

Godot Engine es un motor de videojuegos multiplataforma con múltiples características para crear juegos 2D y 3D desde una interfaz unificada. Él provee un conjunto exhaustivo de herramientas comunes para que los usuarios puedan enfocarse en crear juegos sin tener que reinventar la rueda. Juegos que pueden exportarse en un sólo clic a numerosas plataformas, incluyendo las principales plataformas de escritorio (Linux, macOS, Windows), móviles (Android, iOS) y basadas en la web (HTML5).

Godot es completamente gratuito y de código abierto bajo la licencia permisiva del MIT. Sin condiciones, sin regalías, nada. Los juegos de los usuarios son suyos, hasta la última línea del código del motor. El desarrollo de Godot es totalmente independiente y dirigido por la comunidad, lo que permite a los usuarios ayudar a dar forma a su motor para que coincida con sus expectativas. Está respaldado por Software Freedom Conservancy sin fines de lucro.

Godot es Software Libre y de Código Abierto, disponible bajo la licencia MIT aprobada por la OSI. Esto significa que es libre como en "libre expresión" y gratis como en "cerveza gratis".

En pocas palabras:

Eres libre de descargar y usar Godot para cualquier propósito, personal, sin fines de lucro, comercial o de otro tipo.

Eres libre para modificar, distribuir, redistribuir y mezclar Godot como quieras, por cualquier razón, ya sea no comercial o comercialmente.

Todos los contenidos de esta documentación acompañante son publicados bajo la permisiva licencia Creative Commons Attribution 3.0 (CC-BY 3.0), con atribución a "Juan Linietsky, Ariel Manzul y a la comunidad de Godot Engine".

Los logos e iconos están generalmente bajo la misma licencia Creative Commons. Ten en cuenta que algunas librerías de terceros, incluidas en el código fuente de Godot, pueden tener diferentes licencias.

Para más detalles, revisa los archivos COPYRIGHT.txt

(<https://github.com/godotengine/godot/blob/master/COPYRIGHT.txt>)

al igual que LICENSE.txt

(<https://github.com/godotengine/godot/blob/master/LICENSE.txt>)

y LOGO\_LICENSE.txt en el repositorio de Godot.

([https://github.com/godotengine/godot/blob/master/LOGO\\_LICENSE.md](https://github.com/godotengine/godot/blob/master/LOGO_LICENSE.md)) en el repositorio de Godot.

Mira también la página de la licencia en el sitio web de Godot.

## Plataformas

Para el editor:

Windows  
macOS  
X11 (Linux, \*BSD)

Para exportar tus juegos:

Windows (y UWP)  
macOS  
X11 (Linux, \*BSD)  
Android  
iOS  
Web

## Lenguajes de programación

Los lenguajes soportados oficialmente por Godot son GDScript, Visual Scripting, C# y C++.

## Características del Motor

### Plataformas

Puede correr, tanto el editor como los proyectos exportados:

Windows 7 y posteriores (64-bit y 32-bit).  
macOS 10.12 y posteriores (64-bit, x86 y ARM).  
Linux (64 y 32 bits, x86 y ARM).

Los binarios están vinculados estáticamente y pueden ejecutarse en cualquier distribución si se compilan en una distribución base lo suficientemente antigua.  
Los binarios oficiales se compilan en Ubuntu 14.04.

Ejecutando proyectos exportados:

Android 4.4 y posterior.  
iOS 10.0 y posteriores.  
HTML5 vía WebAssembly (Firefox, Chrome, Edge, Opera).  
Consolas.  
Godot pretende ser la plataforma más independiente y que pueda migrar a otras plataformas con relativa facilidad.

## Editor

### Características:

Editor del Árbol de Escenas.  
Editor de Scripts.  
Soporte para editores externos de código.  
GDScript debugger.

Herramientas de supervisión del rendimiento (Depurador).

Recarga de script en vivo.

Edición de escenas en vivo.

Los cambios se reflejarán en el editor y se mantendrán después de haber cerrado el proyecto en curso.

Inspector remoto.

Los cambios no se reflejarán en el editor y no se mantendrán después de cerrar el proyecto en ejecución.

Réplica de la cámara en vivo.

Mueve la cámara en el editor y mira el resultado en el proyecto ejecutándose.

Usa el editor en docenas de lenguajes contribuidos por la comunidad.

Plugins:

Los plugins del editor se pueden descargar de la librería de assets para extender la funcionalidad del editor.

Crea tus propios plugins usando GDScript para añadir nuevas características para acelerar tu flujo de trabajo.

Descarga proyectos desde la librería de assets en el administrador del proyecto e impórtalos directamente.

Gráficos 2D

Dos renderizadores disponibles :

Renderizador OpenGL ES 3.0 (usa OpenGL 3.3 en plataformas de escritorio).

Visuales de gama alta. Recomendadas en plataformas de escritorio.

Renderizador OpenGL ES 2.0 (usa OpenGL 2.1 en plataformas de escritorio).

Recomendado en plataformas web y de móviles.

Características:

Renderización de sprites, polígonos y líneas.

Herramientas de gama alta para dibujar líneas y polígonos tales como Polygon2D y Line2D.

AnimatedSprite como ayuda para crear sprites animados.

Capas de parallax.

Soporte pseudo-3D al duplicar automáticamente una capa varias veces.

Iluminación 2D con mapas normales.

Sombras definidas o suaves.

Renderizado de fuentes usando mapas de bits (BitmapFont) o rasterización usando FreeType (DynamicFont).

Fuentes de Bitmap pueden ser exportadas usando herramientas como BMFont.

DynamicFont soporta fuentes monocromáticas, así como fuentes coloridas. Los formatos soportados son TF y OTF.

DynamicFont admite contornos de fuentes opcionales con ancho y color ajustables.

Soporte para sobremuestreo de fuentes para mantener fuentes nítidas en resoluciones más altas.

Partículas basadas en GPU con soporte para sombreadores de partículas personalizados.

Partículas basadas en la CPU.

#### Herramientas 2D

Cámara 2D con suavizado integrado y márgenes de arrastre.

Nodo Path2D para representar una ruta en el espacio 2D.

Pueden ser dibujados en el editor o generados proceduralmente.

Nodo PathFollow2D para hacer que los nodos sigan un Path2D.

Clase auxiliar de geometría 2D.

Nodo Line2D para dibujar líneas 2D texturizadas.

#### Física 2D

Cuerpos físicos:

Cuerpos estáticos.

Cuerpos rígidos.

Cuerpos cinemáticos.

Articulaciones.

Zonas para detectar cuerpos que entran o salen de él.

Detección de colisiones:

Formas integradas: línea, caja, círculo, cápsula.

Polígonos de colisión (se pueden dibujar manualmente o generar desde un objeto en el editor).

Cámara:

Cámaras de perspectiva, ortográficas y offset troncocónicas.

**\*\* Representación basada en la física: \*\***

Sigue el modelo Disney PBR.

Utiliza un flujo de trabajo de rugosidad metálica con soporte para texturas ORM.

Mapeo de normales.

GLS3: Parallax/relief mapping with automatic level of detail based on distance.

GLS3: Sub-surface scattering and transmittance.

GLS3: Proximity fade (soft particles).

Desvanecimiento de distancia que puede usar mezcla alfa o difuminado para evitar pasar por la tubería transparente.

El difuminado se puede determinar por píxel o por objeto.

Iluminación en tiempo real:

Luces direccionales (sol / luna). Hasta 4 por escena.

Luces omnidireccionales.

Focos con ángulo de cono ajustable y atenuación.

Mapeo de sombras:

DirectionalLight: Orthogonal (fastest), PSSM 2-split and 4-split. Supports blending between splits.

OmniLight: Dual paraboloid (fast) or cubemap (slower but more accurate). Supports colored projector textures in the form of panoramas.

SpotLight: Textura única.

**\*\* Iluminación global con iluminación indirecta: \*\***

Lightmaps horneados (rápidos, pero no se pueden actualizar en tiempo de ejecución).

Mapas de luz en CPU.

GLS3: GI probes (slower, semi-real-time). Supports reflections.

Reflexiones:

GLS3: Voxel-based reflections (when using GI probes).

Reflexiones horneadas rápidas o reflexiones lentas en tiempo real usando ReflectionProbe.

Opcionalmente, se puede habilitar la corrección de paralaje.

GLS3: Screen-space reflections.

Las técnicas de reflexión se pueden combinar para mayor precisión.

Cielo:

Cielo panorámico (usando un HDRI).

Cielo procedural.

Niebla:

Niebla de profundidad con curva de atenuación ajustable.

Altura de niebla (suelo o techo) con atenuación regulable.

Soporte para color de niebla de profundidad automático dependiendo de la dirección de la cámara (para que coincida con el color del sol).

Transmitancia opcional para hacer las luces más visibles en la niebla.

Partículas:

GLS3: GPU-based particles with support for custom particle shaders.

Partículas basadas en la CPU.

Postprocesamiento:

Mapado de tonos (Linear, Reinhard, Filmic, ACES).

GLS3: Ajustes de exposición automáticos basados en el brillo de la ventana gráfica.

GLS3: Profundidad de campo cercana y lejana.

GLS3: Oclusión ambiental del espacio en pantalla.

Brillo/floración con escalado bicúbico opcional y varios modos de combinación disponibles: Pantalla,

Luz suave, Agregar, Reemplazar.

Corrección de color mediante rampa unidimensional.

Ajustes de brillo, contraste y saturación.

Filtrado de texturas:

Filtrado más cercano, bilineal, trilineal o anisotrópico.

Compresión de textura:

GLS3: BPTC para compresión de alta calidad (no compatible con macOS).

GLS3: ETC2 (no compatible con macOS).

ETC1 (recomendado cuando se utiliza el renderizador GLS2).

GLS3: S3TC (no compatible con plataformas móviles / web).

Shaders

\*2D: \* Vértices personalizados, fragmentos y sombreadores de luz.

Sombreadores basados en texto que utilizan un lenguaje de sombreado inspirado en GLSL

<doc\_shading\_language>.

Editor visual de shaders.  
Soporte para plugins de visual shader.

#### Scripting

##### General:

Patrón de diseño orientado a objetos con scripts que extienden los nodos.  
Señales y grupos para la comunicación entre scripts.  
Soporte para scripting en varios idiomas.  
Muchos tipos de datos de álgebra lineal 2D y 3D, como vectores y transformaciones.

##### GDScript:

Lenguaje interpretado de alto nivel con :ref: tipado estático opcional <doc\_gdscript\_static\_typing>.  
Sintaxis inspirada en Python.  
El resaltado de sintaxis se proporciona en GitHub.  
Use hilos para realizar acciones asincrónicas o hacer uso de múltiples núcleos de procesador.

##### C #:

Empaquetado en un binario separado para mantener bajos los tamaños de archivo y las dependencias.  
Utiliza Mono 6.x.  
Soporte completo para la sintaxis y características de C# 7.0.  
Soporta todas las plataformas.  
Se recomienda utilizar un editor externo para beneficiarse de la funcionalidad IDE.

##### VisualScript:

Lenguaje de scripting visual basado en gráficos.  
Funciona mejor cuando se usa para propósitos específicos (como lógica de nivel específico) en lugar de como un lenguaje para crear proyectos completos.  
GDNative (C, C ++, Rust, D, ...):  
Cuando lo necesite, enlace a bibliotecas nativas para un mayor rendimiento e integraciones de terceros.  
Para la lógica del juego de secuencias de comandos, se recomiendan GDScript o C # si su rendimiento es adecuado.  
Enlaces oficiales para C y C++.  
Utilice cualquier sistema de compilación y funciones de idioma que desee.  
Se mantuvieron los enlaces D, Kotlin, Python, Nim y Rust proporcionados por la comunidad.

#### Audio

Salida mono, estéreo, 5.1 y 7.1.  
Reproducción posicional y no posicional en 2D y 3D.  
Efecto doppler 2D y 3D opcional.  
Soporte para redireccionamiento audio buses y efectos con docenas de efectos incluidos.  
Nodo Listener3D para escuchar desde una posición diferente a la cámara en 3D.  
Entrada de audio para grabar micrófonos.  
Entrada MIDI.

##### APIs usadas:

Windows: WASAPI.  
macOS: CoreAudio.  
Linux: PulseAudio o ALSA.

## Importación

Soporte para plugins de importación personalizados.

Formatos:

### Imágenes:

BMP (.bmp) - No ofrece soporte para imágenes 16-bit per pixel. Sólo imágenes de 1-bit, 4-bit, 8-bit, 24-bit, y 32-bit per pixel son soportadas.

DirectDraw Surface (.dds) - Si hay mipmaps presente en la textura, serán cargados directamente. Esto puede ser usado para conseguir efectos usando mipmaps personalizados.

OpenEXR (.exr) - Soporta HDR (altamente recomendado para cielos panorámicos).

Radiance HDR (.hdr) - Soporta HDR (altamente recomendado para cielos panorámicos).

JPEG (.jpg, .jpeg) - No soporta transparencia debido a las limitaciones del formato.

PNG (.png) - La precisión está limitada a 8 bits por canal cuando se importa (no imágenes HDR).

Truevision Targa (.tga)

SVG (.svg, .svgz) - Los SVG son rasterizados usando NanoSVG cuando se los importa. El soporte es limitado, vectores complejos pueden no renderizarse correctamente. Para vectores complejos, renderizarlos a PNG usando Inkscape es normalmente una mejor solución. Esto puede ser automatizado gracias a su interfaz de línea de comandos.

WebP (.webp)

### Audio:

WAV con compresión opcional IMA-ADPCM.

Ogg Vorbis.

### Entrada

Sistema de mapeo de entrada que utiliza eventos de entrada codificados o acciones de entrada reprogramables.

Los valores del eje pueden ser asignados a dos acciones diferentes con una zona muerta configurable.

Usa el mismo código para soportar tanto los teclados como los gamepads.

Entradas del teclado.

Las teclas pueden ser mapeadas en modo "físico" para ser independientes de la disposición del teclado.

Entradas del ratón.

El cursor del ratón puede ser visible, oculto, capturado o confinado dentro de la ventana.

Cuando se capture, la entrada en bruto se utilizará en Windows y Linux para eludir la configuración de la aceleración del ratón del sistema operativo.

Entrada de gamepad (hasta 8 controladores simultáneos).

Entrada de lápiz/tableta con soporte de presión.

### Navegación

Algoritmo A estrella en 2D y 3D.

Mallas de navegación.

Generar mallas de navegación desde el editor.

### Redes

Redes TCP de bajo nivel utilizando StreamPeer y TCP\_Server.

Redes UDP de bajo nivel usando PacketPeer y UDPServer.

Solicitudes HTTP de bajo nivel usando HTTPClient.

Solicitudes HTTP de alto nivel mediante HTTPRequest.

Soporta HTTPS listo para ser usado utilizando los certificados empaquetados.  
API multijugador de alto nivel usando UDP y ENet.  
Replicación automática mediante llamadas a procedimiento remoto (RPC).  
Admite transferencias ordenadas, confiables y no confiables.  
Cliente y servidor WebSocket, disponible en todas las plataformas.  
Cliente y servidor WebRTC, disponible en todas las plataformas.  
Soporte para UPnP para eludir el requisito de reenviar puertos cuando se aloja un servidor detrás de una NAT.

#### Internacionalización

Soporte completo para Unicode, incluido emojis.  
Almacena las strings de localización usando CSV o gettext.  
Utiliza cadenas localizadas en su proyecto automáticamente en elementos GUI o utilizando la función `tr()`.

#### Integración de ventanas y sistemas operativos

Mover, redimensionar, minimizar y maximizar la ventana generada por el proyecto.  
Cambia el título y el icono de la ventana.  
Solicita atención (hará que la barra de título parpadee en la mayoría de las plataformas).  
Modo de pantalla completa.  
No utiliza en exclusiva la pantalla completa, por lo que la resolución de la pantalla no se puede cambiar de esta manera. Usa un Viewport con una resolución diferente en su lugar.  
Ventana sin bordes (a pantalla completa o no).  
Capacidad de mantener la ventana siempre en primer plano.  
Ventana transparente con transparencia por píxel.  
Integración del menú global en el macOS.  
Ejecutar los comandos de forma bloqueante o no bloqueante.  
Abrir rutas de archivos y URLs usando manejadores de protocolo predeterminados o personalizados (si están registrados en el sistema).  
Analizar los argumentos de la línea de comandos personalizada.

#### Móvil

Compras In-app en Android e iOS.  
Soporte para anuncios utilizando módulos de terceros.

#### Sistema GUI

El GUI de Godot está construido usando los mismos nodos de control que se usan para hacer juegos en Godot. La interfaz de usuario del editor puede ser fácilmente ampliada de muchas maneras usando complementos.

#### Nodos:

Botones.  
Casillas de chequeo, botones de chequeo, botones de selección.  
Entrada de texto usando LineEdit (una sola línea) y TextEdit (múltiples líneas).  
Menús desplegables usando PopupMenu y OptionButton.  
Barras de desplazamiento.  
Etiquetas.  
RichTextLabel para texto formateado usando BBCode.  
Árboles (también pueden utilizarse para representar tablas).



Contenedores (horizontal, vertical, cuadrícula, centro, margen, divisor arrastrable, ...).  
Los controles se pueden rotar y escalar.

Dimensionando:

Anclas para mantener los elementos de la interfaz gráfica de usuario en una esquina específica, en un borde o en el centro.

Contenedores para colocar elementos GUI automáticamente siguiendo ciertas reglas.

Diseños Stack.

Diseños Grid.

Margen y centrado planos.

Divisor arrastrable planos.

Escala a múltiples resoluciones usando los modos de estiramiento 2d o viewport.

Soporta cualquier relación de aspecto usando anclajes y el aspecto de estiramiento "expandir".

Temática:

Editor de temas integrado.

Generar un tema basado en la configuración actual del tema del editor.

Procedimiento basado en la temática vectorial usando StyleBoxFlat.

Soporta esquinas redondeadas/biseladas, sombras de caída y anchos por borde.

Temas basados en texturas usando StyleBoxTexture.

El pequeño tamaño de la distribución de Godot puede hacer que sea una alternativa adecuada a los marcos como Electrón o el Qt.

Animación

Cinemática directa y cinemática inversa.

Soporte para animar cualquier propiedad con interpolación personalizable.

Soporte a los métodos de llamada en las pistas de animación.

Soporte para reproducir sonidos en las pistas de animación.

Soporte para las curvas de Bézier en la animación.

Formatos

Las escenas y los recursos pueden ser guardados en basado en texto o formatos binarios.

Los formatos basados en texto son legibles para los humanos y más amigables para el control de versiones.

Los formatos binarios son más rápidos de guardar/cargar para grandes escenas/recursos.

Leer y escribir archivos de texto o binarios usando File.

Opcionalmente puede ser comprimido o encriptado.

Lee y escribe clase\_JSON archivos.

Lee y escribe archivos de configuración de estilo INI usando ConfigFile.

Puede (des)serializar cualquier tipo de datos Godot, incluyendo Vector, Color, ...

Lee los archivos XML usando XMLParser.

Empaqueta los datos del juego en un archivo PCK (formato personalizado optimizado para una búsqueda rápida), en un archivo ZIP o directamente en el ejecutable para su distribución en un solo archivo.

Exportar archivos PCK adicionales que pueden ser leídos por el motor para soportar mods y DLCs.

Misceláneas

Acceso de bajo nivel a los servidores que permite evitar la sobrecarga del árbol de la escena cuando sea necesario.

Interfaz de línea de comandos para la automatización.  
 Exportar y desplegar proyectos utilizando plataformas de integración continua.  
 Completado de scripts está disponible para Bash, zsh y fish.  
 Soporte para módulos C++ enlazados estáticamente al binario del motor.  
 Motor y editor escrito en C++03.  
 Puede ser compilado usando GCC, Clang y MSVC. MinGW también está soportado.  
 Amigable con los empaquetadores. En la mayoría de los casos, las bibliotecas de sistema pueden ser utilizadas en lugar de las proporcionadas por Godot. El sistema de construcción no descarga nada.  
 Las compilaciones pueden ser totalmente reproducibles.  
 Godot 4.0 será escrito en C++17.  
 Licenciado bajo la licencia permisiva del MIT.  
 Abre el proceso de desarrollo con contributions welcome.

## Política de lanzamiento de Godot

### Versionado de Godot

Godot usa la numeración de versión "major.minor.patch". Sin embargo, no sigue estrictamente el "Versionado Semántico" <<https://semver.org/>>\_. Esto significa que las versiones consideradas "semimenores" por ese estándar (como la 3.1 -> 3.2) muy probablemente introducirán cambios de última hora. Aún así, no habrá tantos cambios de ruptura como un salto de versión "semimayor" como la 3.2 -> 4.0.

En interés de la estabilidad y la utilidad, el lanzamiento de parches puede ocasionalmente introducir pequeños cambios de ruptura también. Al reempaquetar proyectos Godot (por ejemplo, en un Flatpak), asegúrese de utilizar siempre la misma versión de parche que la utilizada para exportar el proyecto inicialmente.

El primer lanzamiento en una serie de lanzamientos mayores o menores no termina con un cero final. Por ejemplo, el primer lanzamiento de la serie 3.2 es "3.2", no "3.2.0".

### Línea de tiempo del soporte de liberación

Las versiones de Godot se soportan durante un cierto tiempo. Mientras que estas duraciones no están grabadas en piedra, aquí hay una tabla con el nivel de apoyo esperado para cada versión de Godot:

Versión	**Fecha de publicación*	Nivel de soporte
Godot 4.0	~2021 (ver abajo)	"inestable" Enfoque actual del desarrollo (inestable).
Godot 3.2	Enero de 2020	"Soportado" Nuevas características compatibles con versiones anteriores (retroportadas desde la rama "master"), así como correcciones de errores, seguridad y soporte de la plataforma.
Godot 3.1	Marzo de 2019	[parcial] Sólo arrays críticos, de seguridad y de soporte de la plataforma.
Godot 3.0	Enero de 2019	[parcial] Sólo arrays críticos, de seguridad y de soporte de la plataforma.
Godot 2.1	Julio de 2016	[parcial] Sólo arrays críticos, de seguridad y de soporte de la plataforma.
Godot 2.0	Febrero de 2016	Sin soporte.
Godot 1.1	Mayo de 2015	Sin soporte.
Godot 1.0	Diciembre de 2014	Sin soporte.

Leyenda: |soporte| Soporte completo - |parcial| Soporte parcial - Sin soporte (fin de vida) - |inestable| Versión de desarrollo

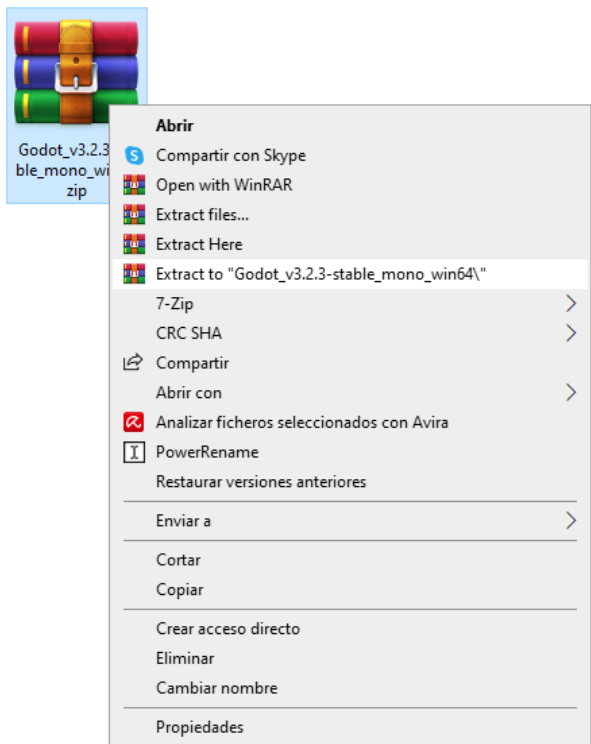
Las versiones de Godot anteriores al lanzamiento no están pensadas para ser utilizadas en la producción y se proporcionan con el mejor esfuerzo.

### Descargar Godot:

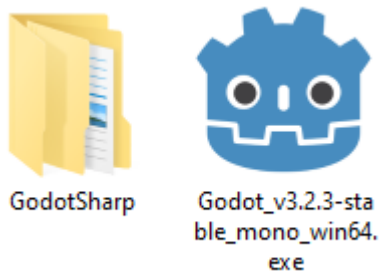
Para descargar Godot acceda desde el siguiente link, <https://godotengine.org/download/windows> y seleccione la opción correspondiente a su sistema (32 o 64 bits).

Si posee otro sistema operativo seleccione la pestaña correspondiente.

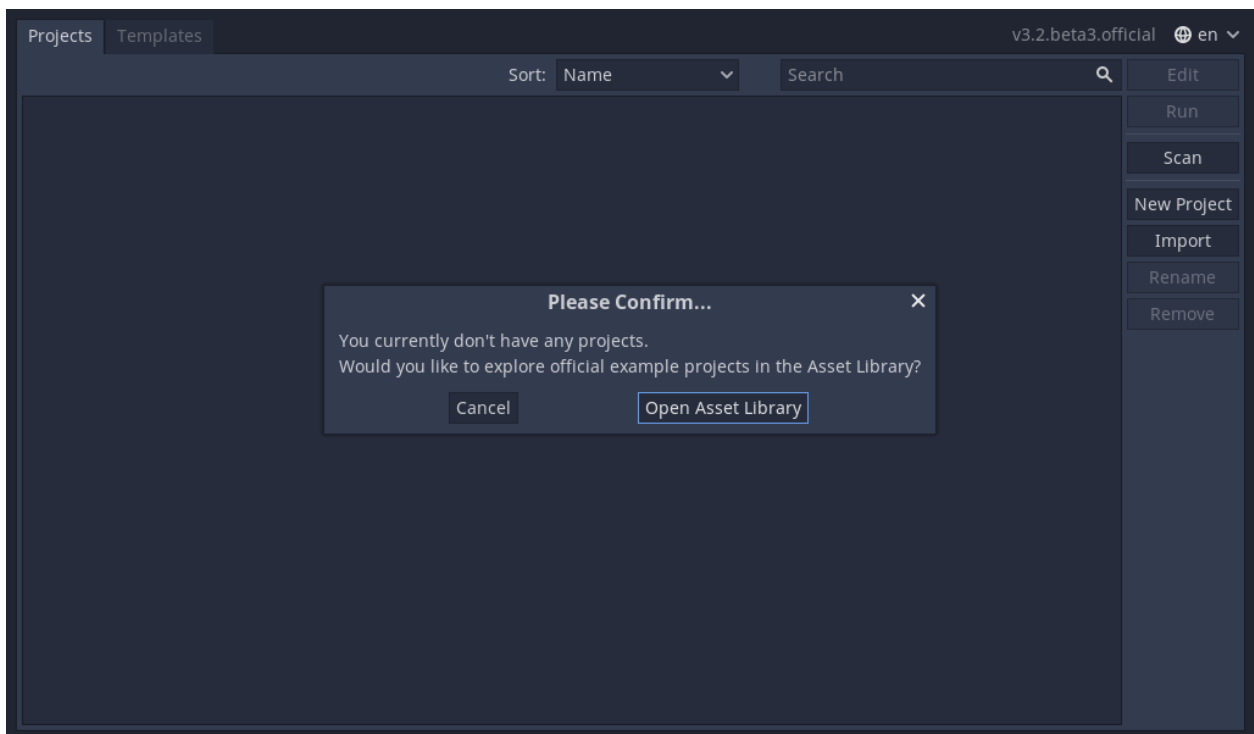
El siguiente paso será descomprimir el archivo correspondiente



Con el archivo extraído, abriremos el archivo ejecutable correspondiente a GODOT



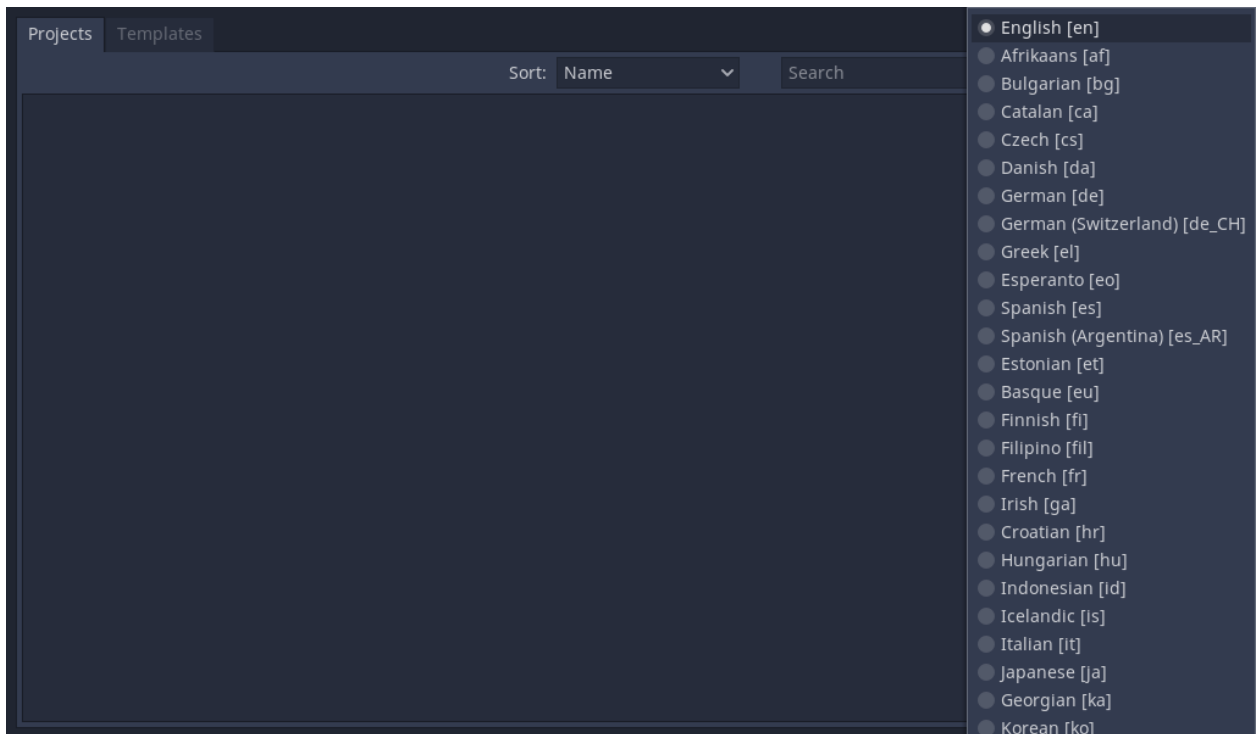
Una vez abierto el programa lo primero que veremos será el Administrador de Proyectos.



El mensaje de advertencia nos indica que no tenemos ningún proyecto creado. El programa nos sugiere, abrir la biblioteca de Assets. En este caso nosotros vamos a cancelar el cuadro de dialogo y ver como crear un proyecto nuevo.

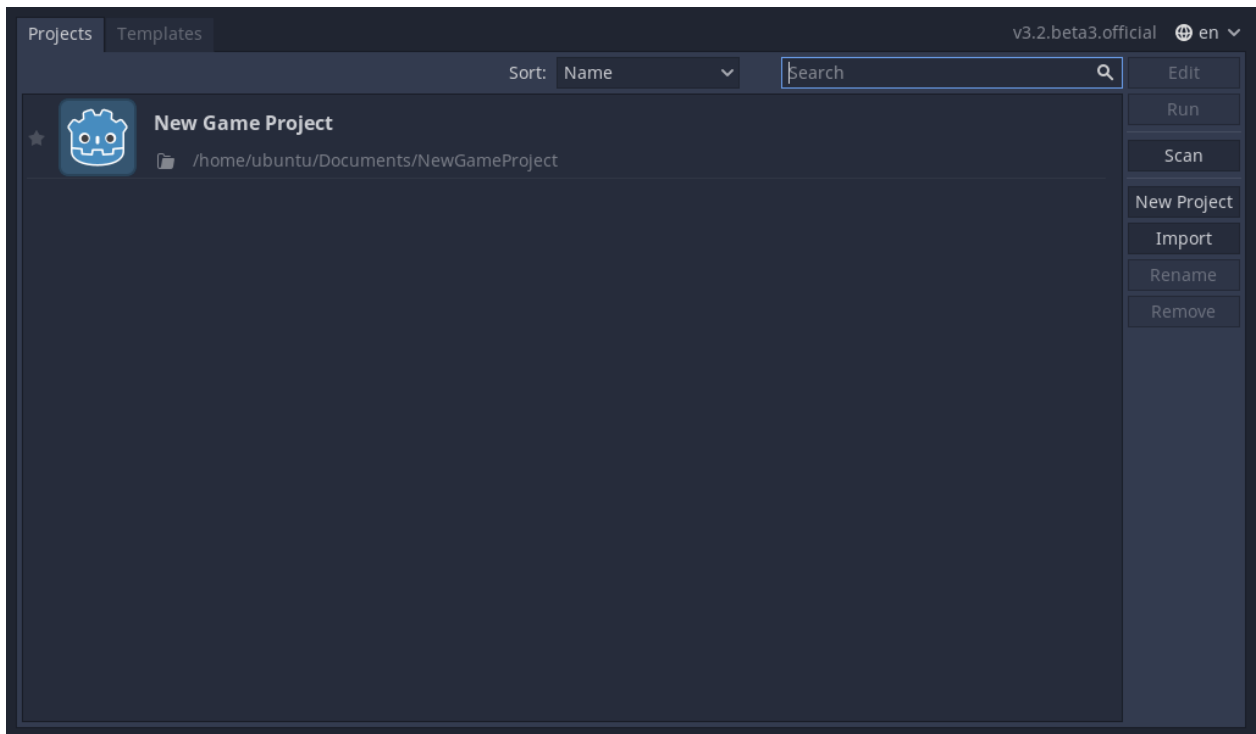
### 1.1.4 Creando un Nuevo Proyecto

En la esquina superior derecha encontrarás un menú desplegable que permite cambiar el idioma del editor.



Para crear un nuevo proyecto, la opción “Nuevo Proyecto” debe ser utilizada. Elije y crea una ruta para el proyecto y especifica el nombre del proyecto.

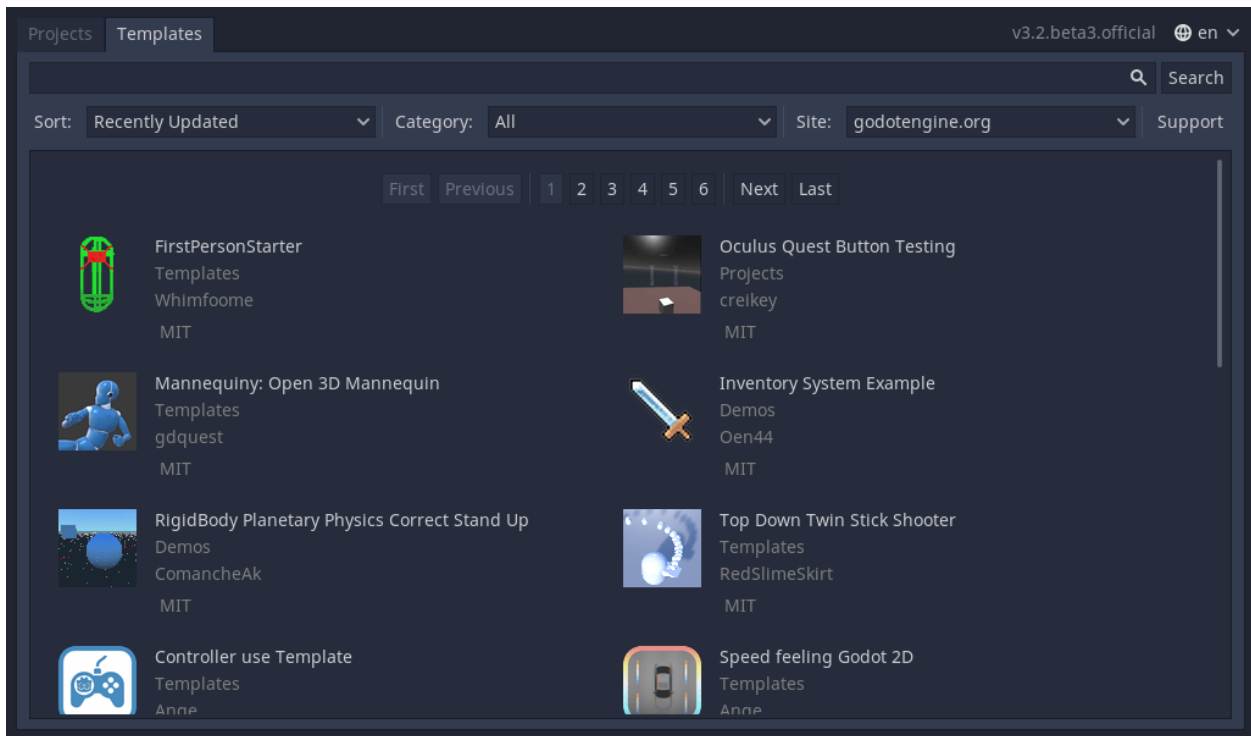




Antes de iniciar a trabajar con el proyecto, vamos a retomar la Biblioteca de Assets.

#### Biblioteca de Assets

En la pestaña Plantillas podrás descargar plantillas de código abierto y demos de proyectos en la Biblioteca de Assets que te ayudarán a comenzar rápidamente. Solo selecciona la plantilla o demo que prefieras, haz clic en descargar, una vez haya finalizado de descargar, haz clic en instalar y elige dónde quieres que este el proyecto.



Este conjunto de páginas tratará sobre cómo utilizar AssetLib (tanto desde Godot como desde el sitio web), cómo puedes enviar tus propios assets y cuáles son las pautas de envío.

Debe tenerse en cuenta que AssetLib es relativamente joven y que puede presentar varios inconvenientes, errores y problemas de usabilidad. Como con todos los proyectos de Godot, el repositorio de código está disponible en GitHub, donde puedes enviar solicitudes y problemas, así que por favor ¡no dudes en visitarlo!

### Tipos de recursos

Tenga en cuenta que hay, a grandes rasgos, dos tipos diferentes de activos que puedes publicar.

Los activos etiquetados como "Plantillas", "Proyectos" o "Demostraciones" aparecen bajo la pestaña "Plantillas" en el gestor de proyectos de Godot. Estos activos son proyectos Godot independientes que pueden funcionar por sí mismos.

Otros activos aparecen dentro del editor de Godot bajo la pestaña de la pantalla principal "AssetLib", junto a "2D", "3D" y "Script". Estos activos están pensados para ser descargados y colocados en un proyecto Godot existente.

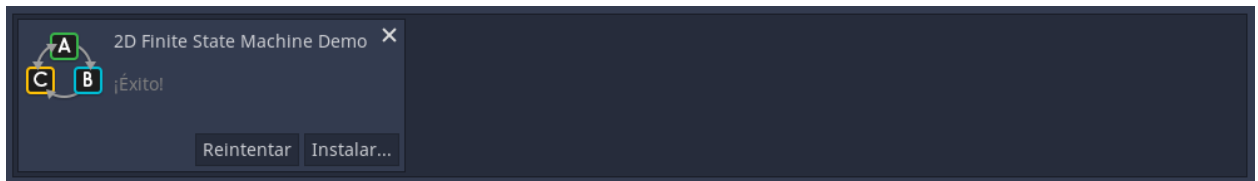
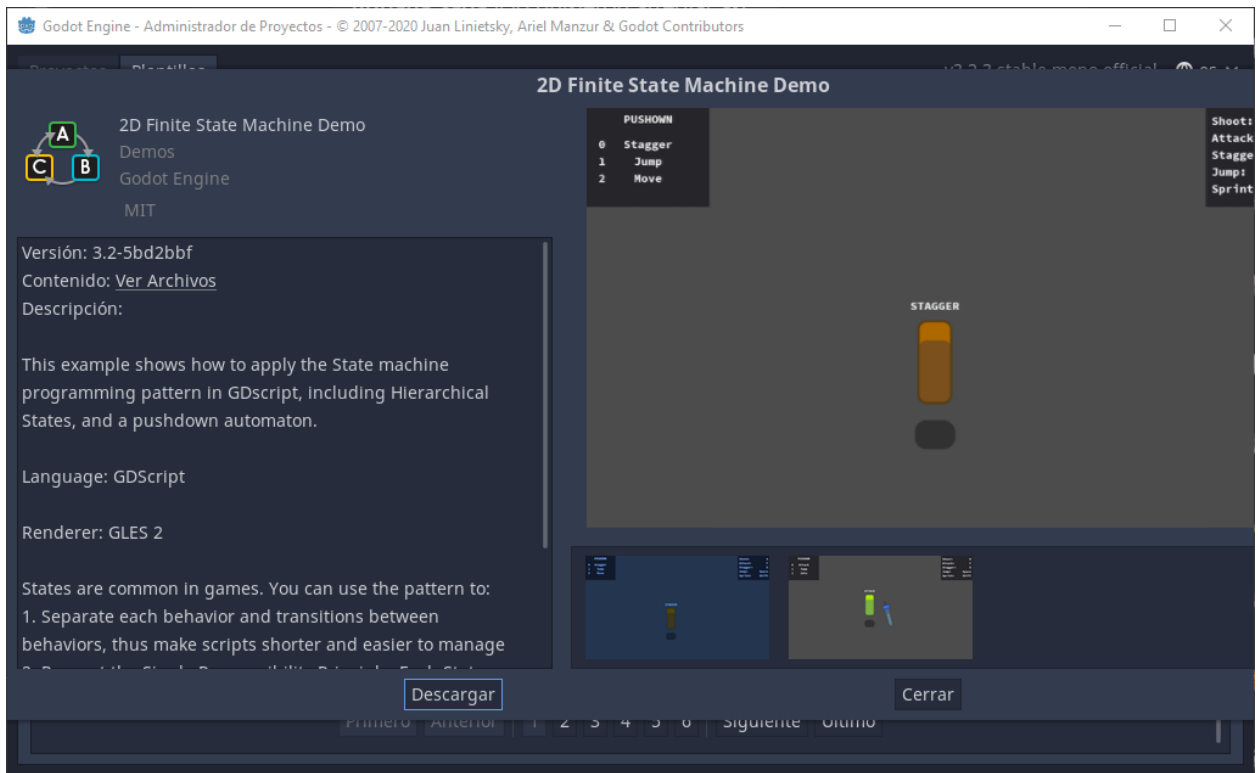
### Preguntas Frecuentes

¿Se pueden subir assets de pago a la biblioteca de assets?

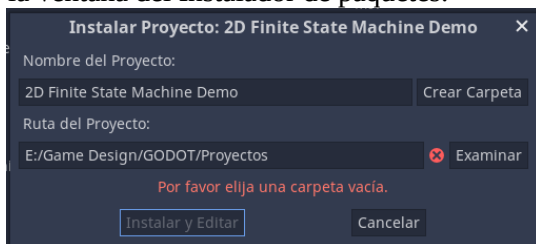
En la oficial no se puede, pero en el futuro podría existir otras bibliotecas de assets que lo permitan. Sin embargo, se puede monetizar y vender los assets de Godot fuera de la Asset Library.

Cuando haga clic en un recurso, verá más información sobre él.

Si haces clic en el botón Instalar, Godot buscará un archivo del activo y seguirá el progreso de la descarga en la parte inferior de la ventana del editor.



Cuando termine, puede proceder a instalarlo usando el botón de instalación. Esto hará que aparezca la ventana del Instalador de paquetes.







Una vez que haya terminado, puede pulsar el botón Instalar, que descomprimirá todos los archivos del archivo, e importará cualquier activo que contenga, como imágenes o modelos 3D. Una vez hecho esto, debería ver un mensaje que indica que la instalación del Paquete se ha completado.

## PLATAFORMAS

### Nueva escena

1. Nuevo nodo 2D -> renombrar "Level\_1"
2. Guardar Escena como: Level\_1.tscn

### Nueva escena

1. Nuevo Kinematicbody 2D

Godot ofrece cuatro tipos de cuerpos físicos, extendiendo CollisionObject2D:

Área2D

NOTA: Area2D será retomado más adelante.

Los otros tres cuerpos extienden PhysicsBody2D:

KinematicBody2D¶

Los cuerpos KinematicBody2D detectan las colisiones con otros cuerpos, pero no se ven afectados por propiedades físicas como la gravedad o la fricción. En su lugar, deben ser controlados por el usuario a través del código. El motor de física no moverá un cuerpo cinemático.

Cuando se mueve un cuerpo cinemático, no se debe establecer su posición directamente. En su lugar, debe utilizar los métodos `move_and_collide()` o `move_and_slide()`. Estos métodos mueven el cuerpo a lo largo de un vector dado, y se detendrá instantáneamente si se detecta una colisión con otro cuerpo. Después de que el cuerpo haya colisionado, cualquier respuesta de colisión debe ser codificada manualmente.

RigidBody2D

Este es el nodo que implementa la física 2D simulada. no se controla un RigidBody2D directamente. En su lugar, se le aplican fuerzas y el motor de física calcula el movimiento resultante, incluyendo las colisiones con otros cuerpos, y las respuestas a las colisiones, como el rebote, la rotación, etc.

Puedes modificar el comportamiento de un cuerpo rígido a través de propiedades como "Masa", "Fricción" o "Rebote", que se pueden establecer en el Inspector.

El comportamiento del cuerpo también se ve afectado por las propiedades del mundo, como se establece en la Configuración del Proyecto -> Física, o mediante la introducción de un Area2D que está anulando las propiedades globales de la física.

Cuando un cuerpo rígido está en reposo y no se ha movido durante un tiempo, entra en reposo. Un cuerpo dormido actúa como un cuerpo estático, y sus fuerzas no son calculadas por el motor de física. El cuerpo se despertará cuando se apliquen fuerzas, ya sea por una colisión o a través del código.

## StaticBody2D¶

Un cuerpo estático es aquel que no es movido por el motor de física. Participa en la detección de colisiones, pero no se mueve en respuesta a la colisión. Sin embargo, puede impartir movimiento o rotación a un cuerpo que colisiona como si se estuviera moviendo, utilizando sus propiedades `constant_linear_velocity` (velocidad lineal constante) y `constant_angular_velocity` (velocidad angular constante).

Los nodos StaticBody2D se utilizan con mayor frecuencia para objetos que forman parte del entorno o que no necesitan tener ningún comportamiento dinámico.

Ejemplos de uso de StaticBody2D:

Plataformas

Cintas transportadoras

Paredes y otros obstáculos

## Modos de cuerpos rígidos¶

Los StaticBody2D y los RigidBody2D pueden ser configurados para utilizar un material de física. Esto permite ajustar la fricción y el rebote de un objeto, y establecer si es absorbente y/o rugoso.

## Physic Material¶

Rígido - El cuerpo se comporta como un objeto físico. Colisiona con otros cuerpos y responde a las fuerzas que se le aplican. Este es el modo por defecto.

Estático - El cuerpo se comporta como un StaticBody2D y no se mueve.

Carácter - Similar al modo "Rígido", pero el cuerpo no puede rotar.

Kinematic - El cuerpo se comporta como un KinematicBody2D y debe ser movido por código.

## Uso de RigidBody2D¶

Si necesita alterar cualquiera de las propiedades relacionadas con la física, deberá utilizar la llamada de retorno `_integrate_forces()` en lugar de `_physics_process()`. En este callback, tiene acceso al `Physics2DDirectBodyState` del cuerpo, que permite cambiar las propiedades de forma segura y sincronizarlas con el motor de física.

- renombrar nodo "nombre del personaje"

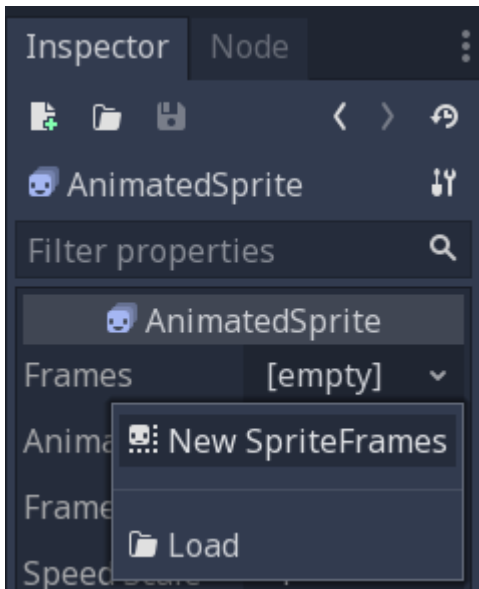


Típicamente, cuando crea o descarga un personaje animado, vendrá en una de dos formas: como imágenes individuales o como una sola hoja de sprite que contiene todos los cuadros de la animación. Ambos pueden ser animados en Godot con la clase `AnimatedSprite`.

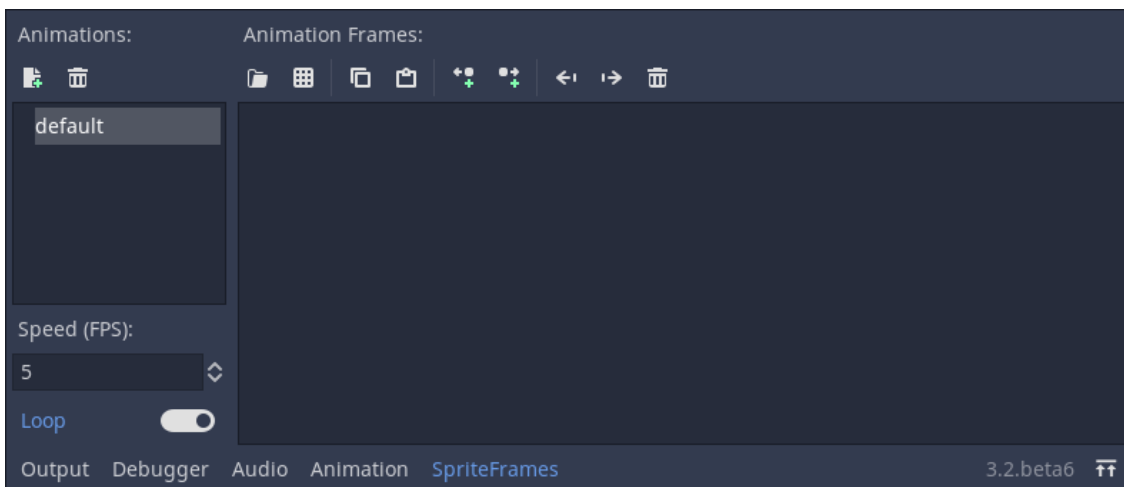
El nodo raíz puede ser Area2D (generalmente utilizado para animar elementos interactivos del escenario como monedas u otros coleccionables) o RigidBody2D. La animación seguirá haciéndose de la misma manera. Una vez completada la animación, puede asignar una forma al CollisionShape2D.

### 3. Nuevo nodo hijo Animatedsprite

En su propiedad SpriteFrames, seleccione "New SpriteFrames".



clic en el nuevo recurso SpriteFrames y verás que aparece un nuevo panel en la parte inferior de la ventana del editor:



Desde el Explorador de archivos en el lado izquierdo, arrastre las imágenes individuales a la parte central del panel SpriteFrames. En el lado izquierdo, es importante cambiar el nombre de la animación de "default" al nombre de su elección.



De vuelta en el Inspector, marque la casilla de la propiedad Play. Ahora debería ver la animación reproduciéndose en la ventana gráfica, cambie el ajuste de Velocidad (FPS, frames por segundo) en el panel SpriteFrames a su preferencia.

Puedes añadir animaciones adicionales haciendo clic en el botón "Nueva animación" y añadiendo imágenes adicionales.

<https://www.kenney.nl/assets/simplified-platformer-pack>

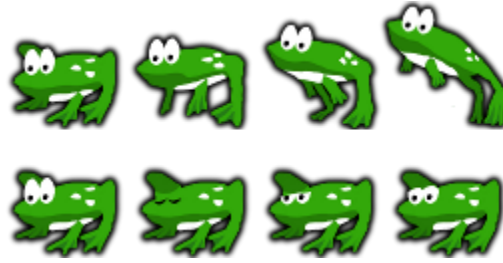


### Practica

1. Frames Nuevo SpriteFrames
2. Renombrar default x Idle
3. Arrastrar "Idle.png"
4. Nueva animación X 5
5. Inspector → Animation: Definir Idle como principal
6. Frames Guardar "Personaje\_Spriteframes.tres (res/recursos/)"

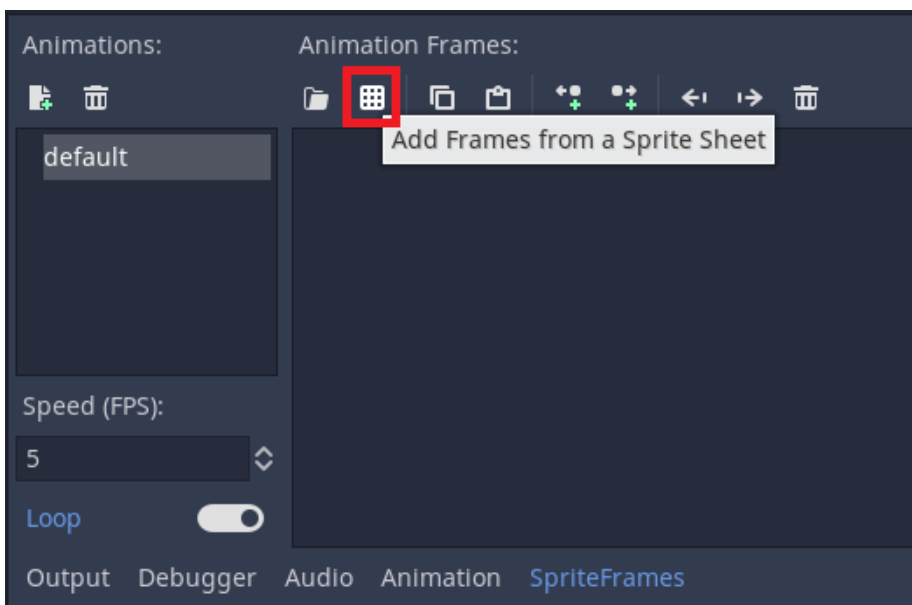
Hoja de Sprite con AnimatedSprite¶

Utilizaremos esta hoja de sprites de dominio público:



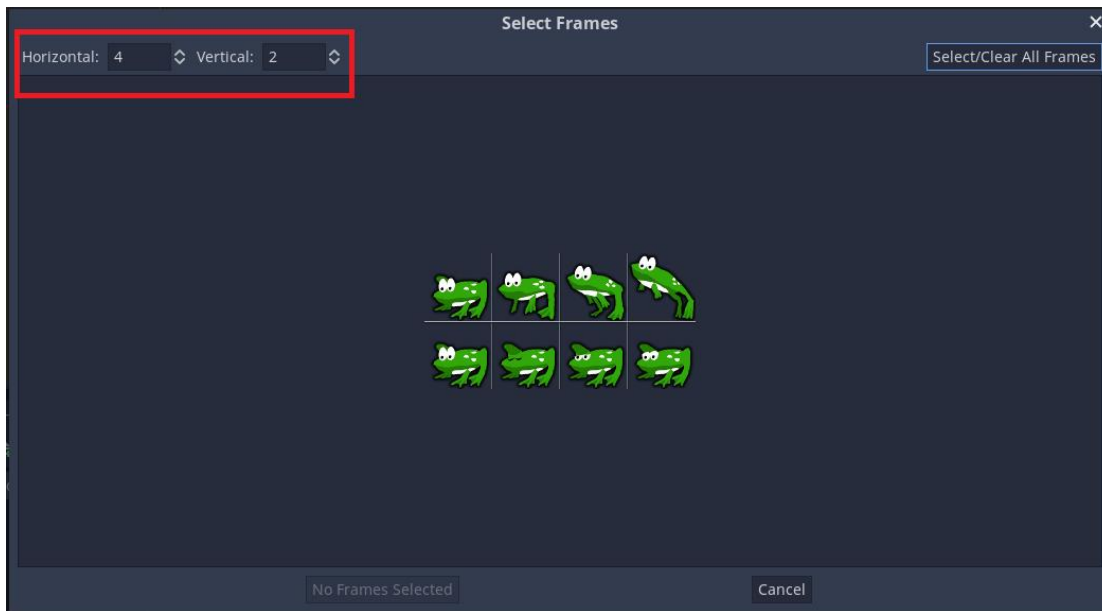
Configure su árbol de escena de la misma manera que lo hizo anteriormente al utilizar imágenes individuales. Seleccione el AnimatedSprite y en su propiedad SpriteFrames, seleccione "New SpriteFrames".

Haz clic en el nuevo recurso SpriteFrames. Esta vez, cuando aparezca el panel inferior, selecciona "Add frames from a Sprite Sheet".

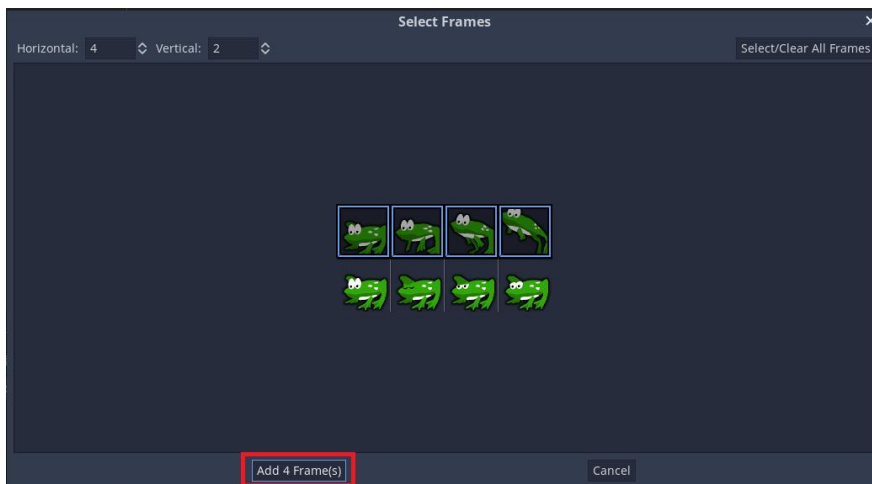


Se le pedirá que abra un archivo. Seleccione su hoja de sprites.

Se abrirá una nueva ventana con su hoja de sprites. Lo primero que tendrá que hacer es cambiar el número de imágenes verticales y horizontales en su hoja de sprites. En esta hoja de sprites, tenemos cuatro imágenes horizontales y dos imágenes verticales.



A continuación, seleccione los fotogramas de la hoja de sprites que quiera incluir en su animación. Seleccionaremos los cuatro primeros y haremos clic en "Añadir 4 fotogramas" para crear la animación.




Ahora verá su animación bajo la lista de animaciones en el panel inferior. Haga doble clic en default para cambiar el nombre de la animación a jump.



Estos ejemplos ilustran las dos clases que puedes utilizar en Godot para la animación 2D. AnimationPlayer es un poco más compleja que AnimatedSprite, pero proporciona una funcionalidad adicional, ya que también puede animar otras propiedades como la posición o la escala. La clase AnimationPlayer también puede ser utilizada con un AnimatedSprite. Experimente para ver qué funciona mejor para sus necesidades.

4. Nuevo nodo hijo CollisionShape2D
5. Shape Capsule ajustar alineado con los pies

Un cuerpo físico puede contener cualquier número de objetos Shape2D como hijos. Estas formas se utilizan para definir los límites de colisión del objeto y para detectar el contacto con otros objetos. Para detectar colisiones, al menos un Shape2D debe ser asignado al objeto. La forma más común de asignar una forma es añadiendo un CollisionShape2D o CollisionPolygon2D como hijo del objeto.

6. Kinematicbody (hijos no seleccionados) 
7. Escena Player "Add script"

```
Player.gd

1 extends KinematicBody2D
2
3 var velocity = Vector2(0,0)
4 const SPEED = 180
5 const GRAVITY = 30
6 const JUMPFORCE = -900
7
8 func _physics_process(_delta):
9     if Input.is_action_pressed("Derecha"):
10         velocity.x = SPEED
11         $AnimatedSprite.play("Caminar")
12         $AnimatedSprite.flip_h = false
13     elif Input.is_action_pressed("Izquierda"):
14         velocity.x = -SPEED
15         $AnimatedSprite.play("Caminar")
16         $AnimatedSprite.flip_h = true
17     else:
18         $AnimatedSprite.play("Idle")
19
20     if not is_on_floor():
21         $AnimatedSprite.play("En_Aire")
22
23     velocity.y = velocity.y + GRAVITY
24
25     if Input.is_action_just_pressed("Salto") and is_on_floor():
26         velocity.y = JUMPFORCE
27
28     velocity = move_and_slide(velocity,Vector2.UP)
29
30     velocity.x = lerp(velocity.x,0,0.3)
```

---

Añadir player como escena instanciada

---

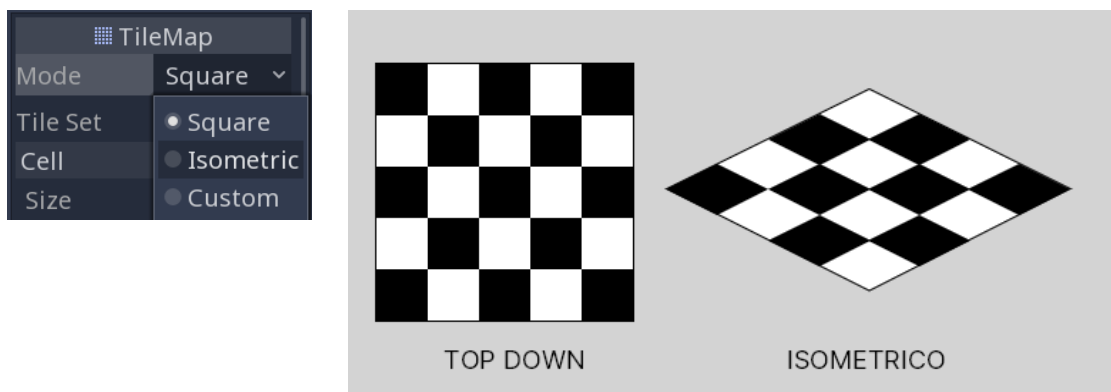
## TILEMAPS

Un tilemap es una rejilla de azulejos utilizada para crear el diseño de un juego.

- permiten dibujar el diseño en una cuadrícula.
- Al estar optimizados para dibujar un gran número de fichas permiten niveles mucho más grandes.
- Se puede añadir formas de colisión, oclusión y navegación a los tiles, añadiendo funcionalidad adicional al TileMap.

Por defecto, un TileMap utiliza una cuadrícula de azulejos.

También puede utilizar un modo "Isométrico" basado en la perspectiva o definir su propia forma de baldosa personalizada.



Bajo la sección "Cell" en el Inspector hay muchas propiedades que puedes ajustar para personalizar el comportamiento de tu tilemap:



**Cell Size (Tamaño de la celda)** Define el tamaño de la cuadrícula. Debe coincidir con el tamaño de los píxeles de tus mosaicos.

**Quadrant size (Cuadrante)** Define el tamaño de los trozos utilizados para el dibujo por lotes. Esto puede afectar negativamente al rendimiento. No lo cambie a menos que sepa lo que está haciendo.

**Custom Transform (Transformación personalizada)** Se utiliza para alterar la forma del mosaico. Utilícelo si tiene mosaicos no cuadrados.

**Half Offset and Tile Origin (Medio Offset y origen de las baldosas)** Estas propiedades afectan a la posición del mosaico en relación con la posición de la cuadrícula.

**YSort (Ordenamiento Y)** Hace que los mosaicos se dibujen en orden de su posición Y, de modo que los mosaicos "más bajos" se dibujen encima de los "más altos".

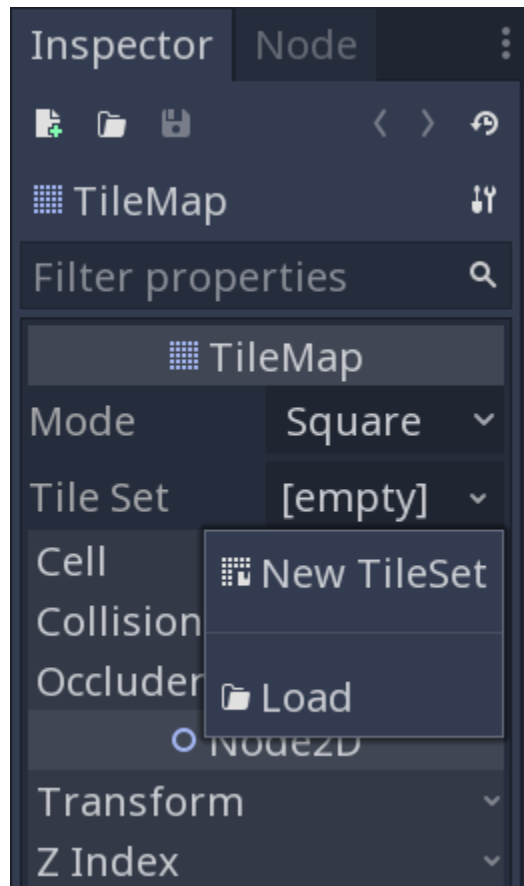


## Creando un TileSet

Una vez configurado el mapa de azulejos, es el momento de añadir un TileSet. Un TileSet es un recurso que contiene los datos sobre tus mosaicos - sus texturas, formas de colisión y otras propiedades. Cuando el juego se ejecuta, el TileMap combina los azulejos individuales en un solo objeto.

Para añadir un nuevo TileSet, haz clic en la propiedad "Tile Set" y selecciona "New TileSet".

Haz clic en la propiedad TileSet, y el panel "TileSet" se abrirá en la parte inferior de la ventana del editor:



En primer lugar, tienes que añadir la(s) textura(s) que utilizarás para las baldosas.

A continuación, clic en "New Single Tile" y arrastre la imagen para seleccionar el mosaico que quiera. Haz clic en el botón "Enable Snap" para facilitar la selección de todo el mosaico. Aparecerá un rectángulo amarillo alrededor del mosaico seleccionado.



## Formas de colisión¶

Si estás haciendo un mapa que necesita colisiones -paredes, suelo u otros obstáculos, por ejemplo- tendrás que añadir formas de colisión a cualquier baldosa que quieras que se considere "sólida".

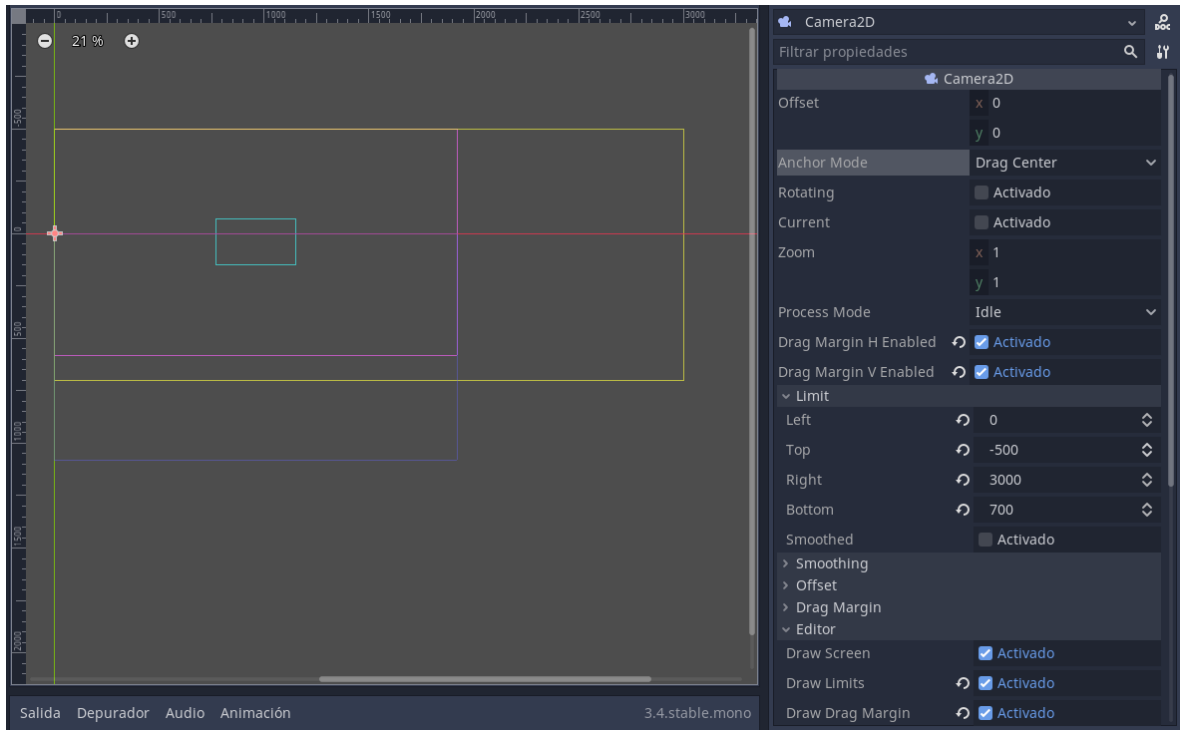
Haz clic en "TileSet" en la parte inferior de la ventana del editor para volver a la herramienta de tileset. Haz clic en el mosaico que has definido previamente (señalado en amarillo). Selecciona la pestaña "Colisión" y haz clic en el botón "Crear un nuevo rectángulo". Asegúrate de que sigues teniendo activada la función de ajuste a la rejilla y, a continuación, haz clic y arrastra el azulejo. Aparecerá una forma de colisión cuadrada en azul claro

## Practica

1. Crear nodo hijo **Tilemap**
2. Importar tiles
3. En el inspector Tileset – Nuevo Tileset
4. Renombrar como "Tile\_Solid"
5. Presionar en Tileset para abrir el Editor de Tileset.
6. Agregar Tiles
7. Tile 1 a Tile 1
8. + Nuevo Tile Individual
9. Region – Arrastrar sobre el tile
10. Colision
11. (Nuevo rectángulo) Arrastrar sobre el tile
12. (Nuevo polígono) Formas irregulares – Modificar los Steps para mayor precisión sobre la forma
13. Guardar Tilemap como recurso
14. Tilemap – Inspector – tileset (guardar) Tileworld.tres
15. Pintar tilemaps sobre el Viewport
16. De ser necesario invertir el orden del player y el Tilemap

## CAMARA2D

1. Añadir nodo hijo Camera2D
2. Current (Camara activa)
3. Hacer hijo del Player
4. Transform – Position (0,0)
5. Activar Draw Margin H enabled
6. Activar Draw Margin V enabled
7. Activar Draw Screen
8. Draw Limits
9. Draw Drag Margin



	Drag Margin	Draw Limits
Left:	0.3	0
Top:	0.2	-500
Right:	0.3	3000
Bottom:	0.2	700

## PARALLAX BACKGROUND

Añadir nodo hijo a la escena Level\_1 -> Parallax Background

Añadir nodo hijo al nodo Parallax Background -> Parallax Layer

Añadir nodo hijo al nodo Parallax Layer -> Sprite

Sprite – Escalar X 2

Y 2

Desactivar Offset – Centered

Parallax Layer – Mirroring                      640 x 2 (o doble del tamaño del sprite)

480 x 2 (o doble del tamaño del sprite)

Parallax Layer – Scale (valor entre 0 y 1)

Repetir las veces necesarias por cada elemento del fondo.

### **FALLZONE (SEÑALES)**

1. Añadir nodo hijo a la escena Level\_1 → Area2D

### **AREA2D**

Los nodos de área proporcionan detección e influencia. Pueden detectar cuando los objetos se superponen y emiten señales cuando los cuerpos entran o salen. Un Area2D también puede utilizarse para anular las propiedades físicas, como la gravedad o la amortiguación, en un área definida.

Hay tres usos principales para Area2D:

Anular los parámetros de física (como la gravedad) en una región determinada.

Detectar cuando otros cuerpos entran o salen de una región o qué cuerpos están actualmente en una región.

Comprobar la superposición de otras áreas.

Por defecto, las áreas también reciben la entrada del ratón y de la pantalla táctil.

2. Renombrar a “Fallzone”
3. Añadir nodo hijo al nodo Area 2d – Colission Shape2D
4. Colocar nodo por debajo de las plataformas (importante que no haya contacto)

### **Señales¶**

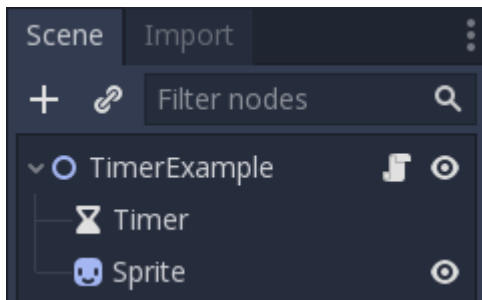
Las señales son la versión de Godot del patrón observador ([https://es.wikipedia.org/wiki/Observer\\_\(patrón\\_de\\_diseño\)](https://es.wikipedia.org/wiki/Observer_(patrón_de_diseño))). Permiten a un nodo enviar un mensaje que otros nodos pueden escuchar y responder.

Las señales son una forma de desacoplar los objetos del juego, lo que conduce a un código mejor organizado y más manejable. En lugar de forzar a los objetos del juego a esperar que otros objetos estén siempre presentes, pueden emitir señales a las que todos los objetos interesados pueden suscribirse y responder. Por ejemplo, en lugar de comprobar continuamente un botón para ver si está siendo presionado, el botón puede emitir una señal cuando está presionado.

### **Ejemplo de temporizador¶**

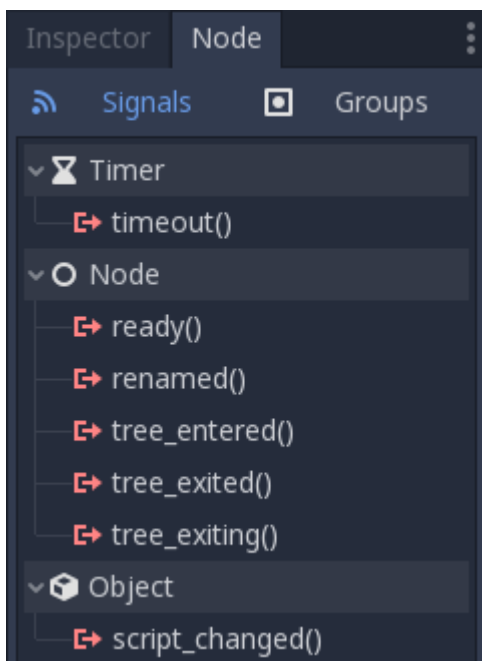
Crea una nueva escena con un Node2D y dos hijos: un Timer y un Sprite. Renombra el Nodo2D a TimerExample.

Para la textura del Sprite, puedes usar el icono de Godot. Adjunta un script al nodo raíz, pero no le añadas ningún código todavía.

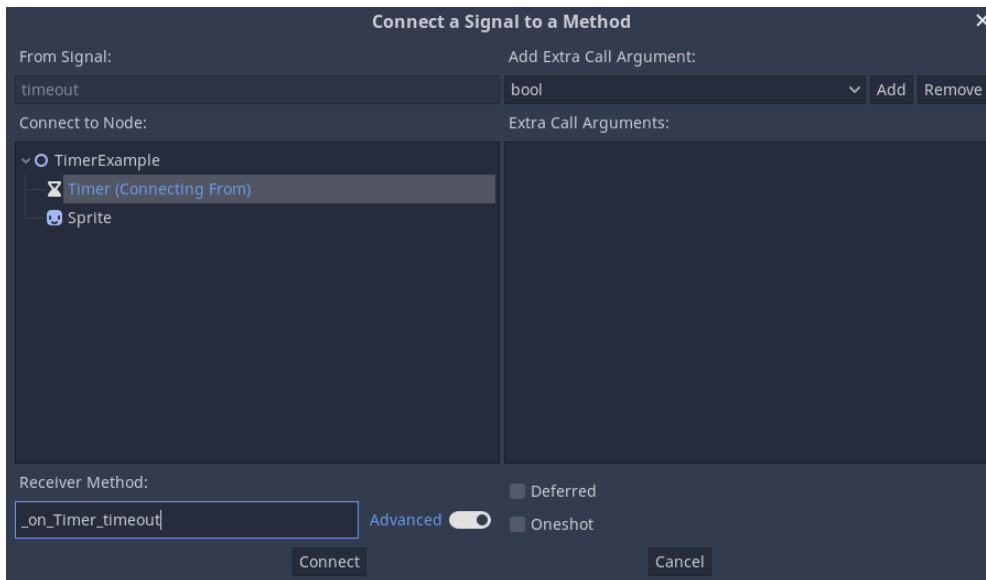


En las propiedades del nodo Temporizador, marque la casilla "On" junto a Autostart. Esto hará que el temporizador se inicie automáticamente cuando ejecute la escena. Puede dejar el tiempo de espera en 1 segundo.

Junto a la pestaña "Inspector" hay una pestaña llamada "Nodo". Haz clic en esta pestaña y verás todas las señales que puede emitir el nodo seleccionado. En el caso del nodo Timer, la que nos interesa es "timeout". Esta señal se emite cada vez que el Temporizador llega a 0.



Haga clic en la señal "timeout()" y pulsa "Conectar..." en la parte inferior del panel de señales. Verá la siguiente ventana, donde puedes definir cómo quieres conectar la señal:



En la parte izquierda, verá los nodos de su escena y podrá seleccionar el nodo que quiere "escuchar" la señal. Observe que el nodo Timer es azul, esto es una indicación visual de que es el nodo que está emitiendo la señal. Seleccione el nodo raíz.

El nodo destino debe tener un script adjunto o recibirá un mensaje de error.

En la parte inferior de la ventana hay un campo denominado "Método receptor". Este es el nombre de la función en el script del nodo de destino que desea utilizar. Por defecto, Godot creará esta función utilizando la convención de nomenclatura `_on_<nombre_del_nodo>_<nombre_de_la_señal>` pero puede cambiarla si lo desea.

Haga clic en "Conectar" y verá que la función se ha creado en el script:

**Extends** Node2D

```
func _on_Timer_timeout():
    $Sprite.visible = !$Sprite.visible
```

Ejecute la escena y verá que el Sprite parpadea cada segundo. Puedes cambiar la propiedad Wait Time del Timer para alterar esto.

## Conectando señales en código¶

También puedes hacer la conexión de señales en código en lugar de hacerlo con el editor. Esto suele ser necesario cuando se instancian nodos a través de código y no se puede utilizar el editor para realizar la conexión.

Recuerde desconectar la señal del temporizador para evitar errores al implementar esta variante.

Para realizar la conexión en código, podemos utilizar la función `connect`. La pondremos en `_ready()` para que la conexión se realice al ejecutarse. La sintaxis de la función es `<nodo_fuente>.connect(<nombre_señal>, <nodo_destino>, <nombre_función_destino>)`.

Conexión con Timer:

```

extends Node2D
func _ready():
    $Timer.connect("timeout", self, "_on_Timer_timeout")
func _on_Timer_timeout():
    $Sprite.visible = !$Sprite.visible

```

### Señales personalizadas¶

También puede declarar sus propias señales personalizadas en Godot:

```

extends Node2D

```

```

signal my_signal

```

Una vez declaradas, sus señales personalizadas aparecerán en el Inspector y podrán conectarse de la misma manera que las señales incorporadas en un nodo.

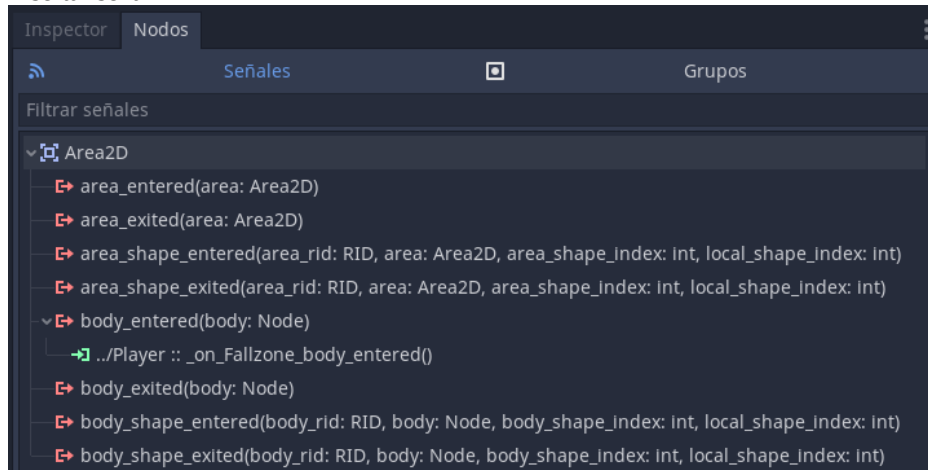
Para emitir una señal mediante código, utilice la función `emit_signal`:

```

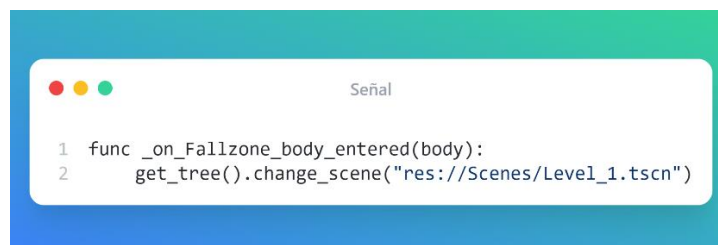
extends Node2D
signal my_signal
func _ready():
    emit_signal("my_signal")

```

#### 5. Insertar señal



6. Fallzone – Nodos – Señales
7. Area2D – body\_entered(body:Node)
8. Conectar Señal a un método – Conectar al Script (Personaje)





## COINS

1. Crear nuevo nodo Area2D
2. Renombrar a “Coin”
3. Guardar Escena
4. Añadir nodo hijo al nodo Area 2d – Colission Shape2D
5. Añadir nodo hijo al nodo Area 2d – AnimatedSprite
6. Nuevo Sprite frames
7. Añadir script a Coin
8. Agregar Signal a Coin Scene (no a la instancia)
9. Instanciar Coin Scene

## CAPAS Y MASCARAS

### Capas de colisión y máscaras¶

Una de las funciones de colisión más potentes, pero a menudo incomprendida, es el sistema de capas de colisión. Este sistema te permite construir complejas interacciones entre una variedad de objetos. Los conceptos clave son las capas y las máscaras. Cada `CollisionObject2D` tiene 20 capas físicas diferentes con las que puede interactuar.

#### Capa de colisión

Describe las capas en las que aparece el objeto. Por defecto, todos los cuerpos están en la capa 1.

#### Máscara de colisión

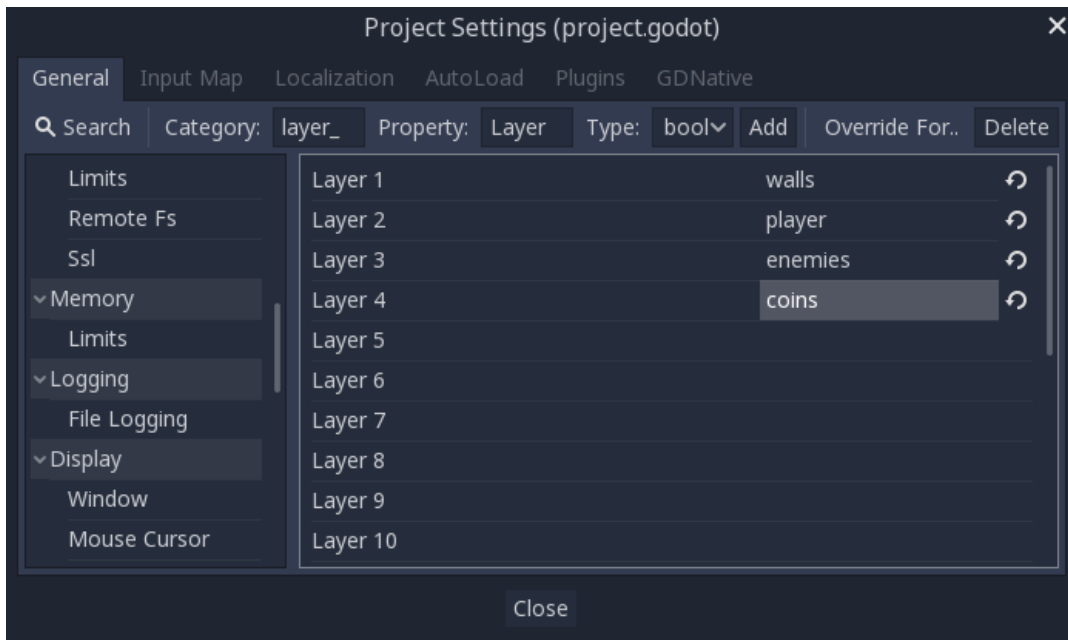
Describe las capas que el cuerpo explorará en busca de colisiones. Si un objeto no está en una de las capas de la máscara, el cuerpo lo ignorará. Por defecto, todos los cuerpos escanean la capa 1.

Estas propiedades pueden ser configuradas a través del código, o editándolas en el Inspector.

Mantener la pista de lo que está utilizando cada capa puede ser difícil, por lo que puede encontrar útil asignar nombres a las capas que está utilizando.

Los nombres pueden ser asignados en la Configuración del Proyecto -> Nombres de Capas.

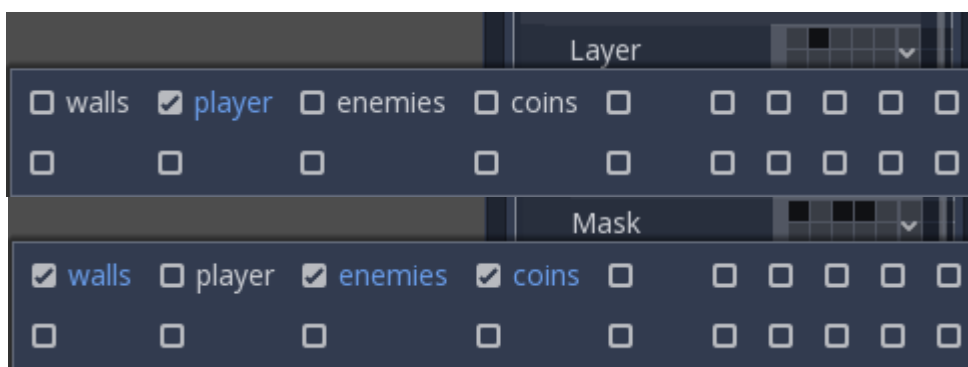




Ejemplo de GUI¶

Tienes cuatro tipos de nodos en tu juego: Muros, Jugador, Enemigo y Moneda. Tanto el jugador como el enemigo deben colisionar con los muros. El nodo Jugador debe detectar las colisiones con el Enemigo y la Moneda, pero el Enemigo y la Moneda deben ignorarse mutuamente.


Comience nombrando las capas "jugador", "monedas", etc. Coloque cada tipo de nodo en su respectiva capa utilizando la propiedad "Layer". Luego establece la propiedad "Mask" de cada nodo seleccionando las capas con las que debe interactuar. Por ejemplo, la configuración del Jugador se vería así:



### Practica

1. Proyecto – Configuración del proyecto – Layer Names
2. 2D Physics
3. Renombrar Capa 1 a Player
4. Renombrar Capa 2 a Platform
5. Renombrar Capa 3 a Fallzone
6. Renombrar Capa 4 a Item
7. Renombrar Capa 5 a Enemy
8. Mover "Coin" scene a capa Item
9. Editar mascarar de colision

## COIN BOUNCE (AnimationPlayer)

El nodo AnimationPlayer  tiene funcionalidades que permiten un amplio rango de posibilidades simples y complejas.

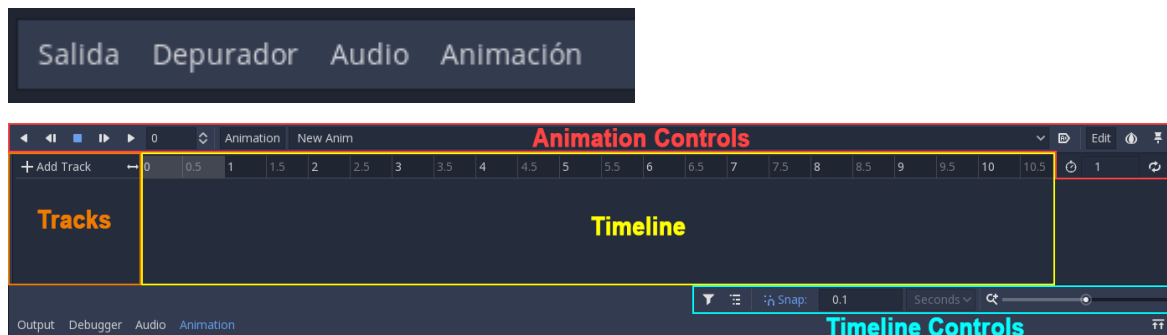
- Crear una animación simple
- Animar cualquier propiedad de cualquier nodo
- Llamar a funciones con las Pistas de Llamada a Función

En Godot, se puede animar cualquier cosa disponible en el Inspector, como transformaciones de nodos, sprites, elementos de UI, partículas, visibilidad y color de materiales, etc. También puedes modificar los valores de las variables del script y llamar a cualquier función.

Para utilizar las herramientas de animación primero tenemos que crear un nodo AnimationPlayer.

El tipo de nodo AnimationPlayer es el contenedor de datos para sus animaciones. Un nodo AnimationPlayer puede contener múltiples animaciones, que pueden hacer una transición automática entre ellas.

Después de crear uno, haga clic en el nodo AnimationPlayer en la pestaña Nodo para abrir el Panel de Animación en la parte inferior de la ventana gráfica.



Controles de animación (es decir, añadir, cargar, guardar y eliminar animaciones)

listado de pistas

La línea de tiempo con fotogramas clave

Los controles de la línea de tiempo y de las pistas, donde se puede hacer zoom en la línea de tiempo y editar las pistas, por ejemplo.

La animación se basa en los fotogramas clave¶

Un fotograma clave define el valor de una propiedad en un momento determinado.





Las formas de diamante representan los fotogramas clave en la línea de tiempo. Una línea entre dos fotogramas clave indica que el valor no ha cambiado.



El motor interpola los valores entre los fotogramas clave, lo que resulta en un cambio gradual de los valores en el tiempo.

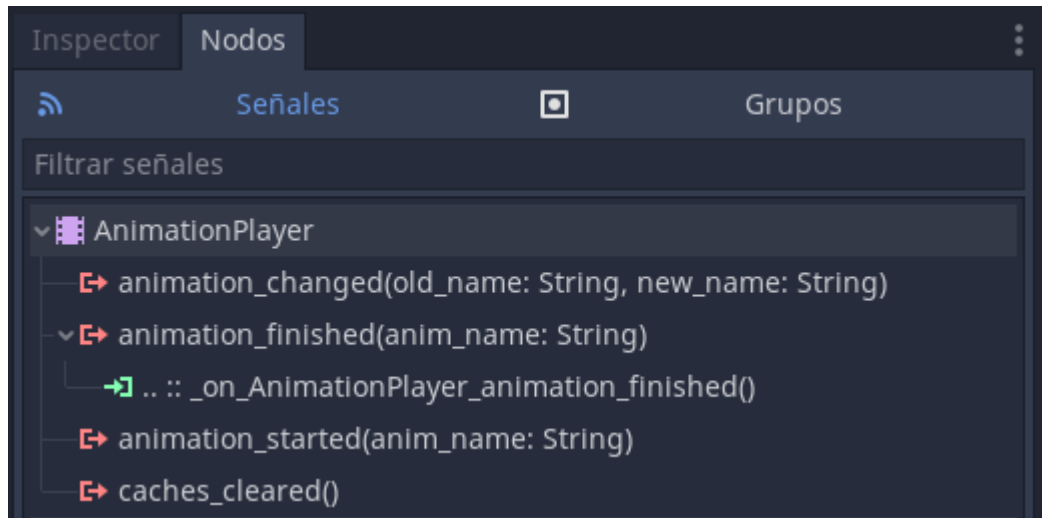
### Practica

#### Escena Coin

1. Añadir nodo hijo al nodo Area 2d – AnimationPlayer
2. Animacion – Animacion – Nuevo (“Bounce”)
3. Seleccionar nodo AnimatedSprite  y seleccionar animación.
4. ¿Crear nueva pista para property position e insertar clave? (SI)
5. Crear 1º keyframe pos.y = 0  en 0 segundos.
6. Crear 2º keyframe pos.y = -150  en 0.5 segundos.
7. Crear 3º keyframe pos.y = -60  en 1 segundos.

En el script de Coin

8. `$AnimationPlayer.play(“Rebote”)`
9. Añadir señal Animation Player – `Animation_finished(anim_name: String)`



10. Func `_on_Animation_Player_animation_finished(anim_name: String)`  
`queue_free()`

## Recoger coleccionables para cambiar de escena (Condición de Victoria)

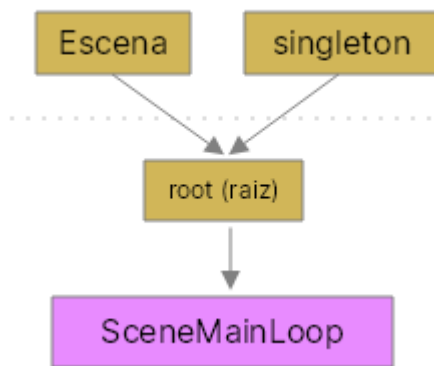
### SINGLETON

El patrón Singleton es una herramienta útil para resolver el caso de uso común en el que se necesita almacenar información persistente entre escenas y pueda almacenar variables globales como la información del jugador.

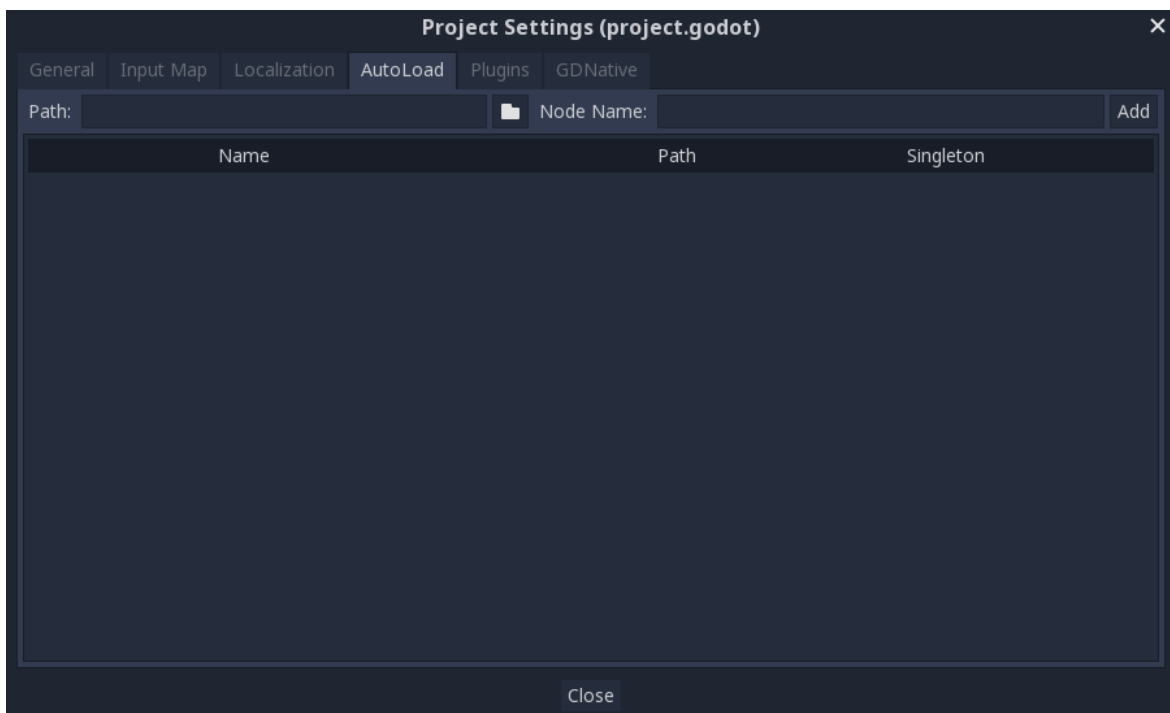
AutoLoad¶

Puedes crear un AutoLoad para cargar una escena o un script que herede de Node.

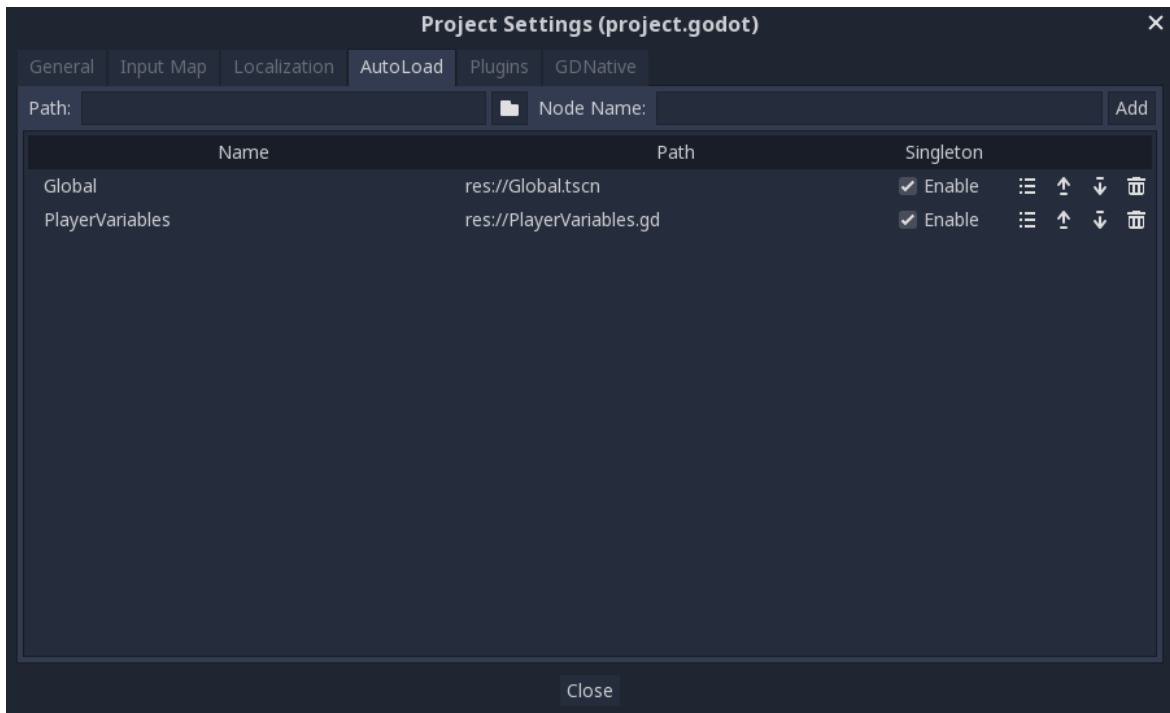
Nota: Cuando se autocarga un script, se creará un Nodo y el script se adjuntará a él. Este nodo se añadirá a la ventana raíz antes de que se cargue cualquier otra escena.



Para cargar automáticamente una escena o un script, seleccione Proyecto > Configuración del proyecto en el menú y cambie a la pestaña AutoLoad.



Aquí puede añadir cualquier número de escenas o scripts. Cada entrada de la lista requiere un nombre, que se asigna como propiedad del nombre del nodo. El orden de las entradas a medida que se añaden al árbol de escenas global puede manipularse utilizando las teclas de flecha arriba/abajo.

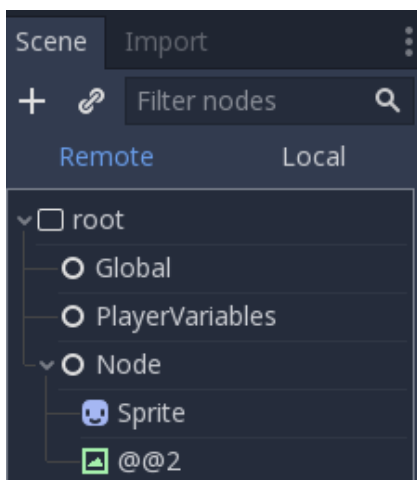


Esto significa que cualquier nodo puede acceder a un singleton llamado "PlayerVariables" si la columna Enable está marcada (que es el valor por defecto):

## GDScript

```
PlayerVariables.health -= 10
```

Tenga en cuenta que se accede a los objetos autocargados (scripts y/o escenas) como a cualquier otro nodo del árbol de escenas. De hecho, en el árbol de escenas en ejecución, verá que aparecen los nodos autoLoad:



## Practica

Player.gd

```
var coins = 0
```

```
func add_coin():
```

```
coins = coins + 1
```

```
func _physics_process(delta):
```

```
if coins == 4:
```

```
    #Indicamos que pase a la siguiente escena o al final del juego
```

```
    get_tree().change_scene("res://scenes/level_2.tscn")
```

Coin.gd

```
func _on_coin_body_entered(body):
```

```
body.add_coin()
```

## **HUD (COINS) \_ Canvas Layer**

Escena (Level\_1)

1. Añadir nodo hijo CanvasLayer
2. Renombrar nodo a "HUD"
3. Añadir nodo hijo Panel
4. Modificar propiedades del nodo Panel

Control – Panel – Custom Styles Styleboxflat

StyleBoxFlat

Bg Color Black Alpha (50)

Corner Radius

Top Left	20
Top Right	20
Bottom Right	20
Bottom Left	20

5. Añadir Assets (Gold Coin)
6. Añadir nodo hijo (Panel) – TextureRect
7. Cambiar texture (coin\_gold)  
Expand      Activado
8. Añadir nodo hijo (Panel) – Label

9. Establecer texto “Text”  
Custom Font – Font – Nuevo DynamicFont  
Dynamic Font – Font – Font Data  
Font Color - #ffcc00  
Establecer texto a “X”
10. Añadir nodo hijo (Panel) – Label
11. Establecer texto “Text”  
Custom Font – Font – Nuevo DynamicFont  
Dynamic Font – Font – Font Data  
Font Color - #ffcc00  
Establecer texto a “##”  
Renombrar a “Coins”

---

## HUD

1. Añadir Script

```
var coins = 0

Func _ready(): [Cuando el nodo entra en escena
$Animation(“Rebote”)
emit_signal (“coin_collected”)

Level_1
coin – signals (“coin_collected”) connect to HUD
Receiver_method(_on_coin_collected)
Hud.gd
coins = coins + 1
_ready()
Agregar
func _physic_process(delta):
# Mover de Player.gd
if coins == 4:
get_tree().change_scene(“res/scenes/level_2.tscn”)
Player.gd
Borrar ( func add_coin(): )
```

## ENEMIGOS

1. Crear escena nueva
2. Añadir nodo Kinematic Body
3. Renombrar nodo (Enemy<sub>1</sub>)
4. Crear nodo hijo CollisionShape2D (Capsule/Rectangle)
5. Crear nodo hijo AnimatedSprite
6. Añadir Script Enemy.gd

## PISAR ENEMIGOS

7. Añadir nodo hijo Area2D (renombrar Topchecker)
8. Añadir nodo hijo collisionshape (capsule)
9. Topchecker  
Layer 5 / Mask 1
10. Añadir señal body\_entered Conector (on\_top\_checker\_body\_entered)

## INTERFAZ

1. Crear escena nueva
2. Nodo base Control (Renombrar TitleMenu)
3. Añadir nodo hijo ColorRect (extender) Rect
4. Guardar Escena
5. Añadir nodo hijo label: texto (“Juego de Plataformas”)  
custom font\_new dinamyc font  
Dinamyc Font\_font\_fort data\_Kiery font.ttf  
size(54), font color, outline color
6. Añadir nodo hijo panel  
Custom Styles styleboxflat  
Bg color, Corner radius
7. Añadir nodo hijo Button: texto “jugar”
8. Renombrar PlayButton
9. Añadir Script (PlayButton)
10. Añadir señal (pressed)

func \_on\_button\_pressed():

get\_tree().change\_scene(“res://scenes/Level\_1.tscn”)

11. Configuración del proyecto  
General – Application – Run

Main Scene: res://Scenes/Menues/TitleMenu.tscn



## WINMENU / GAMEOVER

Crear escena nueva	Control	Control
Renombrar a	WinMenu	GameOver
Añadir nodo hijo	ColorRect	ColorRect
Añadir nodo hijo	Label (Ganaste!)	Label(Game Over)
Añadir nodo hijo	MenuButton (Menu Principal)	
Añadir Script	MenuButton,gd	
Añadir señal	<code>_on_button_pressed()</code> <code>get_tree().change_scene("res://Scenes/Menues/TittleMenu.tscn")</code>	

## HUD.gd

1. Mover script a `_on_coin_collected`  
if coins == 4:  
`get_tree().change_scene("res://Scenes/Menues/WinMenu.tscn")`

2. Borrar `_physics_process`

## Player.gd

`_on_fallzone_body_entered`  
`get_tree().change_scene("res://Scenes/Menues/GameOver.tscn")`

## AUDIO

### TitleMenu

1. Añadir nodo hijo `AudioStreamPlayer`
2. Renombrar a "Music Title Screen"
3. Set stream "Swinging Pants"
4. Autoplay ☒

### Level\_1

1. Añadir nodo hijo `AudioStreamPlayer`
2. Renombrar a "Level\_1\_BM"
3. Set stream "Retrobeat"
4. Autoplay ☒

## Player

1. Añadir nodo hijo `AudioStreamPlayer`
2. Renombrar a "SonidoSalto"
3. Set stream "Jump"
4. En importar de `Jump.ogg` desactivar loop y reimportar

## Player.gd

If `action_is_just_pressed("jump")`:  
`$Sonidosalto.play()`

## Coin

1. Añadir nodo hijo `AudioStreamPlayer`
2. Renombrar a "coin\_collect"
3. Set stream "Powerup2"

## Coin.gd

On\_coin\_body\_entered(body):

\$coin\_collected.play()

## GAME DESIGN

Una lista de referencia de elementos a tener en cuenta al crear un juego de plataformas de desplazamiento lateral, así como al jugarlo. Algunos de estos, no se aplican a todos los estilos, ya que las mecánicas pueden ser muy diversas entre los juegos. Vamos a ver juegos que tienen una jugabilidad de correr y saltar con sistemas adicionales o no.

Cualquier cosa que se planee hacer va a llevar 3-4 veces más tiempo y va a requerir más esfuerzo de lo anticipado al principio. Con la experiencia, se mejorará la predicción del tiempo que se necesita para hacer algo, pero incluso los desarrolladores más experimentados suelen cometer el error de subestimar el tiempo y los recursos que se necesitan para hacer algo pulido.

Ningún elemento puede ser subestimado. Todo es importante. Un juego de desplazamiento lateral en 2D/3D puede ser más difícil de hacer que otros géneros, porque cada pantalla requiere una configuración única si quieres hacer algo sólido. Hacer juegos requiere trabajo. La experimentación, y averiguar qué es lo que funciona requiere tiempo. Cada juego tiene sus propias cualidades. Con la experiencia se hace más fácil y hay algunas cuestiones que funcionan en la mayoría de las situaciones.

### Mecánicas principales

#### CAMARA

La cámara en un juego de plataformas de desplazamiento lateral es una de las partes más importantes. tiene que ser estable y fluida.

En la mayoría de los juegos de plataformas la cámara no debería mantener al personaje en el centro de la pantalla.

La cámara no debería desplazarse hacia abajo cuando un jugador cae en un pozo. La parte inferior de la pantalla debería equivaler a la muerte si quieres dejarlo claro para los jugadores. Tampoco debería desplazarse hacia arriba cuando el jugador salta, a menos que el personaje salga de la parte superior de la pantalla.

Deberías intentar bloquear la cámara en un solo eje para que sólo se mueva de izquierda a derecha (eje X) o de arriba abajo (eje Y) en cualquier sección de un nivel que los jugadores controlen. De este modo, reducirás los desplazamientos de la cámara y mantendrás un marco de referencia.

La cámara no debería desplazarse cuando el jugador salta si se mantiene dentro de los límites de la pantalla, sobretodo, cuando aumenta la velocidad a la que se mueve el personaje. Esto hace que la experiencia sea menos consistente.

Construye tus niveles para que soporten un estilo particular de cámara.<sup>1</sup>

### Vertical



contra.fandom.com



OldGameShelf.com

## SALTO

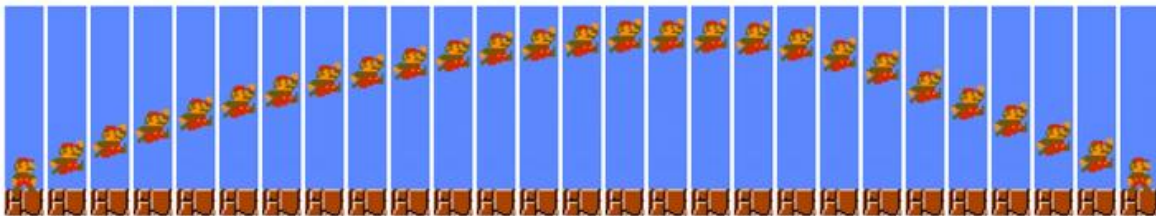
El salto tiene que ser perfecto. Si el juego no es divertido sólo con correr y saltar, debe corregirse y mejorar esa parte antes de seguir adelante. (Este suele ser el problema más importante en un juego de plataformas que no es bueno)

A tener en cuenta

**Demasiado lento:** Si el salto es demasiado flotante, es más difícil aterrizar sobre los enemigos y las plataformas de forma fiable. Esto hará que los jugadores no midan el tiempo de sus saltos y se impacienten esperando que el personaje baje.

**Demasiado rápido:** Si los saltos son demasiado rápidos, los jugadores tenderán a aterrizar delante de los enemigos o las plataformas, ya que saltarán antes de tiempo. Esto a menudo resulta en personajes que saltan hacia los lados de las plataformas en su camino hacia abajo.

Un buen ejemplo de salto tiene buena respuesta, se siente el peso y la gravedad y el porcentaje de errores del jugador es reducido. Debe subir rápido, tener un poco de tiempo de suspensión y luego caer con fuerza (puede cambiar según el tipo de personaje y el juego)



---

<sup>1</sup> <https://youtu.be/I9G6MnhfV7M>

Si el control en el aire durante el salto está disponible, hay que comprobar si los jugadores pueden recuperarse de un mal salto en el aire. Si casi nunca se recuperan, entonces debes aumentar el control de aire. Si se recuperan incluso si reaccionan muy tarde en el salto, entonces puede que tengas demasiado control aéreo. Es un acto de equilibrio.

Si el control del aire después de un salto no está disponible, entonces necesitas construir tus niveles teniendo en cuenta esta característica.

Hay dos estilos principales de salto para elegir.

Digital - Si saltas y dejas de mantener una dirección en el dpad o el stick, el personaje se detiene en seco y cae sin inercia. Este tipo de salto se siente poco natural y dependerá de la experiencia del juego.

Basado en la velocidad - Este salto mantiene la inercia en el aire. Si saltas mientras te mueves y dejas de mantener una dirección en el dpad o el stick, el personaje continuará moviéndose sin frenar. Los jugadores tendrán que compensar la inercia para frenar su movimiento. Este método da al personaje una sensación de peso.

Los saltos pueden ser realizados de forma que los personajes siempre salten a una altura consistente o para que cuanto más tiempo se mantenga el botón, más alto salte el personaje. El segundo método ya que te da la posibilidad de añadir más variedad a los niveles en cuanto a los saltos.

Los saltos también pueden tener sistemas de corrección como:

Agarre de saliente - permite a los jugadores agarrar un saliente al impactar justo por debajo del borde.

Salto a la pared: permite a los jugadores recuperarse al impactar contra una pared justo por debajo de un borde.

Permitir el salto tardío (efecto Willie Coyote) - permitir una pequeña ventana de tiempo en la que los jugadores puedan pulsar el salto, aunque apenas se hayan alejado de un saliente.

Asistencia en los bordes - hacer que los jugadores suban a una plataforma que apenas han alcanzado si no han caído por debajo del borde. Esto es similar al concepto anterior, pero al otro extremo del salto.

### Mecánicas secundarias

## VELOCIDAD

Correr no es necesario para hacer un gran juego de plataformas de desplazamiento lateral, pero da a los jugadores una experiencia más compleja.

Permitir a los jugadores la capacidad de aumentar su velocidad mediante el uso de un botón de correr o una mecánica es un elemento fundamental para algunos de los mejores juegos de plataformas de todos los tiempos. Esto permite que el juego se pueda jugar de dos maneras completamente diferentes. Los jugadores pueden jugar de forma controlada, fácil y consistente, simplemente moviéndose a través de un nivel por defecto.

COLISION

La colisión de los enemigos puede hacer que la experiencia sea mejor o peor.

Si estás cayendo sobre un enemigo al que puedes golpear de esa forma, entonces deberías conseguir golpearlo. Si el enemigo te golpea cuando parece que deberías golpearlo, entonces tienes que solucionar el problema.

La interacción con los enemigos debe ser consistente y predecible, así que no muevas su colisión de forma errática o los animes fuera de sus límites de colisión.

#### Sugerencias de desarrollo

Utilizar distancias de salto medidas - No fije una altura o distancia de salto máxima en ningún salto a menos que sea un salto a una ruta alternativa o secreta.

Reformule secuencias del nivel que resulten incómodas, cosas como golpear a un jugador antes de un salto o hacer que aterrice en un lugar extraño, probablemente debería ser sustituida por algo que resulte mejor si la experiencia se resiente. El testeo de los niveles, aun como si se estuvieran cometiendo errores es importante. Eliminar los momentos incómodos del juego hace que el juego se sienta más pulido. No hace falta mucho para que la gente se sienta mal con la experiencia.

Los nombres importan. Un nivel llamado nivel genérico 01 tiene menos potencial para ser bueno que uno con un nombre pensado previamente y más elaborado.

Todo elemento en el juego puede ser memorable y reconocible. Cualquier, enemigo, objeto, detalle de nivel o personaje del juego en una camiseta, ¿sería atractivo? ¿Podrías decir de qué juego proviene inmediatamente? ¿Te sorprende? ¿Tiene personalidad? ¿Lo recuerdas? ¿Puedes hacerlo más interesante o divertido?

La separación del fondo es importante. No debe haber elementos ambiguos entre el área jugable y los elementos de fondo. Si no se puede interactuar con él, entonces debe ser de un color o brillo diferente al que se puede. Si el jugador intenta saltar a los elementos de fondo pensando que son áreas jugables, debería ser corregido.

Hay cosas que son fáciles de hacer y hay cosas que son difíciles de hacer. Si algo es difícil de hacer, llevará mucho tiempo y, por lo general, será más difícil de cambiar o ajustar. Es muy probable que se pueda hacer algo fácil que dé resultados similares, si no superiores. Además, los resultados son más inmediatos y permiten perfeccionar el proyecto con el tiempo extra.

Cada nivel debe ser recordado y definido por una "única" cosa que destaque. Esto puede ser un momento (un salto, un enemigo). Si se utilizan demasiadas ideas, no se recordará ninguna. Los niveles mas memorables generalmente se pueden definir como "el nivel de (inserta un elemento memorable)" cuando los jugadores se comunican entre ellos.

Una gran técnica para hacer una secuencia memorable es establecer un patrón de juego y luego romperlo cuando los jugadores empiezan a predecirlo y esperarlo.

las cosas en movimiento son más memorables que las estáticas. Se pueden crear todo tipo de configuraciones interesantes para jugar utilizando diferentes patrones de movimiento y formas. Siempre intentando que las cosas sean coherentes y comprensibles en un nivel. Se prefiere que los jugadores adquieran un ritmo de comprensión.

Utilizar demasiadas configuraciones de movimiento diferentes puede acabar siendo un detrimento para la experiencia.

Desincronización - A veces, tener elementos (obstáculos del nivel) que están cronometrados de forma diferente y que se sienten desincronizados entre sí hace que las secuencias de juego sean más interesantes:

Por ejemplo, 3 pilares que se mueven hacia arriba y hacia abajo. Aunque todos se muevan exactamente a la misma altura, se puede cambiar sus tiempos, uno completando su movimiento en 3 segundos, el siguiente completando su movimiento en 9 segundos y el último completando su movimiento en 6 segundos.

Puedes utilizar elementos como monedas, anillos, plátanos, etc. para hacer los niveles más interesantes y mantener a los jugadores ocupados. La recolección de estos objetos debe tener una razón de ser y proporcionar un beneficio al jugador. Demasiados con demasiada frecuencia les resta valor. Si se dan muy pocos, genera frustración en el jugador.

Se pueden utilizar estos elementos para guiar a los jugadores a jugar de una manera determinada, pero sin abusar de esto. Si siempre muestran la ruta perfecta, entonces el jugador probablemente la utilizara. Pueden ser utilizados para ayudar a calcular un salto difícil o situaciones similares.

Un elemento del nivel puede utilizar otros sentidos del jugador para ser interesante. Hay otras formas interesantes de utilizar de plantear los niveles o desafíos. Se puede jugar con el tiempo, la gravedad, el sonido y la música o cualquier otro elemento para hacer algo original y atractivo.

La uniformidad no siempre es buena: si algo se mantiene en la misma dificultad todo el tiempo, no es interesante. Generalmente las canciones o montañas rusas con más variaciones son las más divertidas. Hay que variar la dificultad para mantener a los jugadores interesados e involucrados.

La dificultad debería ser más bien una rampa: fácil, Difícil, Medio. Siempre debe haber algún elemento que suponga un reto, de lo contrario los jugadores no se sentirán conectados. Se presenta la mecánica única del nivel, se desafía al jugador en la mitad, con una situación difícil de afrontar, y luego permitir la relajación para disfrutar la experiencia.

## RECURSOS

### Documentacion

Oficial <https://docs.godotengine.org/es/latest>

Traduccion <https://godot-doc-en-espanol.readthedocs.io/es/latest/index.html>

### GDScript

<https://gd.tumeo.space>

<https://gdscrip-online.github.io>

Godot browser versión <https://editor.godotengine.org/releases/latest/>

### Recursos

<https://github.com/GDQuest>

<https://godotengine.org/asset-library/asset>

<https://godotmarketplace.com>

<https://github.com/godotengine/awesome-godot#readme>

### Tutoriales

#### Español

<https://indielibre.com/category/godotengine/>

<https://nodosgodot.blogspot.com>

#### Inglés

<https://github.com/mistertaftcreates>

<https://kidscancode.org/blog/tags/godot/>

[http://kidscancode.org/godot\\_recipes/](http://kidscancode.org/godot_recipes/)

<https://www.davidepesce.com/godot-tutorials/>

<https://www.codingcommanders.com/godot/index.html>

<https://godottutorials.com>

[https://www.gotut.net/category/godot/godot\\_general/](https://www.gotut.net/category/godot/godot_general/)

<https://laptrinhx.com/tag/godotengine/>

<https://devga.me/tutorials/godot2d/>

<https://godotlearn.com/articles/>