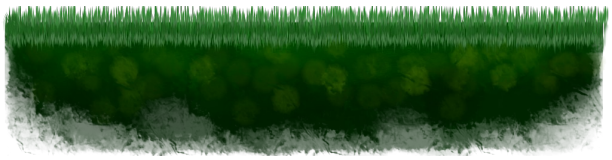
Godot Engine Tutorial
Creando un personaje de plataformas
Parte II

En la primera parte creamos el esqueleto para nuestro proyecto dentro del motor Godot. En esta segunda parte vamos a mover el personaje a los lados y saltar correctamente.



Move()

Lo primero que vamos a hacer es hacer darle a nuestro personaje la posibilidad de moverse de izquierda a derecha presionando las teclas. Ya vimos cómo capturar la entrada de eventos utilizando Input Actions y la función *_input(event)*. Lo siguiente será aplicar un movimiento al personaje.

Para hacer esto ya vimos que con el método apply_impulse(posicion, impulso) no basta. Necesitamos quitarle responsabilidad al motor de fisicas y obtener mayor control sobre el personaje. En lugar "impulsar" al personaje y dejar que la simulación lo mueva sobre el nivel, vamos a mover el personaje "manualmente".

Agregamos la sentencia *set_fixed_process(true)* a la funcion de inicialización *_ready()* y creamos la función *_fixed_process(delta)*.
(Podemos eliminar la función *_input(event)* ya que no la necesitaremos)

Crearemos una nueva función llamada move() dentro del script del personaje.

```
func move(desired_velocity_x, accel, delta):
    velocity_x = lerp(velocity_x, desired_velocity_x, accel*delta)
    set_linear_velocity(Vector2(velocity_x, get_linear_velocity().y))
```

Esta función recibe la velocidad deseada (*desired_velocity*), la aceleración para llegar a esa velocidad (*accel*) y los segundos pasados desde el último frame (*delta*). En base a estos datos la función realiza un cálculo y mueve al personaje seteando su velocidad lineal.

Notar que la función utiliza una variable extra llamada *velocity_x* para guardar el valor de la velocidad en el eje x. Creamos la variable en nuestro script.

Nuestro script se ve así:

```
extends RigidBody2D

const VEL_MAX = 300
const VEL_NONE = 0

var velocity_x = 0
var acceleration = 7


func _ready():
	set_fixed_process(true)

func _fixed_process(delta):

	if Input.is_action_pressed("btn_left"):
		move(-VEL_MAX, acceleration, delta)
	elif Input.is_action_pressed("btn_right"):
		move(VEL_MAX, acceleration, delta)
	else:
		move(VEL_NONE, acceleration, delta)


func move(desired_velocity_x, accel, delta):
	velocity_x = lerp(velocity_x, desired_velocity_x, accel*delta)
	set_linear_velocity(Vector2(velocity_x, get_linear_velocity().y))
```

Si corremos el juego vemos que ahora el personaje se mueve correctamente. Cuando apretamos la tecla izquierda la acción **btn_left** está presionada, por lo que ejecutamos la función ***move()***, lo mismo para **btn_right.**
Notar que cuando no están presionadas ninguna de las dos teclas igualmente llamamos a la función **move()** pero la velocidad deseada ahora es 0.

Jump()

Lo que sigue es hacer saltar a nuestro personaje. Queremos que el motor de física se encargue del movimiento en el salto, así que simplemente aplicamos una fuerza ascendente en el eje Y y dejamos que la simulación se encargue del resto.

Creamos una nueva constante llamada *IMP_JUMP* con valor 700. Además agregamos el código para hacer el salto en la función **_fixed_process(delta)**.

El código resultante:

```
func _fixed_process(delta):

	if Input.is_action_pressed("btn_left"):
```

```
        move(-VEL_MAX, acceleration, delta)
    elif Input.is_action_pressed("btn_right"):
        move(VEL_MAX, acceleration, delta)
    else:
        move(VEL_NONE, acceleration, delta)

    if Input.is_action_pressed("btn_jump"):
        apply_impulse(Vector2(), Vector2(0, -IMP_JUMP))
```

Ahora vemos que podemos saltar... pero muy alto.

¿Cual es el problema?

En realidad son dos:

1. Estamos aplicando la fuerza para el salto en cada frame! Esto significa que en lugar de aplicar la fuerza una sola vez cuando presionamos la tecla barra espaciadora por primera vez, estamos aplicando la fuerza en cada frame en que la barra esté presionada. Esto está mal!
2. El personaje puede saltar aún cuando no está parado en el suelo. Si el personaje está en el aire no debería poder saltar nuevamente. Esto está mal!

Soluciones

La solución al primer problema involucra mantener nosotros mismos información extra sobre el estado de las teclas presionadas. Necesitamos saber por ejemplo, si una tecla estaba presionada en el frame anterior pero ya no, o si una tecla no estaba presionada anteriormente pero ahora si lo está, etc.

Por el momento no existe esta funcionalidad en el motor (aunque creo que está pensada para ser implementada próximamente), así que vamos a utilizar un script que solucione este problema.

Este script está basado en uno que encontré en internet con algunas modificaciones extra para hacer que sea mas fácil su uso. El script se llama *input_states.gd* y pueden descargarlo aquí.

Cómo se usa:

1. Para utilizarlo cargamos el script y luego instanciamos un objeto de él, pasando como parámetro el nombre del Input Action que queremos manejar.
   ```
   var input_states = preload("res://scripts/input_states.gd")
   var btnJump = input_states.new("btn_jump")
   ```
2. En cada frame le pedimos que actualice su información (solo una vez por frame!)
   ```
   btnJump.check()
   ```
3. Cuando queremos leer el estado de un Input Action simplemente utilizamos la función *last()*:
   ```
   If btnJump.last() == btnJump.JUST_PRESSED:
       #el botón jump es presionado por primera vez
   ```

```
    if btnJump.last() == btnJump.PRESSED:
        #el botón jump está presionado
    if btnJump.last() == btnJump.JUST_RELEASED:
        #el botón jump recién fue soltado
    if btnJump.last() == btnJump.RELEASED:
        #el botón jump no está presionado
```

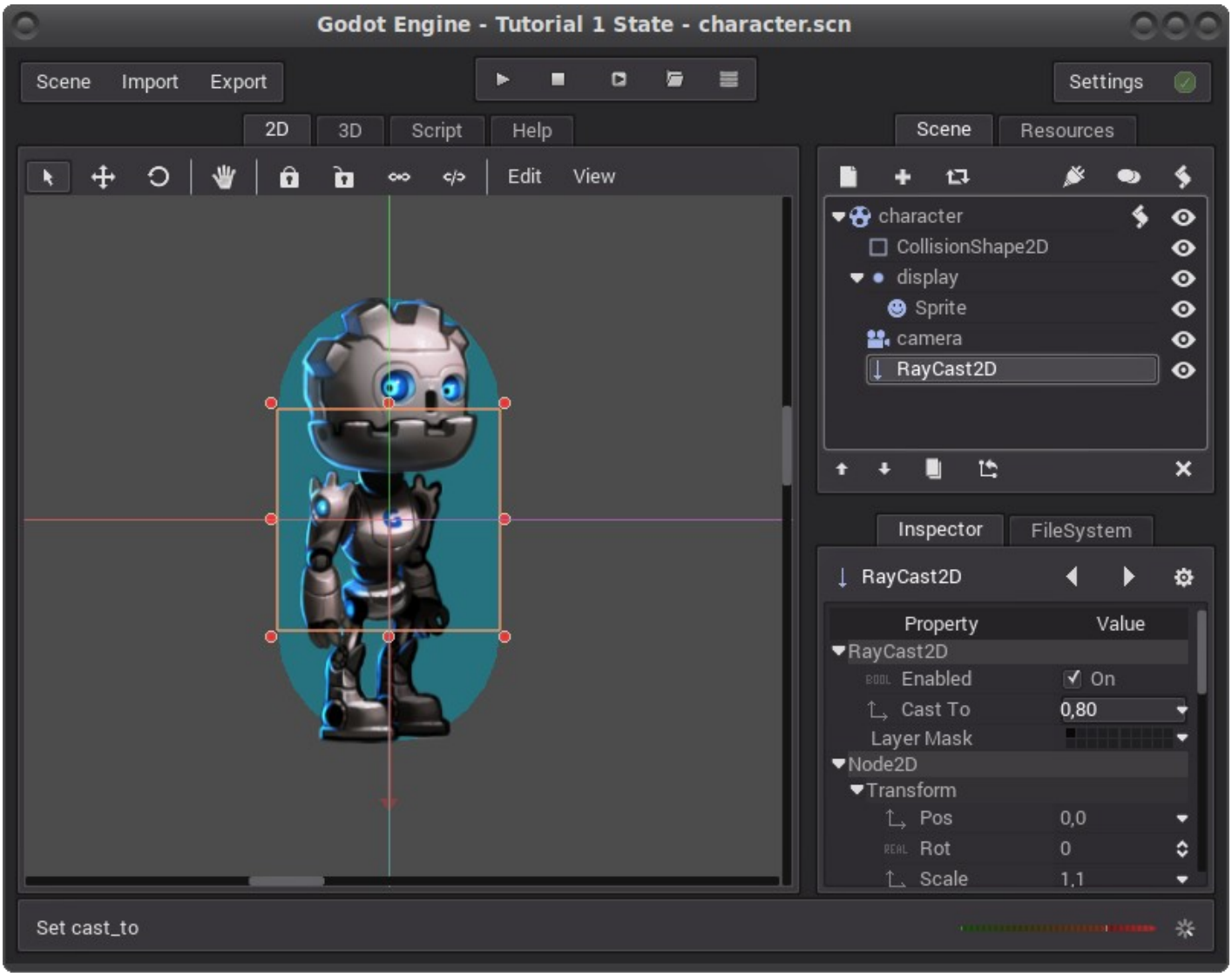
Para solucionar el primer problema utilizamos *input_states.gd* de la siguiente forma en nuestro script:

```
#Creamos las variables para los botones
var input_states = preload("res://scripts/input_states.gd")
var btnLeft = input_states.new("btn_left")
var btnRight = input_states.new("btn_right")
var btnJump = input_states.new("btn_jump")

#Chequeamos el estado de los botones una vez por frame.
#Cambiamos los ifs anteriores por los nuevos
func _fixed_process(delta):

    btnLeft.check()
    btnRight.check()
    btnJump.check()

    if btnLeft.last() == btnLeft.PRESSED:
        move(-VEL_MAX, acceleration, delta)
    elif btnRight.last() == btnRight.PRESSED:
        move(VEL_MAX, acceleration, delta)
    else:
        move(VEL_NONE, acceleration, delta)

    if btnJump.last() == btnJump.JUST_PRESSED:
        apply_impulse(Vector2(), Vector2(0, -IMP_JUMP))
```

Ahora si corremos el juego vemos que el personaje se mueve correctamente. Bueno, casi. Aún existe el segundo problema y el personaje puede saltar incluso cuando está en el aire.

La solución al segundo problema es mucho más simple. Necesitamos saber cuándo el personaje está parado sobre el suelo o sobre una plataforma. Por suerte Godot tiene una herramienta para esto: el nodo Raycast2D. Este nodo crea un "rayo" que colisiona contra otros objetos en el mundo físico y nos devuelve la información de esa colisión.

Para utilizarlo volvemos a editar la escena de nuestro personaje *character.scn*. Agregamos un nodo RayCast2D como hijo del nodo raíz, ponemos su atributo **enabled** en verdadero, y seteamos el vector de cast con los valores (0, 80). Gráficamente se ve en el editor como una flecha que sale por debajo de los pies del personaje.



Lo siguiente es utilizar la información del raycast para saber cuándo el personaje está parado sobre el suelo. Para esto utilizamos la función del nodo Raycast:

- **is_colliding()**: nos devuelve verdadero/falso según colisiona o no

Lo primero que necesitamos hacer es crear una variable más en nuestro script para almacenar la referencia a este nodo y accesarla al momento de realizar el salto:

```
var raycast = null

func _ready():
	set_fixed_process(true)
	raycast = get_node("raycastFloor")
	raycast.add_exception(self)
```

Notar que agregamos una excepción al ***raycast***: no queremos que el ***raycast*** colisione con nuestro personaje (***self***).

Luego cambiamos la condición del salto por una utilizando la información del raycast.

```
if raycast.is_colliding() and btnJump.last() == btnJump.JUST_PRESSED:
		apply_impulse(Vector2(), Vector2(0, -IMP_JUMP))
```

Con esto logramos solucionar también el segundo problema.

Fin de la Segunda Parte

Descargar Parte II: zip con el proyecto terminado

Contacto:

Federico Pacheco
@fede0d
http://fede0d.github.io/