Índex

1 Introducció a GDScript

GDScript és un llenguatge de programació d'alt nivell, orientat a objectes, imperatiu i gradualment tipat creat per **Godot**. Utilitza una **sintaxi basada en sagnat** similar a llenguatges com **Python**. El seu objectiu és optimitzar i integrar-se estretament amb **Godot Engine**, permetent una gran flexibilitat per a la creació i integració de contingut.

Documentació oficial

1.1 El primer script

Agafem l'script de Creating your first script de la documentació de Godot.

```
extends Sprite2D

var speed = 400
var angular_speed = PI


func _process(delta):
    rotation += angular_speed * delta

    var velocity = Vector2.UP.rotated(rotation) * speed

    position += velocity * delta
```

Aquest script està associat a un node **Sprite2D**, per això la primera línia, `extends Sprite2D`, el que fa és heretar les **funcions** (mètodes) i **propietats** (variables) de la classe **Sprite2D**. Per exemple, les variables `rotation` i `position` són propietats de la classe, accessibles des del Inpector (panell dret).

A continuació tenim 2 **variables** definides amb la paraula clau `var`, `speed` i `angular_speed`. De les variables parlarem més endavant.

Per últim tenim la funció `_process`. Una **funció** és un bloc de codi que té un nom i a la qual es pot cridar, el que significa que es pot executar el seu codi cada vegada que escrivim el seu nom. El codi de les funcions den GDScript es troba niuat per sagnats. Al nostre exemple, el codi són les línies:

```
    rotation += angular_speed * delta

    var velocity = Vector2.UP.rotated(rotation) * speed
```

```
    position += velocity * delta
```

`_process` és una funció ja definida en el sistema, el que significa que podem canviar el seu contingut, però que té un significat especial. Aquesta en concret s'executa cada *frame*, unes 60 voltes per segon, el que permet fer moviments de personatges del joc, efectes, lectura de tecles, etc.

La funció `_process` té un paràmetre, `delta`, que és el temps que ha passat des de l'últim frame. Aquest paràmetre és una variable que normalment s'utilitza per a fer que els canvis en els elements succeixen a la mateixa velocitat independentment del `frame rate` (tasa de frames) o `fps` (*frames per secon*). Al nostre codi el multipliquem per `rotation` i `position` per tal que la velocitat de rotació siga homogènia.

1.2 Comentaris

Qualsevol cosa des d'un # fins al final de la línia s'ignora i es considera un comentari.

```
# Això és un comentari d'una línia
```

Els comentaris de vàries línies es poden crear utilitzant `"""` (tres cometes seguides) al principi i al final d'un bloc de text. Tingueu en compte que això crea una cadena, per tant, no s'eliminarà quan es compile l'script.

```
""" Aquest és
 un comentari
 de vàries línies
"""
```

1.3 Variables

Les **variables** són nom simbòlics que contenen un valor. Per exemple:

```
var life = 100
```

siginifica que estem creant (usant la paraula clau `var`) una variable anomenada *life* amb el valor `100`.

Les variables es poden llegir:

```
var life = 100
print(life)
```

imprimirà el valor `100`, que està emmagatzemat en la variable `life`.

També podem modificar el contingut d'una variable una vegada creada amb l'operador d'assignació (=). Sempre es calcula el costat dret i s'assigna a la variable del costat esquerre:

```
var life = 100 # life té el valor 100
life = 50 # life té el valor 50

var damage = 10
life = life - damage # Ara life té el valor 40 (50 - 10)
```

Les variables tenen un tipus (*type*), que és el tipus de dades que emmagatzemen. Els tipus bàsics de GDScript són:

- **`null`**: `null` és un tipus de dades buit que no conté informació i no se li pot assignar cap altre valor.
- **`bool`**: Abreviatura de "booleà", només pot valdre `true` o `false`.
- **`int`**: Abreviatura de "integer", emmagatzema nombres enters (positius i negatius).
- **`float`**: Emmagatzema nombres reals, inclosos els decimals, que s'indiquen amb un punt (`.`).
- **`String`**: Cadena de caracters, entre cometes dobles (`"`).

L'assignació de tipus es fa automàticament, depenent del valor assignat. A més, si canvia el valor de la variable i és d'un tipus diferent, també canvia el tipus de la variable.

Exemples:

```
var a = true; # bool
var b = 5; # int
var c = 4.5 # float
var message = "Welcome" # String
message = 5 # Ara message és int
```

1.4 Operadors

El operadors matemàtics més freqüents són:

```
x + y # suma
x - y # sustracció
x * y # multiplicació
x / y #divisió
x % y # Mòdul, el resultat és el residu de la divisió entera de x entre y.
 ↪  Exemple: 8 % 3 = 2
-x # negació
x ** y # exponenciació, x elevat a y
```

L'operador d'assignació = assigna a la variable de l'esquerra el valor resultant de l'operació de la dreta.

```
x = y # x val y
x += y # x = x + y
x -= y # x = x - y
x *= y # x = x * y
x /= y # x = x / y
```

1.5 Constants

Les constants són variables que no poden canviar quan s'executa l'script. L'ús de la paraula clau `const` permet donar un nom a un valor constant. Si intenteu assignar un valor a una constant després de declarar-la, us donarà un error.

Es recomana utilitzar constants sempre que un valor no haja de canviar. Exemple:

```
const height = 200

func _ready():
    height = 300
```

Aquest codi produeix l'error *"Cannot assign a new value to a constant"*.

1.6 Funcions

Una **funció** és un fragment de codi que s'associa a un nom simbòlic i que pot ser executat en qualsevol moment, escrivint el nom de la funció.

El cos de la funció es troba definit per una indentació cap a la dreta.

Exemple:

```
func welcomeMessage():
    var message = "Welcome"
    print(message)

welcomeMessage() # Imprimeix Welcome
```

En aquest exemple el nom de la funció és `welcomeMessage`. Va seguit d'uns parèntesis entre els quals escrivim els paràmetres (en cas cas no n'hi ha). El cos de la funció són les 2 línies següents. Per últim, la instrucció welcomeMessage() executa la funció imprimint el misatge.

> Les funcions es diferencien de les variables fàcilment: les funcions sempre tenen parèntesis al final del nom, mentre que les variables no.

Les funcions poden tindre 0, 1 o més **paràmetres**, als quals podem passar valor que s'utilitzaran dins de la funció:

```
func welcomeMessage(name):
    var message = "Welcome" + name
    print message

welcomeMessage("Mary") # Imprimeix Welcome Mary
```

> En aquest exemple l'operador + concatena (uneix) les 2 cadenes de text.

Una altre exemple:

```
func addition(a, b):
    var result = a + b
    return result

addition(5, 6) # Imprimeix 11
```

La paraula clau `result` en una funció retorna el valor especificat i termina l'execució de la funció. Si una funció no té cap instrucció `result`, termina al finalitzar i retorna `null`.

Quan necessitem definir una funció, però aquesta no té cos, hem d'escriure la paraula clau `pass`:

```
func empty_function():
    pass
```

1.6.1 Funcions del sistema

Godot té algunes funcions definides, les qual podem utilitzar als nostres scripts. El seu nom sol començar amb _. Les més importants són:

- **`_ready()`**: Es crida quan el node està "a punt", és a dir, quan tant el node com els seus fills han entrat a l'arbre de l'escena. S'executa una vegada al principi. Normalment s'utilitza per a initzialitzar variables. Pertany a la classe Node.
- **`_init()`**: Es crida una vegada quan l'script çomença a executar-se. S'executa abans que `_ready()`. Pertany a la classe `Object`.
- **`_process(delta)`**: Es crida cada fotograma o *frame*. L'argument `delta` és el temps transcorregut des de l'últim frame (en segons). S'utilitza per a implementar el bucle del joc en cada escena.
- **`_physics_process(delta)`**: Es crida cada *frame* de processament de físiques. Similar a `_process`, però s'utilitza en objectes que tenen un comportament de físiques (gravetat, forces, etc).

1.7 Control de flux

1.7.1 Operadors condicionals

Són operadors el resultat dels quals és `true` o `false`:

```
x == y # true si x i y són iguals
x != y # true si x i y són diferents
x < y # true si x és menor que y
x > y # true si x és major que y
x <= y # true si x és menor o igual que y
x >= y # true si x és major o igual que y
not x # negació del valor de x, true o false
!x # el mateix que not x
```

Podem unir vàries condicions amb els operadors and (&&) i `or` (||).

```
x and y # true si x i y són true
x && y # el mateix que x and y
x or y # true si qualsevol dels 2 valors és true
x || y # # el mateix que x or y
```

Exemples:

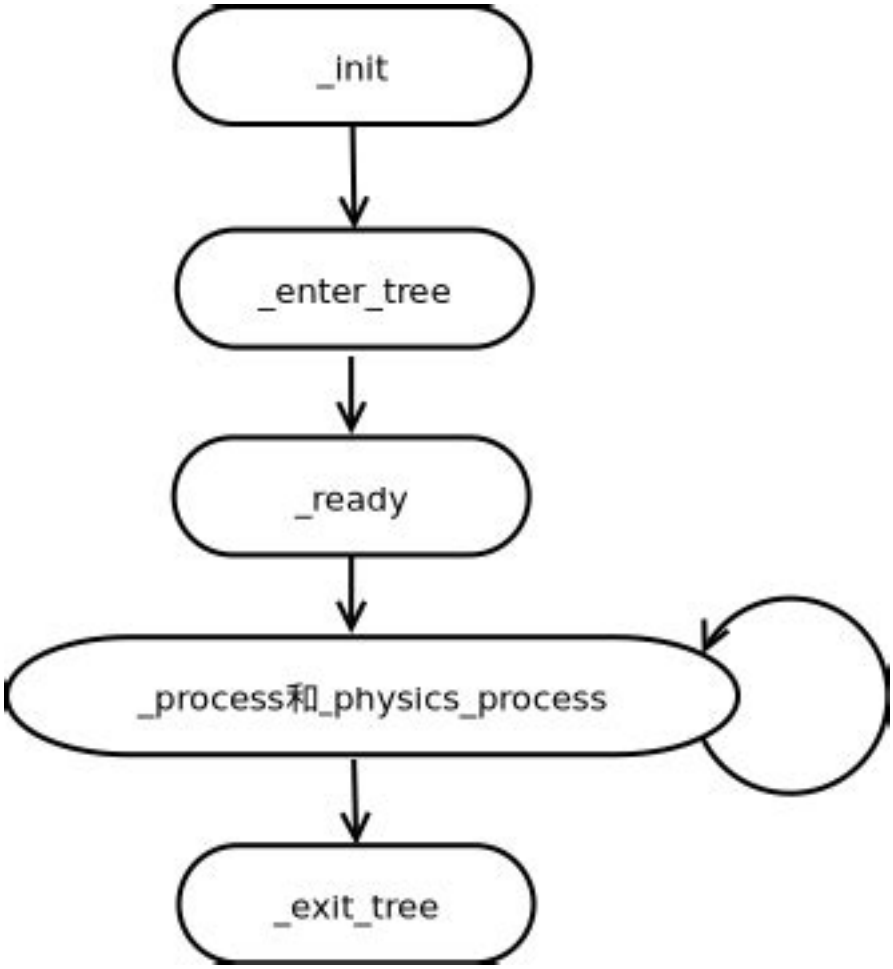

Figura 1: Cicle de vida de Godot

```
var a = 5
var b = 2
var c = 3

a == b # false
a != c # true
a == b + c # true
a < b # false
c > b # true
a <= b +c # true
!(a == b) # true
a > b and b < c # true
a > b && b >= c # false
a > b or b < c # true
a > b || b >= c # true
a < b or b >= c # false
not (a < b or b >= c) # true
```

1.7.2 if/else

Les condicions simples es creen utilitzant la sintaxi `if/else`.

La sintàxi més bàsica és:

```
if condició:
    bloc de codi
```

En la condició s'utilitzen expressions com les vistes a l'apartat anterior, el resultat de les quals ha de ser `true` o `false`. Si el resultat és `true`, s'executa el bloc de codi del `if`. Si el resultat és `false`, no s'executa res i continua el programa.

Exemple:

```
var a = 5
var b = 6

if a < b:
   print("a és menor que b")
```

Podem usar la paraula clau `else` per a especificar el codi que s'executaria en cas que la condició siga `false`:

```
if condició:
    bloc de codi A
else:
    bloc de codi B
```

Exemple:

```
var a = 5
var b = 6

if a < b:
   print("a és menor que b")
else:
   print("a és major o igual que b")
```

Si tenim més de 2 condicions, podem valorar les que no compleixen la primera afegint sentències `elif`:

```
var a = 5
var b = 6

if a < b:
   print("a és menor que b")
elif a > b:
    print("a és menor que b")
else:
   print("a i b són iguals")
```

1.7.3 Bucles `while`

Els bucles serveixen per a executar una mateixa acció mentres es complisca una condició.

El bucle més simple és el `while`:

```
while (condicio):
    instruccions
```

Per exemple:

```
var n = 0; #Iniciem la variable comptador n a 0

#Aquest bucle imprimeix el números de 0 a 9
while (n < 10):
    print(n)
    n += 1
```

1.7.4 Bucles `for`

Els bucles for s'utilitzen principalment per a recorrer un array:

```
var names = ["John", "Marta", "Samantha", "Jimmy"]

for name in names:
    print(name) # Imprimeix tots els noms de l'array
```

En l'exemple anterior tenim un array anomenat *names* el qual recorreguem amb `for name in names`; la variable *name* pren el valor de cada element de l'array *names*.

1.8 Arrays

Els **`arrays`** (també coneguts com *arranjaments* o *matrius*) són seqüències de variables o objectes.

Podem crea un array buit o amb elements:

```
var numbers = [] # Array sense elements
var fruits = ['apple', 'orange', 'banana'] # Array amb 3 elements
```

Podem llegir el valor d'un element amb el seu index. El primer element té l'índex 0:

```
var fruits = ['apple', 'orange', 'banana']
print(fruits[0]) #Imprimeix apple
print(fruits[2]) #Imprimeix banana
```

Amb el mètode `.size()` obtenim el nombre d'elements d'un array:

```
var fruits = ['apple', 'orange', 'banana']
print(fruits.size()) #Imprimeix 3
print(fruits[fruits.size() - 1]) #Imprimeix l'últim element
```

Podem utilitzar el valor de `size` per a recòrrer l'array:

```
var fruits = ['apple', 'orange', 'banana']
for i in range(fruits.size()):
    print(fruits.[i]) # Imprimeix totes les fruites
```

Els arrays podem créixer automàticament cada vegada que afegim un element. Això ho podem fer amb la funció append:

```
var fruits = ['apple', 'orange', 'banana']
fruits.append('tomato')
print(fruits.size()) #Imprimeix 4
print(fruits[fruits.size() - 1]) #Imprimeix tomato
```