Es hora de integrar todo lo que conoces hasta ahora en una única escena. Debes saber que puedes copiar nodos de escena de un proyecto a otro sin problema e importarlos.

Ejercicio calificable P4. A.

> Recopila todos los ejercicios realizados hasta ahora y el conocimiento que has adquirido. Monta en un único proyecto, una escena con dos *habitaciones* llenas de objetos, luces, y con sus materiales. Recuerda que debes tener un *pasillo* que conecte los dos escenarios, y que desde una habitación no se debe visualizar directamente la otra (el *pasillo* de conexión podría ser un ascensor, y las *habitaciones* pueden ser cualquier escenario que quieras simular).

> Añade una cámara a la altura de la cabeza de tu *avatar virtual* sobre el escenario.

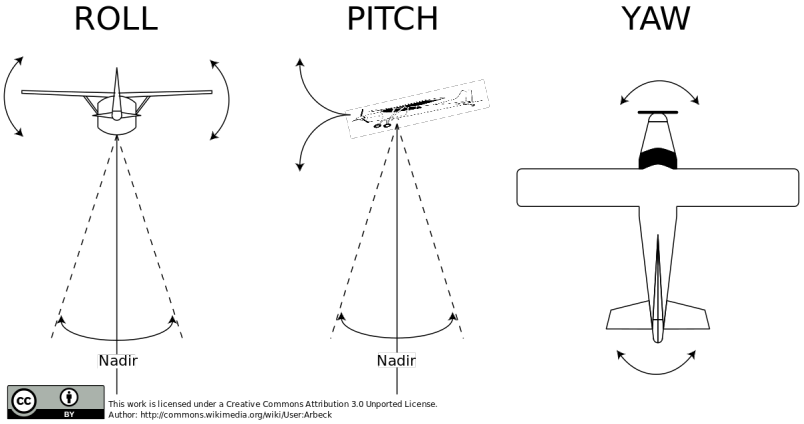
> Configura el entorno de forma adecuada (si estás en el espacio, un fondo estrellado, por ejemplo). Para lugares cerrados pon un entorno negro sin iluminación externa, funciona extraordinariamente bien un precocinado de la iluminación en estos casos.

Ya que tienes tu primera escena real es hora de movernos por el escenario. Para ello debemos programar la cámara y aprender a hacer uso del ratón para interactuar.

1. Cámara en primera persona (*First person view* o FPV)

Lo primero es entender bien dos conceptos:

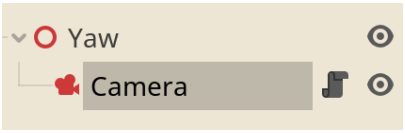
- El punto de pivote: que es el punto a partir del cual se rota un objeto.
- Los tres ángulos de rotación espacial: concretamente el **pitch**, el **roll** y el **yaw**. En la siguiente imagen puedes ver un esquema de a qué eje afecta cada ángulo.



Igual que en la primera práctica, no podemos alterar los tres ángulos al mismo tiempo, sino que tenemos que establecer una jerarquía. Concretamente, necesitaremos una base para el ángulo **yaw**, y podremos entonces usar la propia cámara para el ángulo **pitch**. Como no queremos que la cámara gire lateralmente podemos ignorar el ángulo de **roll**.

Pero antes, un par de cuestiones. La primera, es que hasta ahora no hemos tenido apenas que interactuar con el resto de elementos de la escena, pero a partir de ahora eso es precisamente lo que haremos: interconectar los elementos de la escena entre sí.

Podemos usar una variable para emplearla por simplicidad para un objeto. Por ejemplo, asumamos que tenemos un objeto Position3D como base, y su nombre es *Yaw*, tal y como muestra la figura.



Entonces, para referenciarlo en nuestro *script* asociado en la cámara, simplemente tendríamos que hacer:

```
var Yaw = get_parent()
```

Echa un vistazo a la función *get_parent* en la documentación de Godot.

Todo parece correcto, y lo es… salvo por un *detalle*…

Y es que los nodos no tienen por qué cargar en la escena en la misma hebra en la que estamos trabajando. Además, puede que incluso con una única hebra, el orden de inicialización de los objetos difiera del que pensamos. Por ello, y para asegurarnos de que se ha llamado al método _ready del objeto referenciado, hace falta incluir la orden **onready**. Es muy importante que esta parte te quede clara, por ello echa un vistazo a la *documentación de GDScript*.

Ahora mismo, es posible que no hayamos hecho nada al iniciar el método _ready del objeto Yaw, pero conviene que vayas acostumbrándote a utilizar la partícula de esta forma, y así evitar quebraderos de cabeza más tarde.

```
onready var Yaw = get_parent()
```

Ahora tendríamos que obtener las coordenadas del ratón, a ser posible las coordenadas relativas y no las absolutas. Las coordenadas absolutas del ratón dan la coordenada desde la esquina superior de la pantalla, mientras que las relativas lo hacen desde el **viewport**. Si esto no te queda muy claro es que tienes que repasar la teoría, consulta a tu profesor.

En cualquier caso, primero hay que obtener los eventos del ratón. Todos los nodos (al menos todos los visibles en pantalla) pueden recibir eventos mediante el método _input:

```
func _input(event):
```

event es una variable que puede tener múltiples tipos, y no solamente puede contener información del ratón, sino del teclado, joystick, etc.

¡Y cuidado ahora! Esta función sobrepasa a las acciones definidas desde la interfaz, en el sentido de que estamos tocando los dispositivos a bajo nivel, y eso no deberíamos hacerlo bajo ningún concepto, a menos que sepamos que siempre vamos a tener ese dispositivo conectado. Por ejemplo, en nuestro caso sabemos que forzosamente vamos a utilizar un ratón, y no hay forma sencilla de obtener sus coordenadas mediante acciones.

En la *documentación* puedes leer más acerca de estos detalles y de la interfaz a bajo nivel de Godot para el manejo de eventos.

Para leer las coordenadas del ratón desde el viewport tenemos dos alternativas, que vienen también descritas en la *documentación*.

Fíjate en que primero debemos preguntar por el tipo de datos que contiene la variable **event**:

```
if event is InputEventMouseMotion:
```

Si has programado en el lenguaje Python estarás familiarizado con la diferencia entre == y la palabra reservada **is**, si no básicamente lo que está preguntando **is** es si el objeto pertenece a una determinada clase, a diferencia de la comparación == que permite decir si dos objetos son el mismo.

Puedes depurar de forma sencilla si está leyendo bien las coordenadas mediante la función **print**.

```
print(event.relative.x)
```

A partir de este momento deberías programar una tecla para salir del entorno inmediatamente. Puedes hacer uso de la acción *ui_cancel* que por defecto está asignada a la tecla **Escape**. Para salir del entorno puedes usar esta orden:

```
get_tree().quit()
```

Puedes ocultar ahora el ratón con seguridad de esta forma:

```
Input.set_mouse_mode(Input.MOUSE_MODE_HIDDEN)
```

pero en realidad lo que queremos es capturarlo (ocultarlo y que, además, no salga del viewport):

```
Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)
```

Revisa la documentación de Input, fácilmente podrías hacer que una acción visibilizara el cursor y cambiara el modo de cámara para seleccionar objetos, aunque volveremos a esto más tarde.

Para que el objeto **Yaw** rote, tenemos que cambiar ahora su rotación en el eje local Y del objeto. Para ello, basta con usar la función set_rotation o las funciones rotation. ¡Cuidado! **¡Necesitas rotar el objeto en el espacio local!** Revisa la documentación de estas funciones.

Revisa la documentación, lo que queremos es aplicar el ángulo que obtengamos de la coordenada del ratón en horizontal (valor x del ratón) y aplicarlo al ángulo del objeto Yaw.

Si lo haces verás que es inmanejable. El problema es que el rango de x es muy alto, habrá que dividir por un valor (200 es un buen valor por defecto). Es lo que habrás visto que en muchos juegos llaman **sensibilidad del ratón**.

También es posible que tengas que cambiar el signo del ángulo si la dirección del giro es opuesta a la que esperas.

Ahora deberías tener una cámara que gira horizontalmente, pero habría que aplicar movimiento al Yaw, para ello, aplica una traslación (**¡local!**) al objeto mediante cuatro acciones (fpv_delante, fpv_detrás, fpv_izquierda, fpv_derecha).

Si lo haces bien deberías tener ya la posibilidad de moverte por el escenario, coloca algo, un cilindro una esfera o ¡cualquier otro objeto de entre tus favoritos!

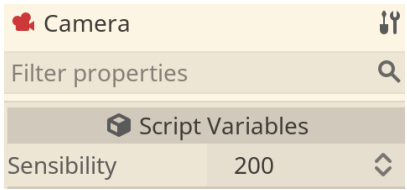
Ahora habría que añadir el eje X de rotación, y este debería ser cambiado en la cámara, o bien tener otro objeto hijo del Yaw y padre de la cámara.

Finalmente, y para dar un toque más profesional, podemos hacer que una variable aparezca en la interfaz, y así no tener que tocar código, por ejemplo hacer esto con la sensibilidad del ratón sería estupendo.

Crea una variable (global) y añade la orden **export**, puedes incluso ajustar el valor por defecto directamente:

```
export var sensibility = 200
```

Si vas a la cámara en el editor, podrás apreciar que ahora tu variable está en la interfaz.



Ejercicio calificable P4. B.

> Integra la cámara que acabas de realizar en el escenario de la práctica P4. A.

> Añade uno o más componentes de interfaz **Label** que muestren los controles y las acciones asociadas a modo de ayuda.

> Establece límites a los ángulos de cámara que consideres necesarios, de tal forma que al mirar arriba o abajo no se pueda invertir la cámara.

2. Manejo de eventos y señales

Aunque siempre podemos acceder a cualquier nodo del árbol, lo normal no es comunicarse con variables sino con señales. Imagina el caos que se puede producir si cada objeto tiene una forma distinta de comunicación.

Lo normal es que en objetos que forma un *paquete indivisible* haya comunicación mediante variables, esto es, normalmente comunicación entre padres e hijos del árbol, que conforman el mismo objeto (como en el caso de la cámara).

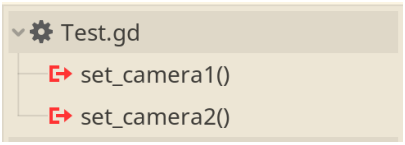
Sin embargo para el resto de comunicaciones lo normal es hacerlo mediante señales. Para entender como funciona, imagina que hay un objeto emisor de señales, por ejemplo, crea un nodo de control, vamos a llamarle **Test**.

Vamos a crear dos cámaras, la **Camara1**, y la **Camara2**.

Queremos añadir dos señales en el nodo de control **Test**, una para activar la cámara 1, y otra para activar la cámara 2. Lo primero es crear dos tipos de señal nuevos (los vamos a llamar camara_1 y camara_2).

```
signal camara_1
signal camara_2
```

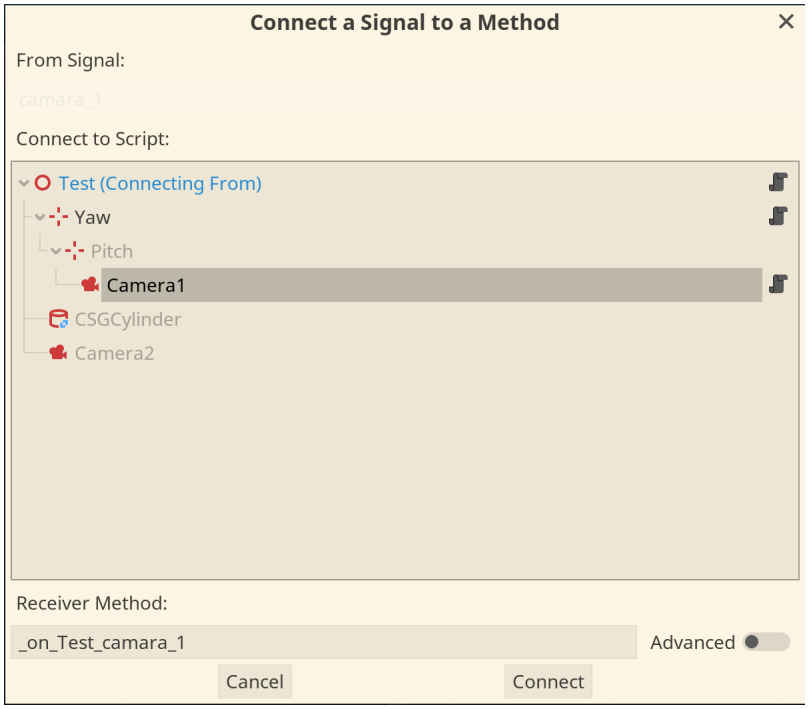
Ahora si vas al nodo, en la parte derecha, al lado de **Inspector** está la pestaña **Node**, y dentro verás todas las señales (**Signals**) que puede enviar nuestro nodo.



Hay dos señales personalizadas llamadas set_camera1() y set_camera2(), por código podemos añadir dos momentos para llamar a esta señal, por ejemplo al pulsar un botón del teclado cambiaremos entre una y otra:

```
emit_signal(«camara_1»)
```

Ahora necesitamos un receptor, que puede ser cada una de las cámaras. Haz doble click en el editor, en la señal camara_1. Aparecerá un mensaje para seleccionar el objeto con el cual conectarlo, elige la primera cámara. Verás que debajo saldrá el nombre del método (callback) al que se llamará cuando la señal se emita (_on_Test_camara_1).

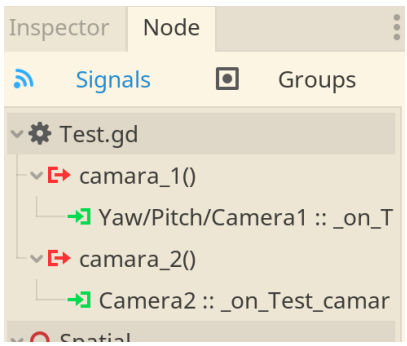


Al pulsar **Connect** se creará el método en la cámara, en este callback simplemente activaremos la cámara:

```
set_current(true)
```

Haremos lo mismo con la señal camara_2 y la segunda cámara. Habrás comprobado que **no se puede conectar una señal a un nodo que no tiene código asociado** (primero hay que añadir un *script* al nodo).

Observa como el nodo de control tiene las señales conectadas con ambas cámaras. Puedes probar ahora el resultado, y verás como cambias de una cámara a otra.



Aquí tienes un *tutorial de señales* que puede servirte para entender mejor todo el mecanismo, de todas formas se explicará también en clase.

Ejercicio calificable P4. C.

> Añade una cámara al ejercicio calificable anterior, y realiza el proceso descrito para cambiar de una cámara a otra. Puedes aprovechar para poner una animación a dicha cámara (como si fuera una cámara de vigilancia).

> Verás que hay un problema, cuando estés visualizando la *cámara de vigilancia* ¡aún podrás mover la cámara de primera persona! Arréglalo para que esto no suceda y solamente puedas manejar la cámara FPV cuando ésta esté activa.

La interfaz de usuario funciona de la misma forma, emitiendo señales y recibiéndolas. Puedes ver un tutorial de interfaces *aquí*, ya que una vez que aprendes lo básico de señales es muy simple de manejar.

Ejercicio de entrenamiento 1:

> Prueba a crear una interfaz de usuario en lugar de usar una tecla para cambiar de una cámara a otra. Por ejemplo, mediante un par de botones. Para ello, haz que el RBM desactive el movimiento de cámara y active la visualización del cursor. Haz que también que aparezca la interfaz (echa un vistazo a los métodos hide y show). Puedes usar un botón para volver a ocultar la interfaz y volver a capturarlo (mediante Input.set_mouse_mode).