Taller 3 GoDot

Parte 2

- Interacciones
- Multiples escenas

Ing. Eduardo Kunysz
EPET12
San Martín de los Andes

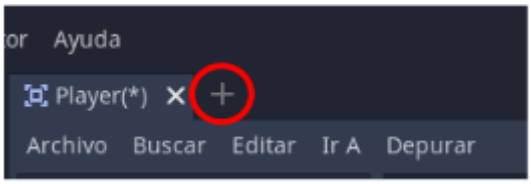
Introducción

Continuaremos con la segunda parte en donde incorporaremos otros personajes e interacciones con los anteriores. Luego de esta Parte tendremos un juego complemente operativo.

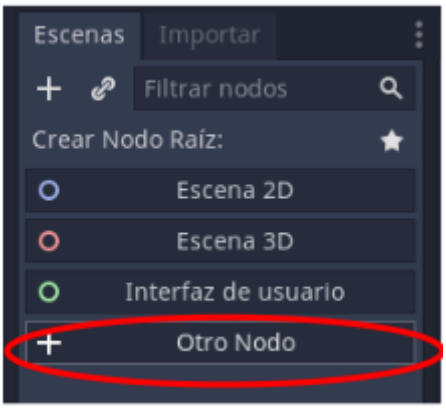
Sumario

Nueva escena Monstruo

Agregaremos una nueva escena presionando en el botón del signo más + que aparece al lado de la pestaña Player(*) en la parte superior de la ventana.

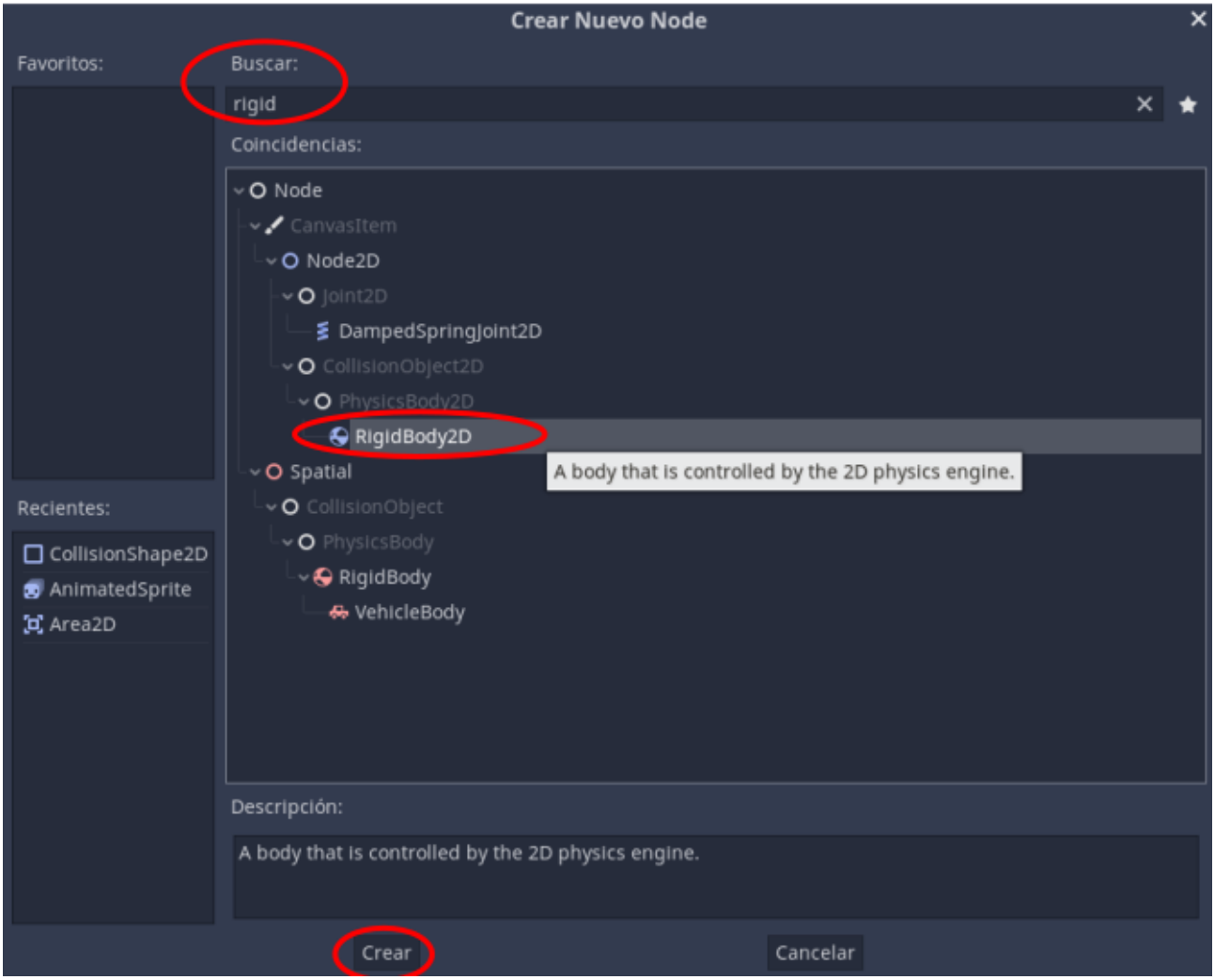


A continuación agregaremos un nodo raíz a esta nueva escena



Nodo RigidBody2D

A continuaciń agregamos el nodo RigidBody2D



Luego cambiamos el nombre del nodo a uno que nos parezca bien, yo le he colocado **Monstruo**.

RigidBody2D

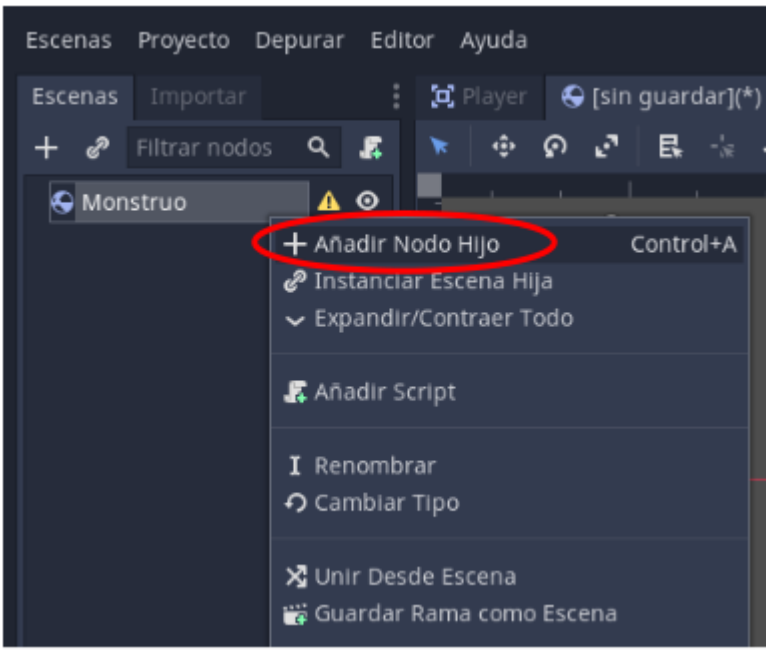
El nodo RigidBody2D es como si estuvieras manejando un objeto en el mundo real que tiene peso y se comporta de manera realista cuando choca con otras cosas. Es como si tus objetos en el juego respondieran a la física de manera natural.

Descripción:

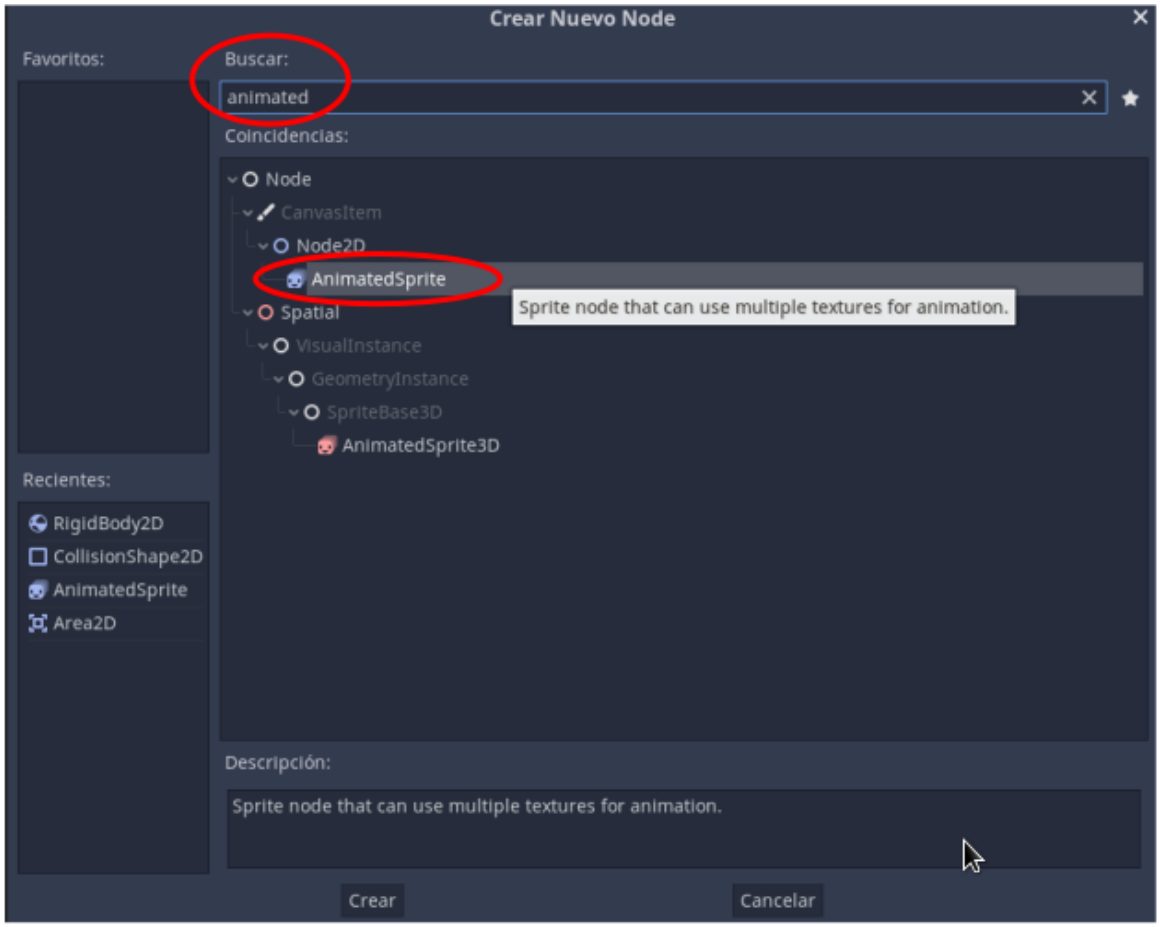
1. Gravedad y peso: El nodo RigidBody2D simula objetos que tienen peso en el juego. Si los dejas caer, caerán hacia abajo como si estuvieran afectados por la gravedad. Es como si estuvieras dejando caer una pelota en el mundo real y ves cómo cae al suelo debido a su peso.
2. Colisiones realistas: Los objetos con RigidBody2D interactúan entre sí como en el mundo real. Si dos objetos chocan, se empujarán y rebotarán de manera realista, teniendo en cuenta sus masas y velocidades. Imagina que estás lanzando dos pelotas una contra la otra y ves cómo rebotan.
3. Movimiento realista: Puedes aplicar fuerzas o impulsos a un objeto con RigidBody2D para que se mueva. Imagina que estás empujando un carro en el juego y ves cómo responde a tus acciones, como si estuvieras empujando algo en el mundo real.
4. Restricciones físicas: También puedes configurar restricciones en un objeto con RigidBody2D, como limitar su rotación o movimiento en ciertas direcciones. Es como si estuvieras ajustando cómo un objeto puede moverse o girar según las reglas del mundo físico.

El nodo RigidBody2D te permite agregar física realista a tus objetos en el juego. Los objetos con RigidBody2D caen debido a la gravedad, chocan y rebotan como en el mundo real, y responden a fuerzas y movimientos de manera natural. Es como si tus objetos en el juego obedecieran las mismas leyes físicas que experimentamos en la vida real.

Vamos añadir un nodo hijo del tipo AnimatedSprite (ya lo usamos en la primer parte de este mismo taller)

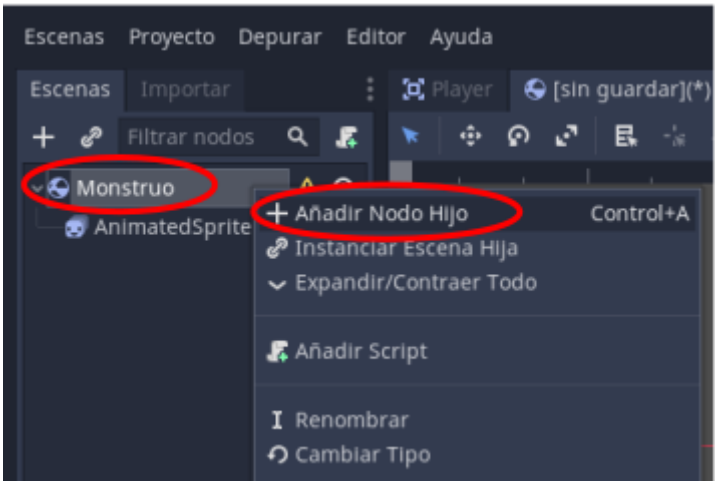


Y ahora buscamos y creamos el nodo AnimatedSprite



Nodo VisibilityNotifier2D

Con el nodo Monstruo seleccionado, añadiremos un nuevo nodo hijo de este (clic derecho añadir nodo hijo), el cual será **VisibilityNotifier2D**



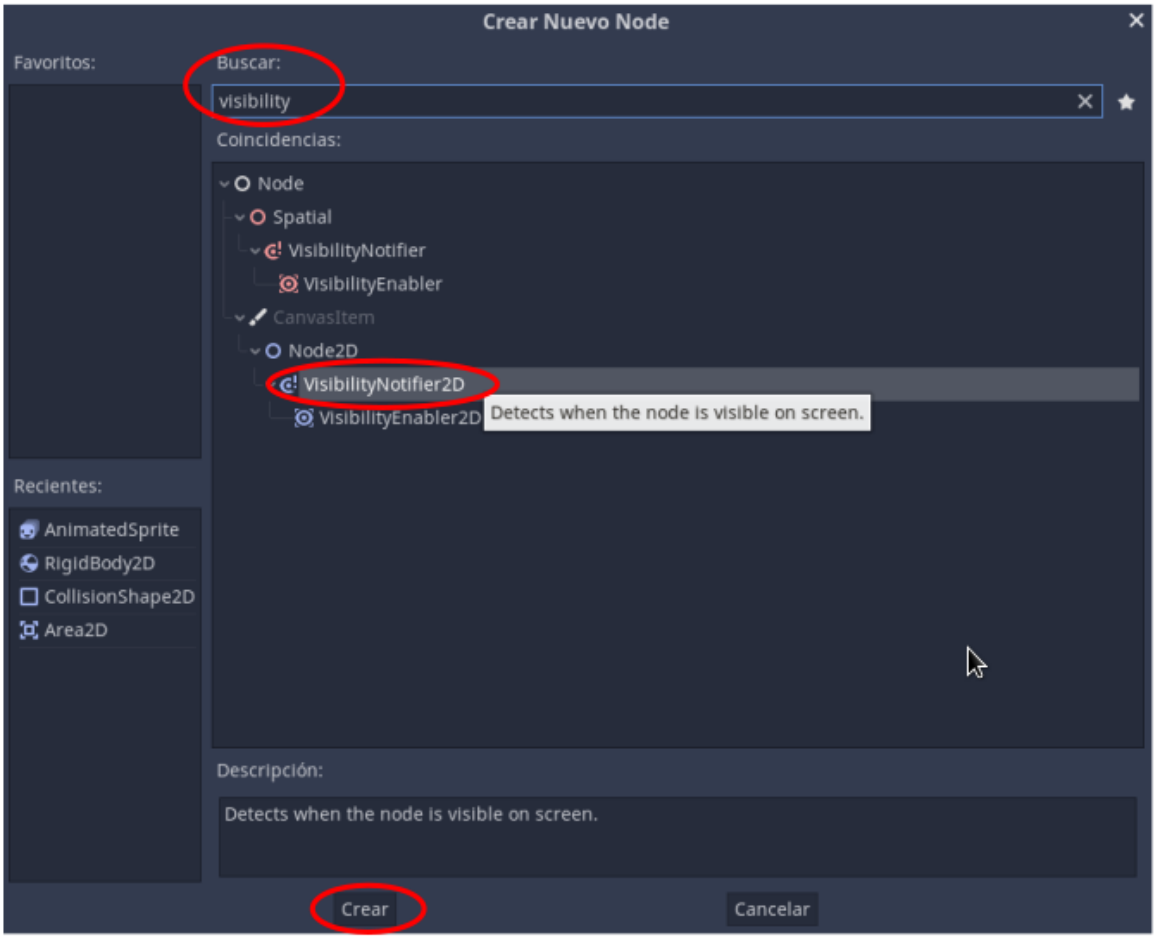
VisibilityNotifier2D

El nodo VisibilityNotifier2D es como si tu personaje tuviera un radar que le permite saber cuándo otros objetos están cerca o fuera de su campo de visión en el juego. Es como si tu personaje pudiera sentir si algo está cerca o lejos y tomar decisiones basadas en eso.

Descripción:

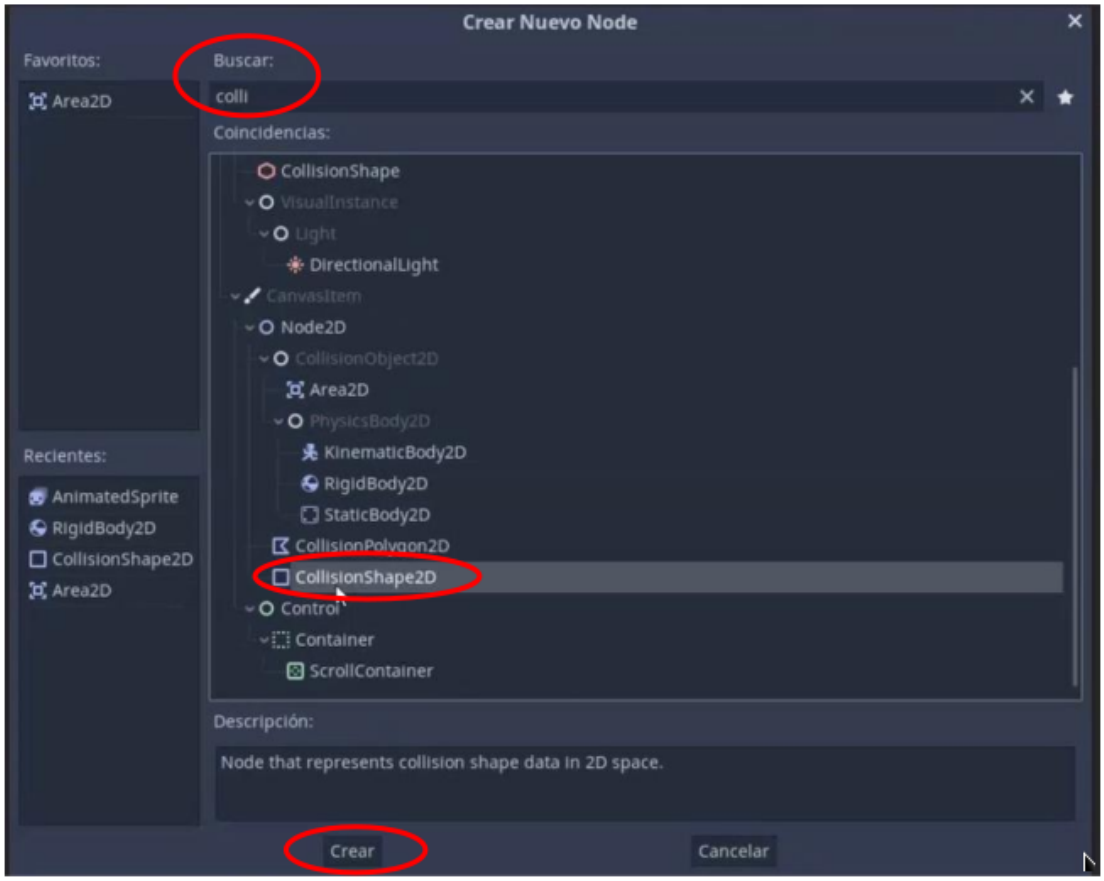
1. Detector de proximidad: El nodo VisibilityNotifier2D detecta cuándo otros objetos están cerca o lejos del objeto al que está vinculado. Funciona como un "sensor de cercanía" en el juego, permitiendo que tu personaje sepa si algo está a su alcance.
2. Eventos de visibilidad: Cuando un objeto entra o sale del campo de visión del VisibilityNotifier2D, se activan eventos. Es como si tu personaje notara cuando algo aparece o desaparece en su entorno y pudiera reaccionar en consecuencia.
3. Control de lógica: Puedes programar tu juego para que haga algo cuando el VisibilityNotifier2D detecte que un objeto está cerca o fuera de la vista. Por ejemplo, podrías hacer que tu personaje reaccione a la presencia de enemigos cercanos o que cambie su comportamiento cuando un objeto desaparezca de su vista.
4. Ahorro de recursos: También puedes usar VisibilityNotifier2D para controlar cuándo los objetos deben procesarse. Si un objeto está fuera de la vista, puedes pausar su lógica para ahorrar recursos de la computadora. Esto es como si tu personaje solo prestara atención a las cosas que puede ver y no se preocupara por lo que está lejos.

El nodo VisibilityNotifier2D es como un "radar" en el juego que permite que los objetos sepan cuándo algo está cerca o fuera de su vista. Es útil para activar eventos basados en la visibilidad y para ahorrar recursos al controlar cuándo los objetos deben procesarse. Es como si tus personajes en el juego pudieran sentir lo que está a su alrededor y tomar decisiones en función de eso.



Creamos un nodo hijo **CollisionShape2D.** Este tipo de nodos ya los hemos conocido en la primer parte de este taller.

Con el nodo Monstruo seleccionado, añadiremos un nuevo nodo hijo de este (clic derecho añadir nodo hijo), el cual será CollisionShape2D

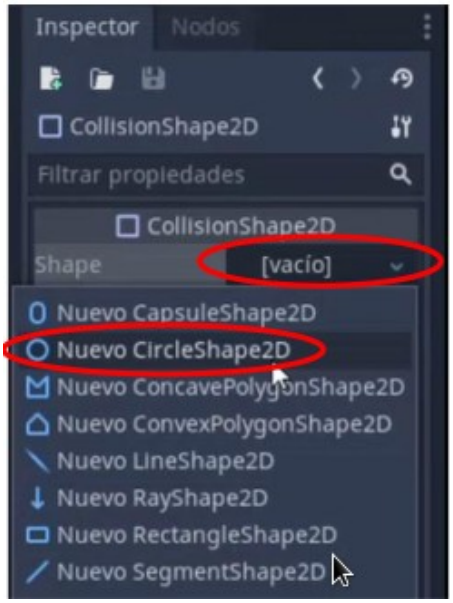


Configuraciones

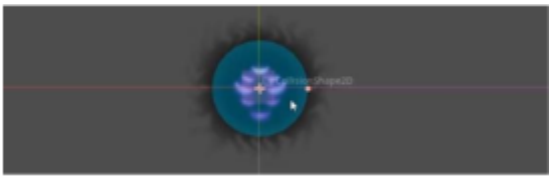
A partir de acá comenzaremos con configuraciones.

Configuramos la forma de colisión

En el grupo de Escenas, seleccionamos el nodo CollisionShape2D y nos dirigimos al inspector, en el cual encontraremos el atributo Shape [Vacío], debemos seleccionar la forma que más nos convenga, en el caso de este enemigo seleccionaremos CircleShape2D (círculo).



Ahora solo cambia el tamaño de la forma de colisión según lo consideres adecuado, ejemplo:

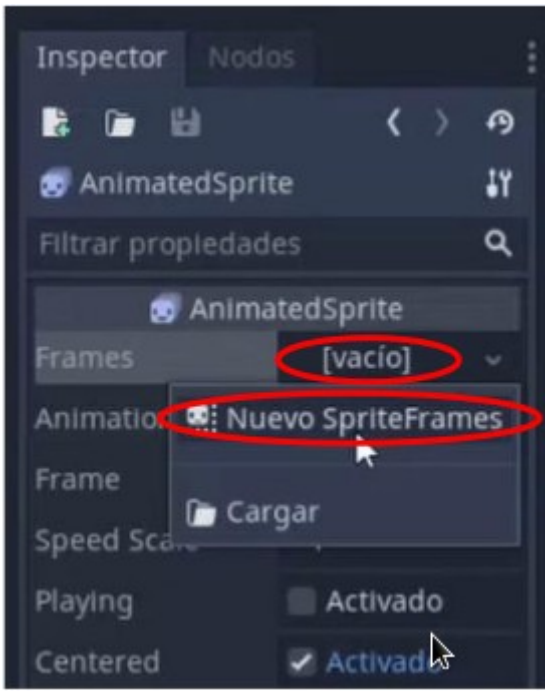


Copiar los sprites del archivo adjunto con los enemigos. Como hicimos en la primer parte. Los pondremos en la misma carpeta que nuestro personaje.

De la misma forma que hicimos en la primer parte, vamos a agregar los sprites.

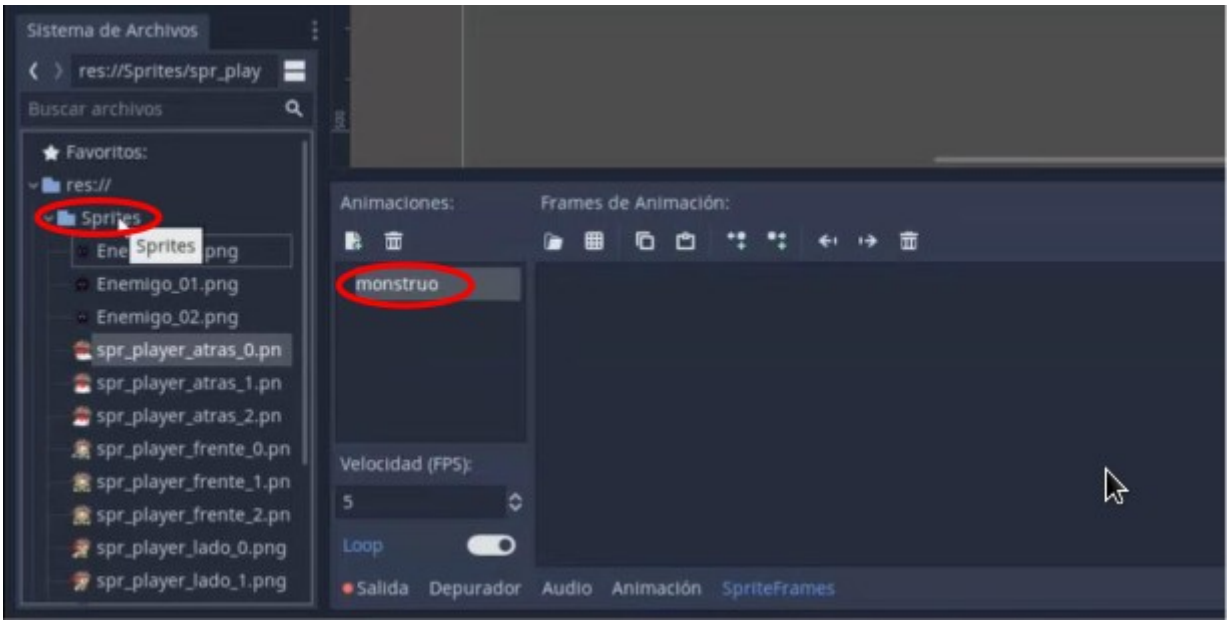
Agregando y animando los Sprites

Con el nodo hijo **AnimatedSprite** seleccionado, nos ubicamos en el Inspector seleccionando el Atributo Frames que aparece **[Vacío],** en el cual elegimos la opción Nuevo **SpriteFrames**.

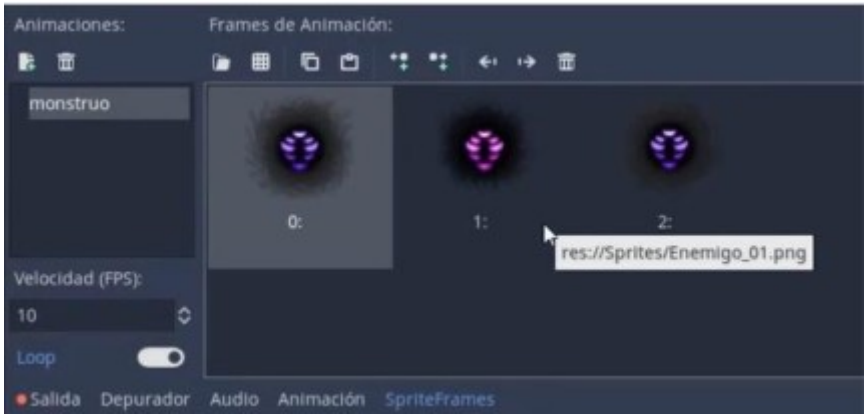


Posteriormente debemos dar clic sobre el recién creado **SpriteFrames** para que se despliegue en la parte inferior de la ventana el panel para agregar animaciones.

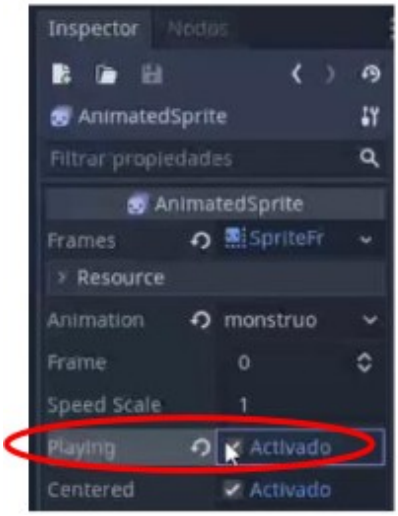
Ahora, en el sistema de archivos ingresas a la carpeta “Sprites” (doble clic sobre dicha carpeta) para que muestre las imágenes. Luego editas el nombre de la animación “default” y le colocas de nombre “monstruo”.

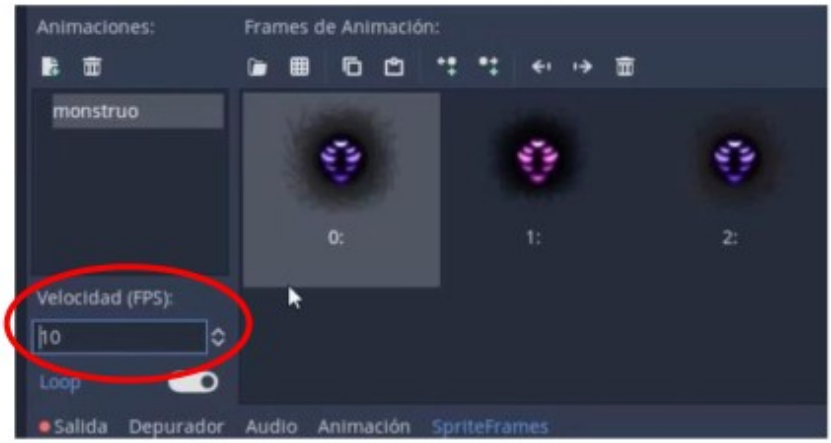


Continuamos arrastrando las tres imágenes .png que le darán vida a nuestro enemigo:



Y por último activas el tributo Playing en el inspector y cambias la Velocidad (FPS) a tu gusto (calculando que la animación se vea bien). (Esto es una gran diferencia a la configuración que hicimos en la parte 1, de esta forma se animará constantemente el personaje).

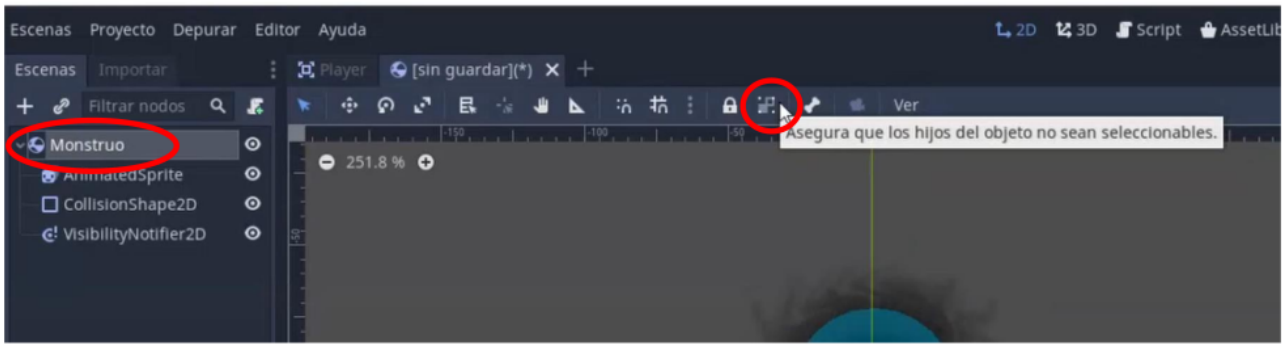




Agrupando a los hijos del nodo

Si intentamos mover a nuestro enemigo, notaremos que la forma de colisión y el sprite se mueven por separado, lo que puede dificultar tu trabajo, entonces, lo que haremos será “agrupar” ambos objetos, haciendo que los nodos hijos no sean seleccionables.

Con el nodo Player seleccionado, presiona sobre el botón que se indica en la imagen:



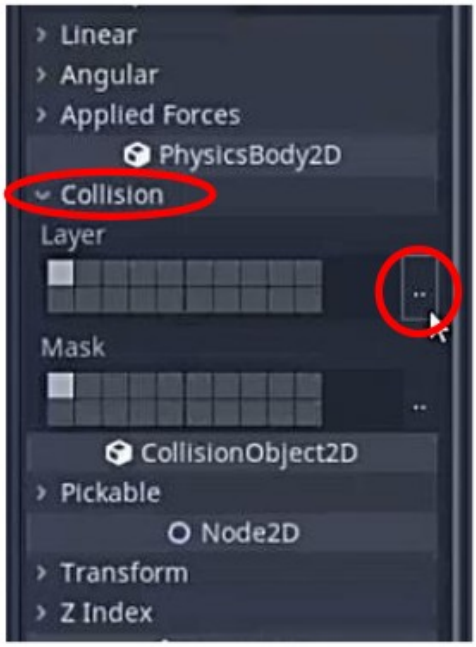
Ahora vamos a guardar esta nueva escena. Tener en cuenta que la primer escena y la segunda se guardan en archivos separados. A esta escena la guardamos con el nombre **Monstruo.tscn**

Cambiando escala de gravedad

Como no queremos que nuestro enemigo “caiga” (más bien queremos que se mueva en toda la pantalla), debemos cambiar la esacala de gravedad. Esto lo haremos con el nodo Monstruo seleccionado y dirigiéndonos al Inspector, igualando atributo **Gravity Scale** a 0.

Editando las físicas del cuerpo 2D

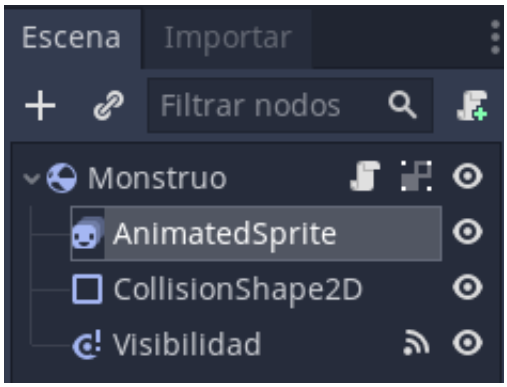
En un futuro programaremos el Script para que el enemigo aparezca muchas veces en pantalla (se múltiplique), entonces, para que este no choque con sus múltiples copias, **desactivamos la casilla de la Capa 1** en Collsion de PhysicsBody2D (en el Inspector y siempre con el nodo Monstruo seleccionado):



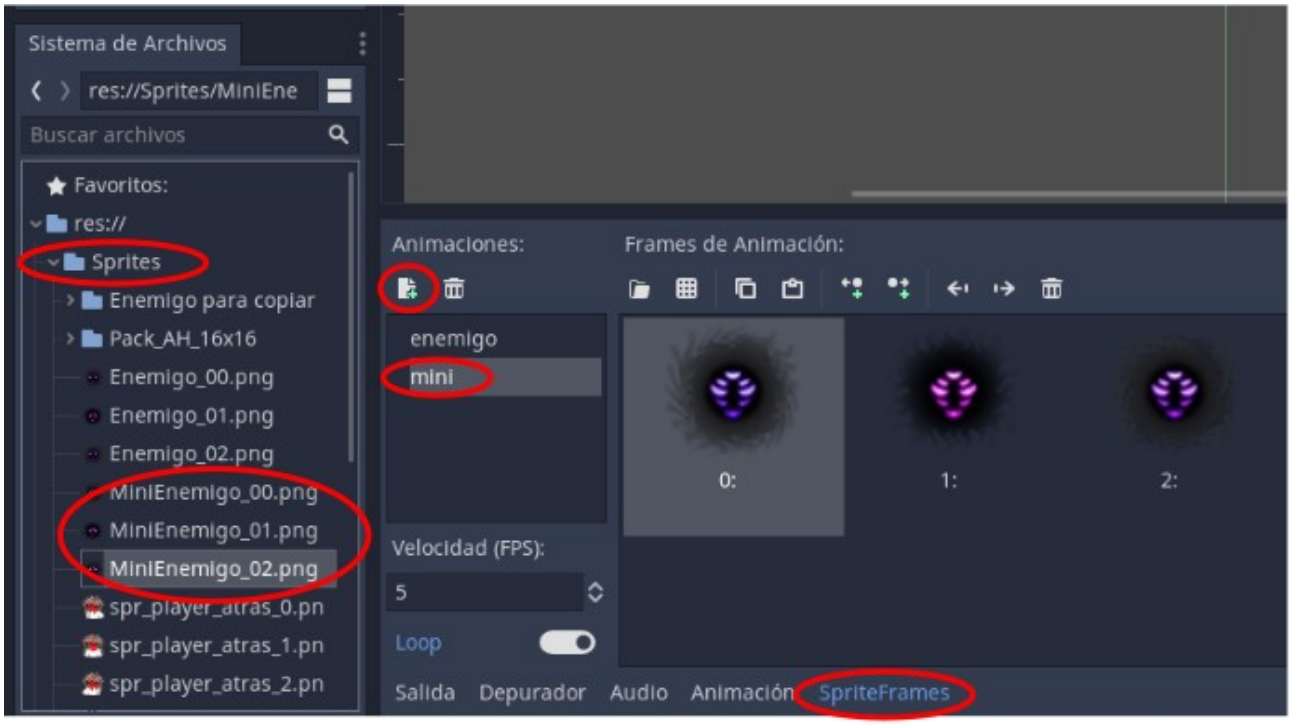
Otra versión del enemigo

Vamos a descomprimir y agregar el enemigo mini en nuestra carpeta de sprites.

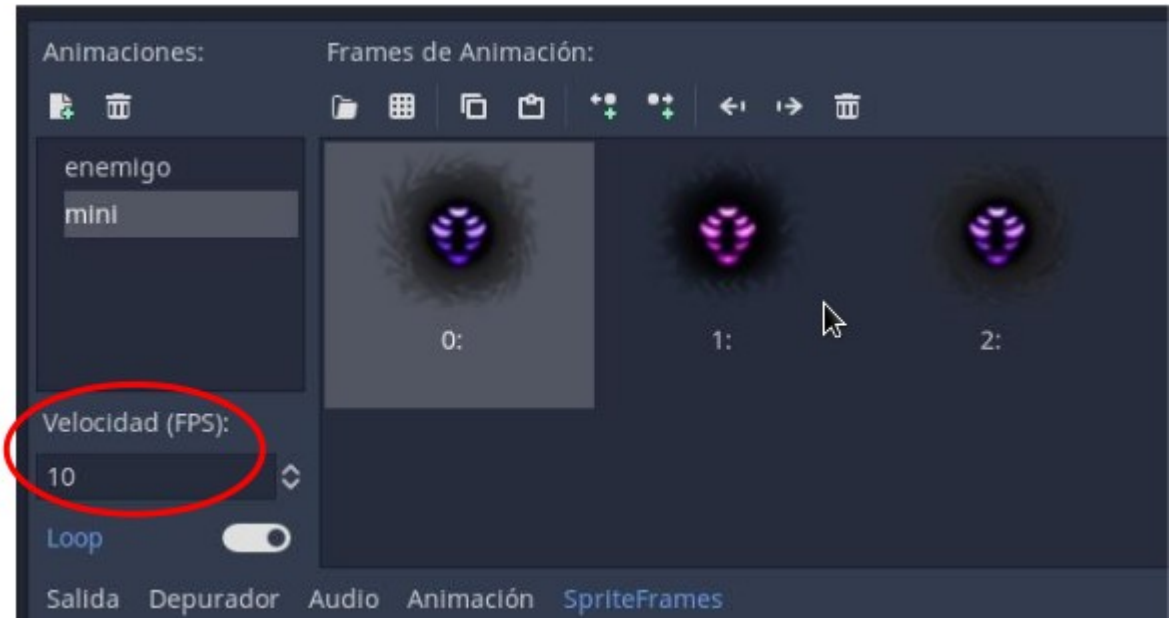
Seleccionamos el nodo AnimatedSprite



Abrimos el panel SpriteFrames, creamos una nueva animación que llamaremos “mini” y arrastramos los sprites que acabamos de descomprimir (navegando en el sistema de archivos)

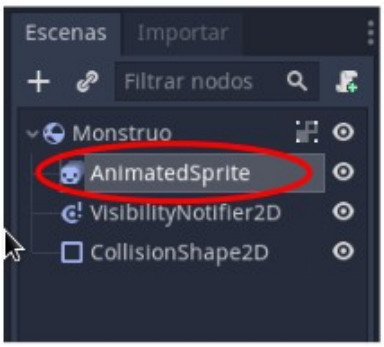


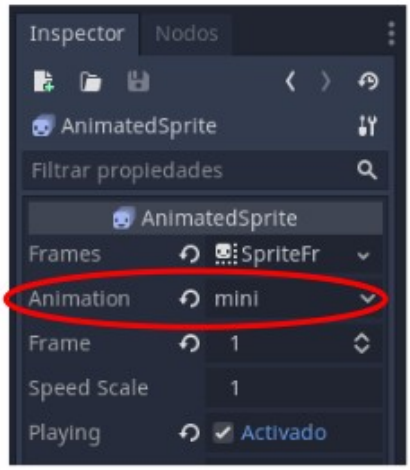
Por último, asignamos la velocidad de frames (FPS) que consideremos adecuada. 10 FPS



Configurando los nuevos sprites

Para que la **CollisionShape** encaje de mejor manera con la versión pequeña de nuestro enemigo, redimensionaremos su tamaño (haciéndola más pequeña, ya que posteriormente en programación podremos aumentar su tamaño en base a los códigos escritos). Para empezar, con el nodo **AnimatedSprite** seleccionado, nos aseguramos de que el atributo **Animation** se encuentra con el valor "**mini**" (o sea, la animación principal de nuestro enemigo será la versión pequeña que acabamos de agregar).

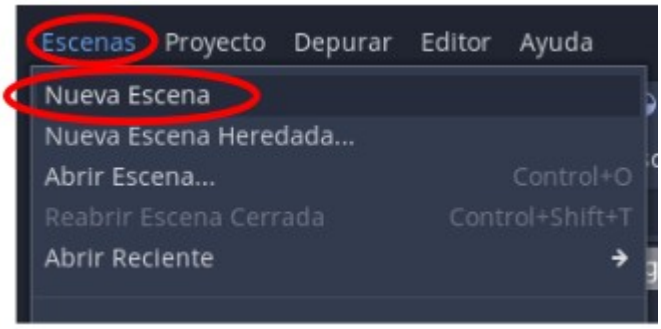




Luego seleccionamos el nodo CollisionShape2d y cambiamos el tamaño de la figura celeste manualmente.

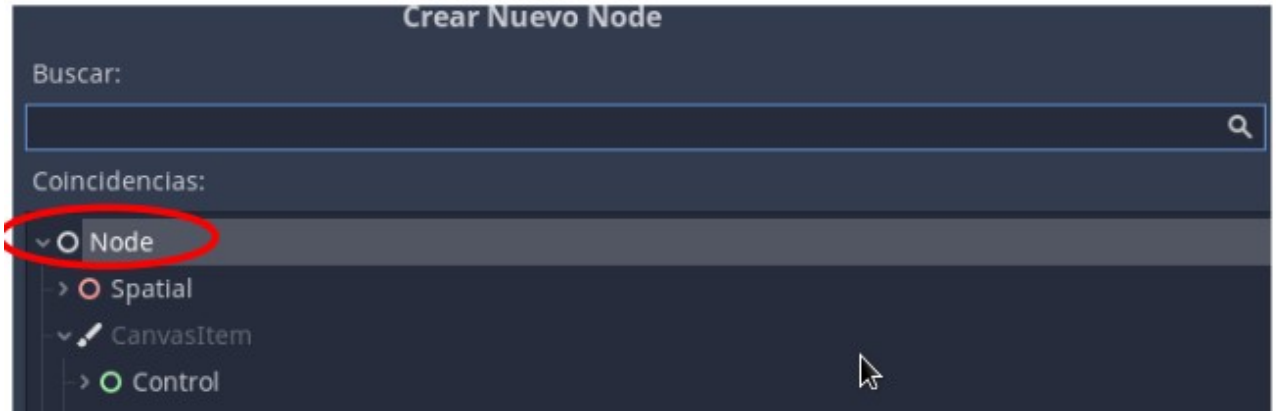
Nueva escena Mundo

En el menú Escena seleccionamos la opción Nueva Escena



Creando un nodo raíz para la nueva escena

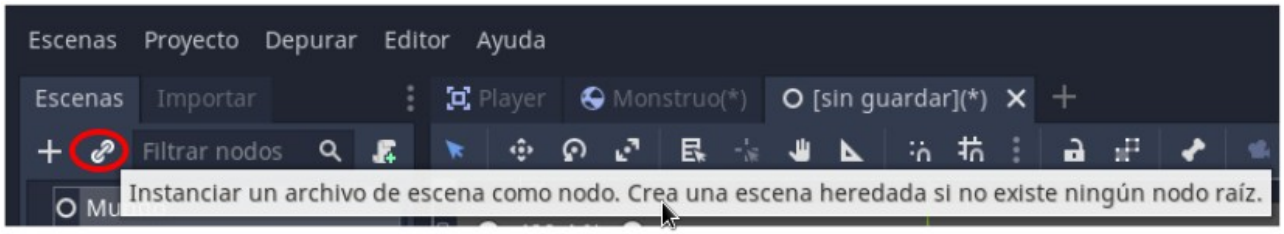
Damos clic sobre + Otro Nodo, buscamos un Nodo normal (Node) y finalizamos el proceso con clic en Crear.



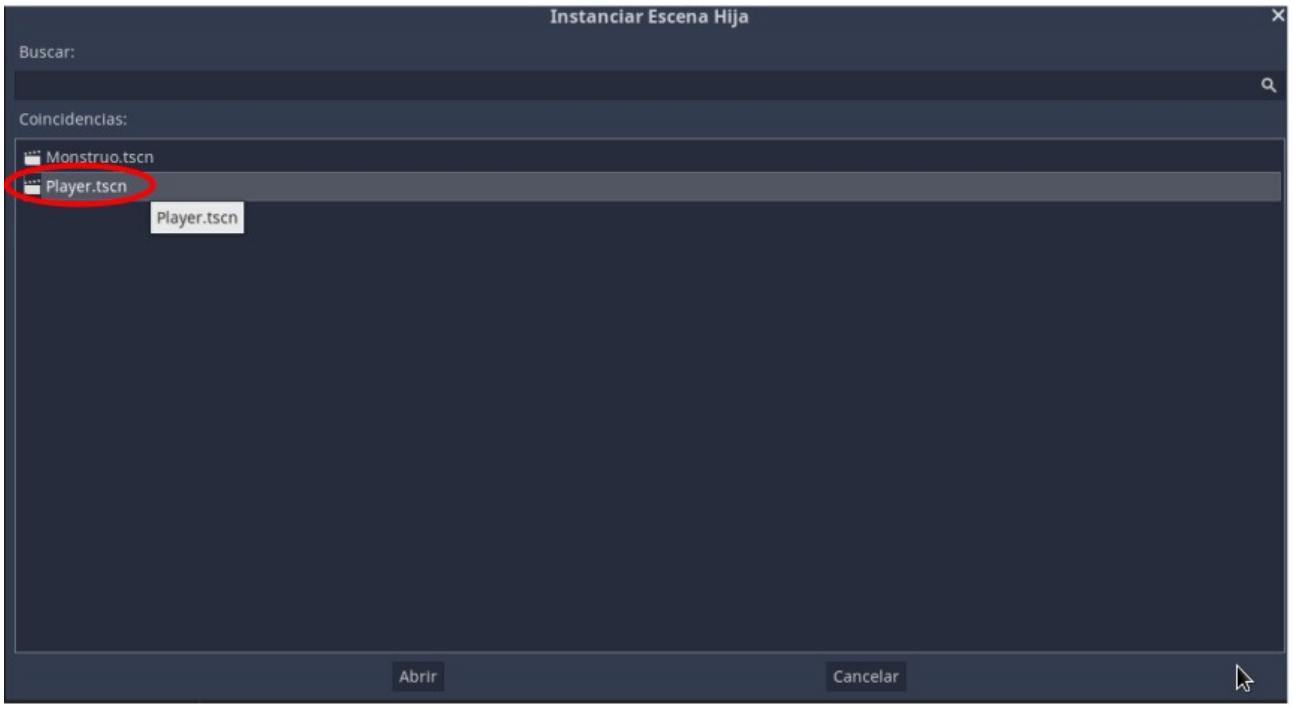
Vamos a asignarle el nombre “Mundo” a nuestro nodo, ya que este representará precisamente al mundo en el que las escenas Player y Monstruo convivirán.

Instanciando Nodo player

Ahora vamos a instanciar a la escena de nuestro Player en esta tercer escena (dicho de otra forma, con la creación de esta instancia estamos diciendo que nuestro jugador principal podrá “aparecer” dentro de nuestra escena Mundo). Para ello, debemos dar clic sobre el botón con forma de cadena que aparece en la parte superior



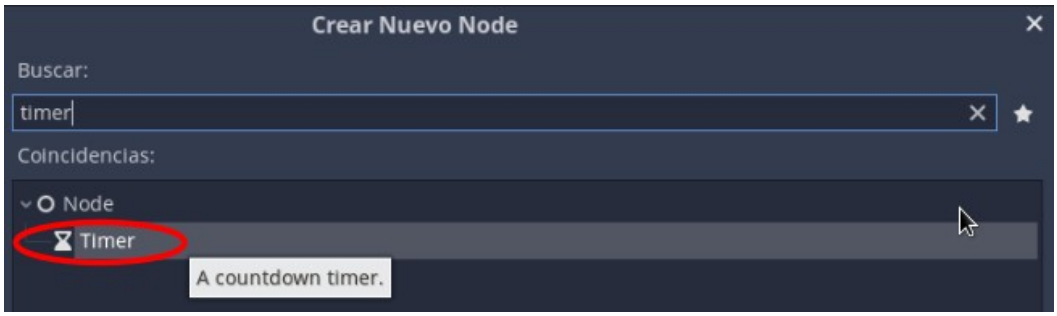
Y luego en la ventana de exploración que aparece, abriremos con doble clic la escena de nuestro jugador principal, es decir, la escena Player.tscn



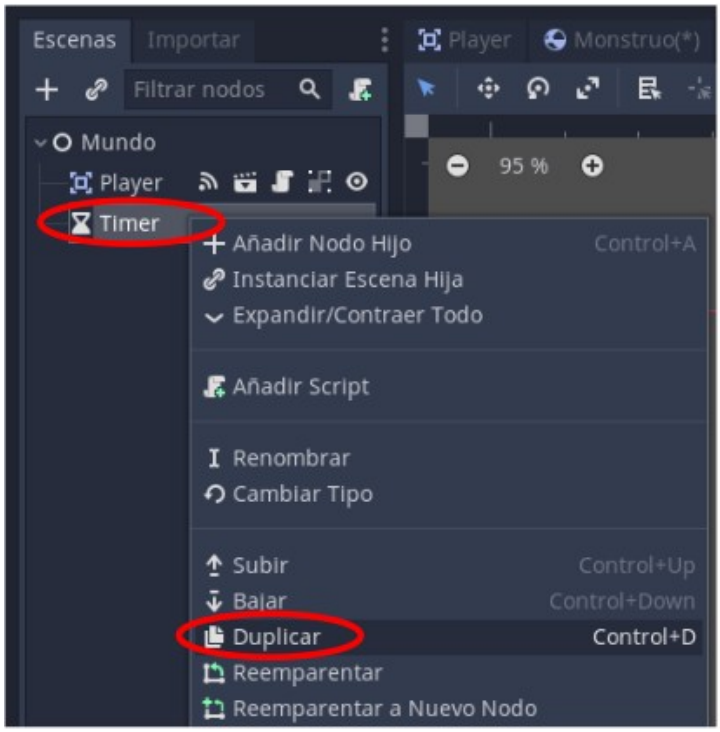
(Notarás que el jugador principal ya aparece en esta nueva escena).

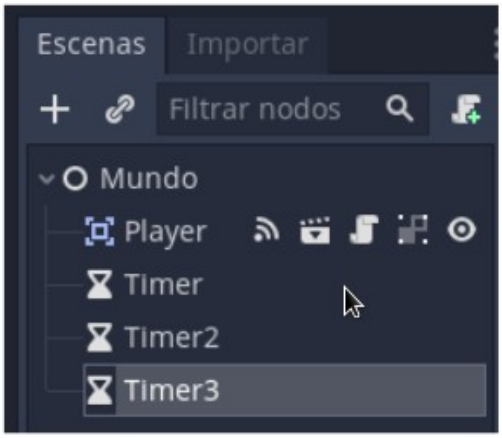
Nodo Timer

Iniciamos agregando un nodo de tipo Timer y luego lo duplicamos hasta tener 3 nodos de los mismos. Clic derecho sobre el nodo raíz (Mundo) y luego clic en Añadir Nodo Hijo



Y duplicamos con Clic Derecho → Duplicar sobre el nodo Timer, hasta tener 3 de estos:





Timer

El nodo Timer en Godot es como un reloj en tu juego que te ayuda a controlar cuándo suceden cosas. Es como si estuvieras configurando una alarma para que algo suceda después de un cierto tiempo, como activar una trampa o hacer que un enemigo aparezca.

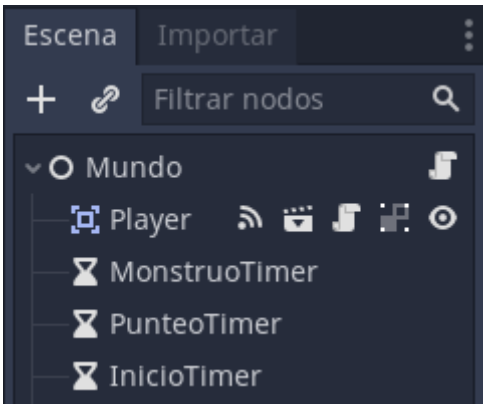
Descripción:

1. **Temporizador en el juego**: El nodo Timer es como un temporizador que puedes ajustar en tu juego. Puedes decidir cuánto tiempo debe pasar antes de que algo ocurra. Por ejemplo, podrías establecerlo para que después de 5 segundos, se abra una puerta en el juego.

2. **Eventos programados**: Cuando el temporizador llega a cero, se activa un evento en tu juego. Es como si la alarma sonara y dijera "¡Hora de hacer algo!". Puedes programar lo que quieras que suceda en ese momento, como que un enemigo se mueva, un objeto aparezca o una trampa se active.
3. **Control de ritmo**: El Timer te permite controlar el ritmo del juego. Puedes decidir cuándo deben ocurrir ciertas acciones, creando una sensación de dinamismo en el juego. Imagina que estás diseñando un juego de carreras y quieres que los obstáculos aparezcan cada cierto tiempo para mantener el desafío.
4. **Ajuste de intervalos**: Puedes configurar el temporizador para que se repita cada cierto tiempo. Esto es útil para crear efectos repetitivos, como que las luces parpadeen o que una trampa se active y desactive periódicamente.

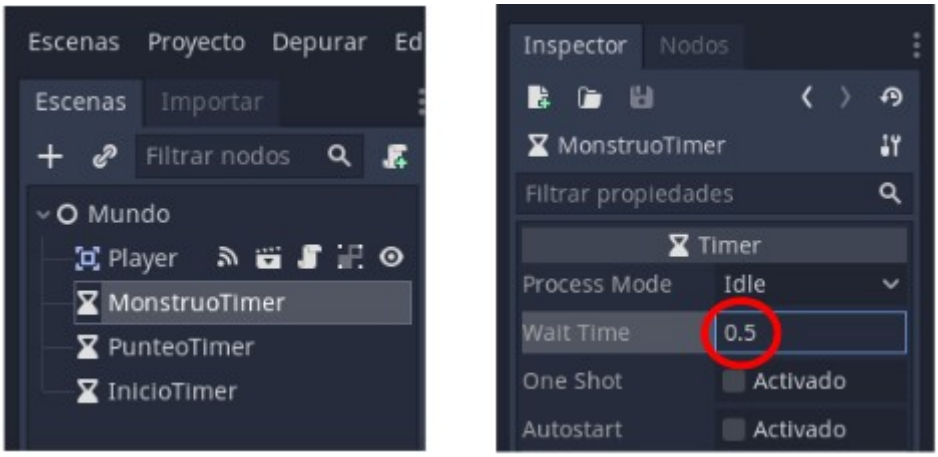
El nodo Timer en Godot es como un reloj que te ayuda a controlar cuándo ocurren eventos en tu juego. Puedes programarlos para que sucedan después de ciertos intervalos de tiempo y crear una sensación de movimiento y emoción en el juego. Es como si estuvieras manejando el tiempo en tu mundo virtual para que todo suceda en el momento adecuado.

Por último cambiamos el nombre de los nodos Timer (Clic Derecho -> Renombrar, sobre los nodos), dejándolos de la siguiente forma

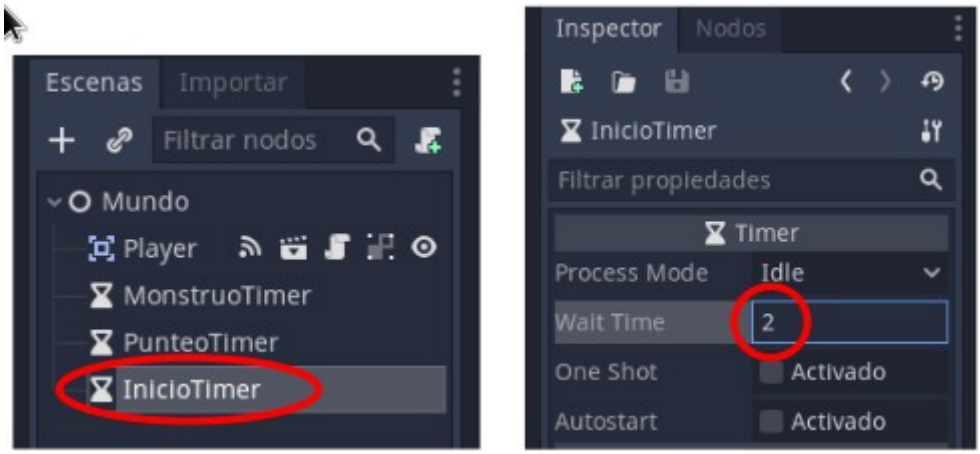


Cambiando valores iniciales de los Timer

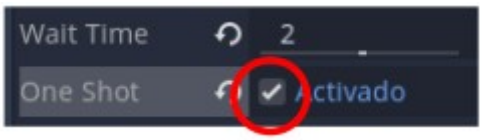
Con el nodo MonstruoTimer seleccionado, nos dirigiremos al **Inspector** y cambiaremos el atributo **Wait Timer** (tiempo de espera), asignándole el valor de 0.5 (para que cada medio segundo se vaya creando un monstruo nuevo).



El Wait Time de PunteoTimer se quedará igual, ya que queremos que el punteo vaya subiendo por cada segundo que el personaje sobreviva. Pero, el Wait Timer de InicioTimer si lo cambiaremos, asignando el valor de 2 (para que el tiempo de espera para reiniciar la partida al perder sea de 2 segundos)

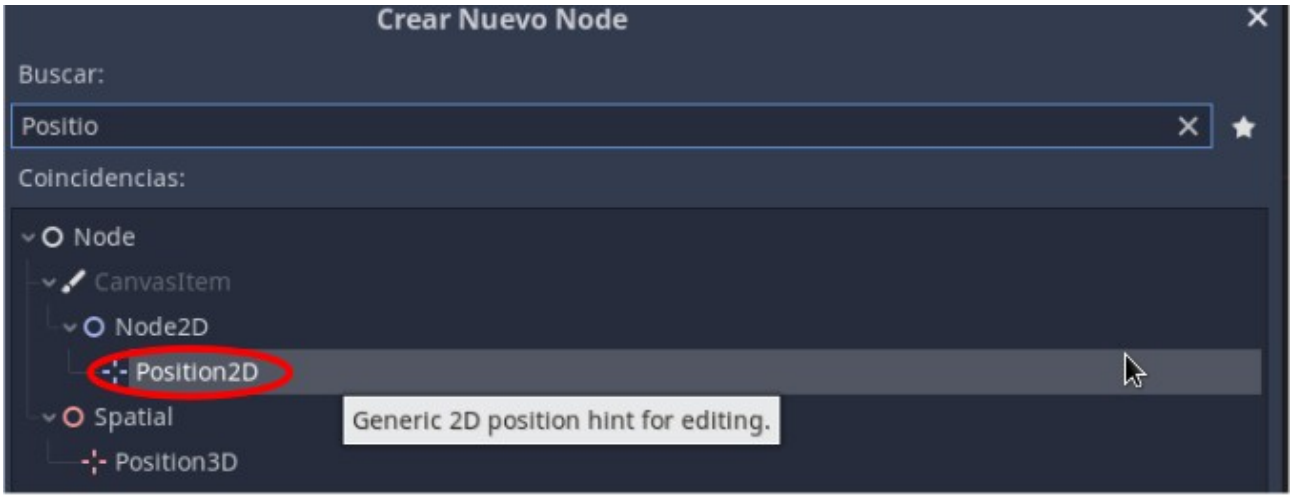


Y por último activamos el atributo OneShot (para que ya no se repita luego de activarse)



Nodo Psition2D

Ahora buscamos el nodo **Position2D** y lo agregamos con doble clic



Este servirá para poder indicar una posición en la pantalla (las coordenadas de inicio de nuestro juego).

Position2D

El nodo Position2D en Godot es como un marcador en un mapa que te dice dónde se encuentra algo en tu juego. Es como si estuvieras señalando una ubicación especial en un mundo virtual.

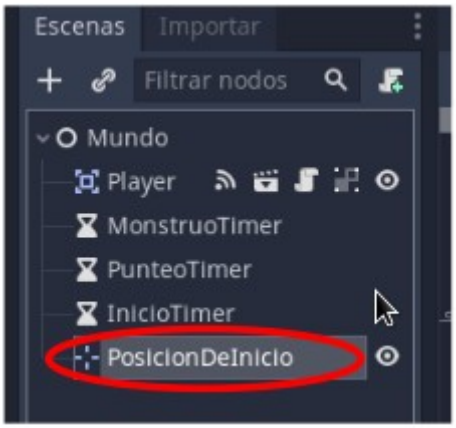
Descripción:

1. **Marcador de posición**: El nodo Position2D es como un marcador en tu juego. Puedes usarlo para indicar dónde se encuentra un objeto, como un personaje, un enemigo o un tesoro. Es como cuando pones una "X" en un mapa para mostrar dónde está algo importante.
2. **Ubicación en el espacio**: Este nodo representa una posición en el espacio del juego, pero no tiene una apariencia visual en sí mismo. Es como si fuera un punto en un mapa que te dice las coordenadas, pero no puedes verlo directamente.
3. **Coordenadas x e y**: El Position2D tiene coordenadas "x" e "y" que indican su posición en la pantalla del juego. Imagina que estás mirando un plano bidimensional y el Position2D te dice exactamente en qué parte de ese plano se encuentra algo.
4. **Usos diversos**: Puedes usar el nodo Position2D para muchas cosas. Puedes ubicar objetos en lugares específicos, calcular distancias entre puntos o incluso crear efectos visuales. Es como si estuvieras dejando pistas en tu mundo virtual para que los objetos y personajes sepan dónde deben estar.

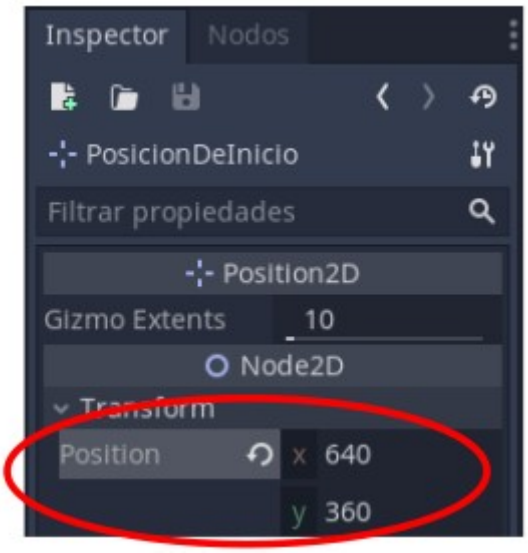
El nodo Position2D en Godot es como un marcador invisible en el juego que te ayuda a ubicar cosas y calcular distancias. Puedes usarlo para señalar lugares importantes y crear un sentido de

ubicación en tu mundo virtual. Es como si estuvieras colocando "etiquetas" en tu juego para saber dónde están las cosas.

Ahora lo renombramos como PosicionDeInicio



Por último, cambiaremos los valores de posición manualmente, de forma que nuestro Player aparezca exactamente en el centro de la pantalla (en el inspector cambiamos el valor de **X** por **640** y **Y** por **360**)



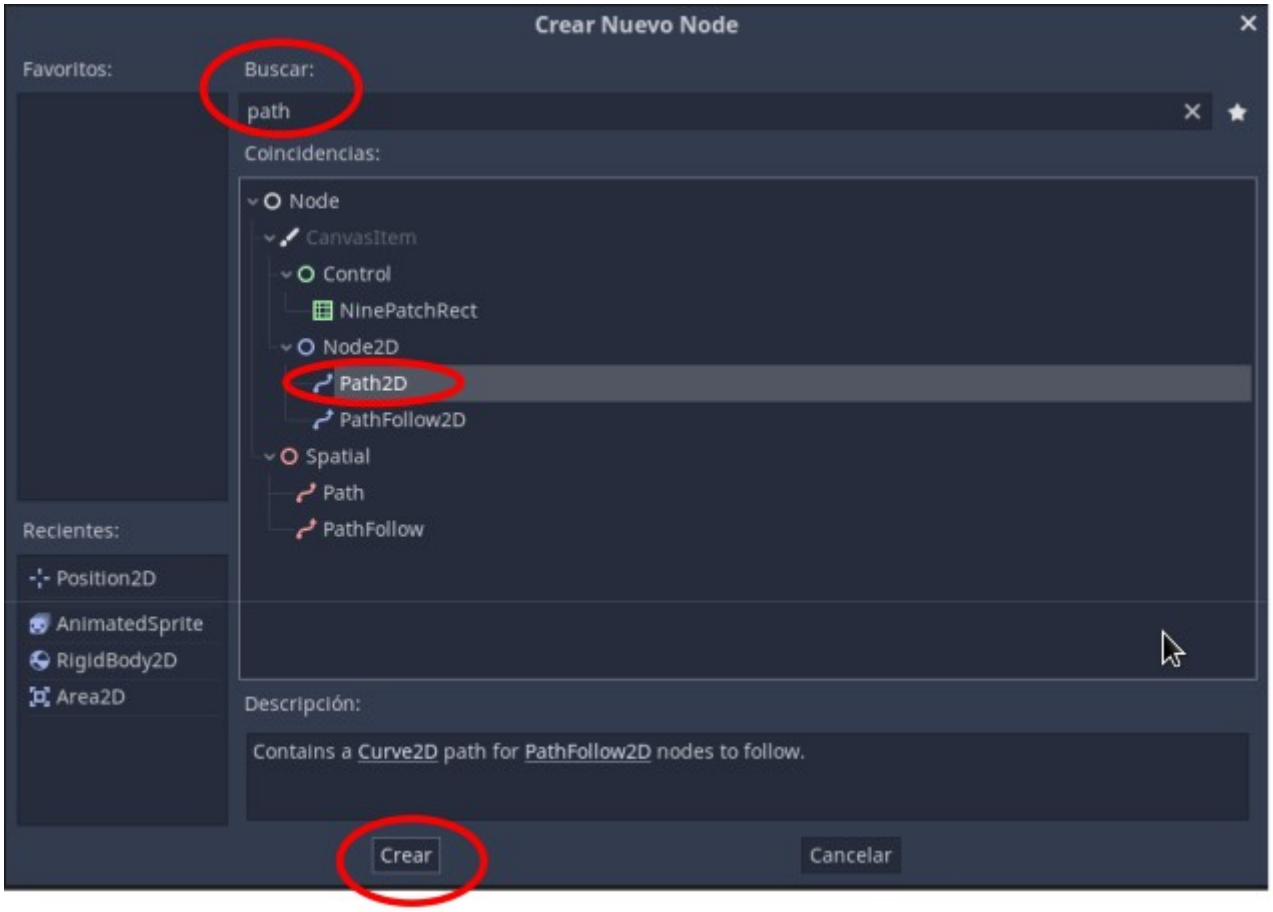
Estos valores os obtuvimos dividiendo a la mitad el tamaño de la ventana de nuestro juego (128/2=640 y 720/2=360).

Guardar todas las escenas.

Nodo Path2D

Con este nodo vamos a agregar un camino alrededor de la pantalla. Seleccionamos el nodo Mundo y agregamos un nuevo nodo hijo de mundo, con clic derecho

Y seleccionamos el nodo de tipo Path2D



Path2D

El nodo Path2D en Godot es como un camino trazado en el suelo que le dice a un personaje o a un objeto cómo moverse en el juego. Es como si estuvieras dibujando una ruta en un mapa para que algo siga ese camino.

Descripción:

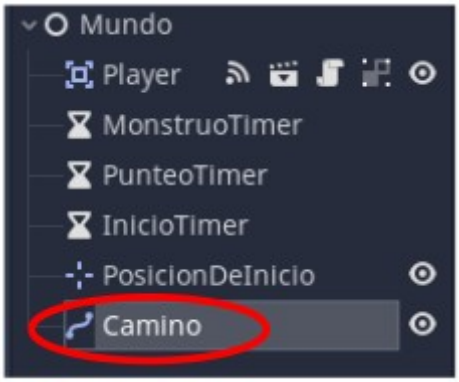
1. **Camino predefinido**: El nodo Path2D es como un camino ya definido en tu juego. Puedes crearlo dibujando puntos en el mapa que se conectan entre sí, como marcar lugares en un mapa con líneas que los unen.
2. **Guía de movimiento**: Imagina que tienes un personaje y quieres que siga una ruta

específica. Puedes usar el Path2D para guiar al personaje a lo largo de ese camino. Es como si estuvieras trazando una línea para decirle al personaje a dónde debe ir.

3. **Animaciones y efectos**: El Path2D también es útil para crear efectos visuales. Puedes hacer que objetos se muevan a lo largo del camino para crear animaciones o mostrar rutas en el juego.
4. **Interpolación**: A medida que un objeto sigue el camino, el Path2D puede calcular automáticamente cómo debe moverse entre los puntos intermedios. Esto se llama interpolación y ayuda a que el movimiento sea suave y realista.

El nodo Path2D en Godot es como un camino dibujado en el suelo que le indica a los objetos y personajes cómo moverse en el juego. Puedes usarlo para crear movimientos predefinidos, animaciones y efectos visuales. Es como si estuvieras creando rutas para tus personajes y objetos para que sigan en tu mundo virtual.

Cambiamos el nombre del nodo para facilitarnos su identificación, poniéndole por ejemplo “Camino”



Y ahora usamos GridSnap para ajustar a cuadrícula (botón con forma de imán junto a líneas de intersección)

Para ello, debemos seguir esta secuencia de clics:

1) Presionamos la opción Añadir punto (en espacio vacío), representado por un ícono de signo de + verde

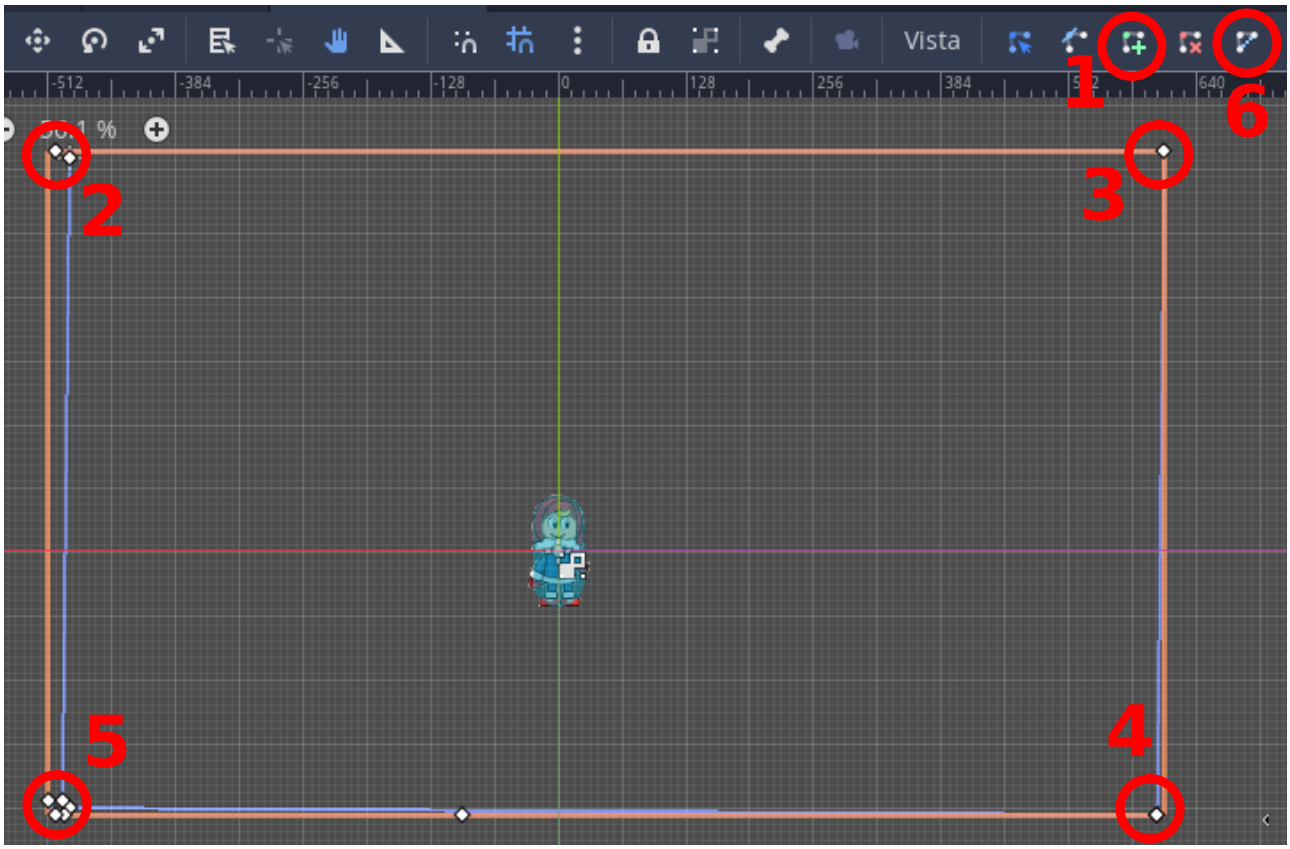
2) Damos clic en la esquina superior izquierda de los bordes de la pantalla de juego

3) Damos clic en la esquina superior derecha de los bordes de la pantalla de juego

4) Damos clic en la esquina inferior derecha de los bordes de la pantalla de juego

5) Damos clic en la esquina inferior izquierda de los bordes de la pantalla de juego

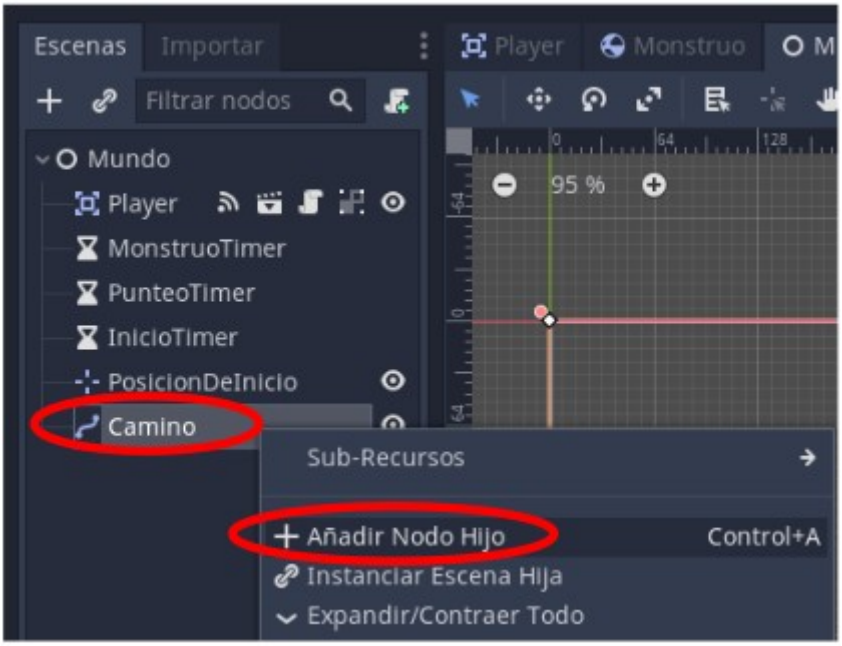
6) Finalizamos el camino con la opción Cerrar Curva



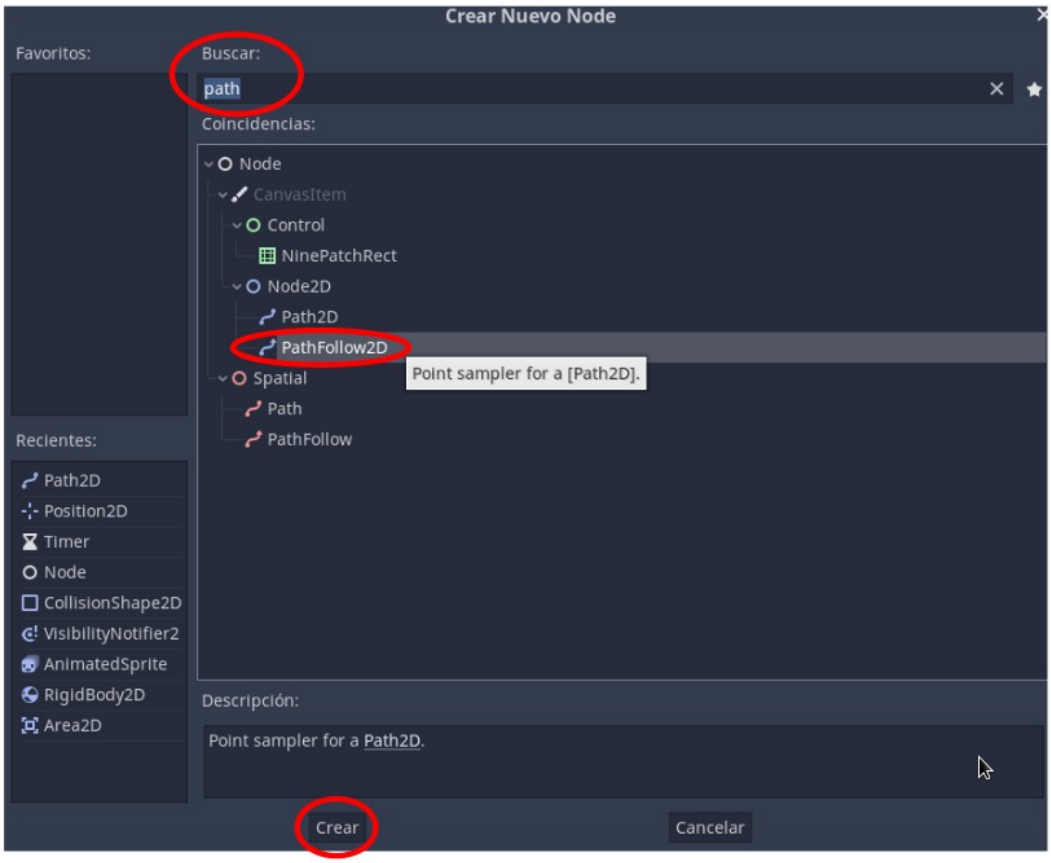
Dicho de otra forma, lo que hicimos fue trazar una ruta en toda la orilla de la pantalla en dirección de las agujas del reloj (si te confundes y lo haces en contra de las agujas del reloj, los enemigos se desplegarían fuera de los márgenes visibles, así que mucho OJO con esto).

Nodo PathFollow2D

Asegurándonos primero de que tenemos seleccionado el nodo Camino, daremos clic derecho y luego elegiremos la opción Añadir Nodo Hijo



El cual será de tipo **PathFollow2D**



PathFollow2D

El nodo PathFollow2D en Godot es como un seguidor mágico que se mueve a lo largo de un camino que tú has creado en el juego. Es como si estuvieras controlando un carrito en una pista de ferrocarril invisible.

Descripción:

1. **Seguimiento del camino**: El nodo PathFollow2D sigue un camino que tú has trazado utilizando el nodo Path2D. Imagina que has dibujado una ruta en el suelo con puntos conectados por líneas. El PathFollow2D es como un pequeño amigo que sigue esa ruta exacta.
2. **Control de movimiento**: Puedes controlar la velocidad a la que el PathFollow2D se mueve a lo largo del camino. Es como si estuvieras ajustando la velocidad del carrito en la pista.
3. **Movimiento suave**: A medida que el PathFollow2D se desplaza a lo largo del camino, el movimiento es muy suave. Se asegura de que el objeto que sigue el camino no dé saltos ni movimientos bruscos.
4. **Información del camino**: Puedes obtener información sobre dónde se encuentra el PathFollow2D en relación con el camino. Puedes saber cuánta distancia ha recorrido o incluso obtener la posición exacta en cualquier punto del camino.
5. **Creatividad y efectos**: El PathFollow2D es una forma genial de crear movimientos específicos para objetos y personajes en tu juego. Puedes usarlo para hacer que algo siga una trayectoria curva o animada.

El nodo PathFollow2D en Godot es como un seguidor que viaja por un camino que tú has creado. Te ayuda a controlar el movimiento suave y constante a lo largo de ese camino, permitiéndote crear movimientos precisos y efectos visuales interesantes en tu juego.

¿Cuál es la diferencia entonces entre Path2D y PathFollow2D?

La diferencia principal entre los nodos `Path2D` y `PathFollow2D` en Godot es su función y cómo se utilizan en el juego.

- **Path2D**: El nodo `Path2D` es el que usas para crear el camino en sí. Puedes agregar puntos al `Path2D` para definir la ruta que deseas que un objeto o personaje siga. Imagina que estás dibujando un camino en un mapa. El `Path2D` contiene la información sobre dónde deben estar los puntos en ese camino y cómo se conectan. Es como trazar el recorrido.
- **PathFollow2D**: Por otro lado, el nodo `PathFollow2D` es como un seguidor que se desplaza a lo largo del camino que has creado con el `Path2D`. Puedes conectar el `PathFollow2D` al

`Path2D` y ajustar la velocidad a la que se moverá. Es como poner un coche en la pista que has dibujado. El `PathFollow2D` sigue el camino que has trazado en el `Path2D` de manera suave y constante.

Cambiaremos el nombre de este nuevo nodo por "**MonstruoPosicion"**

(Este nodo servirá para la posición del camino en la cual se creará el monstruo)

GdScript

Preparando la configuración de nuestro personaje

En la escena de nuestro personaje player vamos a agregar algunas configuraciones a nuestro código y algunas conexiones de señales.

Ocultando el personaje

Ingresamos al Script y agregamos la línea de código **hide()** dentro de la función **_ready**, tal como se muestra a continuación:

```
func _ready():
    #Para que el personaje comience el juego apagado
    hide()
```

De esta forma al iniciar el juego el personaje estará oculto.

Agregando una señal

Declaramos una señal para identificar cuando un enemigo choque a nuestro personaje principal, agregando la línea de código signal golpe (debajo de la declaración de la variable limite), tal como se muestra a continuación.

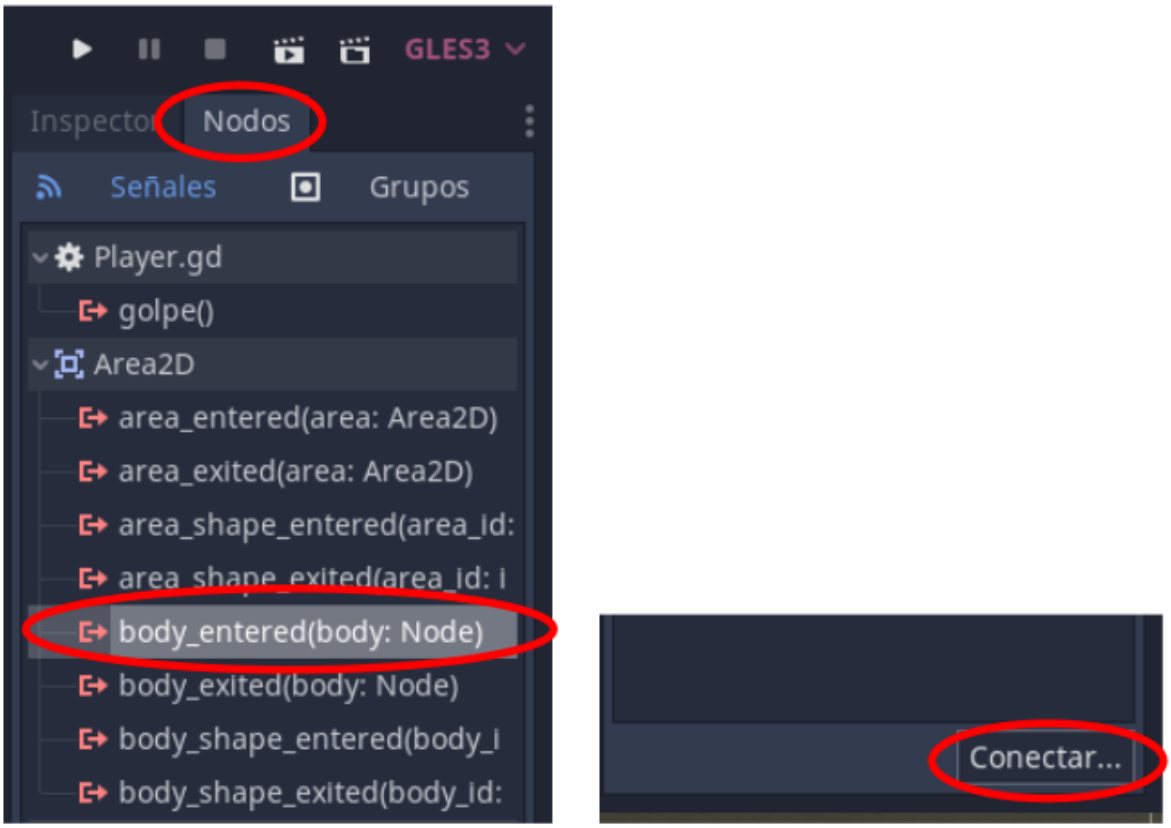
```
var Movimiento = Vector2()
var limite
```

```
signal golpe #identifica cuando un enemigo nos choca
```

Activando la señal cuando exista colisión.

Seleccionamos el nodo **Player**. Ya con dicho nodo seleccionado, nos dirigiremos al panel de Nodos (al lado del Inspector), identificamos y seleccionamos la señal de Area2D llamada **body_entered(body: Node)** y damos un clic en el botón de **Conectar…**

(este proceso es el mismo que hicimos en la configuración de los botones de la calculadora)



Luego de dar clic en Conectar… se mostrará la ventana de conexión de la señal, en la cual podemos elegir el Script que enlazaremos, el nombre del método receptor y otros valores, pero, en esta ocasión dejaremos los valores predeterminados y daremos clic en el botón **Conectar.**

Notarás que automáticamente se agregará el siguiente código al Script:

```
func _on_Player_body_entered(body):
      pass
```

Esta función lo que nos ofrece es un espacio para establecer las sentencias que deben cumplirse cuando se emita la señal de que un enemigo entró en contacto con nuestro personaje.

Escribiendo código para la colisión.

Borramos la línea de código **pass # Replace with function body** y escribimos en su lugar **hide()** para que se desaparezca el personaje cuando se detecte la colisión.

Ahora debajo agregamos la función **emit_signal("golpe")** para que se emita la señal. También desactivaremos la forma de colisión para que deje de detectar choques mediante **$CollisionShape2D.disabled = true**

```
func _on_Player_body_entered(body):
      hide() #para que desaparezca personaje cuando detecte colision
```

```
	emit_signal("golpe")
	$CollisionShape2D.disabled = true
```

Creando la función para la posición inicial.

En una línea en blanco por debajo del código que llevamos hasta el momento y borrando previamente la sangría, escribimos la función **func inicio(posi):** en la cual estaremos agregando **posi** como variable para establecer la posición del personaje.

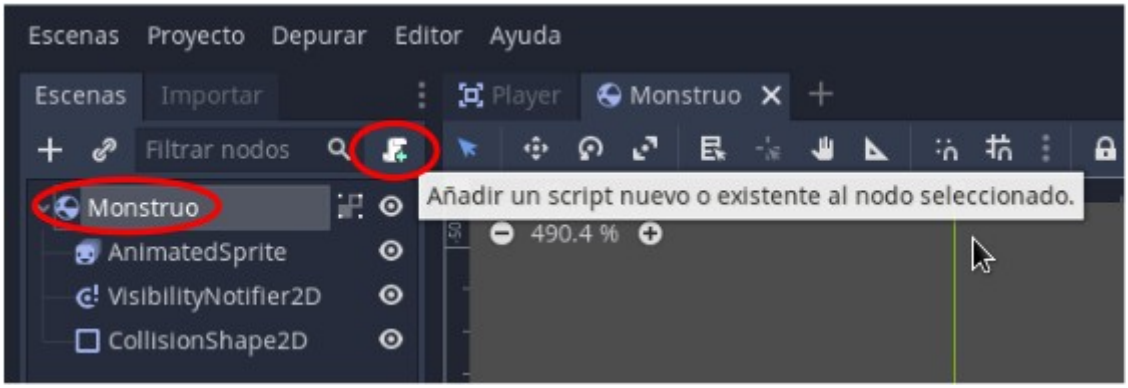
También agregamos el código **show()** para que vuelva a mostrar al personaje al volver a iniciar y, contrario al paso anterior, ahora negamos la opción de des-habilitación de la forma de colisión con el código **$CollisinShape2D.disable = false**

```
#Funcion para la posicion inicial
func inicio(posi):
	position = posi
	show()
	$CollisionShape2D.disabled = false
```

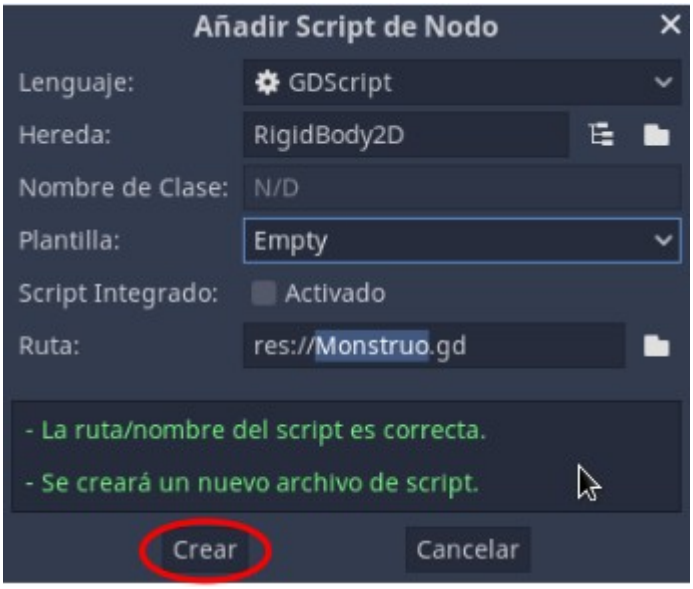
Y con esto finalizamos toda configuración de nuestro personaje principal.

Creando Script para Enemigo

Con el nodo Monsturo seleccionado, presionamos el botón de añadir script:



Y en la ventana dejamos los valores predeterminados, excepto el de **Plantilla**, en el cual seleccionaremos **Empty** (esto servirá para que nuestro script inicie completamente vacío, sin las funciones predefinidas ni ningún otro código)



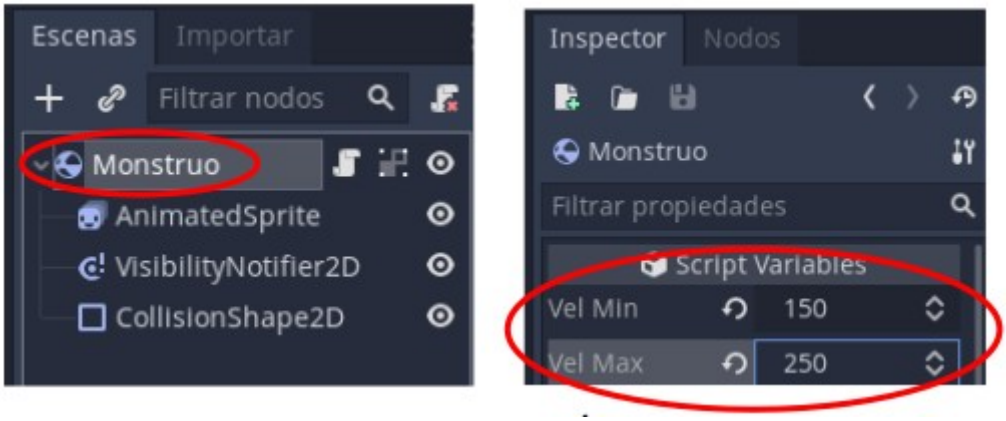
Exportando variables par la velocidad

En el nuevo scrpit, debajo de la línea de código del Extends, exportaremos dos variables de tipo entero (int), una que servirá para nuestra velocidad mínima (vel_min) y otra para nuestra velocidad máxima (vel:max), tal como se muestra a continuación:

```
extends RigidBody2D


export (int) var vel_min
export (int) var vel_max
```

Luego de esto podrás visualizar los atributos **Vel Min** y **Vel Max** en el inspector. Ahora solo agregaremos valores en estos dos atributos y para ello (con el nodo raíz Monstruo seleccionado) colocaremos **150** como Velocidad Mínima y **250** como Velocidad Máxima (puedes cambiar estos valores a tu gusto, según lo consideres conveniente)



Declarando la variable que contenga ambas versiones de nuestro monstruo

Escribimos la siguiente línea de código:

```
export (int) var vel_min
export (int) var vel_max
var tipo_monstruo = ["enemigo", "mini"]
```

Y notarás que dentro de corchetes tenemos las dos animaciones de nuestro monstruo, donde “enemigo” es la versión de tamaño normal y “mini” es la versión más pequeña del mismo enemigo. I**MPORTANTE**: Verifica que los valores “enemigo” y “mini” coincidan con los nombres de tus animaciones en el panel de SpriteFrames.

Creando la función _ready()

Agregamos el código para iniciar la función _ready() y dentro de la misma incluimos una línea de código para hacer un llamado al nodo AnimatedSprite, específicamente a su animación según un dato aleatorio (randi) de las opciones almacenadas en la variable de tipo array creada en el paso anterior

```
func _ready():
      $AnimatedSprite.animation = tipo_monstruo[randi() % tipo_monstruo.size()]
```

randi()

La función randi() en GDScript es una herramienta que permite generar números enteros aleatorios en un rango especificado. Este concepto es similar a arrojar un dado o seleccionar una tarjeta al azar. La función randi() puede ser utilizada en programas de juegos para introducir elementos de aleatoriedad y variabilidad. Cuando se llama a randi(), se proporciona un rango, que incluye un valor mínimo y un valor máximo. La función entonces devuelve un número entero aleatorio que cae dentro de ese rango, lo que brinda la oportunidad de tomar decisiones no predecibles en el desarrollo de juegos.

El valor generado va entre 0 y 2^32-1. Imagina que estás jugando con un dado que tiene muchos lados, en este caso, ¡realmente tiene más de 4 mil millones de lados! Eso significa que puede dar muchos números diferentes. Sin embargo, a veces quieres limitar el número para que no sea tan grande. Aquí es donde entra en juego el operador de módulo (%).

El operador de módulo (%) es como un reloj que cuenta hasta cierto número y luego comienza de nuevo desde cero. En este caso, puedes usar el operador de módulo para mantener el número dentro de un rango específico. Por ejemplo, si aplicas `% 6` a un número aleatorio, el resultado solo puede ser 0, 1, 2, 3, 4 o 5. Es como si el dado tuviera solo 6 lados.

Cambiando el tamaño de la colisión en base a una condición

Con el siguiente código, haremos que la forma de colisión cambie dependiendo de cuál versión del monstruo se ha generado (ya sea el enemigo de tamaño normal o su versión mini)

```
func _ready():
	$AnimatedSprite.animation = tipo_monstruo[randi() % tipo_monstruo.size()]


	if $AnimatedSprite.animation == "enemigo":
		$CollisionShape2D.scale.x = 1.4
		$CollisionShape2D.scale.y = 1.4
```

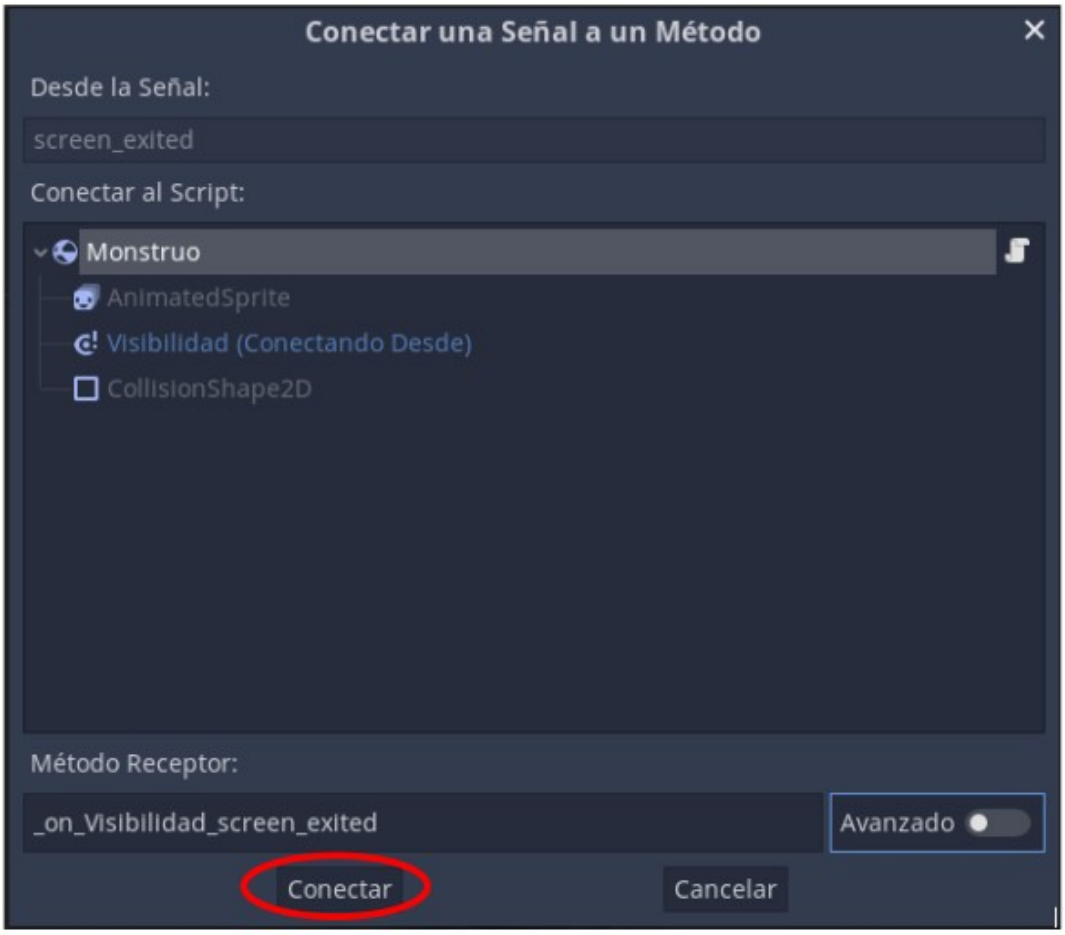
IMPORTANTE: Los valor de 1.4 pueden modificarse al gusto, según sea necesario, lo importantes es ir haciendo pruebas y buscando el valor que mejor se adapte.

Trabajando con el nodo VisibilityNotifier2D

Este nodo nos indicará si nuestro enemigo está apareciendo en pantalla (y podremos hacer que deje de aparecer). Lo primero que haremos será cambiarle el nombre a “Visibilidad” para facilitar su manejo.

Ahora conectaremos la señal que indicará cuando el objeto salga de la pantalla. Para ello, teniendo seleccionado el recién renombrado nodo Visibilidad, nos dirigimos al área de Nodos que aparece en una pestaña al lado de Inspector y daremos doble clic sobre la señal screen_exited()

Y en la siguiente ventana dejamos los valores predeterminados, finalizando con el botón Conectar.



Por último, notarás que se agregó un código al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal.

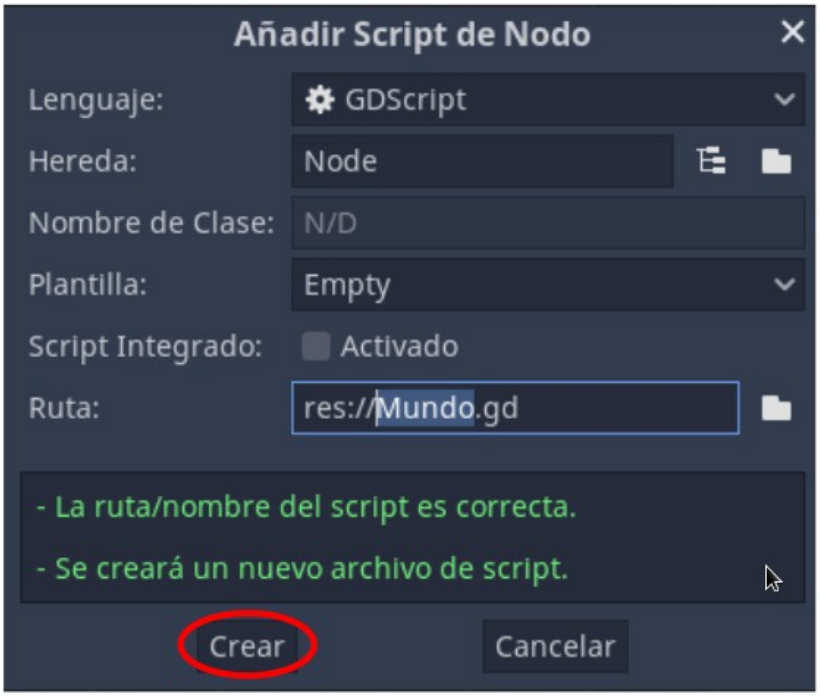
```
func _on_Visibilidad_screen_exited():
      pass # Replace with function body.
```

Por el momento no tocamos nada aca.

Creando Scrpt para el Mundo

Con el nodo Mundo seleccionado, presionamos el botón de añadir script (pergamino blanco con signo + de color verde)

Y en la ventana dejamos los valores predeterminados, incluyendo el de Plantilla, que debería de estar Empty.



Exportando una variable para el paquete de escena

En el nuevo scrpit, debajo de la línea de código del Extends, exportaremos una variable de tipo Paquete de Escena (PackedScene), a la cual le llamaremos **Monstruo**, tal como se muestra a continuación

```
extends Node

export (PackedScene) var Monstruo
```

PackedScene

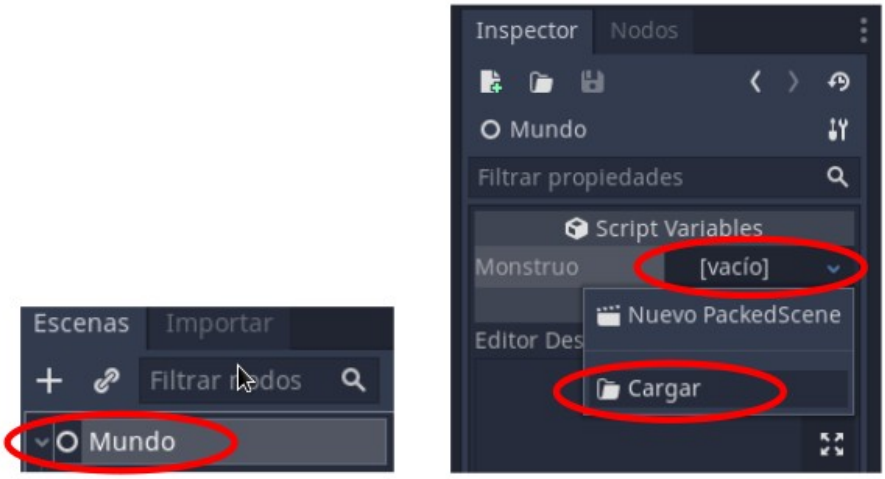
El paquete **`PackedScene`** en Godot es como un conjunto de instrucciones que describe cómo se debe crear y configurar un objeto o una escena en tu juego. Puedes pensar en él como un "kit de construcción" para crear elementos en el mundo virtual. Contiene toda la información necesaria para crear un objeto o una escena, como sus nodos, propiedades, configuraciones y relaciones.

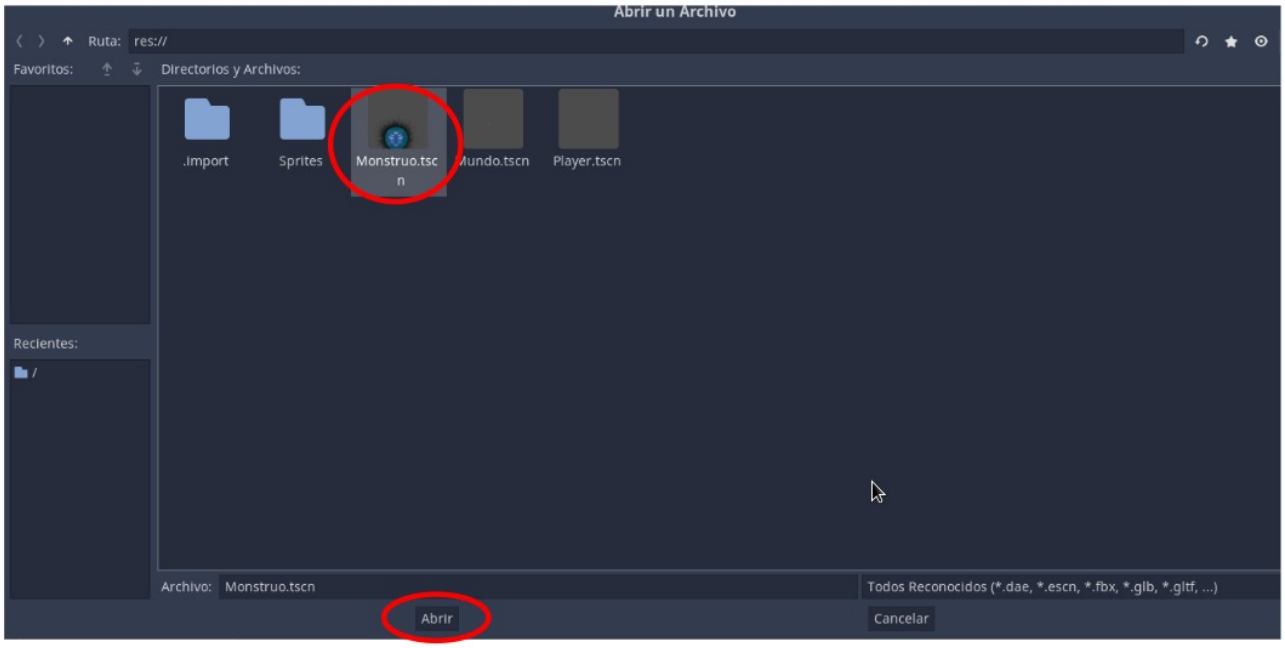
Imagina que estás armando una figura de Lego. Cada pieza de Lego es como un nodo en tu juego, y el paquete **`PackedScene`** sería como un manual de instrucciones que te muestra cómo ensamblar todas esas piezas juntas para crear una estructura completa. Este manual te dice qué piezas usar, cómo conectarlas y cómo configurarlas.

El paquete **`PackedScene`** es un archivo que almacena las instrucciones necesarias para construir un objeto o una escena en tu juego. Te permite crear elementos predefinidos y luego instanciarlos en tu juego cuando los necesites. Es una herramienta poderosa para gestionar y reutilizar contenido en tu proyecto.

Luego de esto, seleccionamos el nodo Mundo en el área de Escenas y nos dirigiremos al Inspector, en donde debemos ubicar la recién exportada variable llamada **Monstruo** con un valor **[Vacío]**, cambiando dicho valor

Vacío al **Cargar la escena Monstruo.tscn** (la cual utilizaremos más adelante para crear los enemigos)





Declarando la variable para el punteo

Escribimos la siguiente línea de código

```
extends Node
export (PackedScene) var Monstruo
var Punteo
```

(Esta variable es la que nos servirá para ir guardando la sumatoria de los puntos obtenidos por el usuario que juegue con nuestro videojuego)

Creando la función _ready() y la función Randomize()

Agregamos el código para iniciar la función _ready() y dentro de la misma incluimos una línea de código para hacer iniciar la función randomize(), la cual servirá para que cada vez que se ejecute el juego el patrón de creación enemigos será distinto.

```
func _ready():
	randomize()
```

Configurando la función para un NUEVO JUEGO

Agregamos el código para crear la función **nuevo_juego()**, la cual será una función inventada por nosotros mismos (o sea que no es de las predeterminadas que ofrece Godot). También estableceremos el valor de nuestra variable **Punteo** en **0** (que es como debe iniciar)

```
func nuevo_juego():
	Punteo = 0
```

Ahora haremos un llamado al nodo Player con referencia a la función inicio y estableceremos dicho inicio en la PosiciónDeInicio

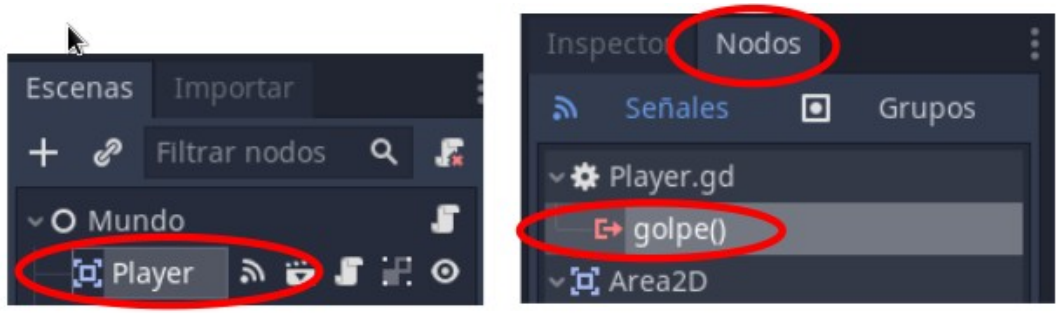
```
func nuevo_juego():
	Punteo = 0
	$Player.inicio($PosicionDeInicio.position)
```

Por último, activaremos el Timer llamado InicioTimer

```
func nuevo_juego():
	Punteo = 0
	$Player.inicio($PosicionDeInicio.position)
	$InicioTimer.start()
```

Identificando cuando el juego finalizó (el Player fue golpeado)

Estando todavía en la escena Mundo, seleccionamos la Instancia **Player** (que está como nodo hijo de mundo) y nos dirigimos a la **pestaña Nodos** que se encuentra al lado del Inspector, para conectar la señal **golpe()** con doble clic sobre la misma



Pero, antes de finalizar la conexión, cambiamos el nombre de la misma colocando "**game_over**" con el único objetivo de facilitar la comprensión del código que se generará automáticamente

Y notarás que se agregó un código de forma automática al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal

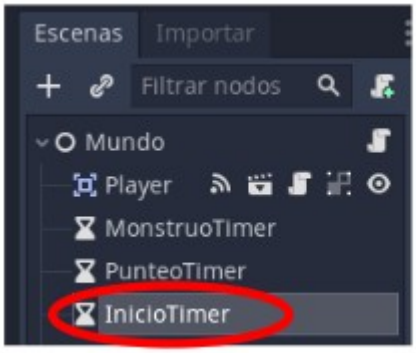
```
func game_over():
	pass # Replace with function body.
```

Empezamos borrando la línea **pass**. Luego detendremos los temporizadores **PunteoTimer** y **MonstruoTimer**, con las siguientes líneas de código

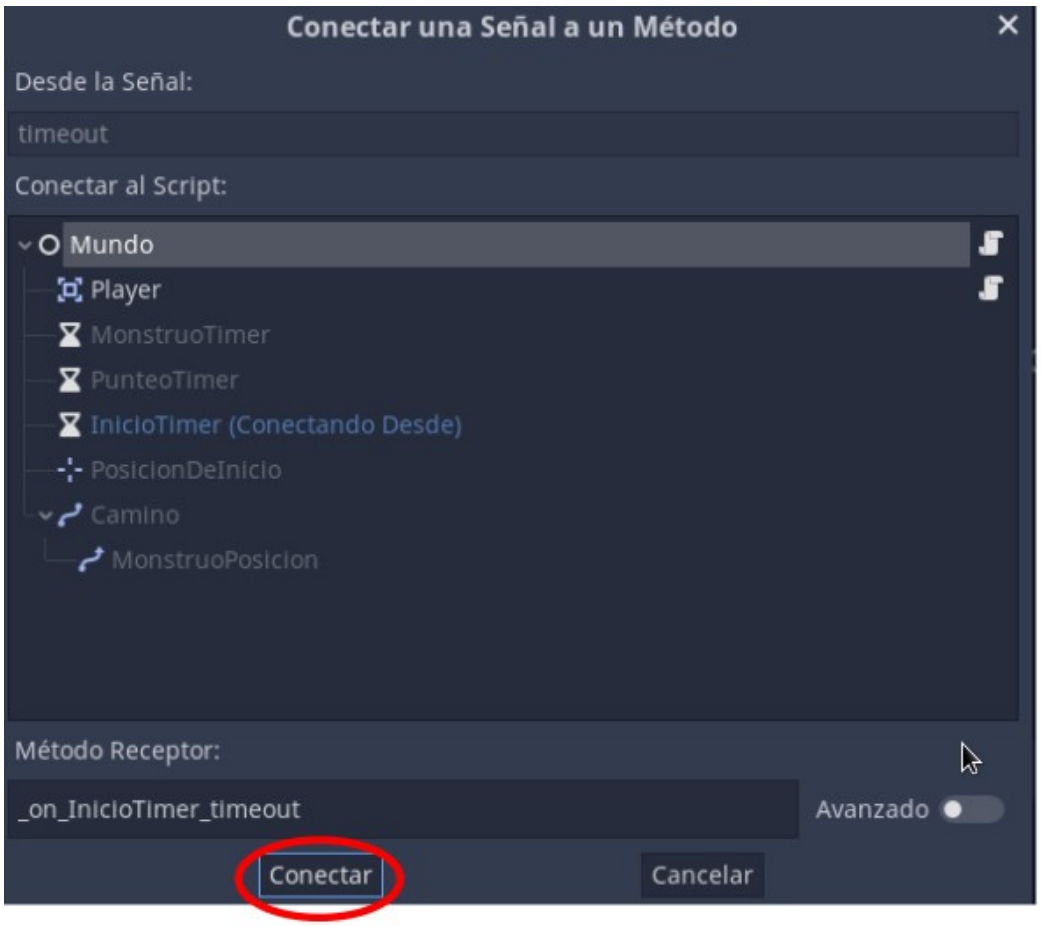
```
func game_over():
	$PunteoTimer.stop()
	$MonstruoTimer.stop()
```

Conectando la señal timeout() con el Inicio

Con el nodo **InicioTimer** seleccionado nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, de la forma que ya aprendimos, el nodo **timeout()**, el cual servirá para que pronto podamos definir qué queremos que pase cuando dicho tiempo termine.



Y finalizamos la conexión, esta vez sin cambiar ningún parámetro predeterminado



Notarás que también se agregó un código de forma automática al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal.

```
func _on_InicioTimer_timeout():
	pass # Replace with function body.
```

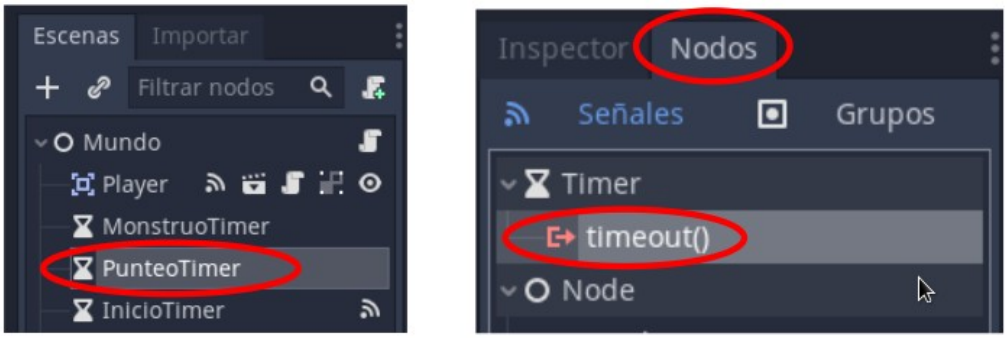
Configurando la función que iniciará nuestro nuevo juego

Empezamos borrando la línea **pass**. Luego inicializaremos los temporizadores **MonstruoTimer** y **PunteoTimer**, con las siguientes líneas de código.

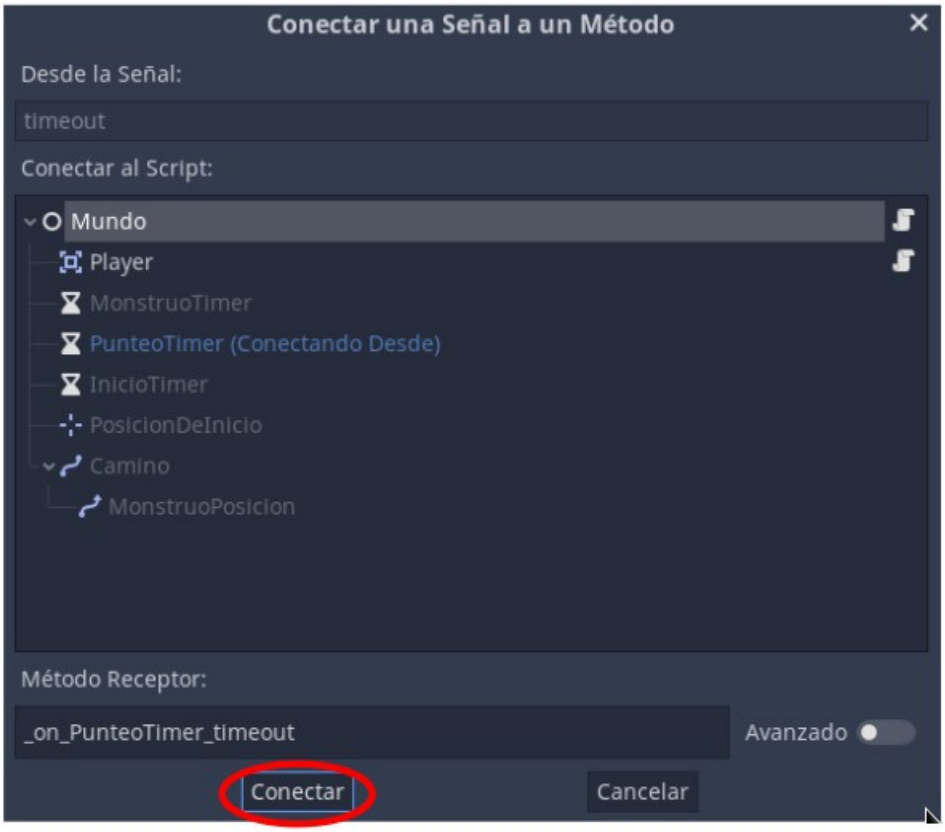
```
func _on_InicioTimer_timeout():
	$MonstruoTimer.start()
	$PunteoTimer.start()
```

Conectando la señal timeout() con el Punteo

Con el nodo **PunteoTimer** seleccionado nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, tal como lo aprendido en pasos anteriores, el nodo **timeout()**, el cual servirá para que pronto podamos definir qué queremos que pase cuando se emita esta señal



Y finalizamos la conexión, de nuevo sin cambiar ningún parámetro predeterminado



Notarás que nuevamente se agregó un código de forma automática al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal

```
func _on_PunteoTimer_timeout():
	pass # Replace with function body.
```

Configurando la función que aumentará el punteo cada segundo

Empezamos borrando la línea **pass**. Ahora indicamos la suma de 1 valor al Punteo (así cada segundo que el jugador permanezca vivo se sumará a su punteo), esto lo lograremos con la siguiente línea de código

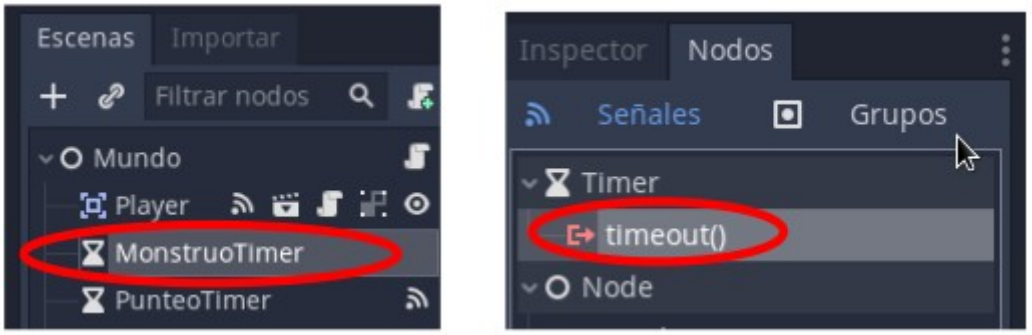
```
func _on_PunteoTimer_timeout():
	Punteo += 1
```

IMPORTANTE: Realmente puedes elegir los nombres que quieras para las funciones creadas en los últimos pasos, pero, en este taller se trató de mostrar la variedad de posibilidades que existen al

momento de crear y nombrar una función, al final cada programador puede desarrollar sus propios criterios y su propia lógica.

Conectando la señal timeout() con el Monstruo

Con el nodo **MonstruoTimer** seleccionado nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, tal como lo hemos hecho antes, el nodo timeout(), el cual servirá para que pronto podamos definir qué queremos que pase cuando se emita esta señal.



Y finalizamos la conexión, de nuevo sin cambiar ningún valor predeterminado.

De nuevo se agregará un código automáticamente al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal

```
func _on_MonstruoTimer_timeout():
	pass # Replace with function body.
```

Configurando la función que generará a los enemigos

Empezamos borrando la línea **pass**. Ahora realizaremos un llamado al nodo de MonstruoPosicion a través del Camino, también lo haremos con el atributo set_offset seleccionando una posición aleatoria en el path, básicamente estaremos indicando un lugar random en el camino para la aparición de un enemigo. Esto lo lograremos con el siguiente código.

```
func _on_MonstruoTimer_timeout():
	$Camino/MonstruoPosicion.set_offset(randi())
```

(La diagonal entre ambos nodos es para hacer un llamado a nodo hijo a través del padre)

Creando el Monstruo instantáneo

Como ya definimos un lugar aleatorio para su aparición, ahora debemos generar el enemigo. Iniciamos declarando la variable temporal **M** (de Monstruo) y la igualaremos a la instancia del Monstruo. Luego, agregaremos este mismo nodo como un hijo (ya que no será ejecutado hasta ser parte de la escena). Todo esto lo lograremos con el siguiente código

```
func _on_MonstruoTimer_timeout():
	$Camino/MonstruoPosicion.set_offset(randi())
	var M = Monstruo.instance()
	add_child(M)
```

Creando la dirección de los enemigos

Al final de lo que llevamos escrito hasta el momento en nuestro Script, declaramos la variable **dire** (que servirá para definir la dirección en que se moverán los Monstruos) y la igualaremos al atributo **rotation** (rotación) del nodo hijo **Camino/MonstruoPosición**. Luego, en una siguiente línea de código, cambiaremos el atributo **position** de la recién creada variable **M** igualándolo a la posición del nodo hijo **Camino/MonstruoPosición**. Esto quedaría desarrollado de la siguiente forma

```
func _on_MonstruoTimer_timeout():
	$Camino/MonstruoPosicion.set_offset(randi())
	var M = Monstruo.instance()
	add_child(M)

	var dire= $Camino/MonstruoPosicion.rotation + PI /2
	M.position = $Camino/MonstruoPosicion.position
```

Una vez realizado lo anterior, agregaremos (+=) a la variable dire, un valor de tipo rand_range (rango aleatorio) que contenga entre sus parámetros los radianes de la siguiente forma: (-PI /4, PI /4), ya que Godot no utiliza como unidad de medida los grados, por lo tanto, si yo quiero representar 45 grados en la rotación, debo hacerlo con su equivalente en radianes, que es: π/4

Ahora sí, pongamos manos a la obra al código que hará posible todo lo indicado anteriormente

```
	var dire= $Camino/MonstruoPosicion.rotation + PI /2
	M.position = $Camino/MonstruoPosicion.position

	dire += rand_range(-PI /4, PI /4)
	M.rotation = dire
```

Definiendo la velocidad de los enemigos

Estableceremos una velocidad lineal, pero de forma aleatoria, con un vector cuyos parámetros sean tomados de un rango aleatorio entre la velocidad mínima y la velocidad máxima (establecidas anteriormente entre 150 y 250) para X y simplemente con un 0 para Y (ya que esta posición no nos interesa cambiarla). Finalmente agregamos un parámetro para que la rotación se genere según la dirección en la que va. El código quedaría de la siguiente forma:

```
	dire += rand_range(-PI /4, PI /4)
	M.rotation = dire
	M.set_linear_velocity(Vector2(rand_range(M.vel_min,M.vel_max),0).rotated(dire))
```

Menu principal del Juego

¡Llegamos hasta aca! ¡Animo! ¡Y todavía no pudimos probar nada de esta parte del taller!. ¡No te preocupes! Aca es en donde se pone bueno!

Vamos a comenzar en armar el menú principal del juego, para ellos .. otra vez, vamos con otra escena.

Escena Interfaz

Agregaremos una nueva escena, la cual albergará nuestro Menú. Para ello, iniciamos dando clic en el menú Escena y luego en la opción Nueva Escena

Nodo CanvasLayer

En la escena recién creada, agregamos un nodo de tipo **CanvasLayer** y lo renombramos como nombre **Interfaz**, para facilitar su manejo.

CanvasLayer

el nodo **CanvasLayer** en Godot es como una hoja transparente en la que puedes dibujar cosas sobre la pantalla del juego. Es como si estuvieras colocando papel encima de la imagen de tu juego y pintando en él.

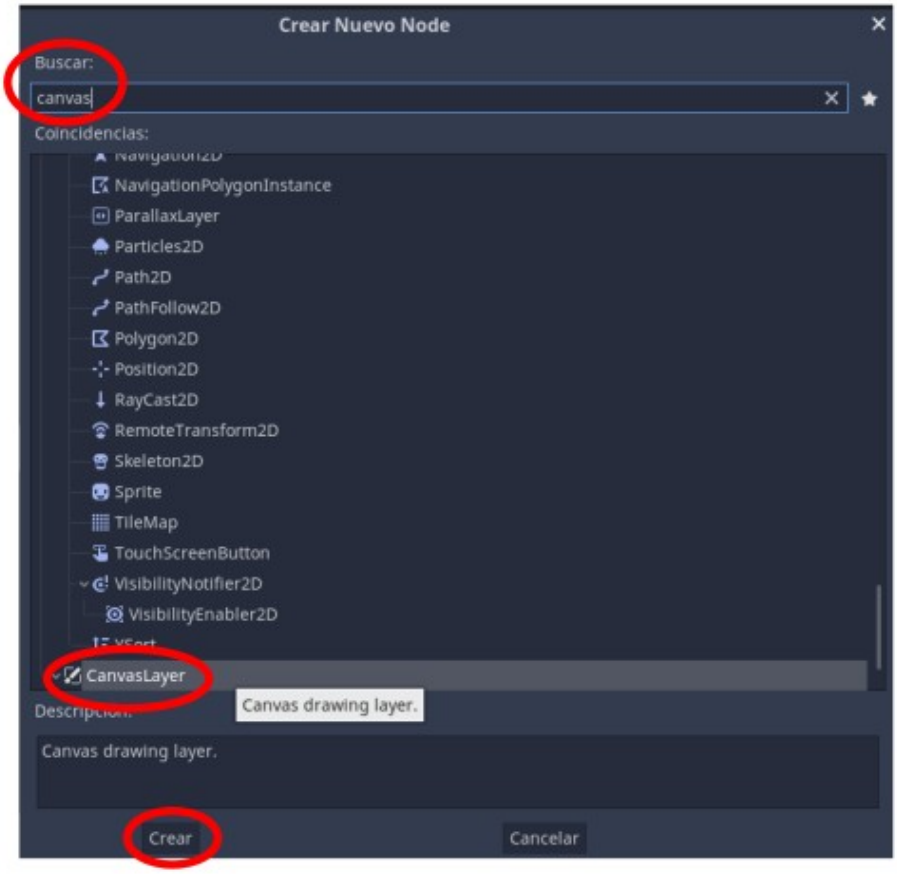
Descripción:

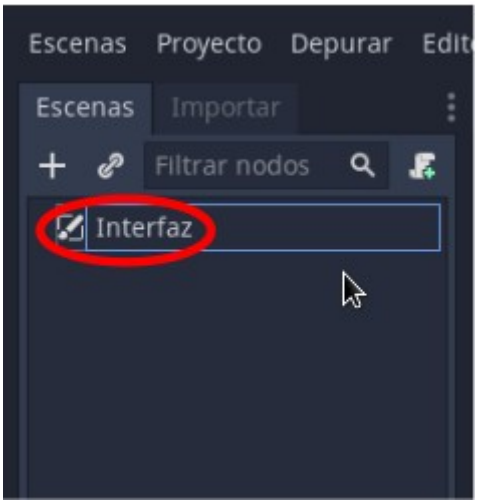
1. **Capa adicional**: El nodo `CanvasLayer` te permite agregar una capa extra a tu juego. Esta capa está "encima" de todo lo demás en la pantalla, como si pusieras un cristal transparente sobre la imagen del juego.
2. **Dibujo y diseño**: En esa capa, puedes dibujar cosas o colocar elementos visuales, como si estuvieras usando un programa de dibujo. Esto es útil para crear interfaces de usuario, menús, indicadores o efectos visuales especiales.
3. **Orden de visualización**: Puedes controlar en qué orden aparecen las capas en pantalla. Por ejemplo, si colocas un `CanvasLayer` sobre otro, lo que dibujes en el primero aparecerá

encima del segundo. Es como si tuvieras láminas transparentes y pudieras cambiar su orden.

4. **Interacción con el juego**: Los objetos en un `CanvasLayer` pueden interactuar con el juego, como botones que se pueden presionar o elementos que responden a los clics del mouse.

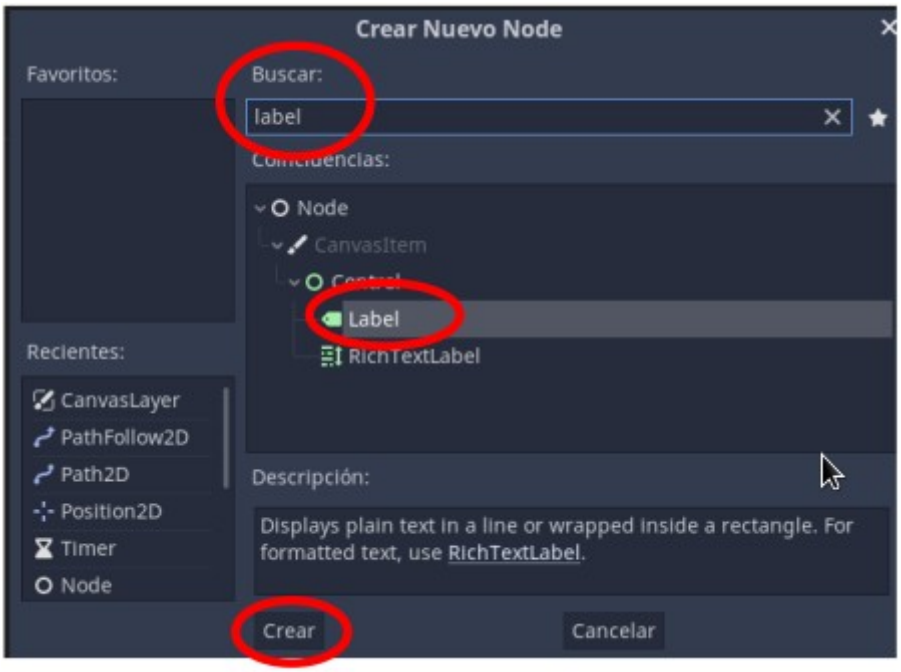
El nodo `CanvasLayer` en Godot te permite dibujar y mostrar elementos adicionales sobre la pantalla del juego. Es como trabajar con capas transparentes para agregar elementos visuales, interfaces de usuario y efectos especiales. Es una forma de personalizar y enriquecer la apariencia de tu juego.



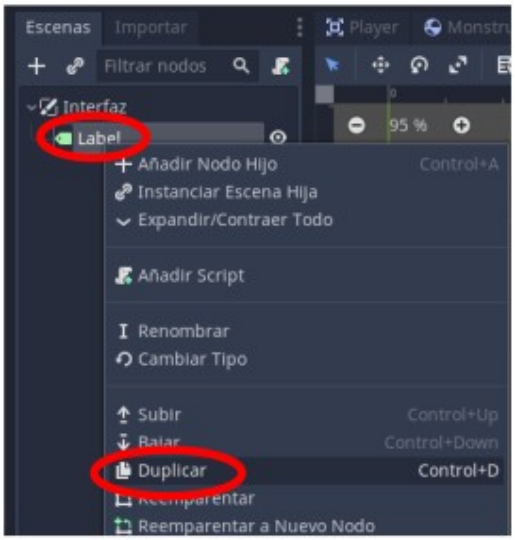


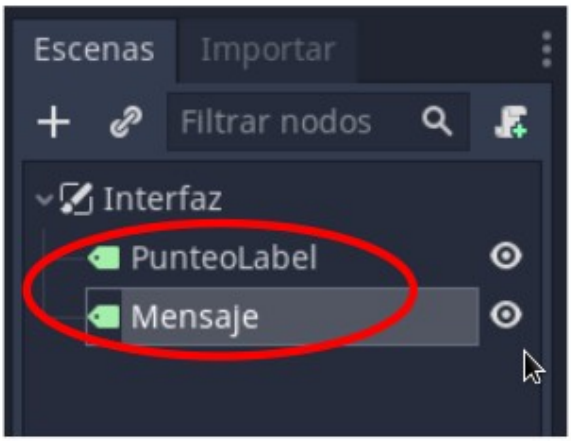
Nodos Label

Para mostrar un texto necesitaremos una etiqueta, es decir, un nodo **Label** (como hijo de interfaz). Este nodo lo añadiremos como hijo



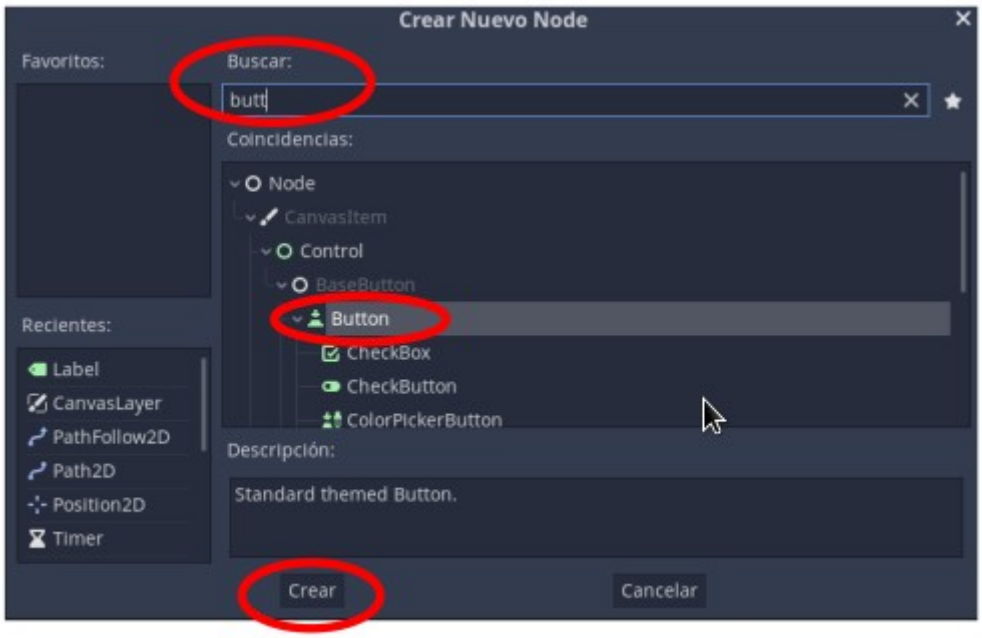
Ahora lo duplicamos y renombramos ambos: PunteoLabel y Mensaje





Nodo Button

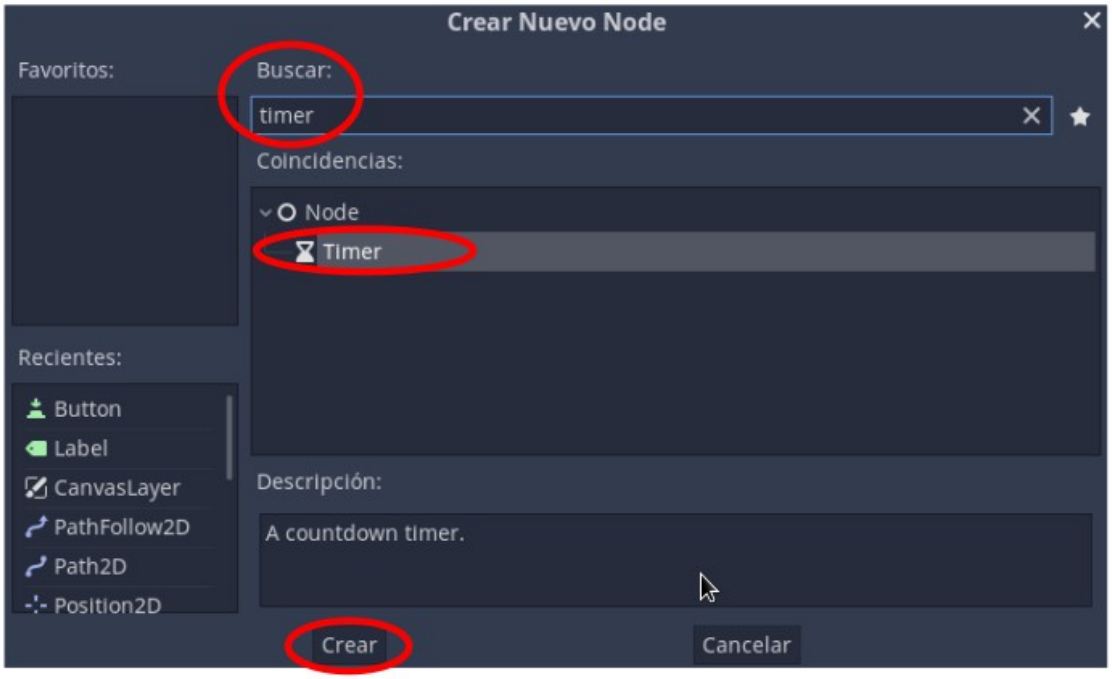
Para la desencadenar el inicio del juego a través de una acción de usuario necesitaremos agregar un botón, es decir, un nodo (hijo de interfaz) **Button**



Antes de continuar, recordemos que no se ha guardado esta nueva escena. Demos clic en el menú Escena y luego Guardar Escena, dejando el nombre predeterminado.

Nodo Timer

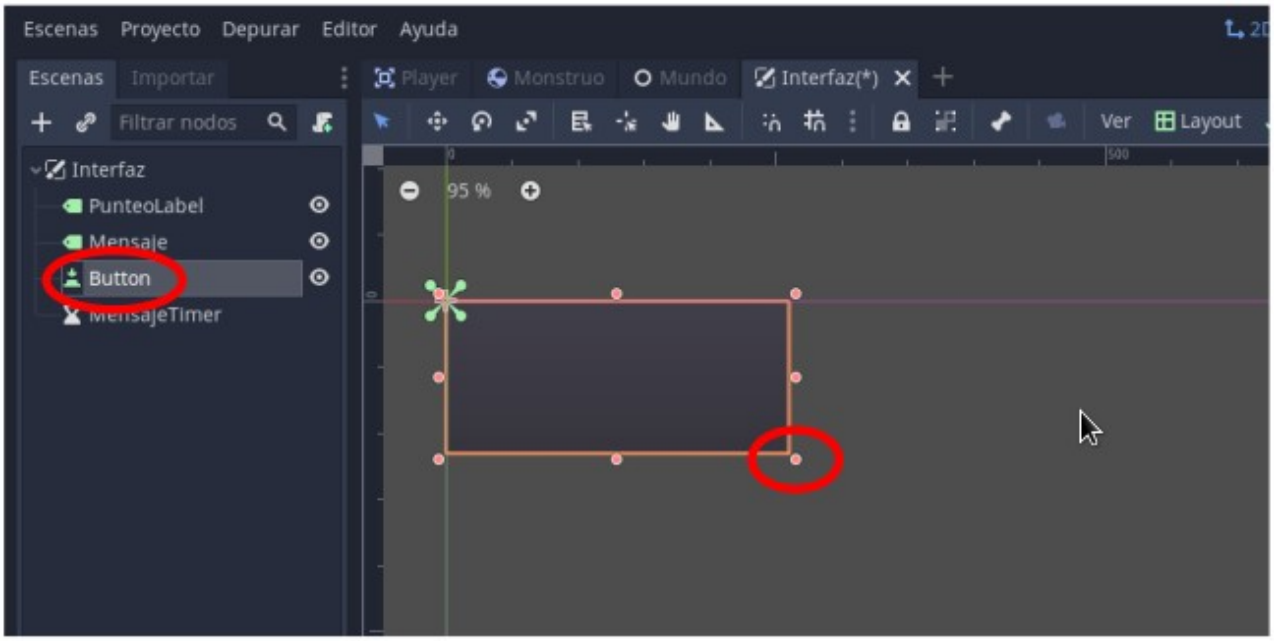
Ahora añadiremos un nodo Timer (hijo de Interfaz)



Y lo renombraremos como **MensajeTimer**

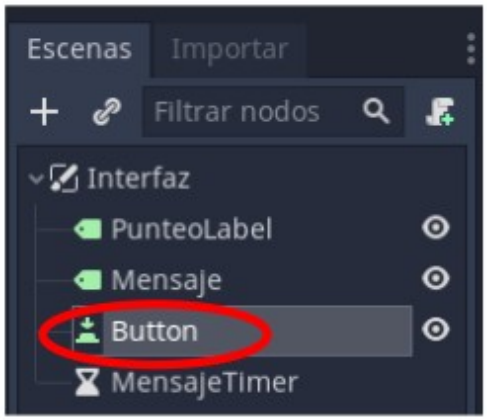
Cambiando el tamaño y posición del botón

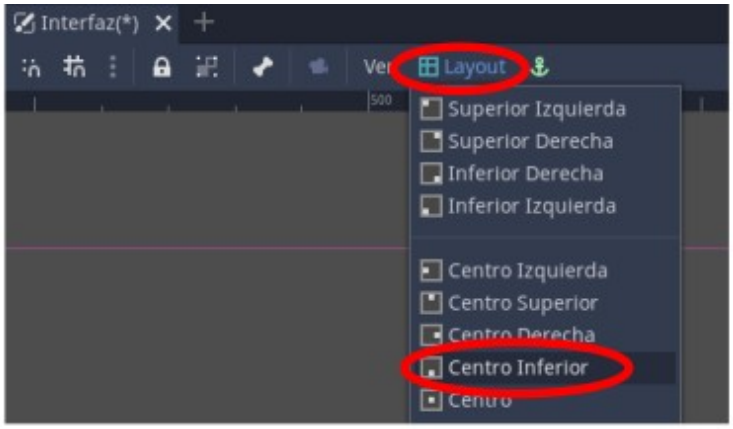
Empezaremos con el **Tamaño**. Para modificar las dimensiones del botón, basta con utilizar los puntos naranjas de las esquinas del mismo y “estirarlo” a tu gusto como hicimos con el taller de la calculadora, esto teniendo seleccionado el respectivo nodo Button



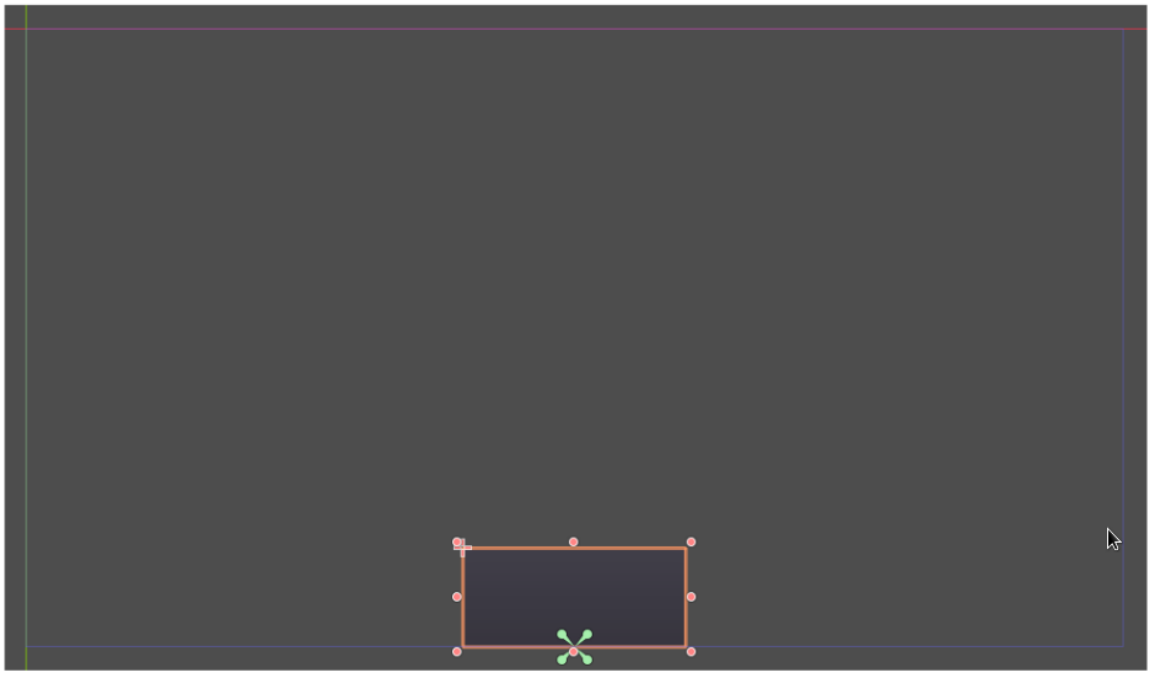
(Procura elegir un tamaño acorde a las dimensiones de la pantalla del juego)

Ahora cambiaremos la Posición. Para ello, con el nodo Button seleccionado, damos clic en la opción Layout que se encuentra en la cinta de herramientas superior y experimentamos hasta encontrar lo que nos convenza.



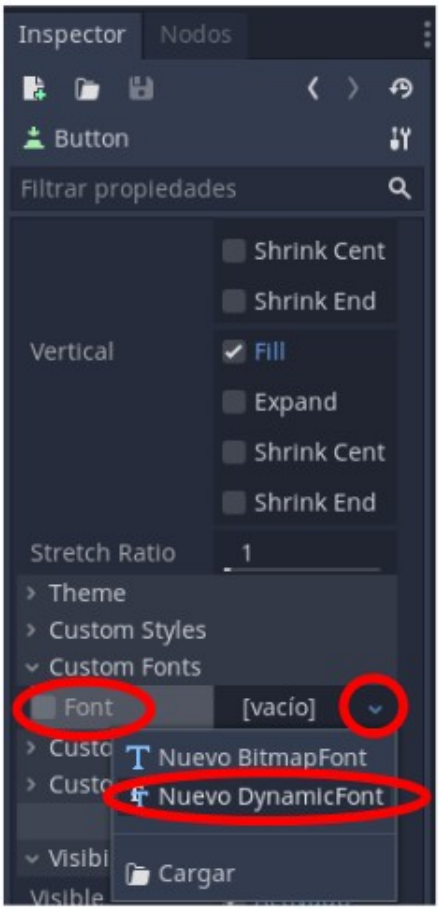


Quedará algo así:

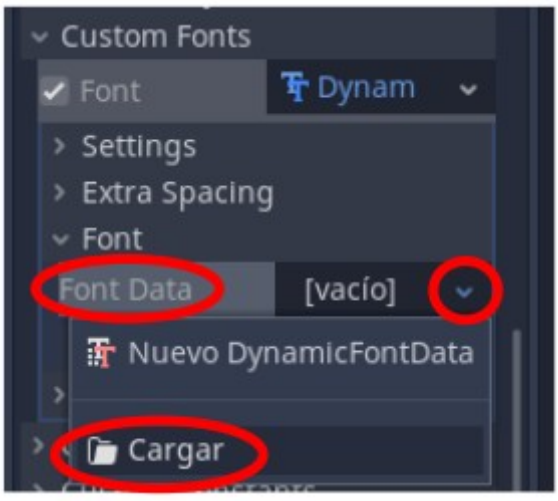


Agregando la fuente (tipo de letra) del botón

Con el nodo Button seleccionado, nos dirigiremos al inspector y agregaremos un tipo de letra (Font). Para ello, ubicaremos el atributo Font (del grupo Custom Fonts) y desplegaremos la lista **[Vacío]**, para luego elegir la opción Nuevo **DynamicFont**



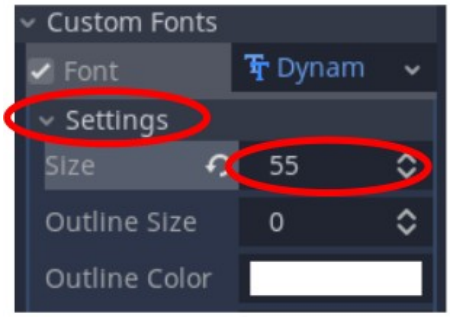
Luego, con el recién agregado valor DynamicFont seleccionado, desplegamos la opción de **FontData** y damos clic en la opción **Cargar**



Y ahí podemos buscar alguna TTF que nos guste de nuestra computadora o descargada.

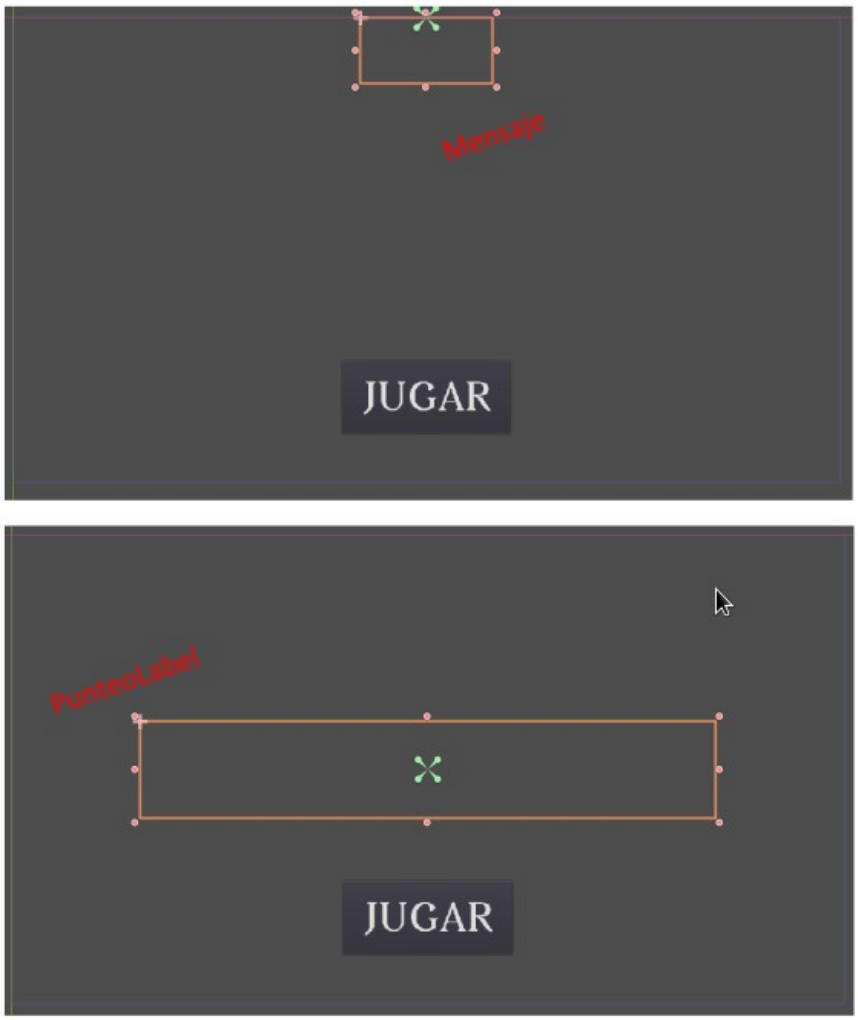
Configurando la fuente del botón

Ahora, desplegamos las opciones de configuración (**Settings**) en dónde podremos modificar a nuestro gusto atributos como el tamaño (**Size**), el borde (**Outline**), entre otros. Yo únicamente cambiaré el **Size a 55.**



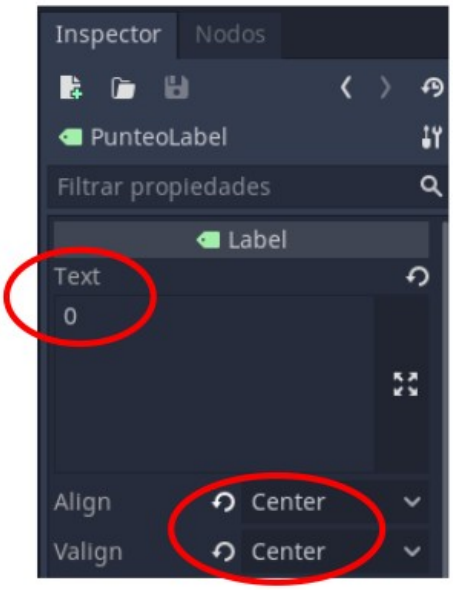
Cambiando el tamaño posición de las etiquetas de texto

De la misma forma que lo aprendimos en los pasos anteriores, cambiaremos el tamaño y posición de nuestras etiquetas de texto (Label). Empezamos haciendo un poco más grande el PunteoLabel (calculando que sea el tamaño adecuado para mostrar el punteo del juego) y un tamaño bastante más grande para el Mensaje (que es donde aparecerá el nombre de nuestro video juego). Luego ubicamos (Layout) PunteoLabel en Centro Superior y Mensaje en el Centro de toda la pantalla



Agregando el Texto a las etiquetas

Con el nodo **PunteoLabel** seleccionado, modificamos los siguientes atributos en el Inspector

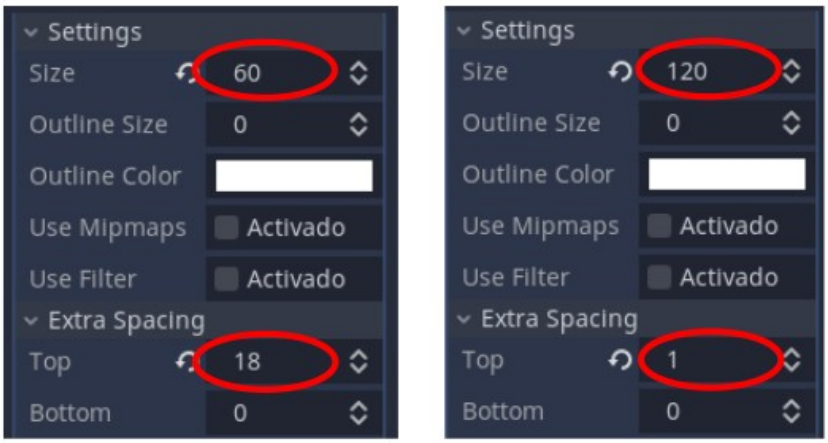


Asignando 0 en el Text (ya que el punteo de nuestro videojuego empezará con 0 hasta que inicie la partida) y

Center en la alineación horizontal (Align) y también en la vertical (Valign)

Para la etiqueta del **Mensaje**, teniendo seleccionado dicho nodo (Mensaje), repetiremos el procedimiento anterior, con la diferencia de que en el atributo Text colocaremos el nombre que queramos darle a nuestro videojuego, (el nombre que quieras) por ejemplo, yo lo llamaré **Mi primer Juego**

Para cambiar el tamaño delas etiquetas:



Agregando el Script para la Interfaz

Con el nodo Interfaz seleccionado, presionamos el botón de añadir script (pergamino blanco con signo + de color verde)

Y en la ventana dejamos los valores predeterminados, incluyendo el de Plantilla, que debería de estar Empty

Creando la señal iniciar_juego

En el Script recién creado, declaramos la signal Iniciar_juego, la cual será la señal a emitirse al presionar el botón de la interfaz para dicha acción

```
extends CanvasLayer


signal iniciar_juego
```

Creando la función mostrar_mensaje

En esta función utilizaremos un argumento el cual llamamos texto (puedes elegir el nombre que prefieras, pero recomiendo este para mayor facilidad), el cual estará igualado al atributo Text de nuestro nodo Mensaje.

También mostraremos el nodo Mensaje e iniciaremos el MensajeTimer. Todo lo anterior, se realizará escribiendo el siguiente código

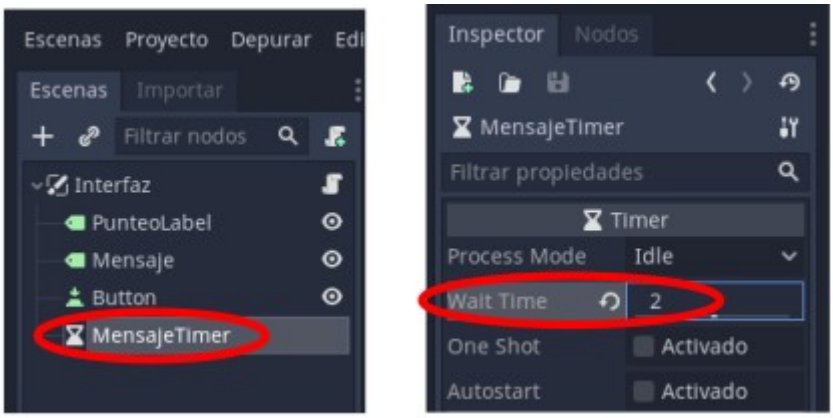
```
func mostrar_mensaje(texto):
	$Mensaje.text = texto
	$Mensaje.show()
	$MensajeTimer.start()
```

Creando función game_over

Con esta función, haremos que el juego regrese a la pantalla principal luego de mostrar un mensaje indicando que el jugador perdió. De forma que mostraremos el texto “**Game Over**” (Juego Terminado), pero, luego de una pausa de tiempo según el parámetro de la señal **timeout** de nuestro timer, volveremos a mostrar el **Button** (botón de JUAGAR), regresando el texto central al nombre del juego (en mi caso sería **Mi primer Juego**) y mostramos dicho Mensaje. Todo lo anterior, se realizará escribiendo el siguiente código

```
func game_over():
	mostrar_mensaje("Game Over")
	yield($MensajeTimer, "timeout")
	$Button.show()
	$Mensaje.text = "Mi Primer Juego"
	$Mensaje.show()
```

Y, por último, en el inspector cambiaremos el atributo Wait Time de nuestro nodo MensajeTimer (con dicho nodo seleccionado) a 2 segundos (de momento pondremos este valor para hacer pruebas, pero seguramente más adelante lo modificaremos según sea conveniente).



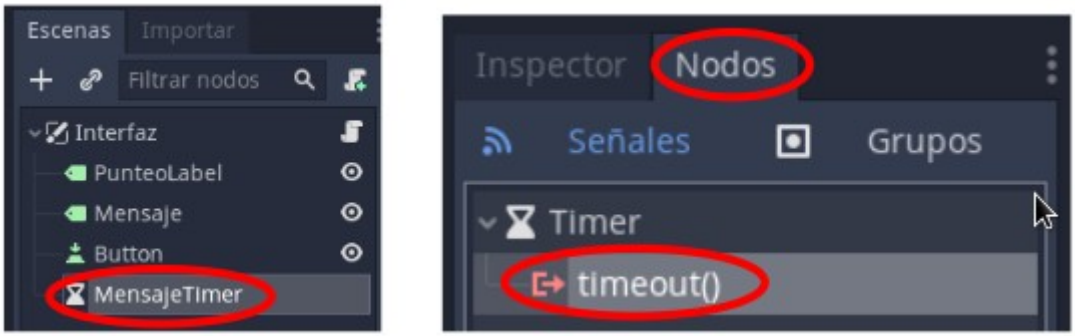
Creando la función que actualizará el punteo

A esta función la llamaremos refresh_puntos y le asignaremos el argumento Puntos, el cual convertiremos a una cadena de texto str (String) que se almacenará en la propiedad .text del nodo PunteoLabel. Esto lo haremos escribiendo el siguiente código

```
func refresh_puntos(Puntos):
	$$PunteoLabel.text = str(Puntos)
```

Conectando la señal timeout() de nuestro MensajeTimer

Con el nodo **MensajeTimer** seleccionado nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, de la forma que ya aprendimos, el nodo **timeout(),** el cual servirá para que pronto podamos definir qué queremos que pase cuando dicho tiempo termine



Y finalizamos la conexión sin cambiar ningún parámetro predeterminado.

Notarás que también se agregó un código de forma automática al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal.

```
func _on_MensajeTimer_timeout():
	pass # Replace with function body.
```

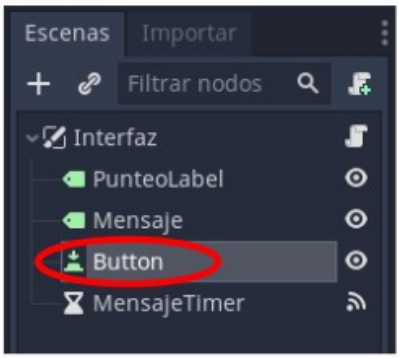
Configurando la función que se generó al conectar la señal timeout()

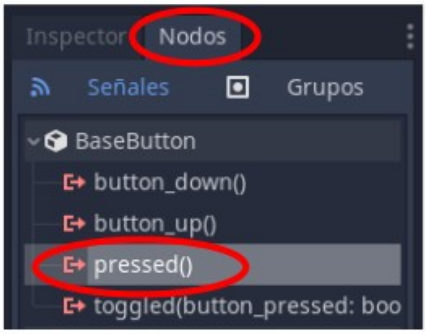
Empezamos borrando la línea **pass**. Luego ocultaremos el nodo **Mensaje**, con la siguiente línea de código

```
func _on_MensajeTimer_timeout():
	$Mensaje.hide()
```

Conectando la señal pressed() al Button

Con el nodo **Button** seleccionado nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, de la forma que ya aprendimos, el nodo **pressed()**, el cual servirá para que pronto podamos definir qué queremos que pase cuando se haga clic sobre el botón





Y finalizamos la conexión sin cambiar ningún parámetro predeterminado.

Notarás que también se agregó un código de forma automática al final de nuestro script, el cual representa a la función que acabamos de crear al realizar la conexión de la señal

```
func _on_Button_pressed():
	pass # Replace with function body.
```

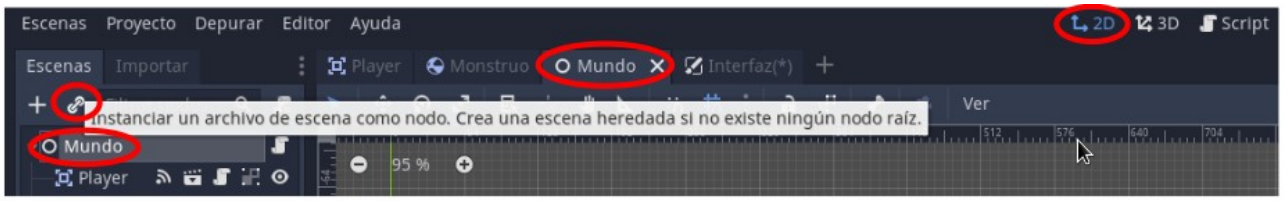
Configurando la función que se generó al conectar la señal pressed()

Empezamos borrando la línea **pass.** Luego ocultaremos el nodo **Button**, seguido de emitir la señal **iniciar_juego**. Esto con la siguiente línea de código

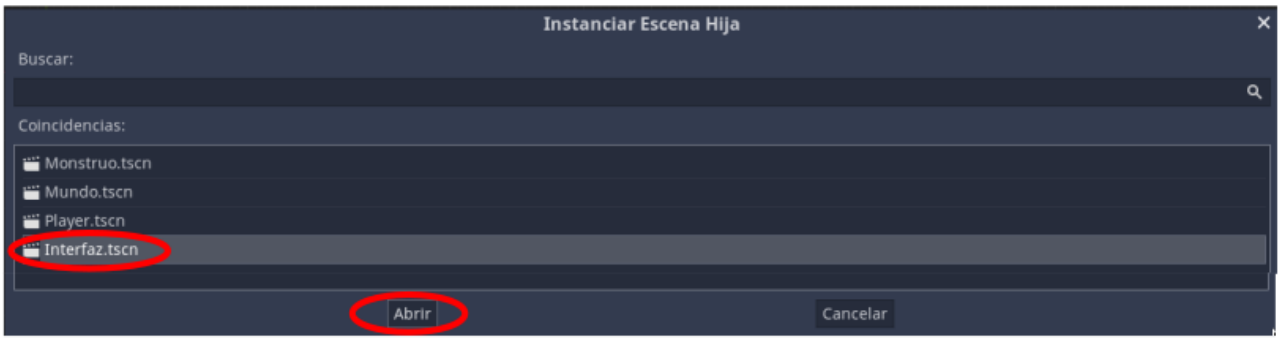
```
func _on_Button_pressed():
	$Button.hide()
	emit_signal("iniciar_juego")
```

Instanciando la escena Interfaz en el Mundo

Nos dirigimos a la vista 2D de la escena Mundo y con el nodo raíz Mundo seleccionado daremos clic en el botón de Instanciar un archivo de Escena como Nodo (el que tiene forma de eslabón de cadena)



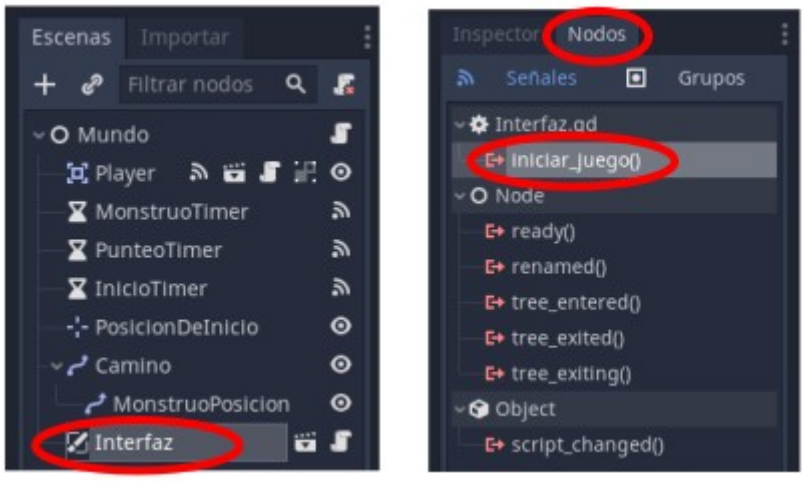
Para luego elegir y Abrir la escena Interfaz.tscn



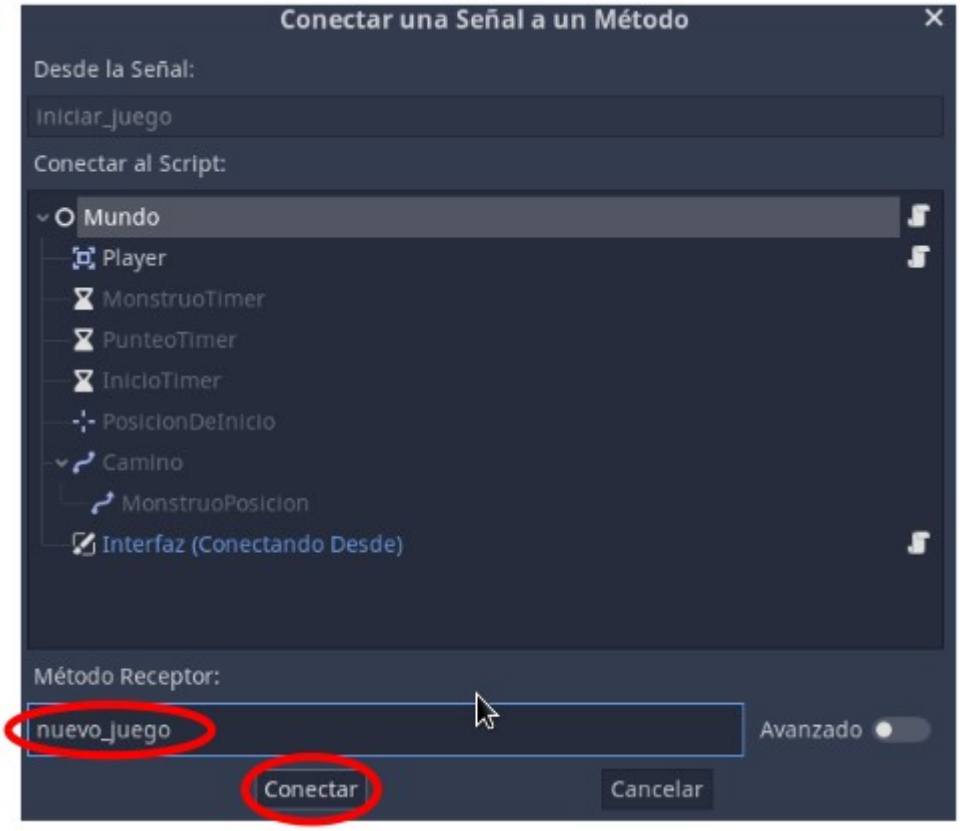
Ahora tendremos una vista general de nuestro juego

Conectando la señal nuevo_juego()

Permaneciendo en la escena Mundo y con el nodo **Interfaz** seleccionado, nos dirigimos a la pestaña Nodos (al lado de Inspector) y conectamos, de la forma que ya aprendimos, el nodo **iniciar_juego()**, el cual ya cuenta con una función previamente programada.



IMPORTANTE: En esta ocasión NO dejaremos los valores predeterminados, sino que asignaremos como nombre **nuevo_juego** (para que coincida con la función programada previamente)



Y notaremos que esta vez, en lugar de crear una nueva función, simplemente nos llevó a la que ya existía con ese nombre (dentro de nuestro Script Mundo.gd)

```
func nuevo_juego():
	Punteo = 0
	$Player.inicio($PosicionDeInicio.position)
	$InicioTimer.start()
```

(No volver a escribir este codigo)

Completando la función nuevo_juego()

Abajo del código mostrado en el paso anterior haremos un llamado a la función mostrar_mensaje de nuestra Interfaz, con el texto “Prepárate…” (para que muestre este mensaje antes de iniciar el nuevo juego), luego haremos un llamado a la función refresh_puntos teniendo la variable Punteo como argumento de entrada.

Esto lo haremos con dos líneas de código

```
func nuevo_juego():
	Punteo = 0
	$Player.inicio($PosicionDeInicio.position)
	$InicioTimer.start()
	$Interfaz.mostrar_mensaje("Preparate...")
	$Interfaz.refresh_puntos(Punteo)
```

Completando la función game_over()

Debajo de la última línea de código que está dentro de la función **game_over()**, haremos un llamado la función del mismo nombre que creamos dentro del script de nuestra Interfaz. Esto lo haremos con una línea de código

```
func game_over():
	$PunteoTimer.stop()
	$MonstruoTimer.stop()
	$Interfaz.game_over()
```

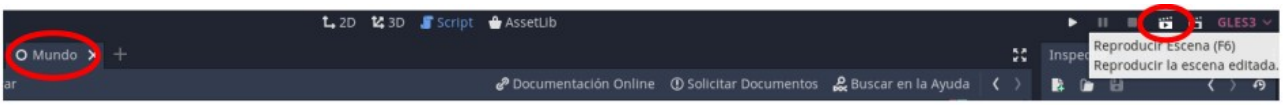
Completando la función on_PunteoTimer_timeout()

Debajo de la última línea de código que está dentro de la función on_PunteoTimer_timeout() , haremos un llamado la función refresh_puntos() de nuestra Interfaz, la cual incluirá como argumento la variable Punteo.

```
func _on_PunteoTimer_timeout():
	Punteo += 1
	$Interfaz.refresh_puntos(Punteo)
```

¡PROBANDO EL JUEGO!

Luego de estar completamente seguro de que ya guardaste todas las escenas, vamos a Reproducir la escena Mundo y…



Tu video juego (en su forma más básica) ya es completamente funcional