Sobre o Godot

O Godot é uma ferramenta para a criação de jogos em duas dimensões (2D) ou em três dimensões (3D). Foi criado originalmente pelo MIT (Massachusetts Institute of Technology) e é um software livre, gratuito e de código aberto.

Baixando a aplicação

Acesse esse endereço para baixar a aplicação:

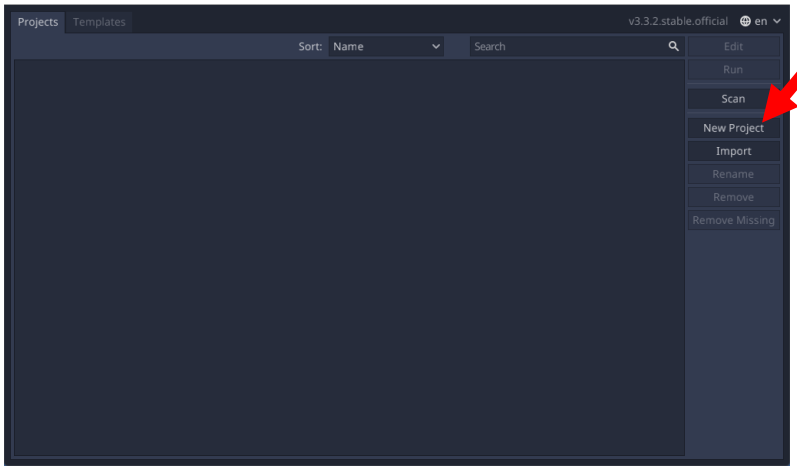
https://godotengine.org/download/

A instalação é bem simples, basta descompactar o ZIP e pronto. A aplicação é "self-contained", ou seja, é só clicar no arquivo descompactado para executar. Há versões para Windows, Mac e Linux.

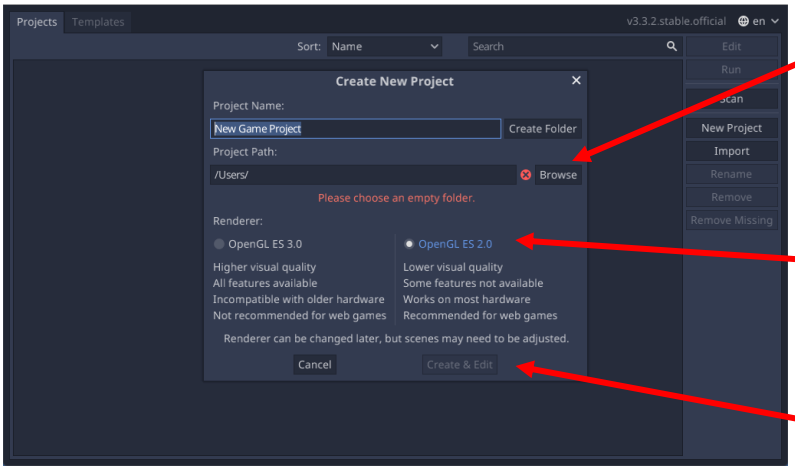
Baixe a `Standard version`, é mais simples em termos de requisitos e indicada para iniciantes.

Criando um jogo

Essa tela é carregada no início da aplicação:



Obs.: É possível alterar o idioma da interface, mas não é recomendado. Além da tradução não ser completa, vai dificultar a consulta da documentação, a maioria está em inglês.



Navegue para escolher a pasta em que irá ficar o projeto

Escolha esse "renderer" (mais simples)

Clique aqui para criar

Tela principal do editor do jogo

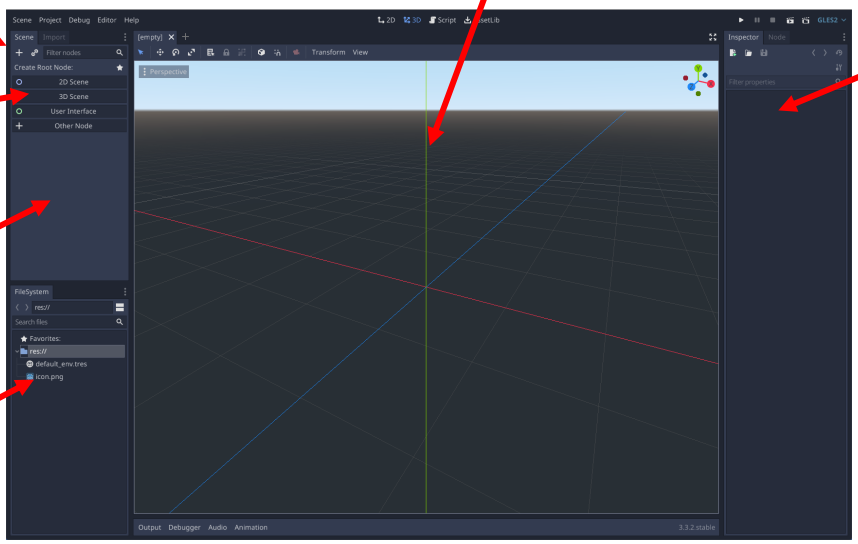
Janela de exibição (viewport) da cena do jogo

Botão + para a criação de nós

Atalhos para facilitar a criação de nós

Árvore de nós da cena

Arquivos do projeto



Inspetor (inspector) de nós e propriedades

Notar que a árvore (tree) dos nós (nodes) está vazia, pois nada ainda foi criado.

Na parte dos arquivos do projeto, há apenas dois arquivos:

`default_env.tres`: arquivo com as configurações básicas do jogo

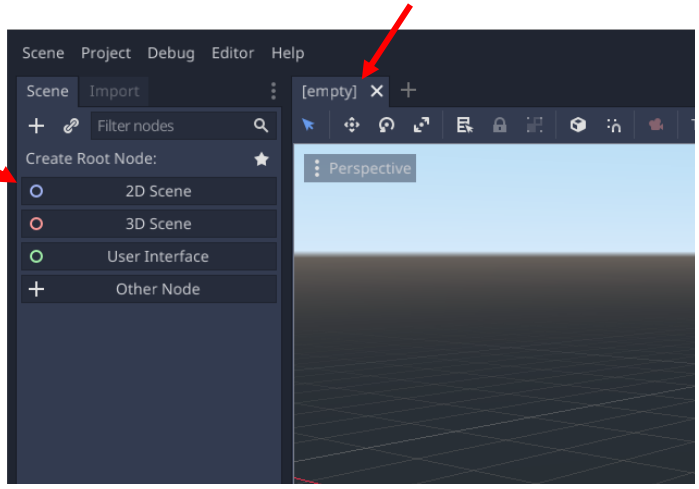
`icon.png`: imagem que vem apenas como exemplo

Criando um jogo muito simples

Um jogo ocorre em uma cena (scene), na qual existem nós (nodes). Um jogo pode ter uma ou mais cenas e uma cena tipicamente possui vários nós. Mas é preciso que cada cena tenha ao menos um nó, chamado de nó principal ou nó raiz (root node).

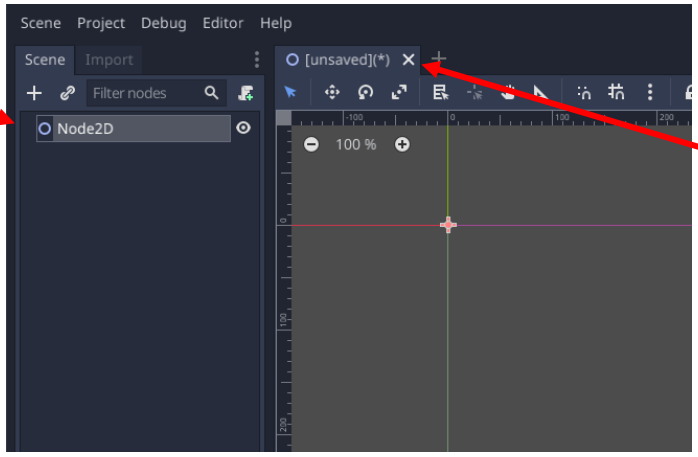
Quando o editor carrega pela primeira vez, é a cena está vazia (empty) e é necessário criar o nó raiz:

Clique aqui para criar um nó raiz do tipo `2D Scene`



Após a criação do nó raiz, a tela ficará assim:

Esse é o nó raiz recém criado

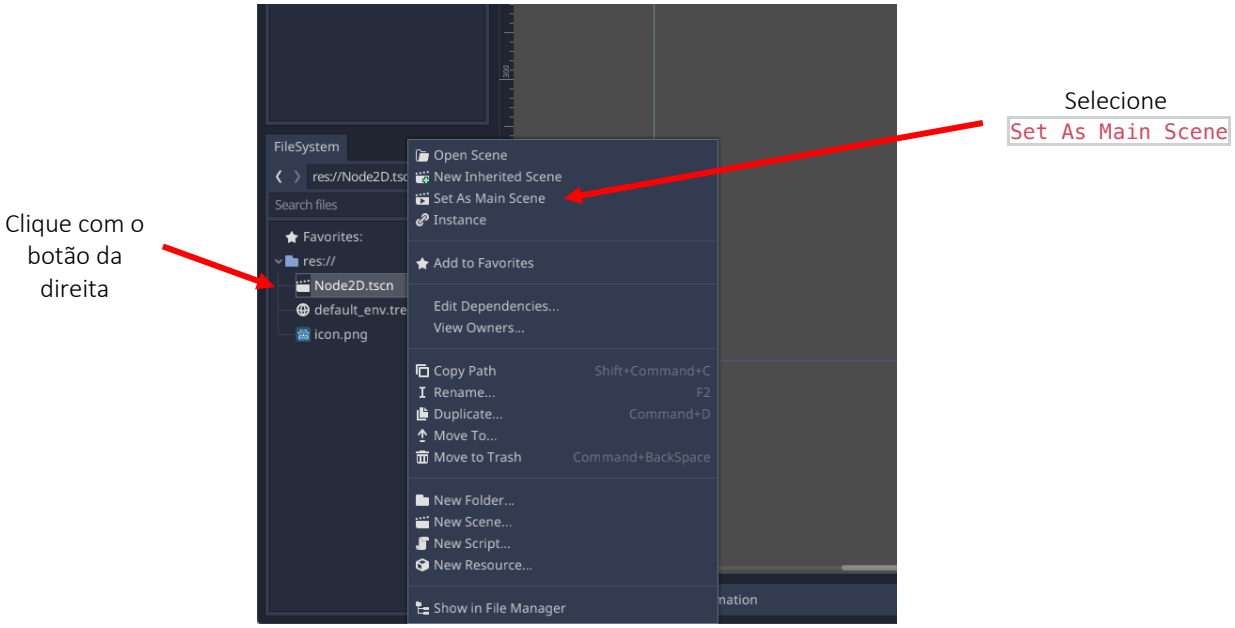


A cena ainda não foi salva (unsaved)

Repare que foi criada também uma cena, mas ela ainda não foi salva (unsaved). Para salvar a cena, pressione `CTRL+S` (Windows) ou `CMD+S` (Mac). Após salvar a cena, observe na parte dos arquivos do projeto (parte inferior esquerda) que surgirá um arquivo `Node2D.tscn`. As cenas (com seus nós) ficam salvas em arquivos TSCN.

Rodando o jogo

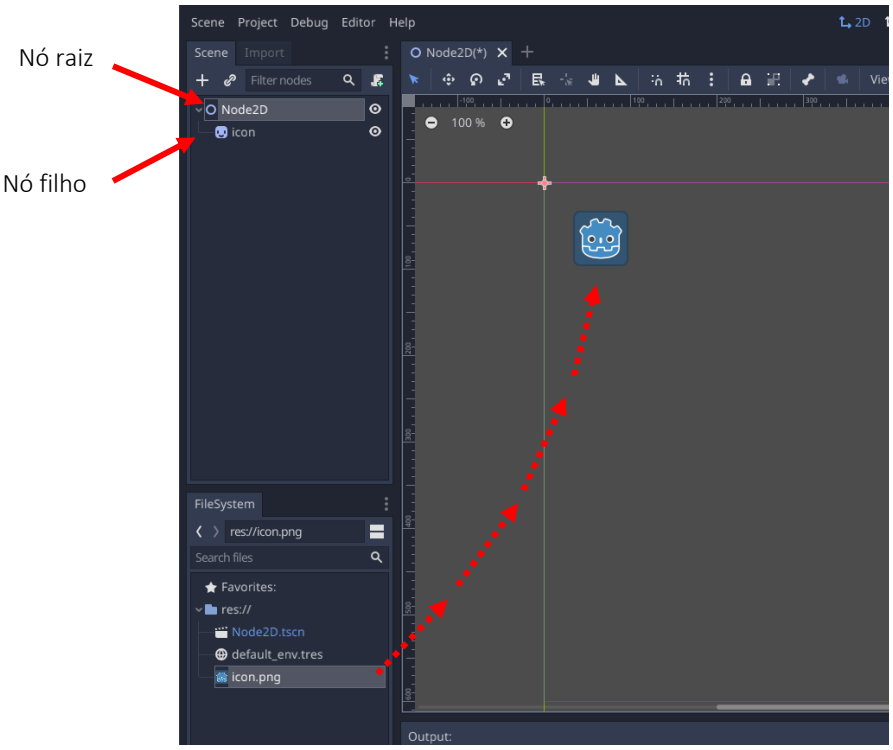
Antes de rodar o jogo, é preciso informar que a cena recém criada é a cena principal (main scene) do jogo. Para isso, clique com o botão da direita no arquivo da cena e selecione `Set As Main Scene`.



Para rodar o jogo, pressione `CTRL+B` (Windows) ou `CMD+B` (Mac). Parabéns! Esse seu primeiro jogo ... mas essa ainda é uma cena vazia, não há nada nela, exceto a informação que é uma cena em duas dimensões (2D).

O primeiro personagem

Vamos criar um personagem nesse jogo. Arraste o arquivo `icon.png` para a janela de visualização (viewport) do jogo:



Note que foi criado um nó, abaixo do nó raiz. Esse é um “nó filho” (child node).

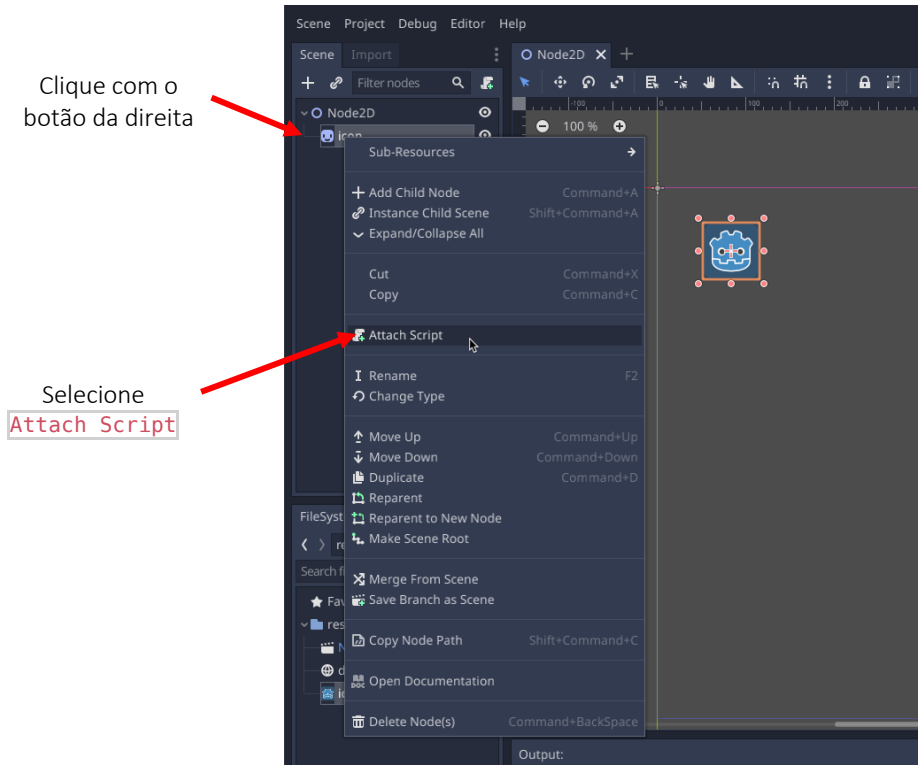
Pressione `CTRL+B` (Windows) ou `CMD+B` (Mac) para rodar o jogo. Agora esse jogo tem um personagem ... mas ele ainda não faz nada.

Movimentando o personagem (meu primeiro programa)

Agora vamos movimentar o personagem, usando o teclado. Para isso, vamos empregar uma linguagem de programação chamada GDScript.

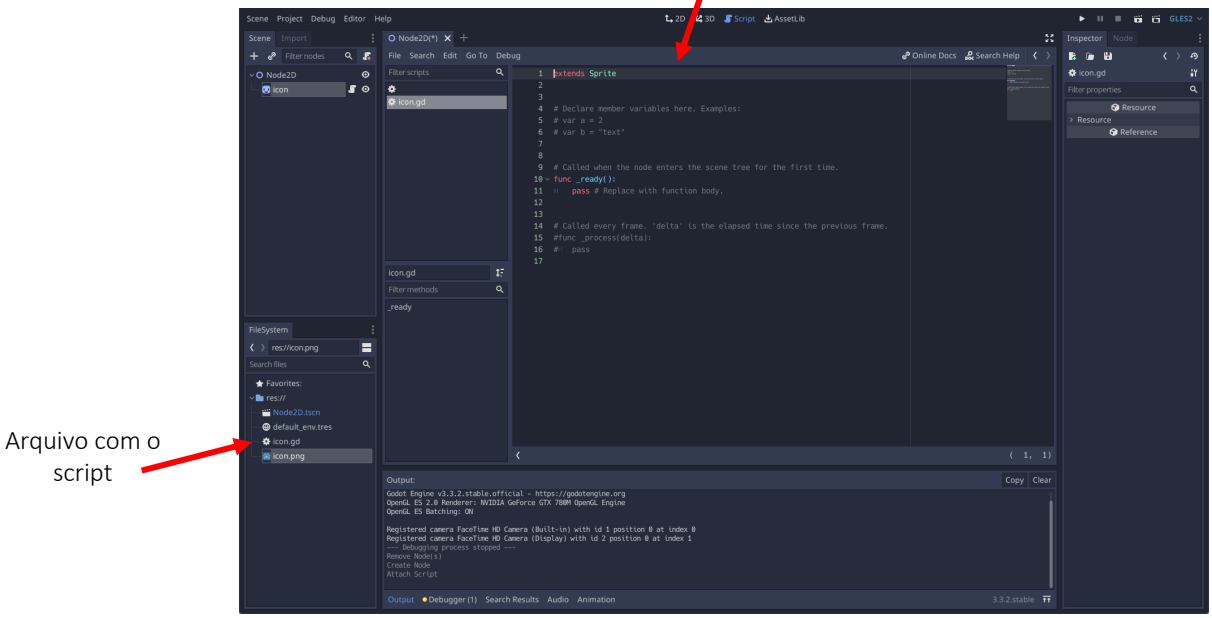
> ⓘ Informação
>
> O **GDScript** é uma linguagem criada especialmente pelos desenvolvedores do Godot e é muito semelhante com a linguagem Python. Assim, para quem nunca programou, aprender GDScript é uma boa forma de se familiarizar com os conceitos básicos do Python.

É necessário criar o script para a programação. Para tanto, clique com o botão da direita no nó `icon` e selecione a opção `Attach Script`.



Quando aparecer a janela da criação do script, clique em `Create` para salvar o arquivo do script (programa).

O editor de scripts



Note na parte dos arquivos do projeto (parte inferior esquerda) que surgirá um arquivo `icon.gd`. Os scripts ficam salvos em arquivos GD. Repare assim, que os scripts não são salvos nos arquivos da cena (TSCN), são salvos em arquivos separados (GD). Essa característica será bem útil para reaproveitar um mesmo script em diferentes cenas, mas isso será visto posteriormente.

Agora, vamos escrever algo nesse script. Insira o código abaixo no editor de scripts, depois da última linha:

GDScript

```
func _process(delta):
   if Input.is_action_pressed("ui_down"):
        position.y += 1
   if Input.is_action_pressed("ui_up"):
        position.y -= 1
```

Salve e rode o jogo. Quando carregar a tela, pressione as teclas para baixo e para cima. Agora sim! Esse é seu primeiro jogo. Não é grande coisa, mas pelo menos você já sabe como o Godot funciona!

Um pouco mais do GDScript

O script recém criado deve ter ficado dessa forma:

```
1  extends Sprite
2
3
4  # Declare member variables here. Examples:
5  # var a = 2
6  # var b = "text"
7
8
9  # Called when the node enters the scene tree for the first time.
10 func _ready():
11     pass # Replace with function body.
12
13
14 # Called every frame. 'delta' is the elapsed time since the previous frame.
15 #func _process(delta):
16 #    pass
17 func _process(delta):
18     if Input.is_action_pressed("ui_down"):
19         position.y += 1
20     if Input.is_action_pressed("ui_up"):
21         position.y -= 1
22
```

Vamos entender o que está escrito:

Classes

```
1  extends Sprite
```

Essa linha faz com que seja requisitada a classe `Sprite`. Em programação, uma `classe` é bloco de código elaborado previamente por outras pessoas, com determinadas características que facilitam o desenvolvimento das aplicações. É como se fosse uma ferramenta, por exemplo, uma tesoura. Imagine que você precise cortar um papel, então você "pega" uma tesoura e corta. Você não precisa fabricar a tesoura, você apenas "pega" a tesoura. No caso, o ato de "pegar" é o sentido do comando `extends`. Ou seja, "extends Sprite" significa "pegue a Sprite" (depois vamos falar sobre o que faz a classe Sprite). Em outras linguagens de programação, equivale a comandos como "request", "import" e "include", entre outros.

Comentários

```
4  # Declare member variables here. Examples:
5  # var a = 2
6  # var b = "text"
```

Nessa parte o script não está fazendo nada. Isso porque no início de cada linha há um caractere `#` (jogo da velha). Esse caractere "neutraliza" tudo que está escrito na linha. É claro, você poderia simplesmente apagar a linha, mas digamos que você ainda não tem certeza de que pode apagar, mas não quer que o programa rode essa linha. Então, basta colocar o caractere `#` no início da linha e ela será ignorada.

O caractere `#` tem também uma outra função muito importante. Serve para escrever comentários, observações e anotações para facilitar o entendimento do código. Esse processo é a "documentação do código", algo muito importante de ser feito, para que outras pessoas possam compreender o que o programador pensou em cada parte da aplicação.

Métodos

```
9  # Called when the node enters the scene tree for the first time.
10 func _ready():
11     pass # Replace with function body.
```

Esse bloco introduz um conceito importante que são os `métodos` (methods) ou funções. No GDScript um método sempre começa com a expressão `func`. No exemplo, está sendo chamado o método `_ready()` que é um método nativo (já existente) de cada nó (node). Esse método é executado cada vez que o respectivo nó entra na cena, ou seja, pode ser usado para rodar determinados comandos no início.

No exemplo, há apenas o comando `pass`, que nada faz. Está colocado apenas como exemplo. Ou seja, o `func _ready()` desse código não está fazendo nada, está aí apenas para ilustrar que poderia ser usado. Experimente, por exemplo, apagar essas três linhas, veja que não farão falta.

Há duas coisas, porém, que merecem ser destacadas nessa parte do código:

1. Note que na linha 9 há um comentário, usando o caractere `#`, explicando o que faz esse método.

2. Note que na linha 11, o comando `pass` não está alinhado com a linha superior. Ele está deslocado à direita, usando o `tab`. Essa formatação é chamada `indentação` e uma característica importante da linguagem GDScript (assim como no Python). Todos os comandos de um método precisam ficar indentados com `tabs`.

Loops

```
14  # Called every frame. 'delta' is the elapsed time since the previ
15  #func _process(delta):
16  #    pass
17  func _process(delta):
18      if Input.is_action_pressed("ui_down"):
19          position.y += 1
20      if Input.is_action_pressed("ui_up"):
21          position.y -= 1
```

Esse bloco trata de um importante método: `func _process(delta)`. Esse método também é nativo de cada nó e é executado de forma contínua, como um loop. Na programação de jogos, os loops são fundamentais para dar a sensação de movimento, ou seja, a cada volta do loop, também chamada de quadro ou frame, os objetos do game são reposicionados, dando a impressão de que se movimentaram. É isso que está escrito no comentário da linha 14.

As linhas 15 e 16 foram mantidas nesse exemplo, apenas para facilitar, mas poderiam ter sido removidas.

> ⓘ Informação
>
> O argumento `delta` contém o tempo decorrido no frame. O Godot executa 60 frames por segundo (60Hz), ou seja, cada frame dura 0,01667 segundos. Você pode ignorar essa informação no momento, mas poderá ser útil no futuro.

Esse bloco de texto usa a expressão `Input.is_action_pressed`, que basicamente devolve qual tecla foi pressionada. No caso, o programa está preparado para reagir quando for pressionada a tecla da seta para baixo (`"ui_down"`) e para cima (`"ui_up"`).

Repare o `if` no início das linhas 18 e 20, bem como o `:` (dois pontos) no final. Essa é a maneira como o GDScript trata uma condição, ou seja, ele analisa o que está escrito entre o `if` e o `:`. Se essa condição for verdadeira, ele vai executar a linha de baixo, caso contrário vai ignorá-la. Note que os comandos sob um `if` também precisam estar indentados.

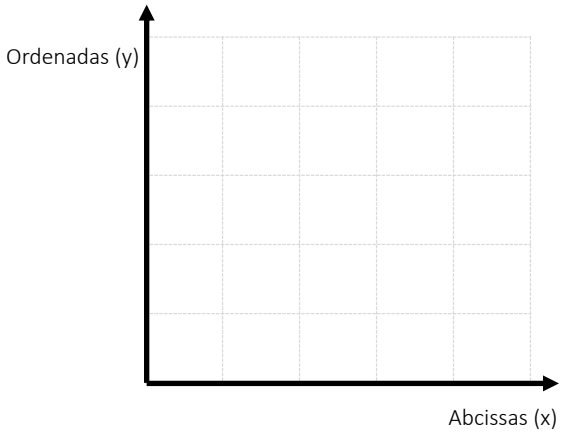
No exemplo, há dois `ifs`. O primeiro faz o personagem descer, o segundo faz o personagem subir, usando o atributo `position`. Lembra da classe `Sprite` comentada anteriormente? Pois bem, o atributo `position` é uma dos vários atributos da classe `Sprite`.

Sistema de coordenadas

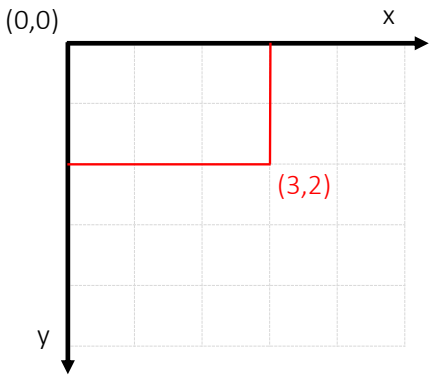
No exemplo, quando é pressionada a seta para baixo (`"ui_down"`) o sistema aumenta o valor de `position.y` em 1 pixel e quando é pressionada a seta para cima (`"ui_up"`), o programa reduz 1 pixel em `position.y`.

Para entender o que significa o `y` em `position`, é preciso compreender o sistema de coordenadas do Godot, que é o padrão empregado nas plataformas de desenvolvimento de games, entre outras aplicações.

Você provavelmente está acostumado com gráficos, em que há dois eixos: um horizontal (ou das abcissas) e um vertical (ou das ordenadas). O eixo horizontal é chamado também de eixo x e o vertical de eixo y.



No sistema de coordenadas dos jogos, o conceito é parecido, porém o eixo y fica invertido. Pode parecer estranho, mas esse é o padrão usado nas aplicações computacionais.



> ⓘ Informação
>
> Repare como está escrita a informação `(3,2)` no exemplo. Significa que a posição do ponto na horizontal (x) é 3 e na vertical (y) é 2.

No programa criado, a notação `position.y` representa a posição vertical do personagem do jogo. Assim, quando se aumenta esse valor, o personagem vai para baixo, e quando se diminui ele vai para cima.

A posição horizontal é representada por `position.x`, mas nesse exemplo não foi usada.

Sprite

Conforme foi ilustrado anteriormente, quando foi arrastado o ícone para a janela de visualização, o sistema automaticamente criou um nó filho, que passou a ser o personagem do jogo. O Godot possui inúmeros tipos de nós, no caso foi usado o tipo `Sprite`. Esse tipo de nó tem a capacidade de exibir imagens (também chamadas de "texturas") a partir de arquivos, no caso está exibindo o arquivo `icon.png`.

Para entender melhor o que aconteceu, em termos de lógica de programação, vamos explorar um pouco mais dois conceitos importantes: `classe` e `objeto`.

Uma `classe`, conforme foi comentado anteriormente, é uma "ferramenta" que contém códigos previamente elaborados. Já um `objeto` é uma "cópia" criada a partir dessa classe. Ou seja, a classe é um modelo (template) a partir do qual são criados os objetos. Imagine, por exemplo, um conjunto habitacional com várias casas. Existe uma planta básica, comum a todas as casas. Essa planta é a classe e as casas são os objetos.

Então, perceba que:

As casas, ao serem construídas, reproduzem as características definidas na planta.

Em programação, fazemos a mesma afirmação da seguinte forma:

Um objeto, ao ser instanciado, herda os atributos definidos na classe.

No exemplo, o Sprite é a classe e o personagem criado é o objeto. Assim, o personagem herda todos os atributos da classe Sprite.

Conclusão

Esse é um tutorial introdutório, permite compreender conceitos básicos de programação, bem como se familiarizar com as principais funcionalidades do Godot. Mas você pode ir muito além disso, alterando os scripts, adicionando novos personagens, trocando as imagens etc.

Saiba mais

Cenas e nós

https://docs.godotengine.org/pt_BR/stable/getting_started/step_by_step/scenes_and_nodes.html

GDScript básico

https://docs.godotengine.org/pt_BR/stable/getting_started/scripting/gdscript/gdscript_basics.html

Sistema de coordenadas e vetores

https://docs.godotengine.org/pt_BR/stable/tutorials/math/vector_math.html

https://docs.godotengine.org/pt_BR/stable/classes/class_vector2.html

https://www.khanacademy.org/math/linear-algebra

Classe Sprite

https://docs.godotengine.org/pt_BR/stable/classes/class_sprite.html

Quer fazer um jogo mais completo? Siga esse tutorial:

https://docs.godotengine.org/pt_BR/stable/getting_started/step_by_step/your_first_game.html

Dicas e informações

ⓘ Dica

Há um padrão para escrever os programas em GDScript, no que se refere ao uso de maiúsculas, minúsculas, espaços, pontuações e anotações. É o **Guia de Estilo do GDScript**. Veja mais em: https://docs.godotengine.org/pt_BR/stable/getting_started/scripting/gdscript/gdscript_styleguide.html

ⓘ Dica

Cada jogo criado fica contido em uma pasta do computador. Dentro de cada pasta há um arquivo `project.godot`. Basta clicar nesse arquivo para carregar a janela do editor do jogo. Isso permite carregar mais de um editor ao mesmo tempo.

ⓘ Dica

O Godot pode utilizar outras linguagens, além do GDScript. Veja em
https://docs.godotengine.org/pt_BR/stable/getting_started/step_by_step/scripting.html