Chapter 2

Godot

In questo capitolo, vedremo come utilizzare il game engine Godot per sviluppare un videogioco RPG 2D.

Questa parte di programmazione, è pensata per essere seguita passo passo, utilizzando il materiale scaricabile dai link forniti come base di partenza per riproporre i passaggi che sono riportati in ogni sezione sul proprio computer. Nel capitolo sono inoltre presenti proposte di esercizi per fare pratica con il game engine e spingere il lettore a iniziare a progettare qualcosa in maniera indipendente da questo documento come, ad esempio, un proprio videogioco o anche dei semplici scenari di prova.

Per quanto ci piacerebbe, putroppo non riusciremo a vedere l'implementazione di ogni singola stanza del videogioco di esempio, per questioni di tempo, spazio ma anche per non risultare noiosi nel ripetere sempre le stesse azioni e, soprattutto, per non spoilerare la trama!

Le repository contenenti i progetti utilizzati nel presente capitolo sono tutte disponibili nella pagina GitHub `"Cosa si cela dietro ai pixel" [Godot]`[1]

2.1 Godot API e Documentation

In questa sezione capiremo cosa sono le API, come vengono utilizzate nei game engine, a cosa servono nella realizzazione di un videogioco e quali sono presenti dentro Godot; infine, vedremo come navigare al meglio la documentazione ufficiale di Godot, disponibile gratuitamente online, per chiarire dubbi. approfondire argomenti, cercare spiegazioni sul funzionamento di funzioni e soddisfare qualche curiosità. Saper navigare una documentazione è una skill fondamentale per un programmatore, in futuro si presenteranno innumerevoli occasioni in cui ci si ritroverà a leggerne una. È dunque buona norma imparare a leggerle fin da subito.

2.1.1 Che cos'è una API

API è l'acronimo di "*Application Programming Interface*", ovvero, un software intermediario che consente la comunicazione tra due applicazioni. In altre parole, una API non fa altro che aiutare due sistemi software a comprendersi a vicenda.

Per capire meglio questo concetto, possiamo immaginare di essere a un ristorante. In questo setting, abbiamo tre figure chiave:

[1] `https://github.com/orgs/rpg-course-godot/repositories`

1. il cliente (User)
2. il cameriere (API)
3. la cucina (Data Source)

Immaginiamo di essere seduti a un tavolo, pronti per ordinare. Anche se sappiamo già cosa vogliamo, l'unico modo per comunicare il nostro ordine alla cucina è tramite il cameriere. In questo scenario, il cameriere rappresenta la API. Noi, i clienti, siamo gli utenti che hanno bisogno di dati o di un particolare servizio, mentre la cucina è la data source o il sistema che processa la nostra richiesta. Quando comunichiamo il nostro ordine al cameriere, non è lui a cucinare ma, bensì, passa la nostra richiesta alla cucina. Una volta che i nostri piatti sono pronti, il cameriere - proprio come fa la API - porta le pietanze dalla cucina a noi. Questo processo è una metafora perfetta per comprendere il funzionamento di una API. Essa prende una richiesta da un utente (il cliente), fa il fetch dei dati o dei servizi richiesti dal sistema (la cucina) e, infine, consegna la risposta all'utente (il cliente).

Tutti i framework e i game engine utilizzano le API per offrirci le funzionalità di cui abbiamo bisogno.

2.1.2 Cos'è una Game Engine API

Alcuni game engine non sono *feature rich*, oppure sono più feature rich in un ambito rispetto che a un altro. L'elenco delle funzionalità che un game engine mette a disposizione è detta "*documentazione API*". Tutti i game engine hanno API simili che forniscono strumenti utili per la realizzazione di un videogioco come, ad esempio, ricevere input utente. Proprio per questo motivo, se si sa realizzare videogiochi in un game engine, molto probabilmente si saprà farlo anche in altri o, quantomeno, si saprà cercare le API di quel particolare game engine che si sta utilizzando.

Il principio core di una API è dunque quello di aspettare un input per offrire un output. Quando utilizziamo una game engine API, tuttavia, non sempre siamo noi in prima persona a doverle fornire un input, ma è bensì il game engine a farlo per noi dietro le quinte. Vi sono quindi casi in cui noi riceviamo solamente l'output. Per far fronte a questa evenienza e capire quale tipo di API abbiamo avanti, ci viene in aiuto la documentazione API.

2.1.3 Game Engine API Format

Nonostante le documentazioni API potrebbero essere diversa a seconda del game engine o game framework che si sta utilizzando, esse presentano tutte il seguente format di base:

- Nome della classe
  - Descrizione della classe, che spiega cosa fa la classe e come usarla, a volte anche mediante esempi di codice
  - Proprietà della classe (variabili)
    - Descrizione di ogni singola proprietà di classe, che illustra, ad esempio, il tipo di data type della proprietà

  - Metodi della classe (funzioni)
    * Descrizione di ogni singolo metodo di classe, che chiarisce se il metodo aspetta parametri di input, se gli input sono opzionali oppure no, il tipo di output e se gli output sono previsti oppure no

Per quanto riguarda invece il "consumare" una API, ovvero il come utilizzare il risultato di una API, questo aspetto dipendente fortemente da quale game engine si sta utilizzando. Nella maggior parte dei casi si deve importare le classi API all'interno degli script prima di poterla utilizzare, tuttavia, Godot gestisce le API in maniera un po' diversa.

2.1.4 Godot API

Con GDScript, ovvero il linguaggio di programmazione nativo di Godot che utilizzeremo nel corso di questo documento, non c'è alcun bisogno di importare una classe. Tutte le classi o hanno uno scope "globale", oppure non hanno bisogno di essere istanziate in un variabile. Come se non bastasse, quando si estende una classe messa a disposizione da Godot e dalla sua sezione API, non solo abbiamo accesso a tutte le sue proprietà e ai suoi metodi, ma anche a tutte funzioni di classe ereditate da essa.

```
extends Node2D

# Now we have access to the Node2D Class API
```

Una cosa abbastanza importante da tenere a mente, è che quando non si usa la keyword `extends` per estendere da una classe già esistente, il file gdscript estenderà dalla classe `RefCounted` di default.

```
# extends Resource

# Now we have access to the Resource Class API
```

In GDScript, inoltre, possiamo ottenere l'accesso a una classe API instanziandola. Ad esempio, se vogliamo utilizzare la classe *RandomNumberGenerator*, possiamo istanzarla e dunque avere accesso a tutti i suoi metodi e alle sue proprietà mediante una variabile.

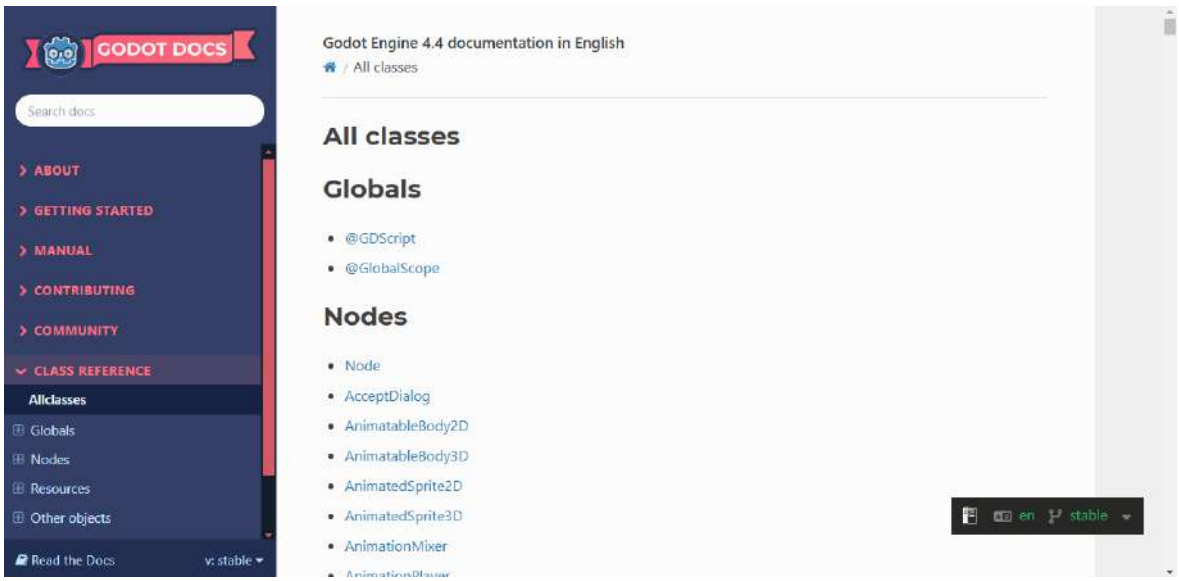
```
var range = RandomNumberGenerator.new()
```

2.1.5 Godot Documentation

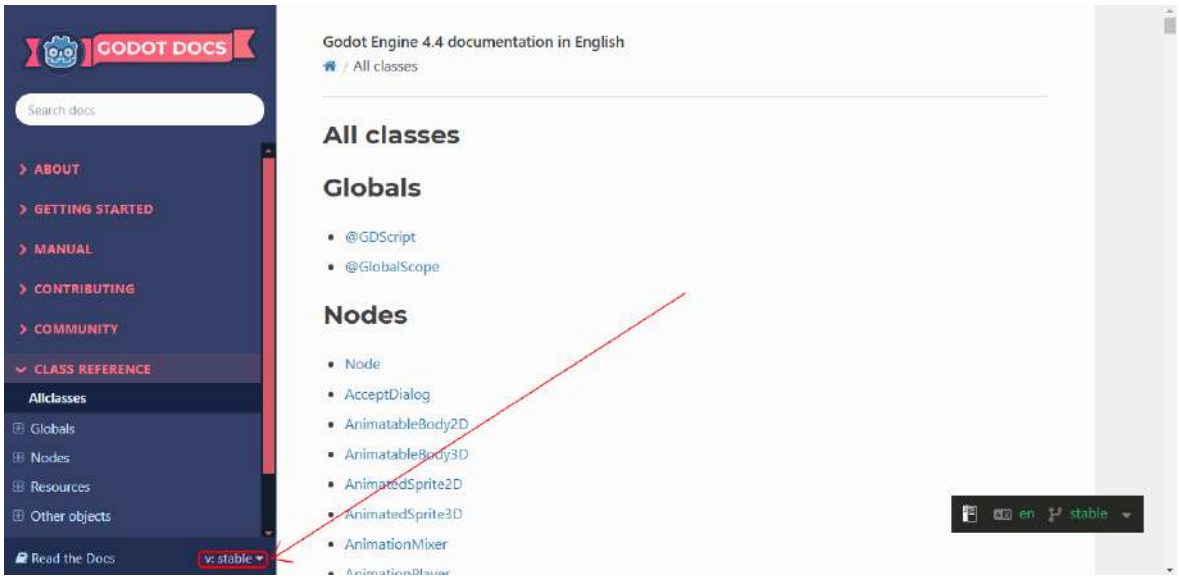
Come già detto in precedenza, Godot è un game engine open source gratuito. Ciò significa che è la community ad aiutare nel migliorare non solo il codice, ma anche la documentazione che Godot fornisce sul proprio sito web. Poiché, quindi, proprio per

questo motivo, la documentazione API potrebbe non essere aggiornata frequentemente e/o spiegare qualcosa in modo chiaro e dettagliato, illustreremo ora a grandi linee come leggerla.

Una volta entrati sul sito della documentazione API[2], la quale, al momento della stesura di questo documento, è aggiornata alla versione 4.4, dovremmo avere una schermata simile a quella in figura.

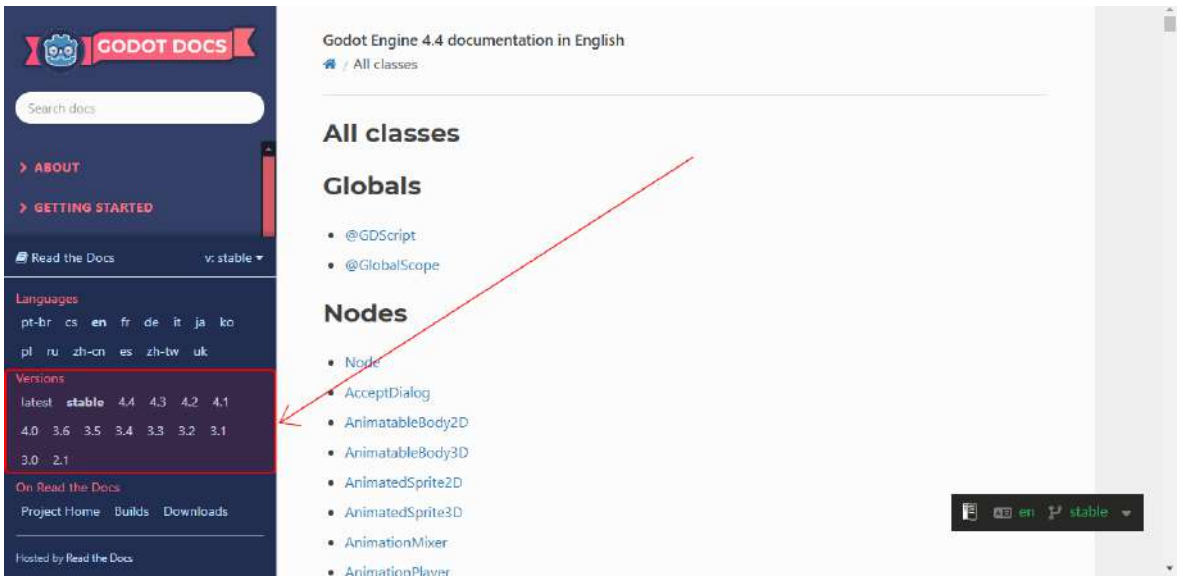


La versione della documentazione può essere cambiata cliccando sull'apposito pulsante in basso a sinistra del menu



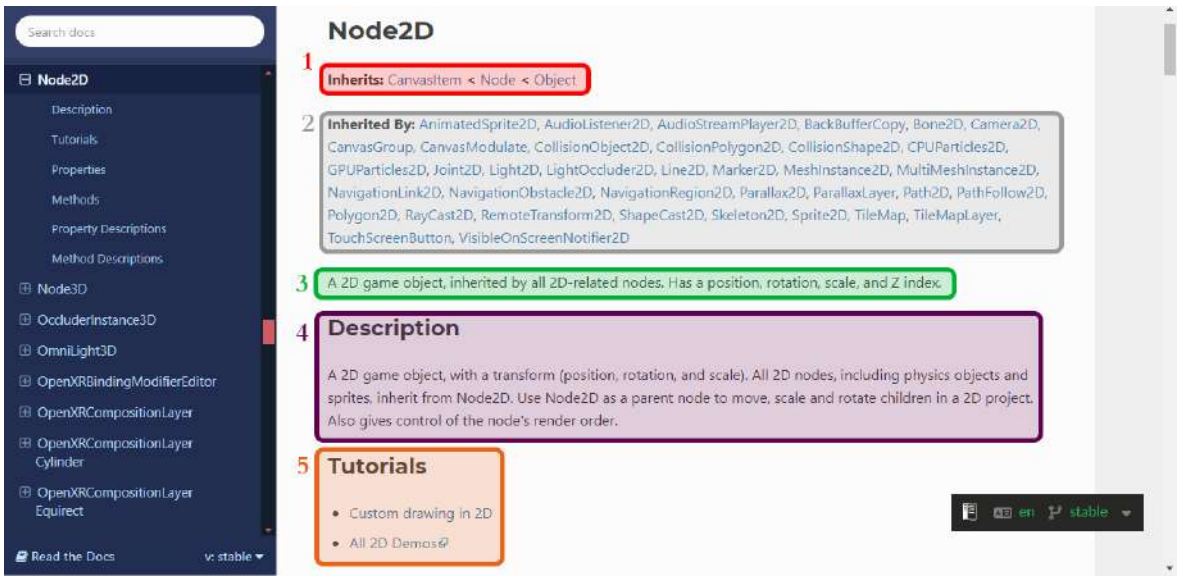
e selezionando la versione desiderata dalla nuova finestra

[2] `https://docs.godotengine.org/en/stable/classes/index.html`



La schermata home della documentazione API di Godot che abbiamo davanti, presenta un elenco di tutte le classi presenti all'interno del game engine. Clicchiamo ad esempio sulla classe "*Node2D*"

La nuova pagina a cui saremo reindirizzati presenta diverse informazioni utili per capire cosa fa e a cosa serve la classe Node2D.

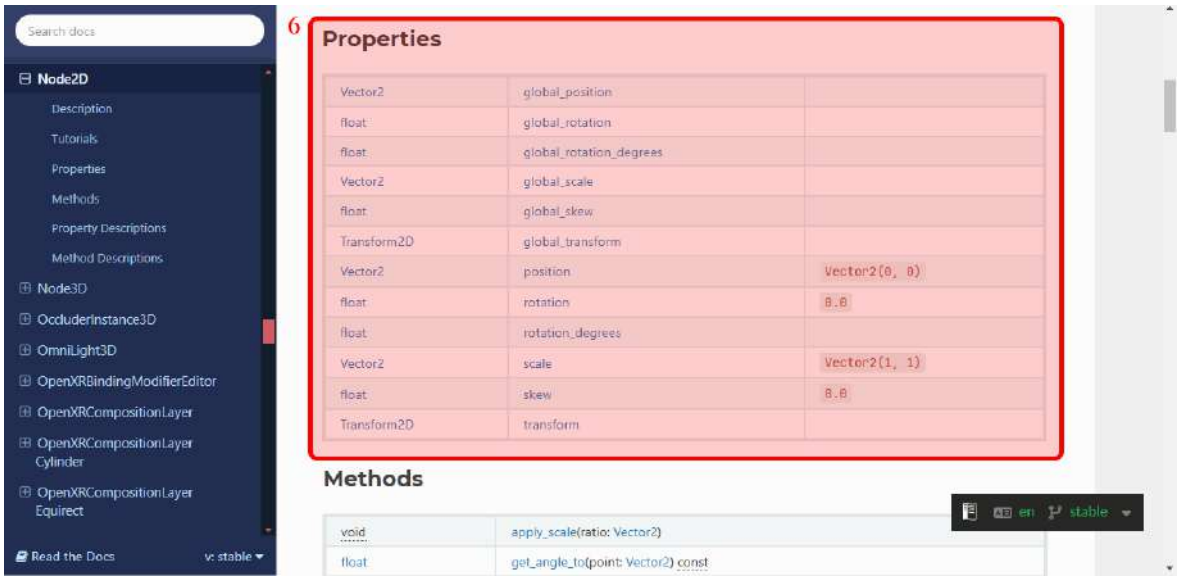


Una delle prime cose che saltano subito all'occhio è la sezione *Inherits* (1), nella quale vengono elencati le classi da cui eredita la classe che stiamo esaminando. Nel nostro caso, la classe Node2D eredita dalla classe *CanvasItem*, la quale a sua volta eredita dalla classe *Node*, che eredita infine dalla classe *Object*. Ciò significa che, estendendo dalla classe Node2D, avremmo accesso a tutti i metodi e le proprietà della classe RefCounted e di quelle Object, Node, CanvasItem e Node2D.

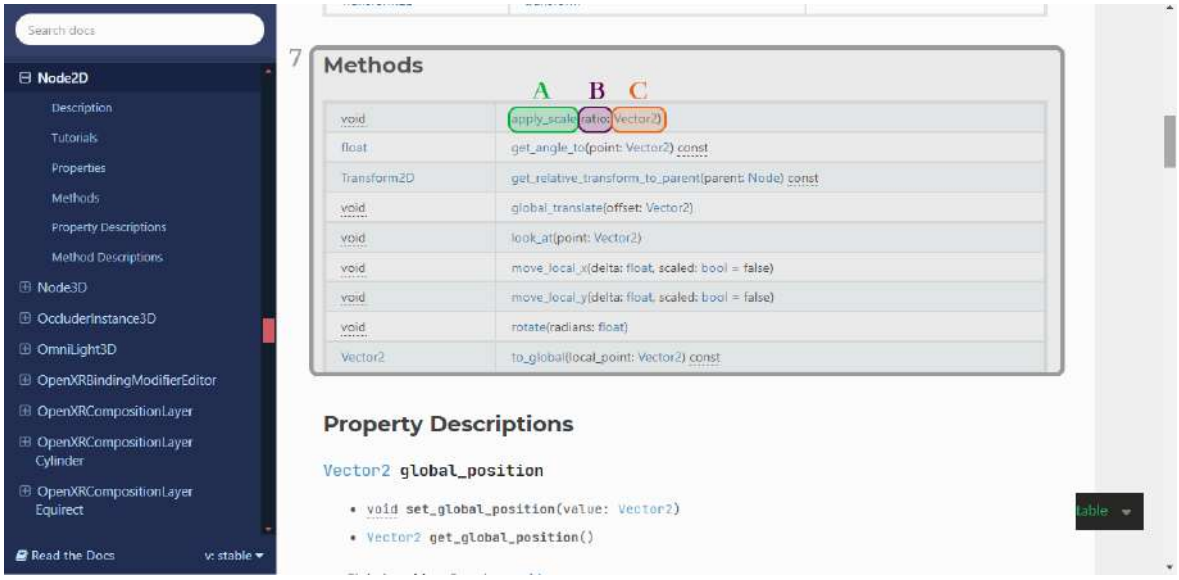
La sezione che segue quella *Inherits*, ovvero *Inherited By* (2), ci presenta invece una lista di tutte le classe che ereditano dalla classe (Node2D nel nostro caso di esempio) che stiamo esaminando.

La scritta subito dopo (3) è invece una breve descrizione, generale e di alcune proprietà principali, della classe in esame.

Continuando in verticale troviamo poi la *Description* (4), ovvero la descrizione dettagliata della classe; eventuali *Tutorials* (5) correlati; l'elenco delle proprietà (variabili) della classe (6), lì dove la prima colonna indica il tipo della variabile, la seconda il nome e, infine, la terza il valore di default;



e la tabella dei metodi (funzioni) (7), lì dove la prima colonna è il tipo di ritorno (*void* indica che non è previsto nessun tipo di ritorno), mentre la seconda è la "*firma del metodo*", composta dal nome del metodo (A), i parametri in input (B) e il loro tipo (C).



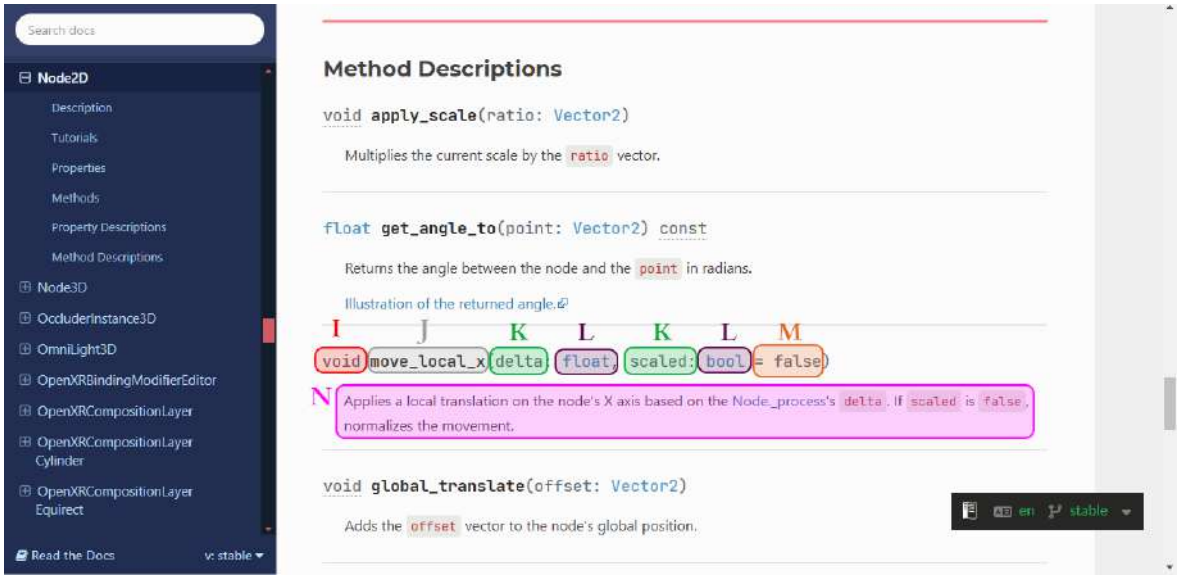
Le sezioni rimanenti contengono invece le spiegazioni dettagliate delle proprietà (*Property Descriptions*) e dei metodi (*Method Descriptions*) della classe considerata. Esaminiamo ad esempio la proprietà "*global_rotation_degrees*".

La prima riga indica appunto il tipo (D) e il nome (E) della proprietà, mentre l'elenco puntato specifica i metodi *setter* (F) e *getter* (G) (sempre nel solito format: tipo di ritorno, nome del metodo, nome e tipo degli input), lì dove:

- il metodo setter è utilizzato per assegnare valori alla proprietà
- il metodo getter è utilizzato per prendere il valore della proprietà

L'ultima riga (H), infine, è una descrizione della proprietà

Per quanto riguarda i metodi, se consideriamo ad esempio "*move_local_x*"



vediamo elencate, in ordine, il tipo di ritorno del metodo (I), il nome del metodo (J), il nome degli input (K) (se vi sono più input, essi sono separati da una virgola), oltre che il tipo (L) e l'eventuale valore di default (M) (ovvero quel valore che viene assegnato all'argomento del metodo nel caso in cui non venisse fornito nessun valore in input) e, infine, la descrizione del funzionamento del metodo stesso (N).

È importante far notare come, nella descrizione del metodo di esempio (N), non sia stato spiegato il metodo utilizzato per eseguire la traslazione. Questo perché noi "consumiamo" una API come una scatola chiusa, ovvero, senza sapere quello che fa l'engine dietro le quinte. Il lavoro che svolge Godot a noi non interessa, ci interessa solo il risultato. Tutte le spiegazione dei metodi sono infatti molto brevi e concise poiché non si soffermano sulla logica che c'è dietro, come è giusto che sia.

Per concludere, le pagine di dettaglio delle API non sono tutte come quella che abbiamo visto. Alcune possono comprendere sezioni aggiuntive o averne addirittura di meno, oppure altre possono avere codice di spiegazione del funzionamento e altre no. È proprio questo il problema di cui si stava parlando prima. Non sempre la documentazione fornisce tutti dettagli e le spiegazioni necessarie per comprendere il funzionamento di una API ma, a volte, queste bisogna capirle studiandole, testando e/o chiedendo sul forum generale della community[3]. Il che è comunque un'ottima forma di allenamento, se non uno dei migliori modi per crescere veramente come programmatori.

Non importa quanta esperienza si ha nella programmazione, è impossibile ricordare tutte le API a memoria; neanche noi le conosciamo tutte, ma sappiamo tuttavia cosa cercare e dove; ed è questo quello che fa davvero la differenza tra un buon e un cattivo programmatore.

[3] `https://forum.godotengine.org/`

2.2 Introduzione a Godot

In questa sezione illustreremo la struttura del game engine, l'interfaccia, gli strumenti principali e alcuni principi chiave estremamente utili per la programmazione di videogiochi.

2.2.1 Godot versioning

La policy di rilascio di Godot è in continua evoluzione.

Godot segue vagamente il *Semantic Versioning*[4] con un versioning system `major.minor.patch`, sebbene con un'interpretazione di ogni termine adattata alla complessità del game engine:

- la versione `major` viene incrementata quando si verificano importanti problemi di compatibilità che implicano lavori di porting significativi per spostare i progetti da una versione major a un'altra Ad esempio, per effettuare il porting di progetti Godot da Godot 3.x a Godot 4.x bisogna prima far passare il progetto attraverso un tool di conversione e successivamente effettuare comunque alcune modifiche manuali per sistemare quello che il tool non è stato in grado di convertire automaticamente.
- la versione `minor` è incrementata quando vengono rilasciate feature che non vanno a creare problemi di compatibilità importanti. Problemi minori di compatibilità potrebbero presentarsi in versioni minor, ma la grande maggioranza dei progetti non dovrebbe esserne influenzata o richiedere lavori di porting significativi.

Consiglio

Effettuare l'upgrade di una versione minor è raccomandata per tutti gli utenti, ma è necessario comunque del testing per assicurarsi che il progetto continui a funzionare come dovrebbe

- la versione `patch` viene incrementata per release di manutenzione che si incentrano a risolvere bug e problemi di sicurezza, implementando nuovi requisiti per il supporto della piattaforma e il miglioramento dell'usabilità backporting safe. Le patch release sono retrocompatibili. Le versioni patch potrebbero includere nuove feature minori che non impattano sull'API esistente e che, quindi, non rischiano di influenzare i progetti esistenti

Consiglio

Effettuare l'aggiornamento a nuove versioni patch è dunque considerato sicuro ed è fortemente raccomandato a tutti gli utenti di un dato stable branch

[4] `https://semver.org/`

Chiamiamo le combinazioni `major.minor` *stable branches*. Ciascun stable branch inizia con una release `major.minor` (senza lo `0` per `patch`) ed è ulteriormente sviluppato per release di manutenzione in un branch Git con lo stesso nome (ad esempio i patch update per il 4.0 stable branch sono sviluppati nel Git branch `4.0`).

In questo documento utilizzeremo la versione `4.1.3 stable`[5] come riferimento per qualsiasi spiegazione, esercizio o approfondimento.

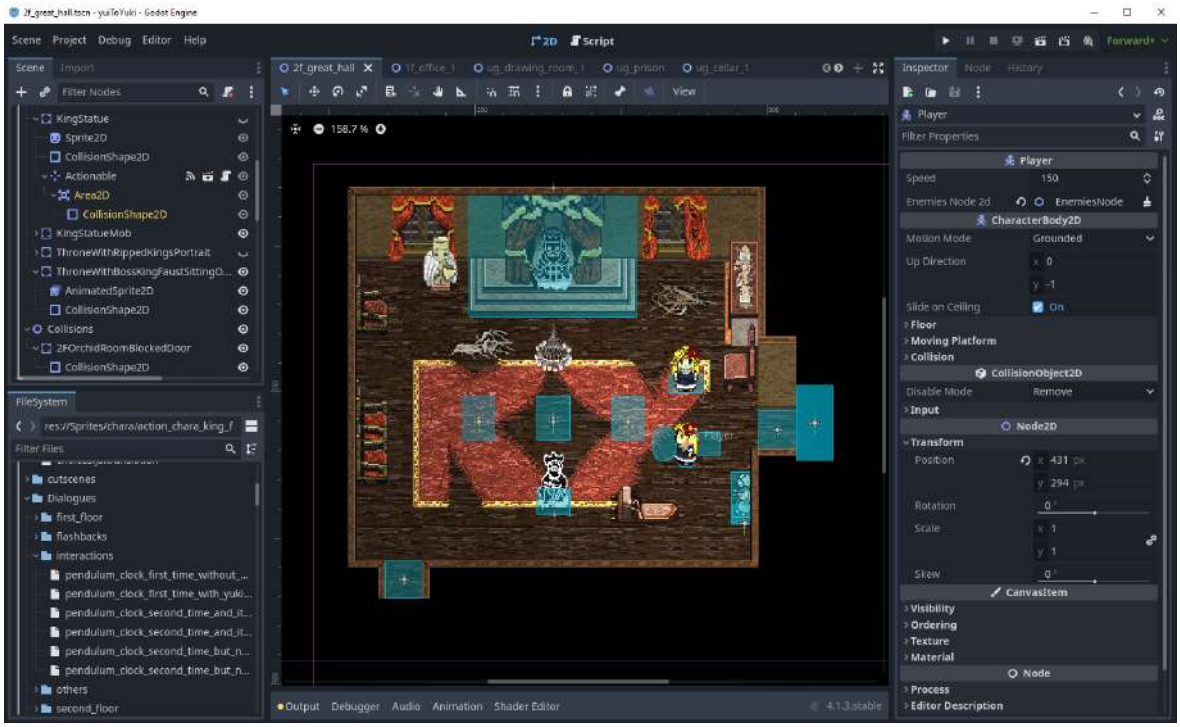
2.2.2 Overview dei principi chiavi di Godot

Tutti i game engine ruotano attorno alle astrazioni che vengono utilizzate per costruire i videogiochi. In godot, un gioco è un **albero** (**tree**) di **nodi** (**nodes**) che vengono raggruppati insieme in **scene** (**scenes**). Questi nodi possono poi essere "cablati", in modo da poter comunicare tra loro, utilizzando i **segnali** (**signals**).

Quelli appena elencati sono i quattro concetti principali che analizzeremo brevemente nelle prossime pagine in modo da fornire un'idea generale sul funzionamento del game engine.

2.2.2.1 Scenes

In godot, ogni gioco viene scomposto in scene (scenes) riutilizzabili. Una scena può essere un personaggio, un'arma, un menu dell'interfaccia utente, una singola casa, un intero livello, o qualsiasi altra cosa venga in mente.

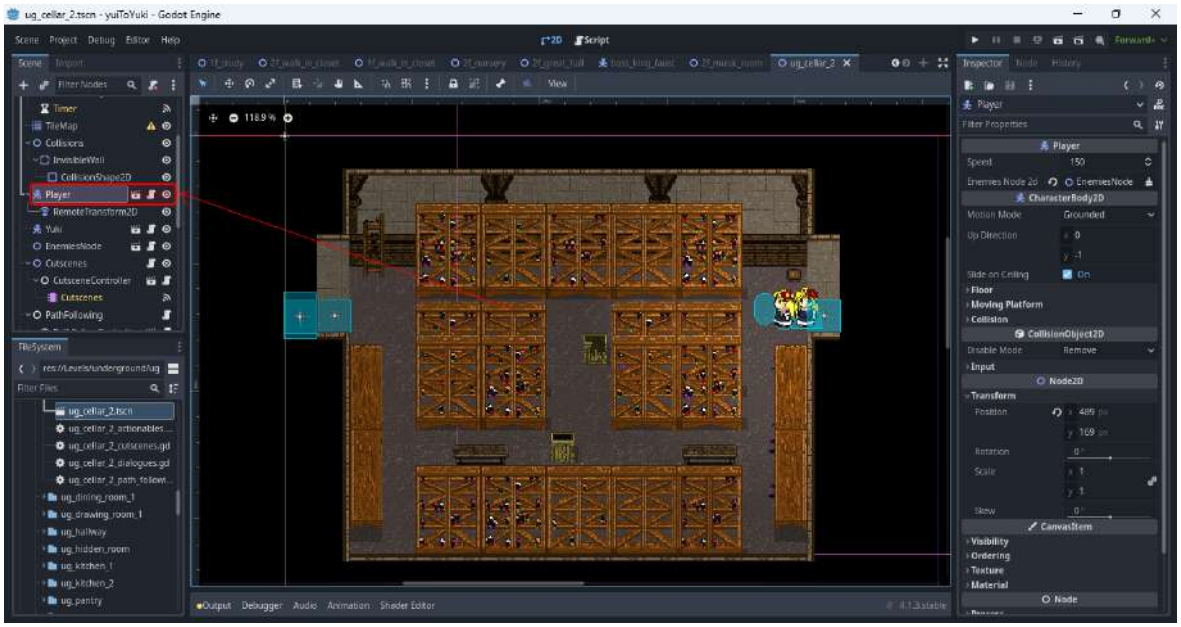


Le scene di Godot sono flessibili; ricoprono sia il ruolo di "*scene prefabbricate*" (in inglese abbreviato in "*prefabs*"), sia quello che hanno le scene di qualche altro game engine.

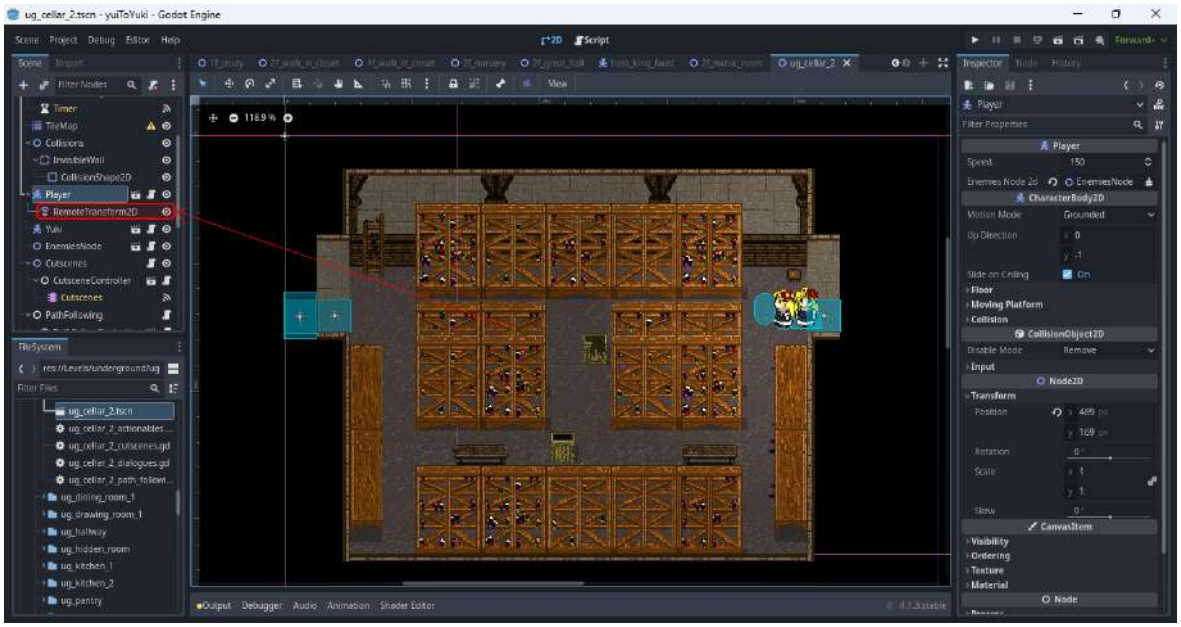
[5] `https://godotengine.org/download/archive/4.1.3-stable/`

Le scene possono anche essere innestate tra loro. Ad esempio, possiamo inserire il nostro personaggio in una scena che rappresenta un livello, e poi fare drag e drop di una scena sulla scena del personaggio, in modo da rendere la scena che abbiamo appena trascinato una scena figlio della scena personaggio.

Una scena che presenta una o più scene al suo interno è detta "*scena padre*" rispetto a quelle scene.

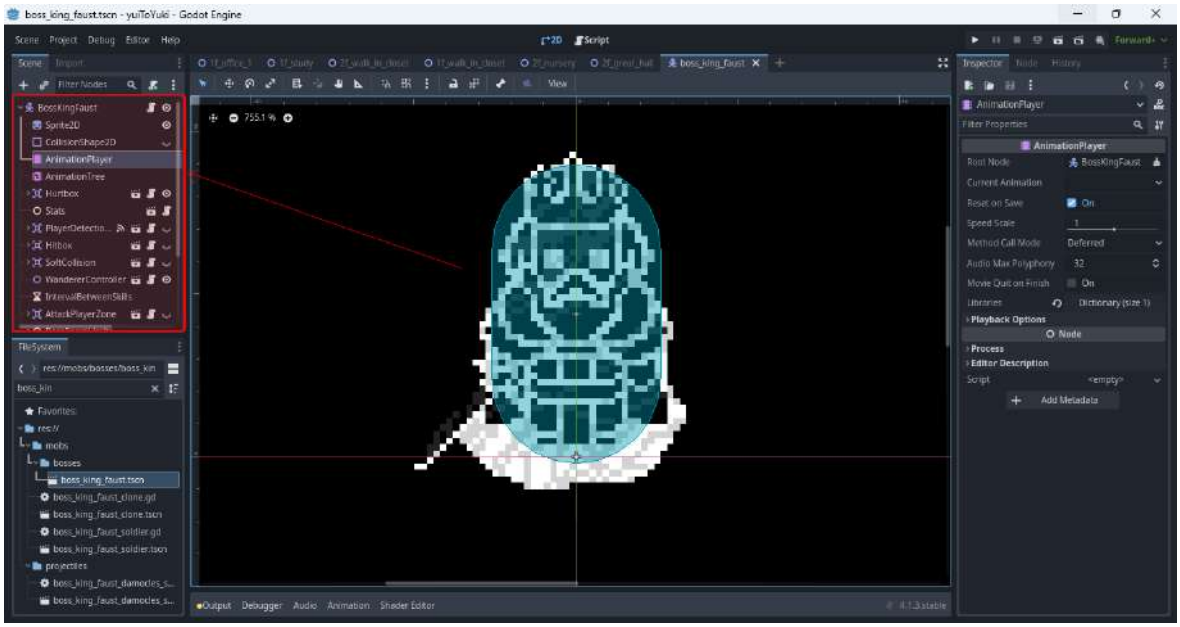


Una scena che ha una scena padre è detta "*scena figlio*" di quella particolare scena padre.



2.2.2.2 Nodes

Una scena è formata da uno o più nodi (nodes). I nodi, che organizziamo in alberi, sono i blocchi più piccoli del nostro videogioco. Segue un esempio dei nodi che compongono un personaggio

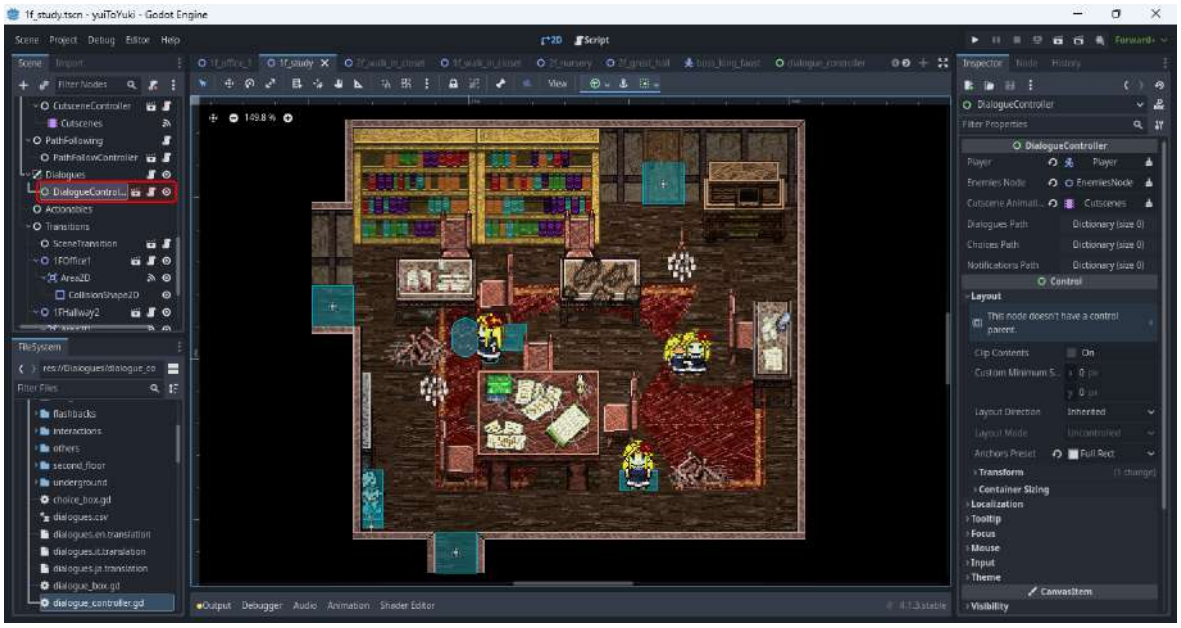


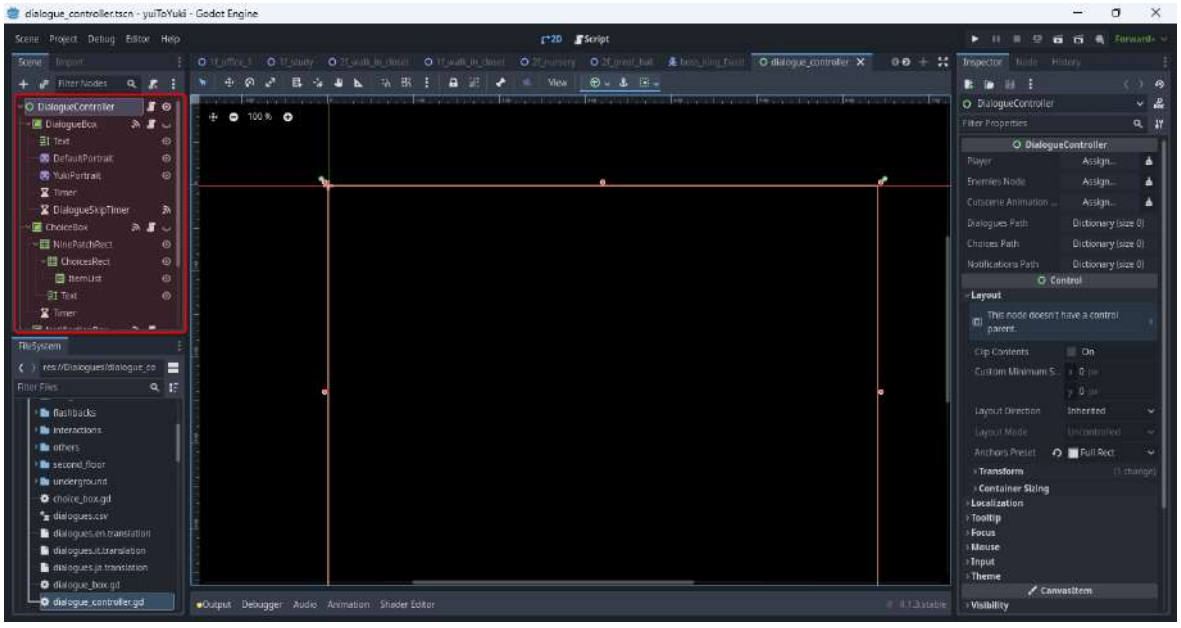
Come possiamo vedere, questo personaggio è formato dal nodo `CharacterBody2D` chiamato "BossKingFaust", un nodo `Sprite2D`, un nodo `CollisionShape2D` e così via.

Nota

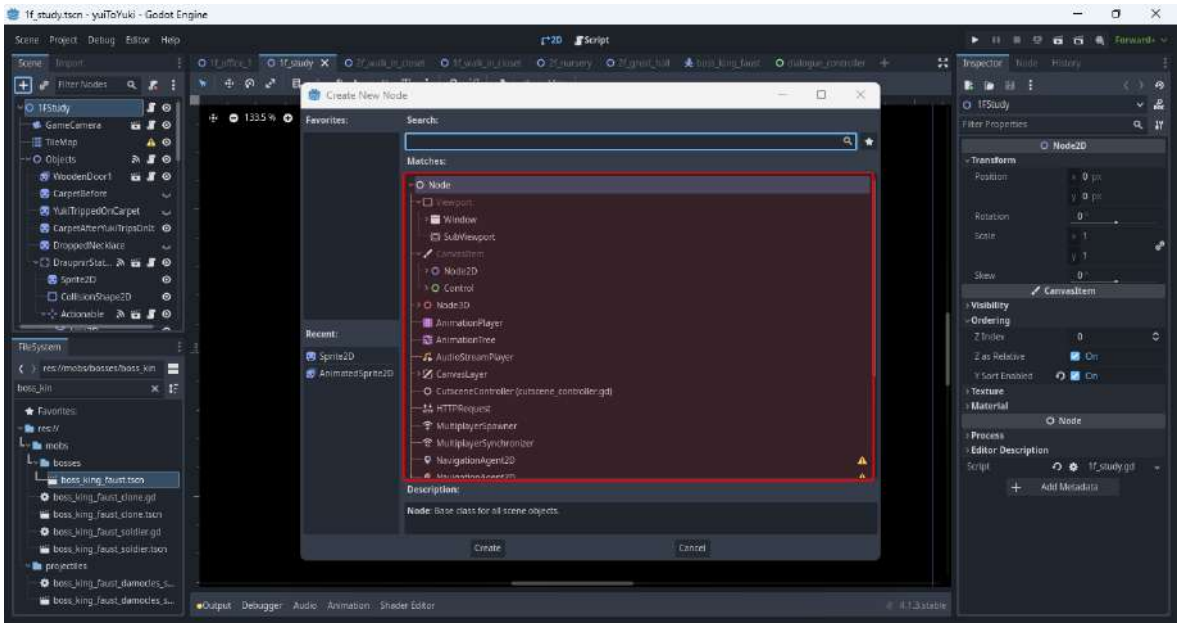
I nomi dei nodi finiscono con "2D" perché questa è una scena 2D. Le loro controparti 3D hanno nomi che terminano con "3D". Bisogna tenere inoltre presente che i nodi "Spatial" sono chiamati "Node3D" a partire da Godot 4.

È importante far notare come nodi e scene abbiano lo stesso aspetto nell'editor. Quando si salva un albero di nodi come una scena, questo verrà mostrato come un singolo nodo poiché la sua struttura interna verrà nascosta dall'editor.





Godot offre una libreria estensiva di tipi di nodi base che possono essere combinati ed estesi tra di loro, per crearne di più potenti. I nodi 2D, 3D, o user interface, ad esempio, saranno utilizzati per fare moltissime cose.

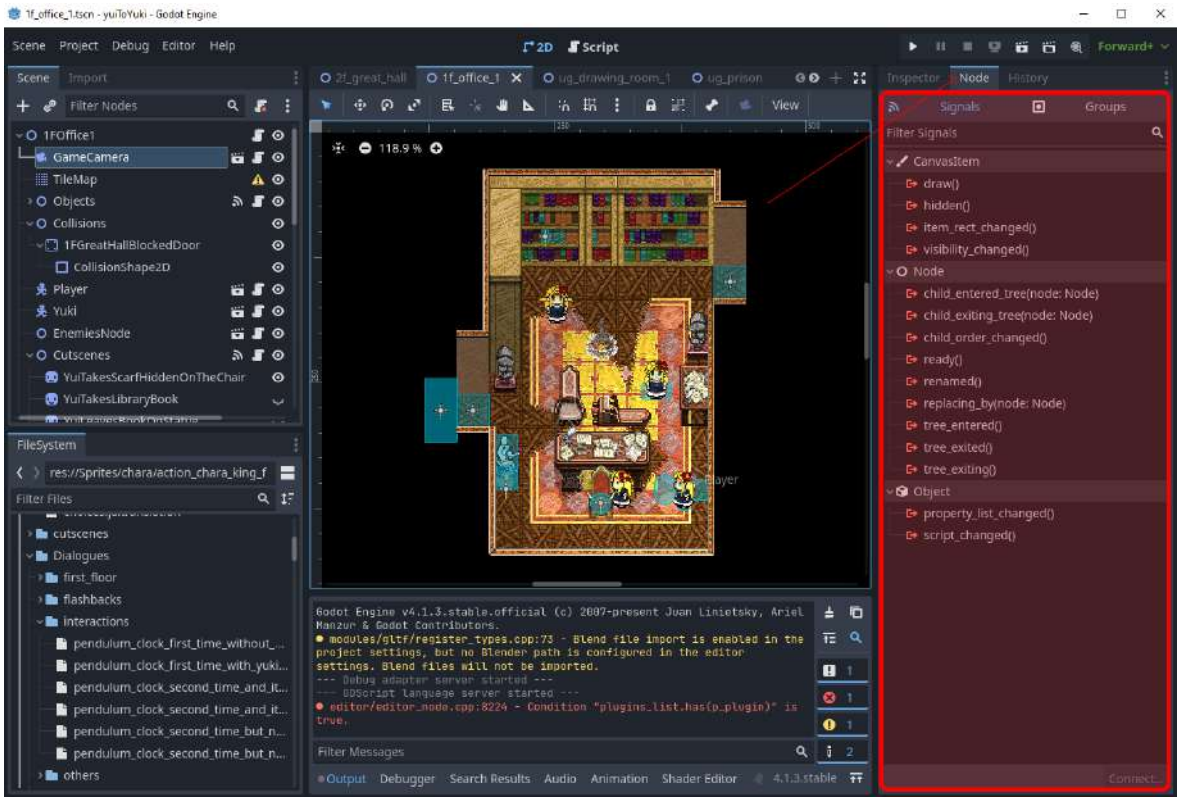


2.2.2.3 The scene tree

Tutte le scene di un gioco si uniscono per formare lo **scene tree**, il quale non è nient'altro che un vero e proprio "albero delle scene". Poiché però le scene sono alberi di nodi, questo vuol dire che anche lo scene tree è, a sua volta, un albero di nodi. Tuttavia, proprio perché le scene possono rappresentare personaggi, armi, porte, ecc. . . , risulta più facile pensare ad un gioco in termini di scene e non di nodi.

2.2.2.4 Signals

I nodi possono emettere dei segnali (signals) ogni volta che un qualche evento si verifica. Questa feature permette di far comunicare i nodi tra di loro senza collegarli direttamente via codice. Ciò conferisce molta flessibilità alla struttura di una scena.



Ad esempio, i bottoni emettono un segnale quando vengono premuti. Questo segnale può essere collegato a un codice che deve essere eseguito in reazione a questo evento, come far partire il gioco oppure aprire un menu.

Altri segnali built-in possono comunicare quando due oggetti sono entrati in collisione, quando un personaggio o mostro è entrato all'interno di una certa area e molto altro ancora. Si può persino creare dei segnali apposta per un gioco.

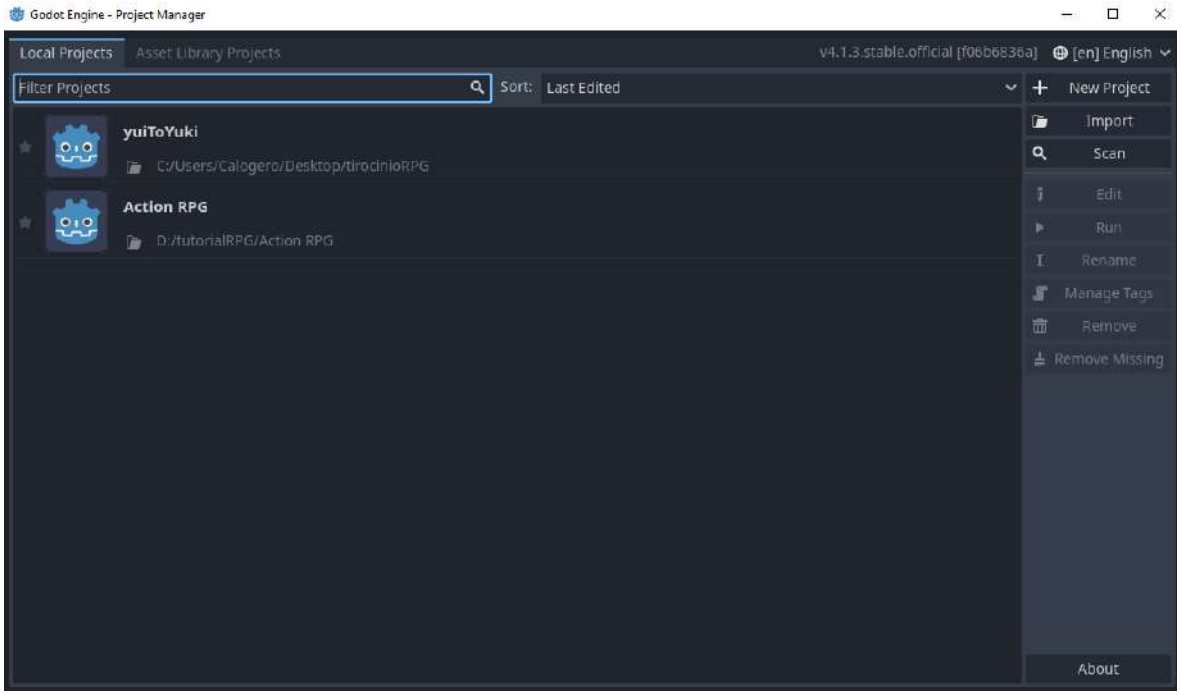
2.2.3 Primo sguardo all'interfaccia di Godot

In questa sezione daremo una breve overview dell'interfaccia di Godot. Illustreremo diverse schermate principali e dock per aiutare il lettore ad abituarsi all'interfaccia quanto più possibile.

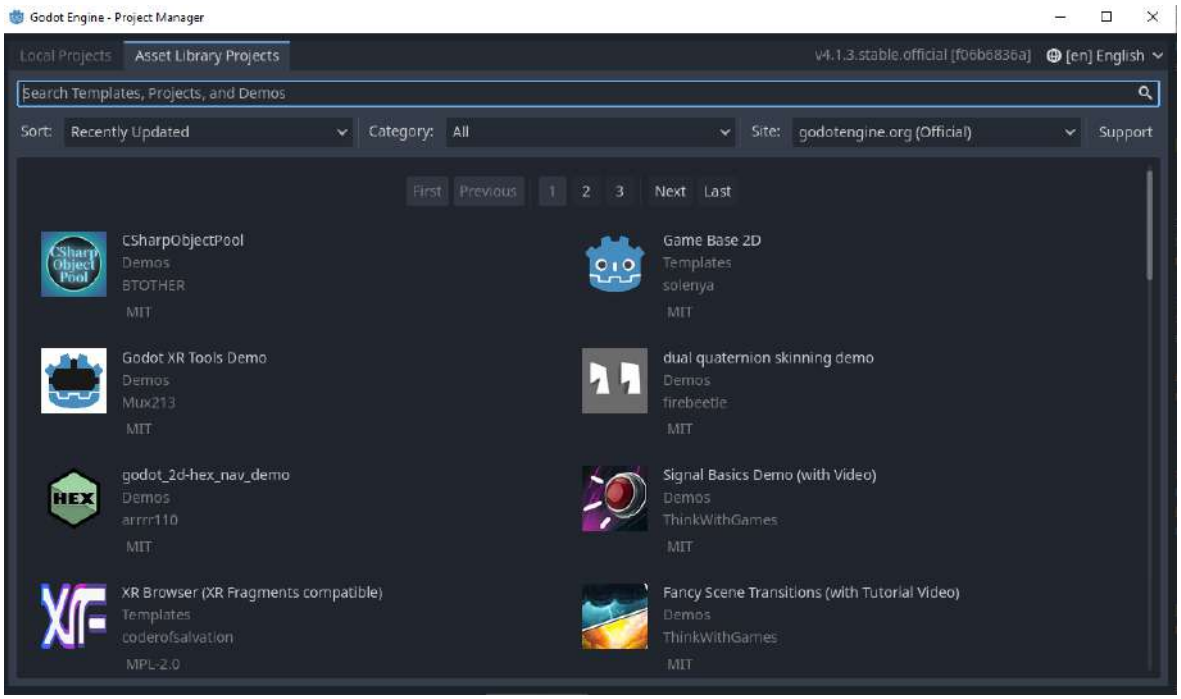
2.2.3.1 Il Project Manager

Quando si lancia Godot, la prima finistra in assoluto ad apparire è il **Project Manager**.

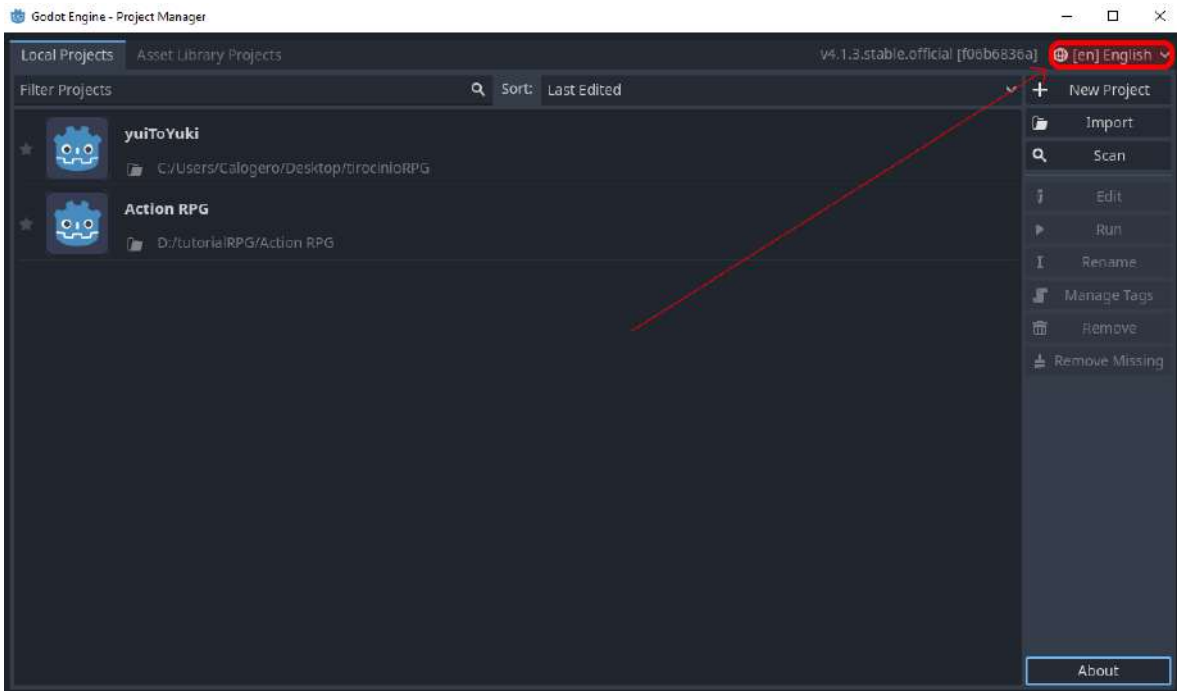
Nella tab di default **Projects** si possono gestire i progetti esistenti, importarli, crearne di nuovi, e molto altro ancora.



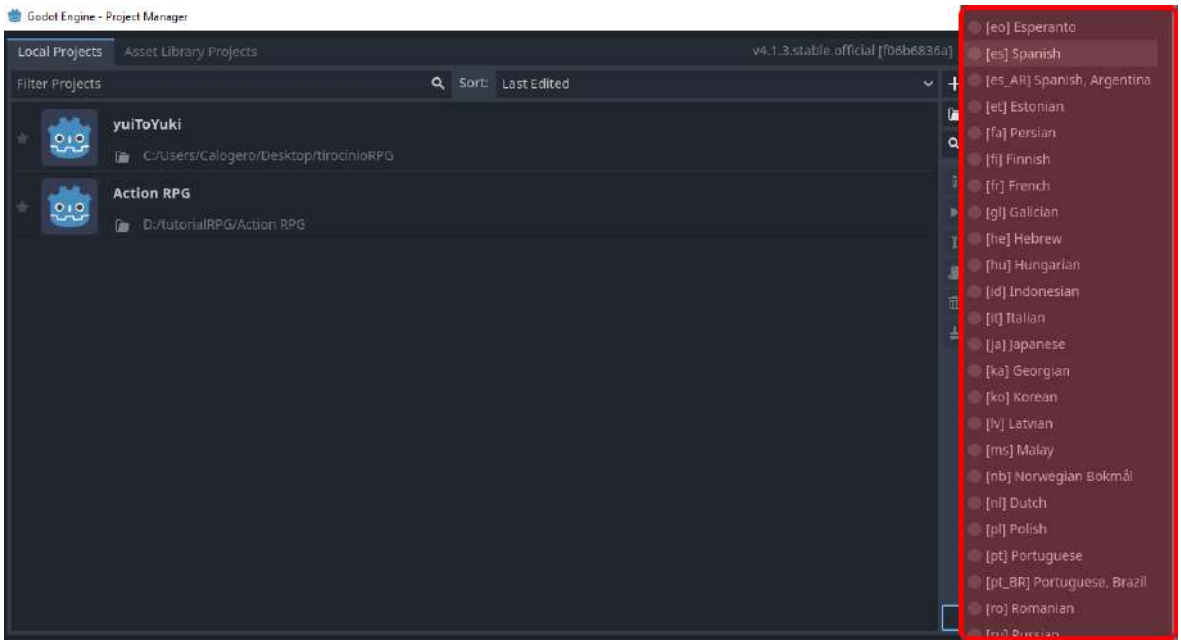
Nella parte superiore della finestra c'è un'altra tab chiamata **Asset Library** dalla quale è possibile cercare progetti demo che includono molti progetti sviluppati dalla community.



Utilizzando il menu drop-down sulla destra della versione dell'engine, nell'angolo in alto a destra della finestra, si può inoltre cambiare la lingua dell'editor cliccando sulla lingua corrente



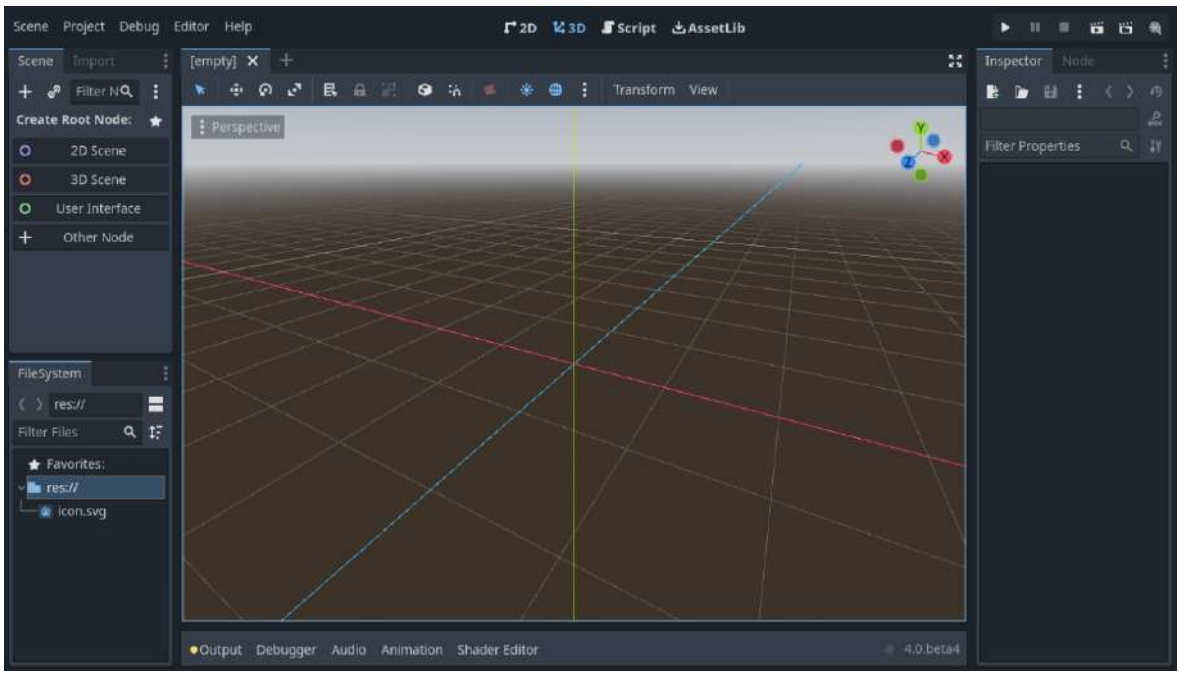
e selezionando una lingua tra quelle disponibili



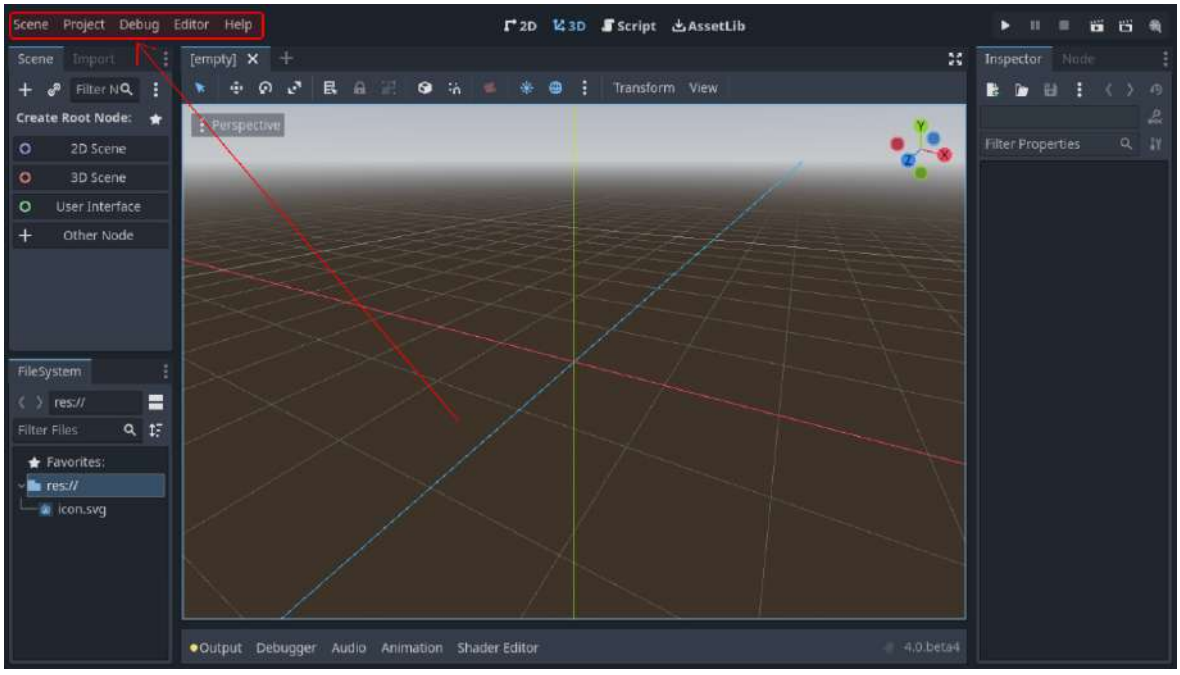
La lingua di default è l'inglese, English (EN).

2.2.3.2 Primo sguardo all'editor di Godot

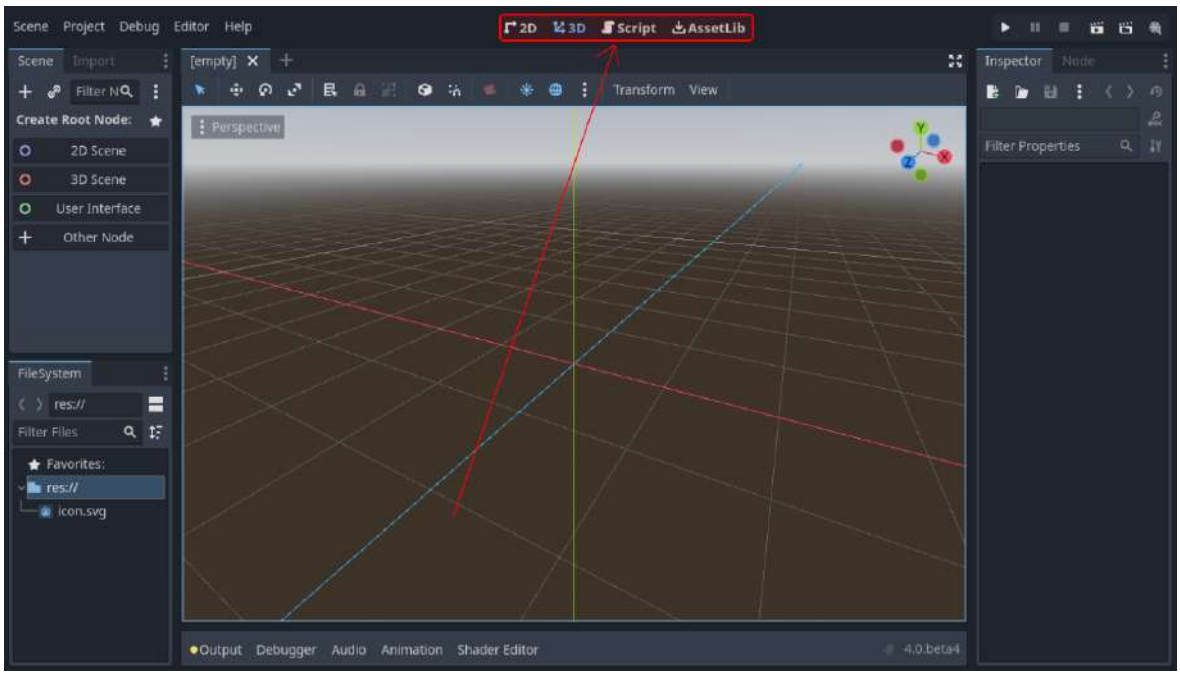
L'interfaccia di editor appare quando si apre un progetto nuovo o esistente. Analizziamo le sue aree principali.



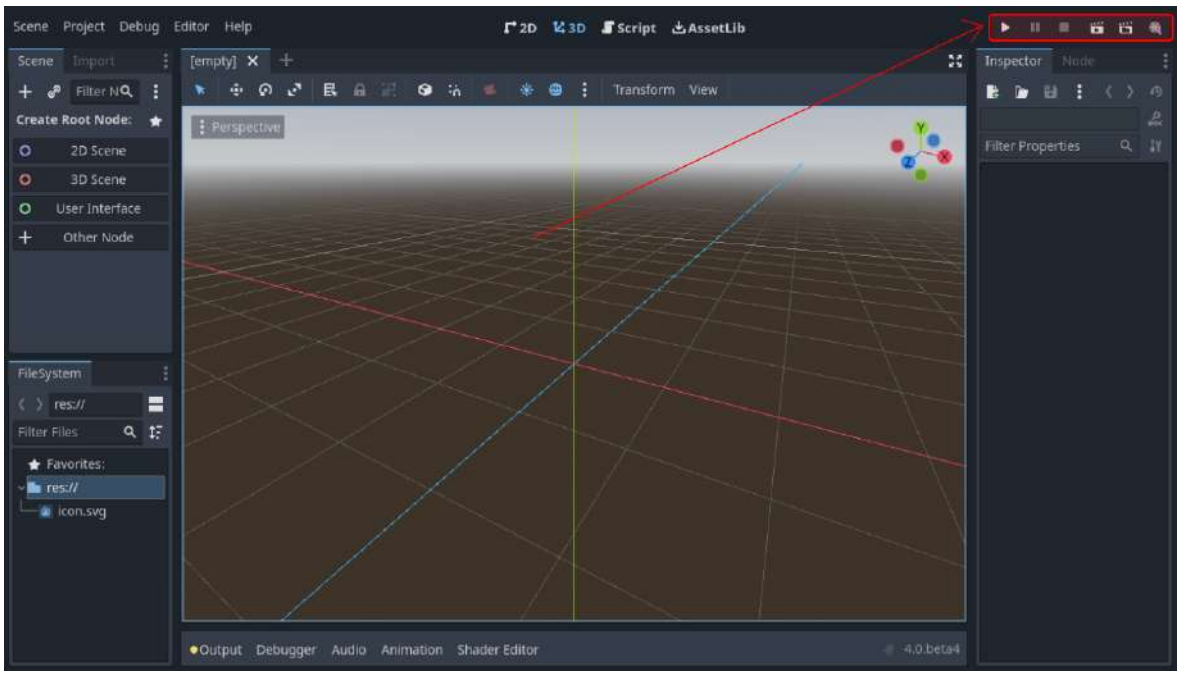
Di default, l'editor presenta **menus**



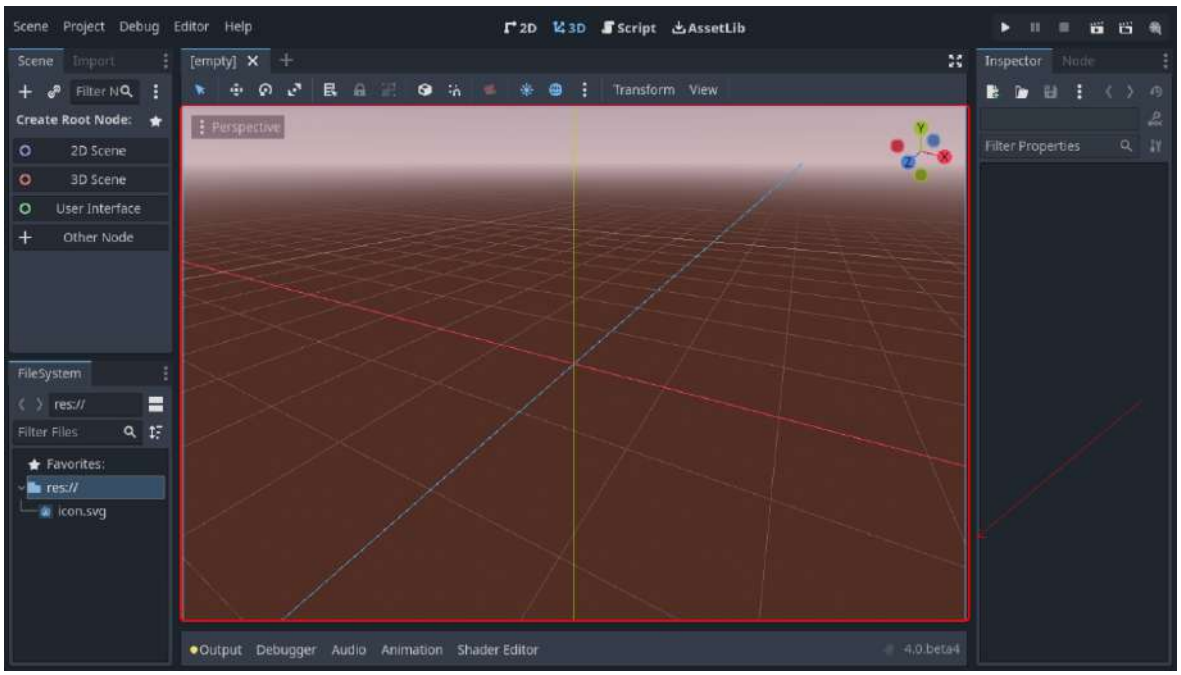
main screens (schermate principali)



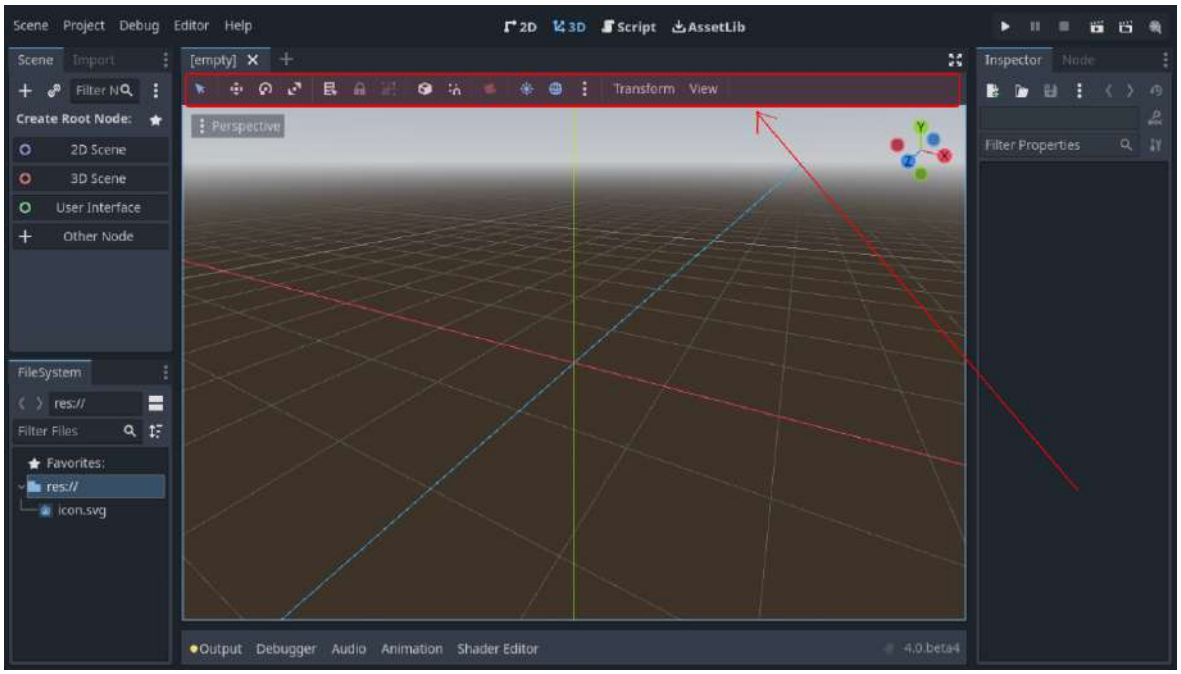
e pulsanti playtest lungo il bordo superiore della finestra.



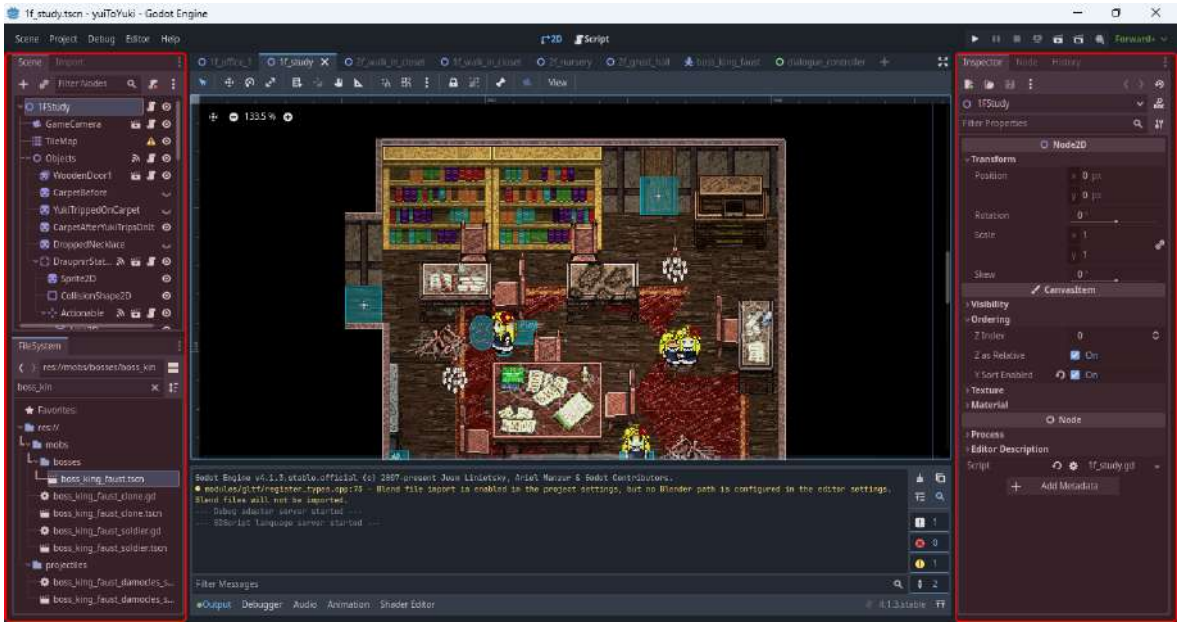
Al centro c'è la **viewport**



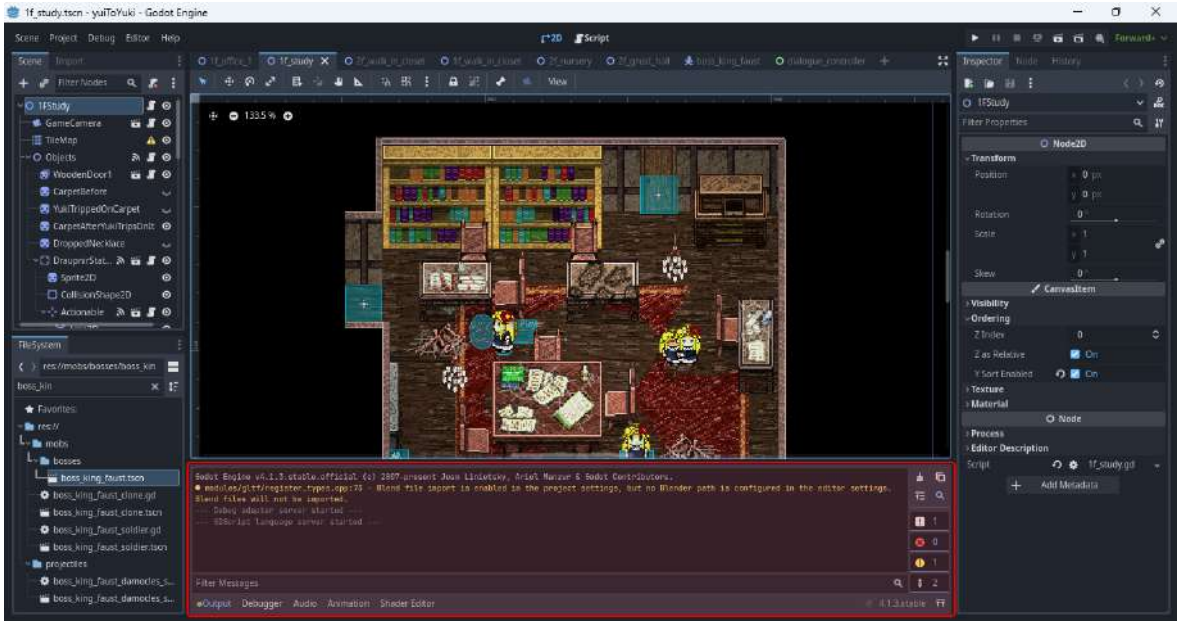
con la sua **toolbar** in alto, dove si trovano i tool per muoversi, scalare o bloccare i nodi della scena.



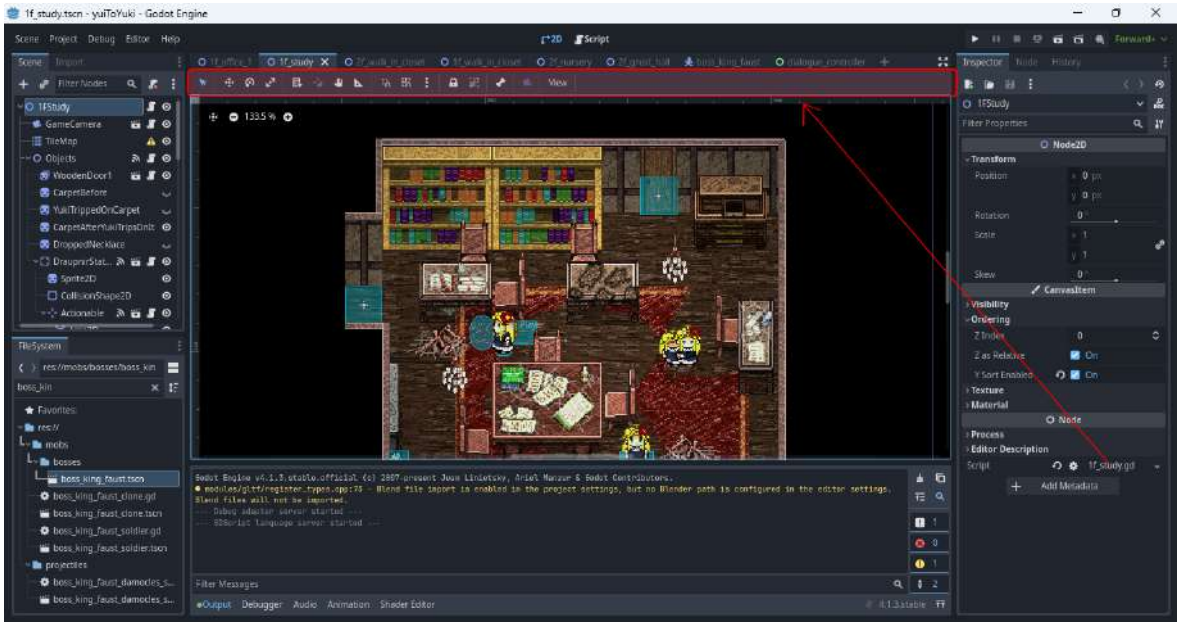
Ai lati della viewport si trovano i **dock**



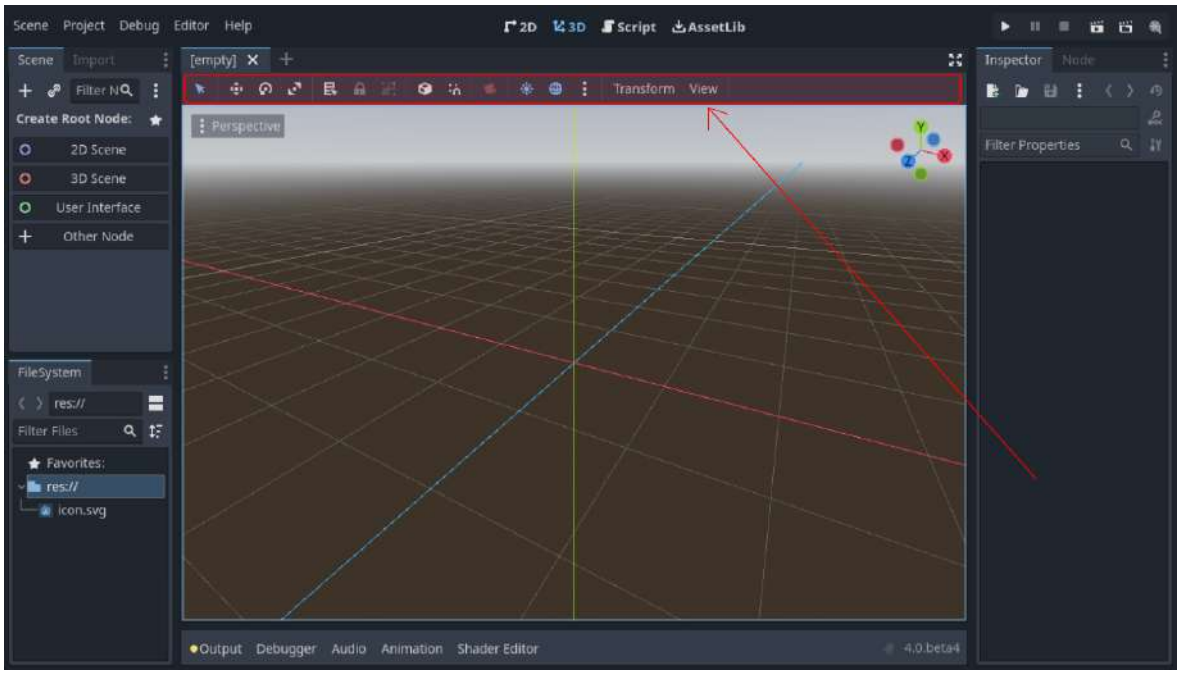
mentre in fondo alla finestra il **bottom panel** (pannello inferiore).



La **toolbar** cambia in base al contesto e al nodo selezionato. Sono di seguito riportate la toolbar 2D

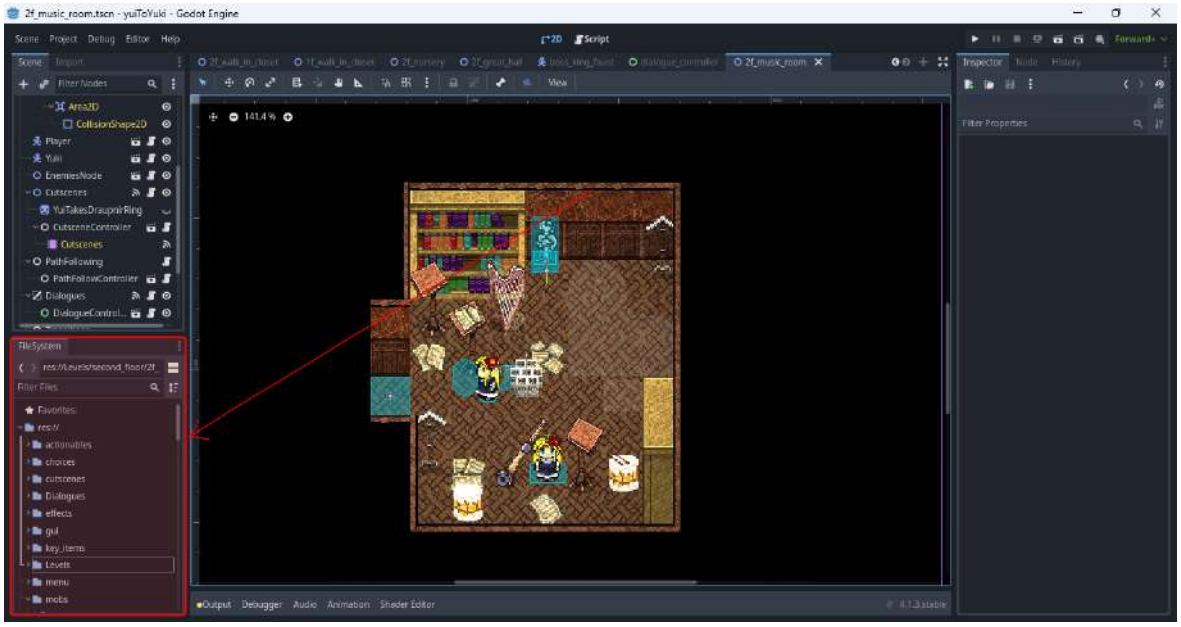


e la sua versione 3D.

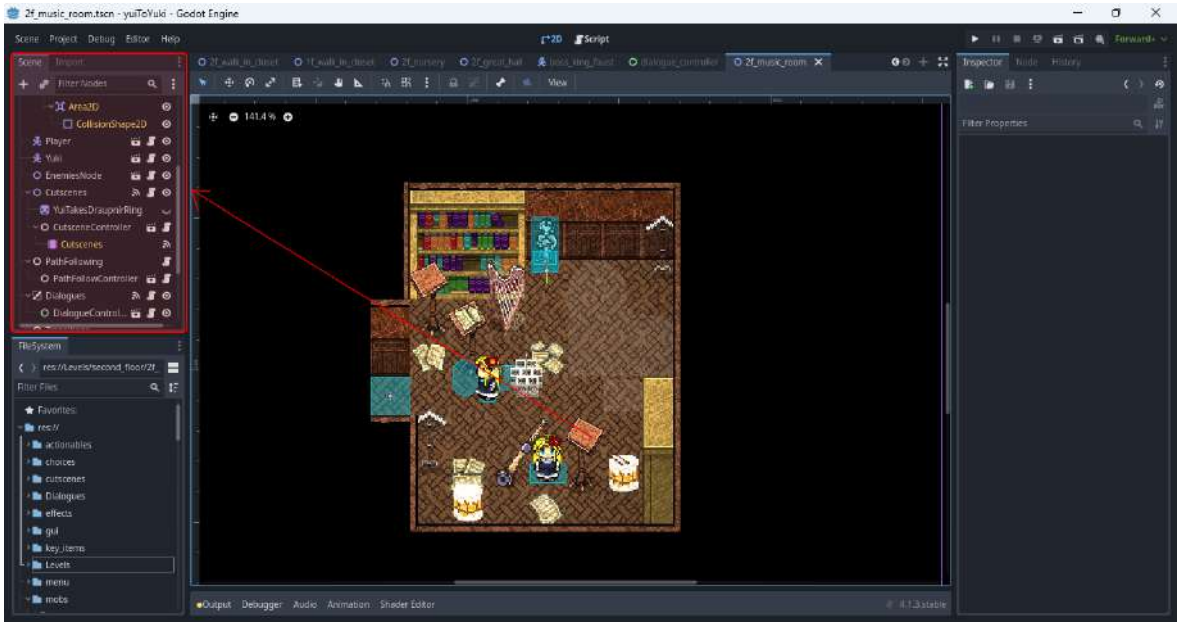


Diamo invece adesso uno sguardo ai dock.

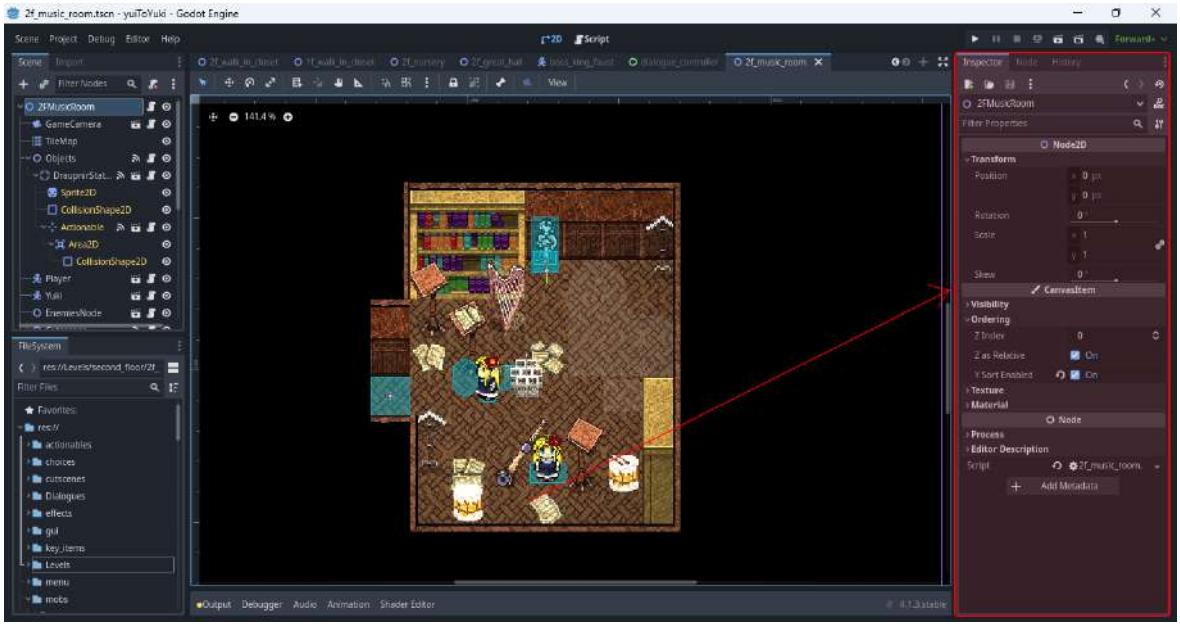
Il **FileSystem** dock elenca tutti i file del progetto, inclusi gli script, le immagini, l'audio e molto altro ancora .



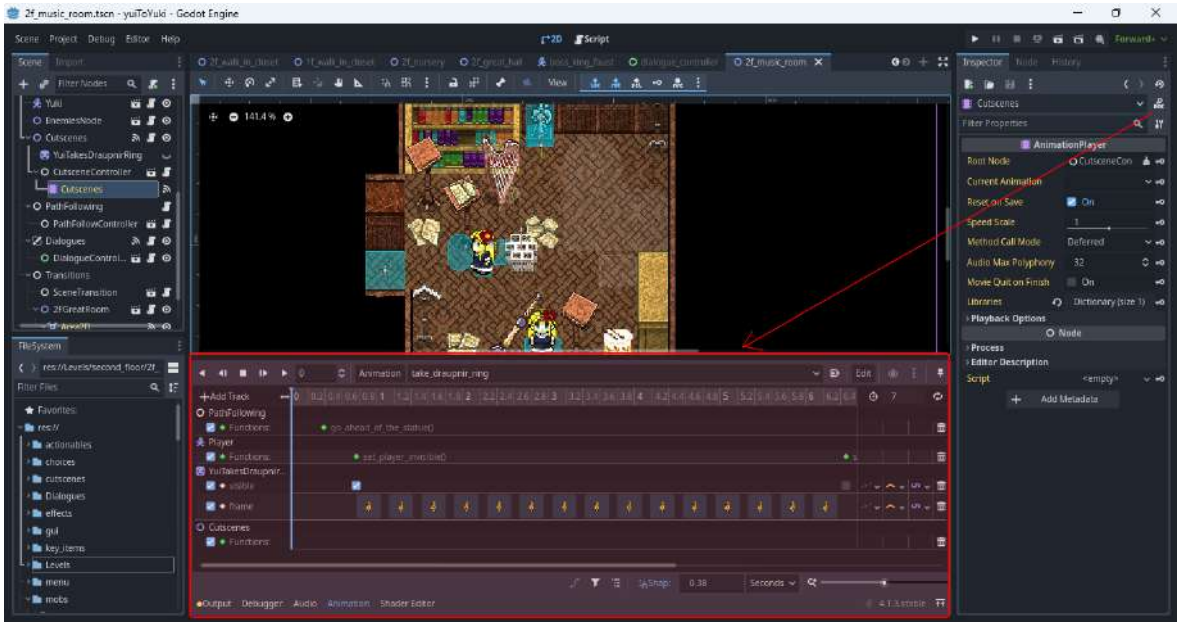
Lo **Scene** dock elenca tutti nodi attivi della scena



L'**Inspector** consente di modificare le proprietà di un nodo selezionato



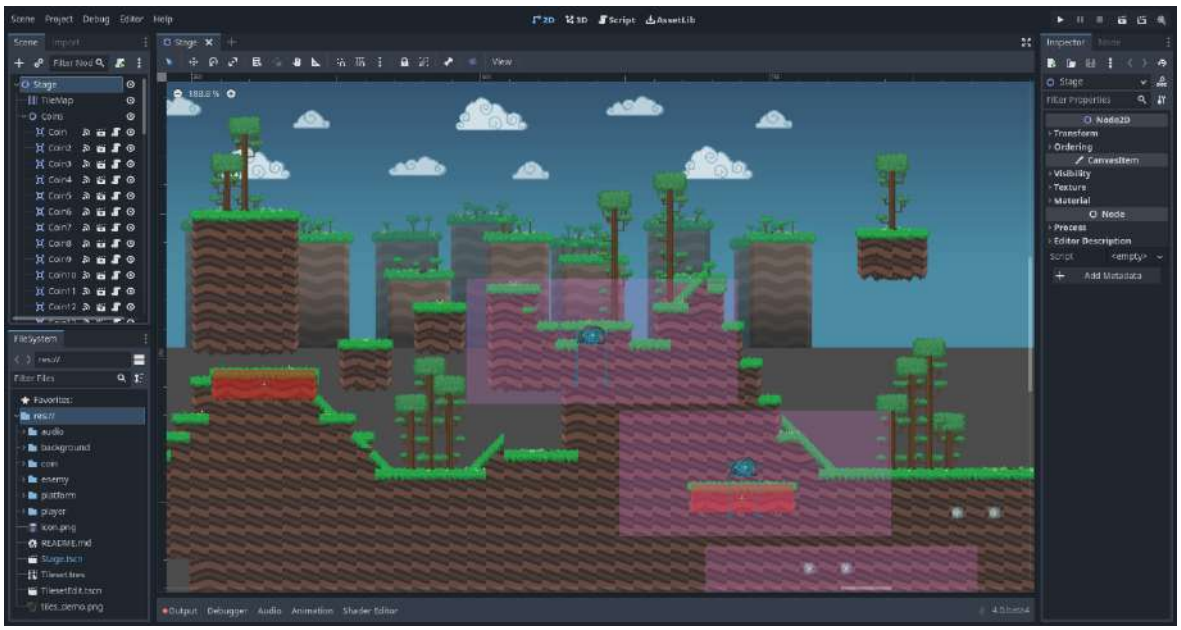
Il **bottom panel**, situato sotto la viewport, è l'host per la console di debug, l'animation editor, il mixer audio, e tanto altro. Le tab sono chiuse di default poiché ognuna di queste può occupare dello spazio sullo schermo prezioso. Quando si clicca su una tab, essa si espande in verticale.



2.2.3.3 Le quattro schermate principali

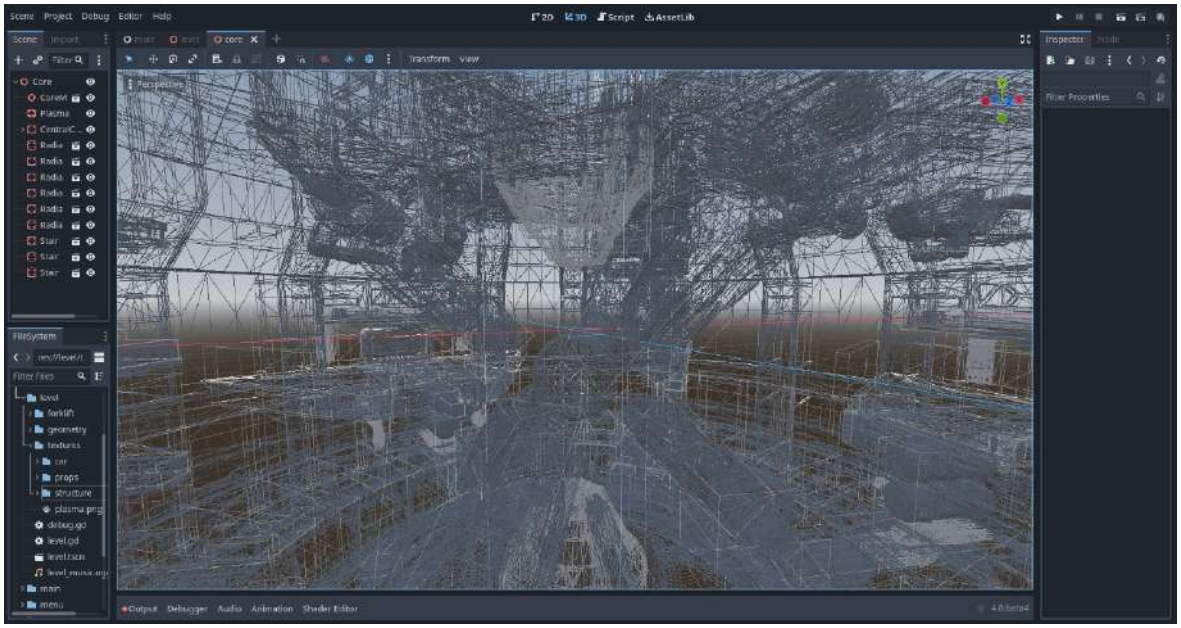
Nella parte superiore dell'editor sono presenti quattro pulsanti per le quattro schermate principali: 2D, 3D, Script e AssetLib.

La **schermata 2D** viene utilizzata per lo sviluppo di videogiochi 2D e la creazione delle interfacce.



6

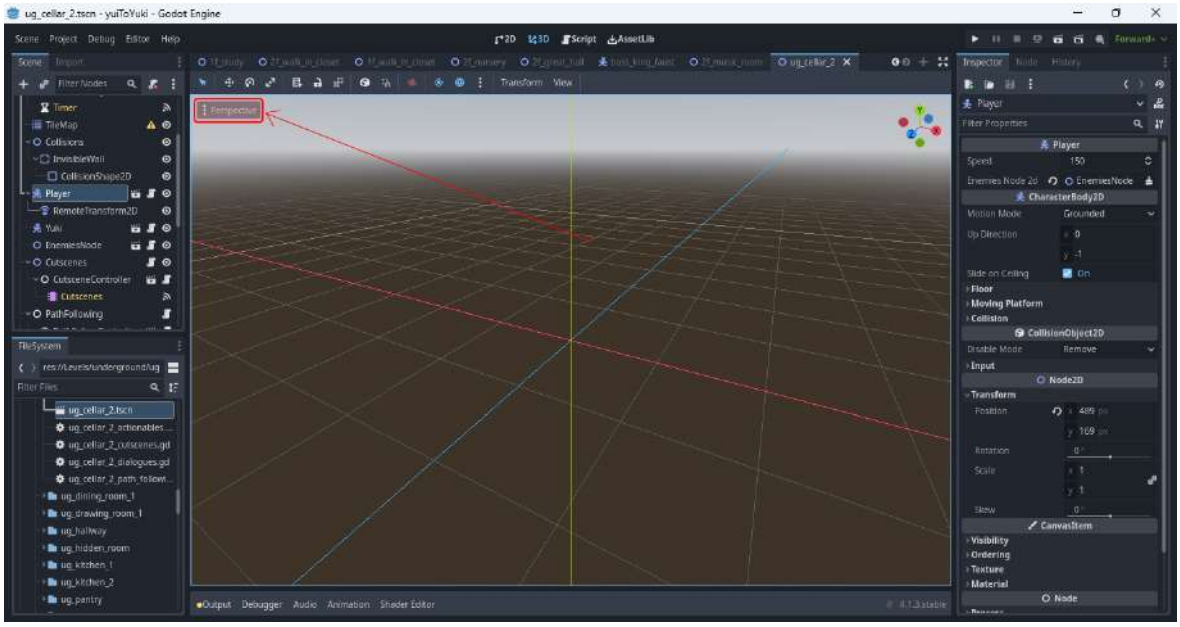
Nella **schermata 3D** si lavora con mesh e luci e si progettano i livelli dei videogiochi 3D.



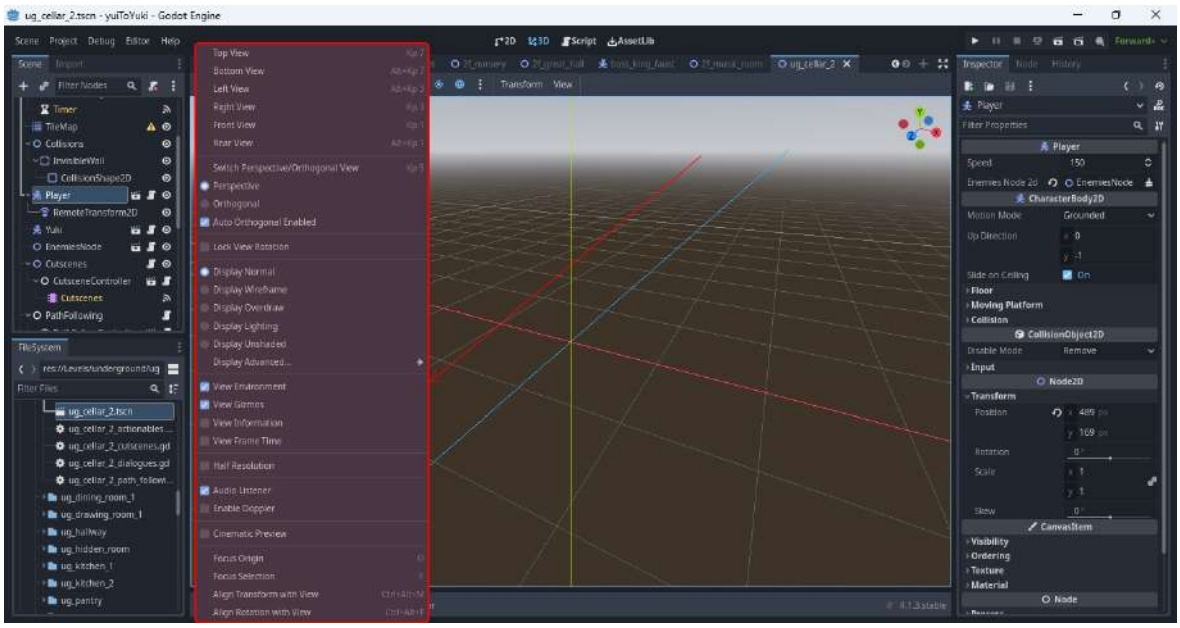
7

Cliccando il pulsante `Perspective` sotto la toolbar

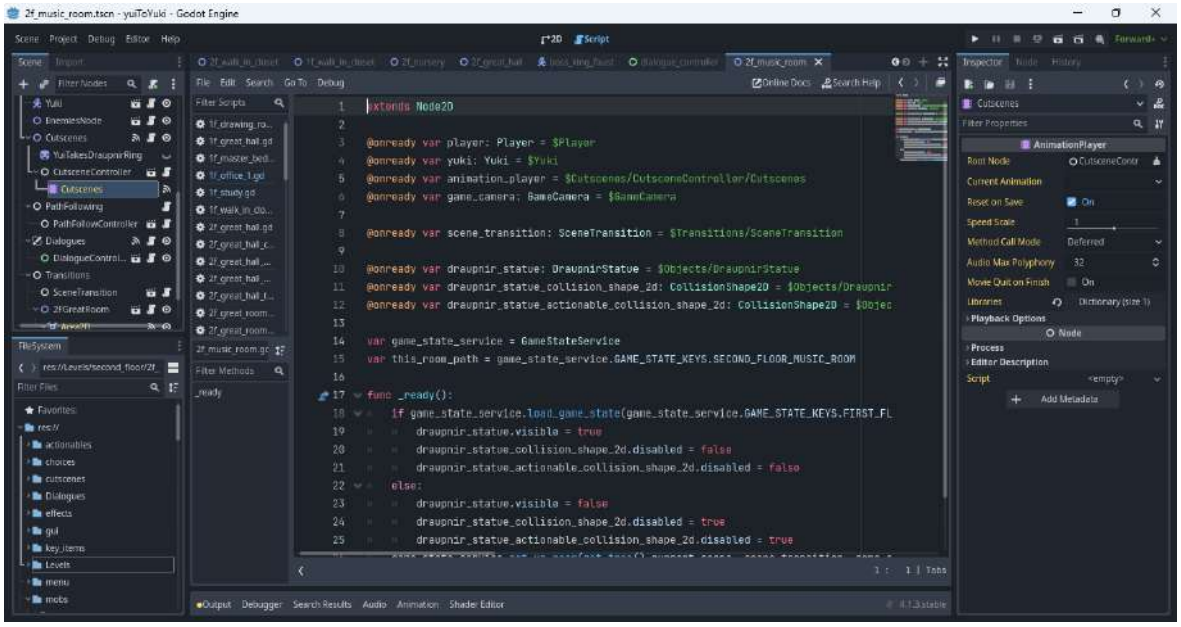
[6]Fonte: Godot Documentation
[7]Fonte: Godot Documentation



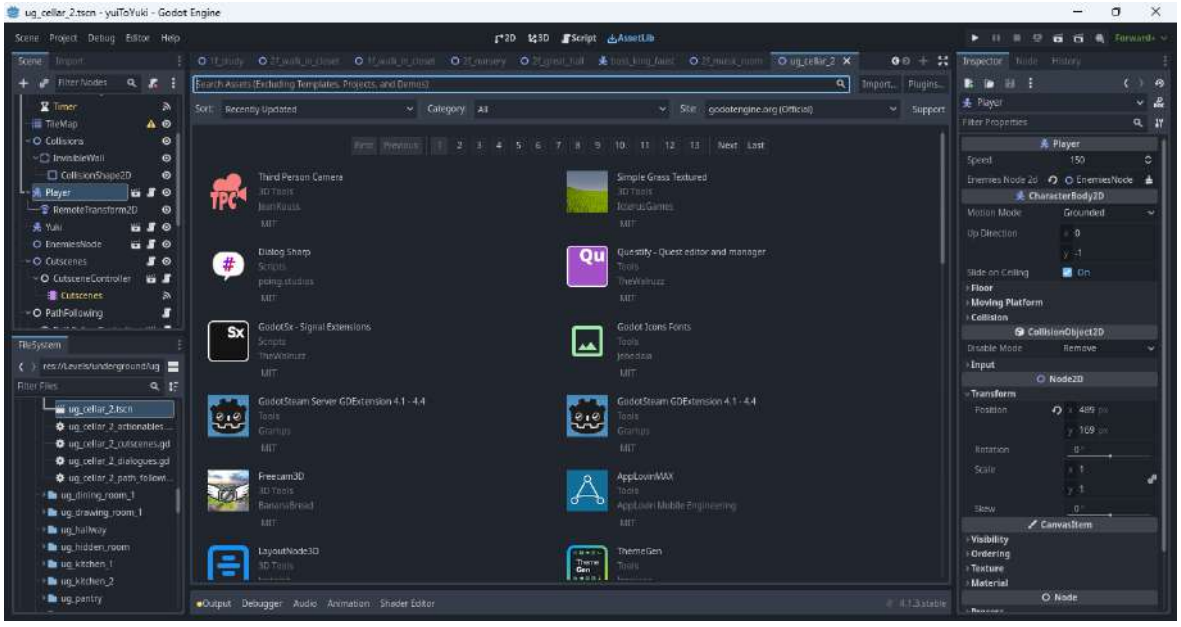
si accede a una lista di opzioni relative alla visualizzazione 3D.



La **schermata degli script** è un editor di codice completo con un debugger, un ricco autocompletamento e reference di codice built-in.



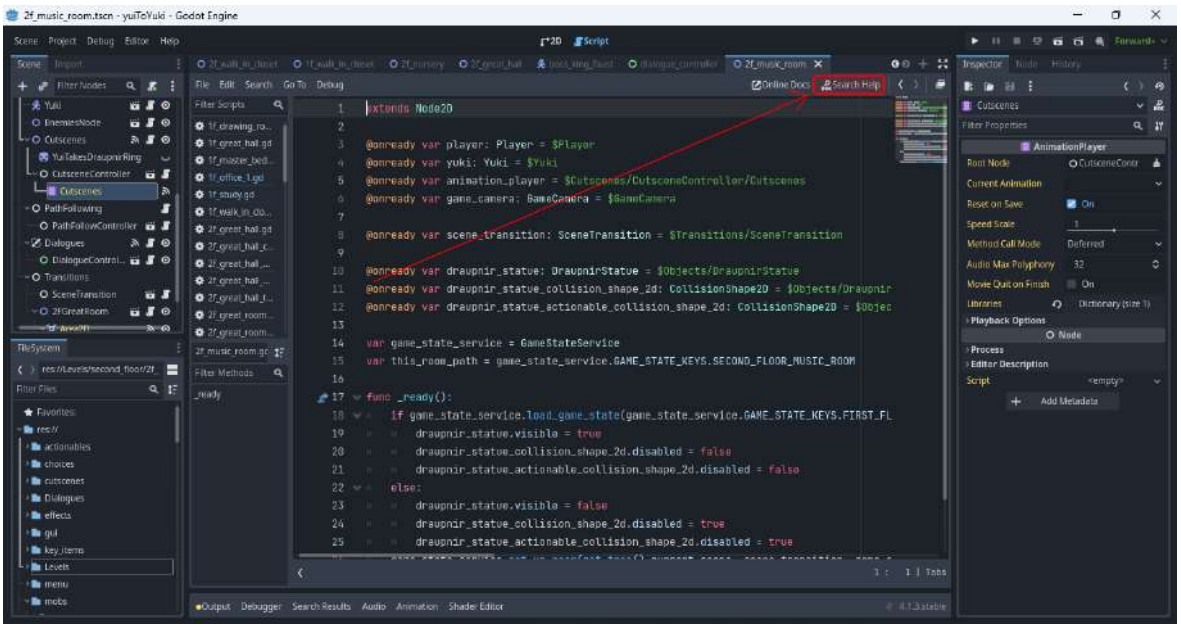
Infine, la **schermata AssetLib** è un libreria gratuita e open source di add-on, script e asset da utilizzare nei propri progetti.



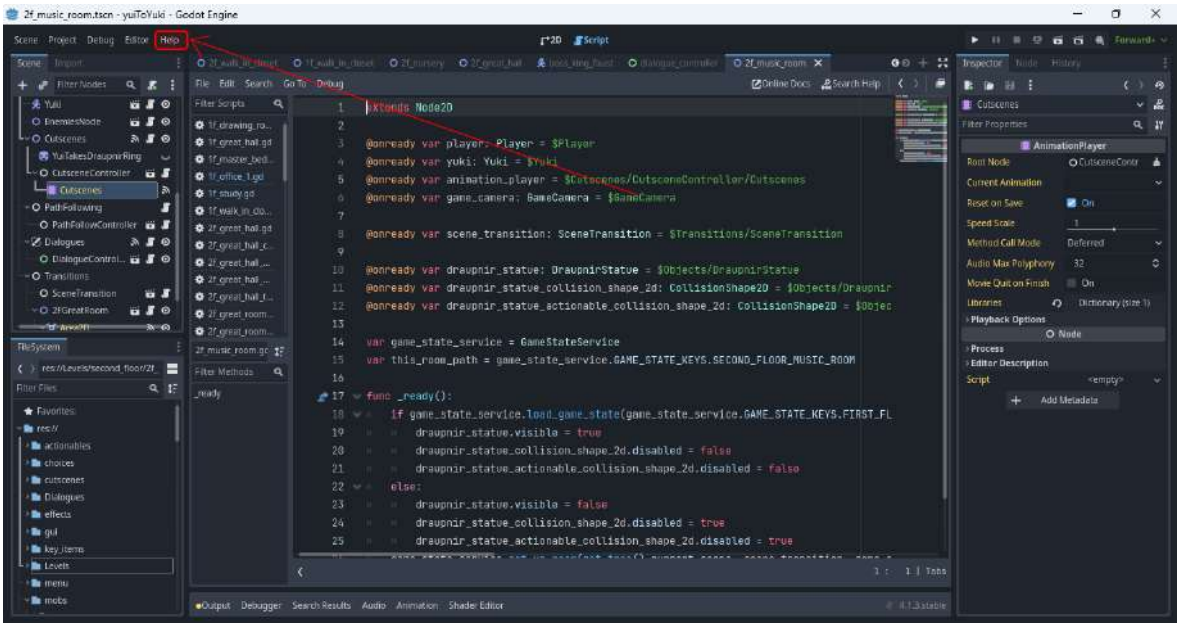
2.2.3.4 Class reference integrato

Godot presenta un class reference built-in mediante cui si possono cercare informazioni su una classe, un metodo, una costante o un segnale in uno qualsiasi dei seguenti modi:

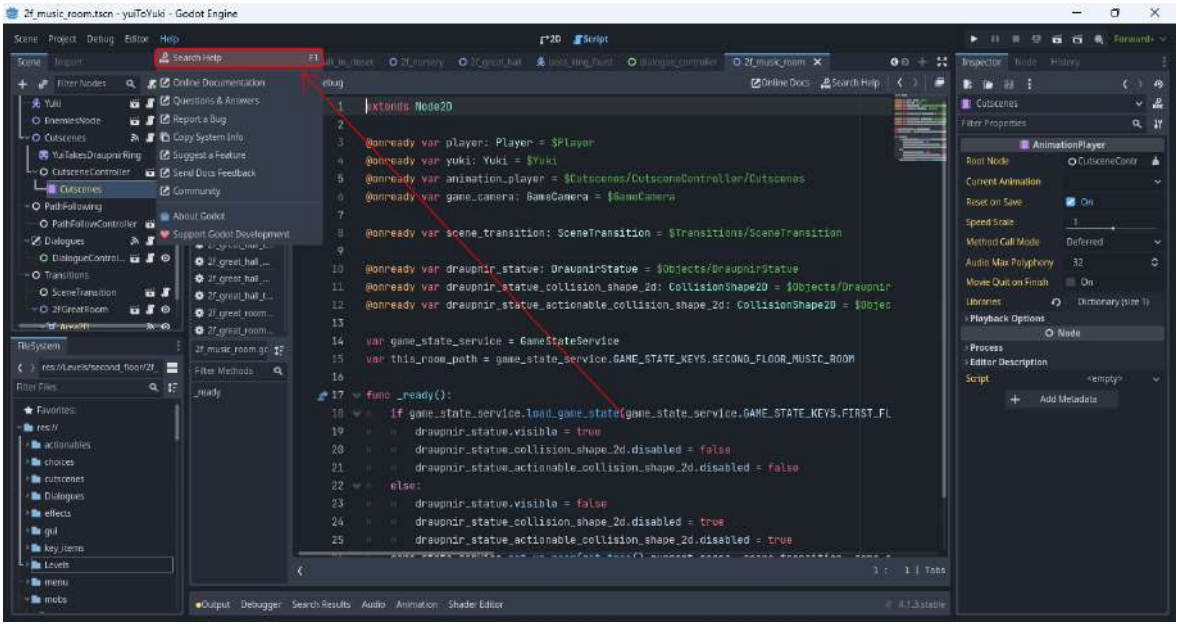
- Premendo `F1` (oppure `Alt + Space` su macOS, oppure `fn + F1` per laptop con una key `fn`) ovunque nell'editor
- Cliccando sul pulsante "Search Help" in alto a destra nella schermata principale degli script



- Cliccando sul pulsante "Help" del menu

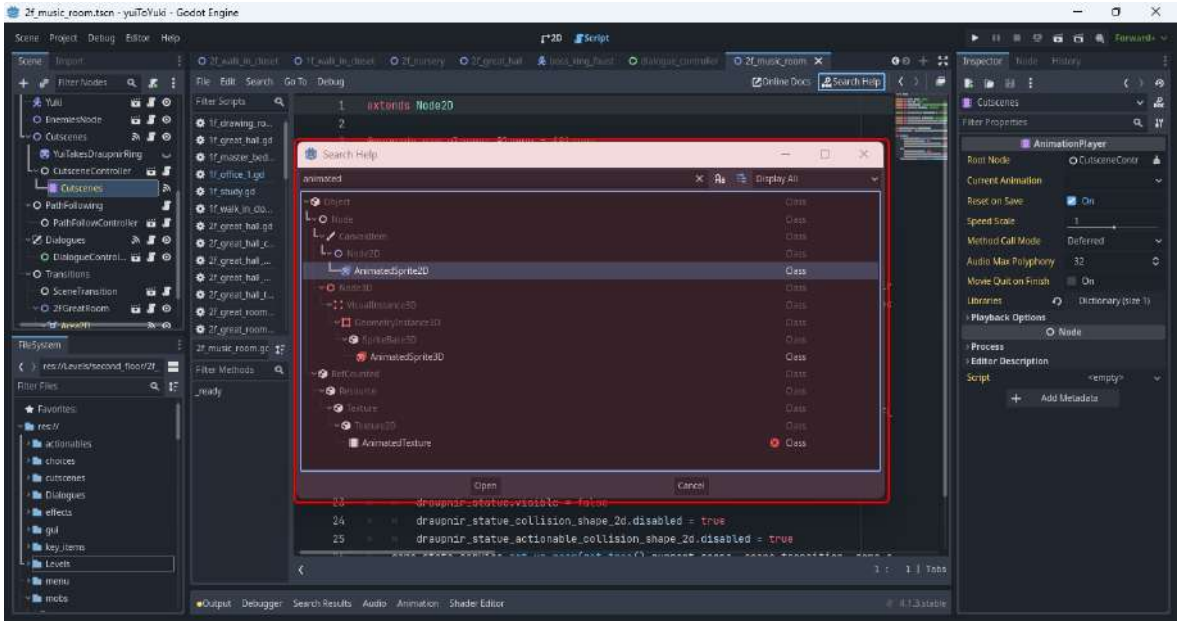


e poi su "Search Help"

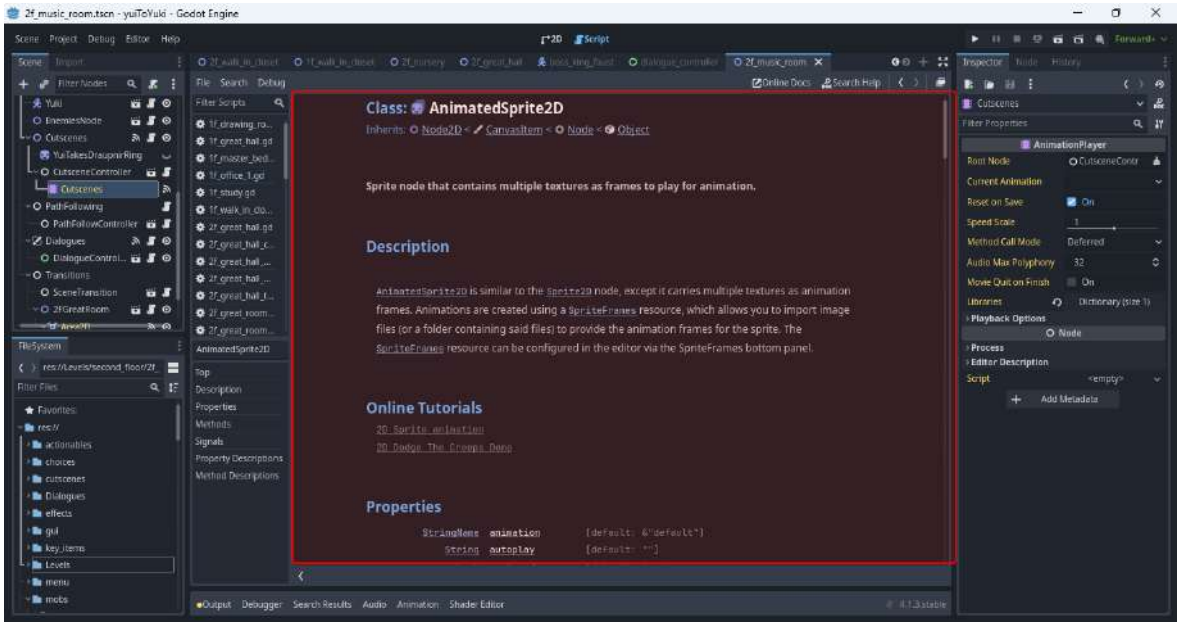


- Cliccando sul nome di una classe, funzione o variabile built-in nello script editor mentre si sta premendo `Ctrl`

Quando si esegue una di queste azioni, appare una finestra a schermo nella quale si può cercare qualsiasi oggetto digitando oppure dare un'occhiata a tutti gli oggetti disponibili.



Doppio-click su un item per aprire la corrispondente pagina nella schermata principale degli script

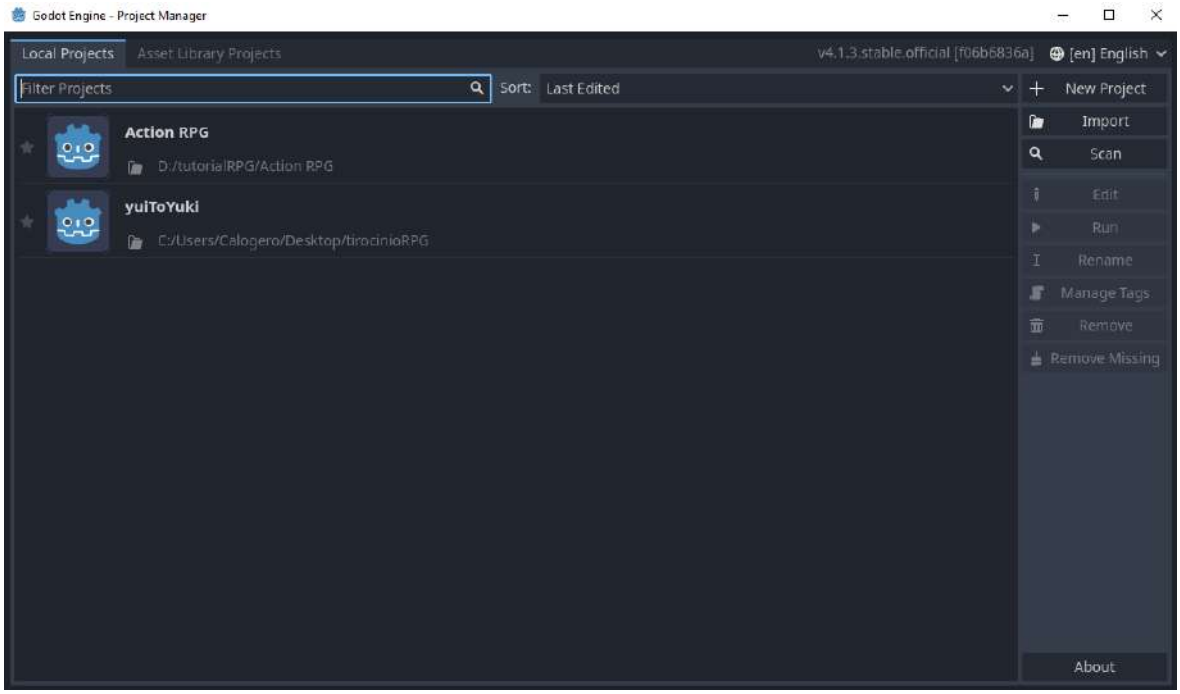


2.2.4 L'interfaccia dell'editor di Godot

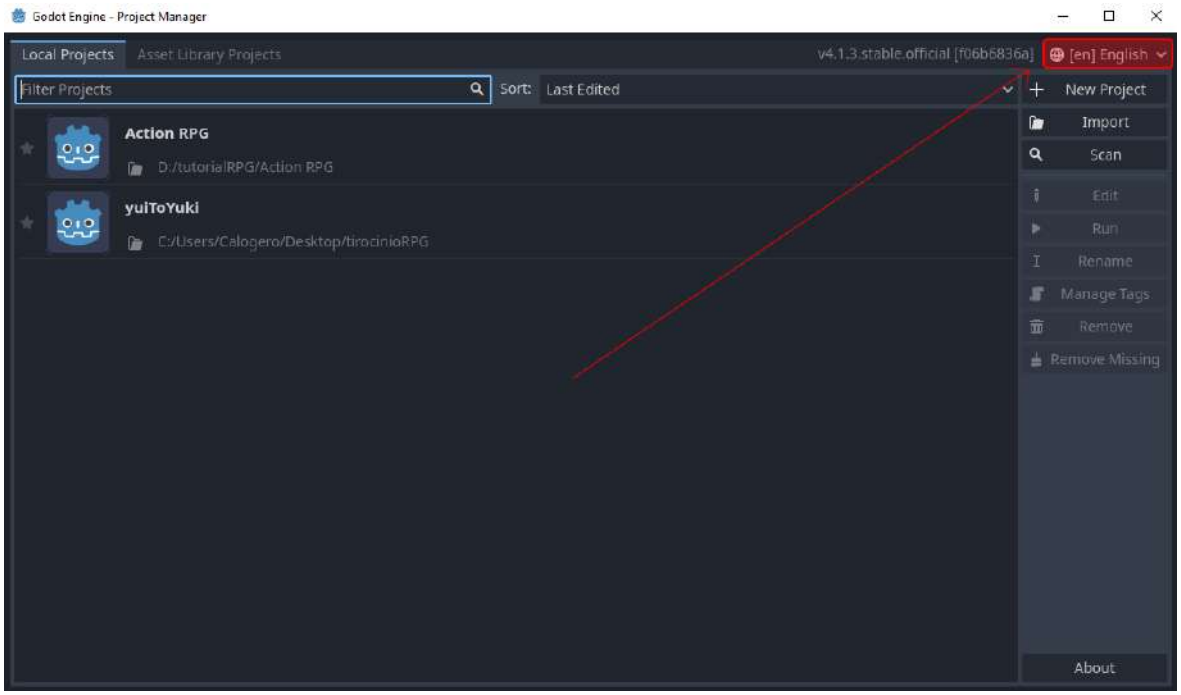
Nelle pagine seguenti, spiegheremo come utilizzare le varie finestre, workspace e dock che compongono l'editor di Godot. Dove appropriato, illustreremo anche specifiche interfacce di altre sezioni dell'editor, come, ad esempio, l'animation editor.

2.2.4.1 Utilizzare il Project Manager

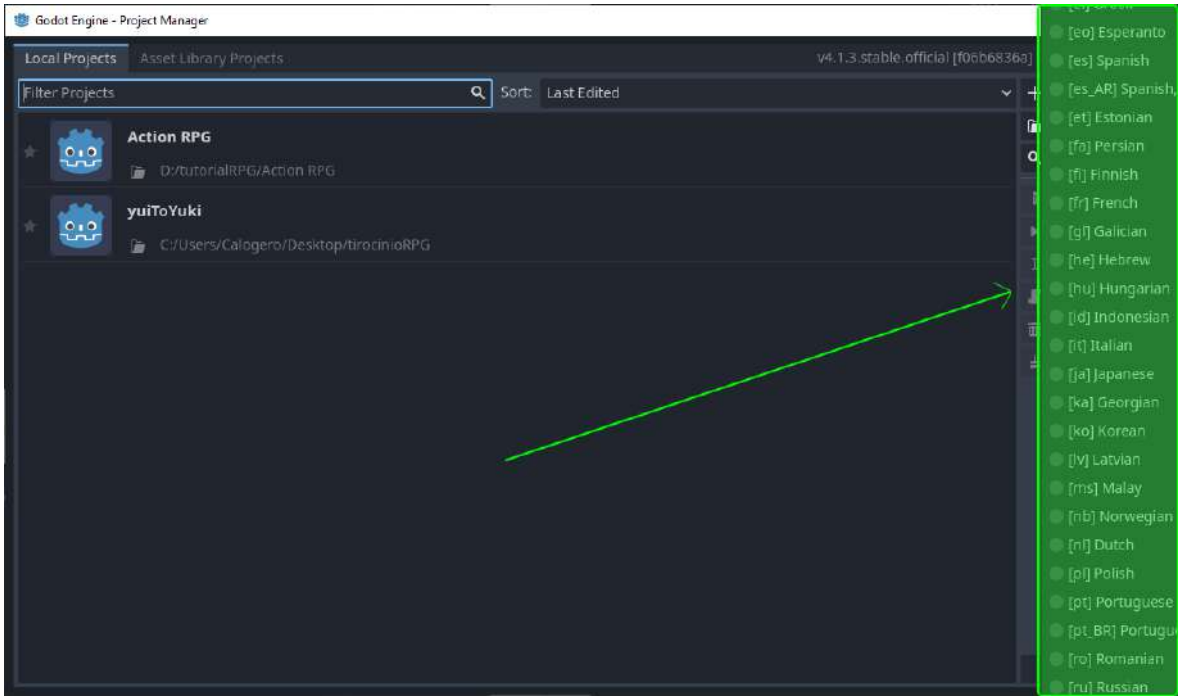
Quando si lancia Godot, la prima finestra ad apparire è il **Project Manager**, il quale permette di creare, rimuovere, importare o eseguire progetti di giochi.



Nell'angolo in alto a destra della finestra, cliccando il pulsante con la lingua corrente

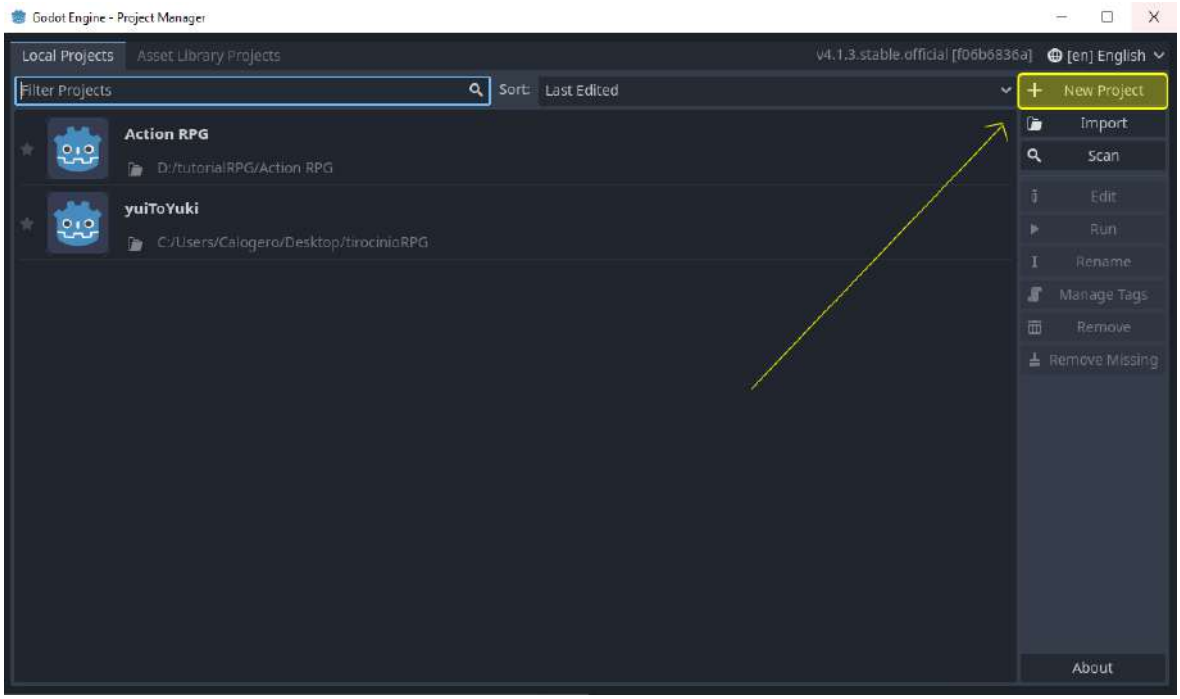


un menu drop-down ci permette di cambiare la lingua dell'editor.

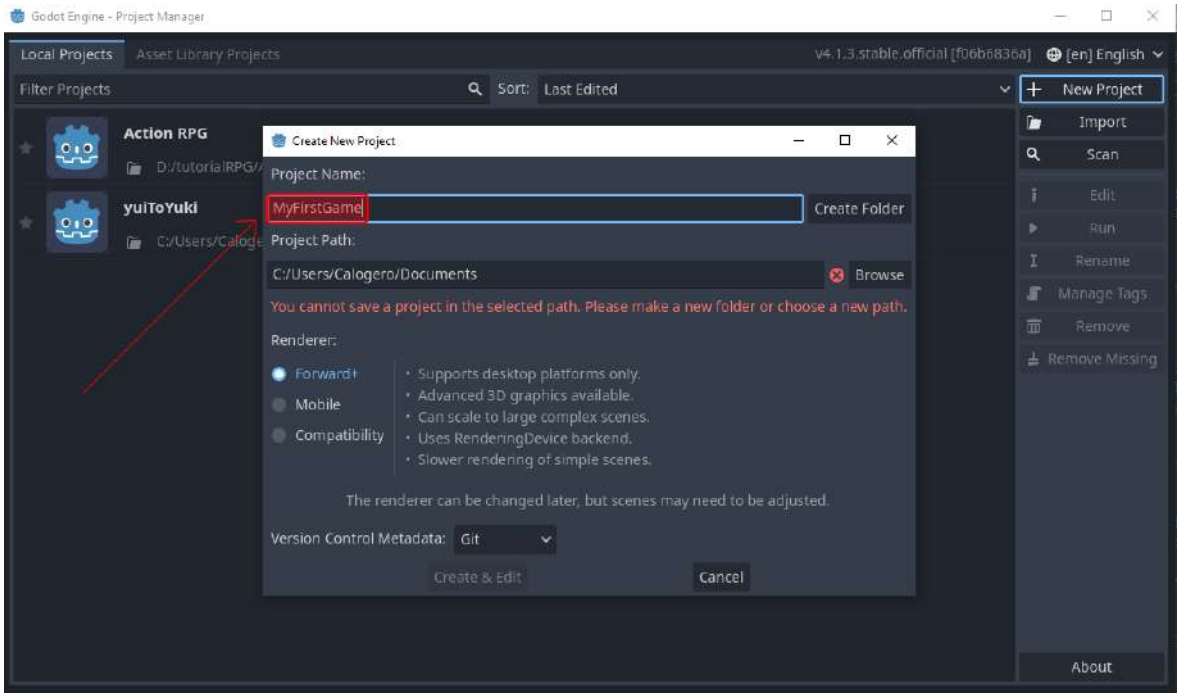


2.2.4.1.1 Creare e importare progetti

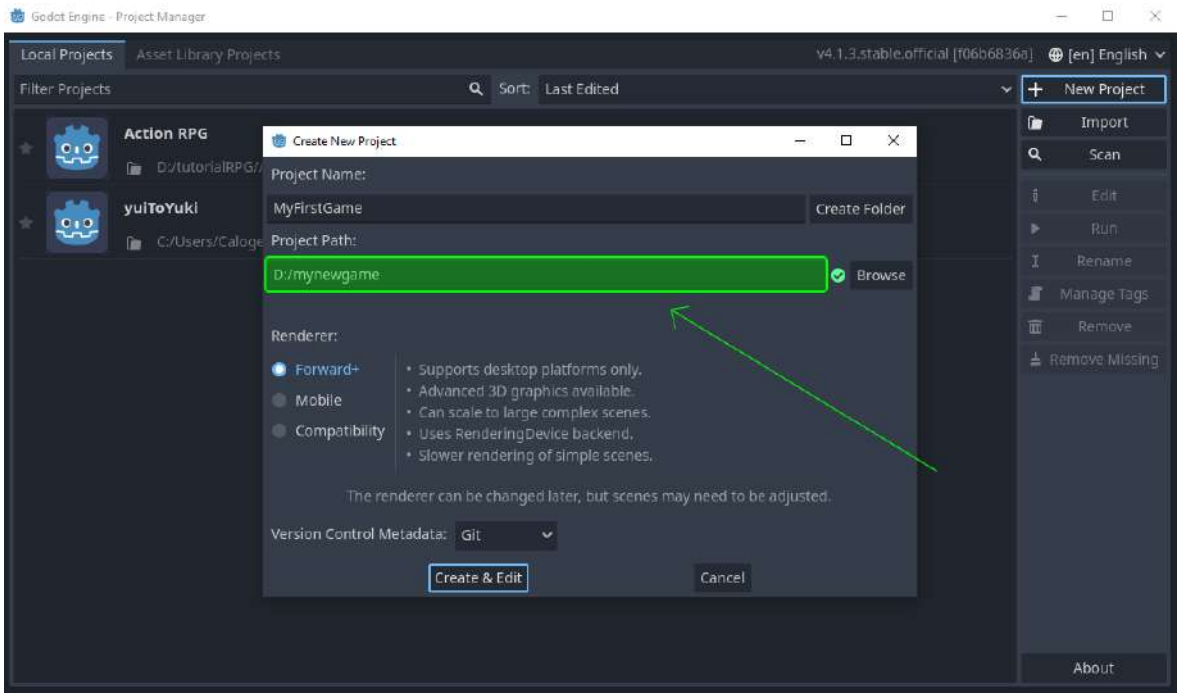
Per creare un nuovo progetto clicchiamo sul pulsante `New Project` nel menu a destra della finestra



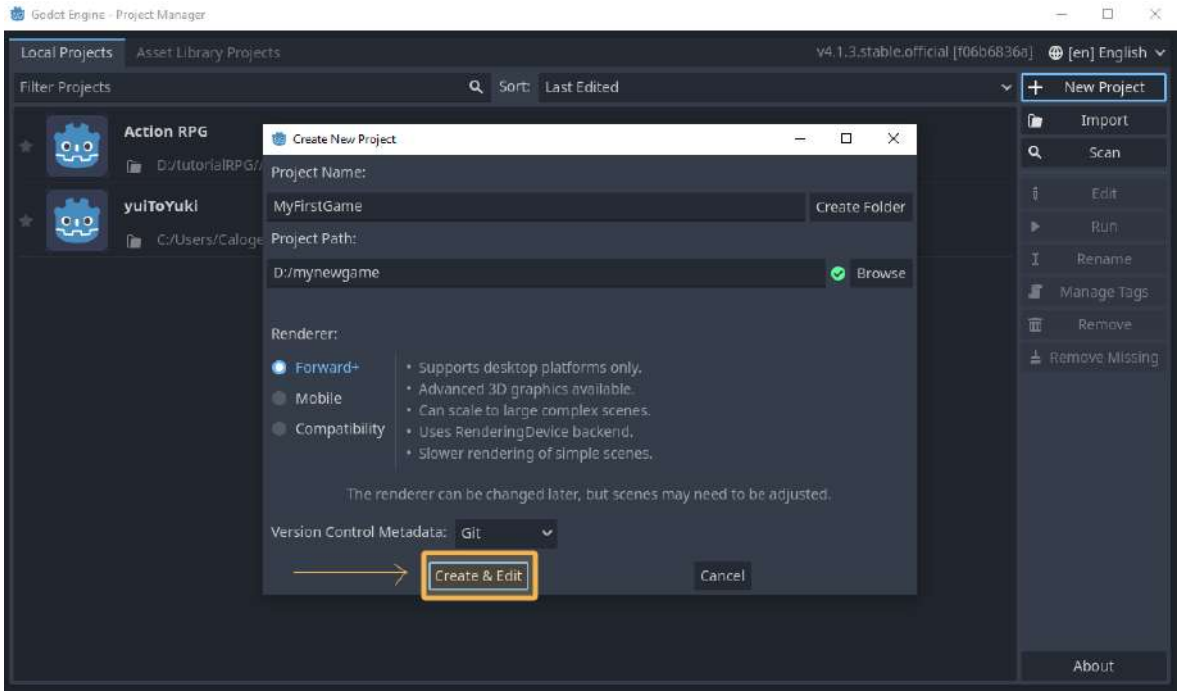
diamo un nome al progetto



scegliamo una cartella vuota sul computer in cui salvare i file

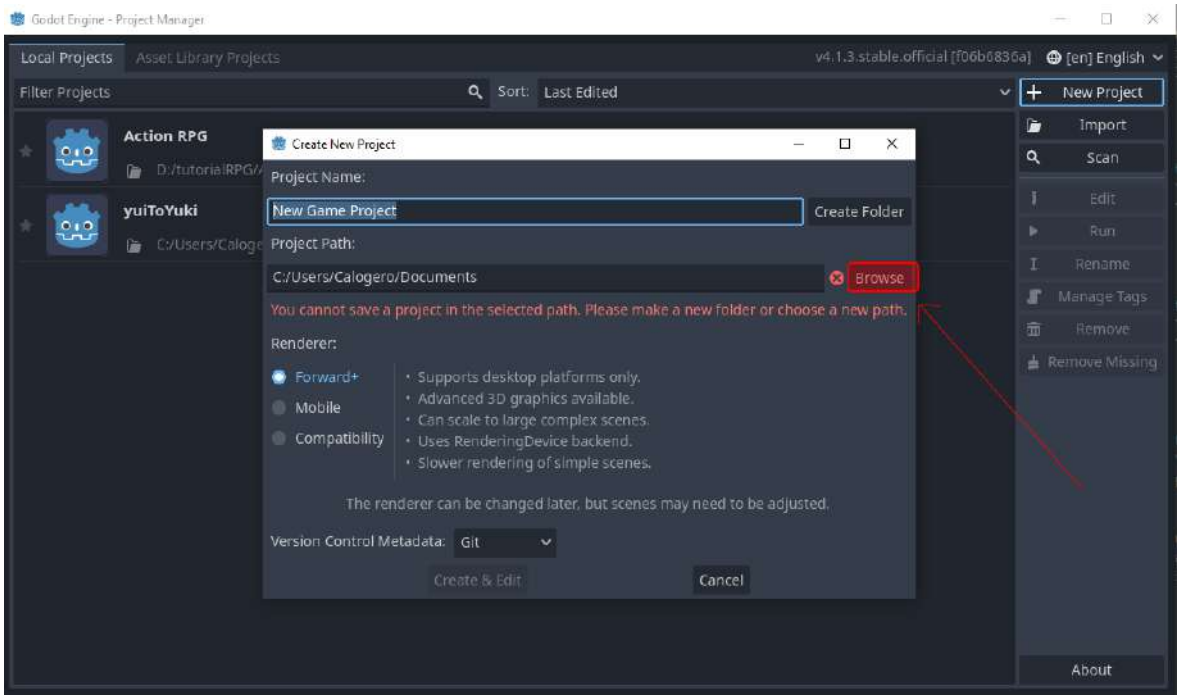


clicchiamo il pulsante `Create & Edit` per creare e aprire il progetto nell'editor

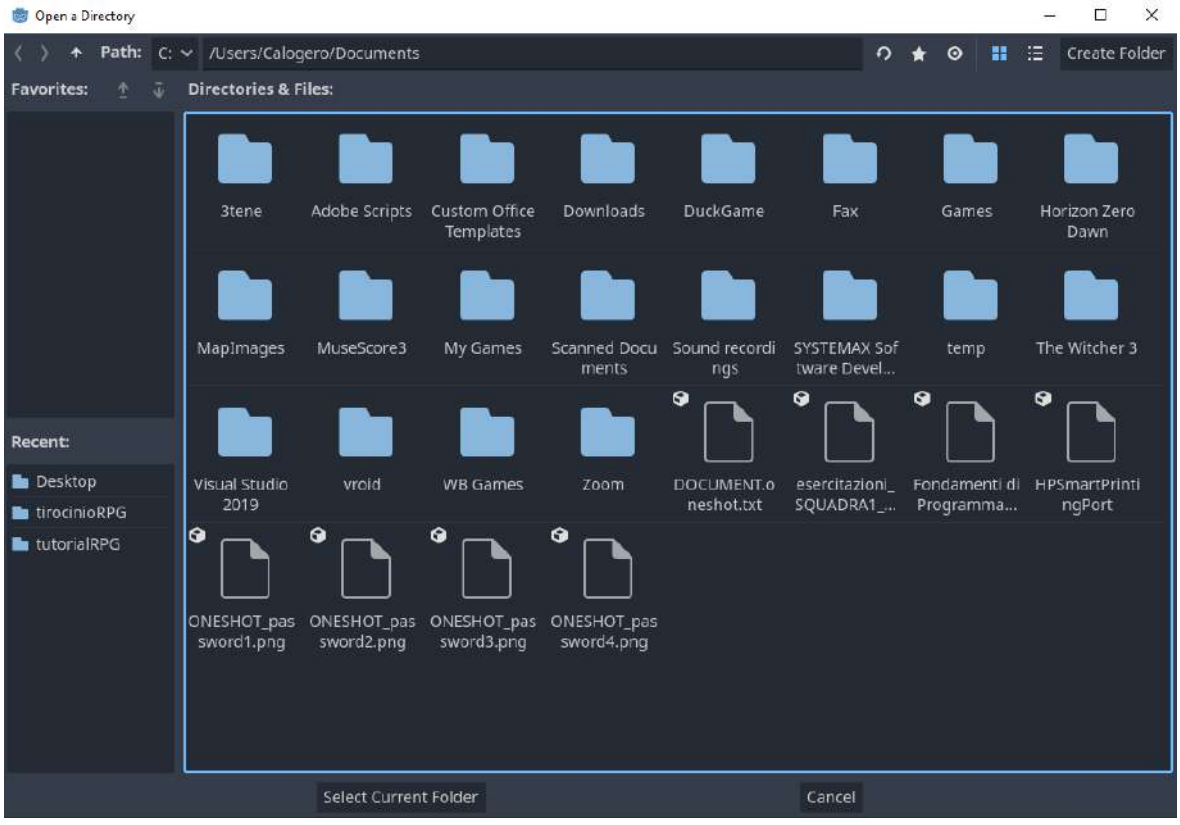


2.2.4.1.1.1 Utilizzare il file browser

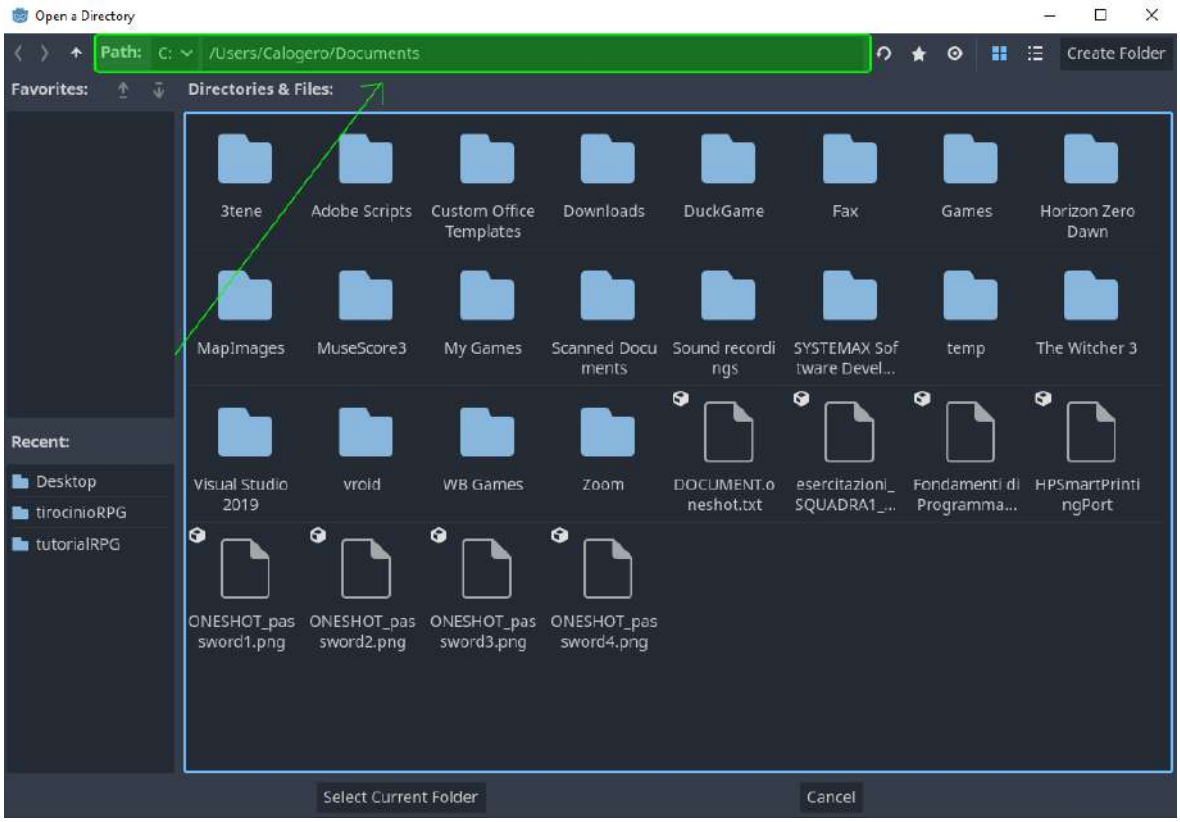
Fare click sul pulsante `Browse`



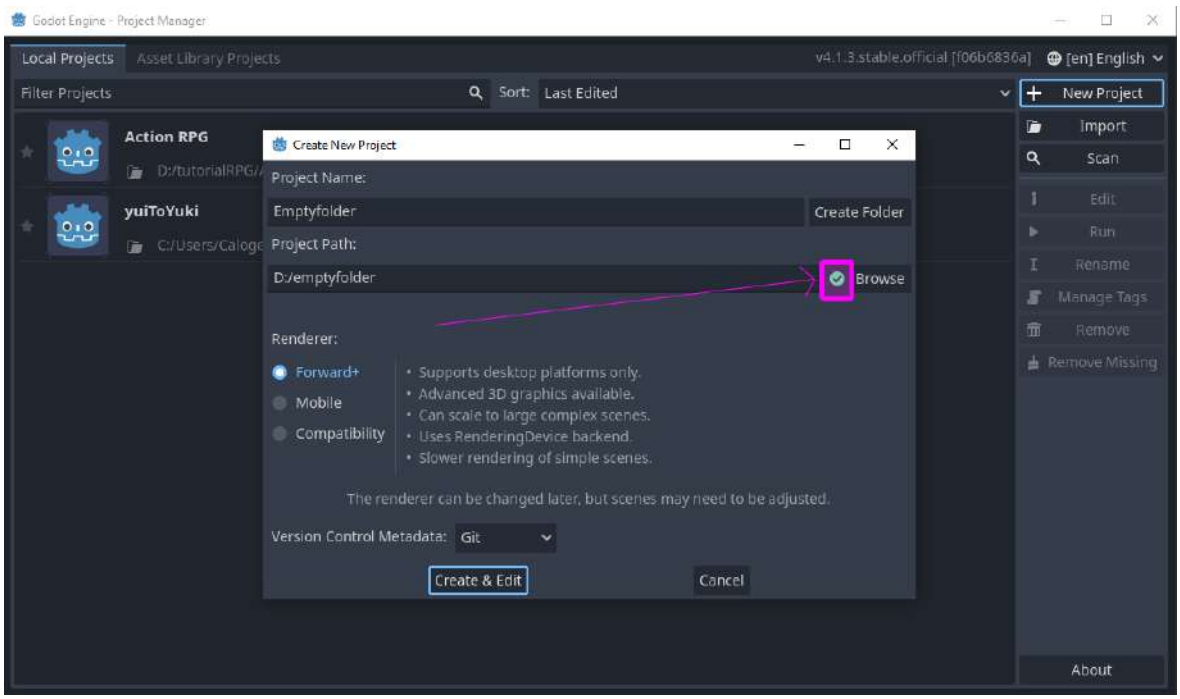
per aprire il file browser di Godot



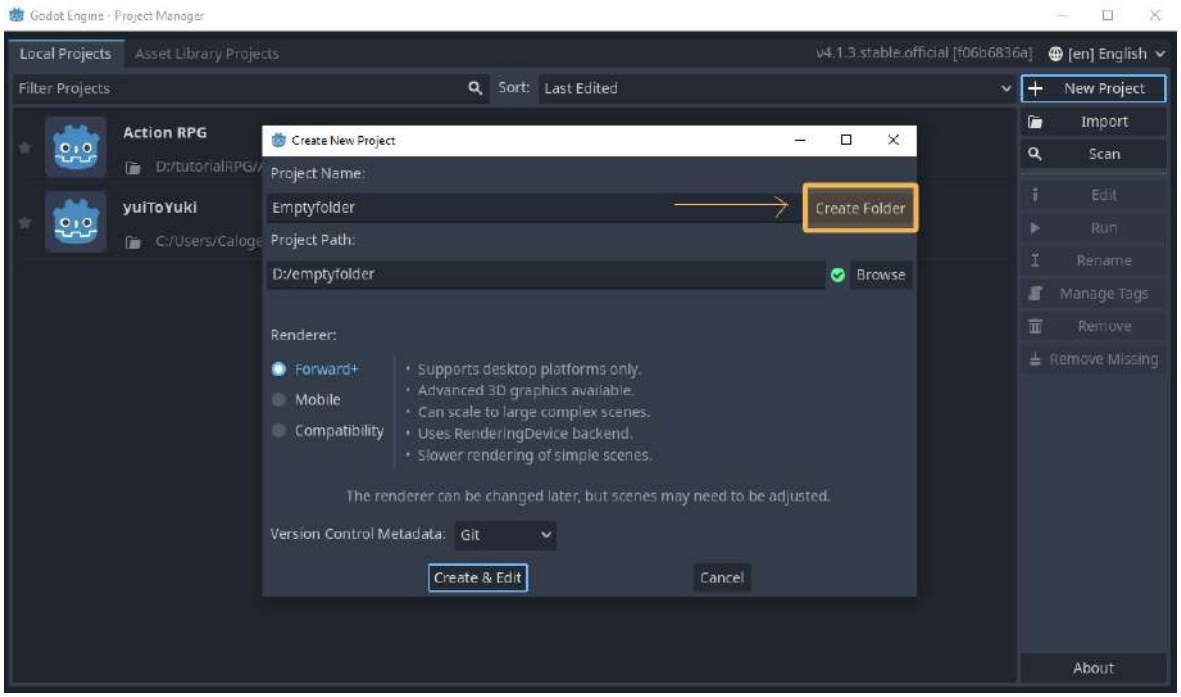
e scegliere una posizione o digitare il path della cartella all'interno del campo `Path`



La spunta verda che compare sulla destra della finestra **Create New Project** indica che l'engine ha rilevato una cartella vuota.

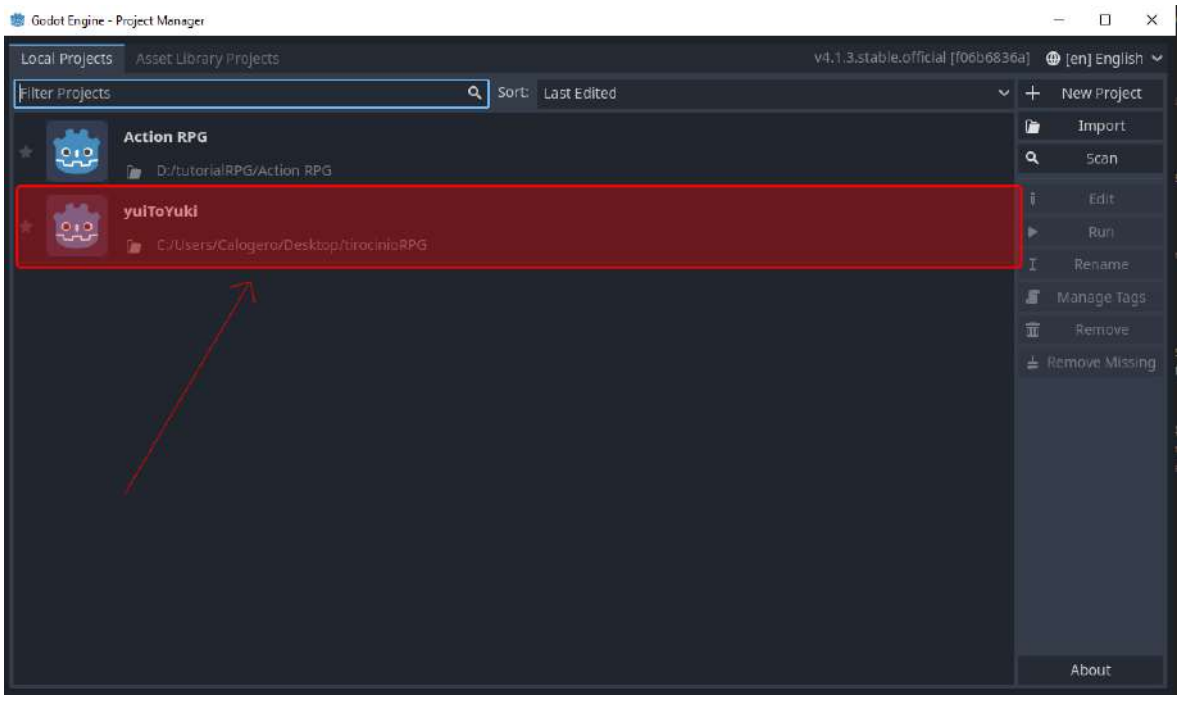


Si può anche cliccare sul pulsante `Create Folder` per creare una cartella vuota sulla base del nome del progetto.

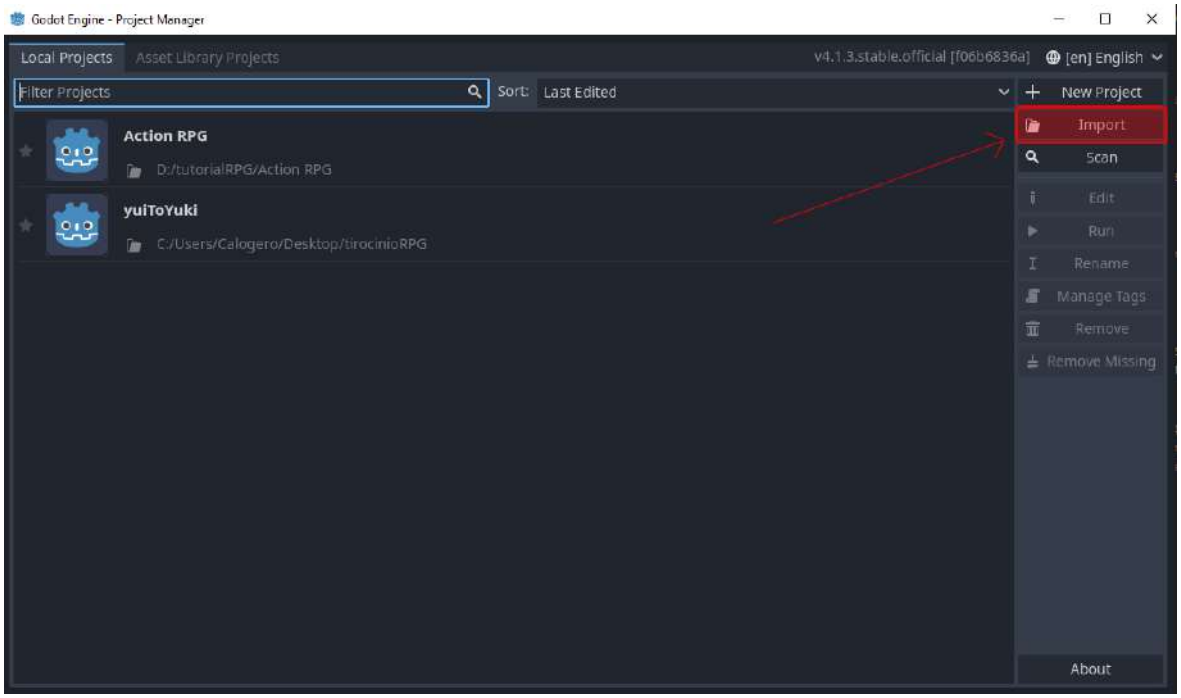


2.2.4.1.2 Aprire e importare progetti

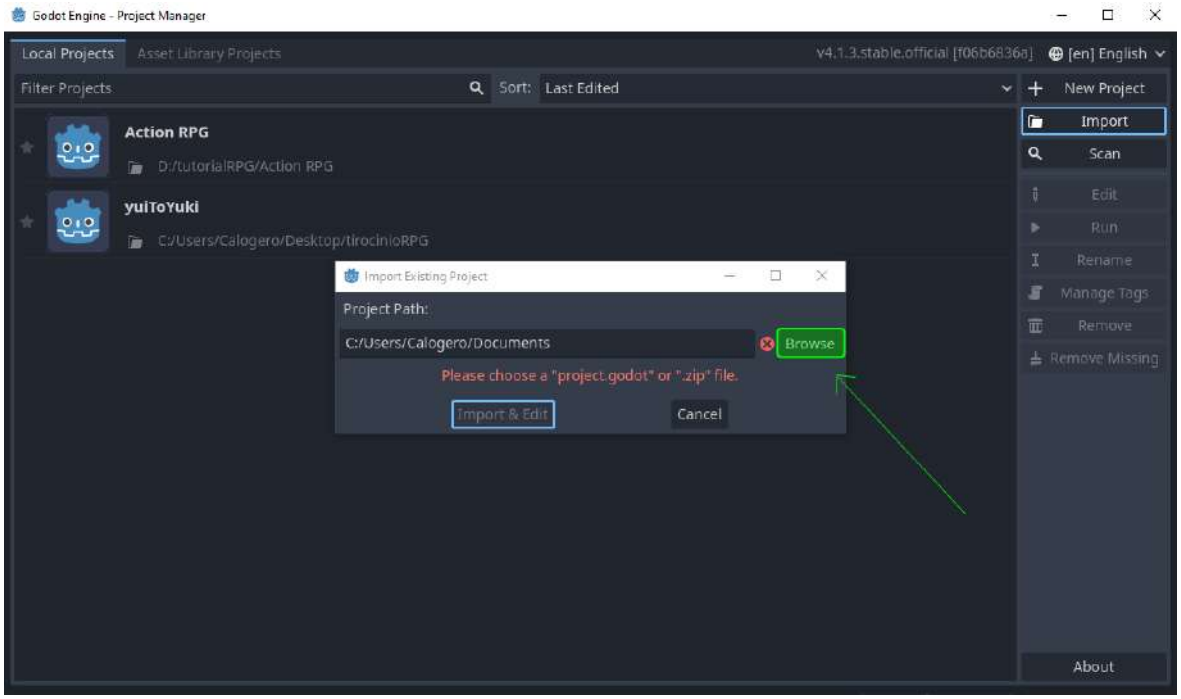
Quando si apre il `Project Manager` la volta successiva alla creazione di un nuovo progetto, questo sarà presente nella lista dei progetti.



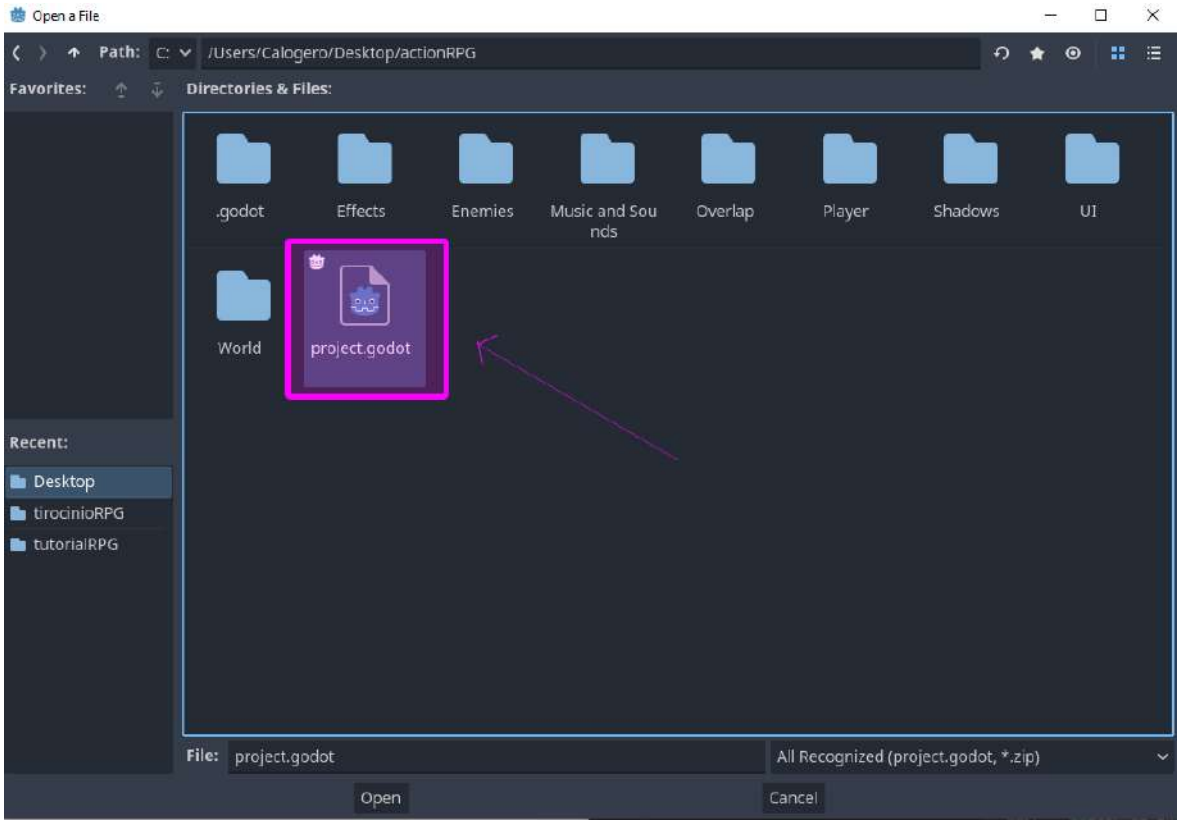
Doppio click per aprirlo nell'editor.
È anche possibile importare progetti esistenti utilizzando l'apposito pulsante `Import`.



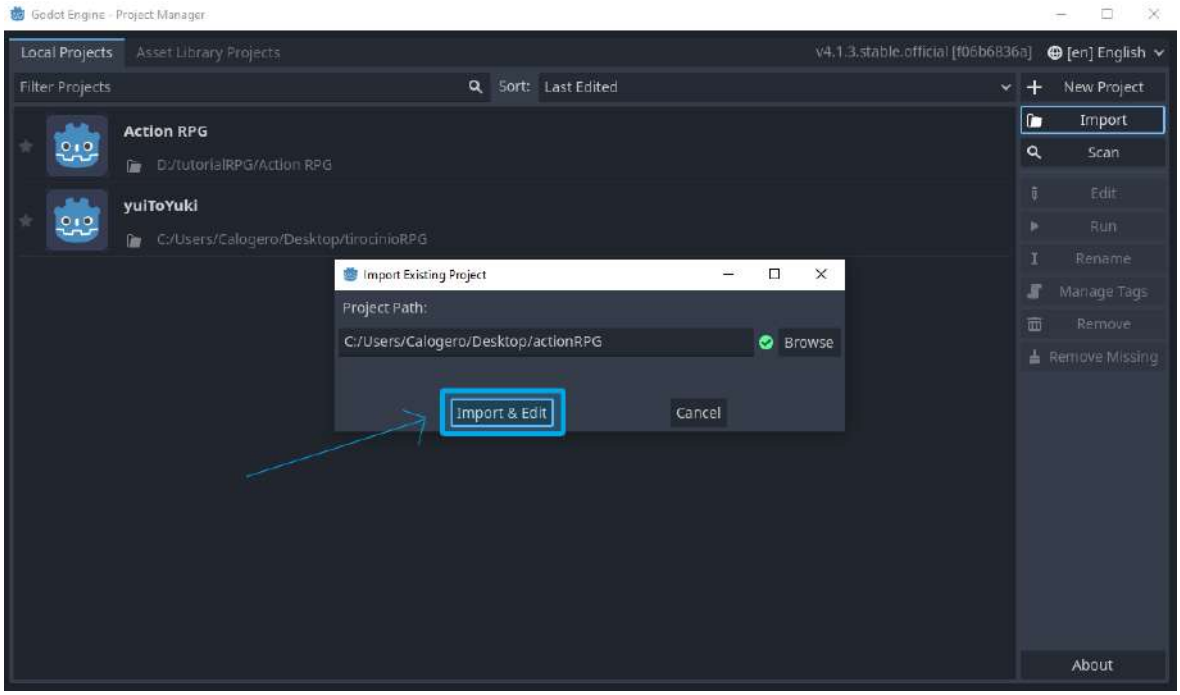
Per importare un nuovo progetto, infatti, basta semplicemente spostarsi sulla cartella che contiene il progetto o il file `project.godot`



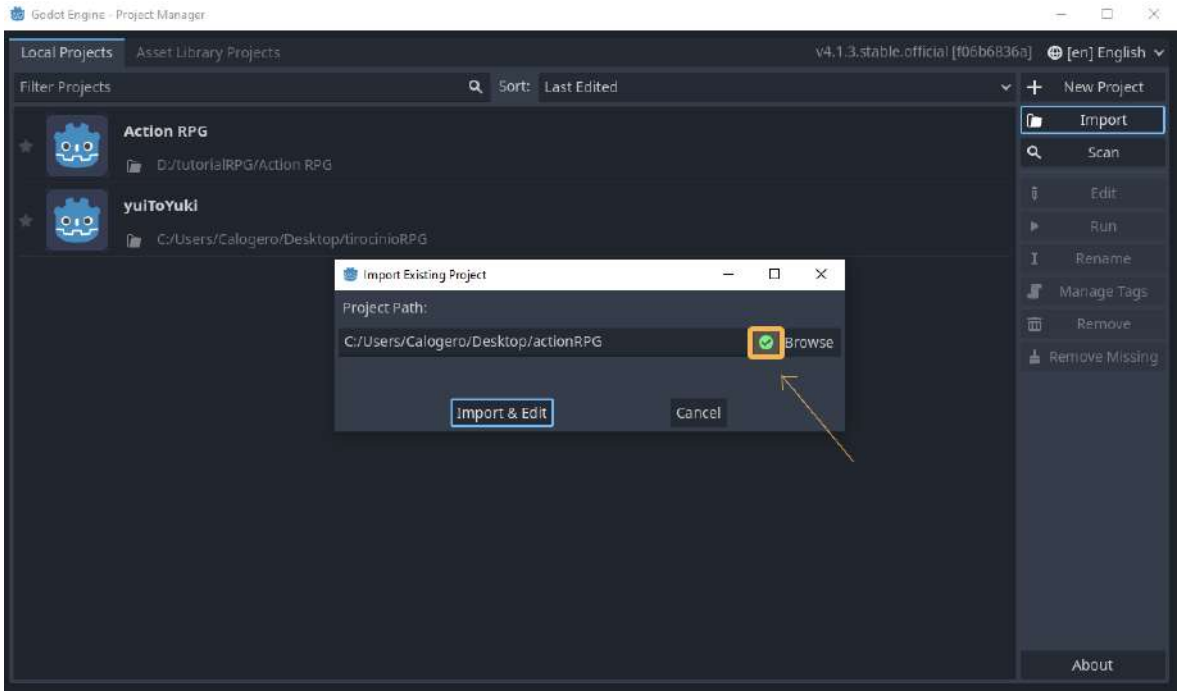
selezionarlo



e poi cliccare sul pulsante `Import & Edit`.

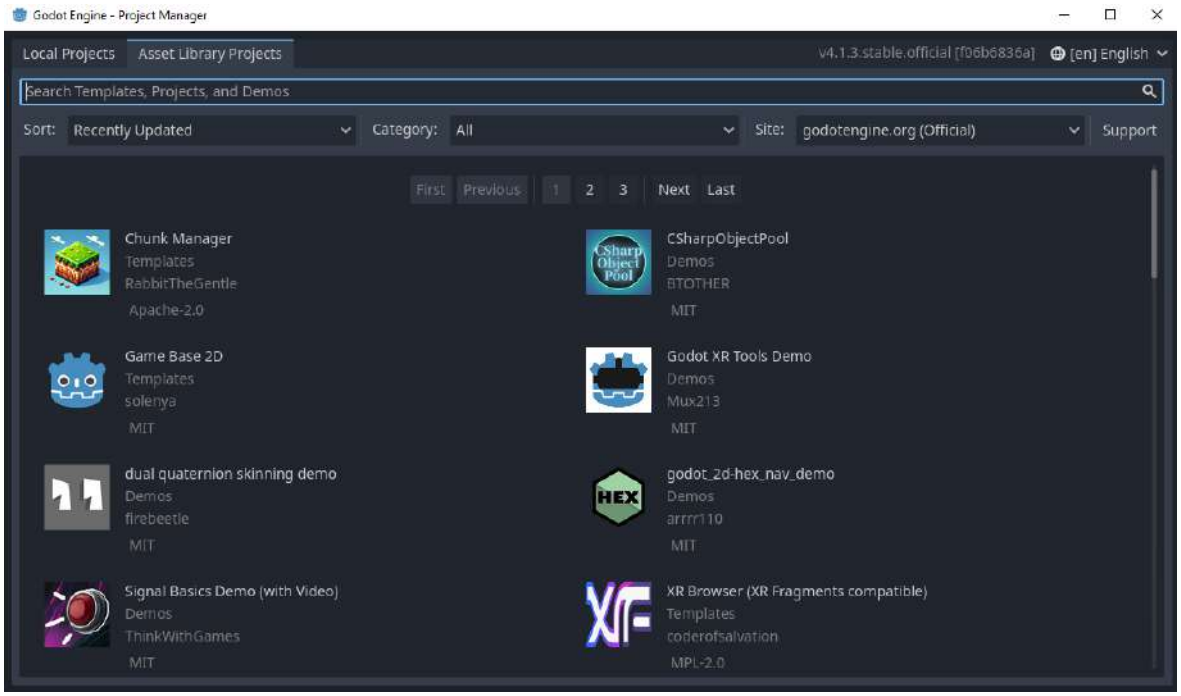


Quando il path della cartella del progetto da importare è corretto, apparirà una spunta verde sulla destra

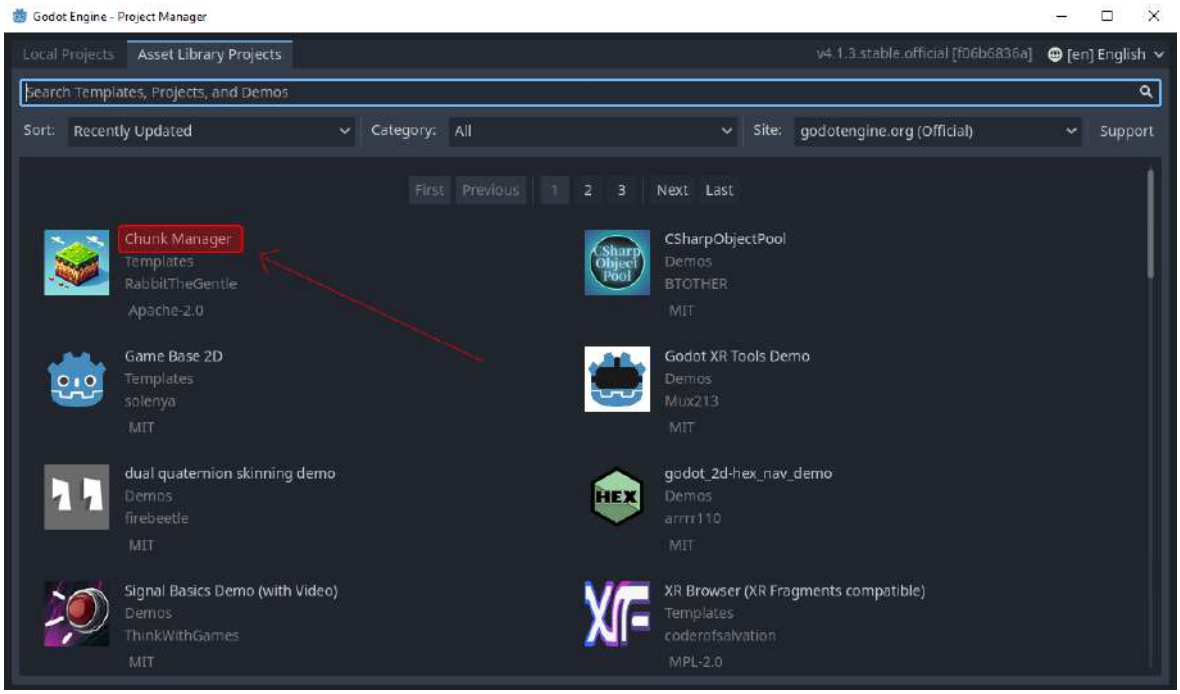


2.2.4.1.3 Scaricare demo e template

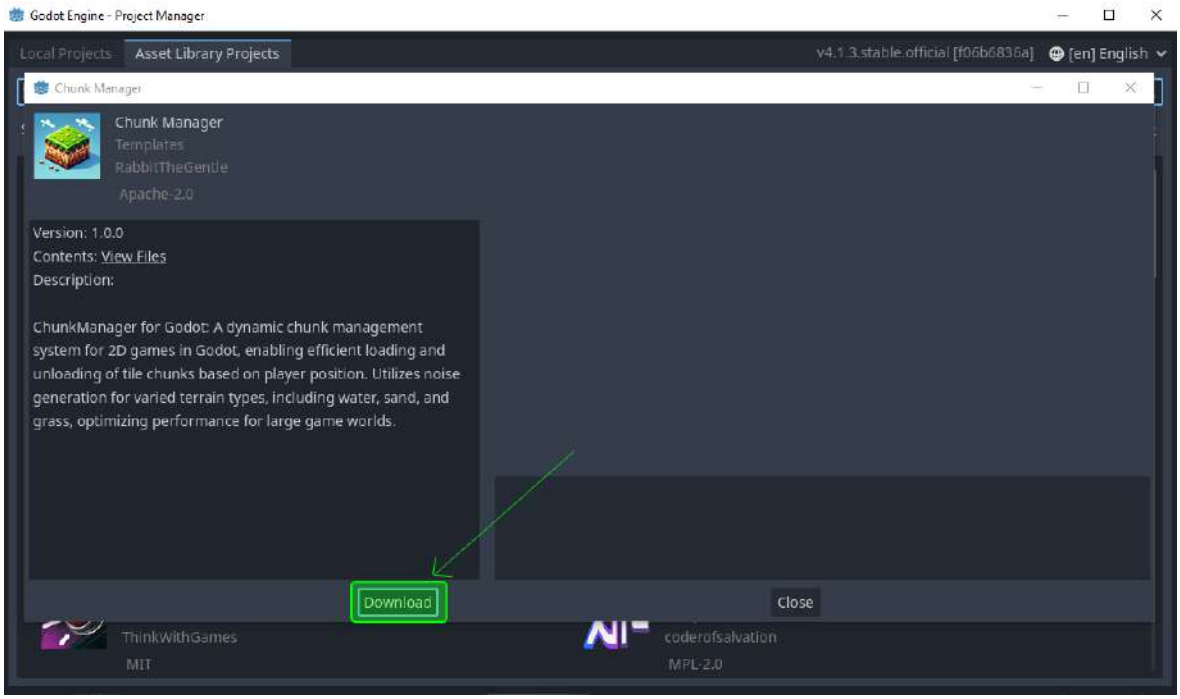
Per iniziare più rapidamente, nell'**Asset Library Projects** tab è inoltre possibile scaricare template e demo di progetti open source dall'**Asset Library**



Per scaricare una demo o un template bisogna cliccare sul titolo



e cliccare sul pulsante `Download` nella pagina che si apre



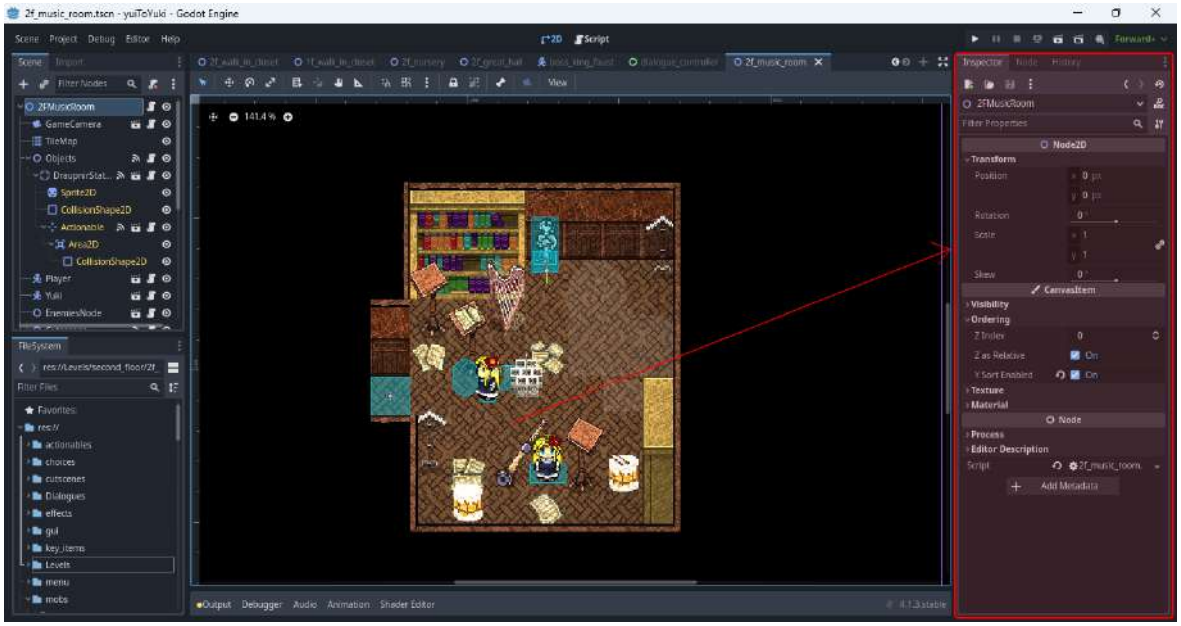
Una volta che il download è terminato, cliccare per installare e scegliere dove si vuole salvare il progetto

2.2.4.2 L'Inspector

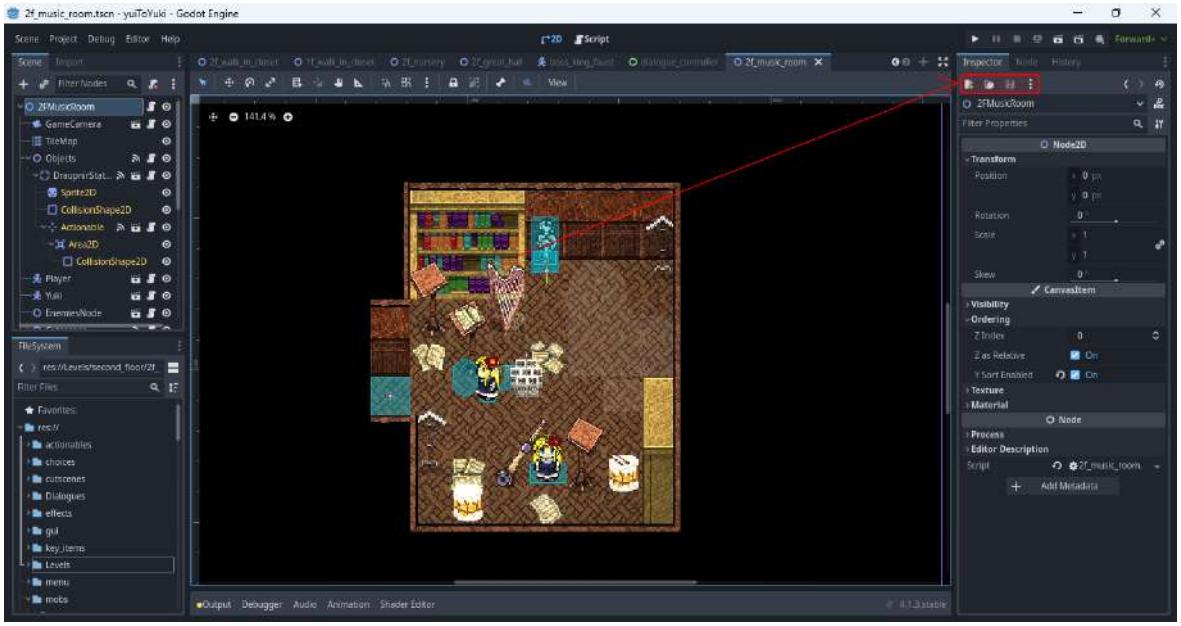
In questa sezione faremo una panoramica dell'Inspector dock.

2.2.4.2.1 Overview dell'interfaccia

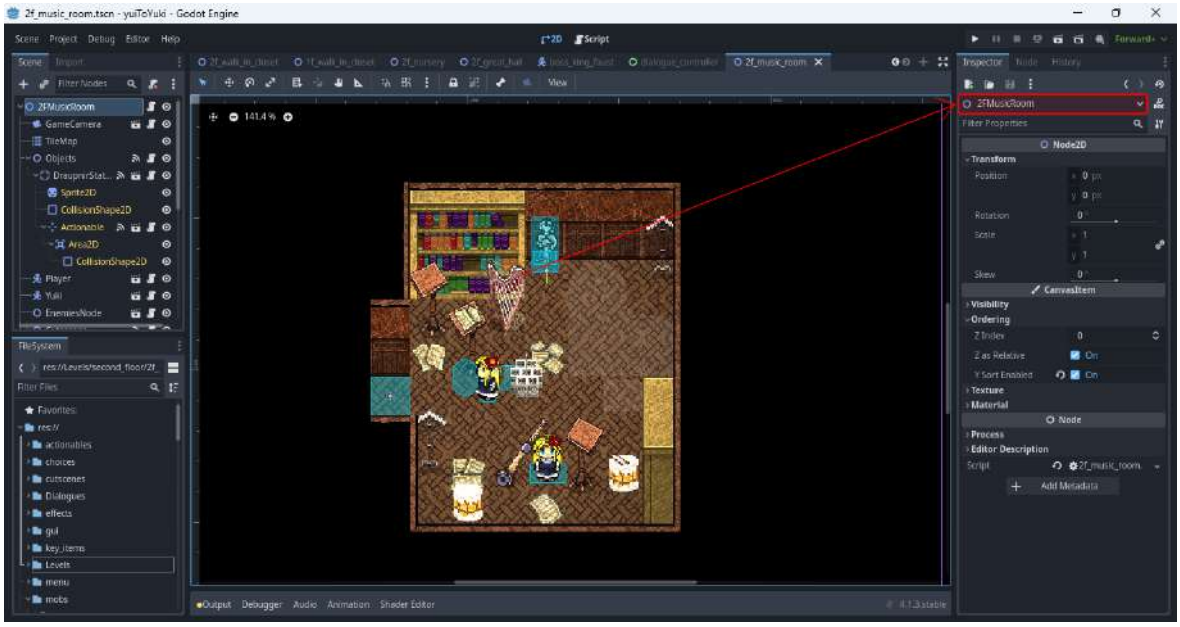
Iniziamo con il guardare le parti principali dell'Inspector dock



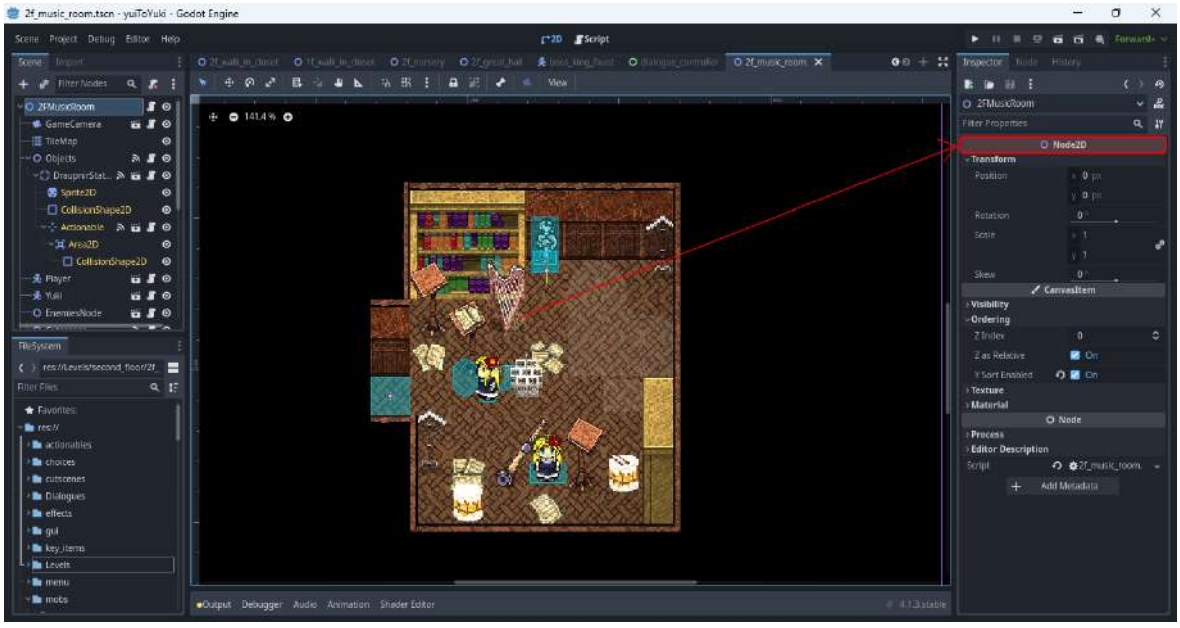
In cima si trovano i pulsanti di file e navigazione



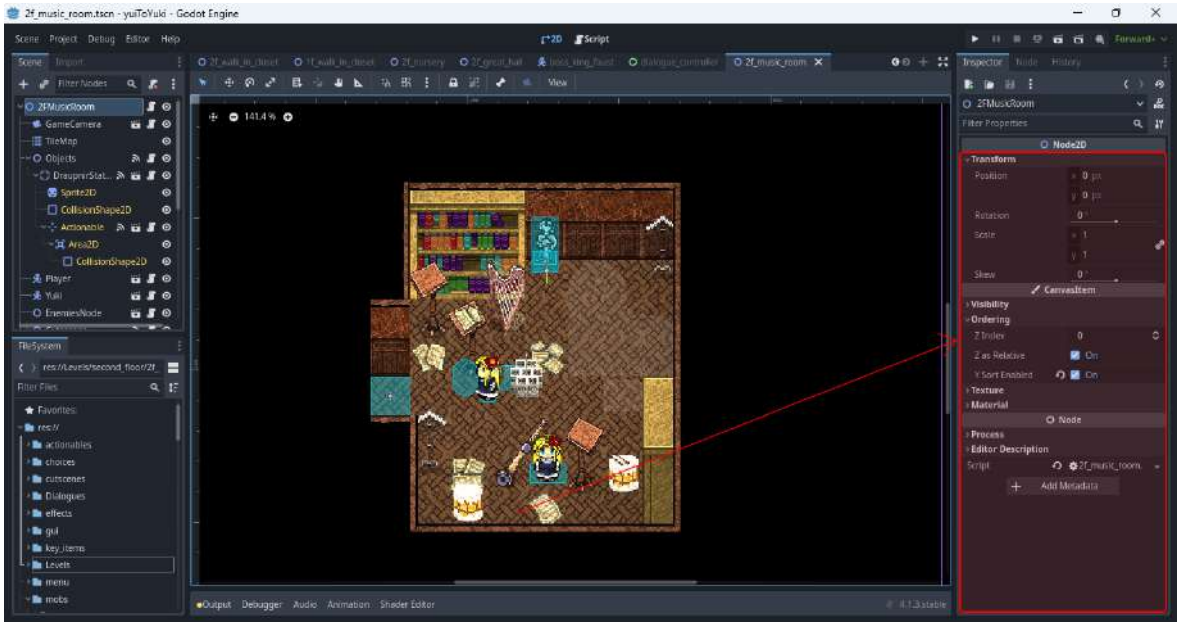
Sotto i pulsanti, sul lato destro, troviamo invece il nome del nodo selezionato



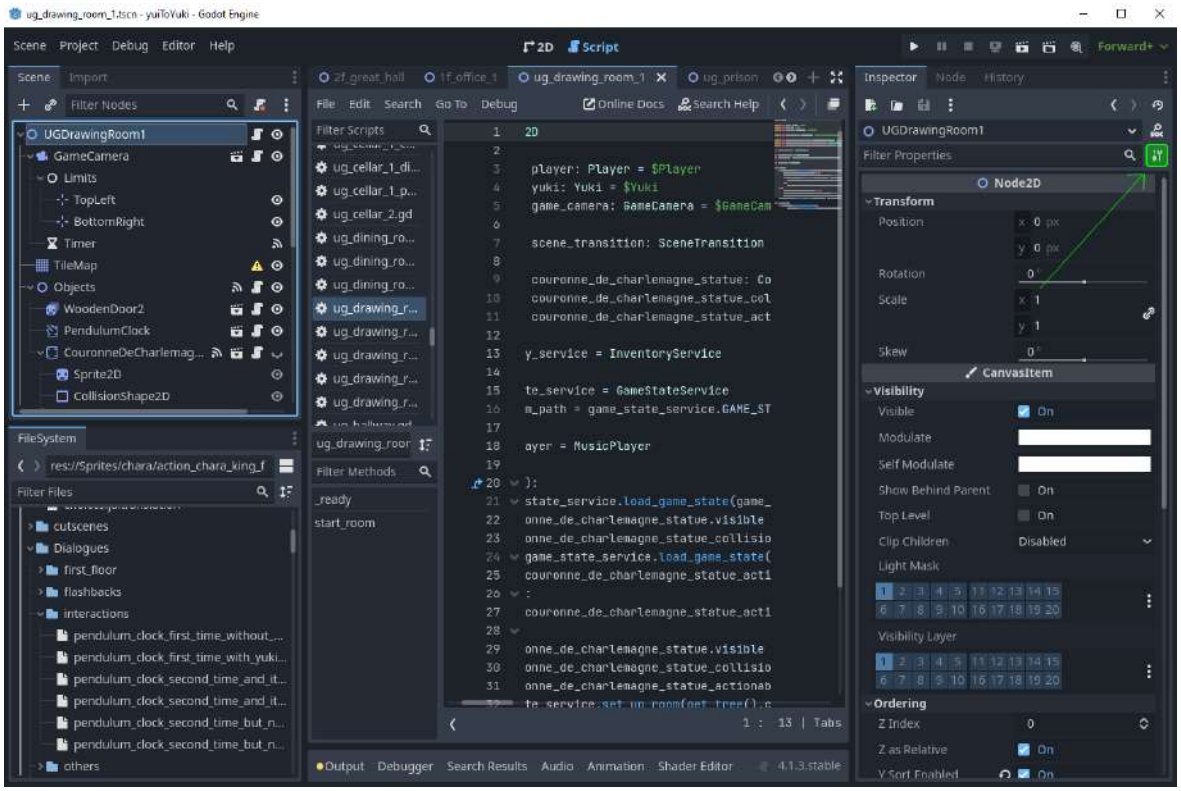
il suo tipo



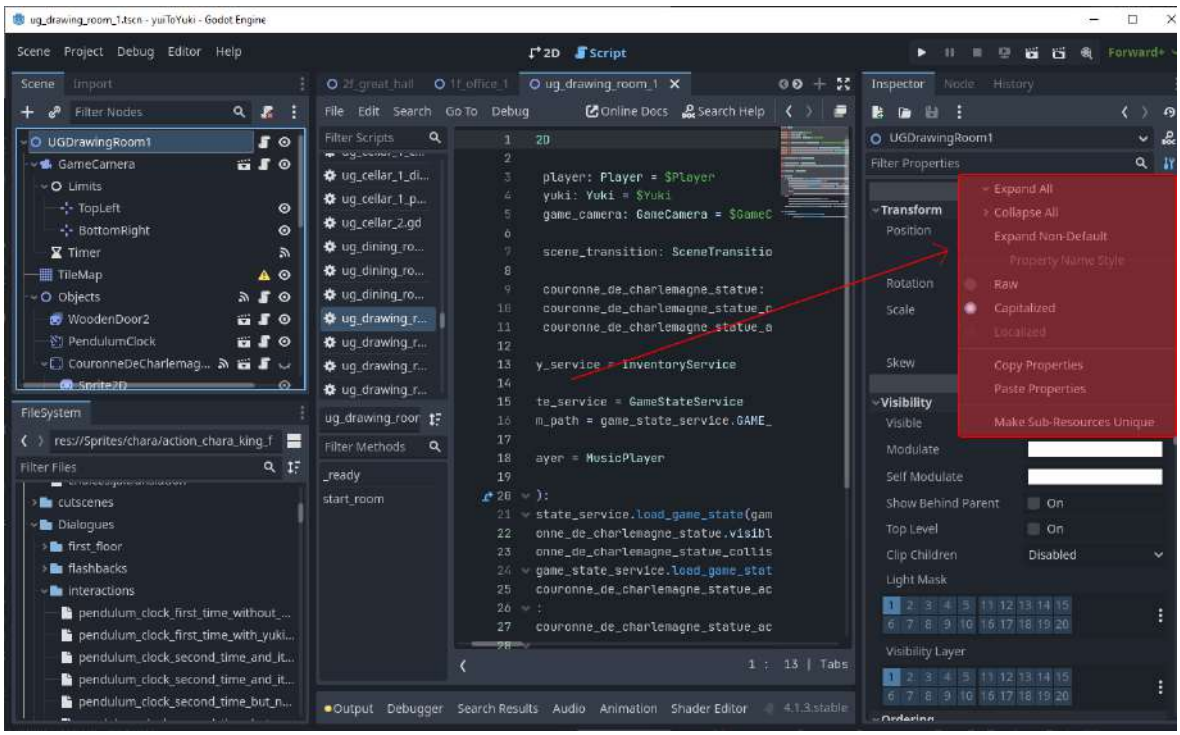
e il tool menu.



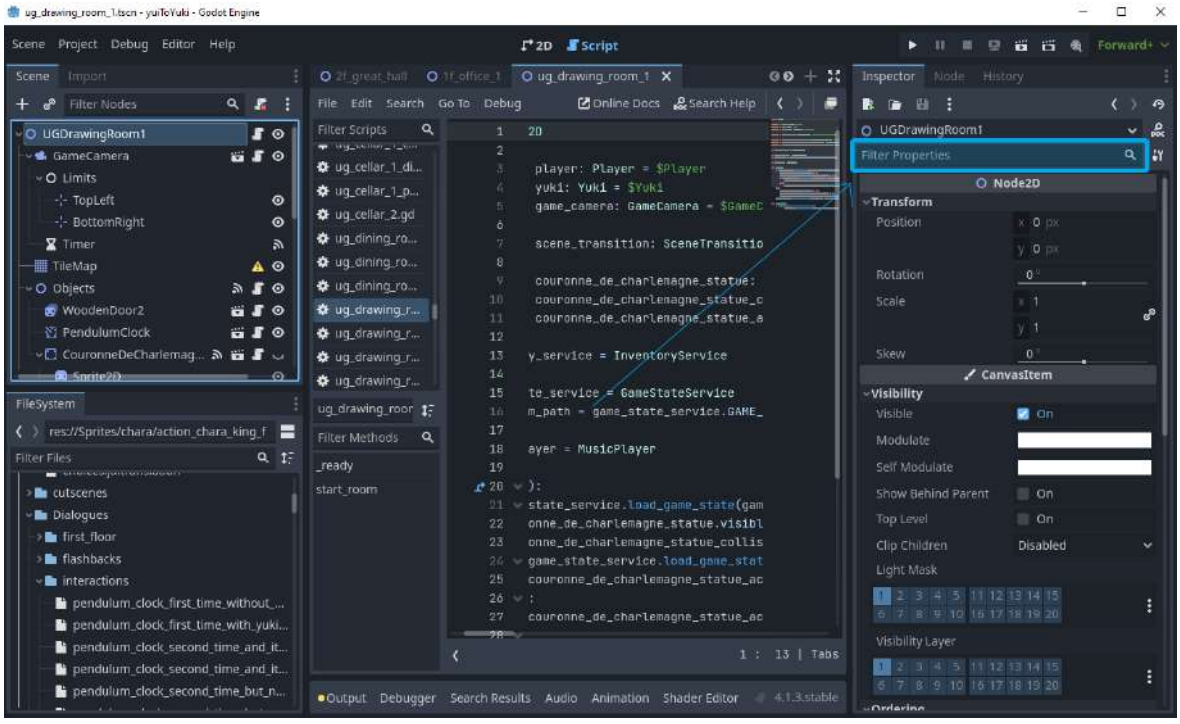
Se si clicca sull'icona del tool menu



apparirà un menu drop-down con alcune opzioni di visualizzazione e di modifica



Tramite la search bar, invece, si può cercare qualsiasi cosa per filtrare le proprietà visualizzate.

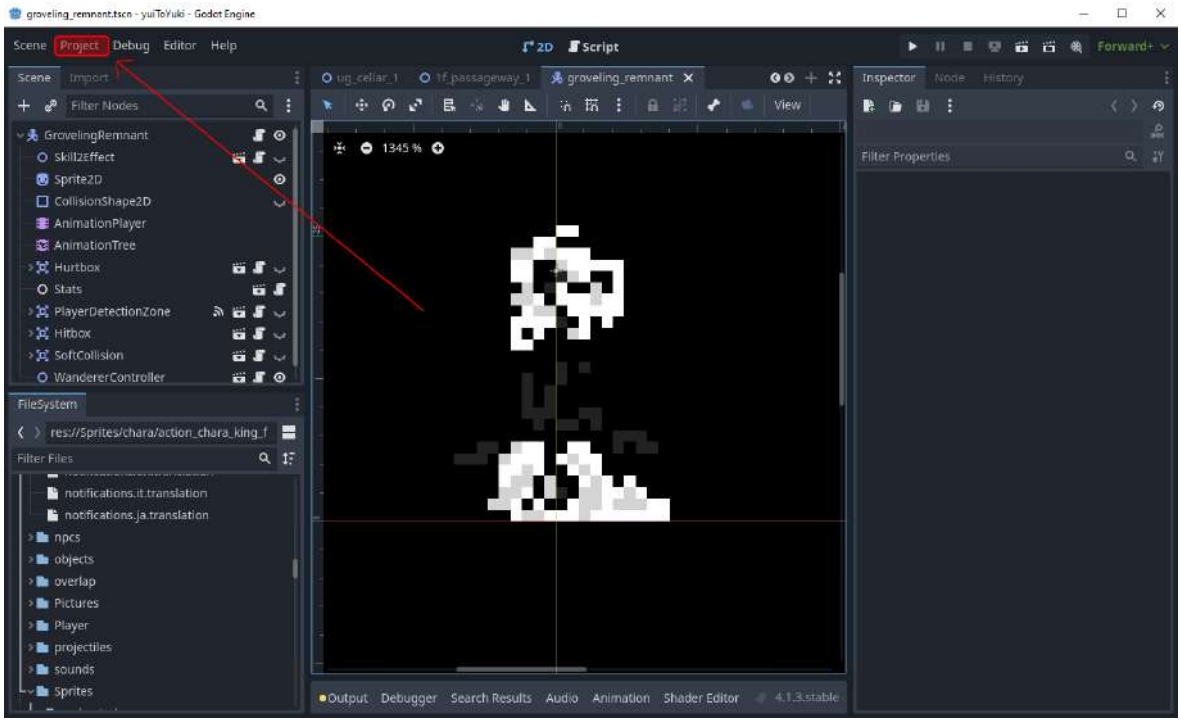


Cancellare il testo cancellerà la ricerca.

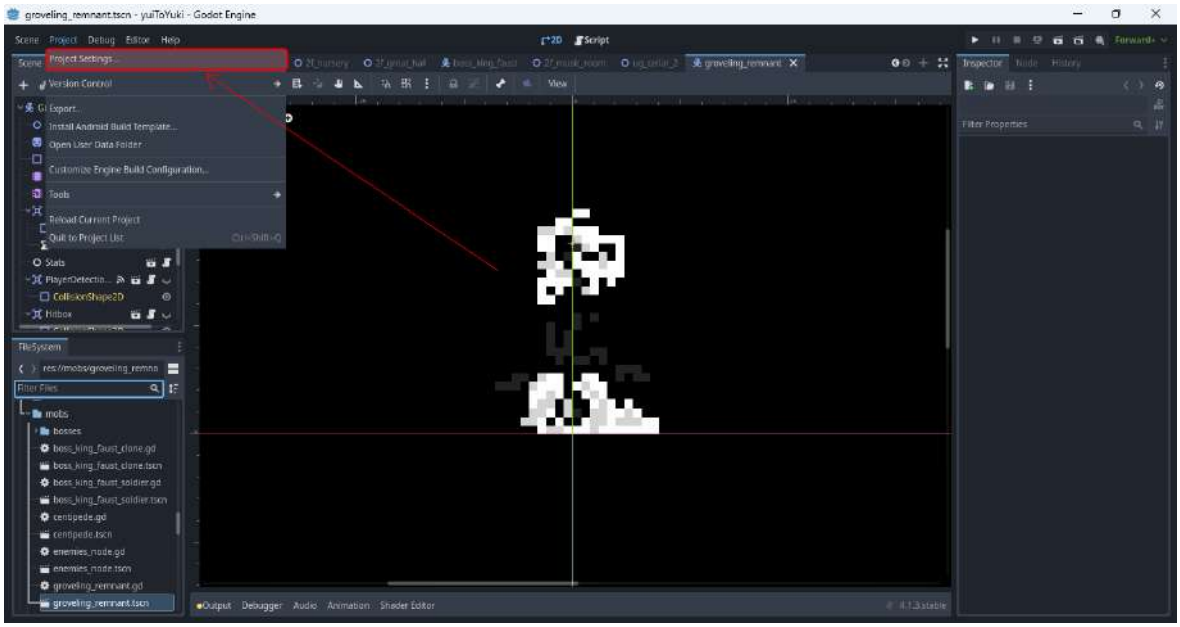
2.2.4.3 Impostazioni del progetto

Godot salva le impostazioni del progetto in un file di semplice testo in formato `INI` chiamato `project.godot`. Esistono dozzine e dozzine di impostazioni che si possono modificare per controllare l'esecuzione di un progetto. Per semplificare il processo, Godot fornisce una finestra di dialogo delle impostazioni del progetto, il quale funge semplicemente da front-end per la modifica del file `project.godot`.

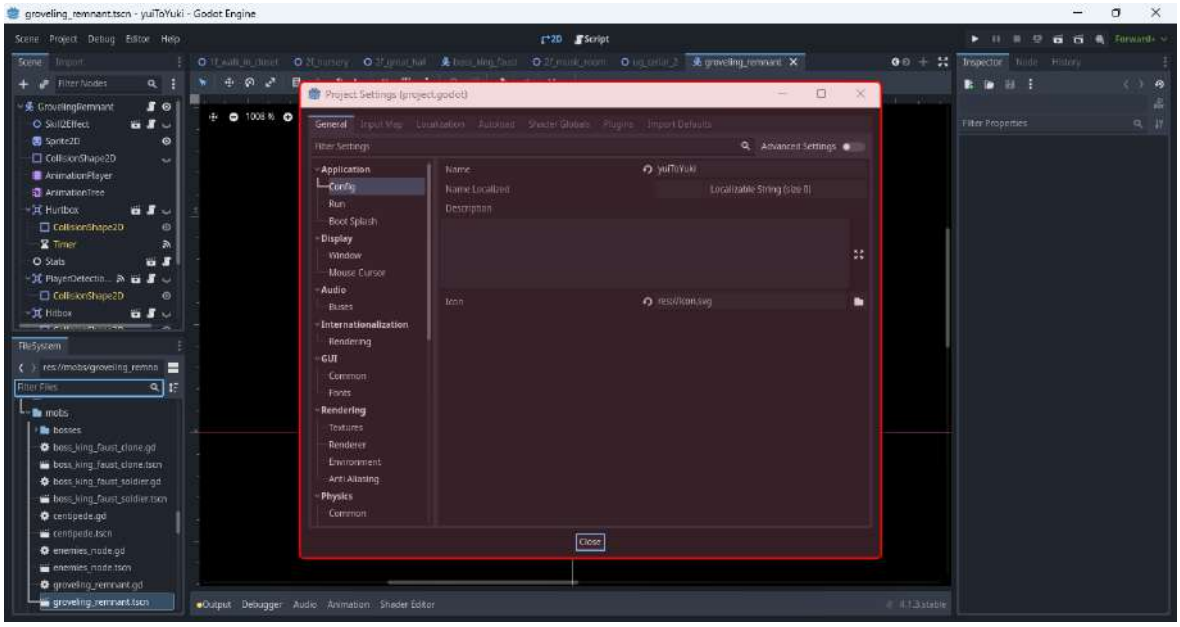
Per accedere alla finestra di dialogo delle impostazioni del progetto cliccare sulla voce `Project` del menu



cliccare su `Project Settings`



quella che si apre è la *finestra di dialogo delle impostazioni del progetto*



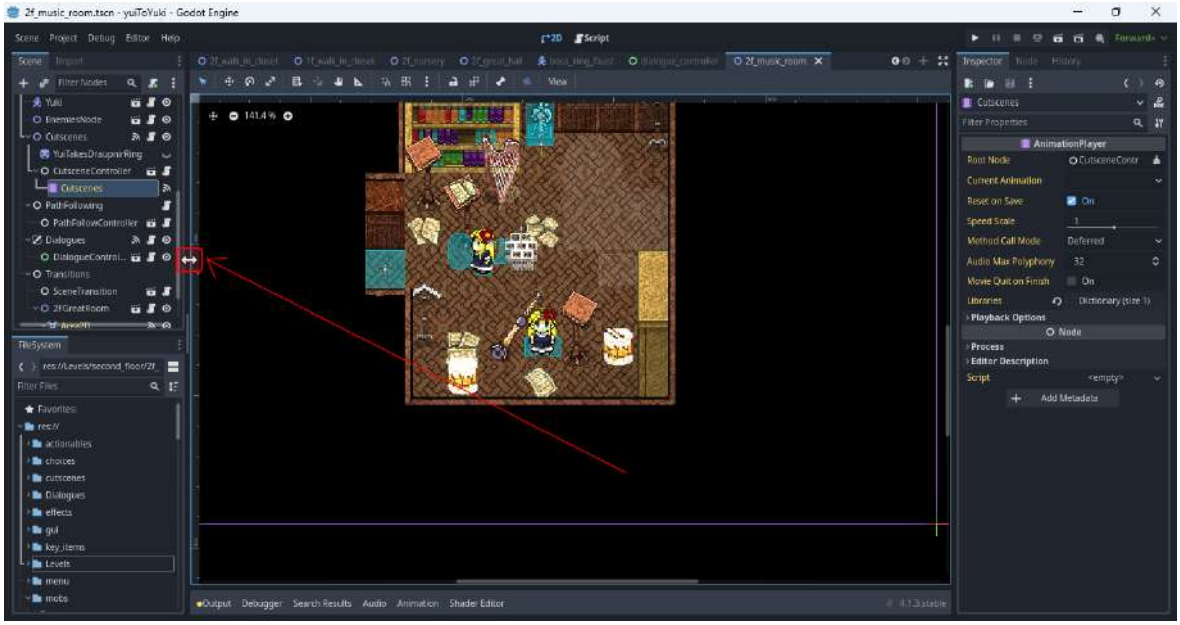
La finestra di dialogo delle impostazioni del progetto fornisce molte opzioni che possono essere salvate in un file `project.godot` e mostra i loro valori di default. Ciò significa che la proprietà verrà salvata nel file `project.godot` e ricordata.

2.2.4.4 Personalizzazione dell'interfaccia

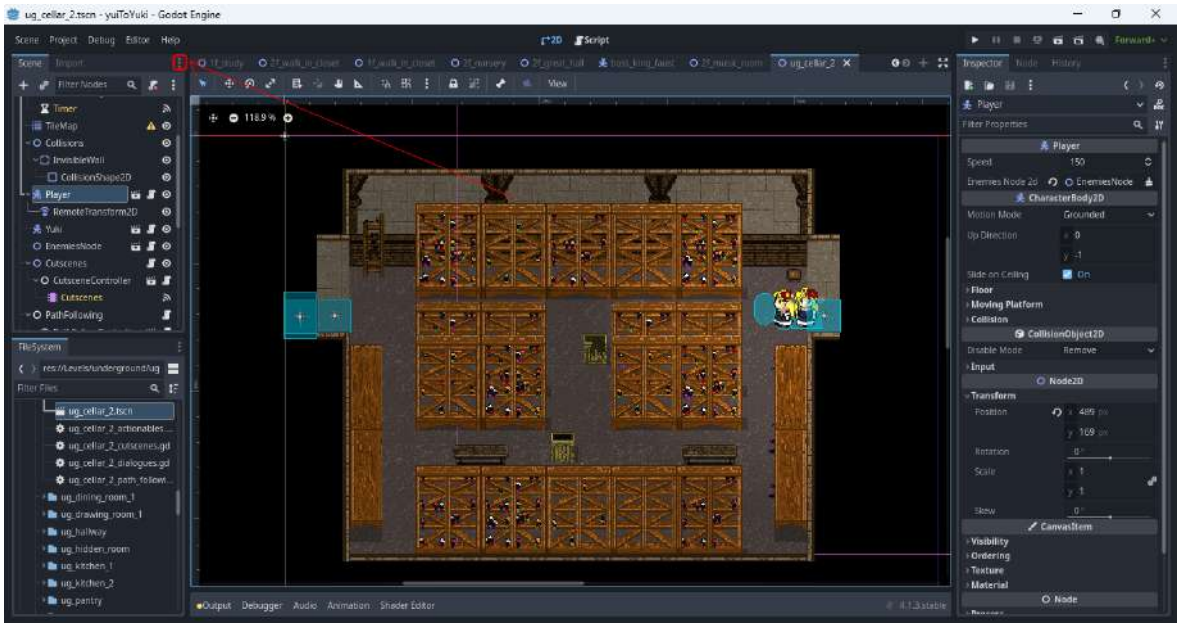
L'interfaccia di Godot vive in una singola finestra per default. A partire da Godot 4.0, è possibile dividere molteplici elementi per separare le finestre e sfruttare meglio i setup multi-monitor.

2.2.4.4.1 Muovere e ridimensionare i dock

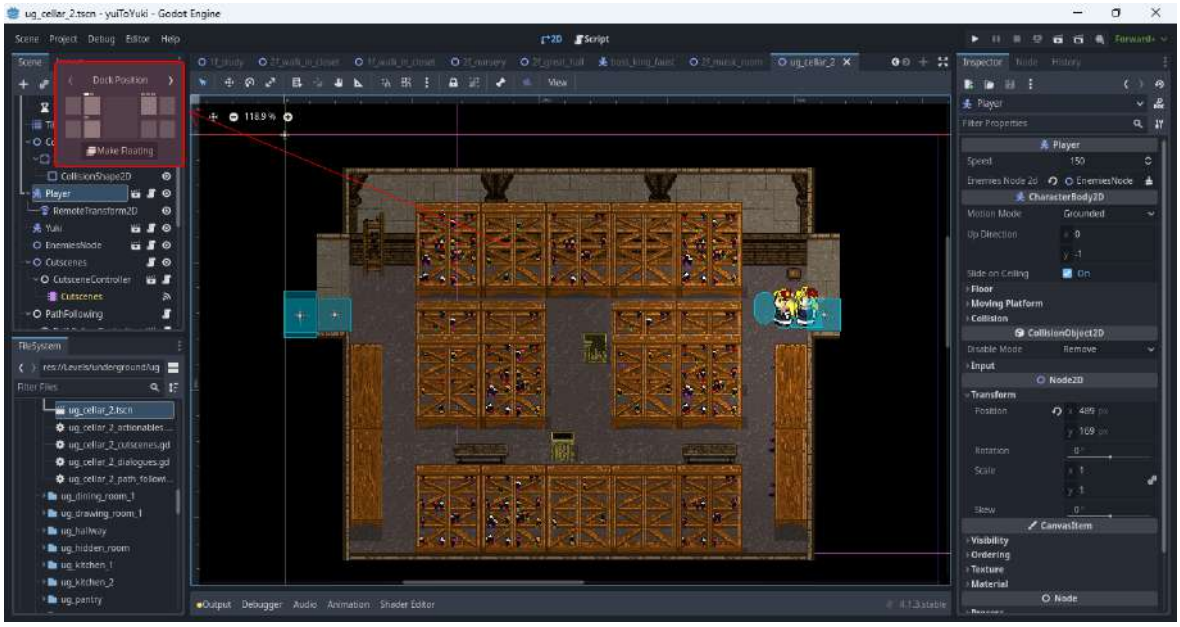
Fare click e trascinare il bordo di qualsiasi dock o panel per ridimensionarlo orizzontalmente o verticalmente.



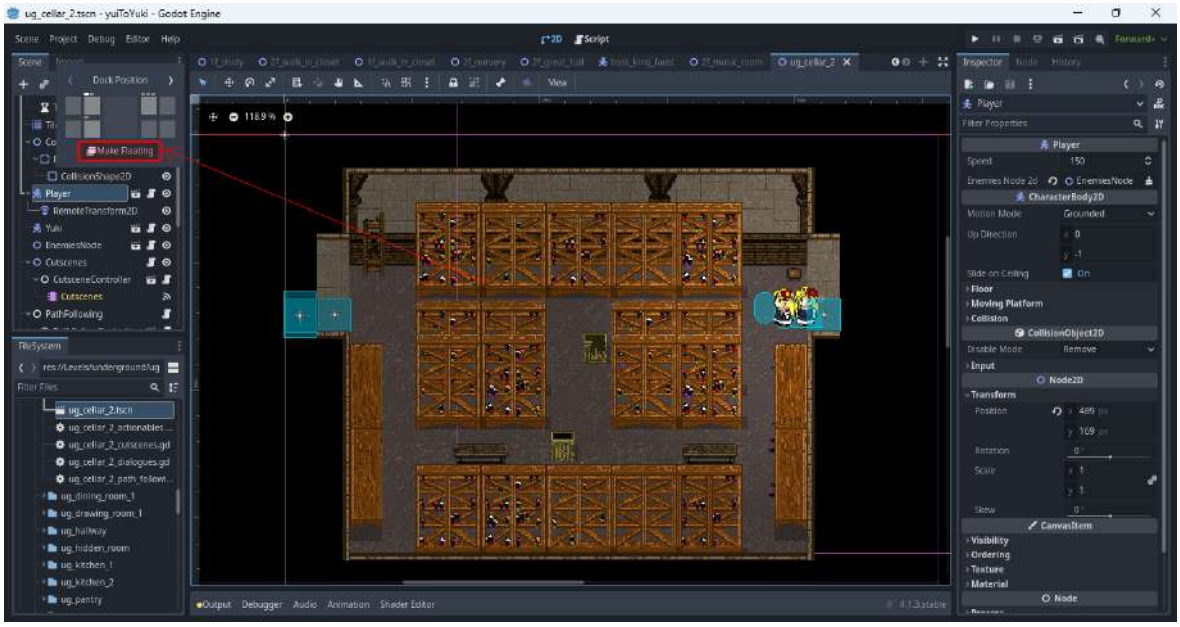
Fare click sull'icona dei "3 puntini verticali" in cima a qualsiasi dock



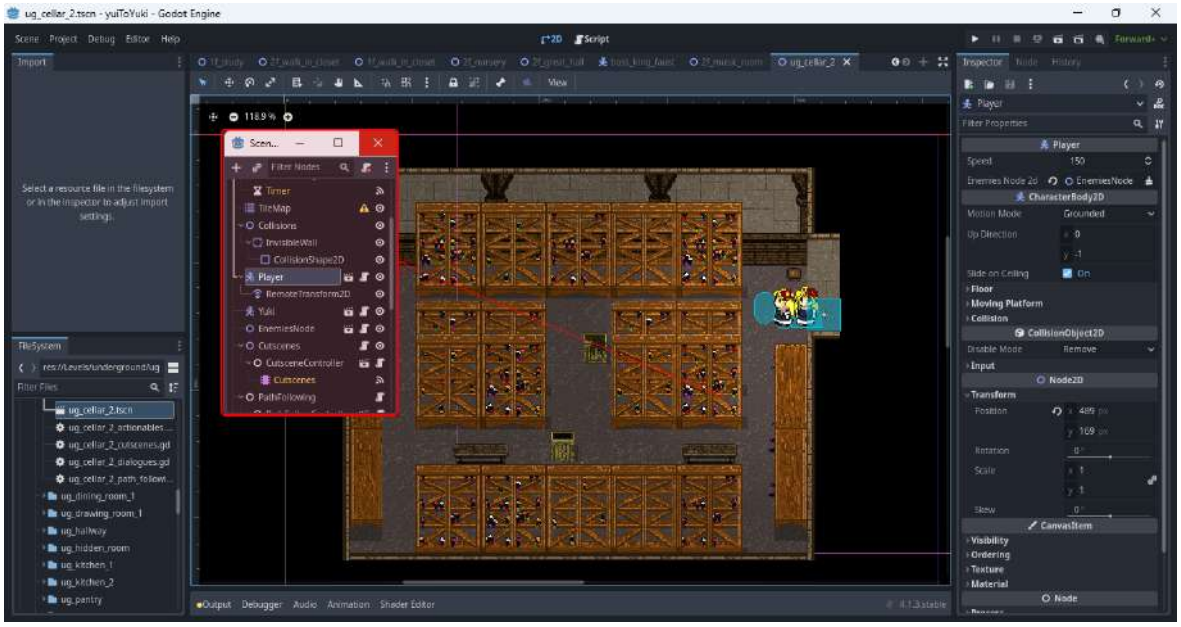
per aprire un sottomenu dal quale si può cambiare la posizione del dock



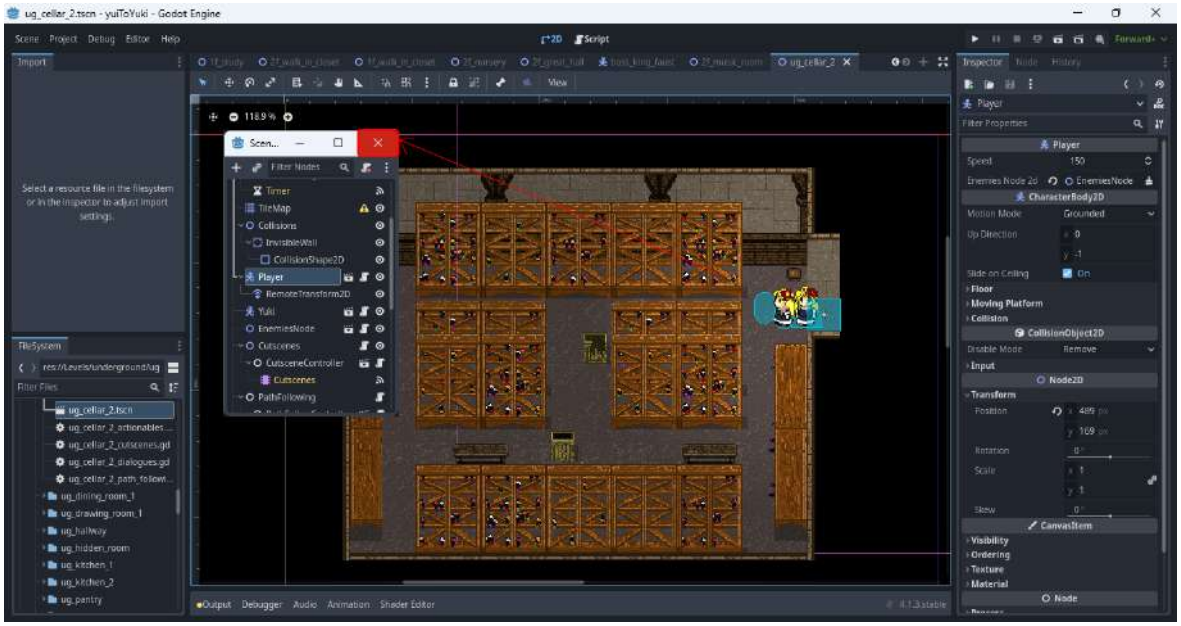
o cliccando sul pulsante `Make Floating`.



dividerlo in una finestra separata



Per portare un floating dock nuovamente nella finestra dell'editor, chiudere la finestra del dock utilizzando il pulsante `x` nell'angolo in alto a destra (o in alto a sinistra su macOS) della finestra stessa.



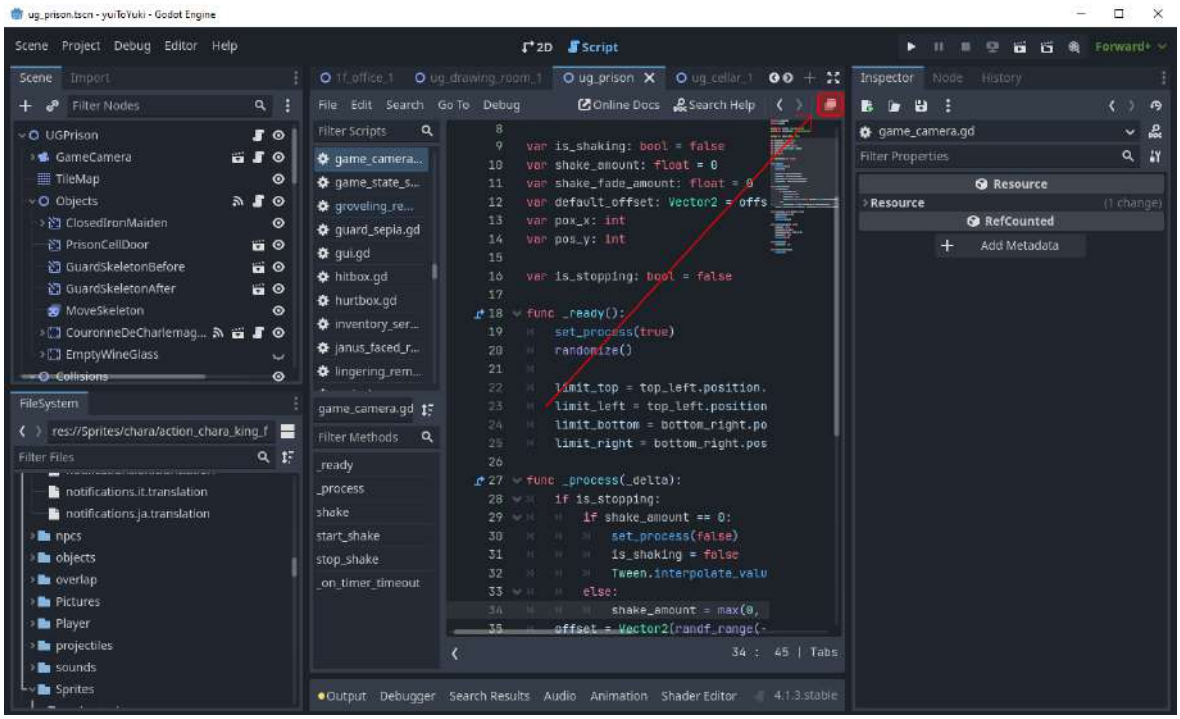
In alternativa, si può premere `Alt + F4` mentre si ha il focus sulla finestra divisa.

2.2.4.4.2 Dividere l'editor degli script o delle shader in una finestra a sé stante

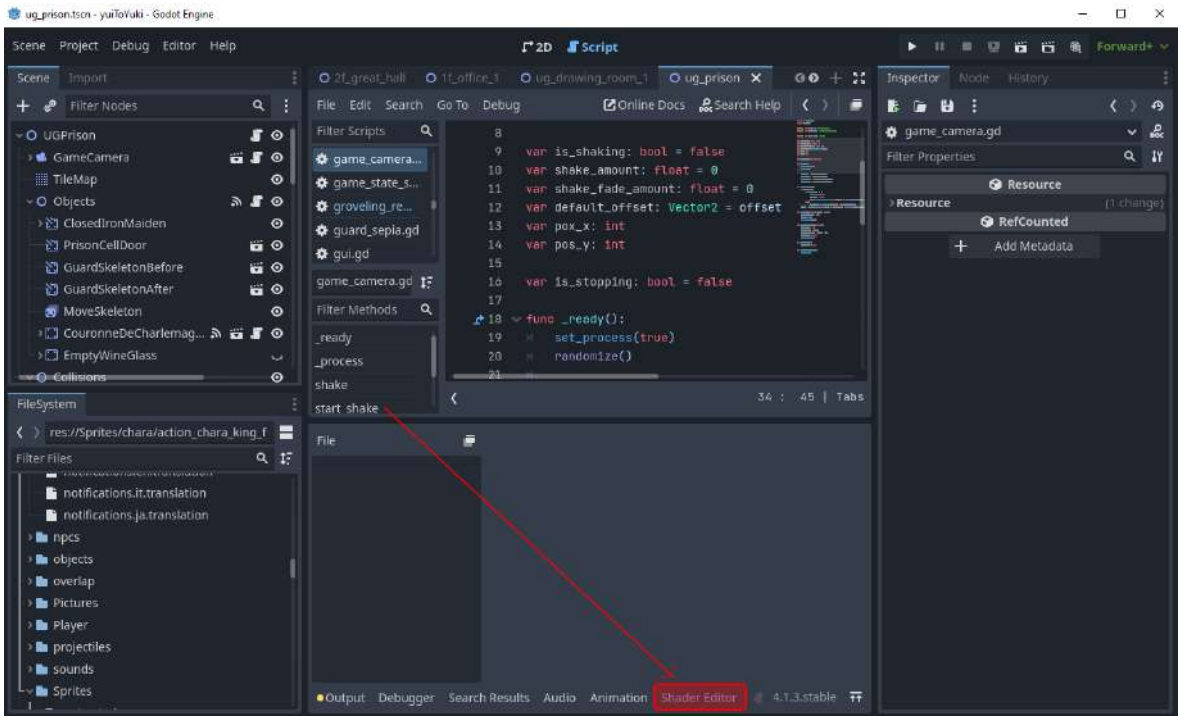
> **Nota**
>
> Questa feature è disponibile solamente sulle piattaforme che supportano la creazione di finestre multiple: Windows, macOS e Linux.
> Questa feature non è disponibile se l'opzione **Single Window Mode** è attiva nelle impostazioni dell'editor.

A partire da Godot 4.1, si può dividere l'editor degli script o delle shader in una finestra a sé stante.

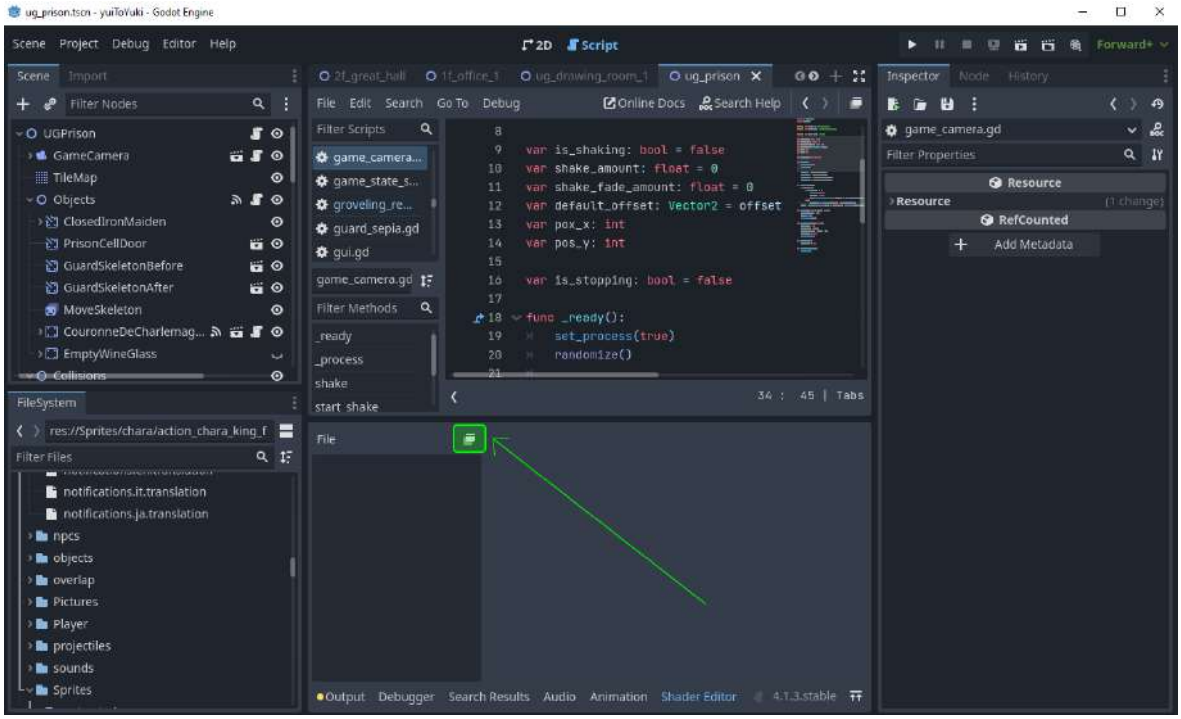
Per dividere l'editor degli script in una finestra a sé stante, fare click sul pulsante corrispondente nell'angolo in alto a destra della finestra dell'editor degli script.



Per dividere l'editor degli shader in una finestra a sé stante,



far click sul pulsante corrispondente nell'angolo in alto a destra della finestra dell'editor.



Per tornare allo stato precedente (con l'editor degli script/shader incorporato nella finestra dell'editor), chiudere la finestra divisa utilizzando il pulsante `x` nell'angolo in alto a destra della finestra (o in alto a sinistra su macOS). Altrimenti, si può premere `Alt + F4` mentre si ha il focus sulla finestra divisa.

2.2.4.4.3 Personalizzazione dei layout dell'editor

È possibile salvare e caricare una configurazione dock a seconda del tipo di task su cui si sta lavorando. Ad esempio, quando si sta lavorando sull'animazione di un personaggio, potrebbe risultare più comodo avere i dock disposti in modo diverso rispetto a quando si sta progettando un livello.

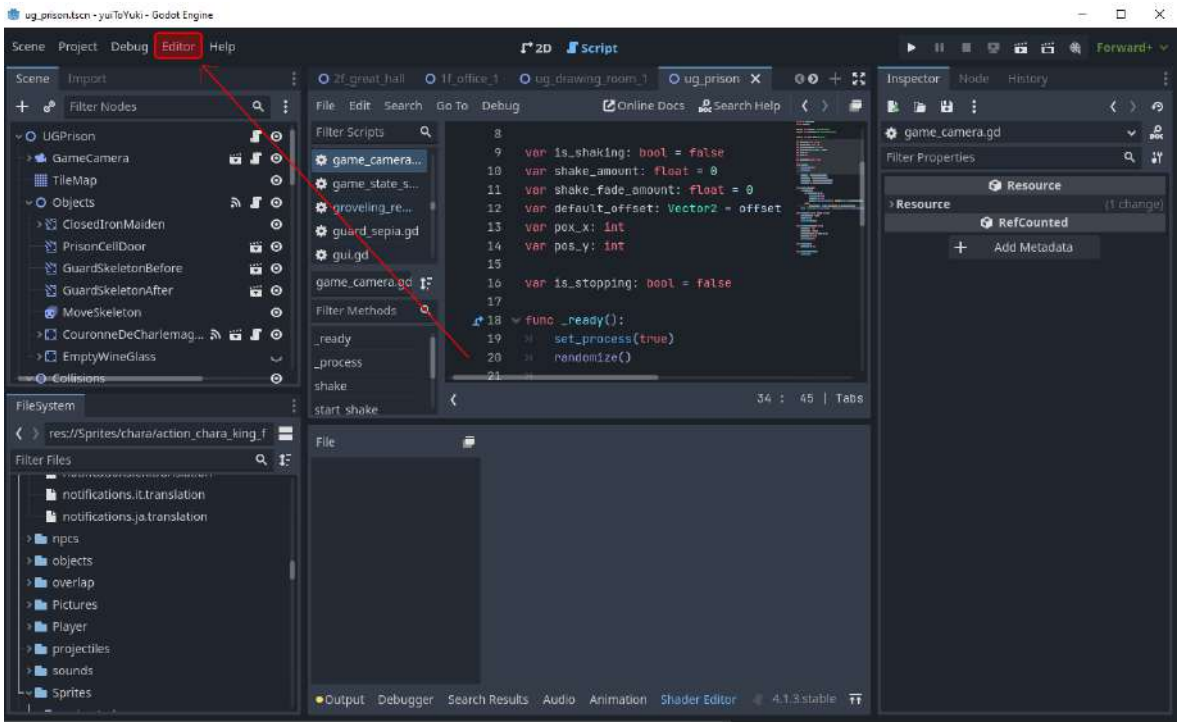
A questo scopo, Godot fornisce un modo per salvare e ripristinare i layout dell'editor. Prima di salvare un layout, è necessario apportare modifiche ai dock che si desidera salvare. Le modifiche che persistono nel layout salvato sono le seguenti:

- spostare un dock
- ridimensionare un dock
- rendere un dock floating
- cambiare la posizione o la dimensione di un floating dock
- proprietà del FileSystem dock: split mode, display mode, sorting order, file list display mode, selected paths e unfolded paths.

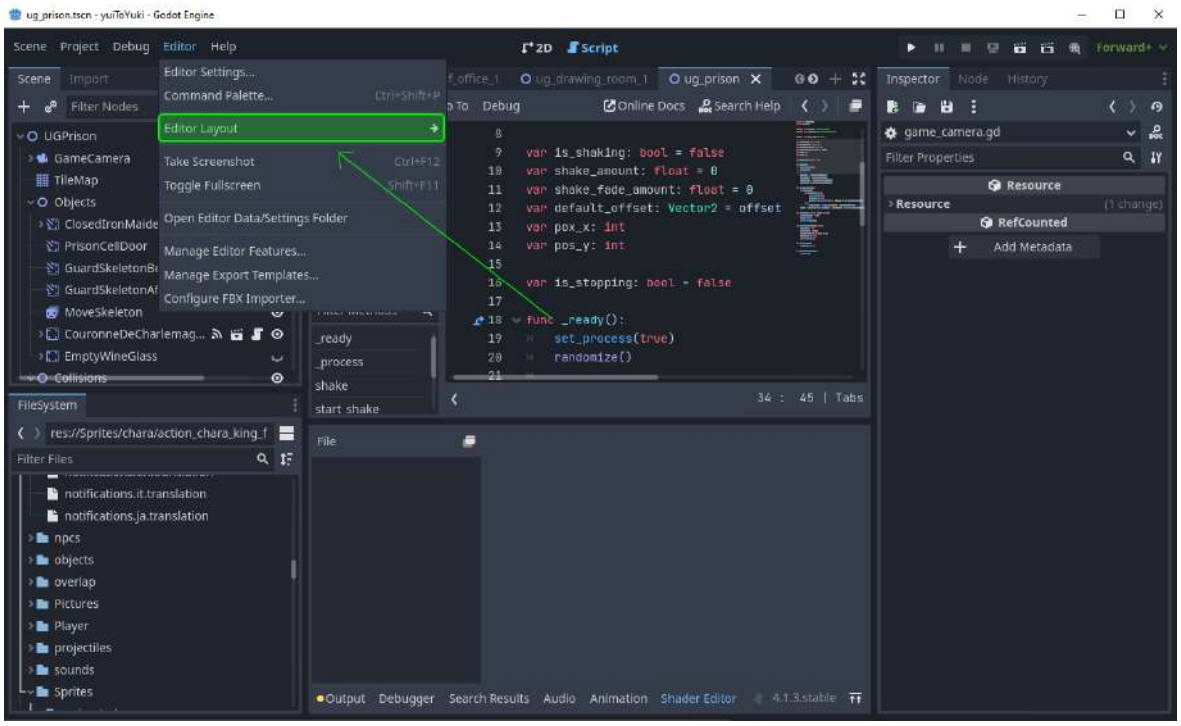
Nota

La separazione dell'editor degli script o degli shader in una propria finestra *non* viene mantenuta come parte di un layout

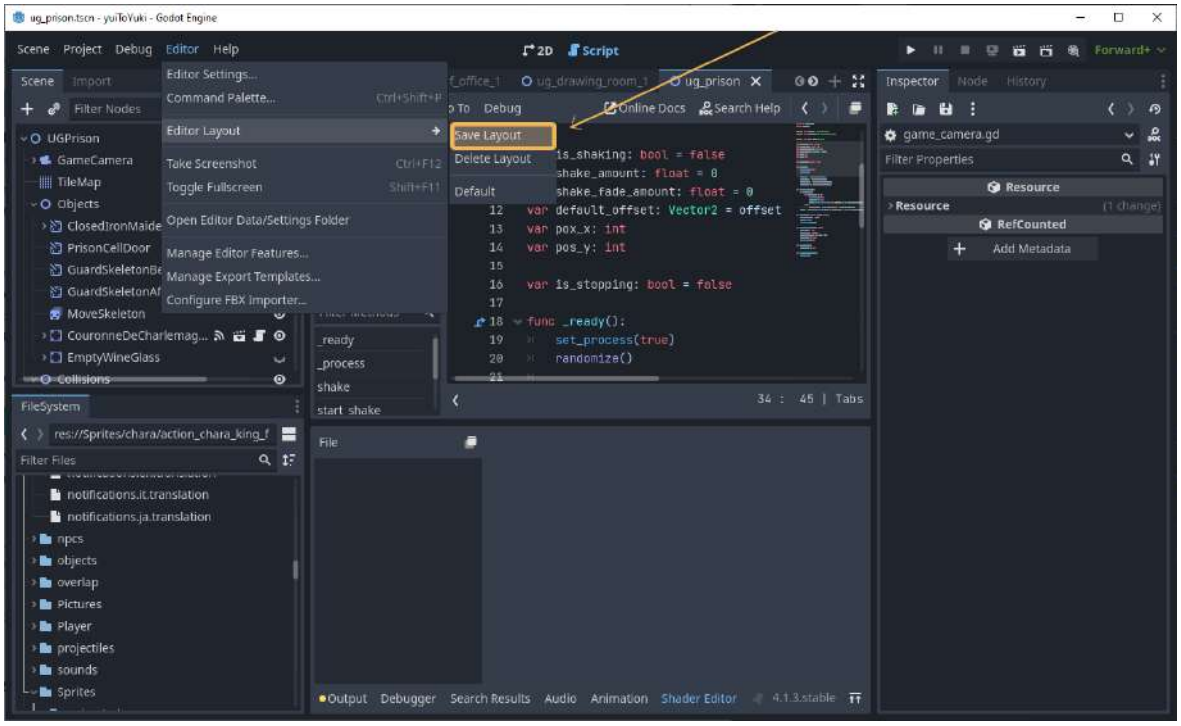
Dopo aver apportato le modifiche, apriamo l'**Editor** menu in cima all'editor



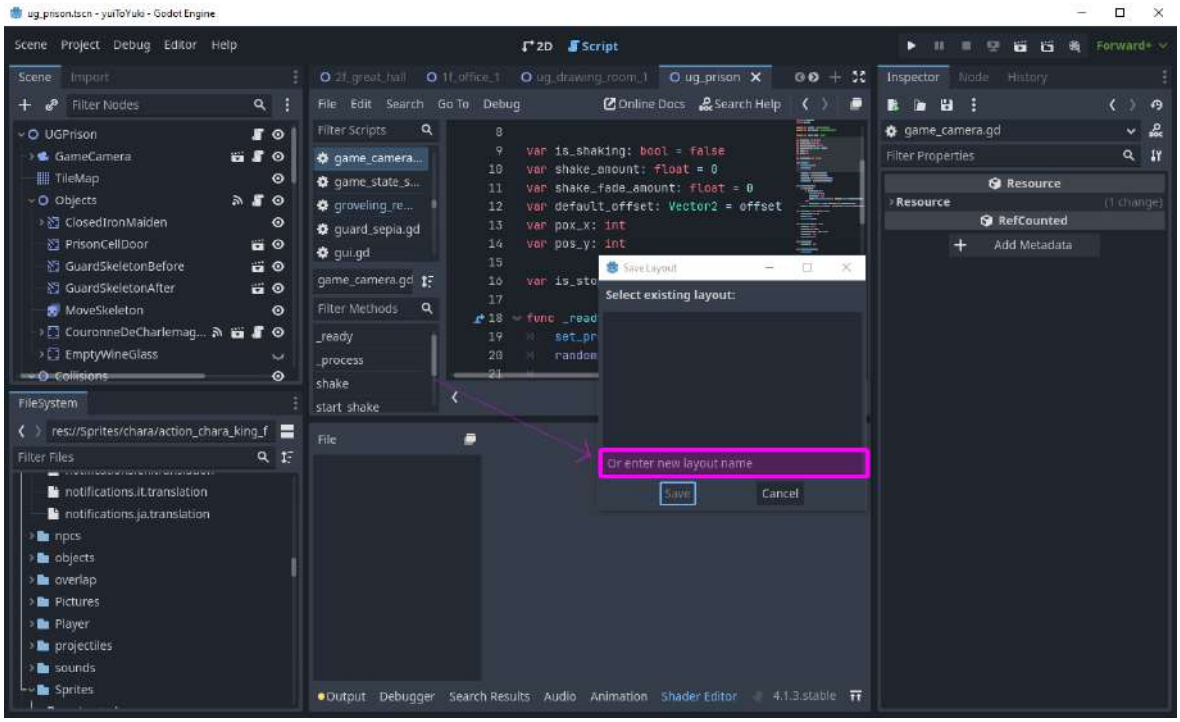
clicchiamo su **Editor Layouts**



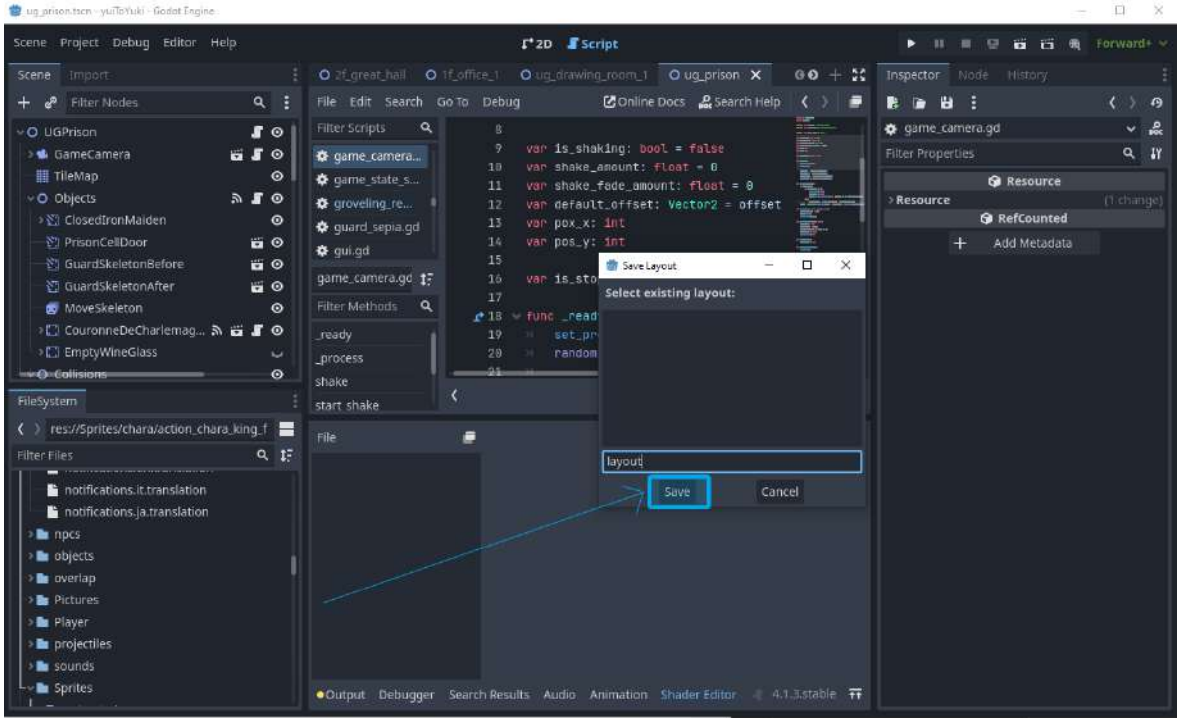
clicchiamo su **Save Layout**



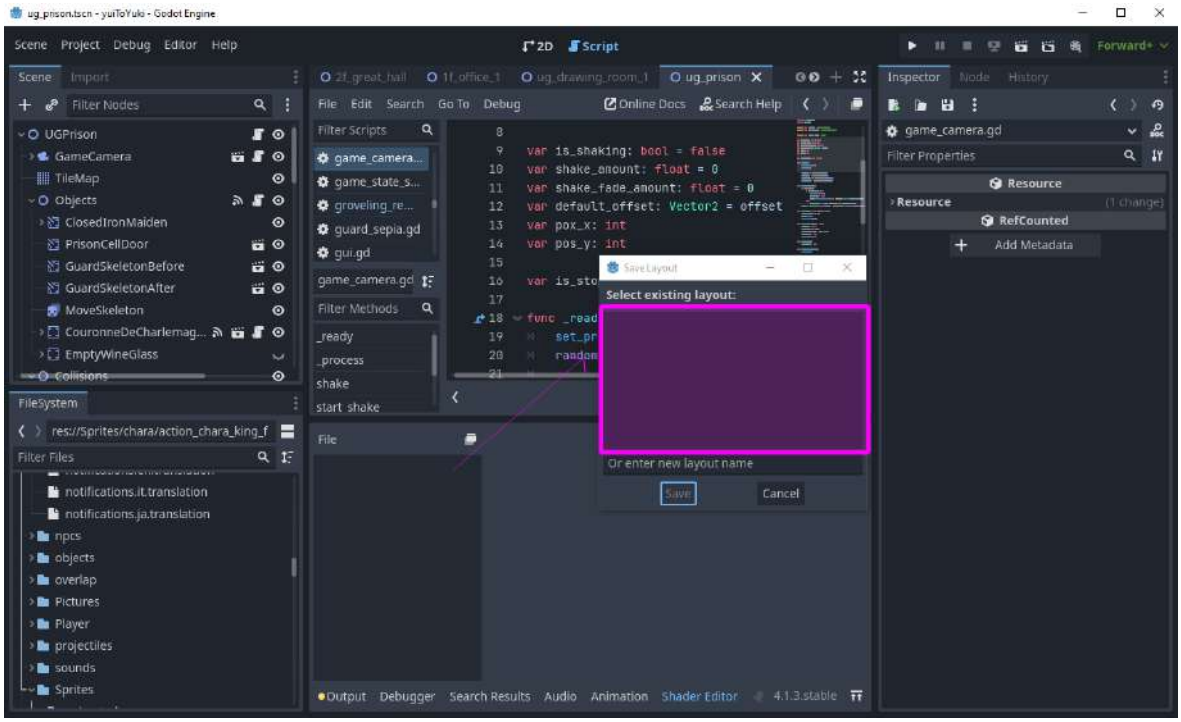
inseriamo un nome per il layout



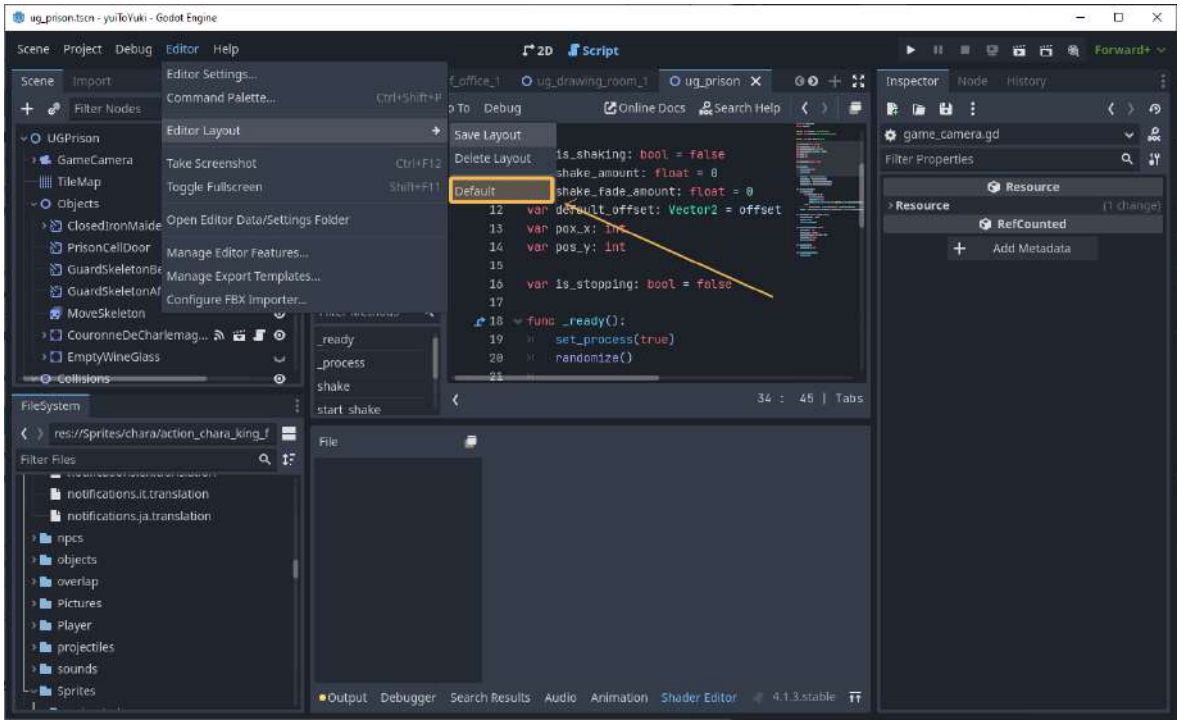
clicchiamo **Save**



Se si ha già salvato un editor layout, si può scegliere di effettuare l'override di un layout esistente utilizzando la lista.



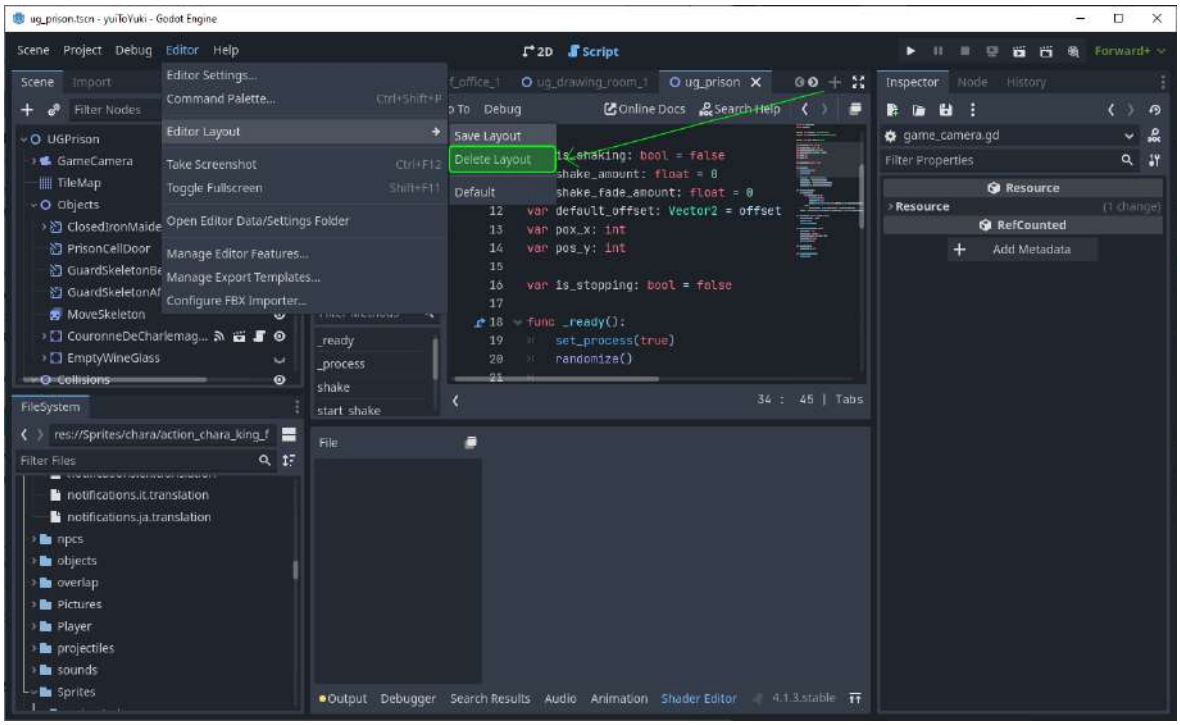
Dopo aver apportato le modifiche, aprendo l'**Editor** menu in cima all'editor e selezionando la voce **Editor Layouts** dalla lista dropdown, si noterà una lista di editor layout salvati, più **Default**, un editor layout hardcoded che non può essere rimosso.



Il layout **Default** corrisponde al layout che ha l'editor di una nuova installazione di Godot senza modifiche alla poszione e alle dimensioni dei dock, e senza floating dock.

Un editor layout può essere cancellato mediante l'apposita opzione **Delete** nell'**Editor**

Layouts dropdown



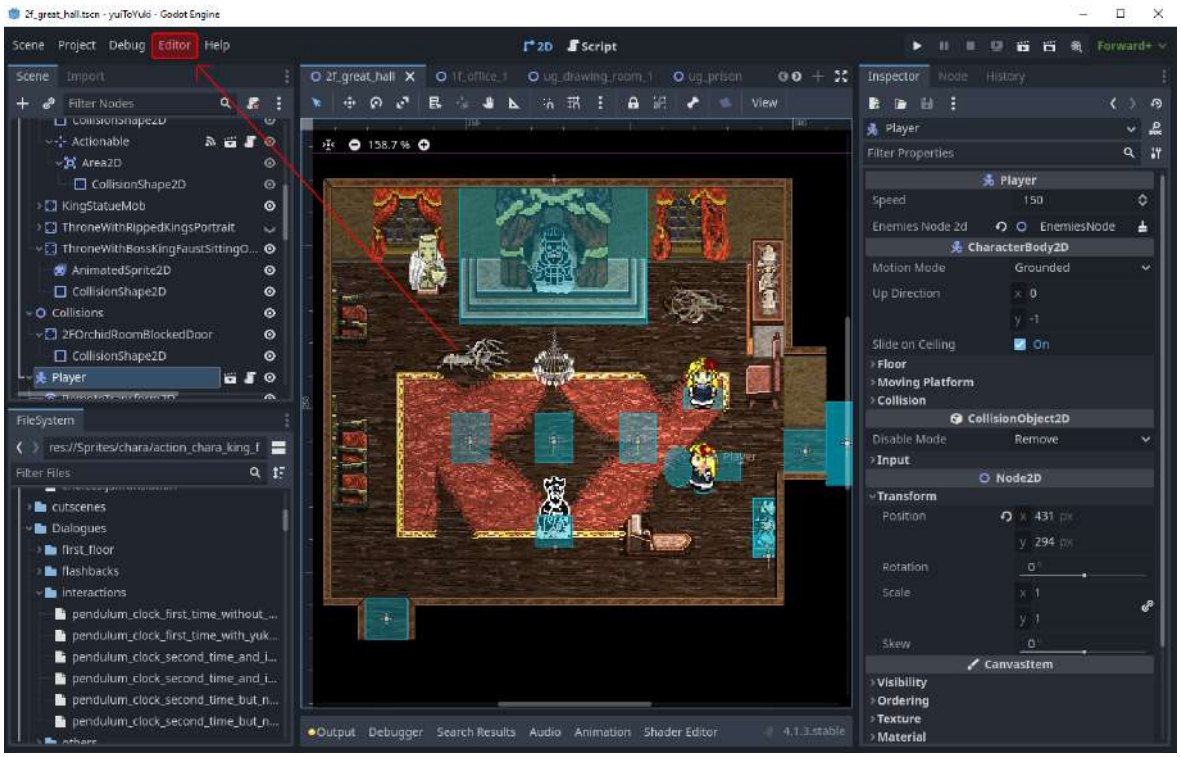
Consiglio

Se si assegna `Default` (case-sensitive) come nome a un editor layout personalizzato, l'editor layout di default verrà sovrascritto. Si noti che l'editor layout `Default` non compare nella lista dei layout da sovrascrivere finché non lo si sovrascrive una volta, ma è comunque possibile scriverne il nome manualmente. Si può tornare al'editor layout di default standard cancellando l'editor layout `Default` dopo averlo sovrascritto (questa opzione non compare se non si ha ancora sovrascritto l'editor layout di default)

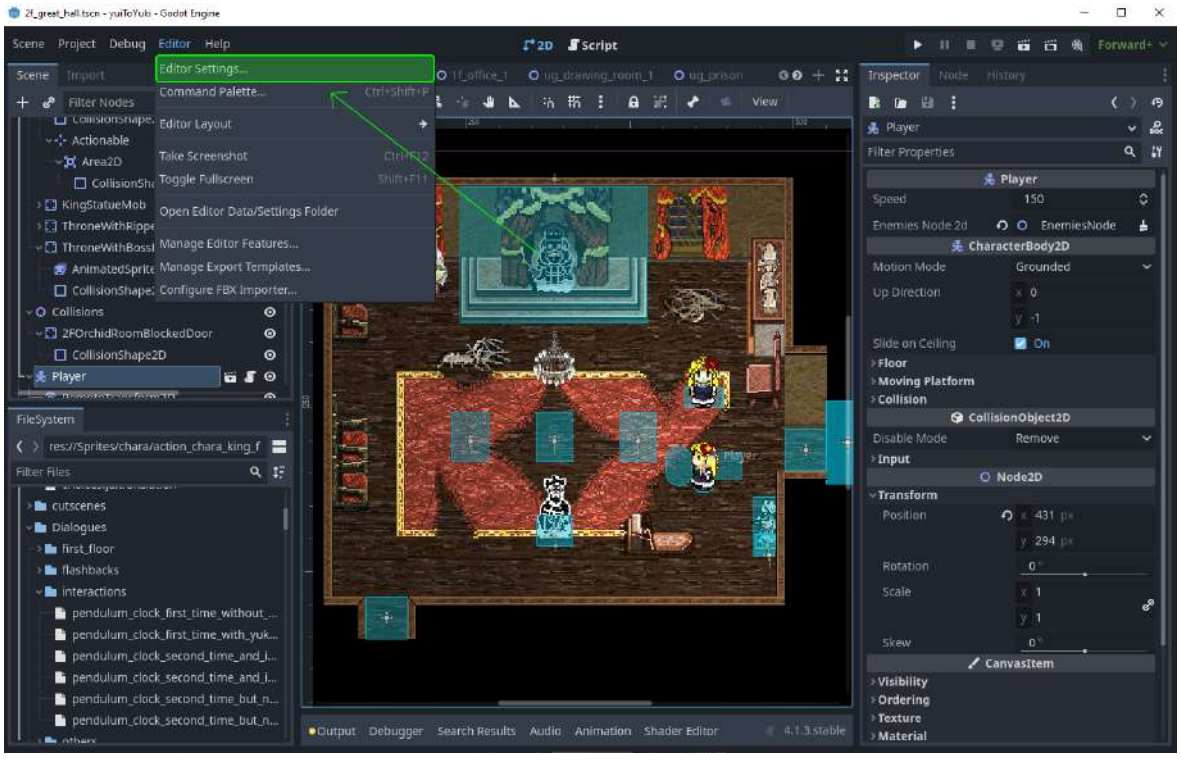
Gli editor layout vengono salvati in un file chiamato `editor_layouts.cfg` nel configuration path dell'**Editor data paths**.

2.2.4.4.4 Personalizzazione delle impostazioni dell'editor

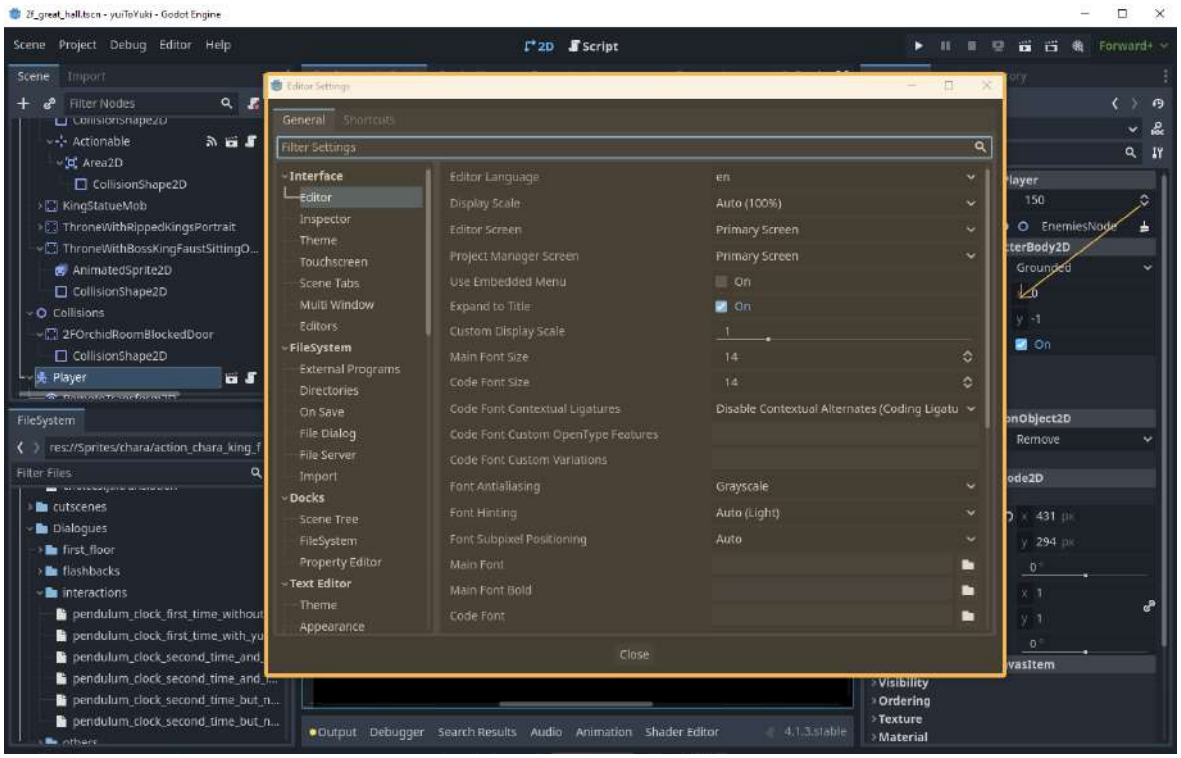
Nell'**Editor** menu, situato nella parte superiore dell'editor



si trova l'opzione **Editor Settings**



la quale apre una finestra simile alla finestra **Project Settings**, ma con le impostazioni utilizzate dall'editor.



Queste impostazioni sono condivise in tutti i progetti e non vengono salvate tra i file del progetto.

Alcune impostazione normalmente modificate sono:

- **Interface > Editor > Editor Language**: controlla la lingua di visualizzazione dell'editor. La lingua può essere cambiata anche nell'angolo in alto a destra del Project Manager

- **Interface > Editor > Display Scale**: controlla la grandezza degli elementi UI che vengono visualizzati a schermo. L'impostazione di default **Auto** trova un valore accettabile in base al DPI e alla risoluzione del proprio display. Il fattore di scaling fornito dal display viene preso solamente su macOS e non su Windows o Linux, a causa di limitazioni dell'engine.

- **Interface > Editor > Single Window Mode**: se attivo, forza l'editor a utilizzare una singola finestra. Questo disabilita alcune feature come il dividere l'editor di script/delle shaders in una propria finestra. La single-window mode può essere resa più stabile, specialmente in Linux se si sta utilizzando Wayland

- **Interface > Theme > Preset**: il theme preset dell'editor da utilizzare. Il **Light** theme preset può essere più facile da leggere se si è all'aperto o in una stanza illuminata. Il **Black (OLED)** theme preset può ridure il consumo di corrente sui display OLED, che sono sempre più comuni nei computer portatili e nei telefoni/tablet

- **FileSystem > Directories > Autoscan Project Path**: può essere impostato a un path di una cartella che sarà automaticamente scansionata in cerca di progetti dal project manager, ogni volta che viene avviato

- **FileSystem > Directories > Default Project Path**: controlla la posizione di default in cui vengono creati i nuovi progetti creati dal Project Manager

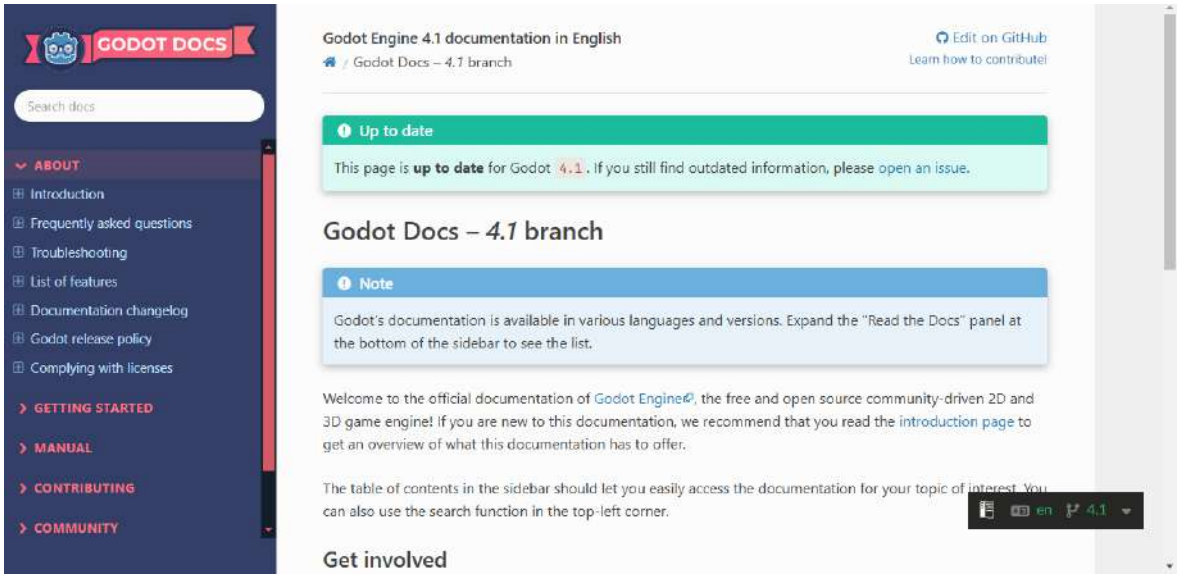
Si può inoltre andare di hover con il mouse sul nome di una impostazione dell'editor, nella finestra **Editor Settings**, per mostrarne la descrizione.

2.2.5 Come imparare nuove feature

Godot è un game engine feature-rich. C'è molto da imparare. In questa sezione illustreremo come unirsi alle community online e come utilizzare il manuale online e/o le reference di codice built-in per imparare nuove feature e tecniche.

2.2.5.1 Come utilizzare al meglio il manuale online

Il manuale online documenta tutti i concetti e le feature disponibili dell'engine. Quando si vuole imparare un nuovo argomento, si può iniziare dando uno sguardo alla sezione corrispondente del sito web ufficiale[8].

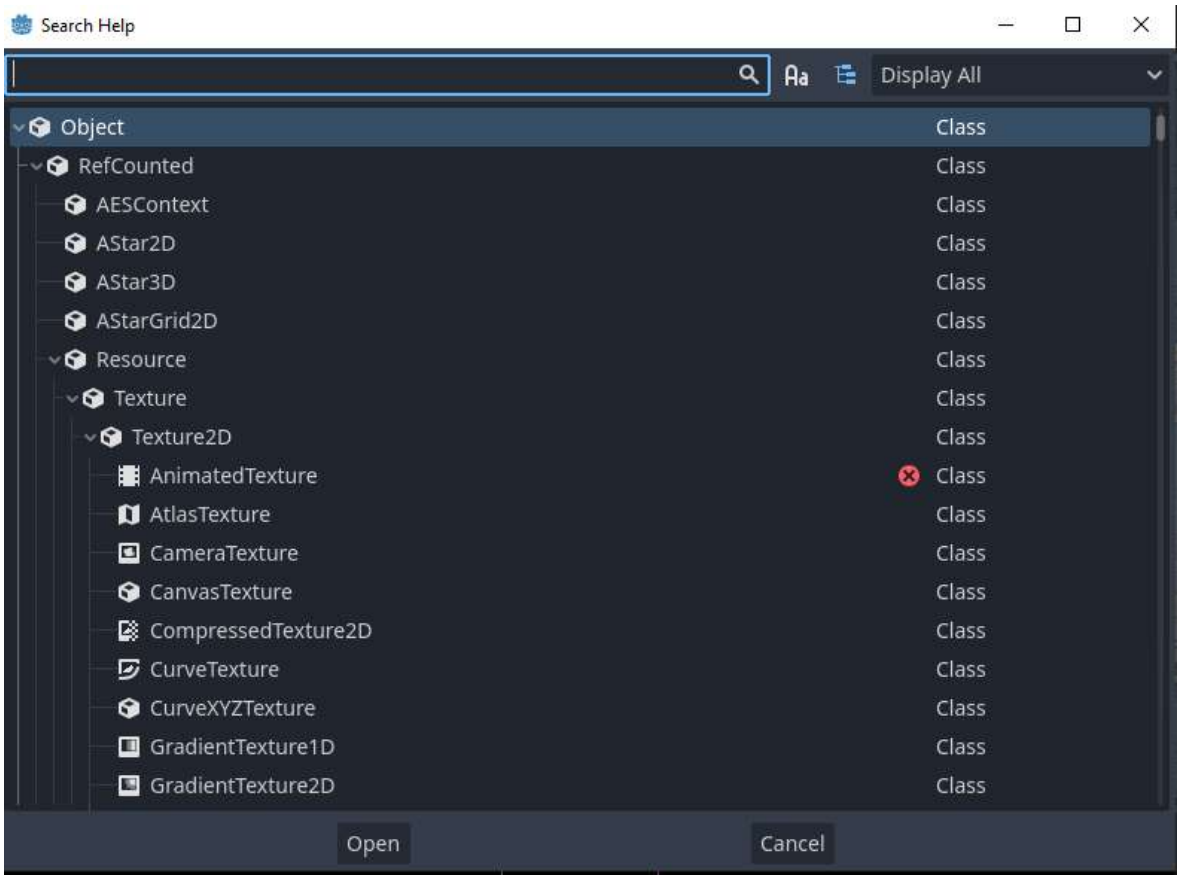


Il menu sulla sinistra permette di esplorare svariati argomenti, mentre la barra di ricerca può aiutare a trovare pagine più specifiche. Se una pagina di un certo tema esiste, spesso questa sarà linkata ad altri argomenti correlati.

Il manuale ha un partner class reference che spiega tutte le funzioni e le proprietà delle classi di Godot intanto che si sta programmando. Mentre il manuale copre come utilizzare l'editor, le feature e i concetti generali, la reference riguarda solamente l'utilizzo dell'API di scripting di Godot, alla quale si può accedere sia online che offline. Si raccomanda la navigazione della reference in modalità offline, dall'interno dell'editor

[8] `https://docs.godotengine.org/en/4.1/`

di Godot, mediante una delle azione viste in precedenza, come ad esempio andare su "Help" -> "Search Help" oppure premendo il tasto `F1`.



Per visualizzarla online si può andare invece nella sezione "Class Reference"[9] del manuale.

Una pagina della class reference presenta le seguenti informazioni:

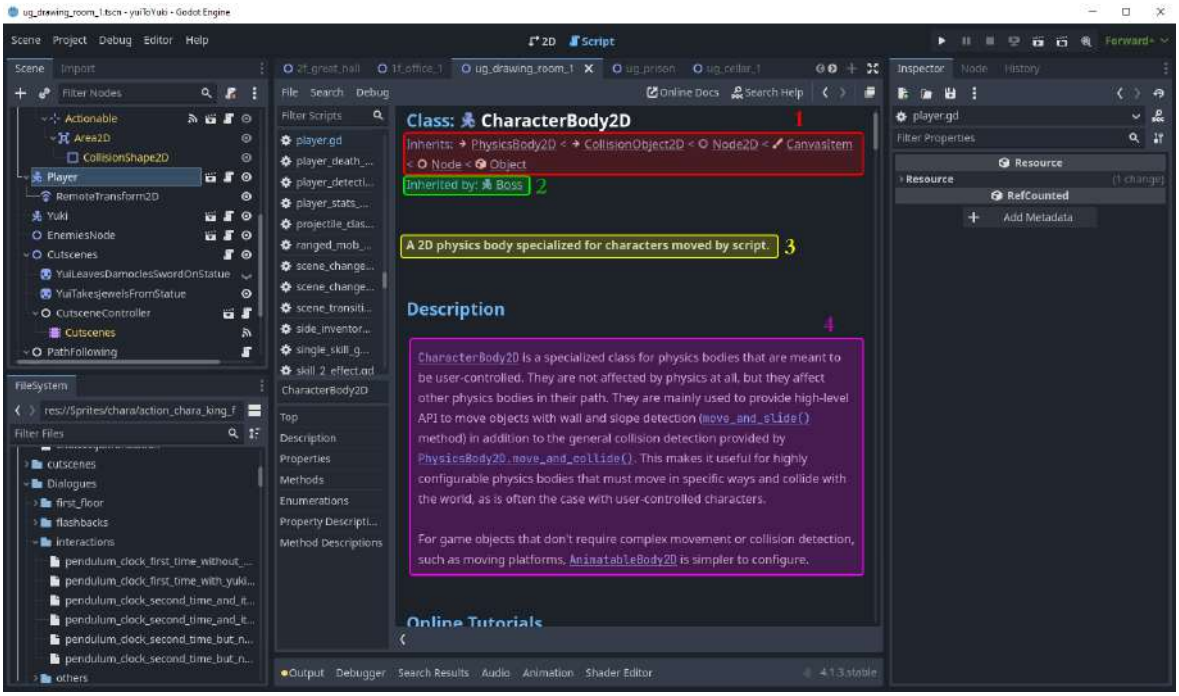
1. Dove si trova una classe all'interno della gerarchia ereditaria. I link in cima possono essere cliccati per aprire la pagina delle classi parent e guardare le proprietà e i metodi ereditati da un tipo.

2. Le classi che ereditano dalla classe, anch'esse cliccabili

3. Un riepilogo sul ruolo della classe e i suoi casi d'uso

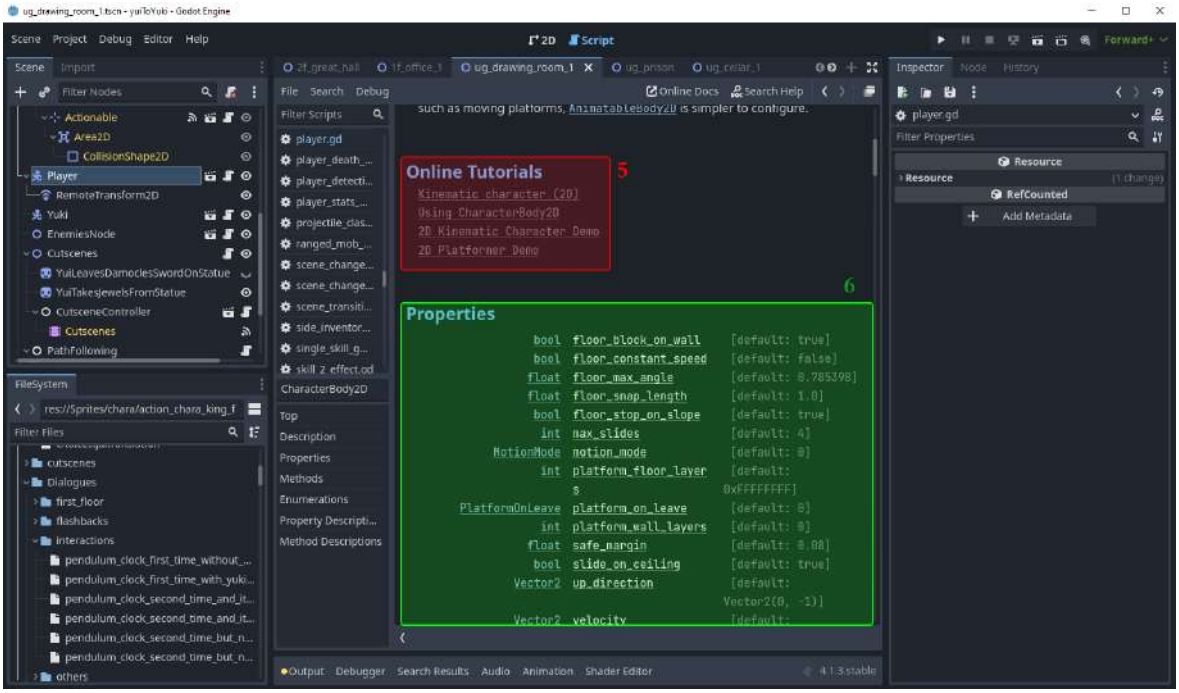
4. Una spiegazione sulle proprietà, i metodi, i segnali, gli enum e le costanti della classe

[9] `https://docs.godotengine.org/en/4.1/classes/index.html#doc-class-reference`



5. Link alla pagina del manuale che descrive più nel dettaglio la classe

6. Lista di proprietà, segnali e metodi della classe



Tutte le parole o le frasi sottolineate, come ad esempio il nome, una proprietà, un metodo, un segnale o una costante di una classe, possono essere inoltre cliccate per visualizzare ulteriori dettagli e spiegazioni.

2.2.5.2 Imparare con la community

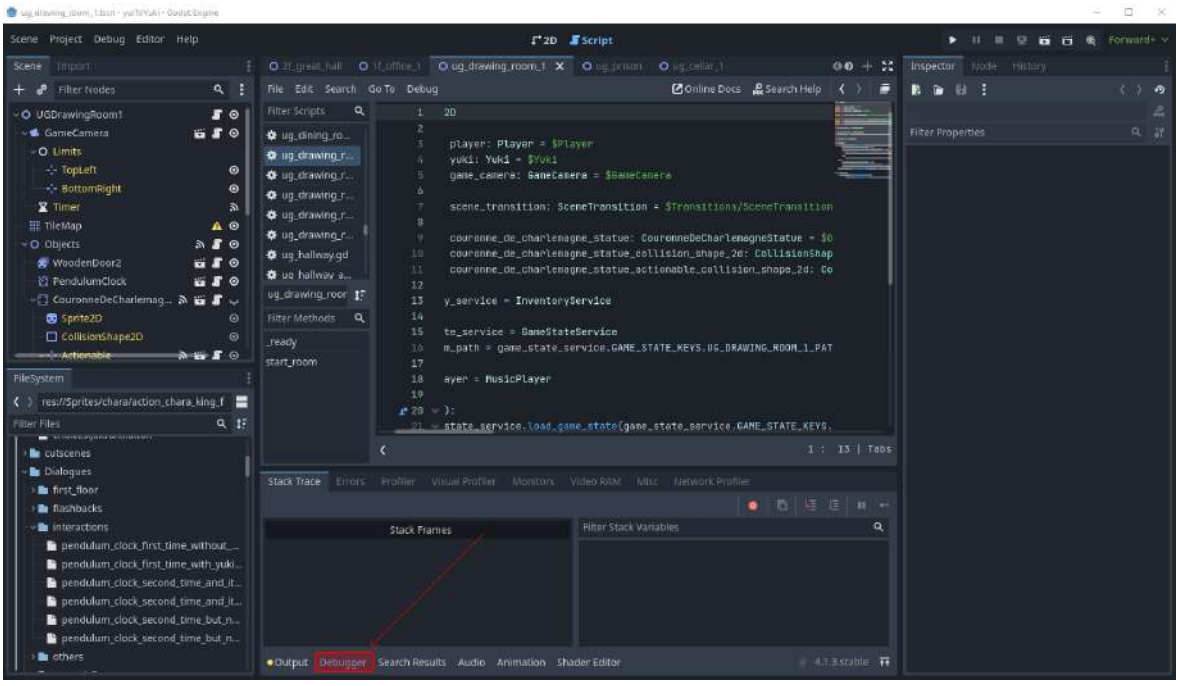
La community di Godot è in continua crescita. Se si è incappati in un problema oppure si ha bisogno di aiuto per comprendere meglio come raggiungere un qualcosa, si può chiedere agli altri utenti di una delle tante community attive[10].

Il posto migliore per fare domande e trovarne alcune già risposte è il sito ufficiale di "Questions & Answers"[11]. Queste risposte appaiono nei risultati di search engine come google e possono essere salvate, permettendo così agli utenti di trarre quanto più beneficio possibile dalle discussioni sulla piattaforma. Una volta aver posto una domanda, si può inoltre condividere il link su altre piattaforme social. Prima di fare una domanda, è consigliabile cercare risposte già esistenti che potrebbero aver già risolto il problema.

Fare domande nel modo corretto e fornire dettagli aggiuntivi può aiutare gli altri utenti a rispondere meglio e più velocemente. Consigliamo di includere le seguenti informazioni quando si vogliono fare domande:

- **Descrivere il proprio obiettivo**. È consigliato spiegare che cosa si vuole raggiungere dal punto di vista del design. Se si stanno avendo problemi nel capire come far funzionare una propria soluzione a un problema, potrebbero esservene di più semplici e diverse in grado di raggiungere lo stesso risultato

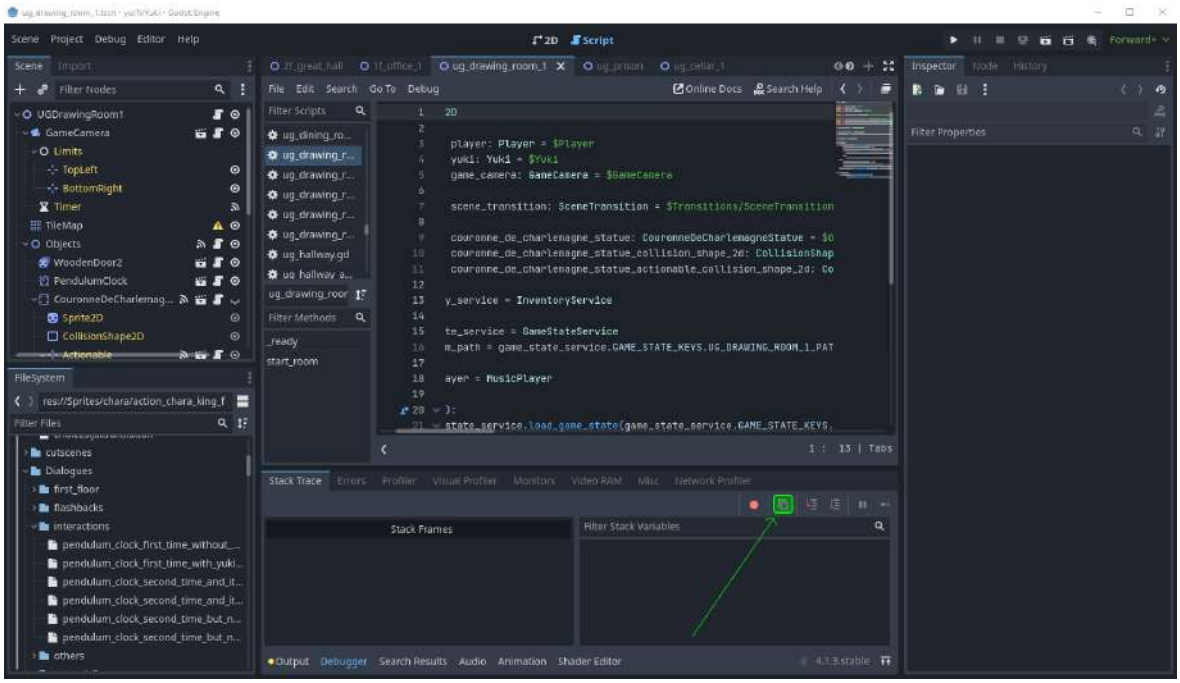
- Se si verifica un errore, bisogna **condividere il messaggio di errore esatto**, il quale può essere copiato direttamente dal debugger nel bottom panel dell'editor



cliccando sull'icona "Copy Error".

Sapere cosa dice può aiutare i membri della community a identificare meglio le ragioni che lo hanno scatenato

[10] `https://godotengine.org/community/`
[11] `https://ask.godotengine.org/`



- Se c'è del codice di mezzo bisogna **condividerne anche solo una parte**. Gli altri utenti non saranno in grado di aiutare a risolvere un problema senza guardare il codice. Per condividere il codice come testo basta copiare e incollare un code snippet breve in una chat box o utilizzare siti web come Pastebin[12] per condividere lunghi file

- **Condividere uno screenshot** (e non una foto fatta con il cellulare che molto probabilmente ha del riflesso ed è di bassa qualità) del proprio dock *Scene* insieme al codice scritto. La maggior parte del codice che si scrive influenza i nodi delle scene. Si dovrebbe dunque pensare a queste scene come a una parte del proprio codice sorgente.

- Condividere un video dell'esecuzione del videogioco può essere **utile per effettuare troubleshooting**. Per registrare lo schermo del computer si possono utilizzare programmi come "OBS Studio"[13] e "Screen to GIF"[14], mentre servizi cloud come streamable[15] per effettuare l'upload e la condivisione gratuita dei video

- Citare la versione che si sta utilizzando in caso non sia una versione stable di Godot. Le risposte a un problema potrebbero essere diverse a seconda della versione e, inoltre, le feature disponibili e l'interfaccia evolvono rapitamente

Seguire queste linee guida non solo aumenterà le chance di ricevere la risposta che si stava cercando, ma farà anche risparmiare tempo a noi e a chi ci sta aiutando.

[12] `https://pastebin.com/`
[13] `https://obsproject.com/`
[14] `https://www.screentogif.com/`
[15] `https://streamable.com/`