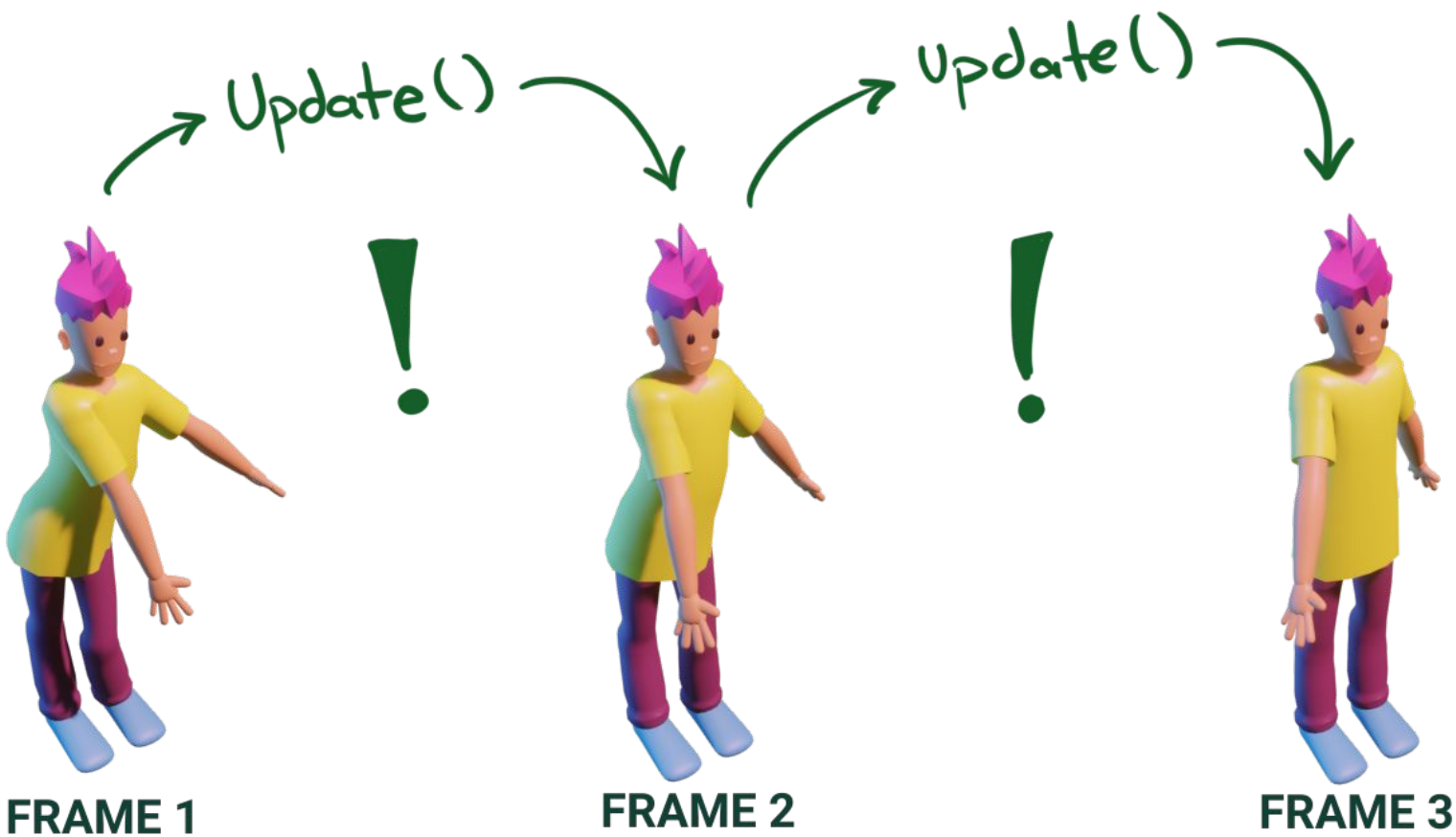
¿Será lo mismo en un juego? ¡NO!

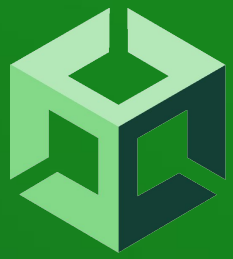
- El Tiempo entre frame y frame:
  - Es diferente en cada frame
  - Depende de la dificultad de renderizado
  - Depende de la máquina

¿Recuerdas el ciclo de vida de un Script?



Update: Entre frame y frame

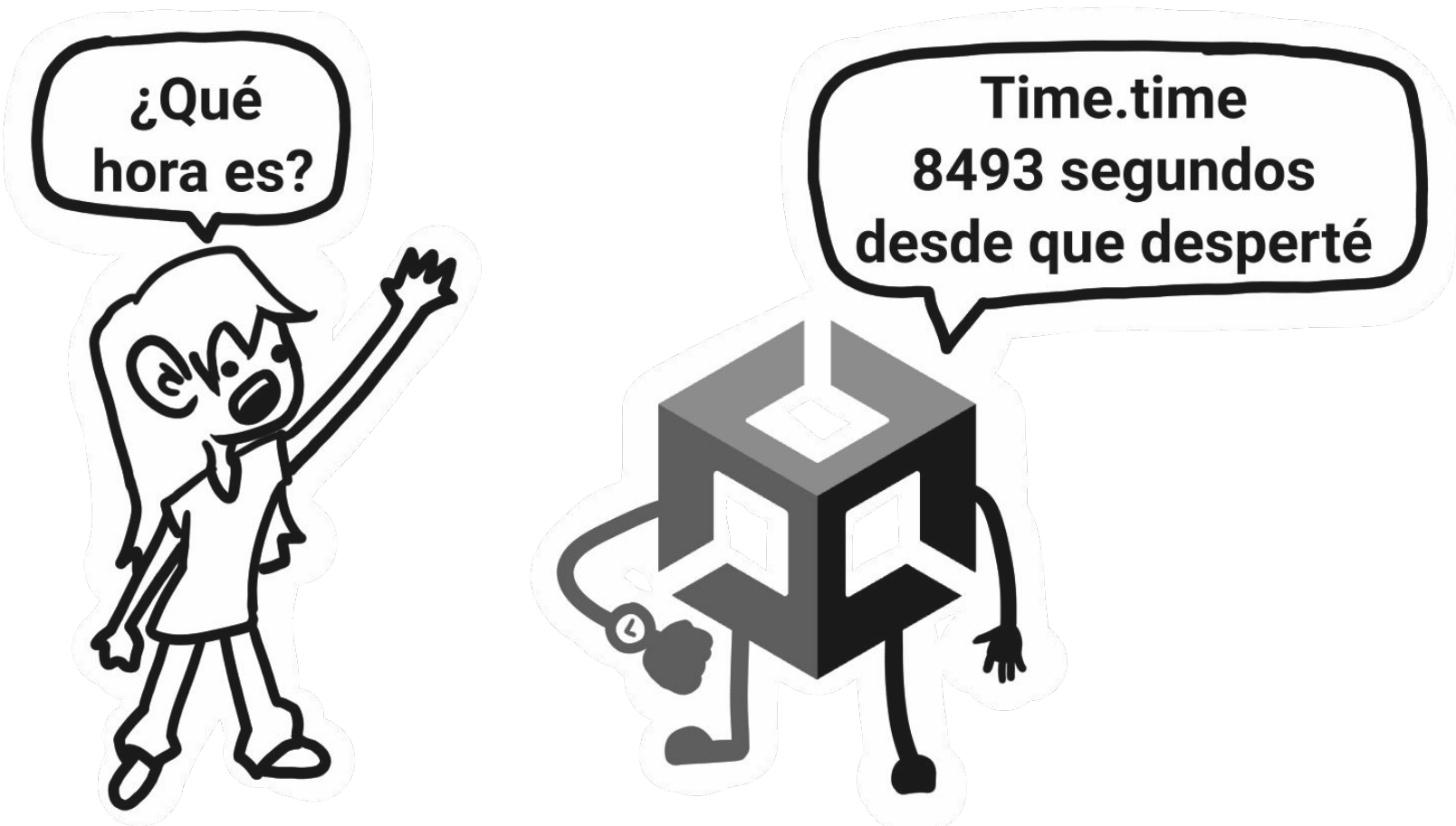




Contando y *controlando* el tiempo

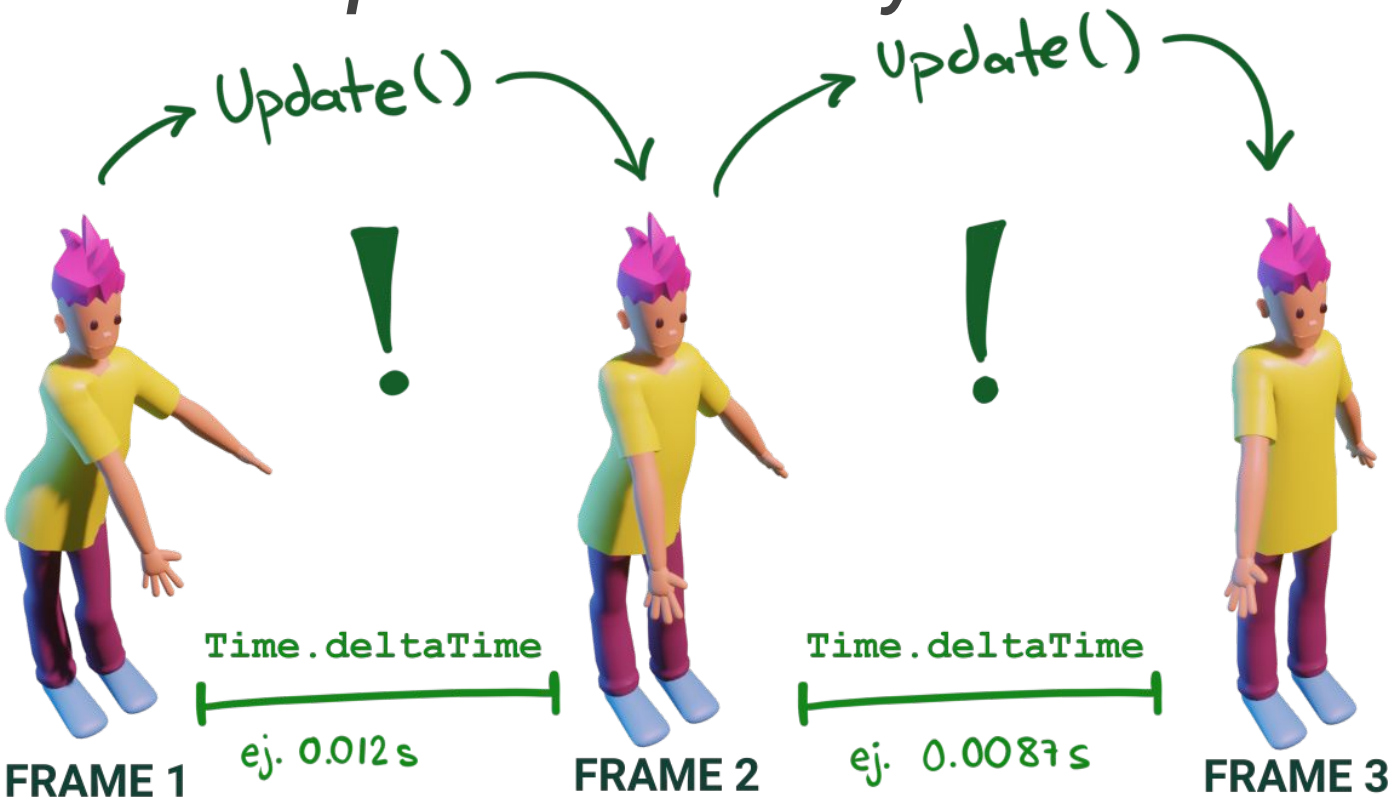
Tiempo total, tiempo delta y escala del tiempo

Forma 1 (difícil):
`Time.time`



Forma 2 (fácil): Time.deltaTime

Tiempo entre frame y frame



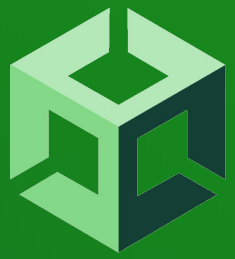
QUEST

¿Qué usarías para resolver los siguientes problemas?

¿Time.time ó Time.deltaTime?

- Simular un semáforo
- Verificar si un ataque está en cooldown
- El jugador puede ir al casino y girar la ruleta para obtener un premio y puede volver a hacerlo luego de 1 hora

No hay respuestas incorrectas :)



Separando instrucciones mediante tiempo

¡Las poderosísimas corrutinas!

Forma 3: (ejecución fácil, aprendizaje difícil)

Corrutinas

- Es un **método**
- Pausas controladas
  - Espera un tiempo determinado
  - Espera un frame
  - Espera hasta que esto sea true
  - Espera hasta que otra corrutina se detenga

Estructura de una corrutina

```
using UnityEngine;
using System.Collections;

public class ProbadorDeCorrutinas : MonoBehaviour {
  void Start () {
    this.StartCoroutine(MiCorrutina("me gustan las galletas"));
  }

  IEnumerator MiCorrutina (string mensaje) {

    Debug.Log("quieres escuchar un secreto?");

    yield return new WaitForSeconds(1.5f);

    Debug.Log(mensaje);
  }
```

invocación

ejecución

!!!

Nombre

Parámetros

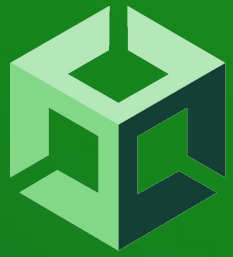
pausa

No olvides:

- Ejecutarlo al invocarlo
- Poner una pausa
- IEnumerator
- MonoBehaviour activo

Tipos de pausas en las corrutinas

- Espera unos segundos:
  yield return new WaitForSeconds(s);
- Espera un frame
  yield return null;
- Espera hasta que **f()** devuelva **true;**
  yield return WaitUntil(f);
- Espera hasta que **otraCorrutina()** termine
  yield return StartCoroutine(otraCorrutina());

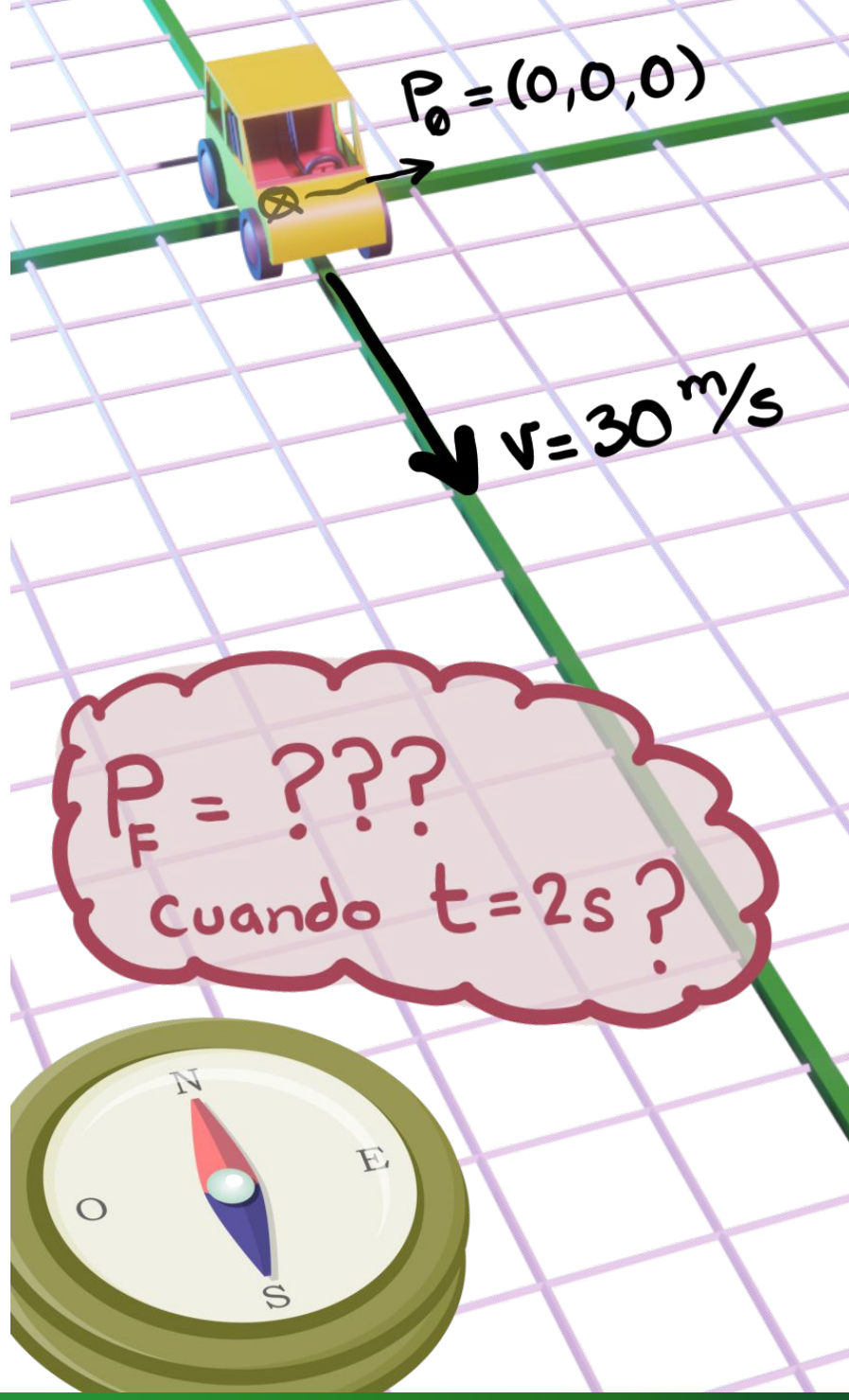


Movimiento y tiempo

¿Recuerdas MRU de física?

Un carro va a una velocidad de 30 m/s (108 km/h) hacia el sur. Si el carro se encuentra en la posición (0, 0, 0):

¿A qué distancia se encontrará luego de 2 segundos?



Solución:

$v = \frac{d}{t} \Rightarrow \boxed{d = vt}$

$t = 2s$

$V = 30\ m/s$

$\Rightarrow d = (30\ m/s)(2s)$

$= 60m$ ... pero dónde??

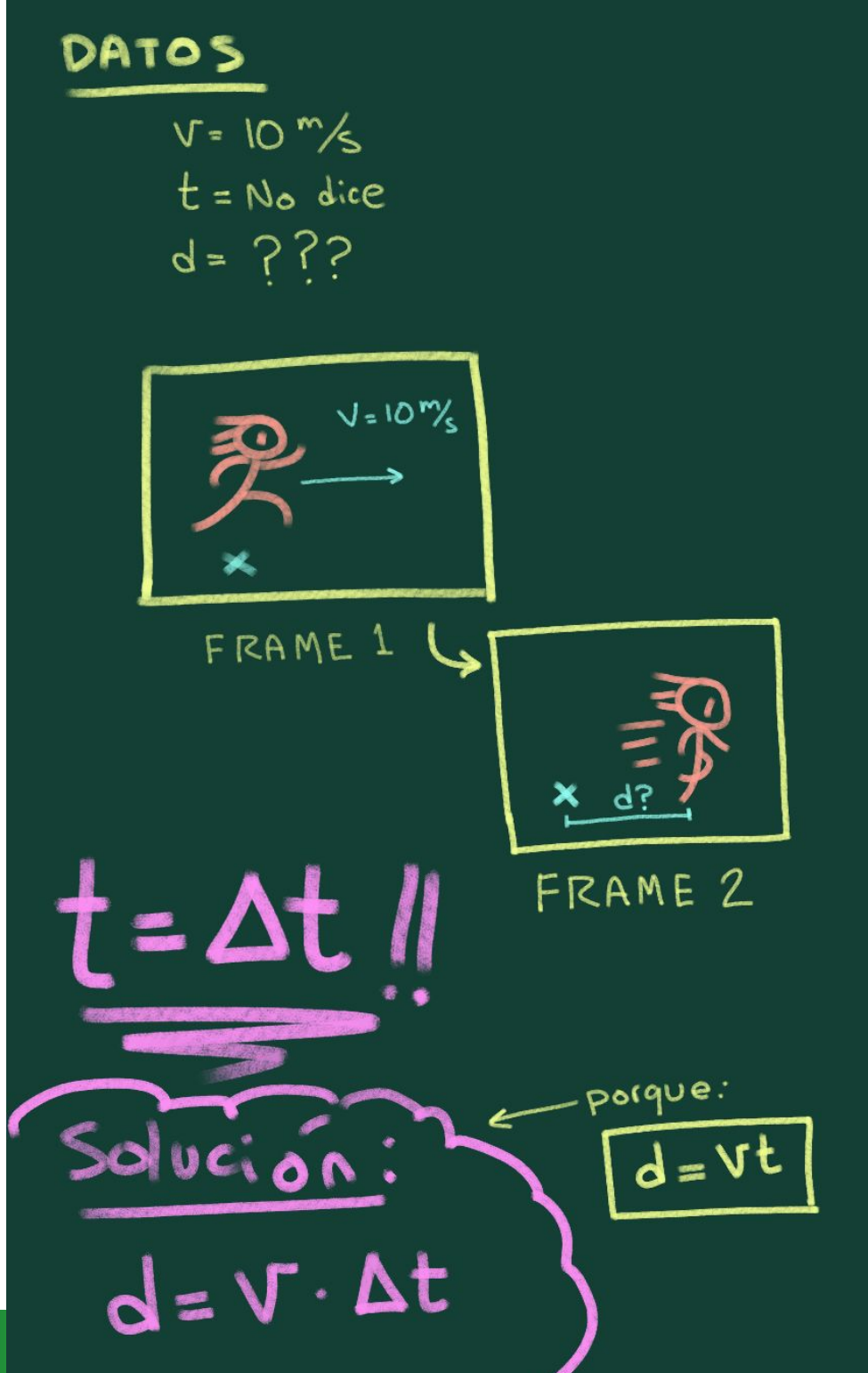
>al sur

Respuesta $\therefore P_F = (0, 0, -60)\ m$

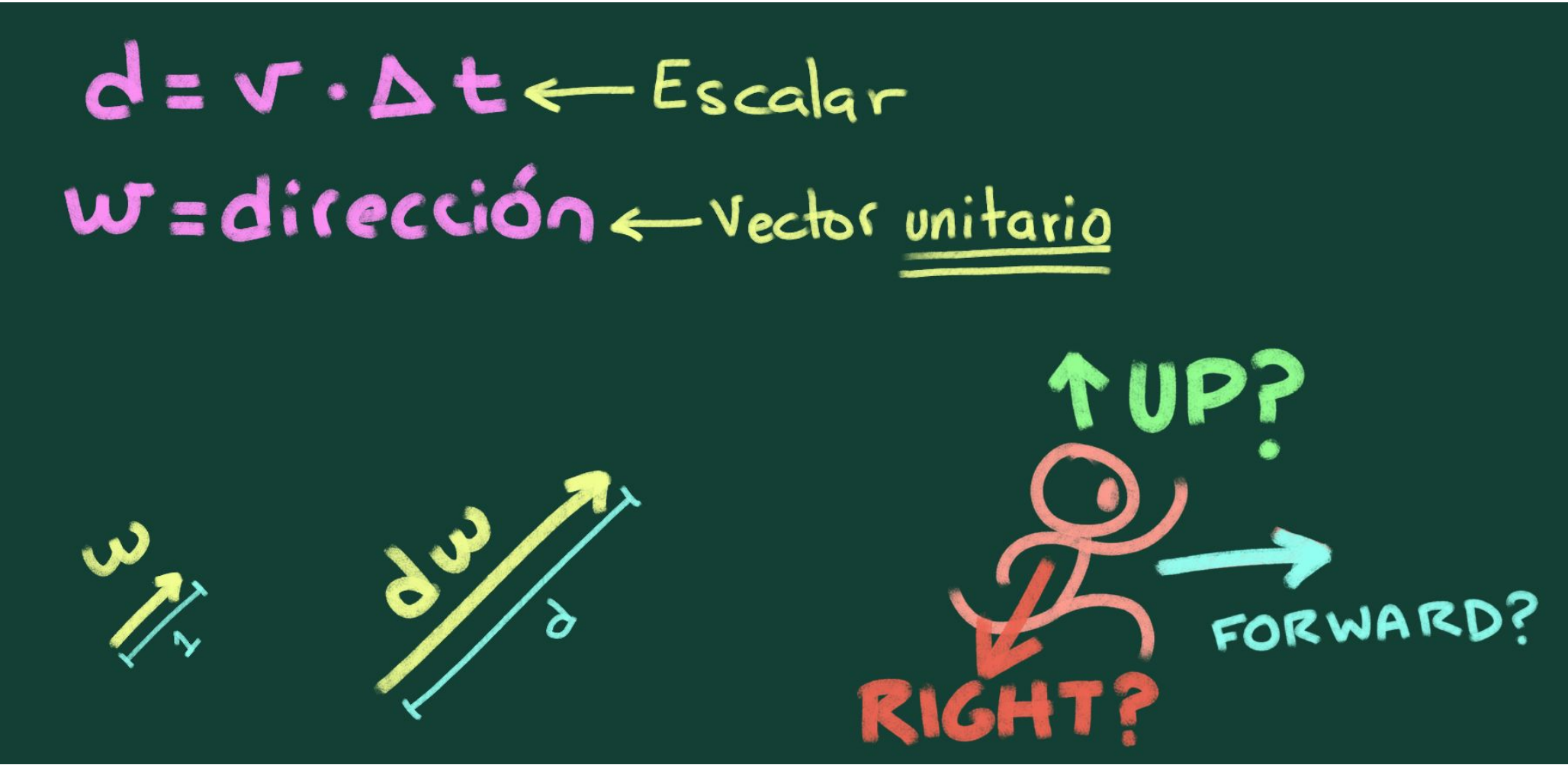
¡Ahora dentro del juego!

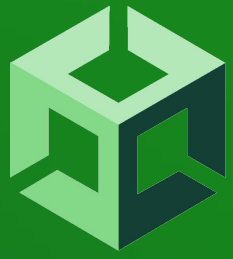
Si mi personaje lleva una velocidad de 10 m/s…

¿Qué distancia debería haber recorrido entre este frame y el anterior?



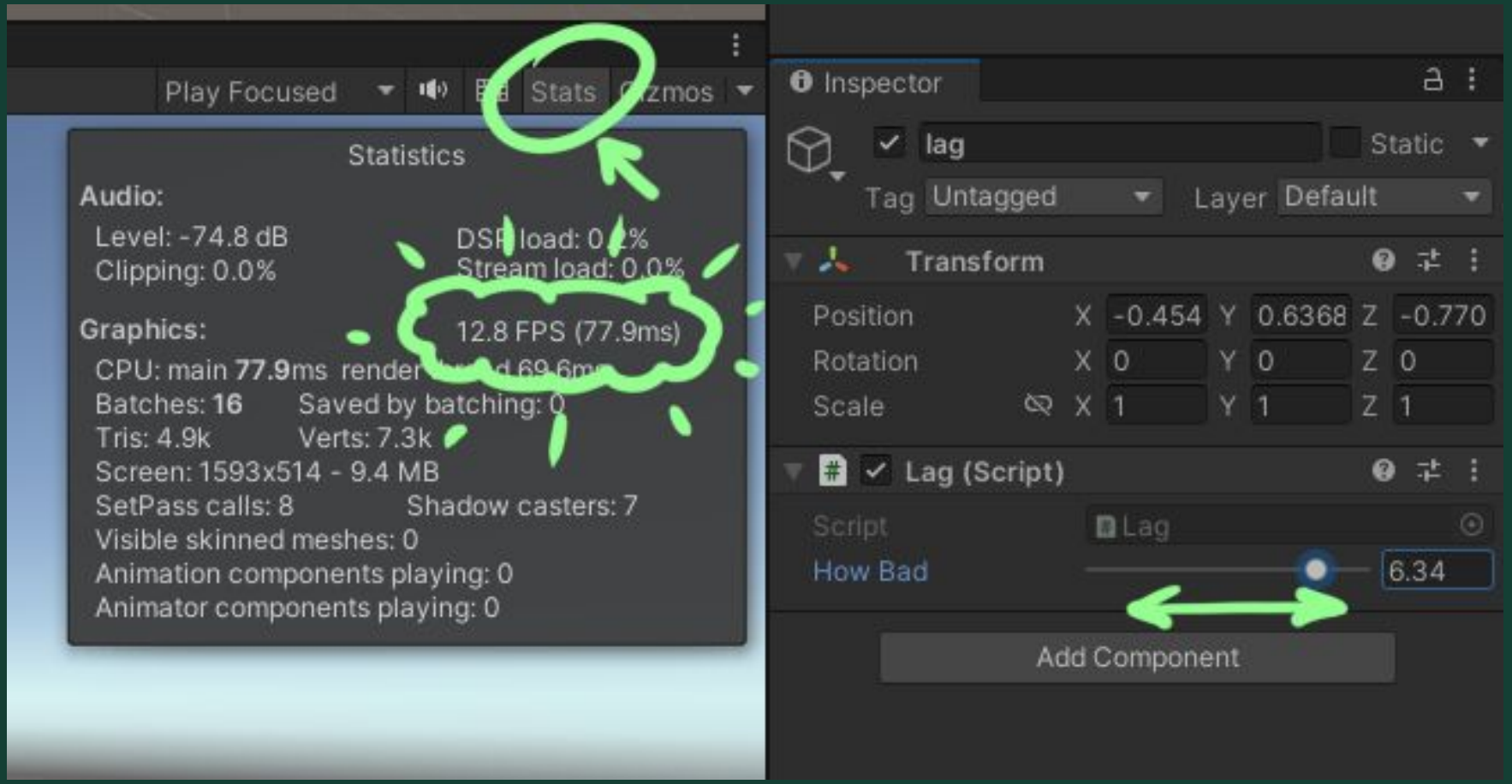
¿Pero en qué dirección?





Creando movimiento independiente del framerate

```
using UnityEngine;

public class Lag : MonoBehaviour {
  [Tooltip("Comienza desde el más bajo y sube "+
           "gradualmente. ¡Este código puede " +
           "congelar tu editor, úsalo con "+
           "cuidado!")]
  [Range(3, 7)]
  public float howBad = 3;

  void Update () {
    for (int i=0; i<Mathf.Pow(10, howBad); i++);
  }
}
```


Play Focused
Stats
Gizmos
Statistics
Audio:
Level: -74.8 dB
Clipping: 0.0%
Graphics:
12.8 FPS (77.9ms)
CPU: main 77.9ms
Batches: 16
Saved by batching: 0
Tris: 4.9k
Verts: 7.3k
Screen: 1593x514 - 9.4 MB
SetPass calls: 8
Shadow casters: 7
Visible skinned meshes: 0
Animation components playing: 0
Animator components playing: 0
Inspector
lag
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Lag (Script)
Script
Lag
How Bad
6.34
Add Component

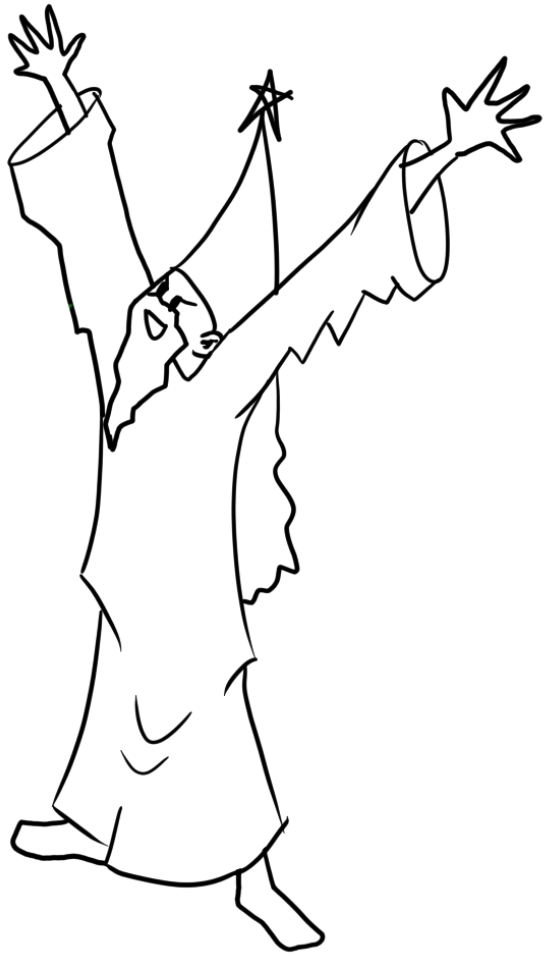
¡Veámoslo en acción!

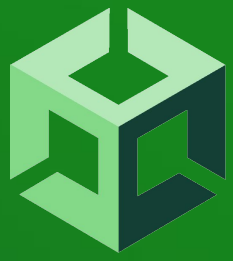
¡Pausa el video!

- Haz que una esfera siga a un cubo usando deltaTime.
- Haz que un cilindro siga al mismo cubo sin usar deltaTime.
- Pruébalo moviendo al cubo en la ventana Scene, ponle LAG a la escena para ver la diferencia.

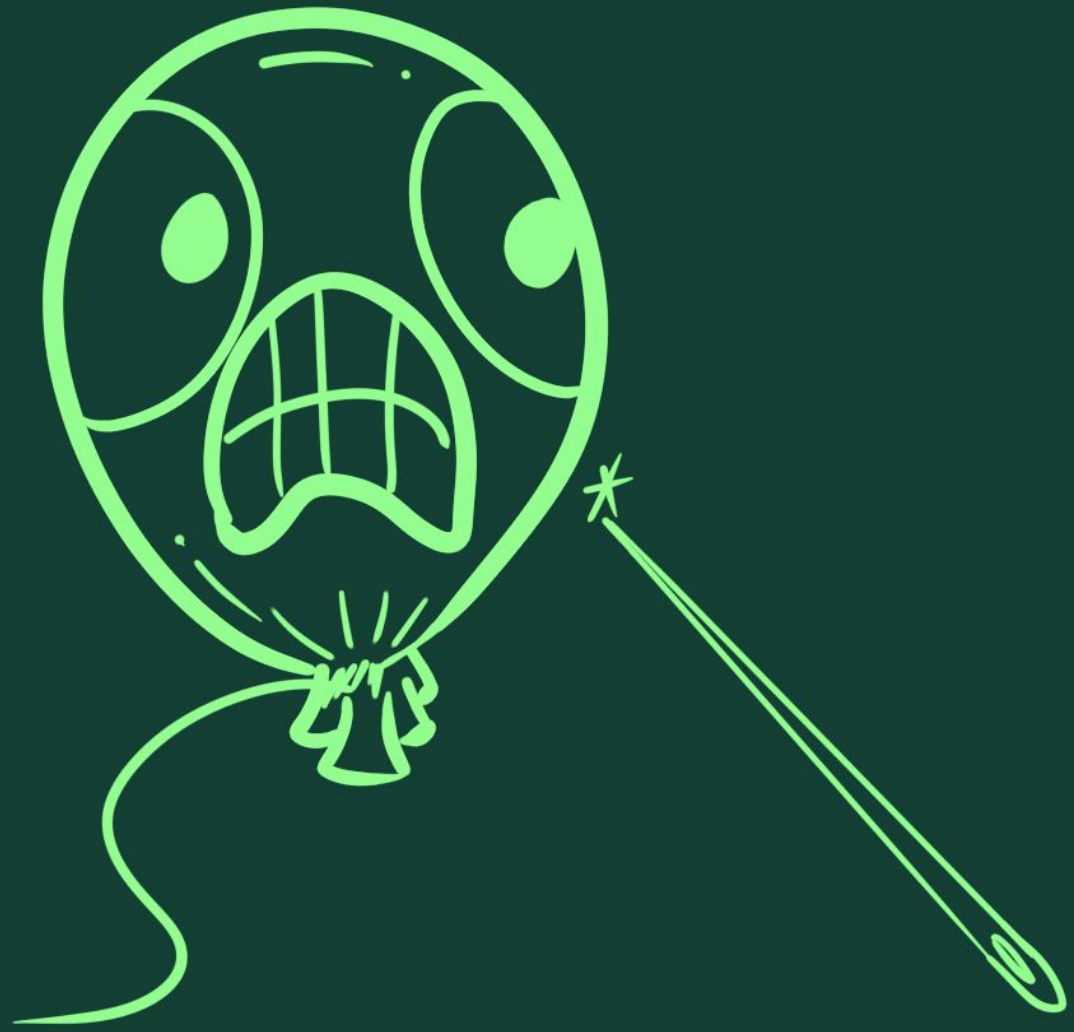
¡Ahora eres un maestro del tiempo y el espacio!

- ¿Por qué se multiplica por deltaTime?
- Movimiento continuo
- Direcciones



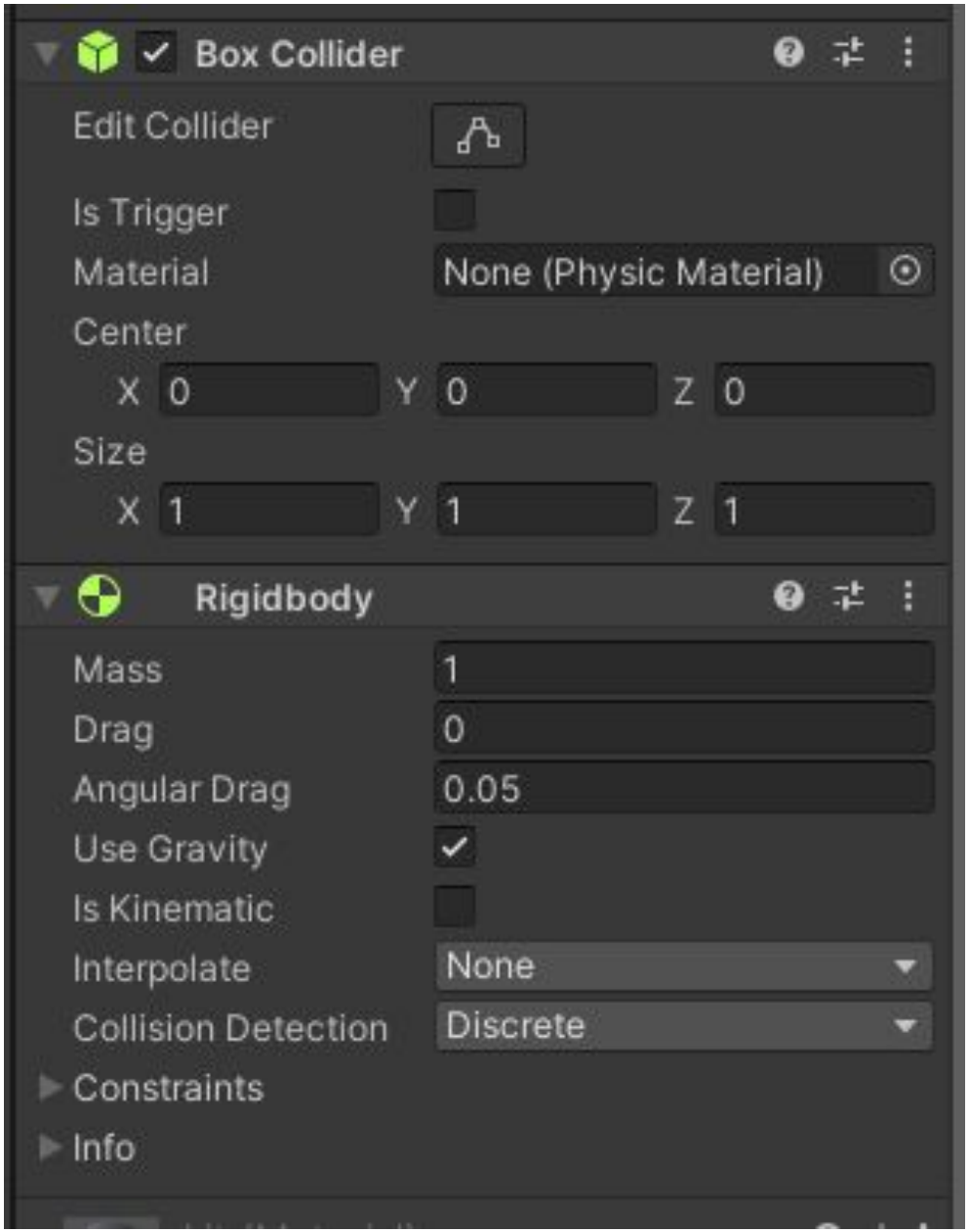


GameObjects que responden a su entorno



¿Están en contacto?

- **Collider**
  - Figuras convexo
- **Rigidbody**
- ~~**Distancia**~~



Cuando hay contacto:

- **Entra en contacto**
  - `OnCollisionEnter(Collision c)`
  - `OnCollisionEnter2D(Collision2D c)`
- **Sale de contacto**
  - `OnCollisionExit(Collision c)`
  - `OnCollisionExit2D(Collision2D c)`
- **Permanece en contacto**
  - `OnCollisionStay(Collision c)`
  - `OnCollisionStay2D(Collision2D c)`

Triggers en contacto

- **Entra**
  - `OnTriggerEnter(Collider c)`
  - `OnTriggerEnter2D(Collider2D c)`
- **Sale**
  - `OnTriggerExit(Collider c)`
  - `OnTriggerExit2D(Collider2D c)`
- **Permanece**
  - `OnTriggerStay(Collider c)`
  - `OnTriggerStay2D(Collider2D c)`

¡Colisiones en acción!

¡Triggers en acción!

Troubleshooting: Colisiones

¡¿Por qué no colisiona?!

RECIBE MENSAJES DE **COLISIONES**		
	Collider	Rigidbody + Collider
Collider	**NO**	**SÍ**
Rigidbody + Collider	**SÍ**	**SÍ**

NINGUNO es Trigger

AL MENOS UNO no es kinemático

¡¿Por qué no colisiona?!

RECIBE MENSAJES DE **TRIGGERS**		
	Collider	Rigidbody + Collider
Collider	**NO**	**SÍ**
Rigidbody + Collider	**SÍ**	**SÍ**

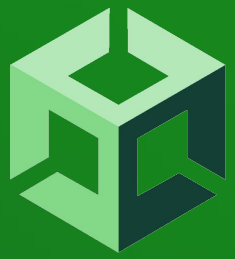
AL MENOS UNO
es Trigger

¡¿Por qué no colisiona?!

Descarga el *step by step*

¡Ahora puedes configurar y detectar colisiones!

- Triggers y colliders
- La fuerza con la que impactan dos Rigidbodies
- Mi Script sabe cuando hay contacto
  - Entra
  - Sale
  - Permanece
- Troubleshooting: ¿Por qué no colisiona?



Sensores láser

Raycast, Spherecast, Planos imaginarios

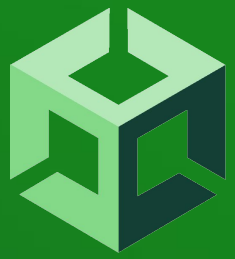
¿Sensor láser?
Raycast

- Línea recta
- Detecta colliders
- Sensible a capas
- ¡Optimizado!
- ¡Conocido!

¡Veamos los Raycast en acción!

Raycasts
(y Spherecasts y planos)

- RaycastHit
- Collider + GetComponentInParent
- Trayectoria recta
- 3D y 2D



Leyendo al jugador

¡Input! Teclado y un poco de Gamepad

Botones

- Input.GetKeyDown(KeyCode.*);
- Input.GetKeyUp(KeyCode.*);
- Input.GetKey(KeyCode.*);

True/False

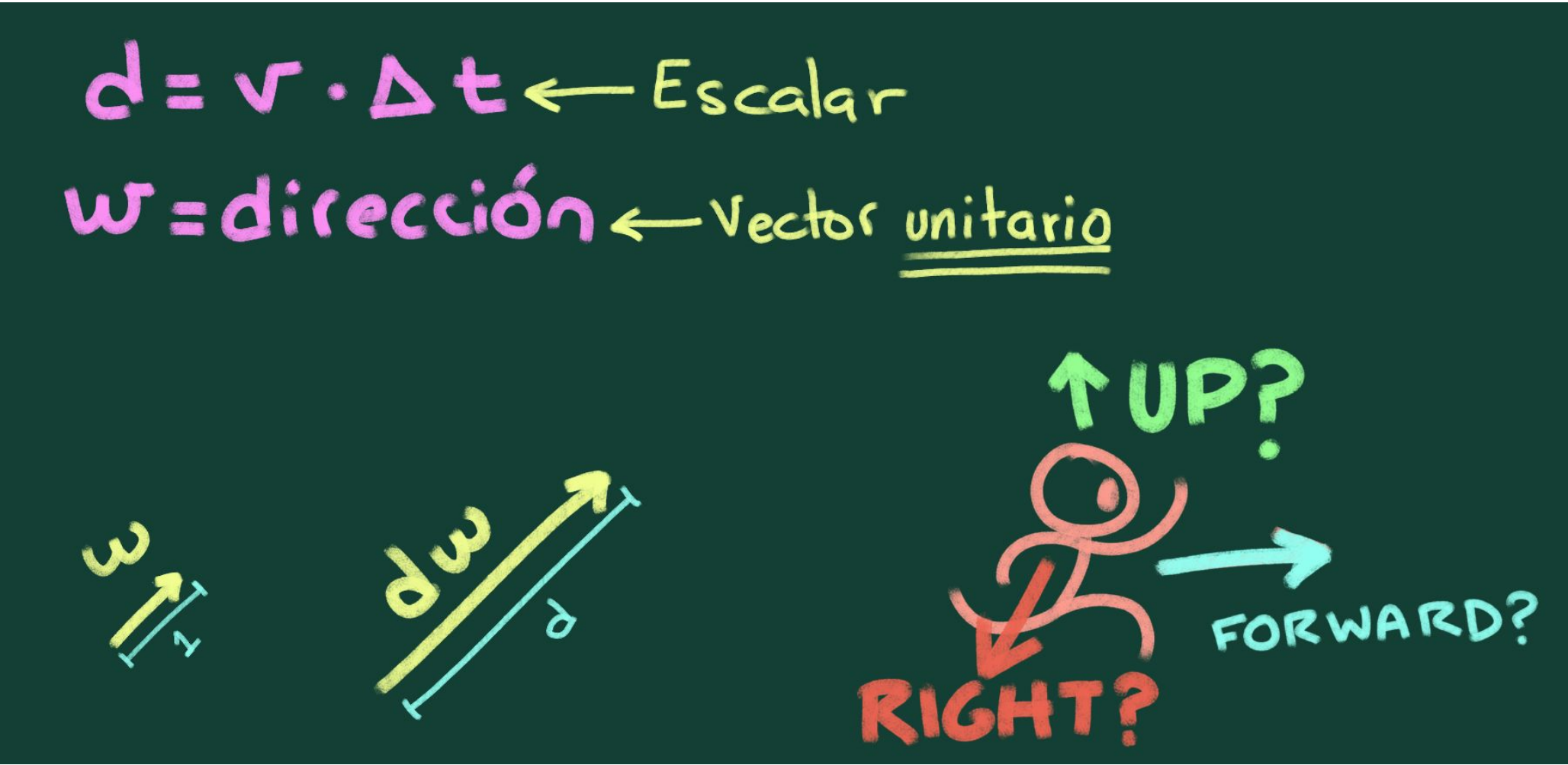
Ejes

Input.GetAxis(“Nombre”);

- **“Horizontal”**
- **“Vertical”**
- **“Mouse X”/”Mouse Y”**
  “Mouse ScrollWheel”

float ∈ [-1, 1]

Moviendo un objeto:
Rapidez y dirección



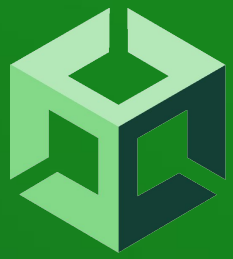
Ejes

Input.GetAxis(“Nombre”);

- **“Horizontal”**
- **“Vertical”**
- **“Mouse X”/”Mouse Y”**
  “Mouse ScrollWheel”

float ∈ [-1, 1]

¡Veamos al input en acción!



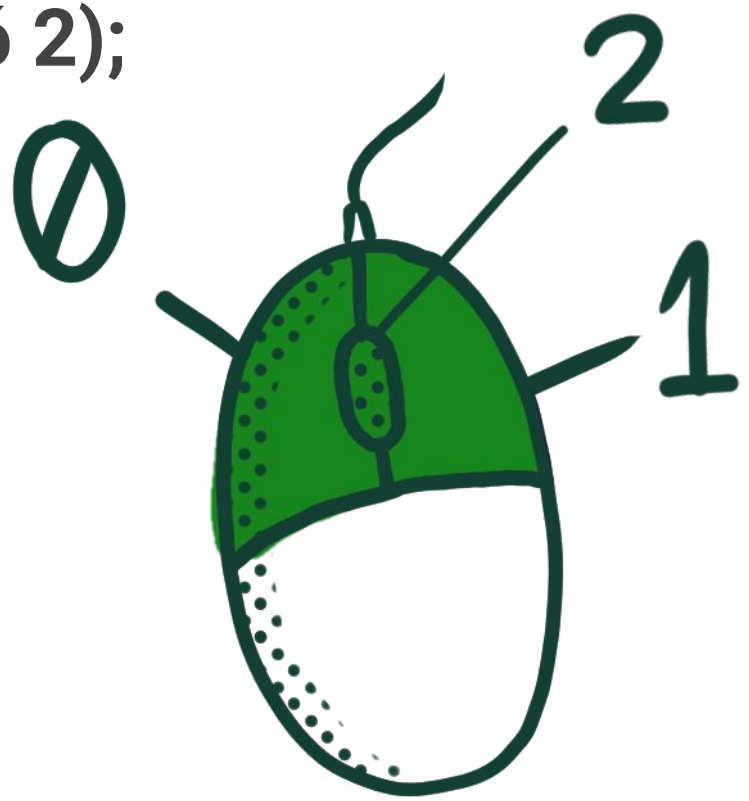
Leyendo al jugador

El Mouse

Mouse: Botones

Input.GetMouseButton(0 ó 1 ó 2);
Down y Up

- 0: Clic izquierdo
- 1: Clic derecho
- 2: Clic del medio



Mouse: Posición

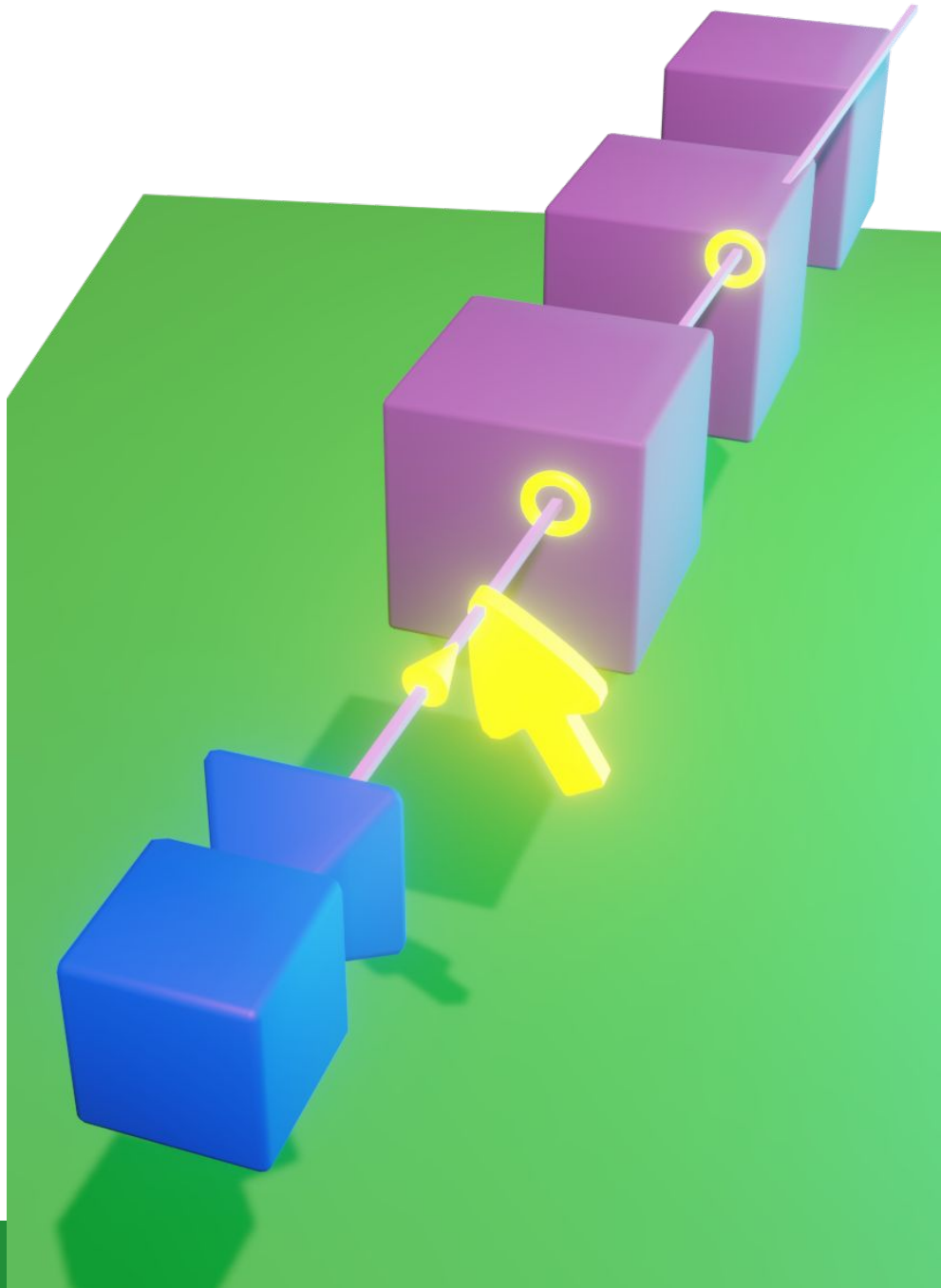
Mouse: Posición

- **Vector3 v = Input.mousePosition;**
- **Vector3 w = ScreenToWorldPoint(v)**

El eje Z es la profundidad

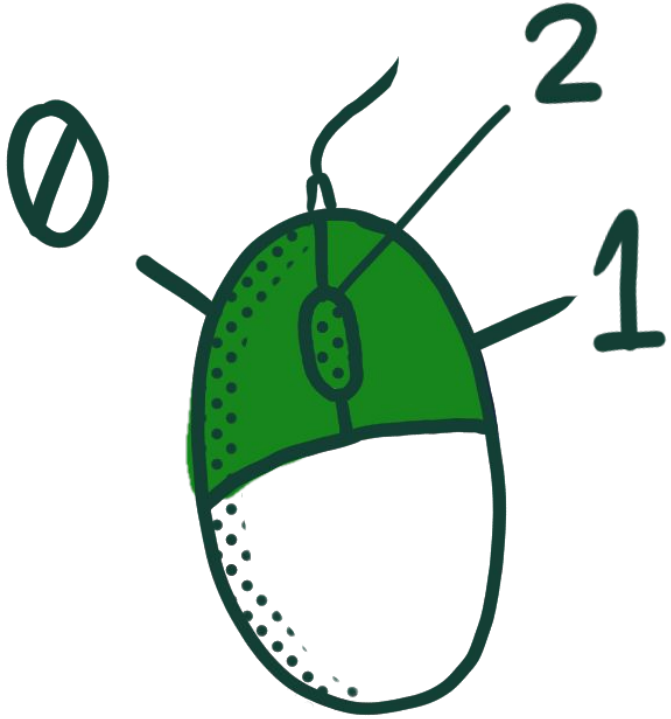
Mouse raycast

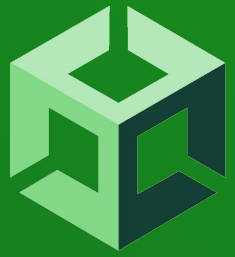
Un rayo que captura colliders que se encuentran bajo el mouse



¡Veámoslo en acción!

Ahora puedes leer el mouse





Encontrando componentes

Encontrar componentes

- Encontrar en este GameObject
  - Xyz a = **GetComponent**<Xyz>()
- Encontrar en algún padre
  - Xyz a = **GetComponentInParent**<Xyz>
- Encontrar en algún hijo
  - Xyz a = **GetComponentInChildren**<Xyz>
- Encontrar en escena
  - Xyz a = **FindObjectOfType**<Xyz>();

Encontrar *todos* los componentes

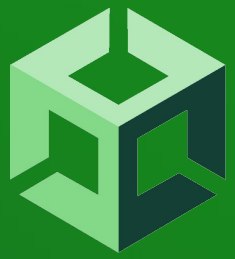
Plural en vez de singular

- **Xyz[]** a = Get**Components**<Xyz>();
- **Xyz[]** a = Find**Objects**OfType<Xyz>();

Veámoslo en acción

Tip de arquitectura:
GameObjects como carpetas

- Abaratar GetComponentInChildren
- Confiar en GetComponentInParent
- ¡Amigable con los Prefabs!



Hijos y padres

Transform: Array de arrays

- transform.GetChild(i);
- transform.childCount;
- transform.Find(“nombre”);
- transform.Find(“a/b/c”);
- transform.parent;

- arr[i]
- arr.Length
- --
- --
- --

```
for (int i=0; i<transform.childCount; i++) {
  Debug.Log(transform.GetChild(i);
}

foreach (Transform child in transform) {
  Debug.Log(child);
}

// ------------------------------------

void PrintAll (Transform t) {
  foreach (Transform child in t) {
    Debug.Log(child);
    PrintAll(child);
  }
}
PrintAll(transform);
```

Veámoslo en acción

Tip de arquitectura:
A veces: transform > array

PROS:

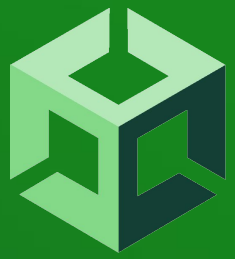
- Se añaden/eliminan objetos automáticamente
- No se borra por accidente

CONS:

- Tienen que compartir un mismo padre

¡Ahora puedes comunicarte con parents y children!

- transform.childCount
- transform.GetChild(i)
- transform.parent
- foreach (Transform child in transform)



Comunicación telepática

Eventos

¡Creemos un evento!

¡Invoquemos un evento!

¡Subscribámonos/ Desubscribámonos a un evento!

Eventos estáticos vs. eventos normales

¡Tu turno!

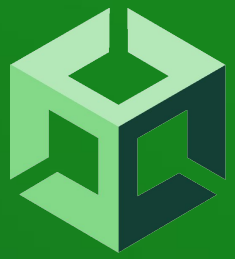
El timbre del recreo de cada escuela en una zona en un juego de simulación tipo SimCity.

¡Tu turno!

En *Cult of the Lamb*, un enemigo mató al borreguito y su culto debe perder fe.

¡Tu turno!

En MGS V se acaba de desatar una tormenta de arena, la cual reduce la visibilidad de los enemigos, reduce la probabilidad de sobrevivir a la maniobra Fulton y hace que el apoyo aéreo sea inaccesible.



Inicializando variables

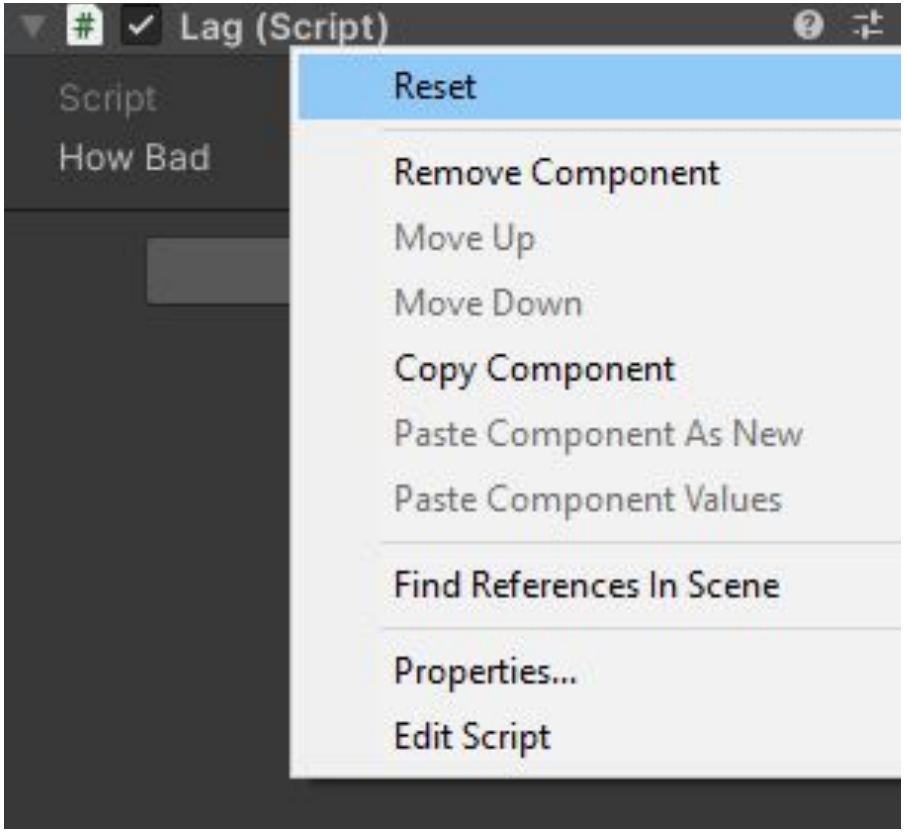
Un tip de arquitectura

void Reset()

Se ejecuta cuando un script…

- Es añadido
- Se resetea

NUNCA SE EJECUTA EN PLAY MODE



Tip de arquitectura:

Reset() > Start()

PROS:

- Difícil de romper
- Legible
- Flexible

CONS:

- Solo en Editor

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class ResetVsStart : MonoBehaviour {
  [Header("Configuration")]
  public float jumpHeight = 5;

  [Header("Information")]
  public float jumpVelocity;

  [Header("Initialization")]
  public Rigidbody body;
  public Collider c;

  void Reset () {
    float a = Physics.gravity.magnitude;
    float v = 2 * a * jumpHeight;
    jumpVelocity = Mathf.Sqrt(v); // NOOOO!!
    body = GetComponent<Rigidbody>();
    c = GetComponent<Collider>();
  }

  void Start () {
    float a = Physics.gravity.magnitude;
    float v = 2 * a * jumpHeight
    jumpVelocity = Mathf.Sqrt(v); // OKI!
    body = GetComponent<Rigidbody>();
    c = GetComponent<Collider>();
  }
```

Tip de arquitectura:

get > Start()

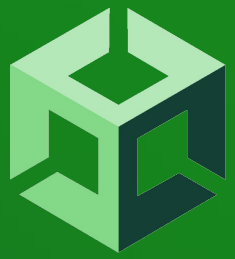
PROS:

- SoT
- Legible (Código)

CONS:

- Caro*
- Illegible (Editor)

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class ResetVsStart : MonoBehaviour {
  [Header("Configuration")]
  public float jumpHeight = 5;

  // [Header("Information")]
  public float JumpVelocity {
    get {
      float a = Physics.gravity.magnitude;
      return Mathf.Sqrt(2 * a * jumpHeight)
    }
  }

  [Header("Initialization")]
  public Rigidbody body;
  public Collider c;

  void Reset () {
    body = GetComponent<Rigidbody>();
    c = GetComponent<Collider>();
  }

  public void Jump () {
    body.AddForce(JumpVelocity,
                  ForceMode.velocityChange)
  }
}
```



Configuración, Información e Inicialización

Un tip de arquitectura

Configuración

- Se pueden modificar libremente
- ¡Cualquier valor!

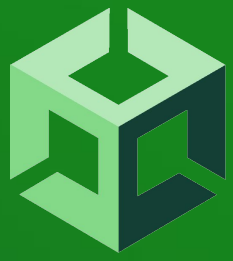
Información

- Seteadas por este mismo script
- Modifícala bajo tu propio riesgo

Inicialización

- Setear en el *Reset*, *Start* o *Editor*
- Su valor es fundamental

¡Hagamos un script con esta estructura!



¿Qué hace un ingeniero de herramientas?

y ¿Cómo se crean herramientas para Unity?

¿Ingeniero de herramientas?

- Edit mode != Play mode
- Facilitan el uso del motor de juegos
  - Editores de niveles
  - Editor de configuración
  - Identifican necesidades de desarrollo
  - ¡Proponen soluciones y las implementan!

Algunas herramientas

- ¡Gizmos y Debug!
  - **void OnDrawGizmos () {}**
  - **Gizmos** class
- **void Reset () {}**
- **[ExecuteInEditMode]**
- **Editor** class

¡Veamos los Gizmos en acción!

Gizmos y Debug

- **Dibuja** overlays en escena
- Solo en **modo edición**
- Va en **OnDrawGizmos**
- **Debug** puede ir en Update

¡Veamos
[ExecuteInEditMode]
en acción!

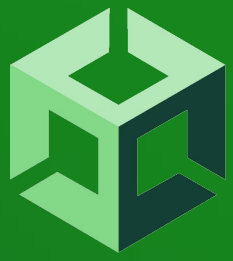
[ExecuteInEditMode]

- Modo edición y modo juego
- **if (Application.isPlaying)**
- **DestroyImmediate**
- **PrefabUtility.InstantiatePrefab**
- **#if UNITY_EDITOR / #endif**

¡Veamos `Edit` class en acción!

¡Ahora conoces el rol del Ingeniero de Herramientas!

- Facilita el uso del editor
- Herramientas:
  - Editor
    - Botones (Inspector y escena)
    - Raycast (Escena)
    - Undo
  - Reset
  - [ExecuteInEditMode]
  - Gizmos y Debug

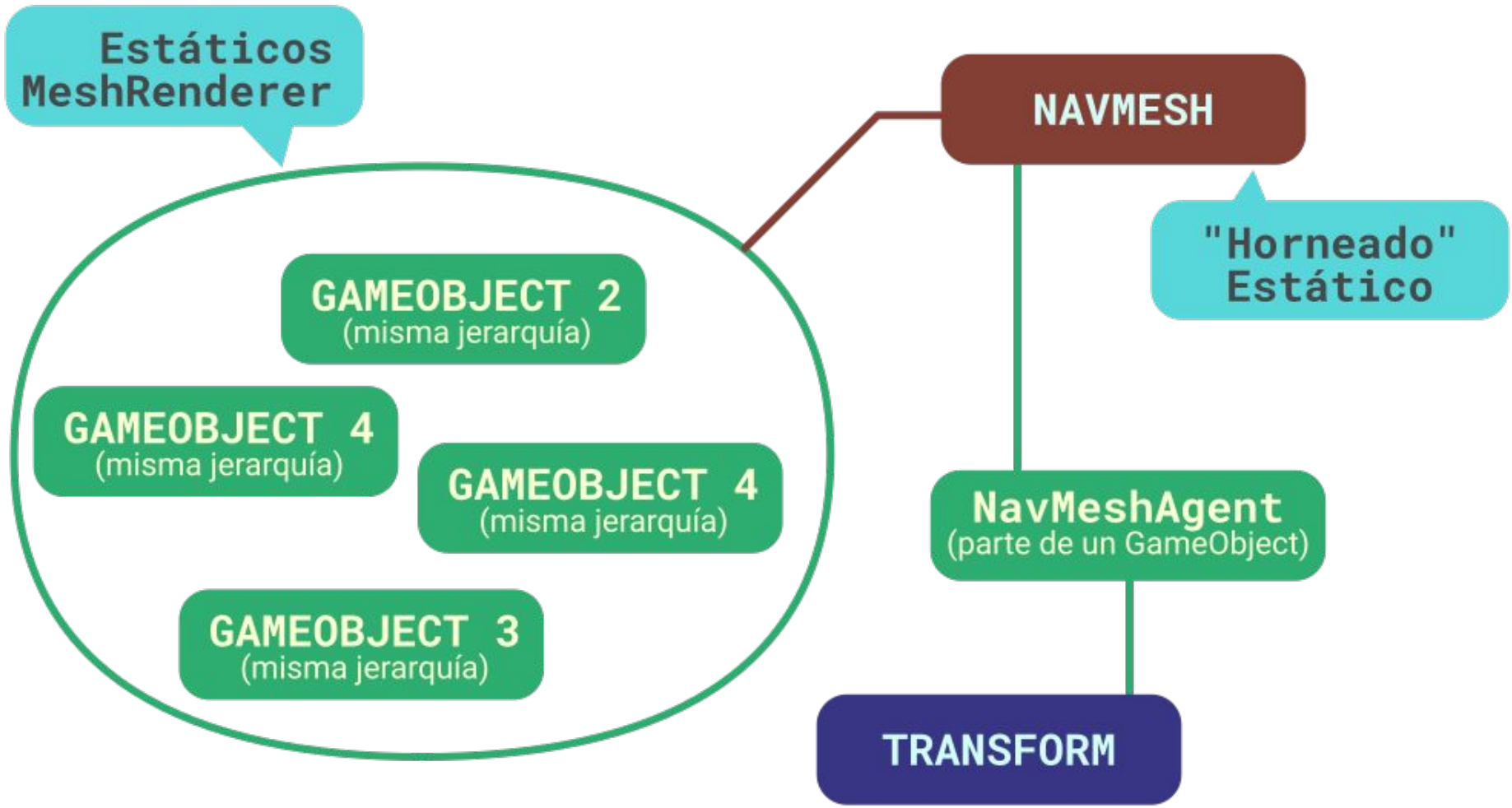


Inteligencia artificial

En un chasquido

Navigation Mesh

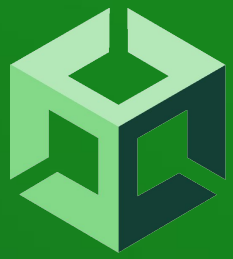
- Estático y horneado
- No usa colliders ni física
- Indica áreas caminables
- Encuentra paths óptimamente


Estáticos
MeshRenderer
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)
NAVMESH
"Horneado"
Estático
NavMeshAgent
(parte de un GameObject)
TRANSFORM

¡Veamos al NavMesh en acción!

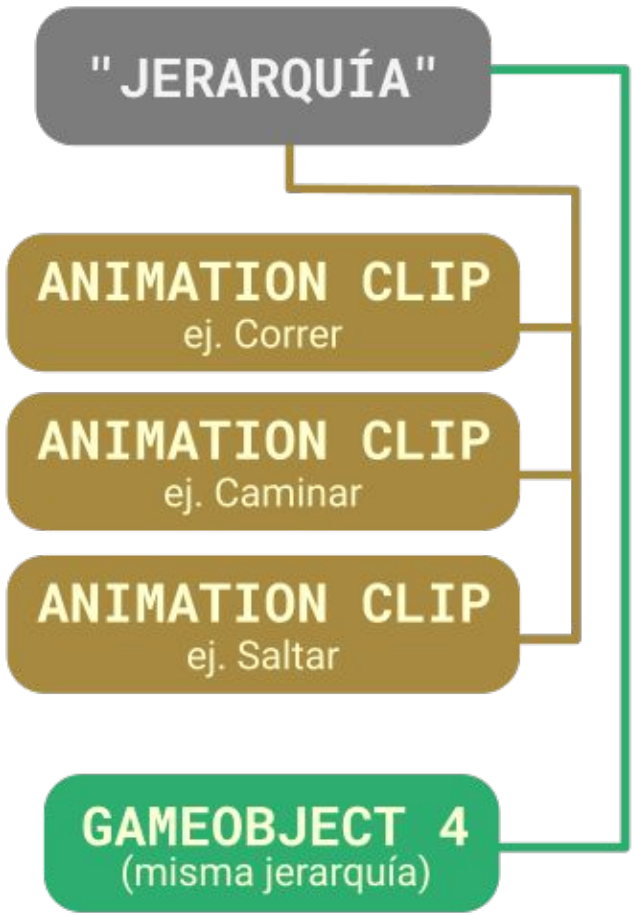
¡Ahora puedes crear NavMesh!

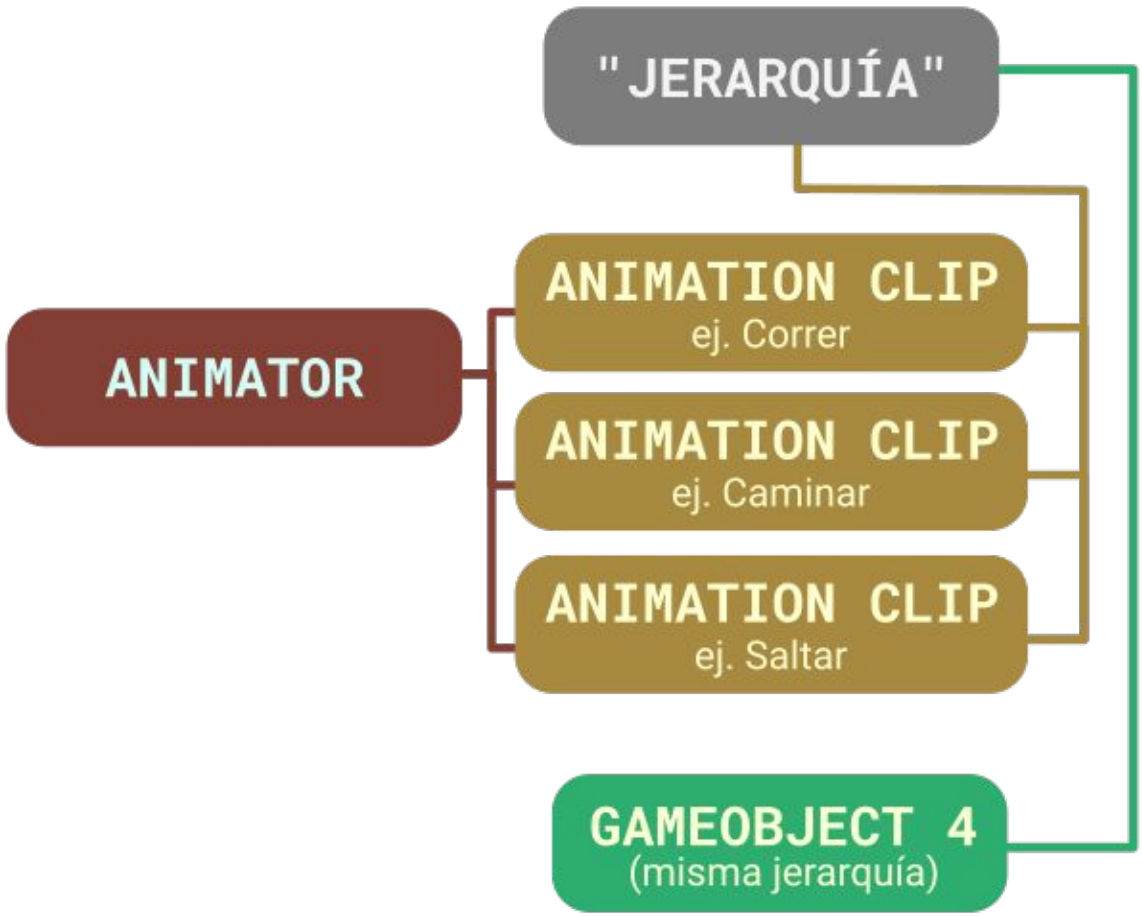
- NavMesh: Horneado y estático
- NavMeshAgent: navegan el NavMesh
  - SetDestination
  - Path
  - Acceleration
  - Warp
- SetDestination es *asíncrono*

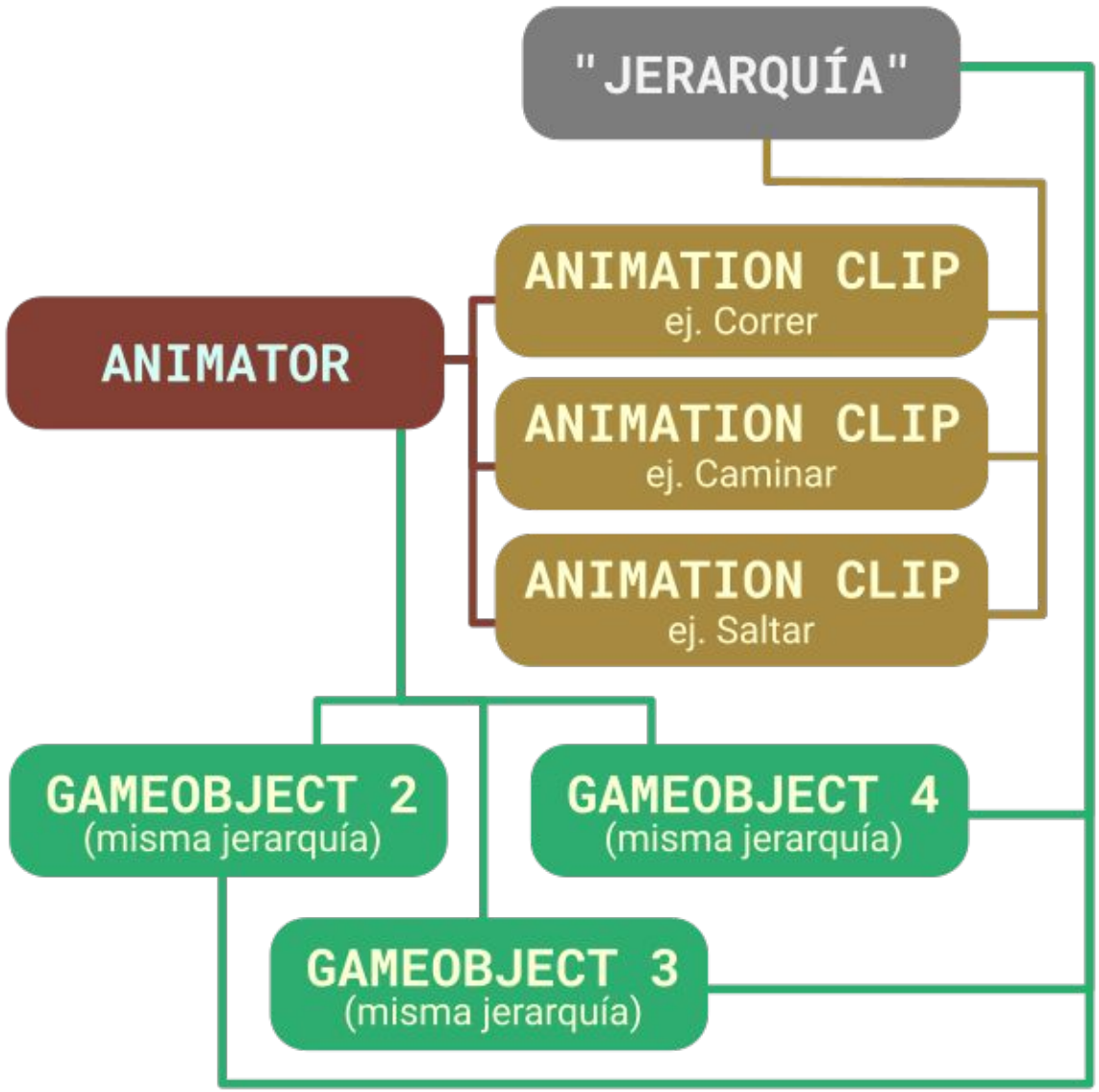


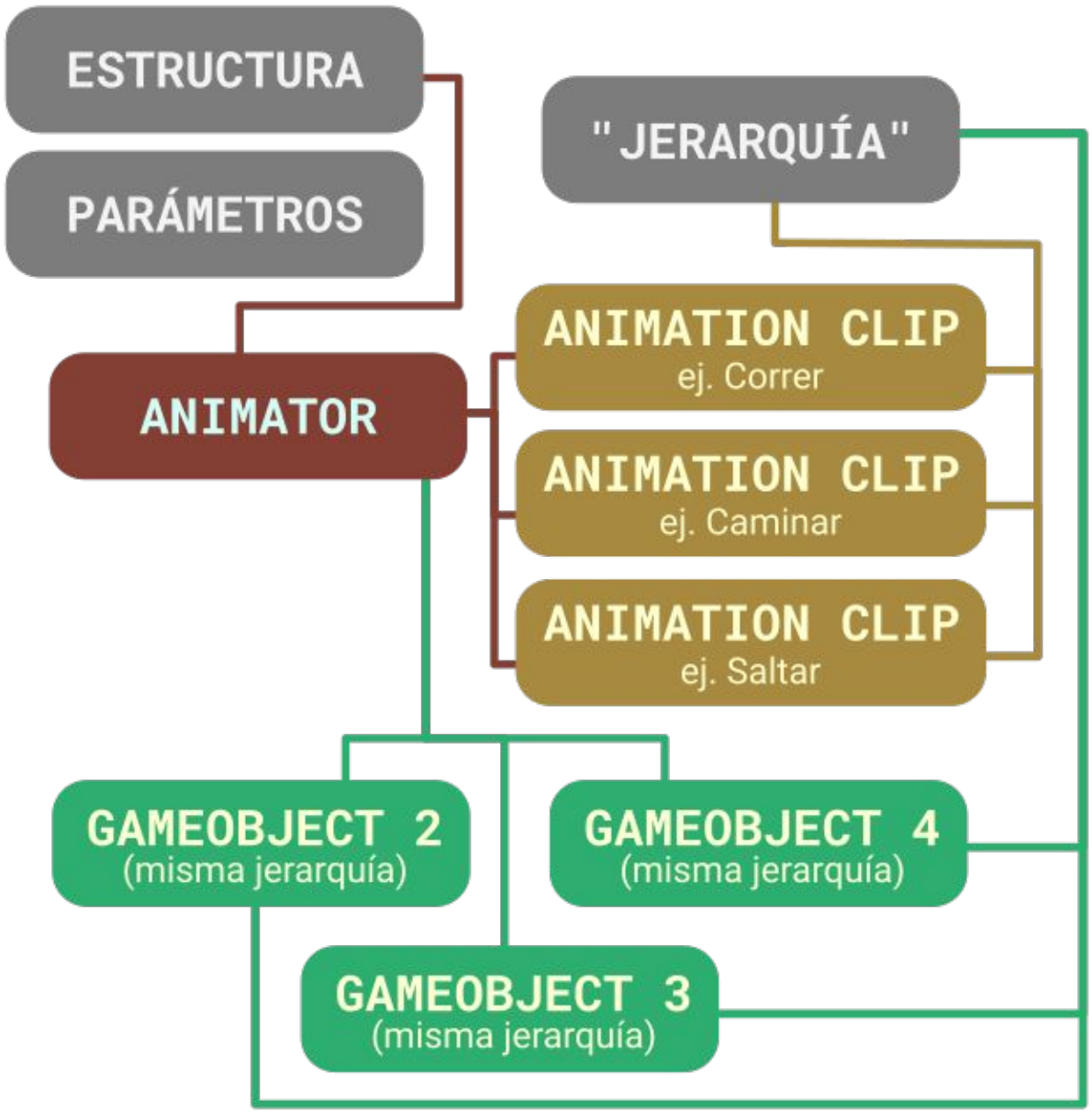
Animaciones

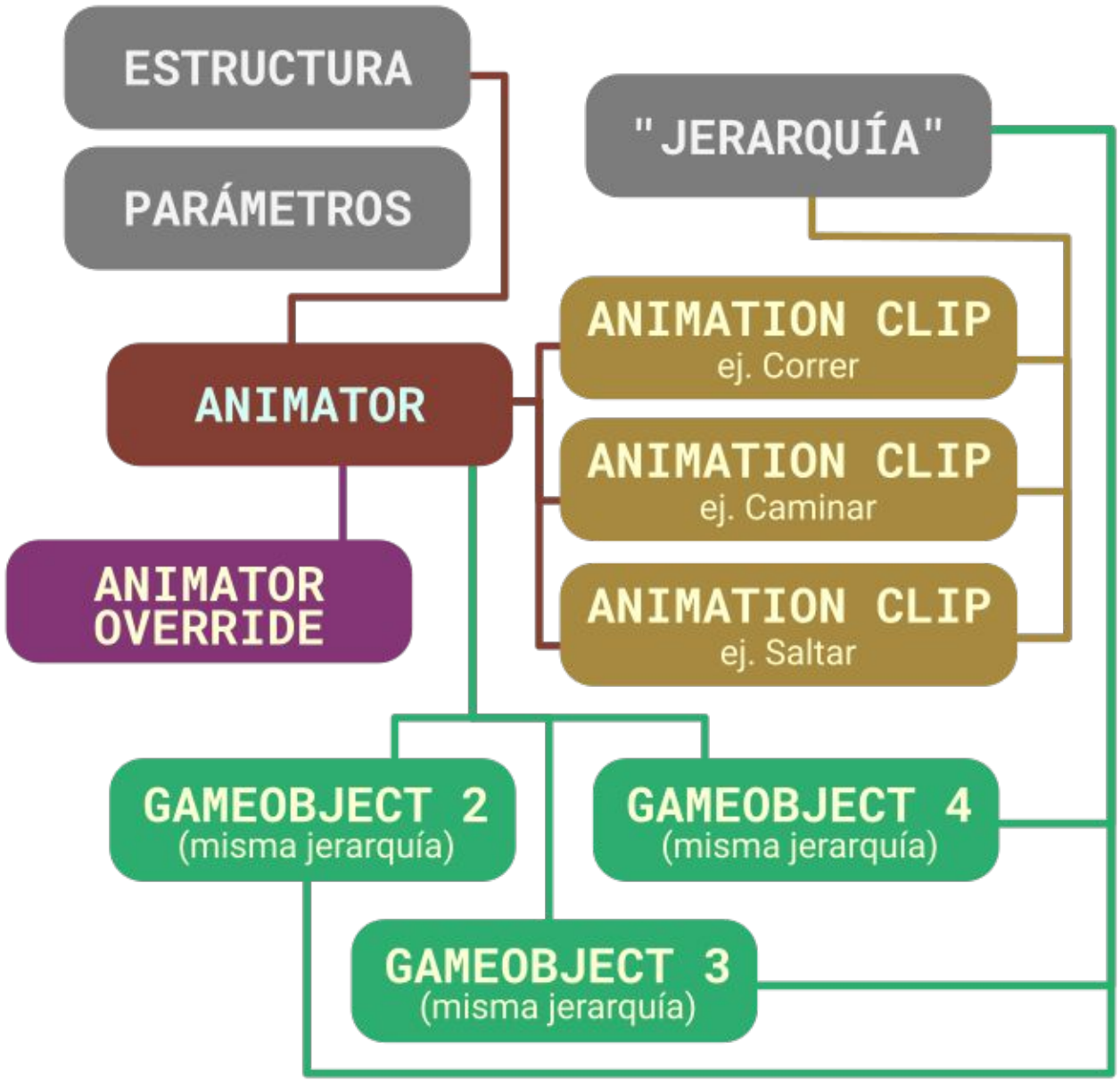
2D y 3D

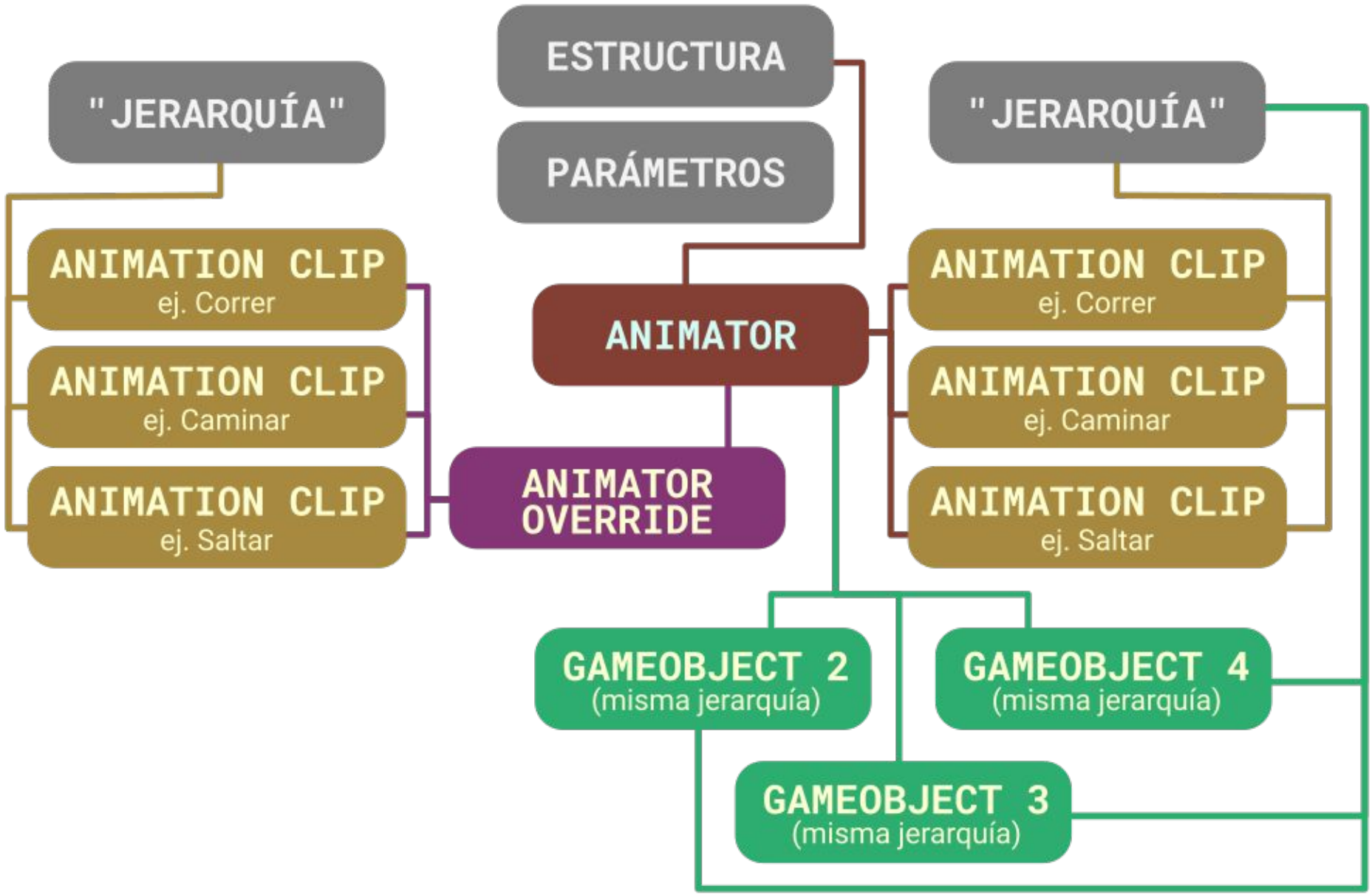

"JERARQUÍA"
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 4
(misma jerarquía)

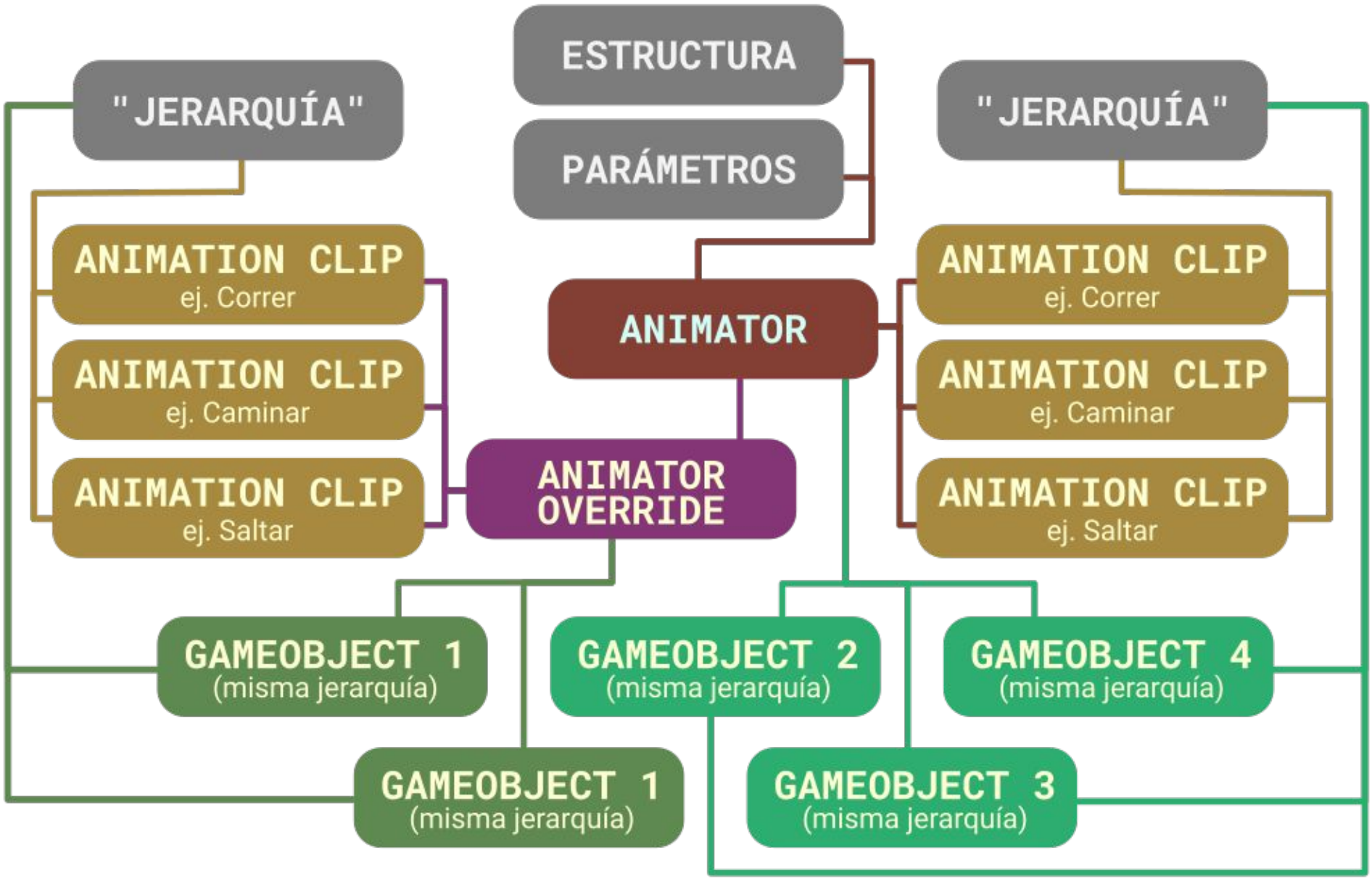

"JERARQUÍA"
ANIMATOR
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 4
(misma jerarquía)


"JERARQUÍA"
ANIMATOR
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)


ESTRUCTURA
PARÁMETROS
"JERARQUÍA"
ANIMATOR
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)


ESTRUCTURA
PARÁMETROS
"JERARQUÍA"
ANIMATOR
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
ANIMATOR OVERRIDE
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)

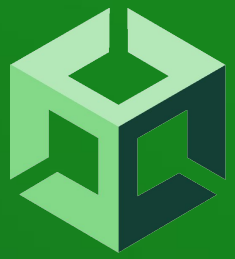

ESTRUCTURA
PARÁMETROS
"JERARQUÍA"
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
ANIMATOR
ANIMATOR OVERRIDE
"JERARQUÍA"
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)


ESTRUCTURA
PARÁMETROS
"JERARQUÍA"
"JERARQUÍA"
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
ANIMATOR
ANIMATOR OVERRIDE
ANIMATION CLIP
ej. Correr
ANIMATION CLIP
ej. Caminar
ANIMATION CLIP
ej. Saltar
GAMEOBJECT 1
(misma jerarquía)
GAMEOBJECT 2
(misma jerarquía)
GAMEOBJECT 4
(misma jerarquía)
GAMEOBJECT 1
(misma jerarquía)
GAMEOBJECT 3
(misma jerarquía)

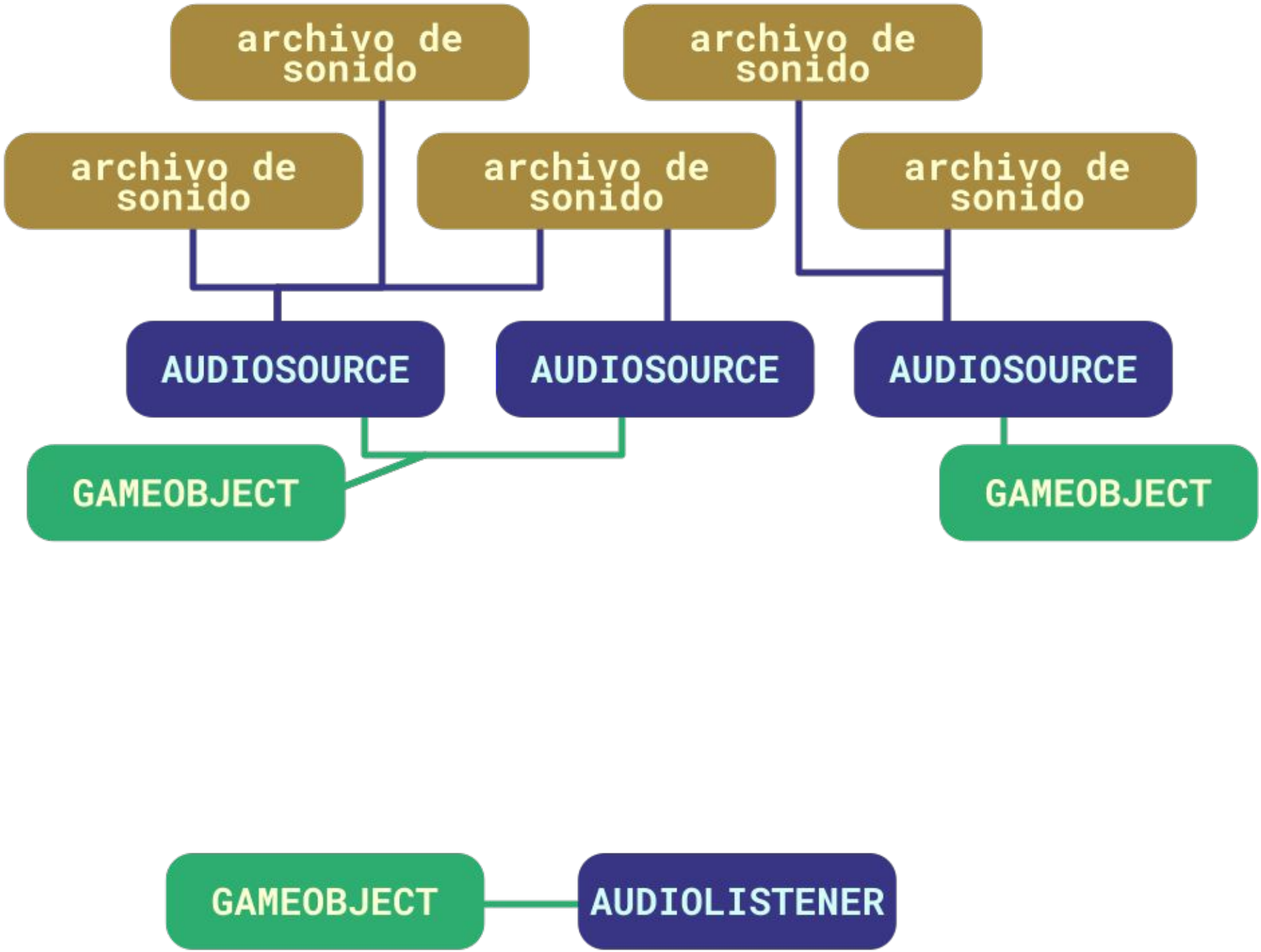
¡Veamos las animaciones en acción!

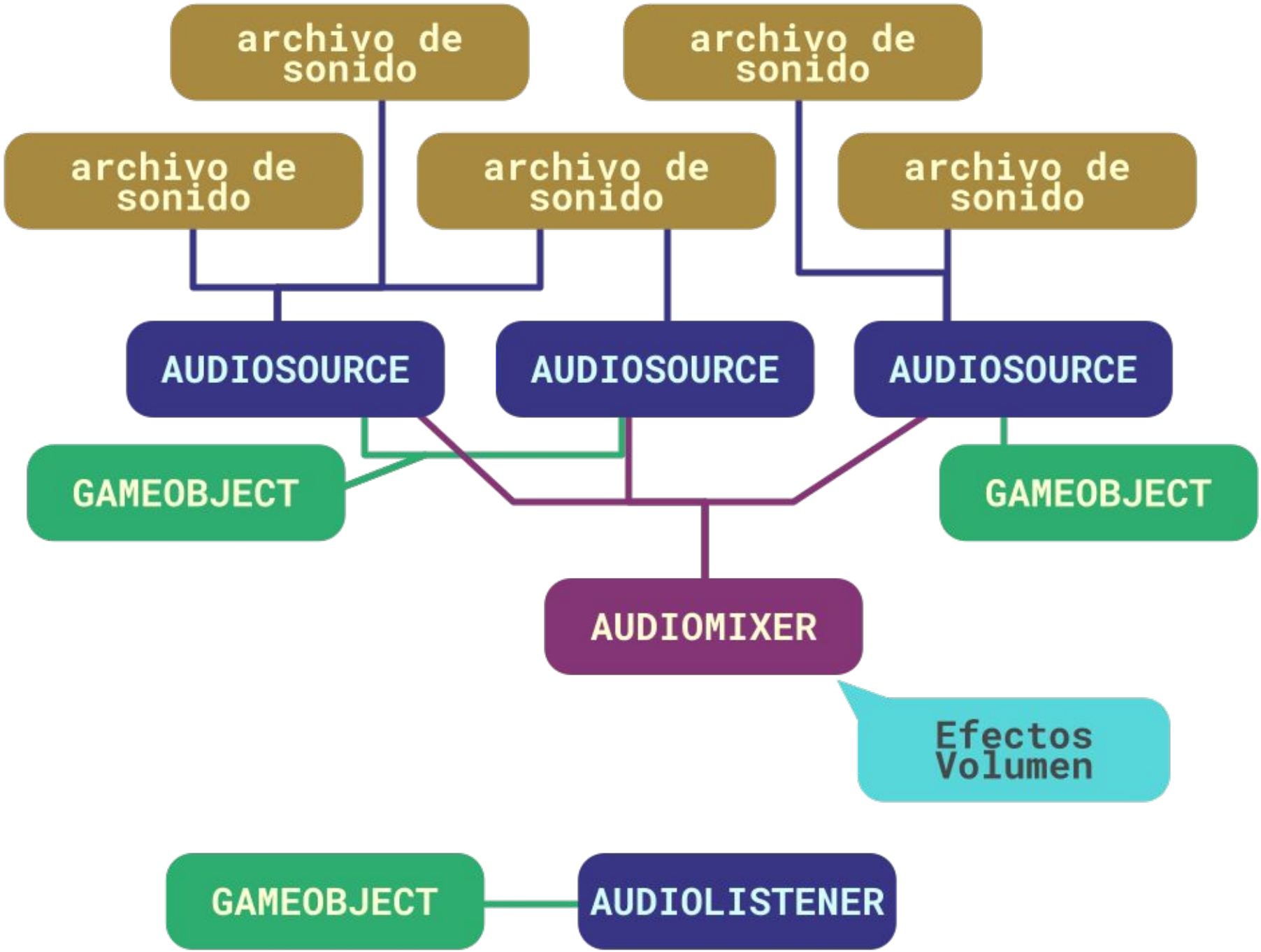
¡Ahora puedes crear y controlar animaciones!

- Dependen de la jerarquía y nombres
- Animator:
  - Máquina de estado
- Parámetros:
  - Comunicación script -> animaciones
- Animator Override:
  - *Animator* con diferentes *animations*.



AudioMixers

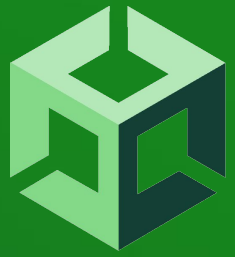

archivo de sonido
archivo de sonido
archivo de sonido
archivo de sonido
archivo de sonido
AUDIOSOURCE
AUDIOSOURCE
AUDIOSOURCE
GAMEOBJECT
GAMEOBJECT
GAMEOBJECT
AUDIOLISTENER


archivo de sonido
archivo de sonido
archivo de sonido
archivo de sonido
archivo de sonido
AUDIOSOURCE
AUDIOSOURCE
AUDIOSOURCE
GAMEOBJECT
GAMEOBJECT
AUDIOMIXER
Efectos
Volumen
GAMEOBJECT
AUDIOLISTENER

¡Veamos un AudioMixer en acción!

AudioMixers

- Agrupan AudioSources
- Controlan el sonido
- Añaden efectos



¿Qué es el UI...

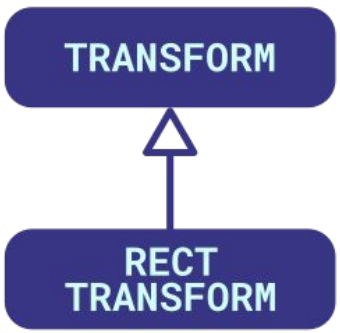
... y cómo se crea?

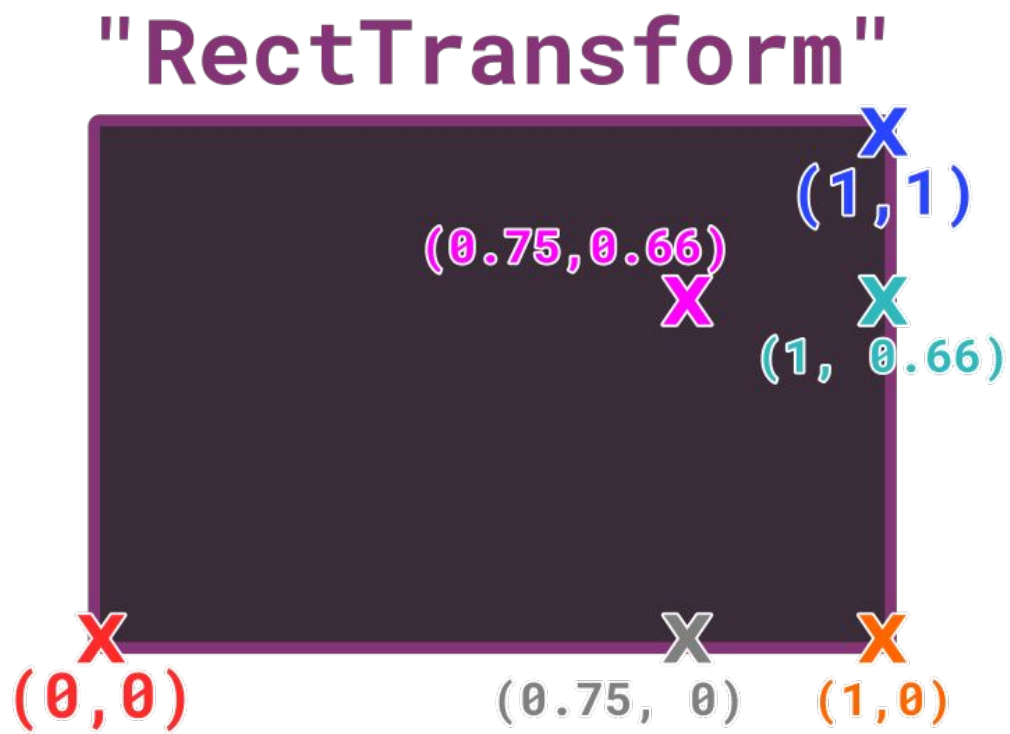
Screen space != World space

¡Canvas space!

- Canvas space != World space != Screen space
- Transform -> RectTransform
- Se renderiza “a su manera”

¿RectTransform?



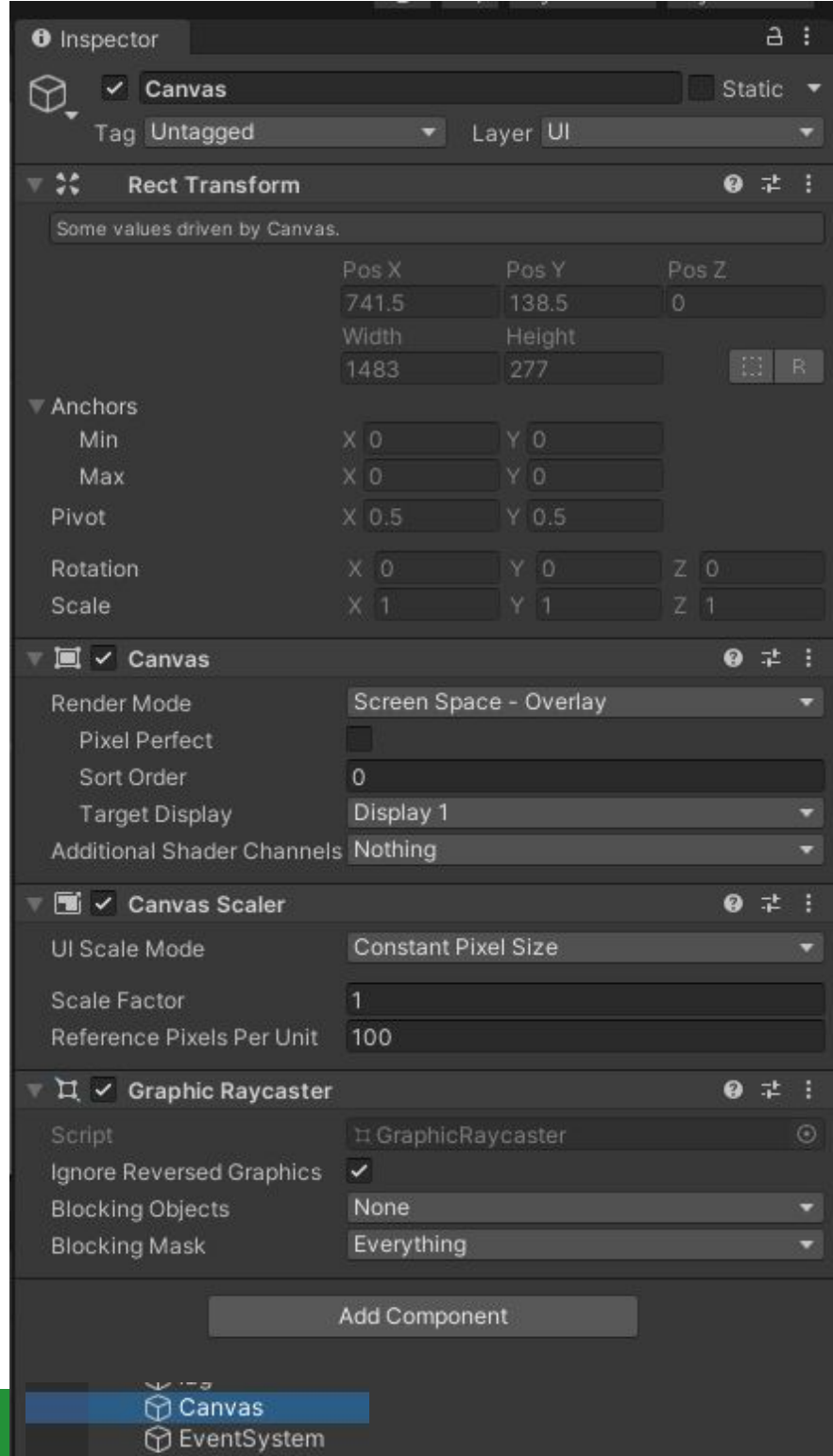


El Canvas

Canvas:
Cómo se renderiza

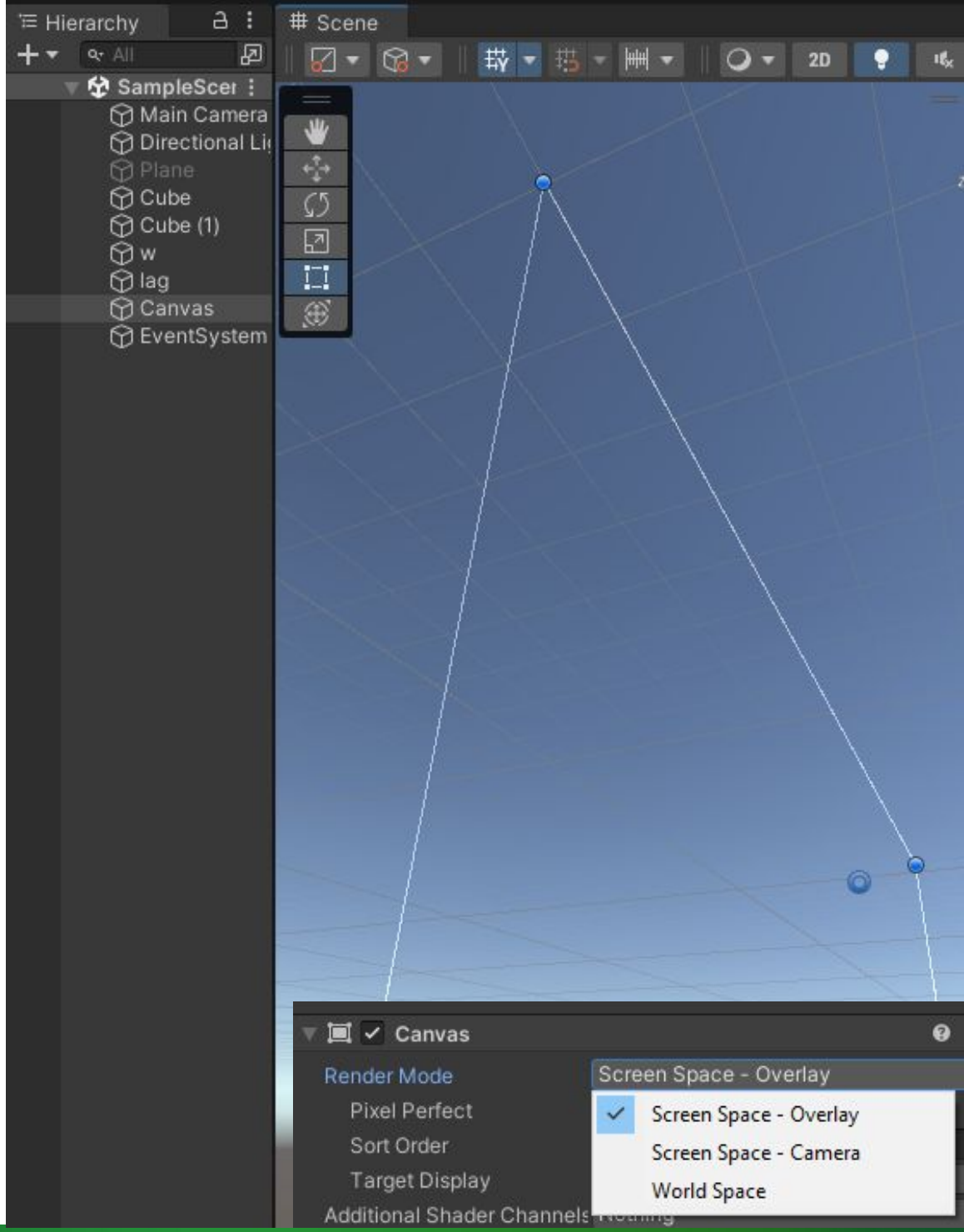
CanvasScaler:
Responsividad

GraphicRaycaster:
Interacción



RenderMode

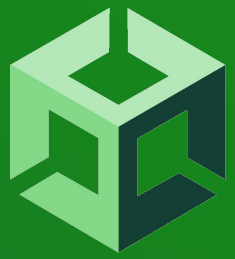
- Screen space - Overlay
- Screen space - Camera
- WorldSpace



¡Creemos un canvas!

¡Ahora puedes crear UI!

- Diferentes render modes
- Solucionar problemas de interacción
- Layouts > Posicionar a mano
- Responsividad: Anchors y pivotes



Manejo de cámaras con Cinemachine

(Y cómo instalar paquetes)

PackageManager

¡Ahora puedes instalar paquetes y usar Cinemachine!

- Activar/Desactivar cámaras
- Transiciones suaves
- Seguimiento automático
- PackageManager