

Godot: Part I

the absolute basics

CSIE Challenge 2025

2025-04-28

[Godot 官方文件](#) 


Outline

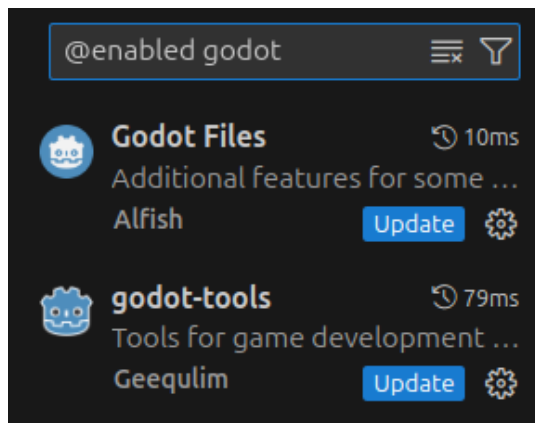
1. Environment Setup
2. The Godot Engine
3. GDScript
4. Live Demo / Code Review?

Environment Setup

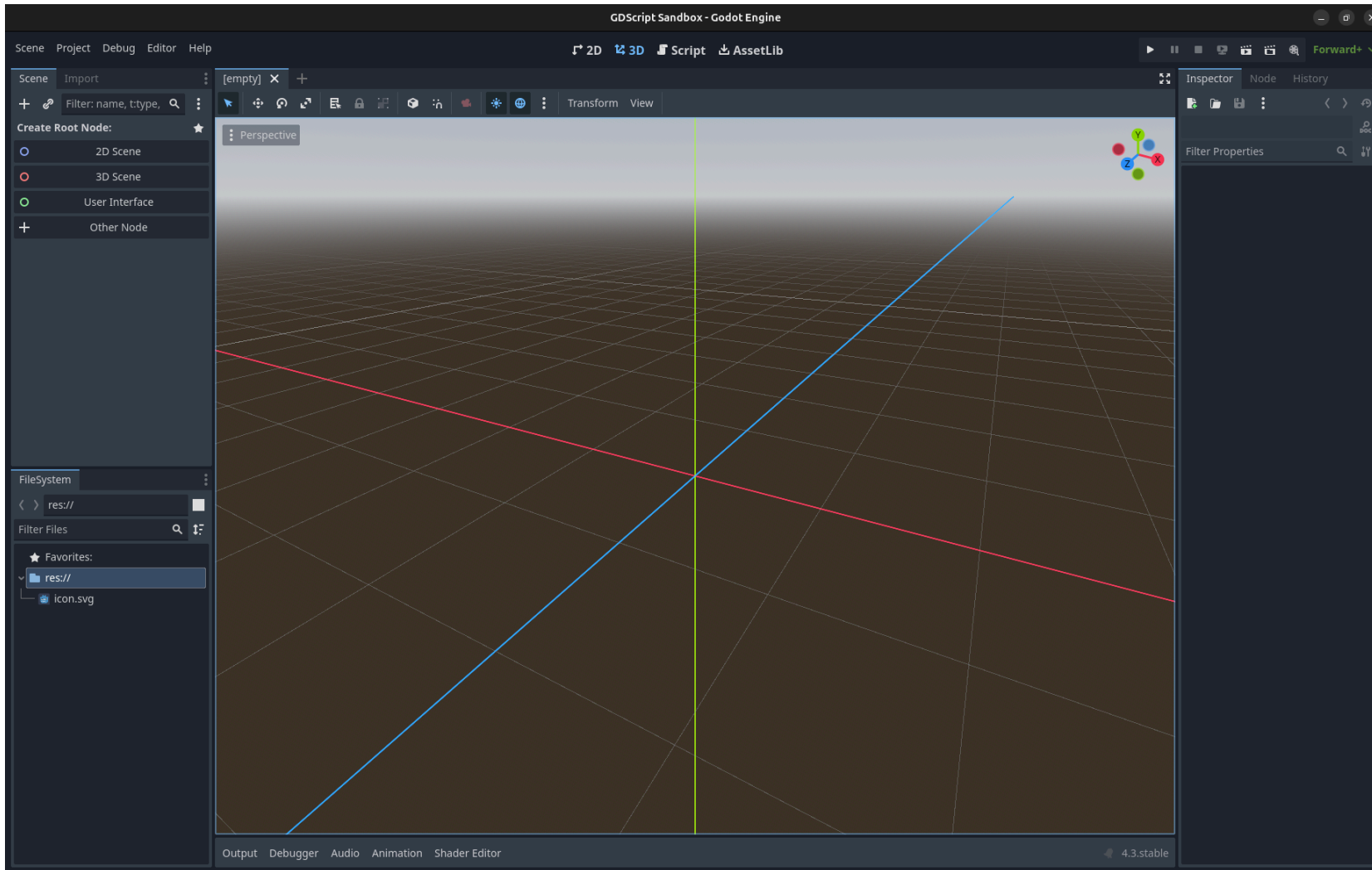
[Godot 官網](#)

- Linux
 - 下載  解壓縮  executable
- Windows/MacOS
 - ^_(\ツ)_/^

內建編輯器不難用，但是如果你堅持的話你可以 [這樣](#) 



破病還蠻多的，使用體驗可能跟內建編輯器沒有差非常多



The Godot Engine

遊戲是一個每 $1/\text{FPS}$ 秒跑一次的迴圈

- 處理輸入
- 更新遊戲物件
- 渲染畫面

遊戲是一個每 $1/\text{FPS}$ 秒跑一次的迴圈

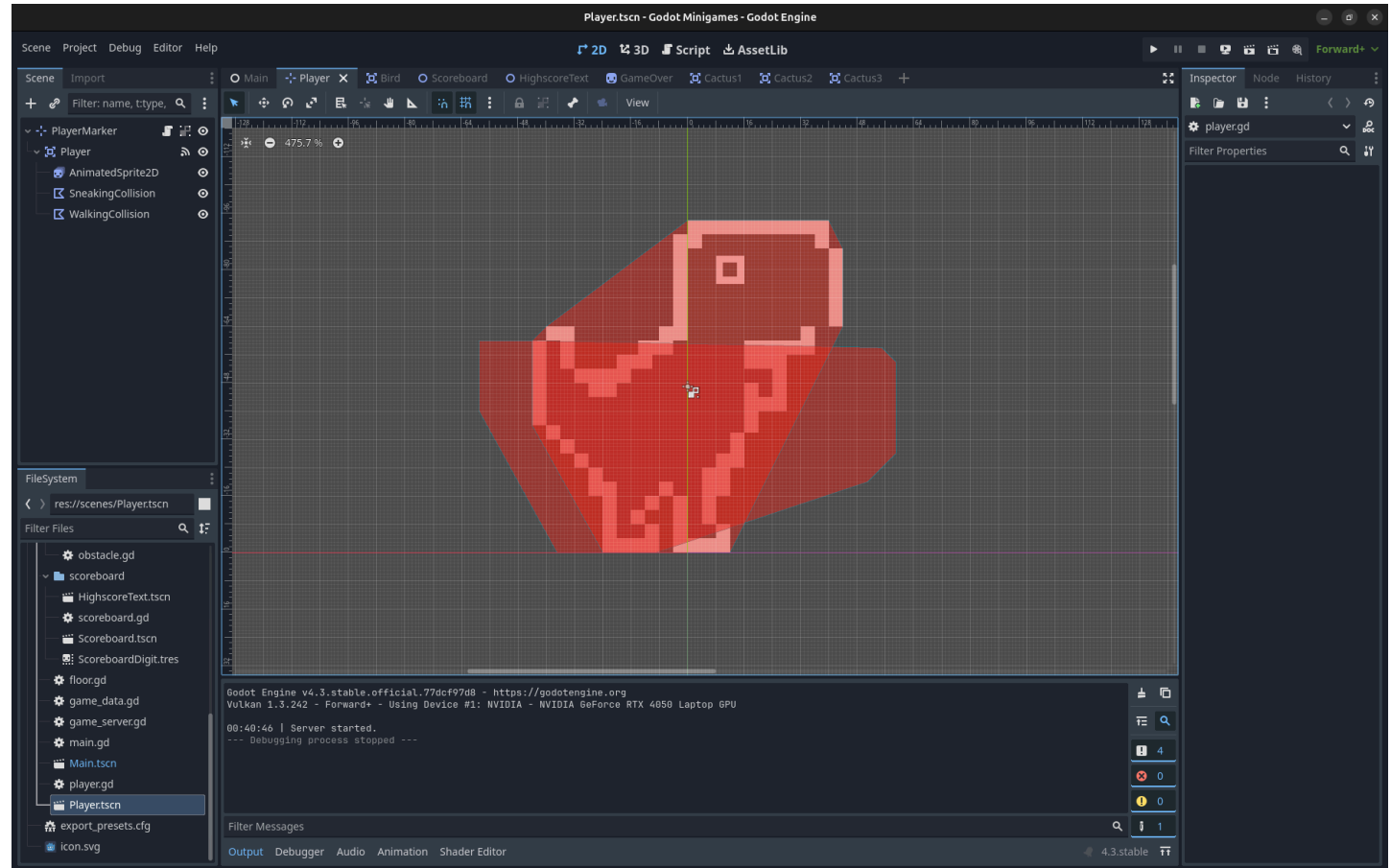
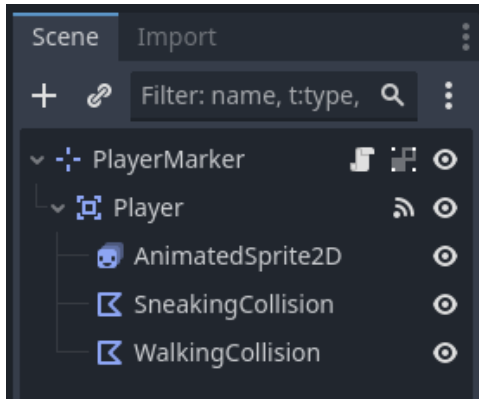
- 處理輸入
 - 攔截鍵盤、滑鼠事件，叫醒對的遊戲物件做出反應
- 更新遊戲物件
 - 更新物件狀態（大小位置、顯示材質、動畫）
 - 物件之間相互溝通、交互作用
- 渲染畫面
 - 計算物件位置
 - 顯示物件

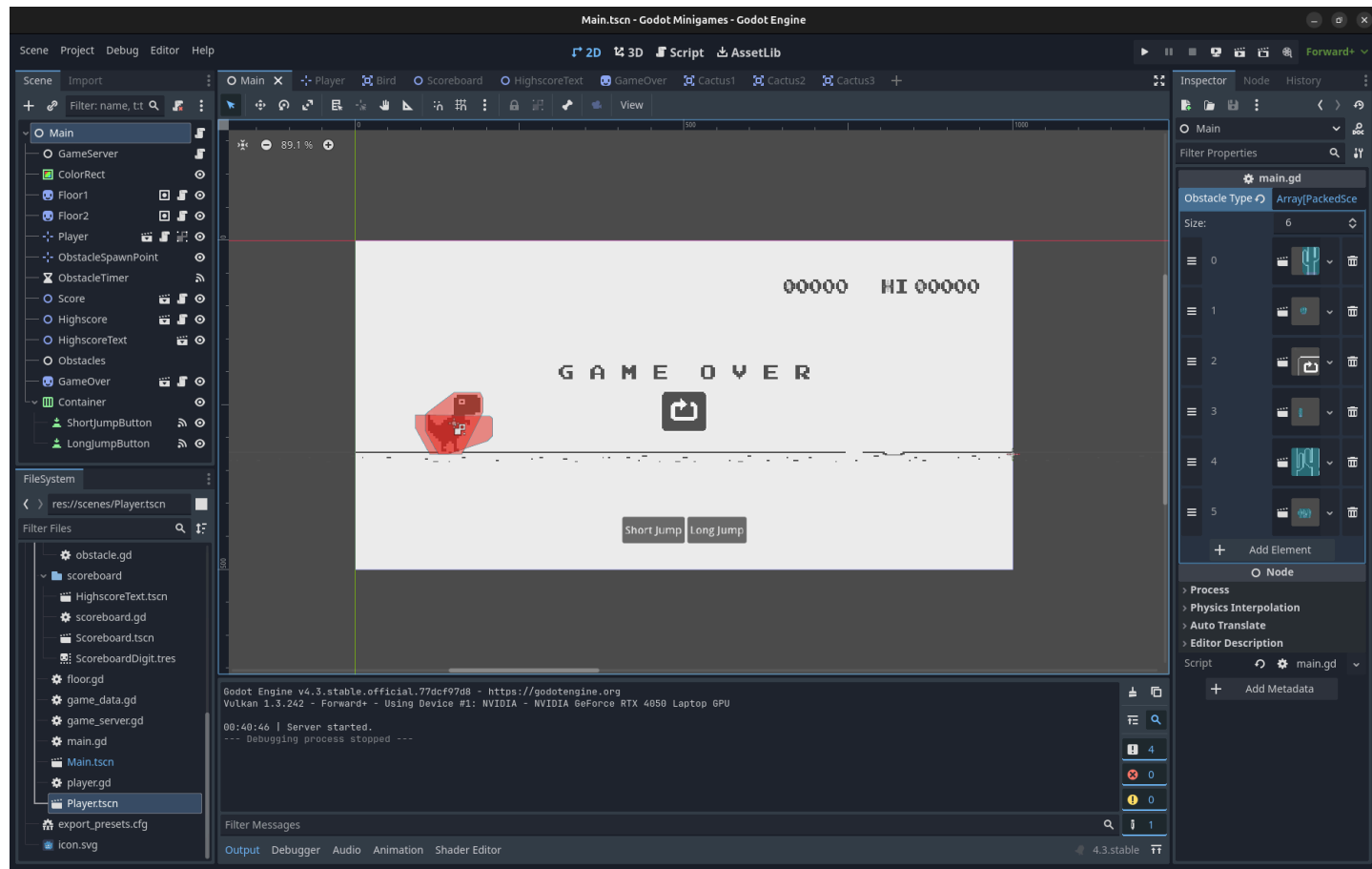
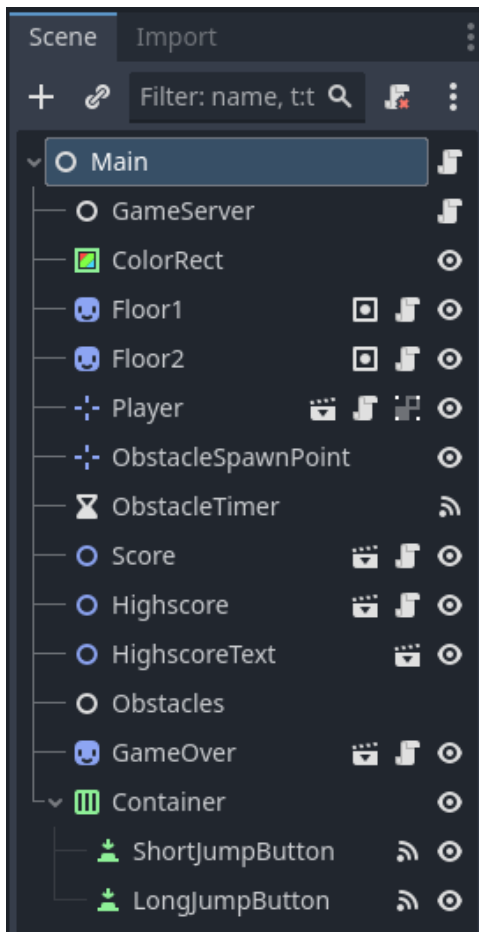
遊戲引擎的工作：

- 處理跟遊戲邏輯無關的工作
 - 渲染
- 讓你自己寫遊戲邏輯
 - 腳本或模組
- 提供一套工具讓你處理常見的任務
 - 輸入處理
 - 物件之間的溝通機制

[Overview of Godot's key concepts](#)

- Node：單一個遊戲物件，可以是一張圖片、一個碰撞箱
- Scene：一棵 Node 組成的樹，可以代表一個角色、一個場景
- Signal：node 之間溝通的管道
- Script：黏在一個 node 上，讓 node 有額外的行為

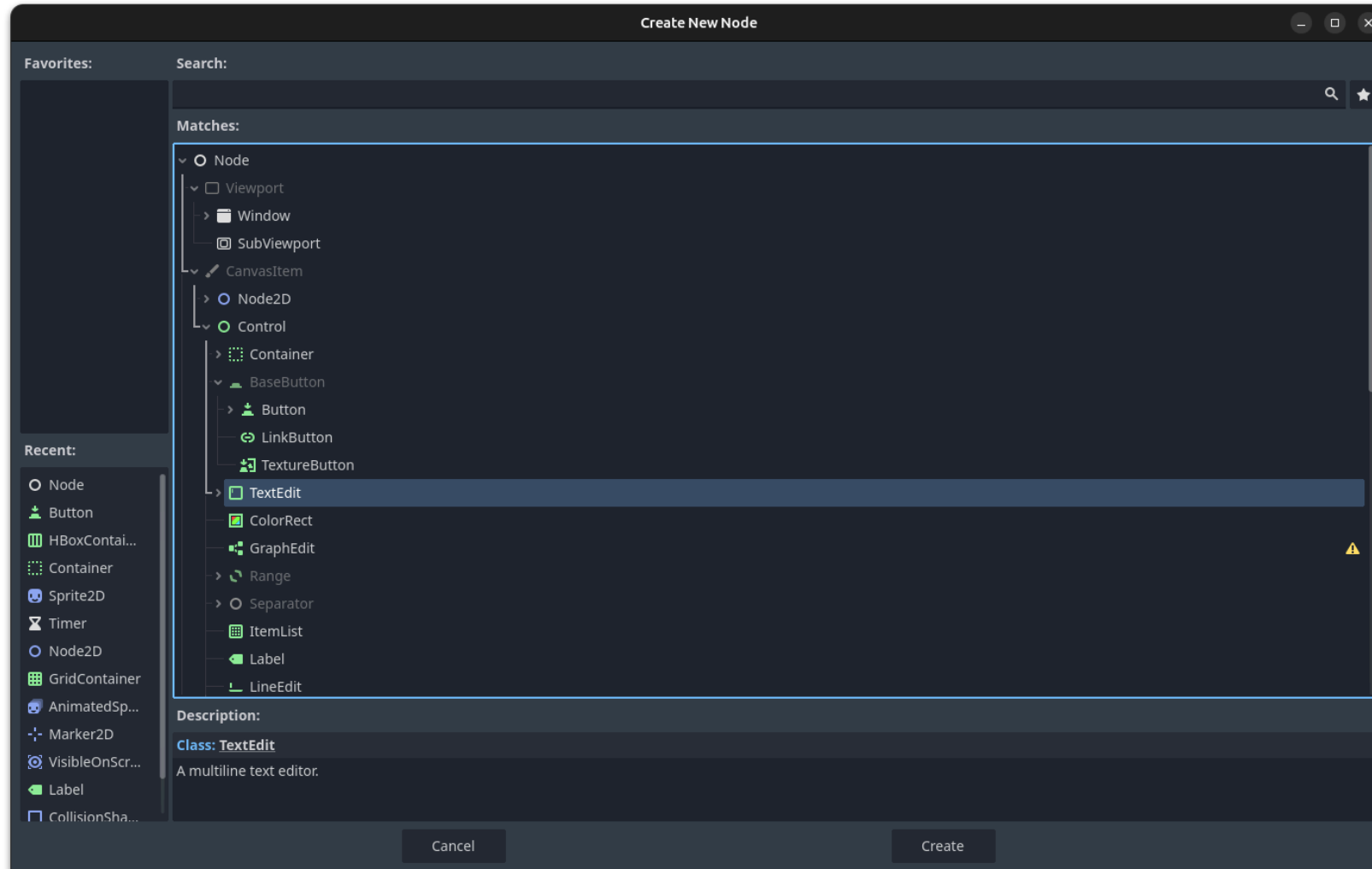




各種 Node 列表 [↗](#)

Node 有八百萬種，像是

- Sprite2D：一張 2D 圖片或材質
- RigidBody2D：處理 2D 物理 [Physics introduction](#) [↗](#)
- Button：按鈕
- Timer：計時器

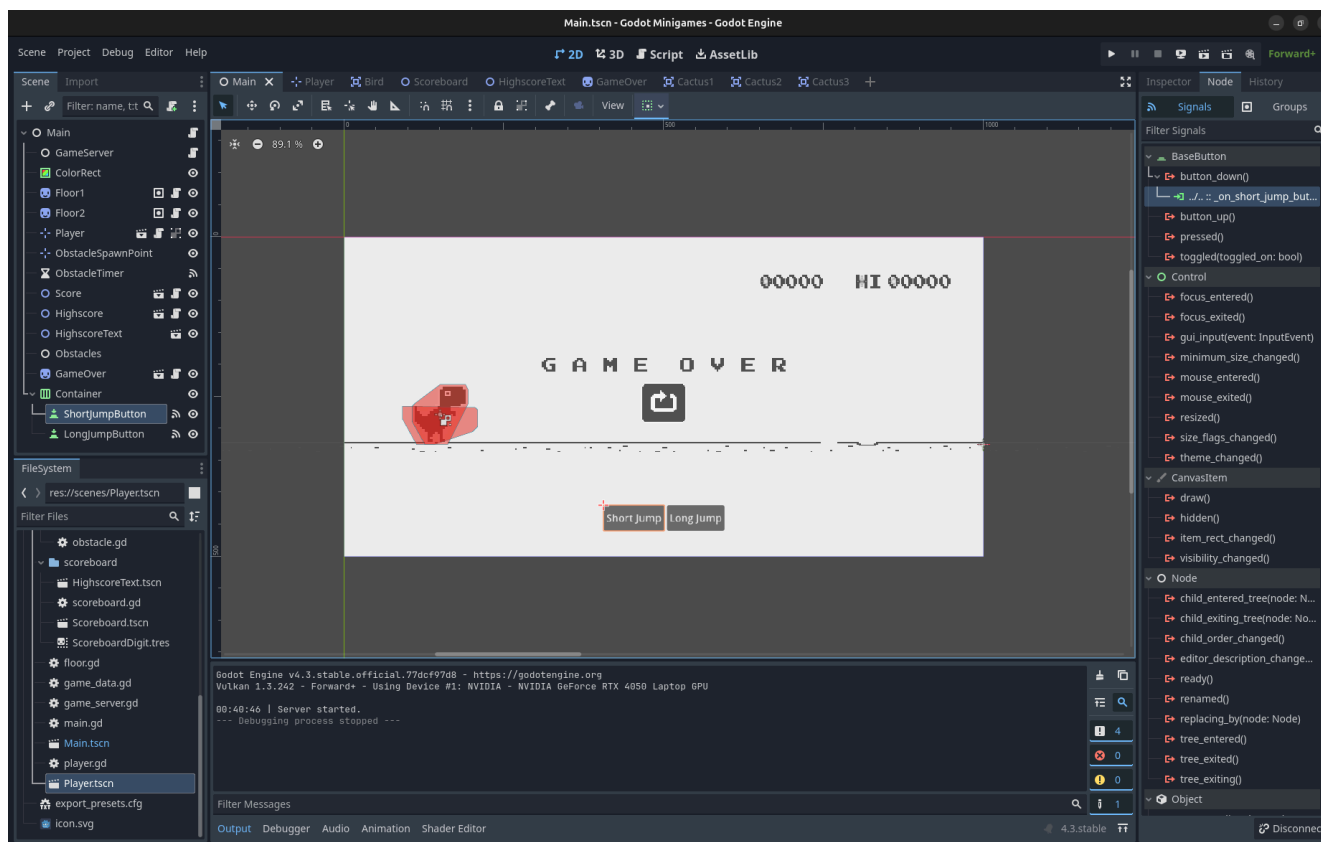


signal 是一種事件收發的機制，隨便一個人發信號，預先指定的一些函數就會被叫醒






- 按鈕被按下去，退出遊戲
- 多人遊戲 client 傳一個請求，傳回覆回去
- 血量歸零，遊戲結束
- ...etc.

跟 [Node.js EventEmitter](#) 和 [Challenge 2024 EventManager](#) 無限的像

signal 也有八百萬種，還可以自訂新的 signal



可以寫一份腳本黏在一個 node 上面，為那個 node 加上額外功能

- 每一幀往哪裡移動一點、往哪個方向轉動一點
([Idle and Physics Processing](#) )
- 接到輸入的時候做點什麼 ([Input Class](#) )
- 加入或刪除 node/scene ([Nodes and scene instances](#) )、
[Node.queue_free\(\)](#) )
- 對一整群 node 做一件事情 ([Groups](#) )

Godot 支援 GDScript 和 C ，我們今年計畫用 GDScript

例：小恐龍遊戲的根節點

```
1 func _process(delta: float) -> void:
2     if Input.is_action_pressed("ui_cancel"):
3         game_exit()
4     elif is_waiting && \
5         Input.is_action_just_pressed("game_jump"):
6         game_start()
7         is_waiting = false
8     elif is_playing:
9         game_update(delta)
```


GDScript

純新手向教學

- [Learn to code with GDScript - Godot Docs](#) 
- [Learn GDScript From Zero](#) 

你各位可能都不需要這個

[GDScript - Godot Docs](#) 

GDScript 很像 python ([Example](#) )，把 language reference 看過去你就大概會了

隨便瀏覽一下 [GDScript keywords](#) 的話，大部分都跟 python 長很像

- `as`：型別轉換
- `signal`：signal
- `preload`：有一點像 `import`

大概跟 python 差不多的部份

- Identifiers
- Operators
- Constants/Enums
- Statements and control flow
- Functions/Callable

就是你想的像 python 那樣

有一些內建的特別的 class method 叫做 `_init`、`_process` 之類的，像是 python `__init__` 的感覺

Prepend a single underscore (`_`) to virtual methods functions the user must override, private functions, and private variables

— [GDScript style guide](#) 

Type	Convention	Example
File names	snake_case	<code>yaml_parser.gd</code>
Class names	PascalCase	<code>class_name</code> <code>YAMLParse</code>
Node names	PascalCase	<code>Camera3D</code> , <code>Player</code>
Functions	snake_case	<code>func load_level():</code>
Variables	snake_case	<code>var particle_effect</code>
Signals	snake_case	<code>signal door_opened</code>
Constants	CONSTANT_CASE	<code>const MAX_SPEED = 200</code>
Enum names	PascalCase	<code>enum Element</code>
Enum members	CONSTANT_CASE	<code>{EARTH, WATER, AIR, FIRE}</code>

[GDscript operators](#) 

一大堆，隨便看過去大概知道有什麼就好
絕大部分跟 python 一樣

你想的那樣

enum 基本上就是一次讓你宣告一坨常數

```
1 enum {TILE_BRICK, TILE_FLOOR, TILE_SPIKE, TILE_TELEPORT}
2 # Is the same as:
3 const TILE_BRICK = 0
4 const TILE_FLOOR = 1
5 const TILE_SPIKE = 2
6 const TILE_TELEPORT = 3
```

跟你想的一樣

- `if`、`elif`、`else`
- `while`、`for`、`break`、`continue`

長得跟 `switch` 很像但是數學比較好的 `match` 魔法

```
1 match <test value>:  
2     <pattern(s)>:  
3         <block>  
4     <pattern(s)> when <pattern guard>:  
5         <block>  
6     <...>
```

魔法規則太多了所以 [match - Godot Docs](#) 



```
1 func function_name(arg1: int) -> void:  
2     pass
```

大概只是把 python 的 `def` 換成 `func`

- class method 不用把 self 加進參數 😊
- 不需要 `bind(self)`
- 比較嚴格的 type annotation 😊
- static ✅、lambda function ✅

跟 python 比較不一樣的部份

- Classes
- Types、Variables、Type Specification、Literals
- Exports、Annotations
- Signals、Await

- 一個檔案自己是一個 class，裡面可以有其他的 class
- `extends` 、`super` 
 - 沒有多重繼承
 - 沒有 virtual function/class，也許只能 `assert(false)`
- `ClassName.new()` (\approx `new ClassName()`)

[Object Class - Godot Docs](#)

- `_init()` (\approx `__init__(self)`)
- `free()`

[Node Class - Godot Docs](#)

- 初始化
 - `_enter_tree()`
 - `_ready()`
- game loop
 - `_process()`
 - `_physical_process()`
- 輸入處理
 - `_input()`
 - `_unhandled_input()`
- `queue_free()`

Initialization Order [↗](#)

1. `null` or default value
2. member variables *without* `@ready`
3. `_init()`
4. exported values
5. `_enter_tree()`
6. member variables *with* `@ready`
7. `_ready()`

大概跟 python 很像

```
1 var a                                # variant
2 var b = 48763                        # variant
3 var my_vector2: Vector2             # Vector2
4 var my_node: Node = Sprite2D.new()  # Node
5 var another_vector2 := Vector2()    # Vector2 (inferred)
```

瑣碎的細節

- casting (as) : subclass 轉 parent class、或是有好好定義的 conversion、或是吃 error
- static 、 [getters/setters](#)  

Variables can optionally have a type specification. When a type is specified, the variable will be forced to have always that same type, and trying to assign an incompatible value will raise an error.





— [Variables - Godot Docs](#) 



TLDR：python 的 type annotation 是在搞笑，但是 GDScript 的不是

- 有指定型別的話，你可以確定他一定是那個型別
- assign 垃圾給有指定型別的變數會噴 runtime error 但是執行前不會有警告
- 同理你有 [Typed array](#) 可以用，不過只能是一維的
- 但是 [GDScript 沒有 union 之類的酷炫型別](#)

一些比較有趣的內建型別

- Array、Dictionary、Callable：跟 python 差不多概念
- Variant：可以裝任意型別的變數
- [NodePath](#)  – 在 node tree 上的路徑
 - 不過靠 \$Node/Path 就可以解決很多問題
- [GDScript literals](#)  [Node.get_node](#) 
- Color、vector[23][i]、Transform[23]D
- [各種 Packed array](#) 
- Signal：signal

- Pass by value/reference 🧠
 - 大概跟 python 一樣
 - String、vector 之類的是 pass by value
 - Object、array、dictionary 之類的是 pass by reference
 - 不是內建的東西都算 Object
- None ➡ null、True/False ➡ true/false （我們需要秦始皇）

export 讓你的 class member 可以從右邊那欄編輯，並且跟 scene 一起存檔

有一大堆選項可以讓你的編輯區看起來很好看

[GDScript exported properties - Godot Docs](#) 

Annotation 列表 [↗](#)

- 看起來像給你的 property 或 method 一些 decorator（但不是）
- 不可以自訂，**GDScript 沒有 decorator** ([ref 1](#) [↗](#) [ref 2](#) [↗](#))

[Signal Class - Godot Docs](#)

signal 自己是一個 type, 可以宣告成一個 class member, 也可以當變數傳來傳去

- `signal_name.connect(callable)`
- `signal_name.emit(args)`

使用例：記分板和角色都是某個根節點的小孩，角色碰到某個道具要更新記分板？

- `await signal_name`：暫停執行，直到某個 `signal` 被觸發
- `await function_call(args)`：暫停執行，直到某個函數結束

有 `await` 的函數自動變 `coroutine`，要呼叫他就要 `await` 或是把回傳值丟掉

```
1 var v1 = await some_coroutine() # ok
2 var v2 = some_coroutine() # no
3 some_coroutine() # ok
```

看起來很像 python 或 javascript 的 `async/await`，但相當不一樣

- 一次只能 `await` 一件事
- 不能趁你 `await` 的時候先做別的事情等等再等他

Live Demo / Code Review?
