JOGOS DIGITAIS
Introdução Godot **Prof. Diego**

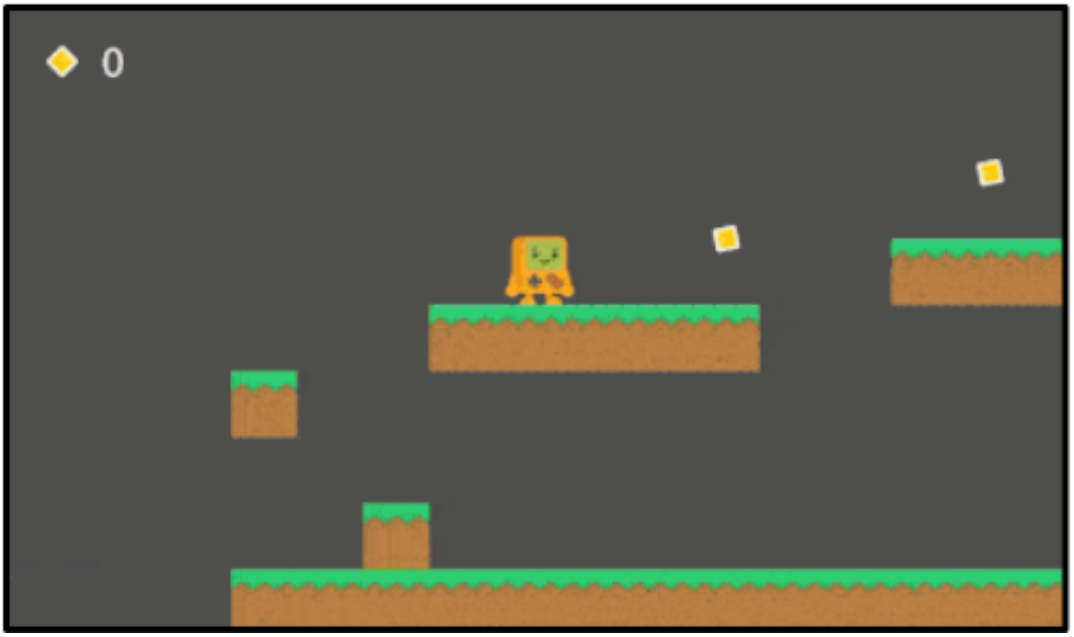
Como fazer seu primeiro jogo 3D com Godot

Introdução

Você está ansioso para começar a fazer seus próprios jogos?

O desenvolvimento de jogos nunca foi tão popular antes - com vendas chegando aos bilhões e milhares de desenvolvedores independentes aprimorando suas habilidades e dando vida às suas criações. Mesmo que você seja um iniciante que nunca codificou antes, com inúmeros mecanismos disponíveis, praticamente qualquer pessoa pode criar e programar seu projeto de jogo dos sonhos. Neste tutorial abrangente, aprendamos a criar seu primeiro jogo usando Godot, um motor de jogo gratuito e de código aberto que permite criar jogos 2D e 3D.

Devido à sua natureza de código aberto, o que significa que os usuários podem adicionar e remover coisas do motor em seu lazer, ele rapidamente vem ganhando grande popularidade. Com uma comunidade vibrante pronta para ajudar, é uma escolha perfeita para criar seu primeiro jogo. Depois de instalar Godot e aprender o básico do editor, vamos criar um jogo de plataforma 2D, então prepare-se para começar a desenvolver jogos!

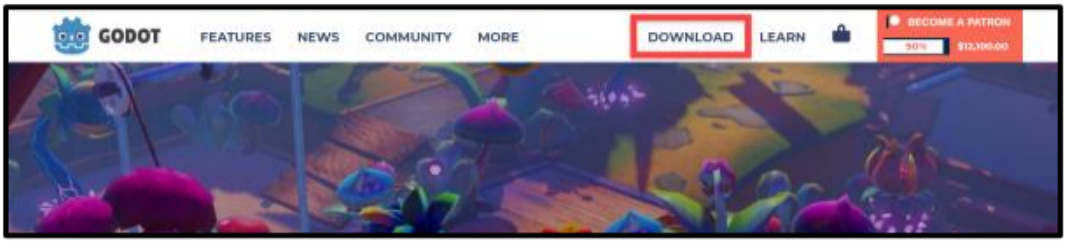


Arquivos

Para este projeto, vamos precisar de alguns ativos para o jogador, peças, moedas, inimigo, etc. Você pode criar o seu próprio ou usar os apresentados neste tutorial. Os ativos podem ser baixados aqui. Assets

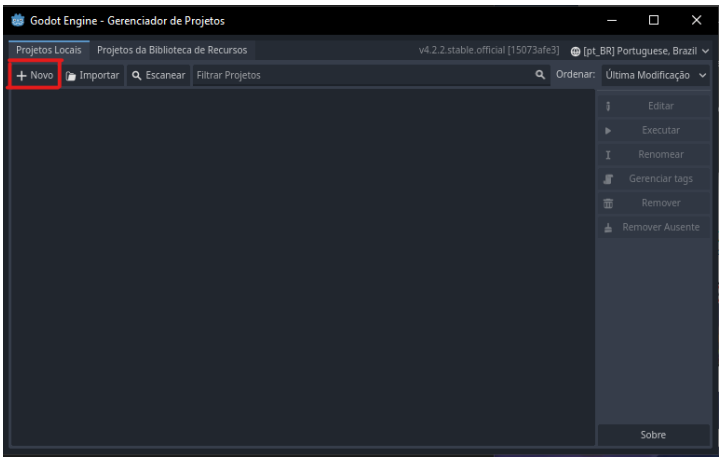
Instalando o Godot

Para baixar e instalar Godot, vamos ao `https://godotengine.org/`. Aqui, você pode ver os recursos do Godot, páginas da comunidade e muito mais. Queremos clicar no botão Download.



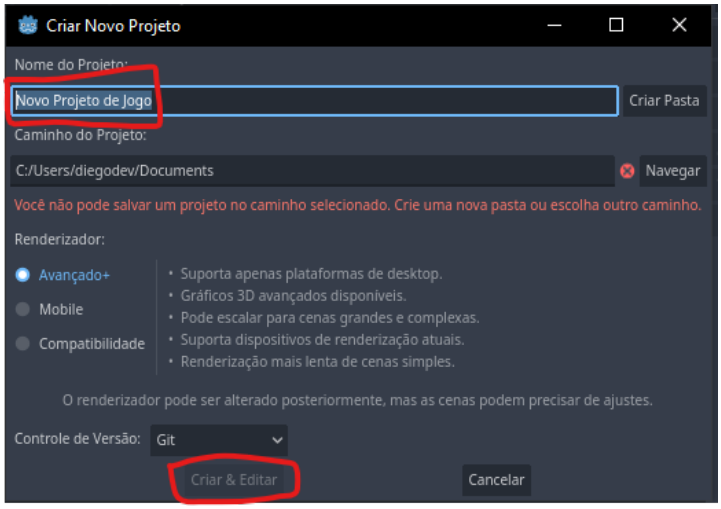
Criando um novo projeto

Abra o aplicativo Godot que baixamos para ver o **Gerente de Projeto**. Aqui, podemos criar projetos, visualizar outros e baixar modelos. Clique no botão **Novo projeto** para criar um novo projeto.



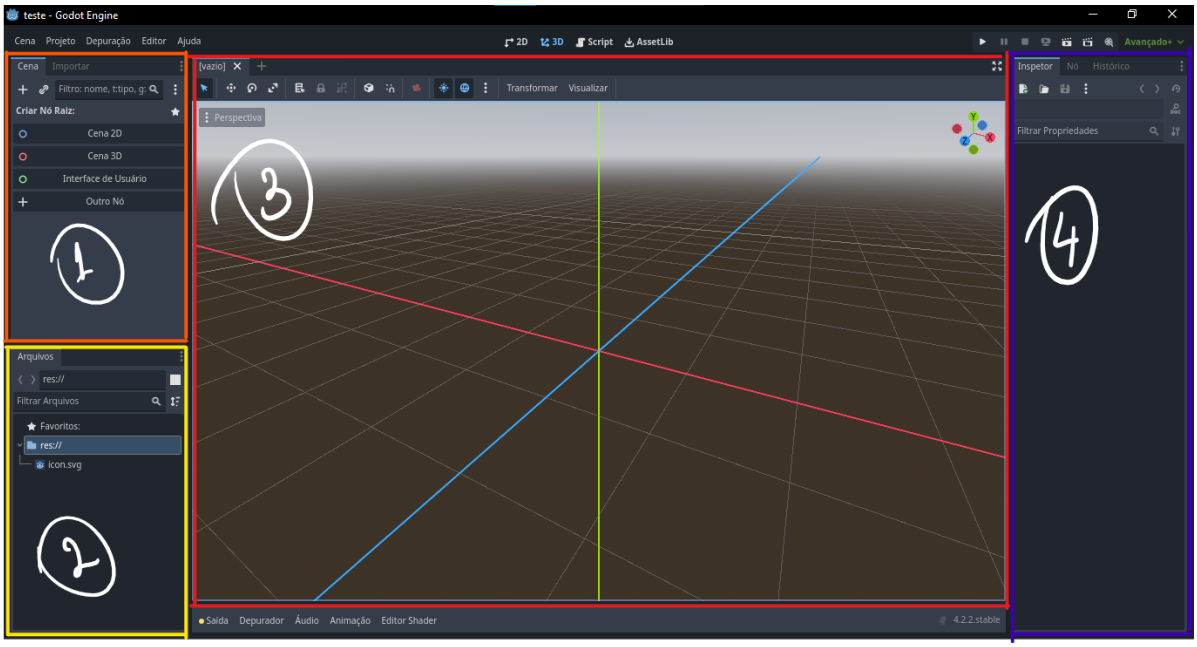
Isso abrirá uma nova janela.

- 1. Digite um nome para o seu projeto;
- 2. Clique no botão **Criar pasta** para criar uma nova pasta para o projeto na pasta documentos;
- 3. Clique no botão **Criar e Editar** para iniciar o mecanismo e começar a criar o jogo.



Explorando o Editor

Quando o editor aparece, pode parecer bastante assustador com todos os botões e opções, mas vamos dividi-lo.



Godot tem 4 painéis principais que usaremos para criar nosso jogo e cada um serve a um propósito específico.

- 1. Este aqui é o painel Cena. Ele exibirá o layout do nó e a hierarquia da cena em que estamos atualmente. Veremos o que é uma cena e um nó muito em breve.
- 2. O painel FileSystem nos mostra todos os ativos e arquivos que temos. Sprites, modelos, scripts, cenas, pastas, áudio, etc.
- 3. É aqui que podemos ver e criar o nosso jogo. Movendo coisas, selecionando, roteirizando etc. Acima deste painel estão quatro botões e eles alternam o que o painel se torna. Podemos alternar entre os modos 2D e 3D, o editor de scripts e a biblioteca de ativos externos.
- 4. Este é o Inspetor e isso nos mostra os detalhes de um nó quando selecionamos um. A posição, rotação e quaisquer outros atributos que possamos modificar

Como o Godot funciona?

Acima estávamos falando de cenas e nós. Quais são eles? Bem, um jogo em Godot é composto por uma hierarquia de nós. Um nó pode ser qualquer coisa: um player, câmera, modelo 3D, luz, interface do usuário, etc. Os nós compõem todas as entidades do seu jogo e também têm a capacidade de ser filhos de outro nó. Aqui está um exemplo de um jogador em um jogo. Primeiro, temos o nó KinematicBody que pode controlar o movimento e algumas interações físicas. Então, como um filho disso, temos um colisor e um nó de sprite. Quando o nó do corpo cinemático se move, gira, etc, os outros se seguirão. Mesmo se excluirmos o nó pai, os filhos seguirão.

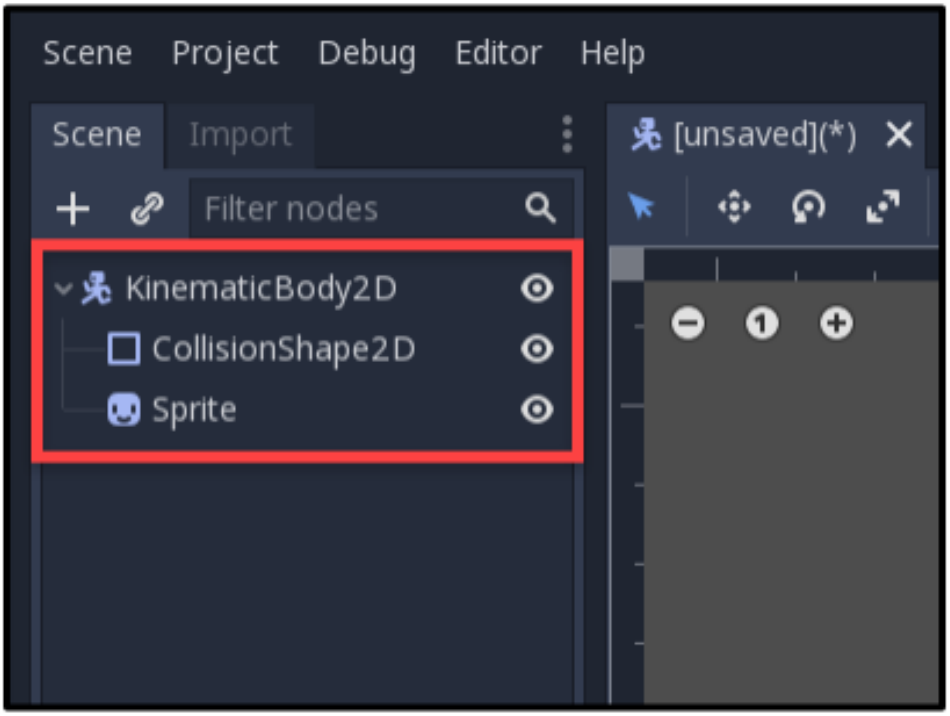
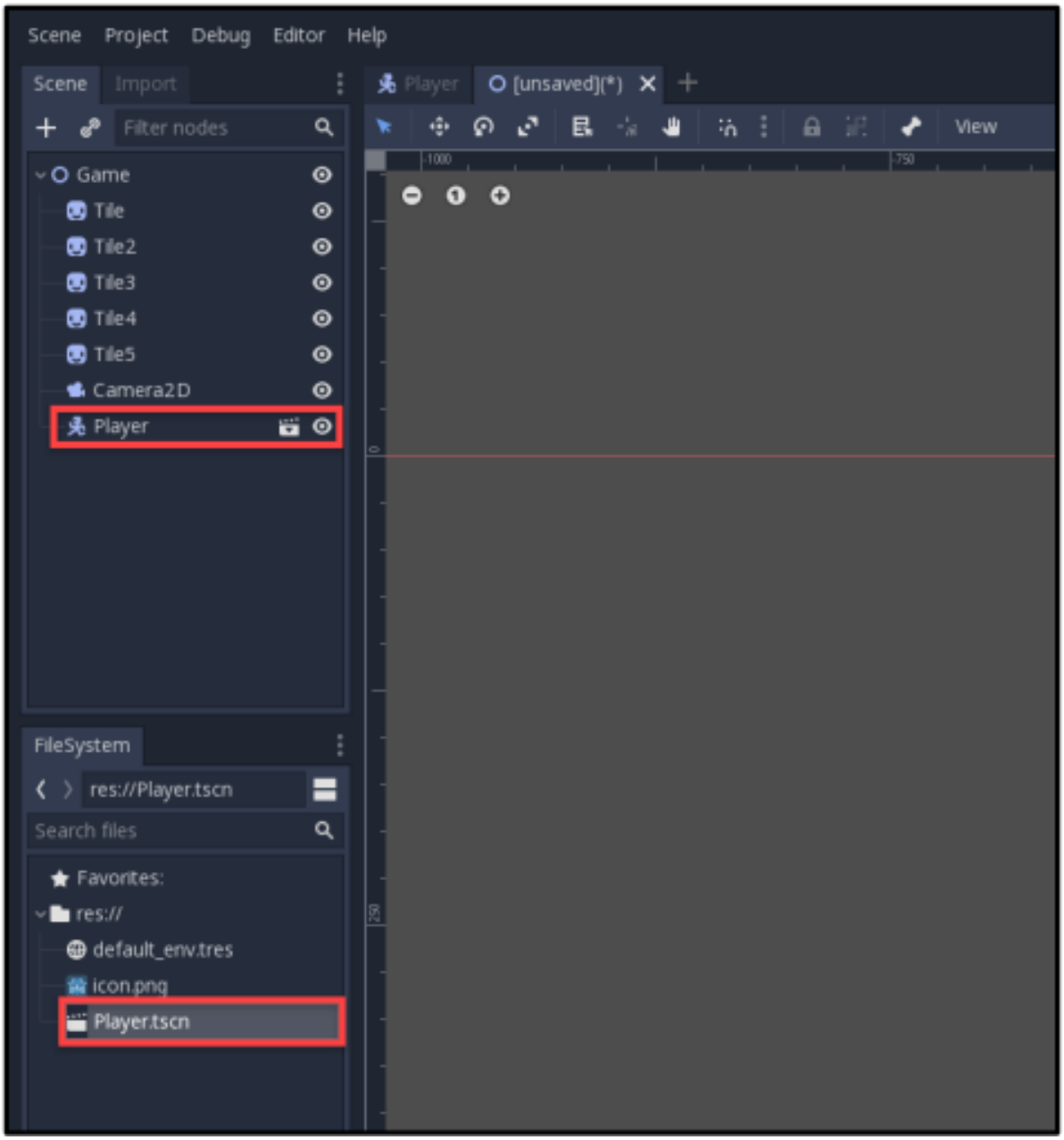


Figura 1: Enter Caption

Jogos em Godot são compostos de muitos nós pais-filhos para criar os vários elementos e sistemas diferentes de um jogo. Como um jogo é composto de nós, ele acabará chegando a um ponto em que há centenas ou até milhares deles no painel da árvore de cena. Isso tornará difícil encontrar certos e, no geral, tornará o trabalho no jogo confuso. Para resolver esse problema, podemos dividir nossos nós em cenas. As cenas são pacotes de nós independentes que podemos então soltar em outras cenas como nós. Vamos pegar nosso exemplo de jogador e transformar essa hierarquia de nós em uma cena. É um arquivo salvo no FileSystem que podemos então arrastar para outra cena.



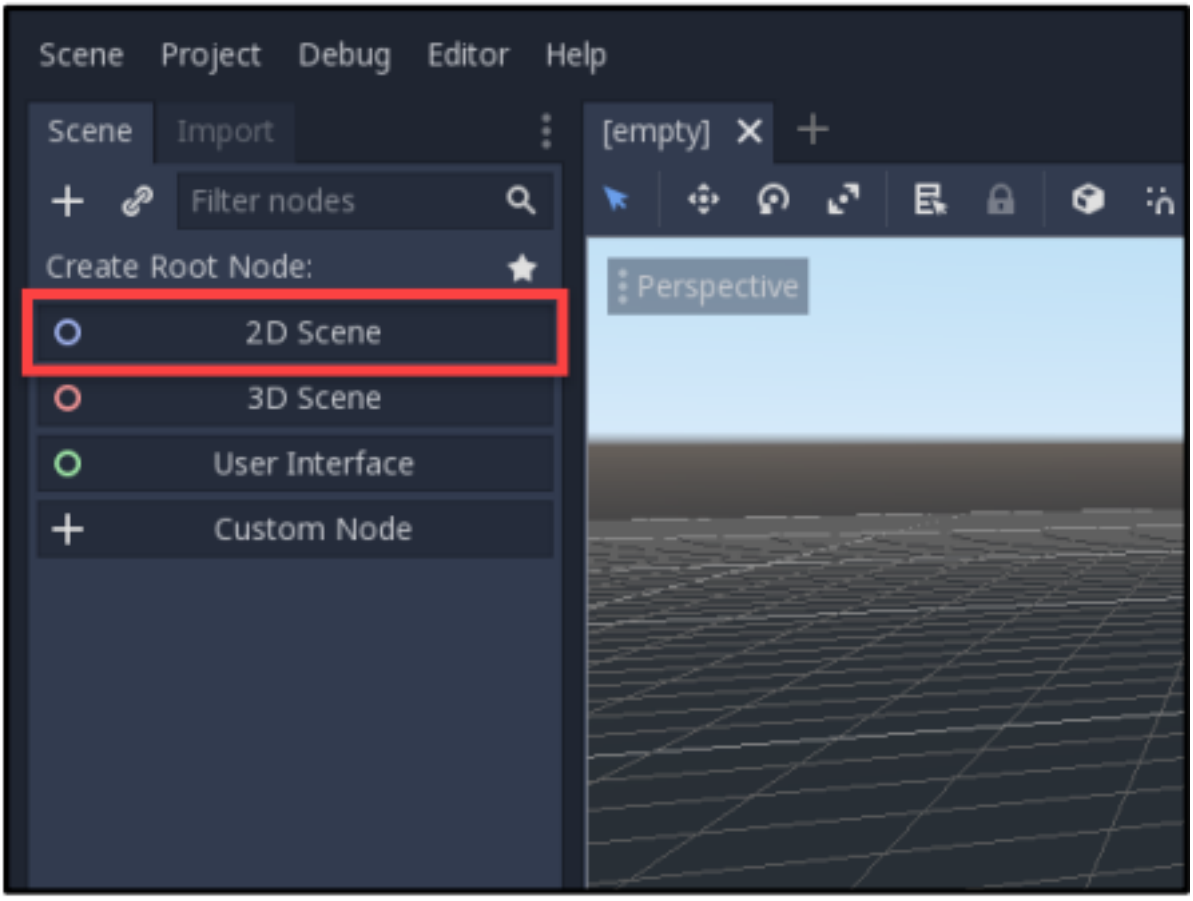
Um benefício de dividir seu jogo em nós também é o fato de que podemos remover a necessidade de repetir estruturas de nós comuns. Em vez de ter 100 nós de bloco que todos têm um sprite, colisor, etc todos na mesma cena, podemos apenas criar um desses como uma cena e arrastar em várias instâncias.

Exploraremos nós e cenas ao longo deste projeto.

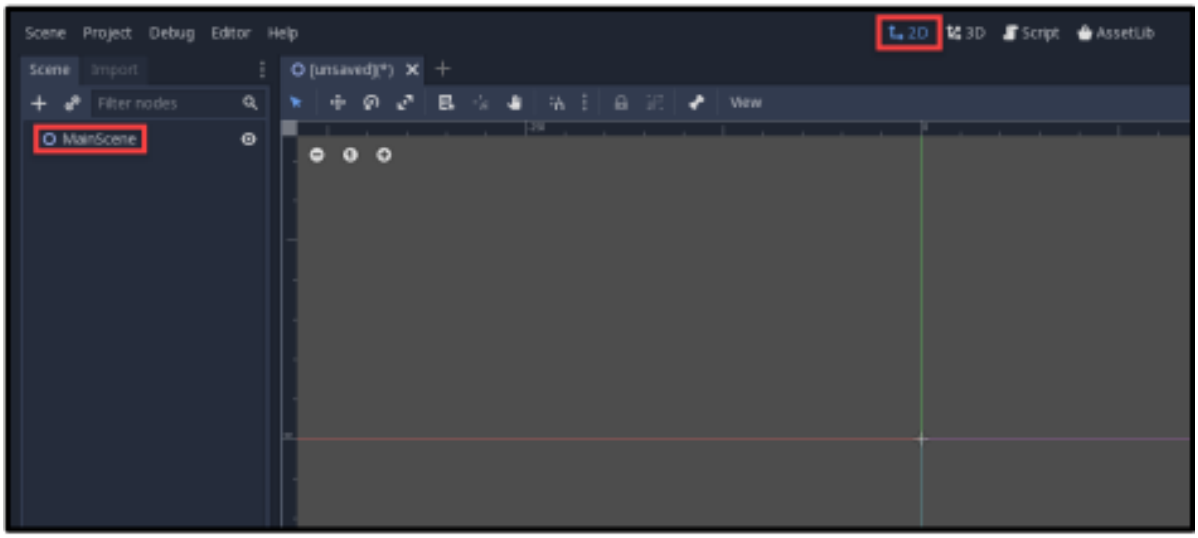
Criando nossa Primeira Cena

Tudo bem, vamos começar nosso projeto de plataforma 2D. Primeiro, precisaremos criar uma cena principal que será a base para o nosso jogo, contendo outras cenas, como o jogador, peças, moedas e inimigos.

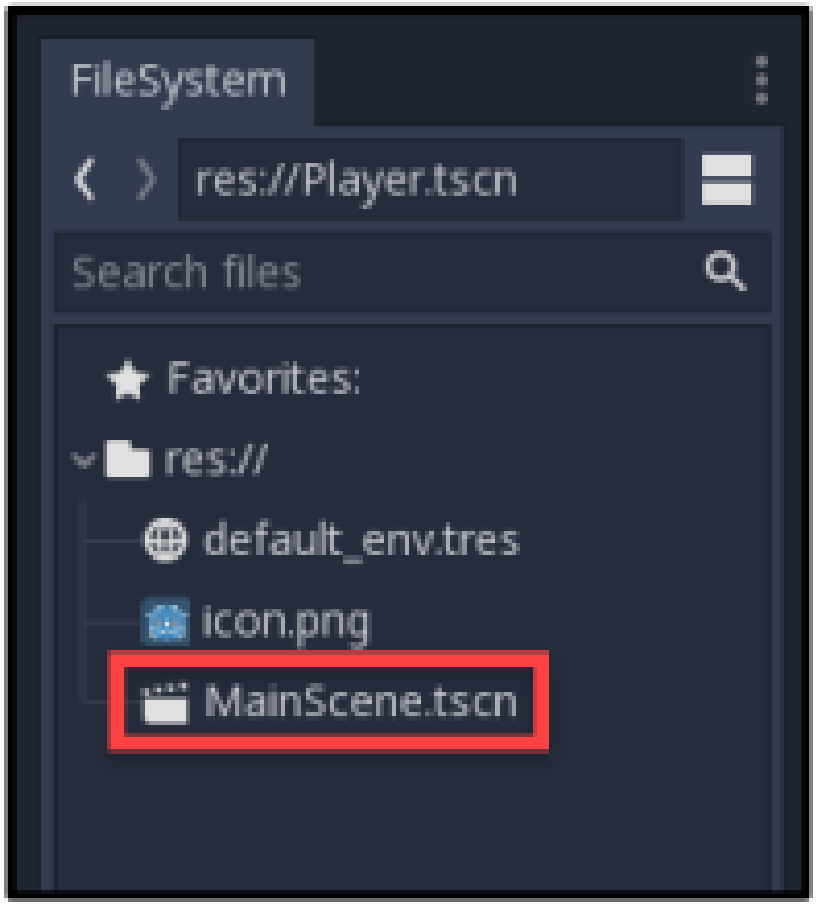
No painel de cena, clique no botão **2D Scene** para criar uma nova cena 2D.



Com nosso novo nó, vamos clicar duas vezes nele e renomeá-lo para **MainScene**.



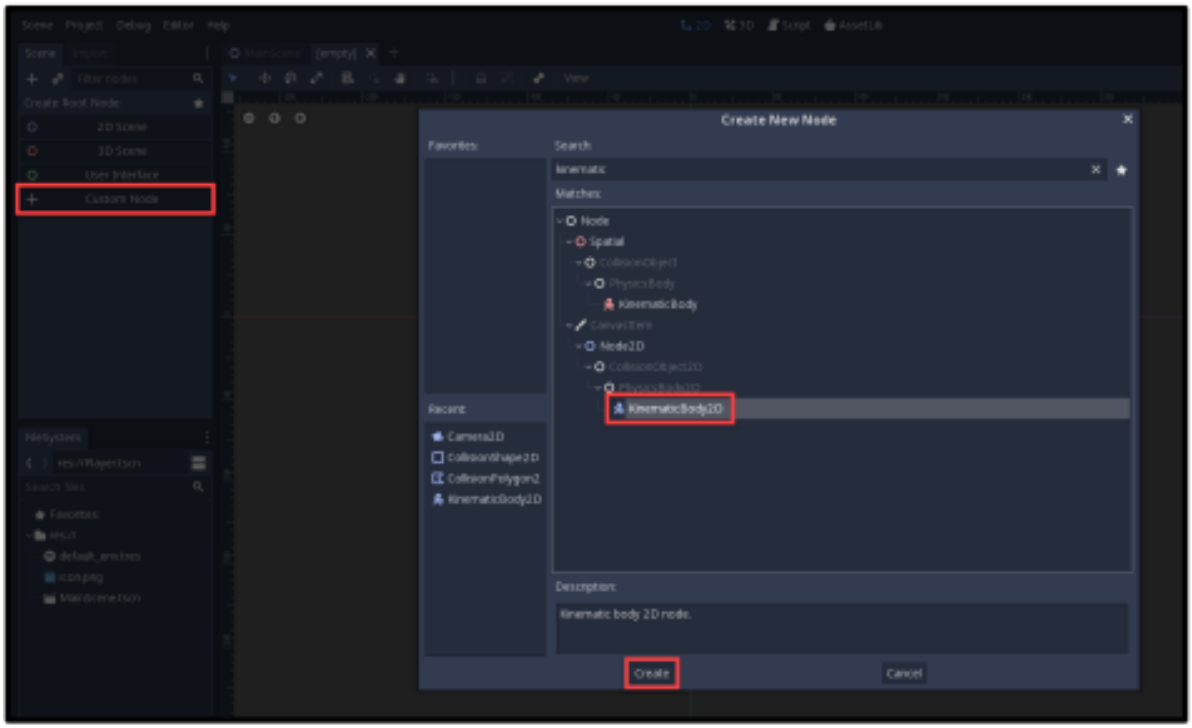
Também é bom salvar suas cenas ao criá-las. Salve com **Ctrl** + **S** ou Cena > Salvar cena. Pressione enter e você deve vê-lo no sistema de arquivos.



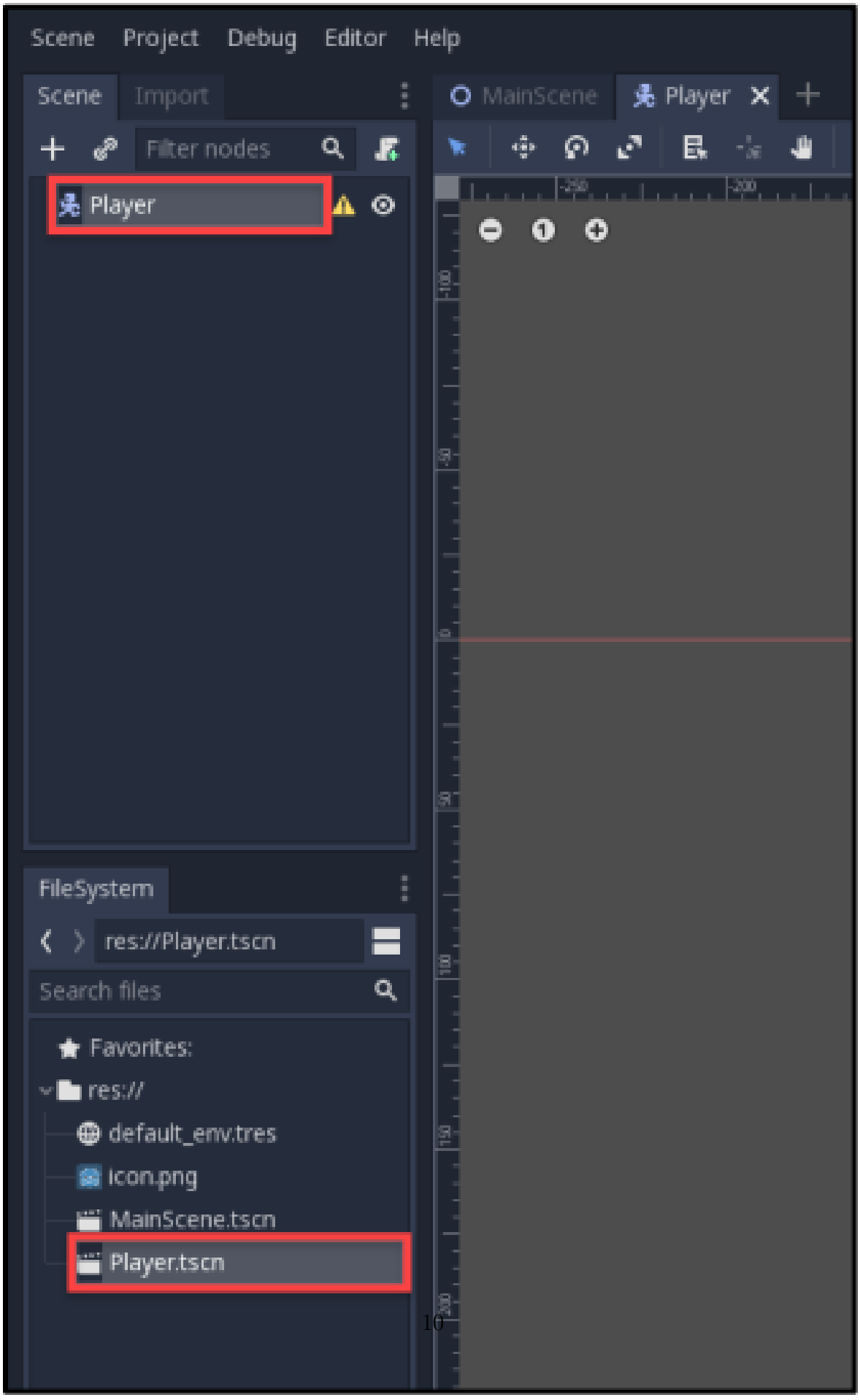
Criando o Jogador

Conseguimos nossa cena principal. Em seguida, criamos nossa cena de jogador que irá conter todos os nós que precisamos e o script correspondente.

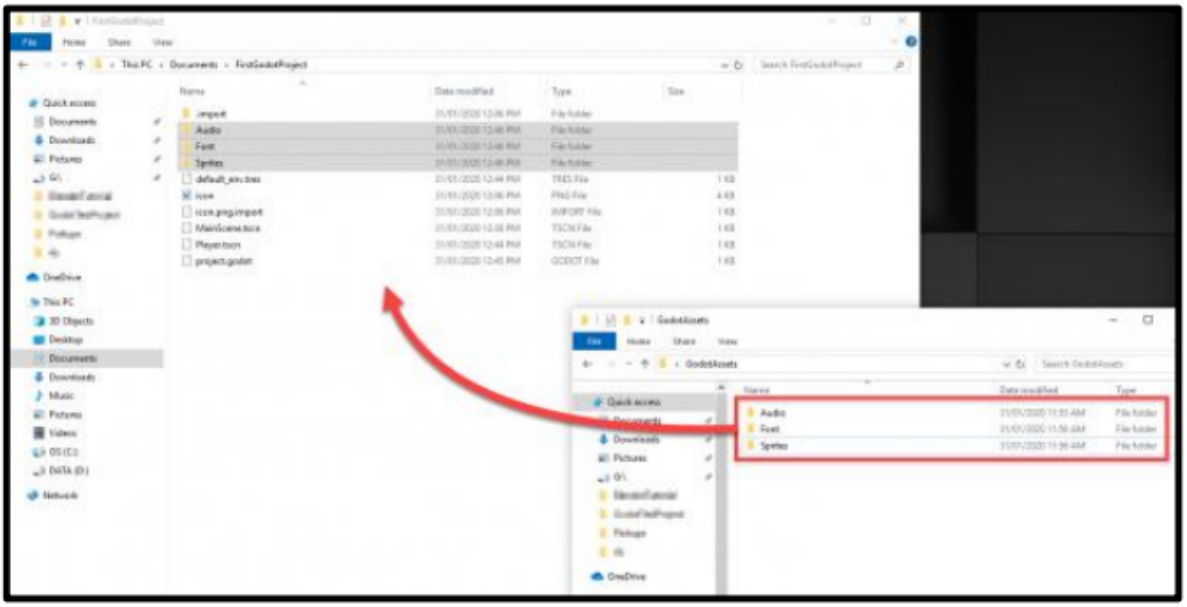
Para criar uma nova cena, podemos ir Cena > Nova Cena. No painel de cena, selecione **Outro Nó**. Isso abrirá uma janela e queremos criar o nó **KinematicBody2D**.



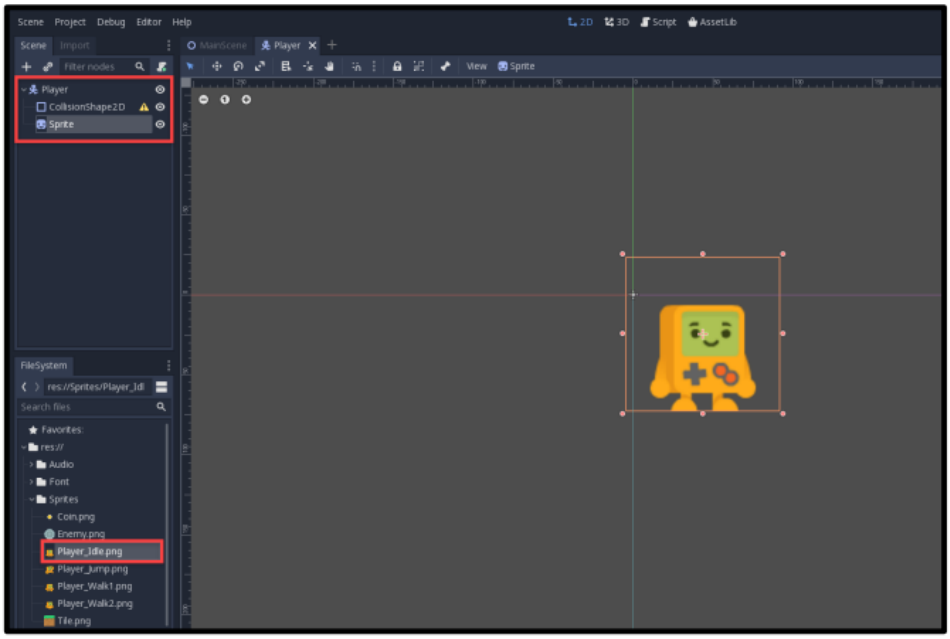
Renomeie o nó para **Player** e salve-o.


Scene
Project
Debug
Editor
Help
Scene
Import
Filter nodes
Player
MainScene
Player
FileSystem
res://Player.tscn
Search files
Favorites:
res://
default_env.tres
icon.png
MainScene.tscn
Player.tscn

Com nosso jogo de plataforma, precisaremos de alguns sprites e outros ativos. Incluído com este tutorial, é um arquivo .ZIP contendo todos os ativos que precisaremos. Faça o download e arraste as três pastas para a nossa pasta do projeto Godot.

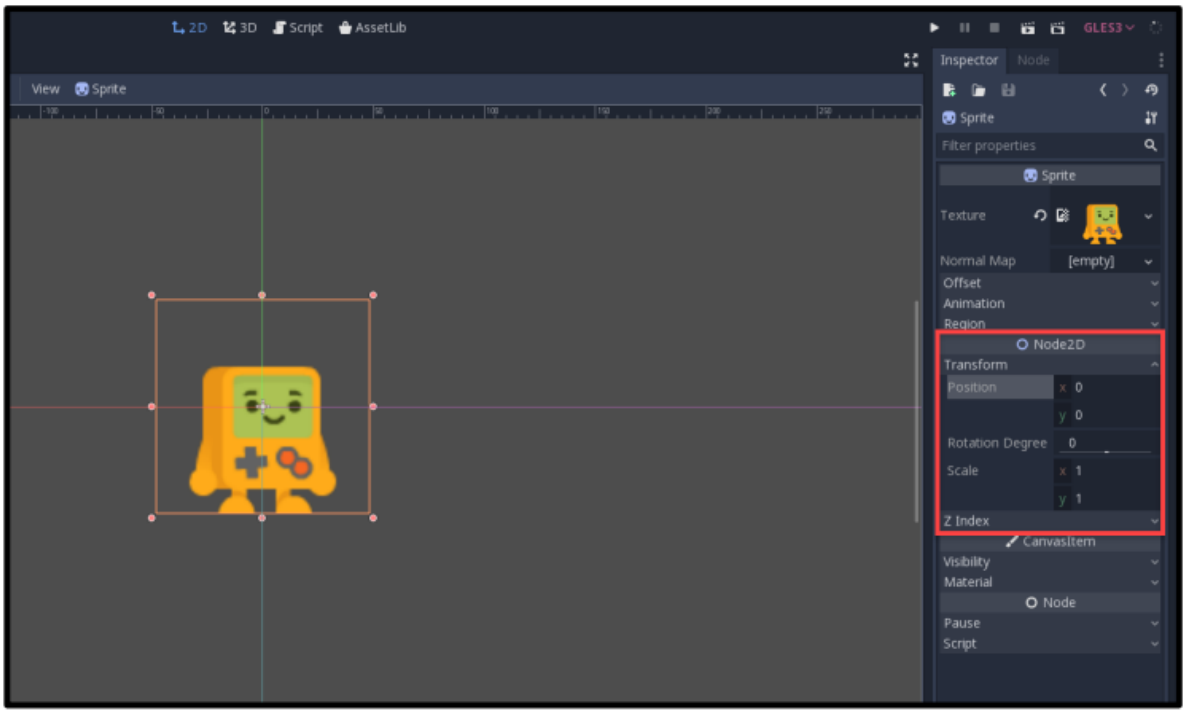


De volta a Godot, vamos clicar com o botão direito do mouse no nó Player e selecionar **Adicionar nó filho**. Queremos adicionar um nó **CollisionShape2D** como filho. Em seguida, no sistema de arquivos, localize a imagem **Player_Idle** e arraste-a para a janela de cena para adicioná-la como um nó filho. Renomeie esse nó para **Sprite**.

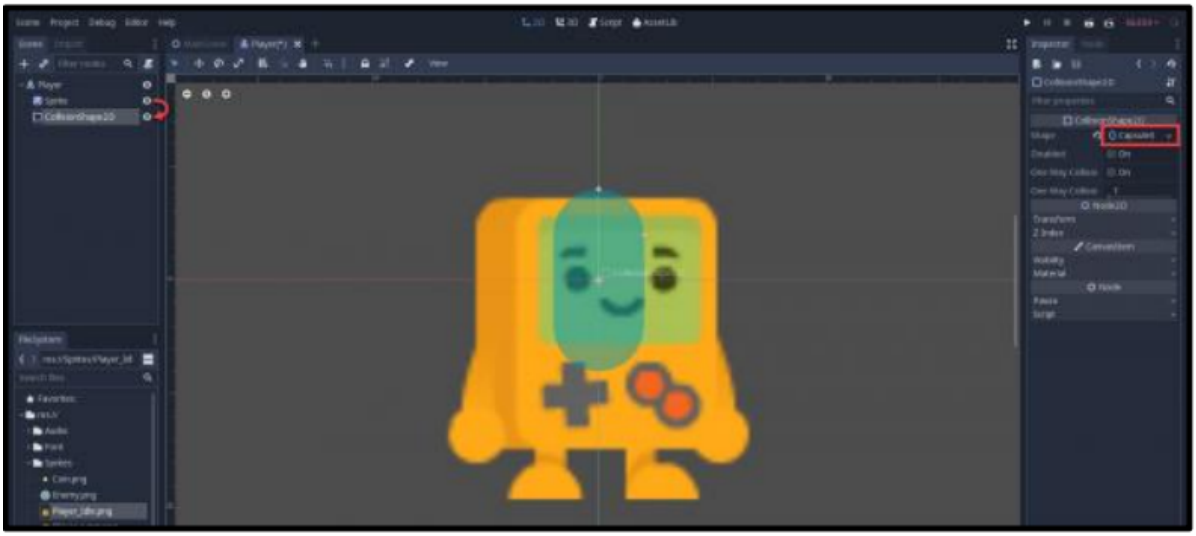


Queremos que nosso sprite seja centralizado, então selecione-o e mais no Inspetor:

- Abra o menu suspenso Transformar
- Defina a posição como 0, 0

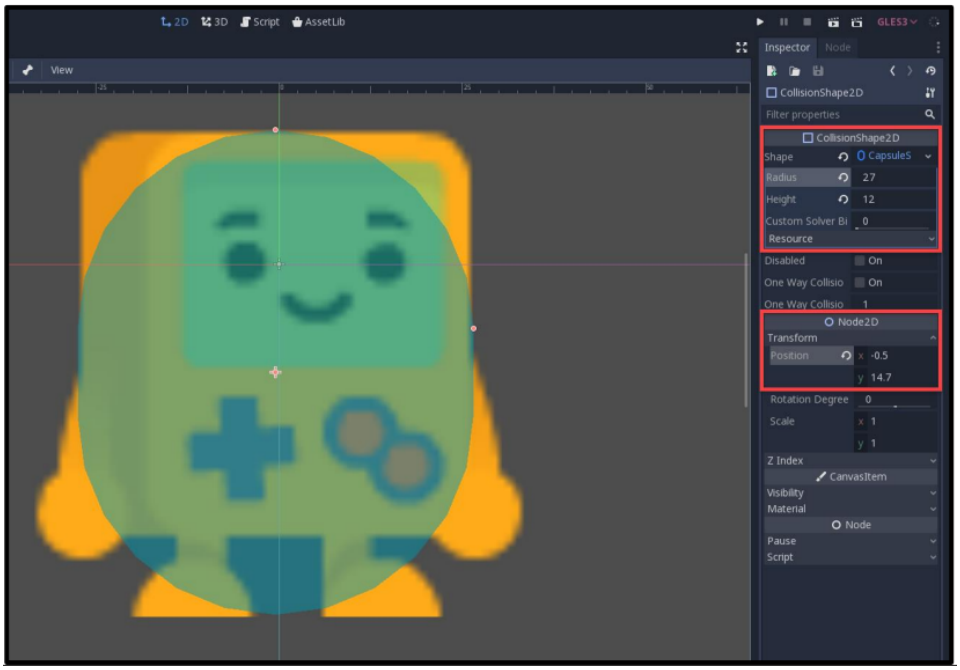


Você pode ver que o nó de colisão tem um símbolo de erro ao lado. Isso significa que precisamos dar uma forma a ele. Selecione o nó de colisão e, no inspetor, defina **Shape** como Cápsula. Para torná-lo visível, reordene a hierarquia do nó arrastando o nó do colisor para baixo do nó do sprite.



Para editar as propriedades do Collider, selecione a forma no Inspetor e ela abrirá mais opções.

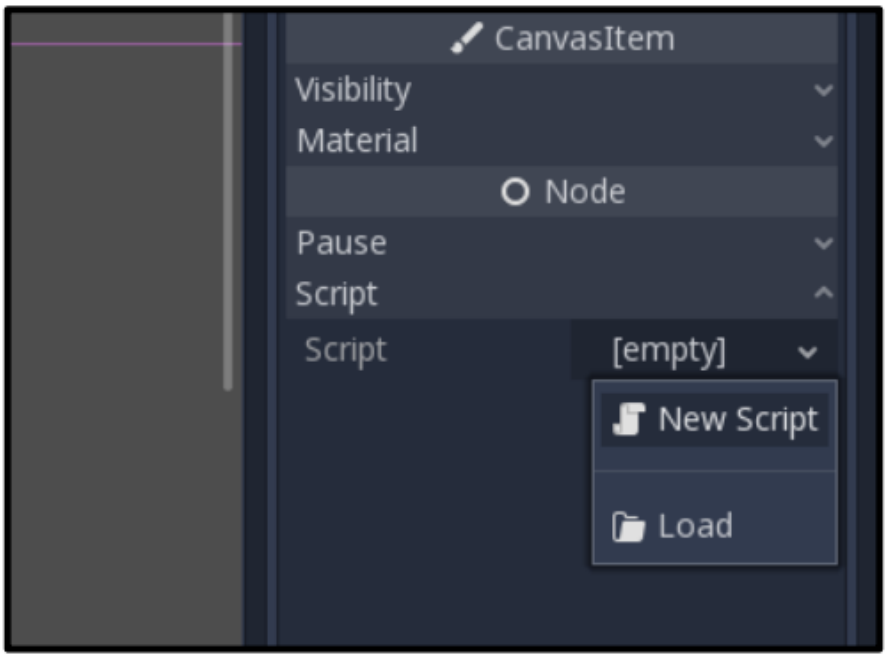
- Ajuste o **Radius** para 27
- Ajuste a **Height** para 12
- Ajuste a **Position** para -0,5, 14,7



Script do Player

Agora que temos a configuração da estrutura do nó, podemos começar a criar scripts para o nosso **player**. Isso envolverá o movimento, pular e coletar moedas.

Para criar um script, selecione o nó pai do **Player** e, no Inspetor, crie um novo script. Isso abrirá outra janela, basta pressionar enter.



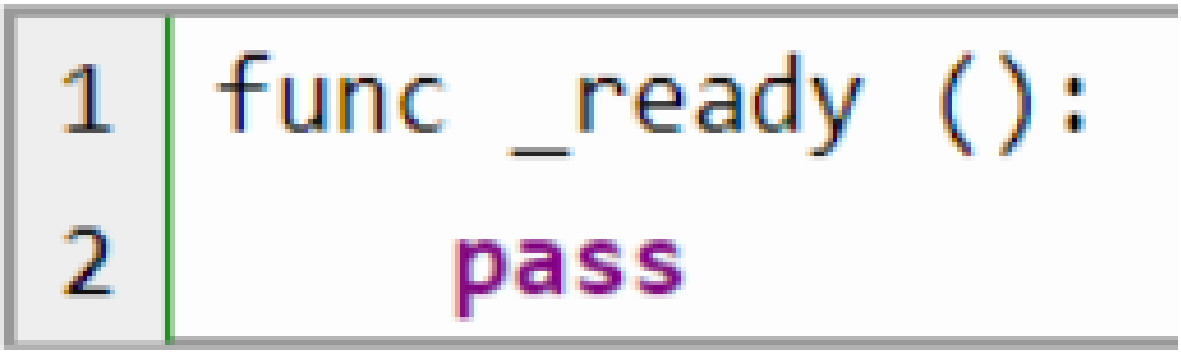
O que você verá então é que a janela principal mudará do modo 2D para o modo Script.

Godot usa sua própria linguagem de script chamada **GDScript**. Isso é semelhante em sintaxe ao Python. Não vamos abordar todos os aspectos da linguagem ou os conceitos básicos de programação - portanto, uma compreensão básica é necessária daqui para frente.

Agora no roteiro temos duas coisas.

```
1 extends KinematicBody2D
```

Extends é semelhante ao uso ou importação. Estamos estendendo a partir do objeto 2D do corpo cinemático ao qual estamos conectados, então teremos acesso direto a esses atributos.



Esta é uma função que é basicamente um bloco de código reutilizável que podemos chamar. A função **_ready** é incorporada ao Godot e é chamada uma vez quando o nó é inicializado.

O **pass** é simplesmente uma linha de preenchimento para definir a função.

Se estiver vazio, ocorrerá um erro. O passe não faz nada. Vamos começar adicionando algumas variáveis.

```gdscript
# stats
var score : int = 0

# physics
var speed : int = 200
var jumpForce : int = 600
var gravity : int = 800

var vel : Vector2 = Vector2()
var grounded : bool = false
```

Um **Vector2** define um valor x e y que pode ser usado para posição, escala, rotação, velocidade. Vamos usá-lo para armazenar a velocidade atual do nosso jogador.

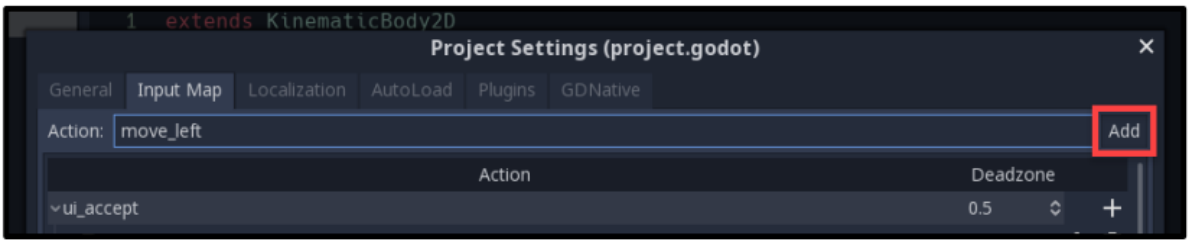
Precisamos de mais uma variável. Esta é uma referência ao componente sprite que é um nó filho. **onready** significa que encontraremos o nó Sprite quando o nó for inicializado.

```gdscript
# components
onready var sprite = $Sprite
```

Agora que temos nossas variáveis definidas, começaremos a colocar nosso jogador

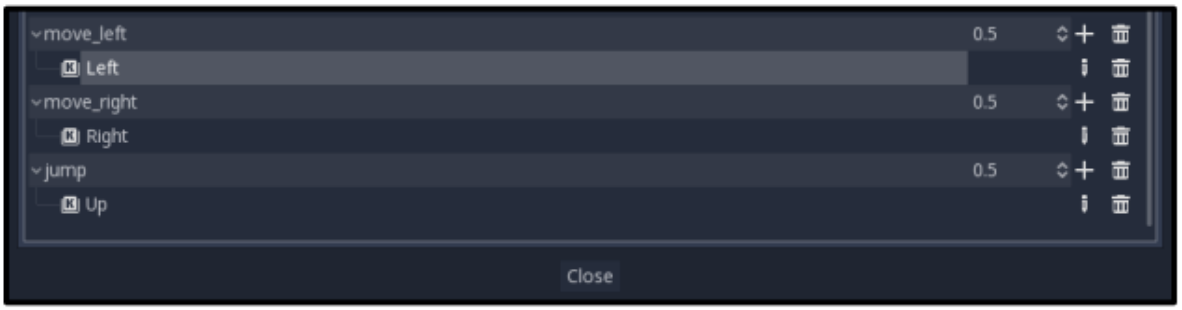
em movimento. Para fazer isso, primeiro queremos definir algumas entradas importantes.

Abra a janela Configurações do Projeto (Configurações do Projeto > do Projeto) e, nela, navegue até a guia **Mapa de Entrada**. No campo Ação, insira um nome para a entrada e clique em adicionar.



Queremos criar 3 novas ações.

- move_left
- move_right
- jump



Feito isso, podemos fechar a janela e continuar criando scripts.

Agora criamos uma função que está embutida em Godot. A função **_physics_process** é chamada 60 vezes por segundo e é o que usamos para cálculos físicos. Podemos remover o **pass** assim que começarmos a preencher a função.

O parâmetro **delta** é o tempo entre cada quadro. Podemos multiplicar nosso movimento por isso, a fim de nos movermos com base em pixels por segundo, em vez de pixels por frame.

```
1 func _physics_process (delta):
2     pass
```

Primeiro, queremos redefinir a velocidade horizontal. Em seguida, verifique as entradas de teclas de movimento à esquerda e à direita. Estes irão alterar a velocidade horizontal.

```
1 # reset horizontal velocity
2 vel.x = 0
3
4 # movement inputs
5 if Input.is_action_pressed("move_left"):
6     vel.x -= speed
7 if Input.is_action_pressed("move_right"):
8     vel.x += speed
```

Em seguida, moveremos o jogador usando a função **move_and_slide** que se moverá ao longo de uma determinada velocidade, detectando colisores e outras coisas. O segundo parâmetro é o solo normal (para que lado o solo está apontando?).

```
1 # applying the velocity
2 vel = move_and_slide(vel, Vector2.UP)
```

Depois disso, podemos aplicar a gravidade e verificar se estamos pressionando o botão de salto e no chão. Se sim, pule.

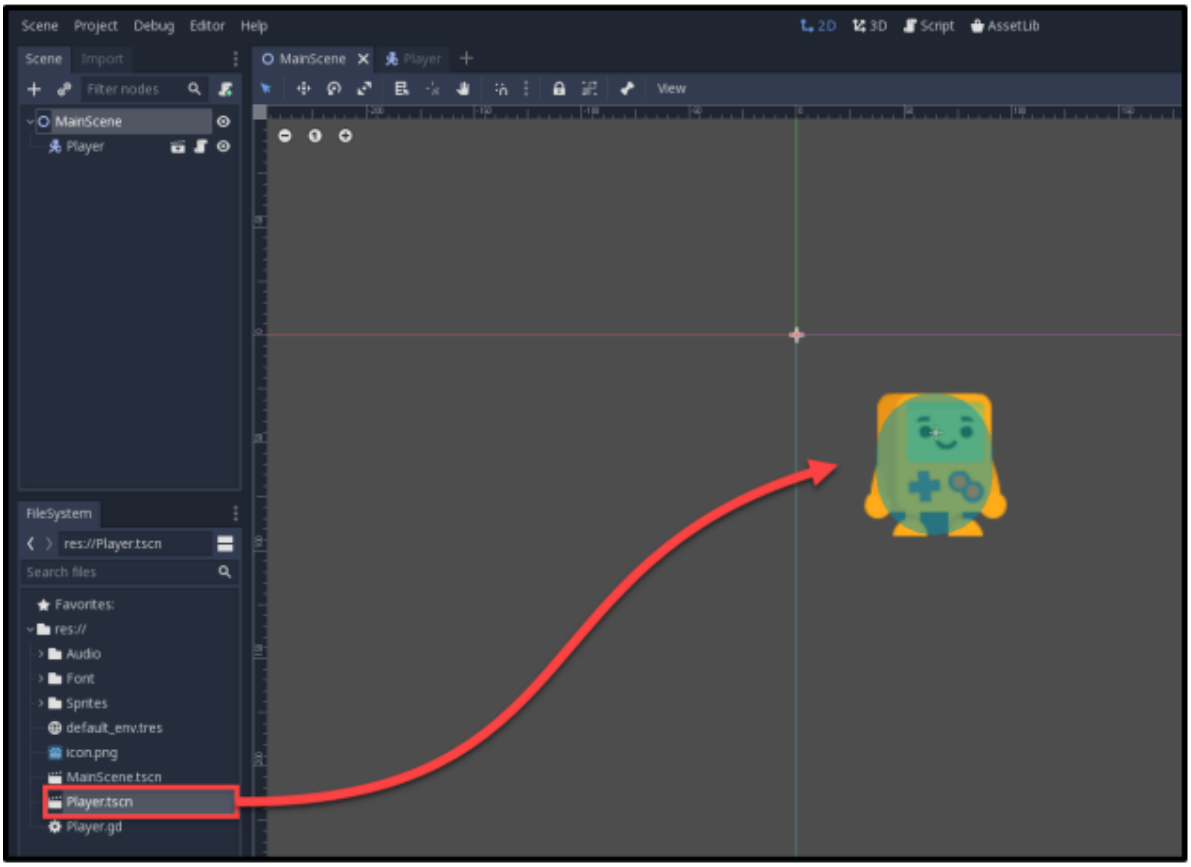
```
# gravity
vel.y += gravity * delta

# jump input
if Input.is_action_pressed("jump") and is_on_floor():
    vel.y -= jumpForce
```

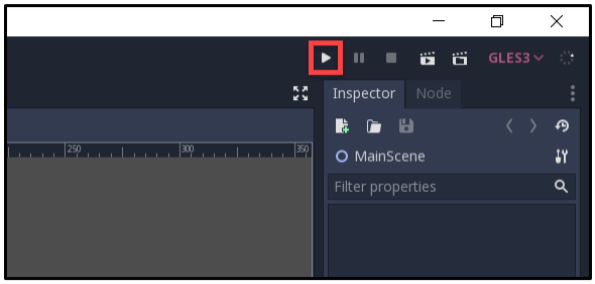
Finalmente, podemos inverter o sprite dependendo de qual caminho estamos nos movendo.

```
# sprite direction
if vel.x < 0:
    sprite.flip_h = true
elif vel.x > 0:
    sprite.flip_h = false
```

Terminamos o script por enquanto, então voltamos para o modo **2D** e ir para o **MainScene**. Aqui, queremos arrastar a cena **Player.tscn** do sistema de arquivos



O que podemos fazer agora é testar o jogo. No canto superior direito da tela, clique no botão Reproduzir.



Uma janela aparecerá dizendo que não há nenhuma cena principal **selecionada**. Clique no botão Selecionar e selecione a cena **MainScene.tscn** como a cena padrão. O jogo deve então abrir, mas rapidamente cairemos fora da tela devido à gravidade. O que precisamos fazer é criar uma cena de mundo que podemos duplicar na cena principal.

Mais isso fica para a proxima aula! ☺