Godot C#: Visão Geral

Prof. Thiago Felski Pereira, MSc.

Visão Geral

- Essa será nossa primeira experiência com o GODOT e também com um motor de jogos.
  - Por isso precisamos entender um pouco de cada uma dessas coisas antes de nos aprofundarmos em qualquer conteúdo.

- Tópicos:
  - O Godot.
  - A licença MIT.
  - Motor de jogos.
  - Principais conceitos.
  - Matemática vetorial.

UNIVALI

Visão Geral

- Godot fornece um enorme conjunto de ferramentas comuns, para que você possa se concentrar apenas em fazer seu jogo sem reinventar a roda.

- Godot é totalmente gratuito e de código aberto sob a licença muito permissiva do MIT. Sem amarras, sem royalties, nada. Seu jogo é seu, até a última linha do código do motor.

UNIVALI

Licença

- Godot utiliza Licença MIT permissiva (também chamada de licença Expat)
- Esta licença concede aos usuários uma série de liberdades:
  - Você é livre para usar Godot Engine, para qualquer finalidade
  - Você pode estudar como Godot Engine funciona e alterá-lo
  - Você pode distribuir versões não modificadas e alteradas do Godot Engine, mesmo comercialmente e sob uma licença diferente (incluindo proprietária)
- A única restrição a essa terceira liberdade é que você precisa distribuir o aviso de direitos autorais e a declaração de licença do Godot Engine sempre que o redistribuir

UNIVALI

Download

- Iremos baixar o Godot para C#, mas se preferir existe suporte para outras 4 linguagens de programação: GDScript, VisualScript, and, via its GDNative technology, C and C++

- Baixe o software em
  - https://godotengine.org/download/windows/
  - Selecione a opção com suporte a C# (Opção destacada em **CINZA**)

UNIVALI

Motor de jogos

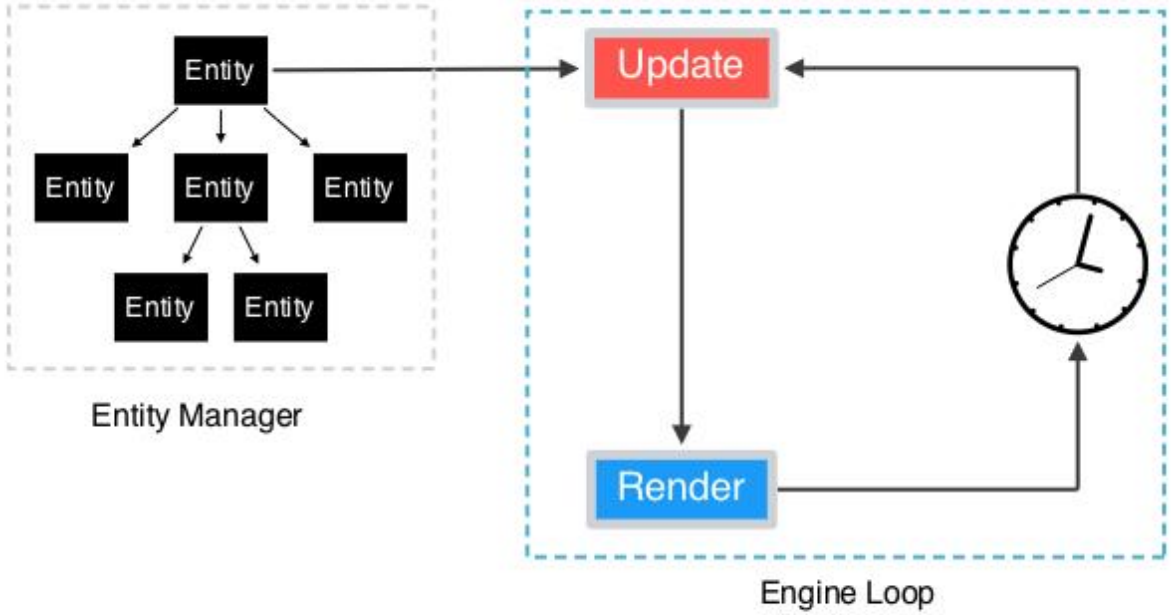
- É um *framework* projetado para construção e desenvolvimento de jogos.
- Seu objetivo é abstrair os detalhes complexos do desenvolvimento de jogos como física e renderização gráfica.
- Permitindo que o programador foque nas questões mais diretamente ligadas ao seu jogo.

- Para entender o funcionamento completo de um motor de jogos, precisaríamos estar familiarizados com programação concorrente.
  - Mas, nesse momento, é suficiente imaginar que um jogo é formado por vários objetos indepententes que irão interagir entre si por meio de sinais.

UNIVALI

Motor de jogos

- Diagrama de um fluxo geral de um motor de jogos:
  - A cada frame (Tempo) o jogo será atualizado (Update) e renderizado (Render) sendo que as atualizações serão solicitadas pelas entidades (Entity) que compõem o jogo.
  - Entre o Update e o Render é possível tratar interações como colisões, atritos e gravidade.



Programação em C#

- Porque C# se a linguagem principal é o GDScript?
  - Porque o C# é utilizado em todo lugar, enquanto o GDScript é utilizado apenas no Godot.
  - Você pode utilizar todo “arsenal” do .Net para seu jogo, como por exemplo utilizar bibliotecas de aprendizado de máquina para criar inimigos que aprendem com seus próprios erros.
  - Utilizada em outros ambientes de desenvolvimento de jogos populares: Unity.

UNIVALI

Godot: Principais conceitos

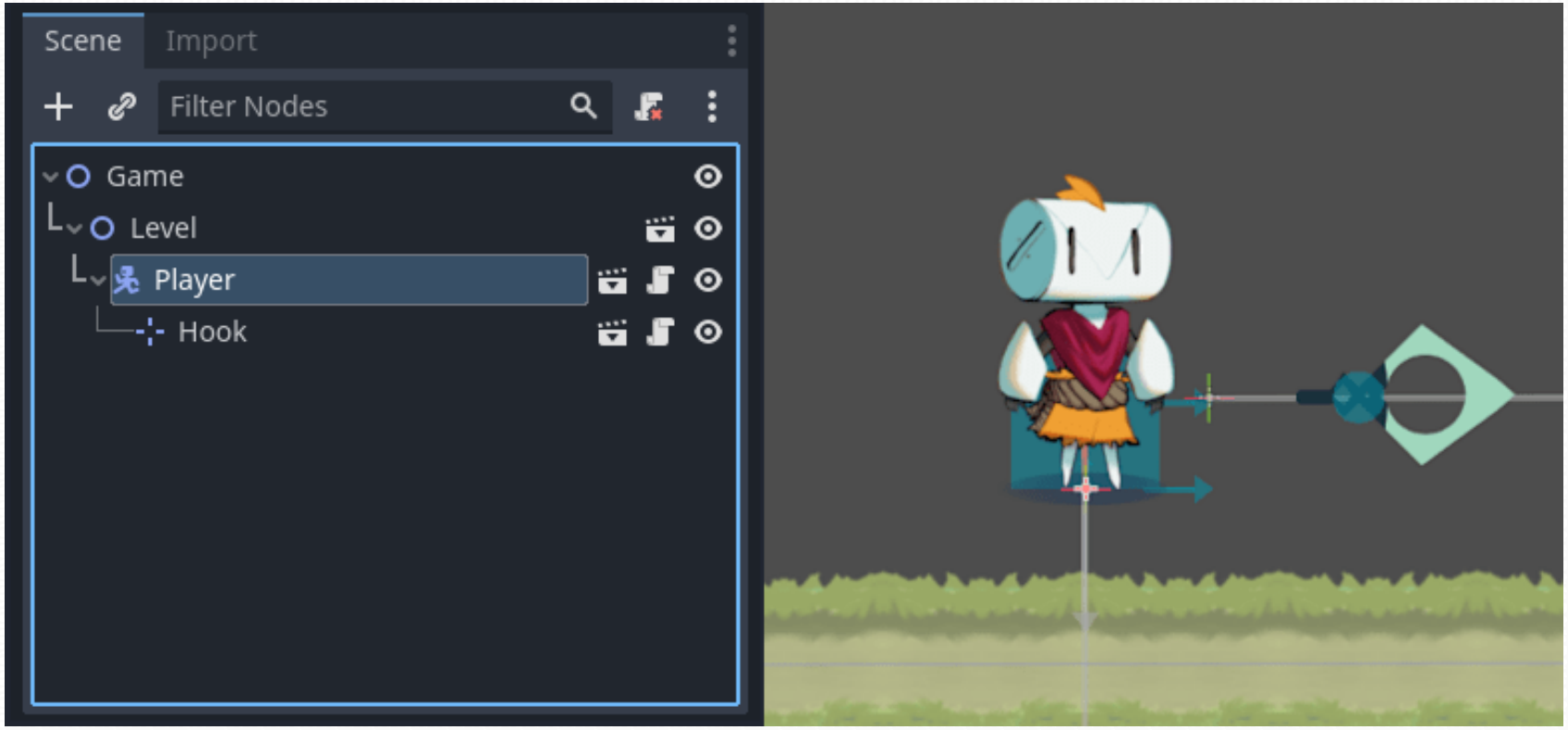
- Todo jogo em Godot é composto por uma árvore de **nodes** (Nós) que podem ser agrupados em **scenes** (cenas).

- **Scenes**
  - Um jogo é composto por cenas reutilizáveis.
  - Uma cena pode ser um personagem, um menu na interface do usuário, ...
  - As cenas fazem o papel tanto das cenas em si quanto de *prefabs* de outros motores de jogos.
  - Também é possível encapsular cenas.
    - Por exemplo, pode-se colocar o personagem em um nível e arrastar uma cena como filha dela.

UNIVALI

Godot: Principais conceitos

- **Scenes**
  - Um jogo é composto por cenas reutilizáveis.
  - Também é possível encapsular cenas.
    - Por exemplo, pode-se colocar o personagem em um nível e arrastar uma cena como filha dela.

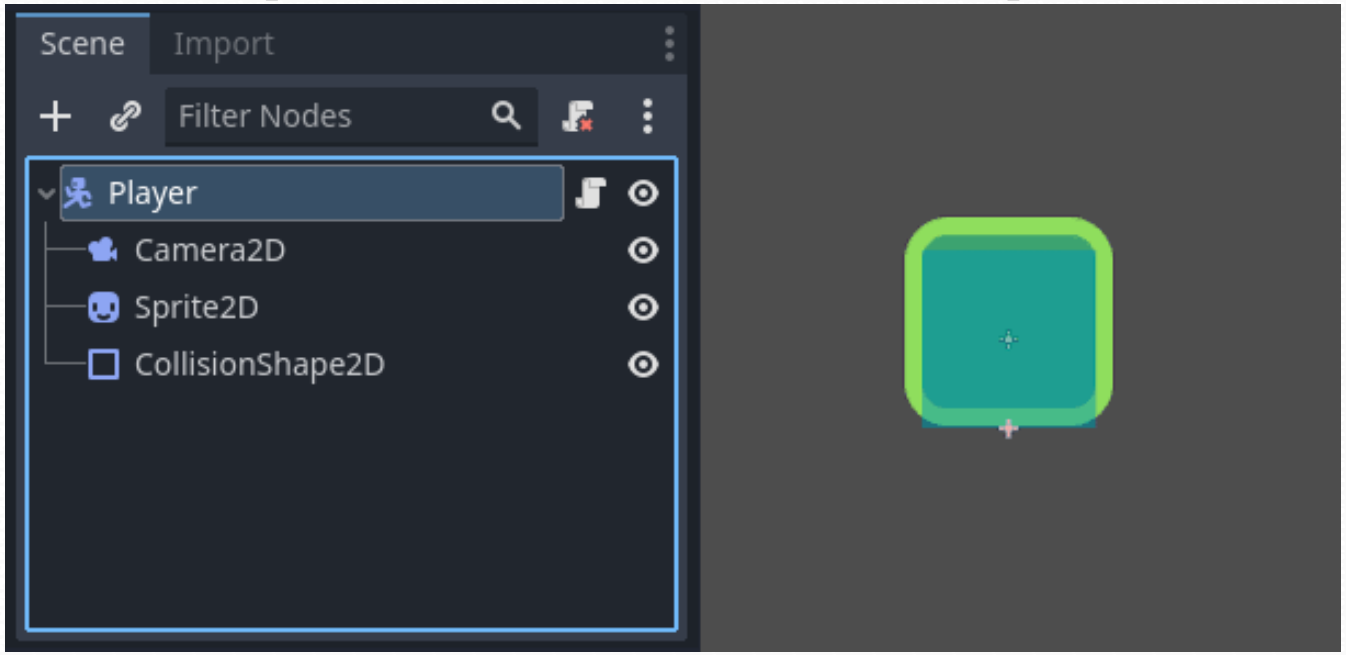


UNIVALI

Godot: Principais conceitos

- **Nós**
  - Uma cena é composta por um ou mais Nós.
  - Nós são o menor bloco construtivo que podem ser organizados em árvores.

  - A imagem, a seguir é composta de um Nó `CharacterBody2D` renomeado como Player, uma `Camera2D`, um `Sprite2D` e um `CollisionShape2D`.



UNIVALI

Godot: Principais conceitos

- **Nós**
  - Todos os nodos são executados no estilo *top-down*.
    - Mas em que ordem serão executados os Nodos ao lado?
    - Para testar, vamos colocar os seguinte trecho de código em todos os Nós.


```
public override void _EnterTree() {
    GD.Print(Name + " enter tree");
}
1 reference
public override void _Ready() {
    GD.Print(Name + " ready");
}
6 references
private bool teste = true;
1 reference
public override void _Process(double delta) {
    if (teste) {
        GD.Print(Name + " process");
        teste=false;
    }
}
```


UNIVALI

Godot: Principais conceitos

- **Nós**
  - Significado das funções presentes no trecho de código.
    - `_EnterTree()` é chamada quando o Nó entra na árvore pela primeira vez. Ocorre quando é instanciado ou quando é instanciado ou quando `AddChild()` é utilizado.
    - `_Ready()` é chamado quando o Nó e todos seus filhos são instanciados na árvore e estão prontos.
    - `_Process()` é chamado a cada frame (normalmente 60 vezes por segundo) em cada Nó na árvore.

  - Ao executar apenas o nó raiz, a ordem de execução não gera dúvidas.

```
TesteRaiz enter tree
TesteRaiz ready
TesteRaiz process
```

UNIVALI

Godot: Principais conceitos

- **Nós**
  - Ao executar a cena completa é possível observar que o `_Ready()` não executou na ordem apresentada na árvore.
    - Isso porque o Nó só fica pronto após todos os seus filhos também ficarem.
  - *"Os pais devem cuidar dos filhos e não o contrário."*
    - Esse ditado é válido também para o Godot.
    - Em outras palavras, um Nó filho terá dificuldade de acessar um Nó pai na função `_Ready()`

```
TesteRaiz enter tree
TesteFilho1 enter tree
TesteFilho3 enter tree
TesteFilho2 enter tree
TesteFilho3 ready
TesteFilho1 ready
TesteFilho2 ready
TesteRaiz ready
TesteRaiz process
TesteFilho1 process
TesteFilho3 process
TesteFilho2 process
```

UNIVALI

Godot: Principais conceitos

- **Nós**
  - A comunicação entre nodos pode ser feita de muitas formas, mas existe um jeito certo.
  - A **comunicação do jeito errado** pode acabar com longos caminhos que irão quebrar sempre que a estrutura do seu projeto mudar.

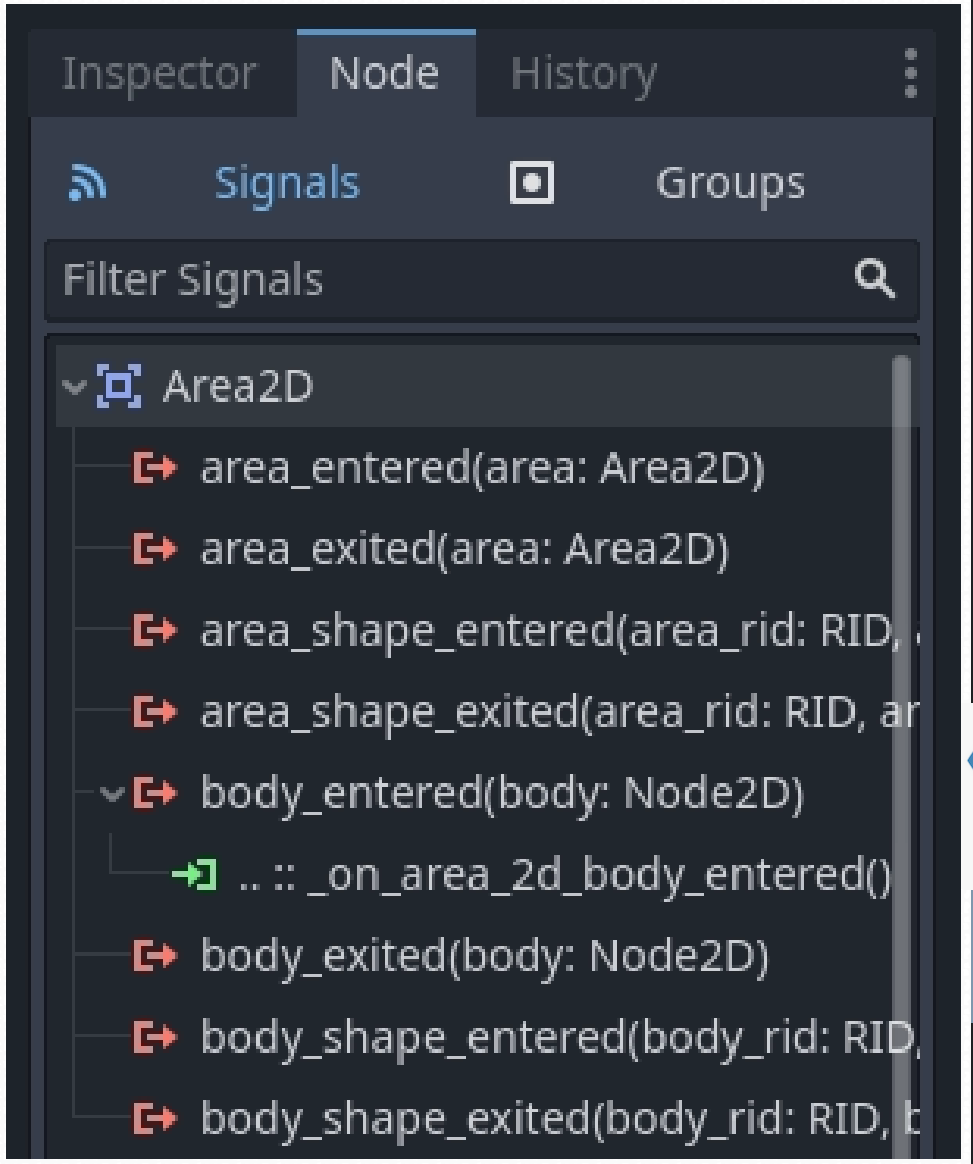
```
GetNode("../../Nodo/OutroNodo");
GetParent().GetParent().GetNode("Nodo");
GetTree().Root.GetNode("Nodo/OutroNodo");
```

  - A comunicação do jeito certo segue a seguinte regra:
  - Chama se está abaixo e sinaliza se está acima (*Call down, signal up*).
    - Se um nodo está chamando um filho (*down*), então `GetNode()` é apropriado.
    - Se um nodo precisa se comunicar acima na árvore (*up*), então signal é apropriado.

UNIVALI

Godot: Principais conceitos

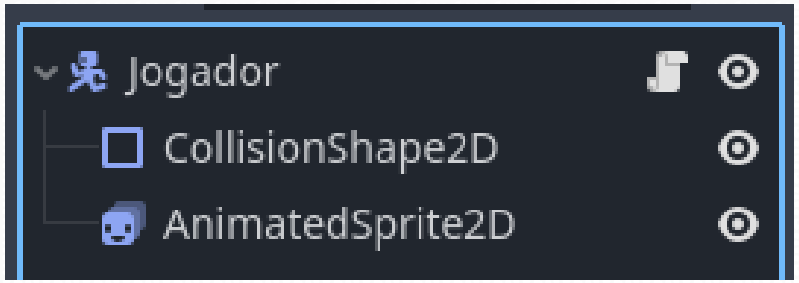
- **Signals**
  - Nós emitem sinais quando um evento ocorre.
  - Esta funcionalidade permite que os Nó se comuniquem.
  - Os sinais dão flexibilidade ao estruturar as cenas.
  - Por exemplo:
    - Botões emitem sinais quando apertados. Pode-se conectar esses sinais para rodarem um código em reação a esse evento.
    - Outros sinais podem dizer quando dois objetos colidiram, quando um personagem ou monstro entrou em uma determinada área, ...



UNIVALI

Godot: Principais conceitos

- **Nós /** `GetNode()`



  - O script no Jogador precisa informar o AnimatedSprite2D qual animação rodar baseado no movimento do Jogador.
  - Nessa situação `GetNode()` funciona corretamente.

```
public override void _Process(double delta) {
    Vector2 velocity = Velocity;
    if(velocity.X > 0) {
        GetNode<AnimatedSprite2D>(AnimatedSprite2D).Play("correr");
    } else {
        GetNode<AnimatedSprite2D>(AnimatedSprite2D).Play("parar");
    }
}
```


UNIVALI

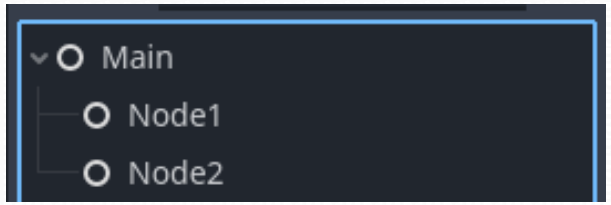
Godot: Principais conceitos

- **Nós /** `Signal`
  - Sinais devem ser utilizados para chamar funções em Nós que estão no mesmo nível ou acima na árvore.
  - Pode-se conectar sinais no editor (eles devem existir antes do jogo começar) ou no código (para Nó que você está instanciando em execução)
  - Criar sinais por código pode ser mais complicado, mas facilita o reuso e teste do Nó.
  - A sintaxe para criar um sinal é:
  - `[Signal] public delegate void sinalEventHandler();`
  - A sintaxe para emitir um sinal é:
  - `EmitSignal("sinal"); //EventHandler() é omitido`
  - A sintaxe para conectar um sinal é:
  - `NoDoSinal.Connect("sinal", new Callable(NodoDaFunção, "Função") );`

UNIVALI

Godot: Principais conceitos

- **Nodes /** `Signal`



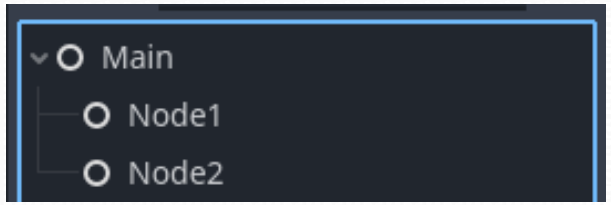
  - Considere que o Node1 precisa informar ao Node2 que a tecla ENTER foi pressionada, como eles estão no mesmo nível da hierarquia, o recomendado para fazer essa comunicação é o uso de sinais.
  - **Node1**: Criação e emissão do sinal.

```
public partial class Node1 : Node {
    4 references
    [Signal] public delegate void BotaoEventHandler();
    1 reference
    public override void _Process(double delta) {
        if(Input.IsActionJustPressed("ui_accept")) {
            GD.Print("Sinal Botao Enviado");
            EmitSignal("Botao");
        }
    }
}
```


UNIVALI

Godot: Principais conceitos

- **Nodes /** `Signal`



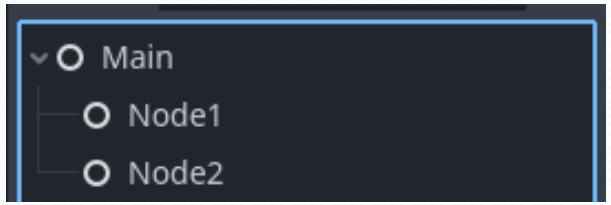
  - Considere que o Node1 precisa informar ao Node2 que a tecla ENTER foi pressionada, como eles estão no mesmo nível da hierarquia, o recomendado para fazer essa comunicação é o uso de sinais.
  - **Node2**: Função que será ativada pelo sinal.

```
public partial class Node2 : Node {
    1 reference
    public void SinalRecebido() {
        GD.Print("Sinal Botao Recebido");
    }
}
```


UNIVALI

Godot: Principais conceitos

- **Nodes /** `Signal`



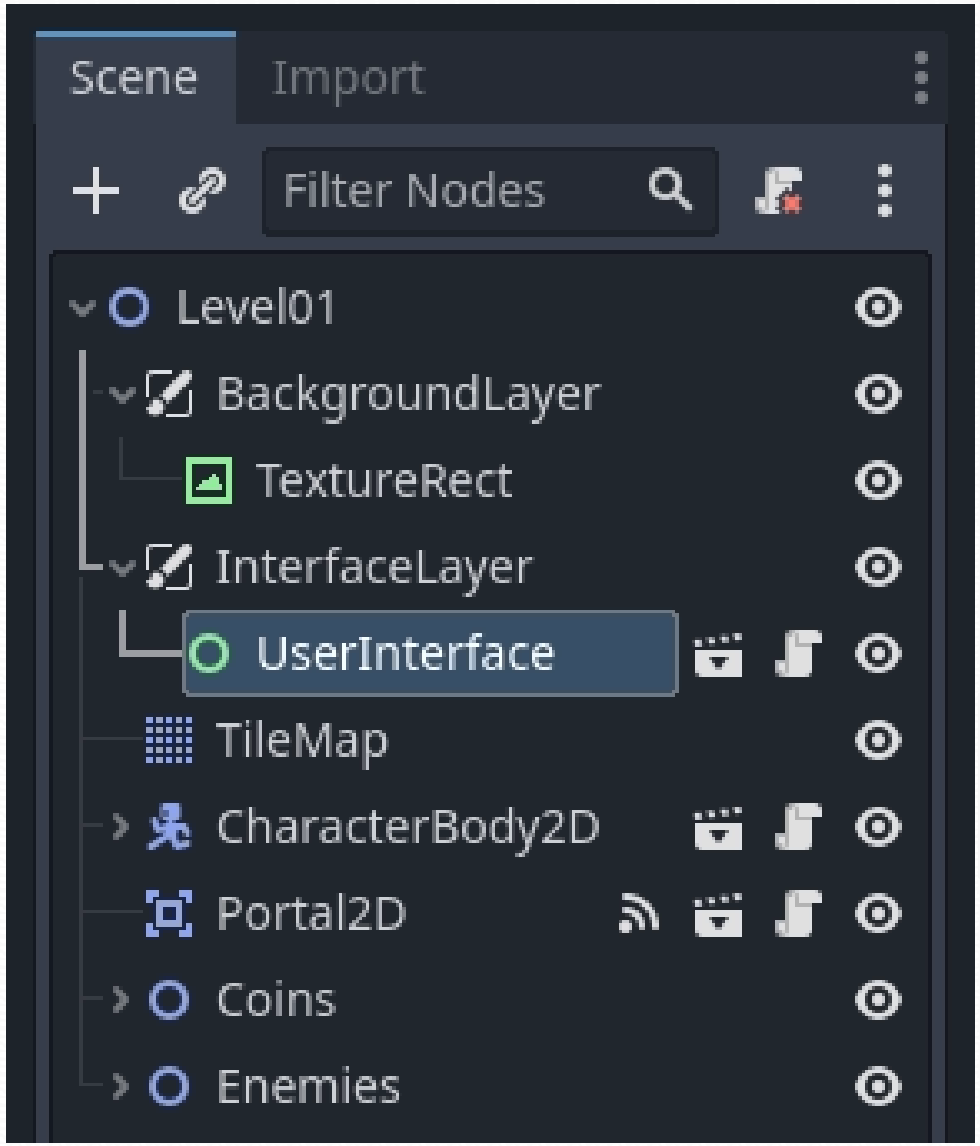
  - Considere que o Node1 precisa informar ao Node2 que a tecla ENTER foi pressionada, como eles estão no mesmo nível da hierarquia, o recomendado para fazer essa comunicação é o uso de sinais.
  - **Main**: Conecta o sinal à função chamada por ele.

```
public partial class Main : Node {
    1 reference
    public override void _Ready() {
        GetNode<Node>("Node1").Connect("Botao", new Callable(GetNode<Node>("Node2"), "SinalRecebido") );
    }
}
```


UNIVALI

Godot: Principais conceitos

- **Árvore de Cenas**
  - É possível importar uma cena em outra, ao fazer isso, a cena importada terá sua hierarquia ocultada e será vista como um nodo pela cena principal.
  - No seu jogo as cenas são reunidas para montar a árvore de cenas do jogo.



UNIVALI

Godot: Principais conceitos

- **Scripts**
- São códigos que anexados aos nodos para estender as funcionalidades da classe em que são anexados.
- Usaremos scripts sempre que precisarmos programar algo no nosso jogo, ou seja, não programaremos diretamente em uma classe, mas em uma extensão dessa classe.

UNIVALI


UNIVALI