Oficina de programação de jogos

Criando o jogo Flappy (no motor de jogos Godot)

Teoria

Saman Bemel Benrud
@samanbb

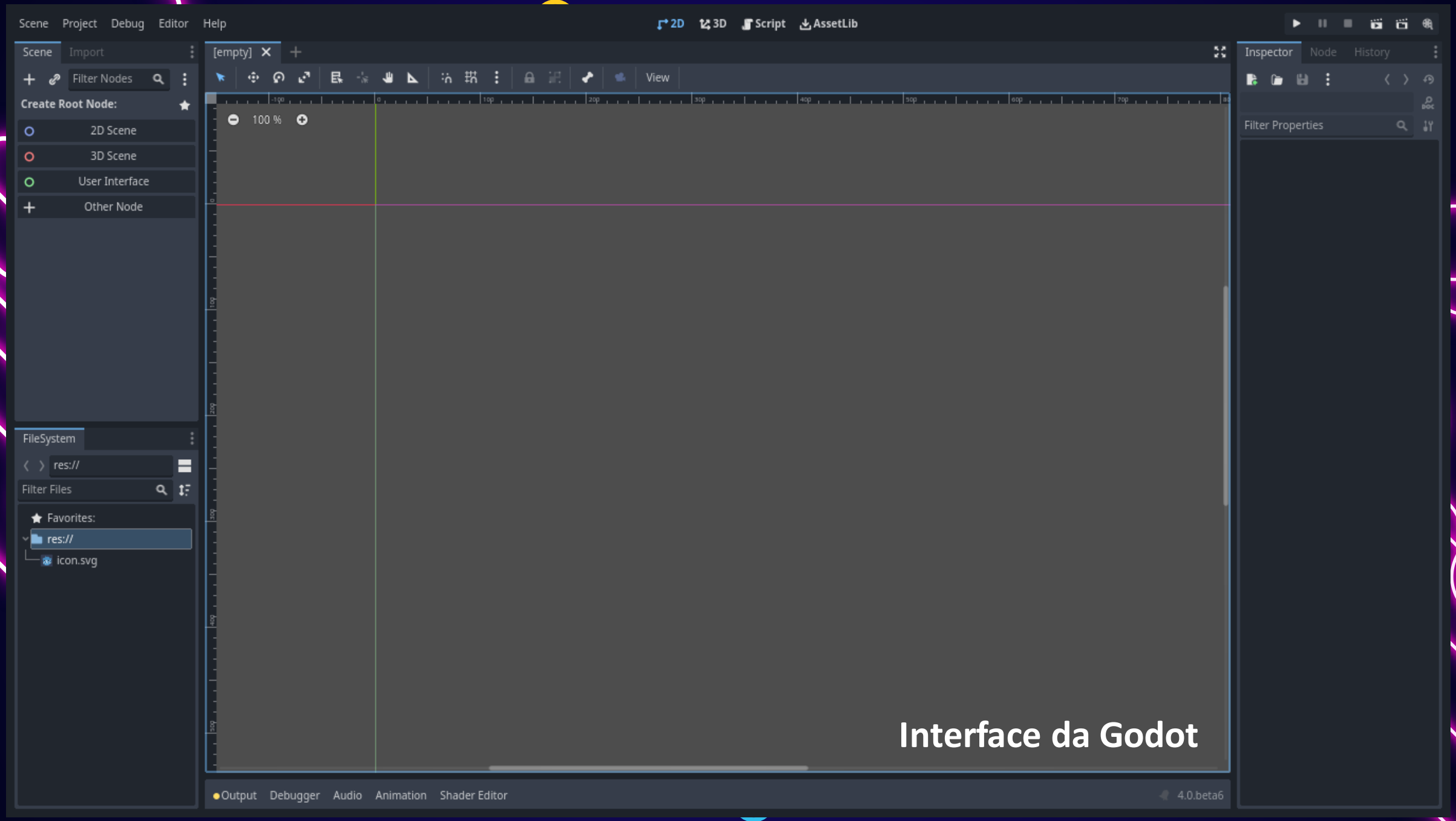
Game developers: with enough if statements and while loops I can do literally anything.

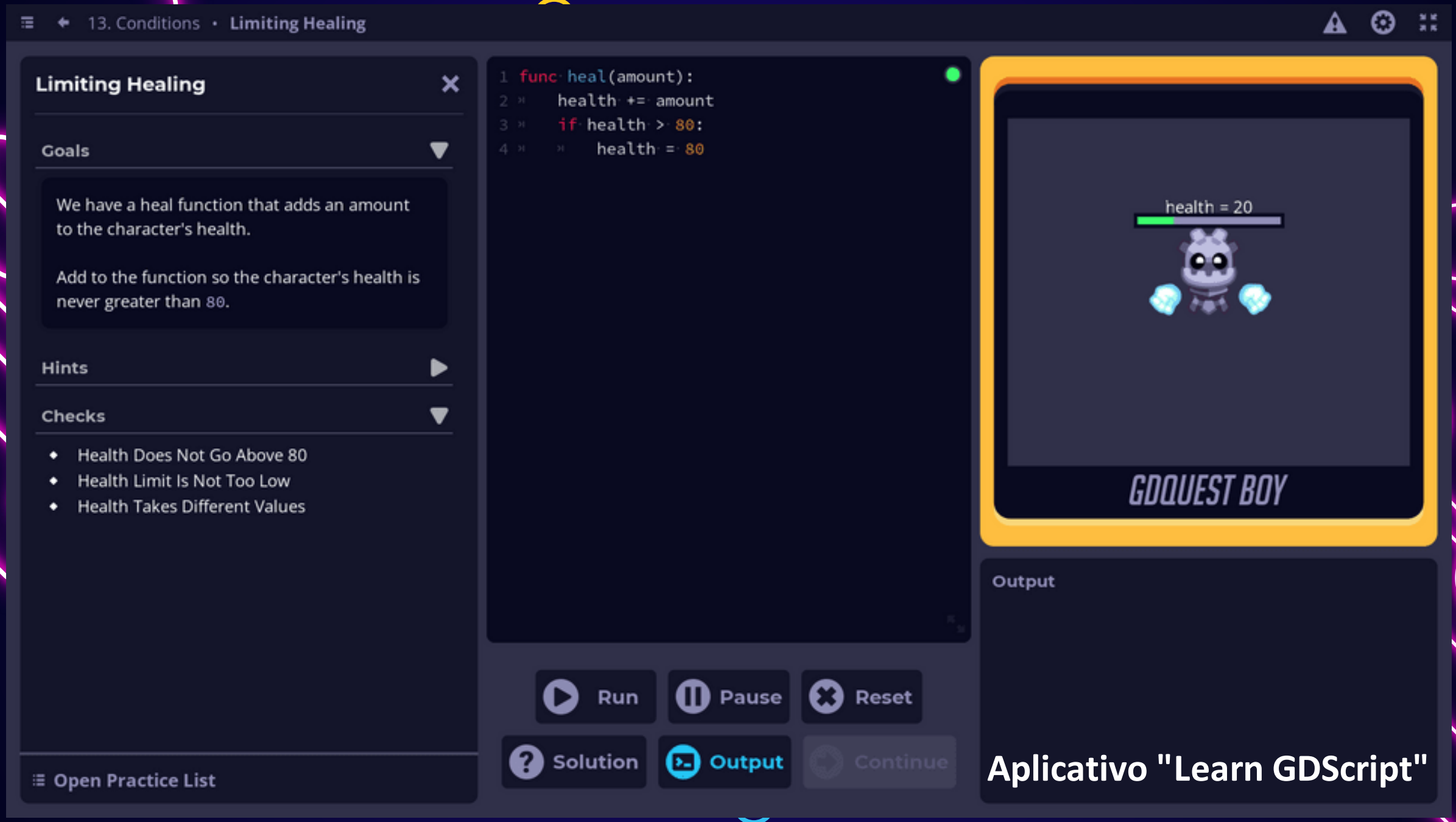
Web developers: I will use graph theory and a hand crafted functional state management framework to create this sign up form.

9:06 PM · 19/02/2021 · Twitter Web App

Olá Mundo!

```
func _ready():
    $Label.text = "Hello world!"
```


Scene
Project
Debug
Editor
Help
2D
3D
Script
AssetLib
Scene
Import
Filter Nodes
Create Root Node:
2D Scene
3D Scene
User Interface
Other Node
[empty]
View
100 %
Inspector
Node
History
Filter Properties
FileSystem
res://
Filter Files
Favorites:
res://
icon.svg
Output
Debugger
Audio
Animation
Shader Editor
4.0.beta6
Interface da Godot



Aplicativo "Learn GDScript"

Linguagens

Heatmap.cpp

```cpp
#include "Heatmap.h"
#include <TileMap.hpp>
#include <SceneTree.hpp>
#include <Viewport.hpp>
#include <deque>
#include <TileSet.hpp>
#include <algorithm>

namespace godot {
    inline float lerp(const float& a, const float& b, const float& t) {
        return a + t * (b - a);
    }

    void Heatmap::_register_methods() {
        //public
        register_method("_ready", &Heatmap::_ready);
        register_method("_draw", &Heatmap::_draw);
        register_method("_process", &Heatmap::_process);
        register_method("best_direction_for", &Heatmap::best_direction_for);
        register_method("calculate_point_index", &Heatmap::calculate_point_index);
        register_method("calculate_point_index_for_world_position",
               &Heatmap::calculate_point_index_for_world_position);

        //semi-private
        register_method("_on_Events_player_moved", &Heatmap::on_Events_player_moved);

        //properties
        register_property<Heatmap, NodePath>("pathfinding_tilemap", &Heatmap::m_pathfinding_tilemap,
              NodePath());
```

10k godot-platformer-2d/game/src/Native/Heatmap/Heatmap.cpp 25:0 Top7:35AM 0.97 LF UTF-8 C++//l master

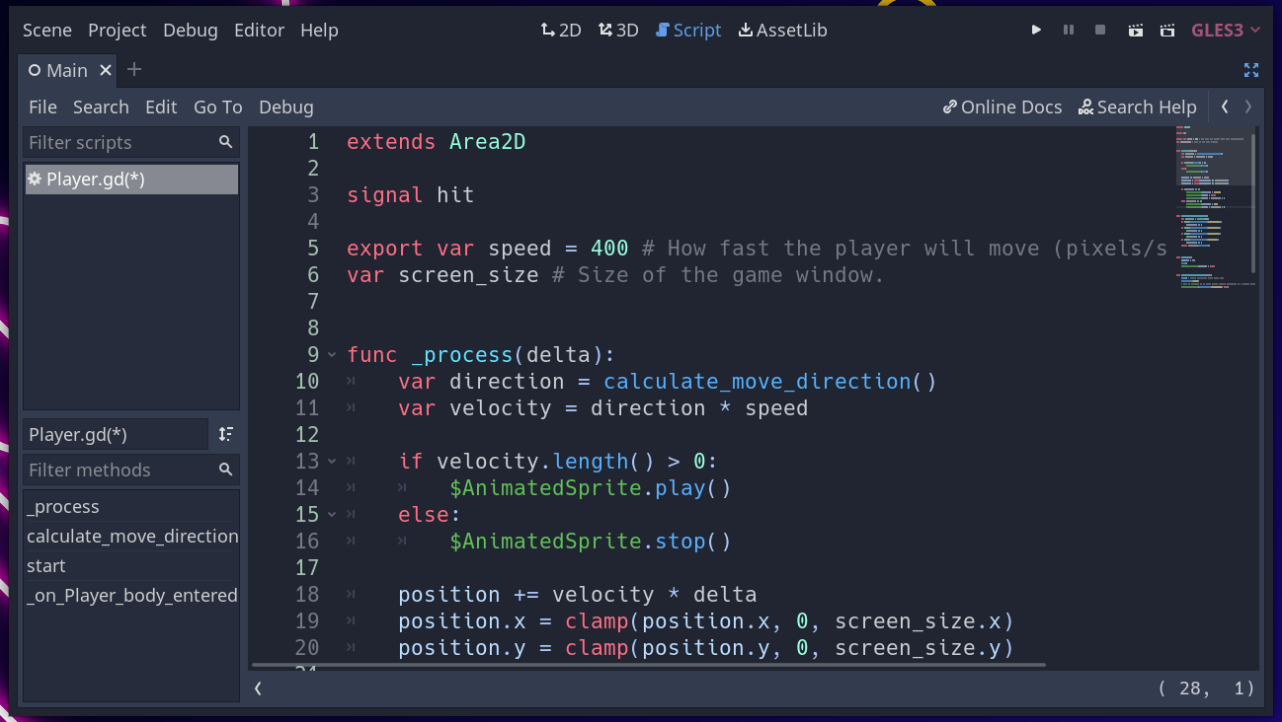
C++

C#

```
if (Input.IsActionPressed())
    "ui_accept"
    "ui_cancel"
    "ui_down"
    "ui_end"
    "ui_focus_next"
    "ui_focus_prev"
    "ui_home"
    "ui_left"
    "ui_page_down"
    "ui_page_up"
    "ui_select"
```

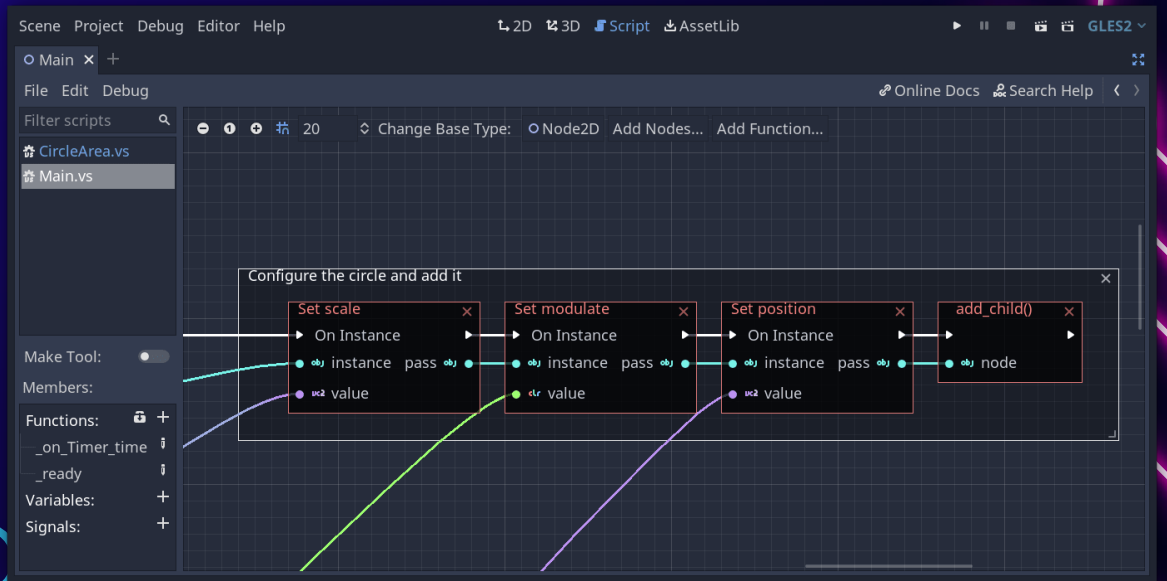
```
NodePath Sprite = ;
    "Camera"                    NodePath
    "Platform"                  NodePath
    "Platform/CollisionShape"   NodePath
    "Platform/Sprite"           NodePath
    #if
    #region
    A<>
    AABB
    AcceptDialog
```

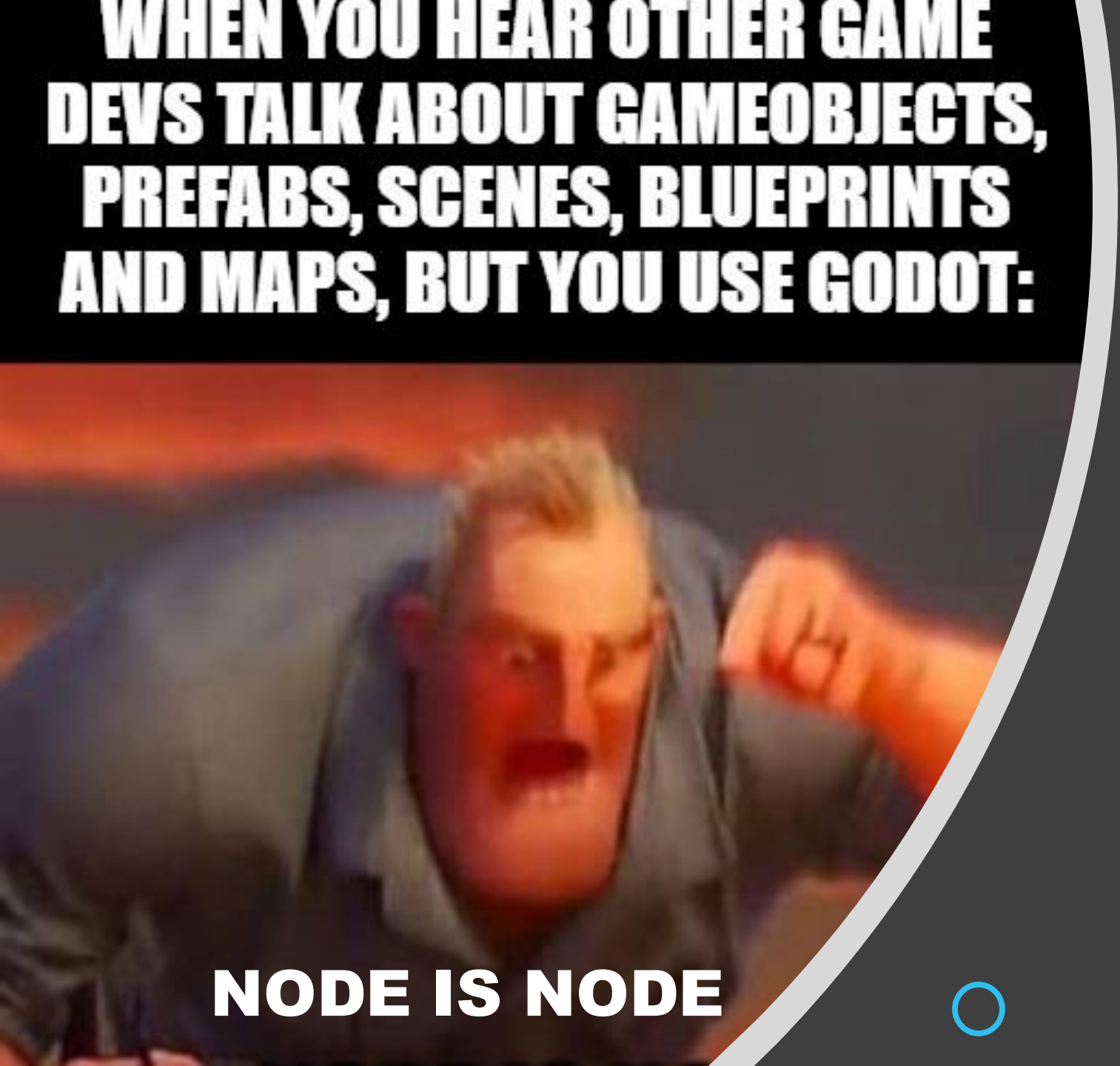
```
GetNode<Sprite>().Visible = true;
    "Camera"                    NodePath
    "Platform"                  NodePath
    "Platform/CollisionShape"   NodePath
    "Platform/Sprite"           NodePath
    #if
    #region
    AABB
    AcceptDialog
    AcceptEvent
    AccessViolationException
    Action<>
```



GDScript

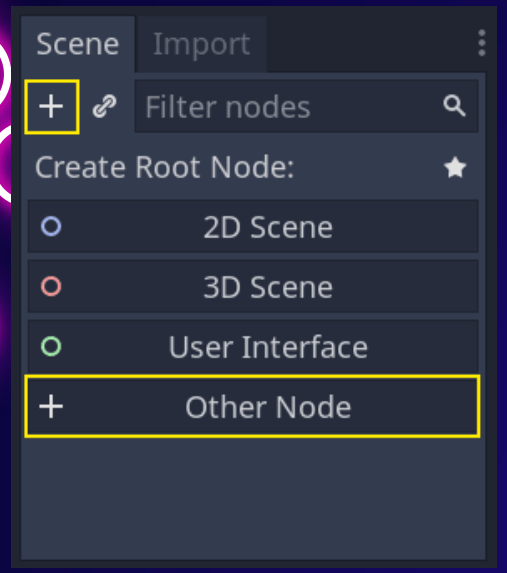
Visual Script



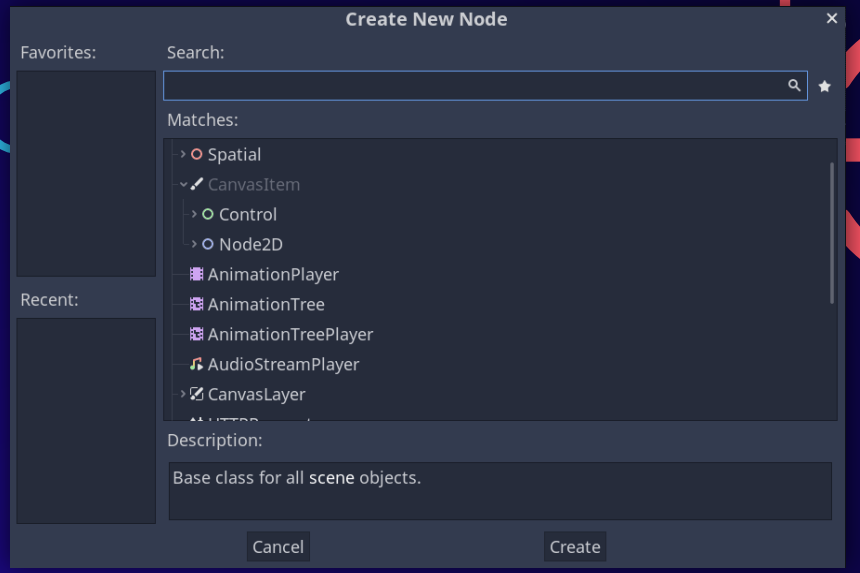


Nodes & Scenes

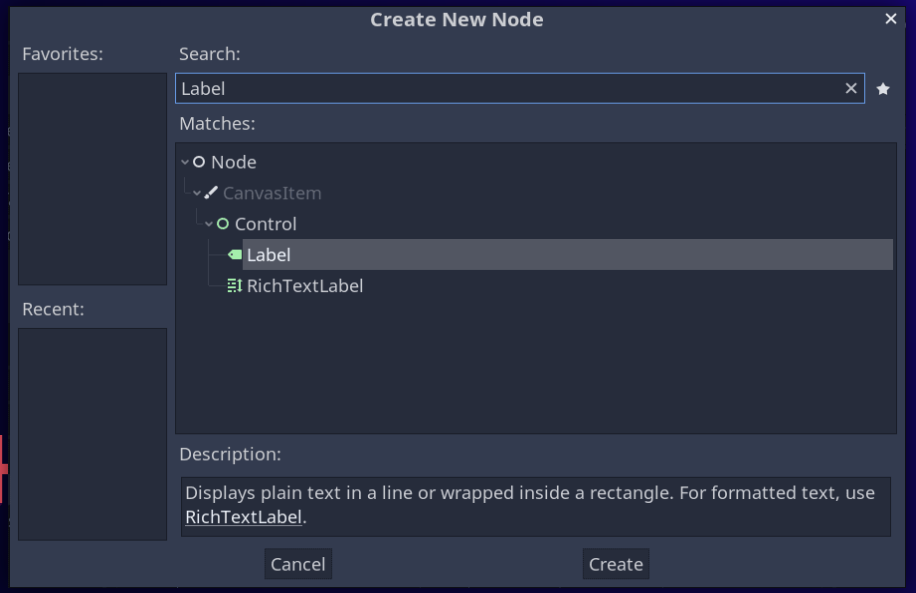
Criando uma cena



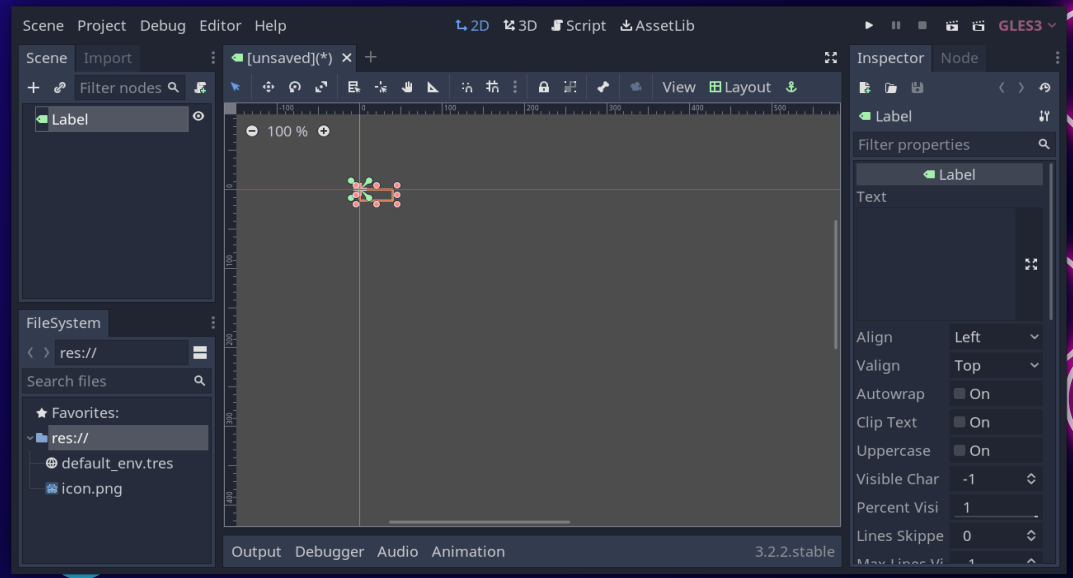
Selecionar nó

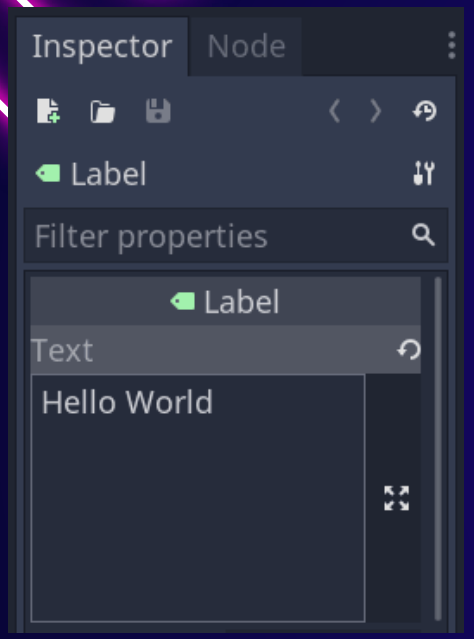


Escolher nó "Label"

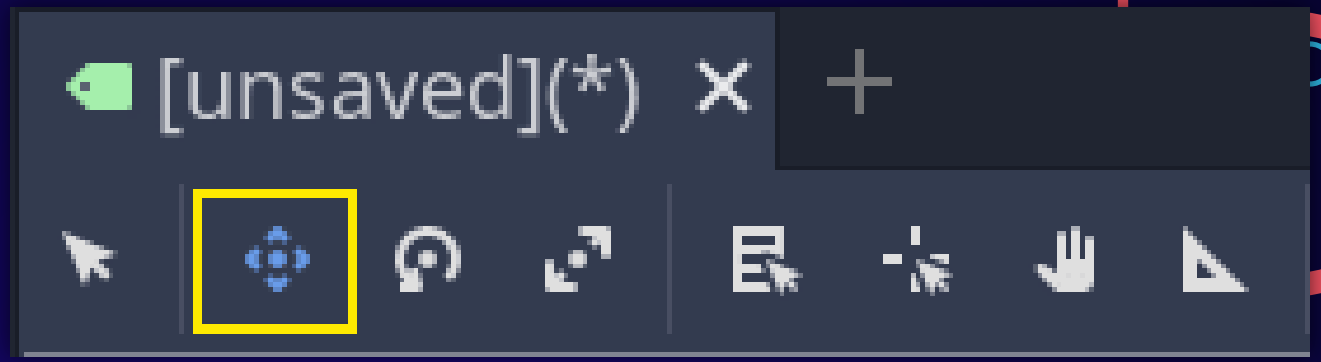


Label inserida

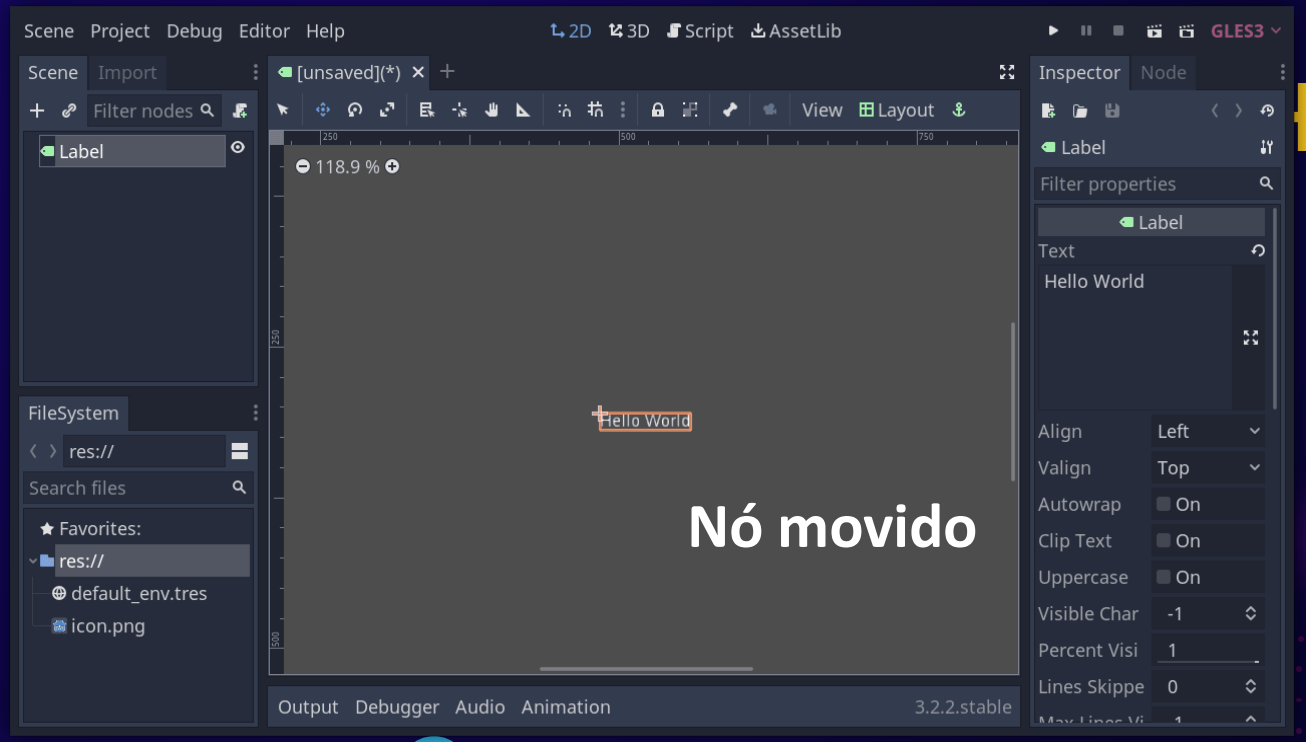




Propriedades do nó (Text é um atributo do tipo String)

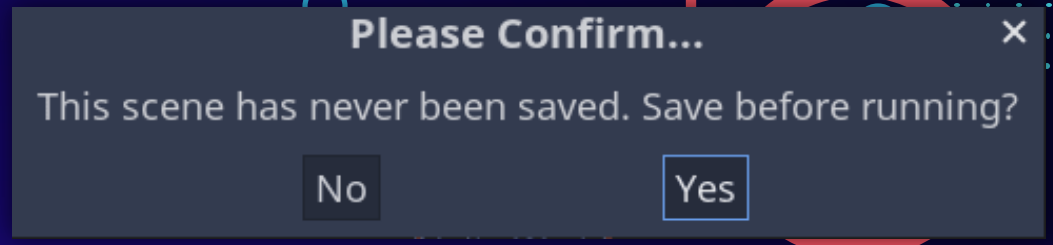


Ferramenta de movimentação

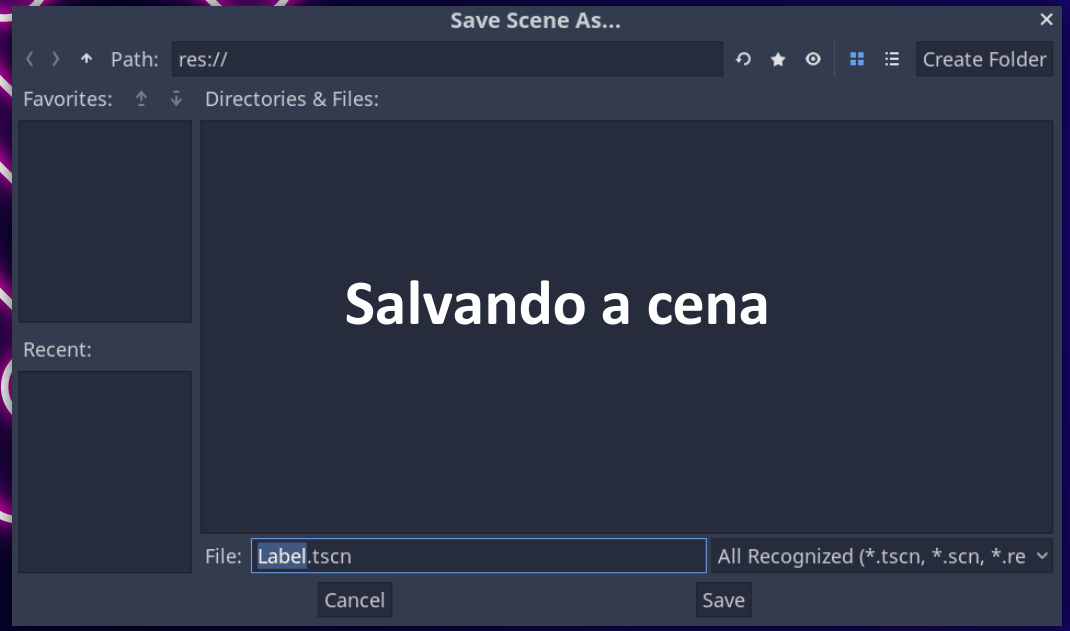


Nó movido

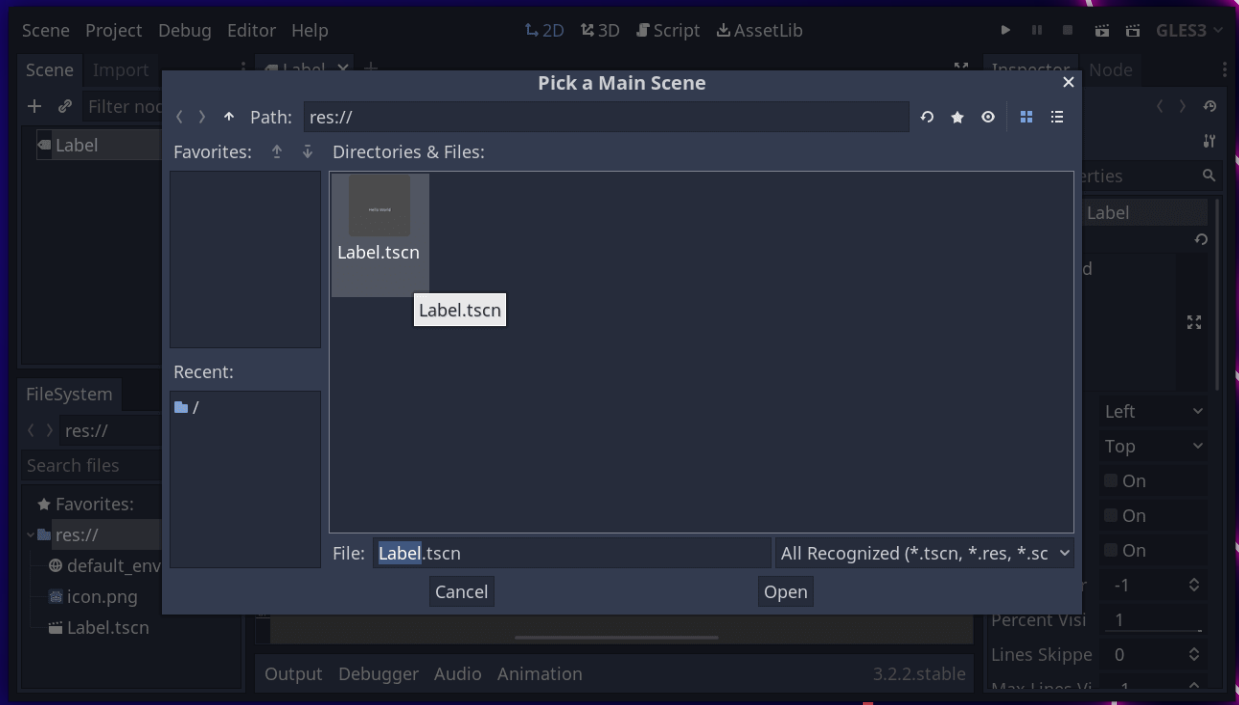
Executar cena



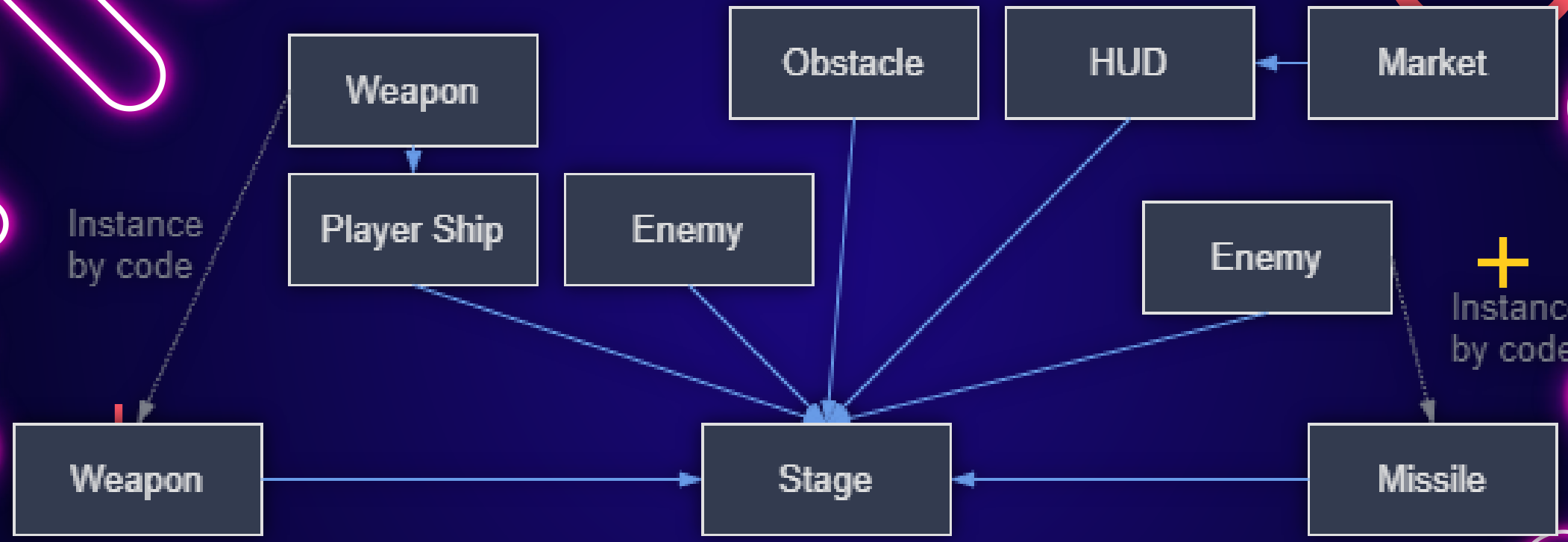
É necessário salvar a cena antes



Salvando a cena

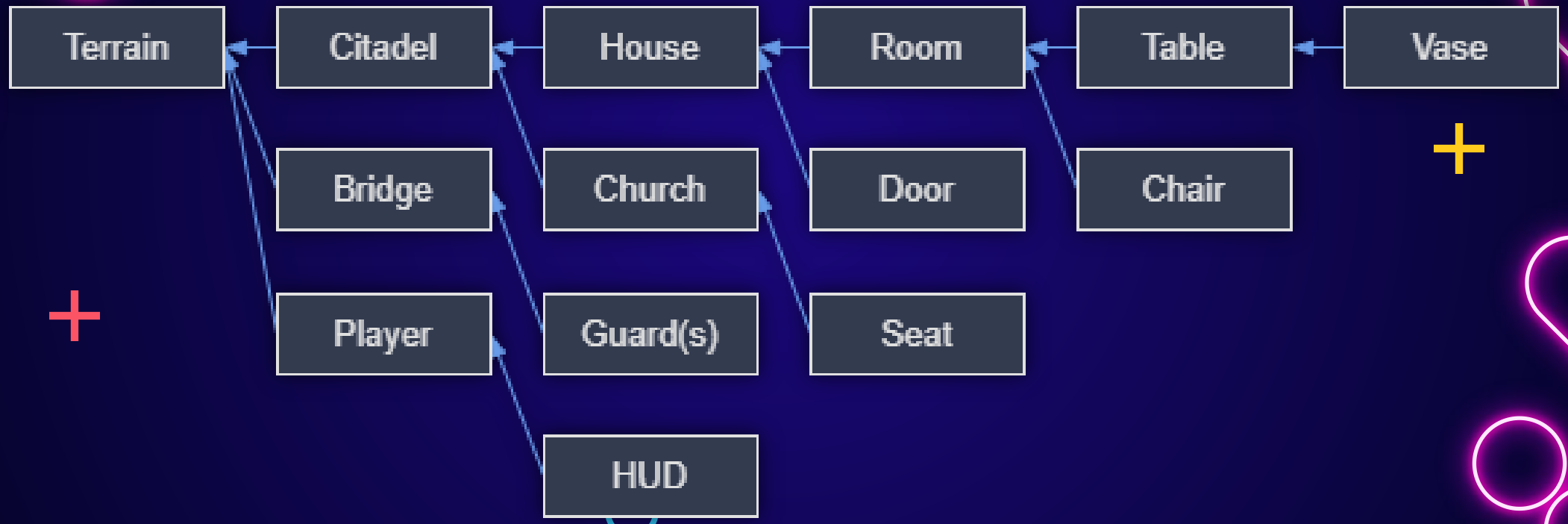


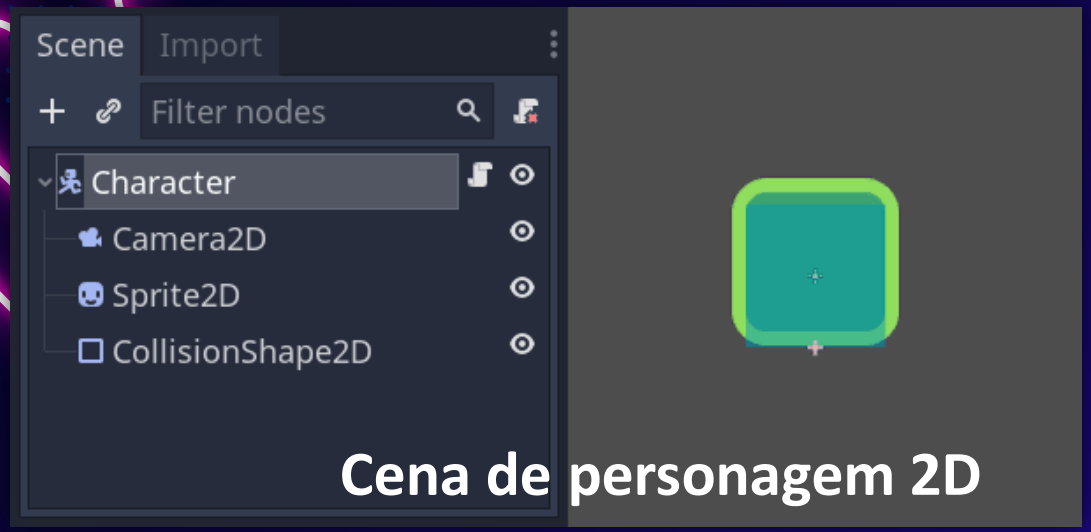
Usamos a cena salva como a principal



Uma árvore com instâncias de nós forma uma cena

A hierarquia das cenas é uma forma de agregação que permite projetar níveis mais intuitivamente

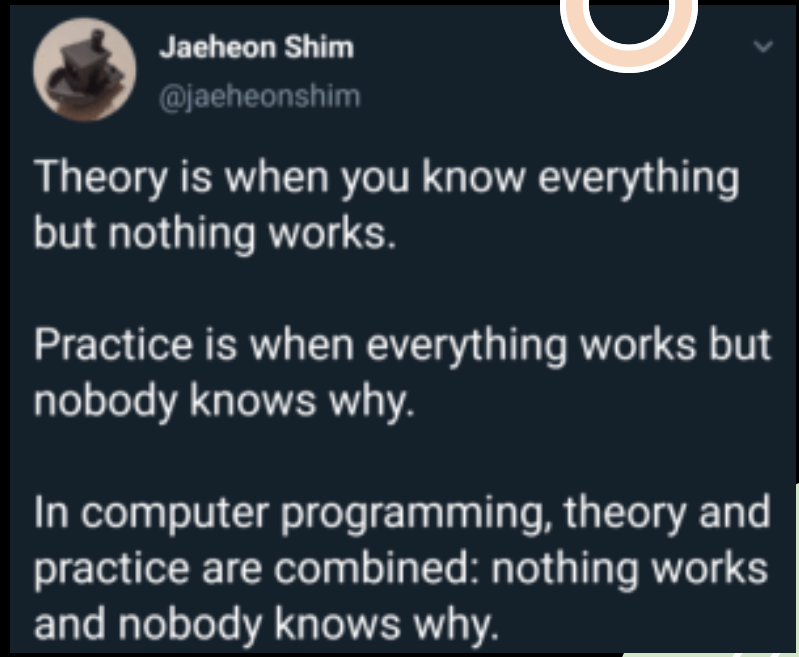




Cena de personagem 2D

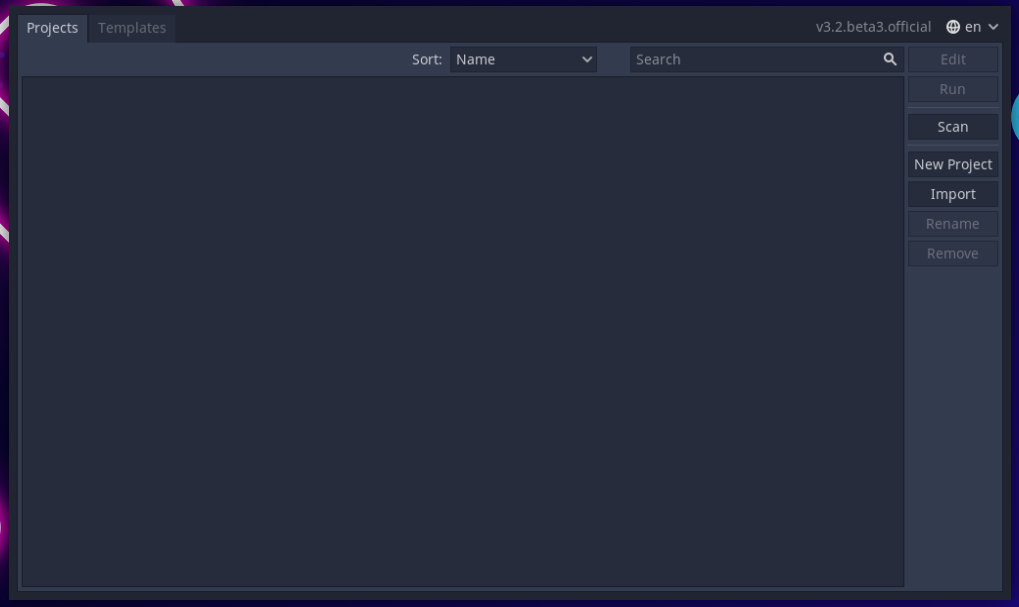
Cena de personagem 3D

Prática

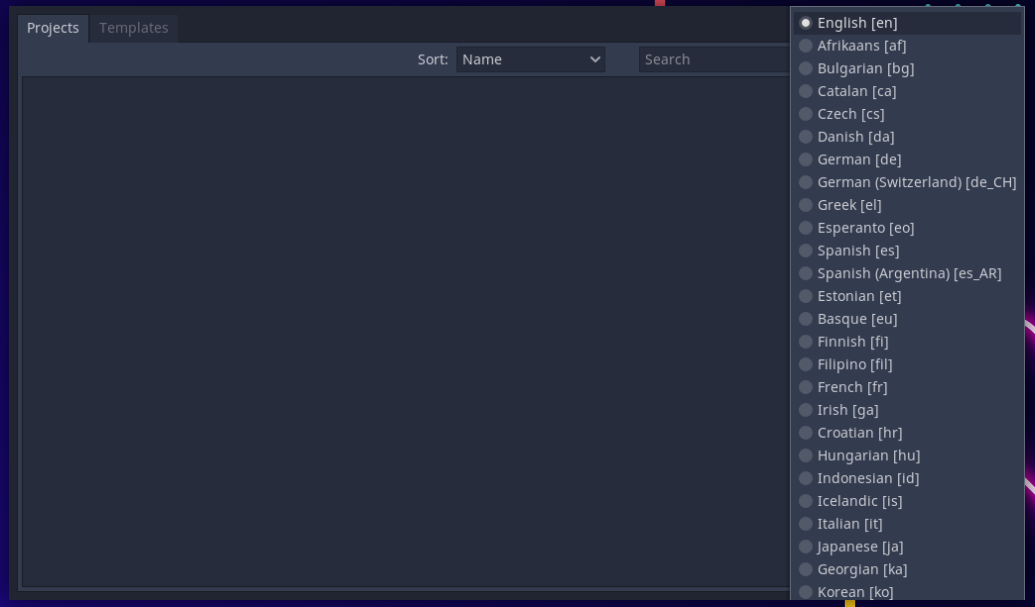

Jaeheon Shim
@jaeheonshim
Theory is when you know everything but nothing works.
Practice is when everything works but nobody knows why.
In computer programming, theory and practice are combined: nothing works and nobody knows why.

Editor

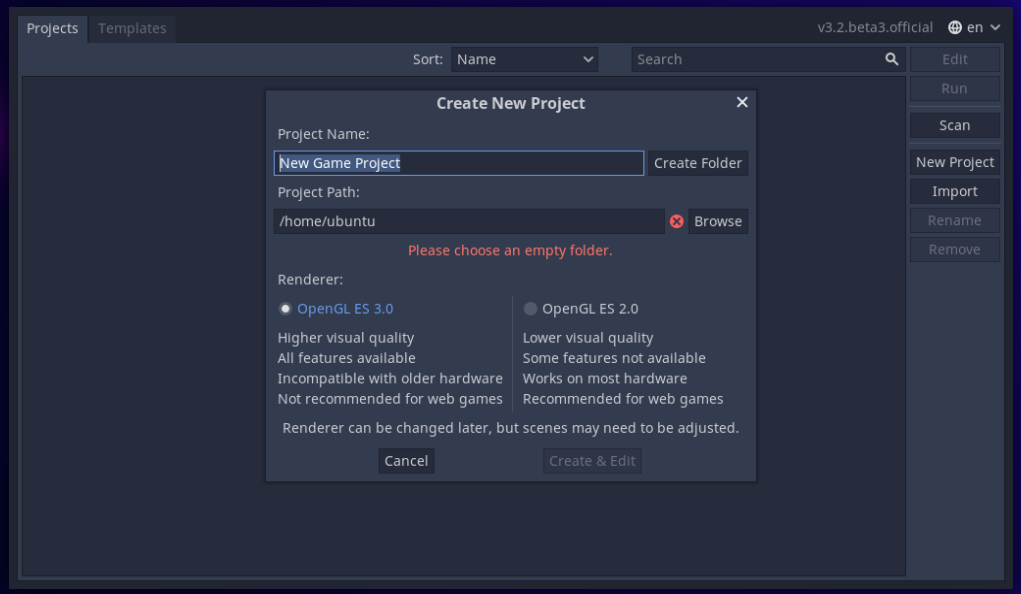
Gerenciador de projetos



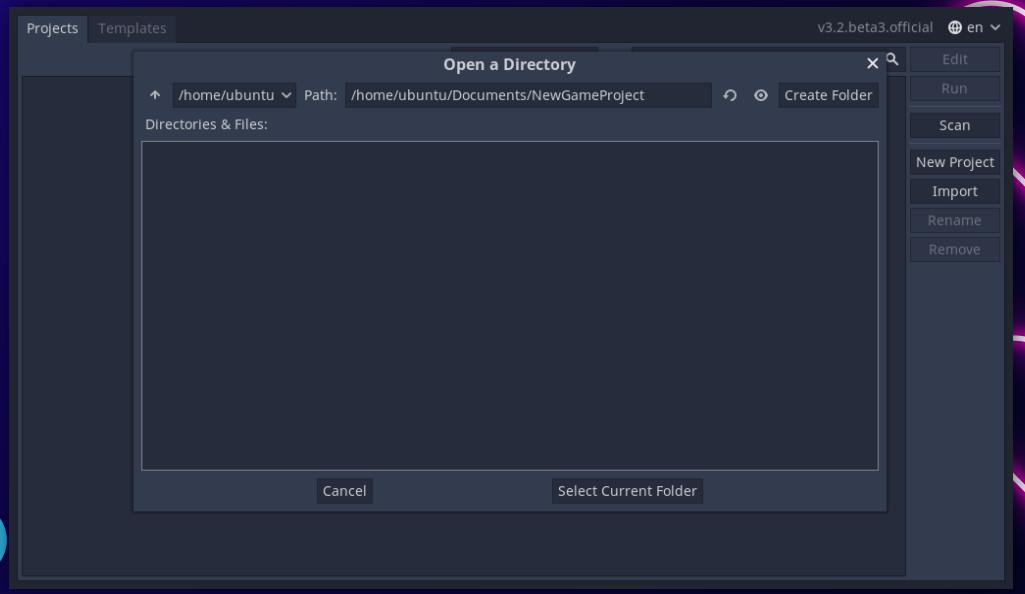
Seleção de idiomas



Novo projeto



Navegador de arquivos



Templates & demos

Projects | Templates

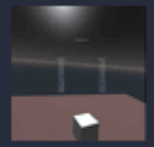
v3.2.beta3.official | en

Search

Sort: Recently Updated | Category: All | Site: godotengine.org | Support

First | Previous | 1 | 2 | 3 | 4 | 5 | 6 | Next | Last

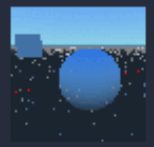
FirstPersonStarter
Templates
Whimfoome
MIT



Oculus Quest Button Testing
Projects
creikey
MIT

Mannequiny: Open 3D Mannequin
Templates
gdquest
MIT

Inventory System Example
Demos
Oen44
MIT



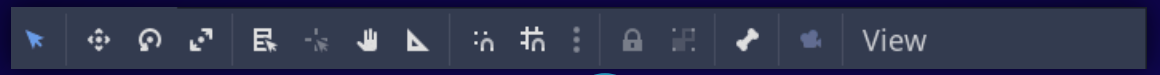
RigidBody Planetary Physics Correct Stand Up
Demos
ComancheAk
MIT

Top Down Twin Stick Shooter
Templates
RedSlimeSkirt
MIT

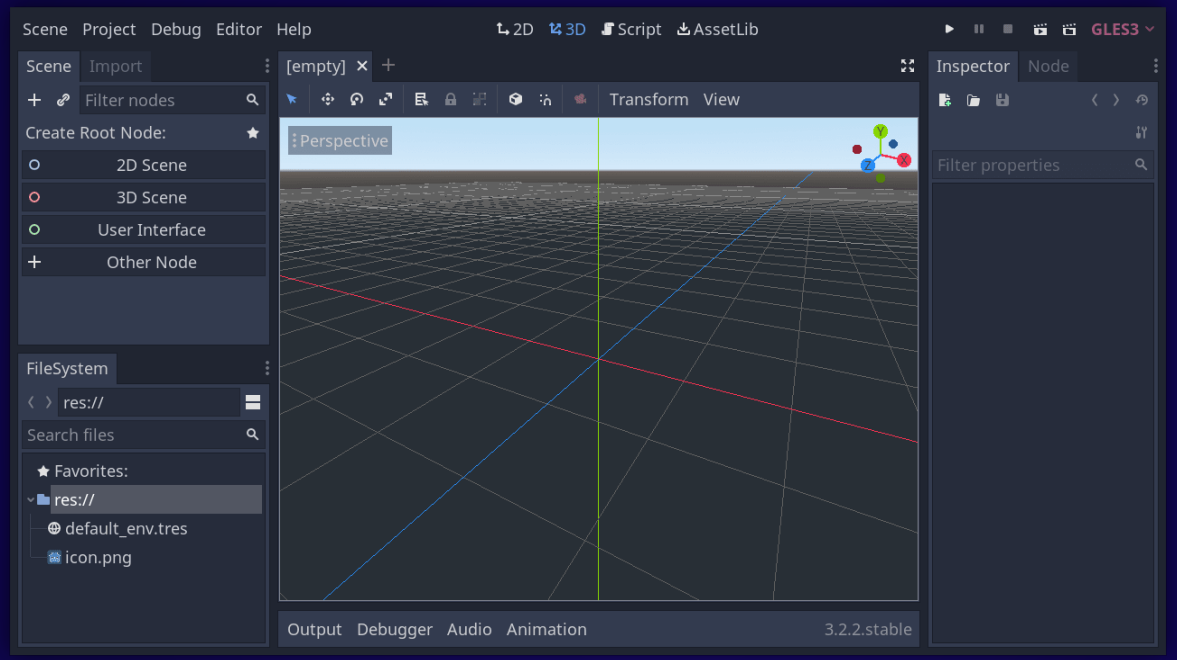
Controller use Template
Templates
Ange

Speed feeling Godot 2D
Templates
Ange

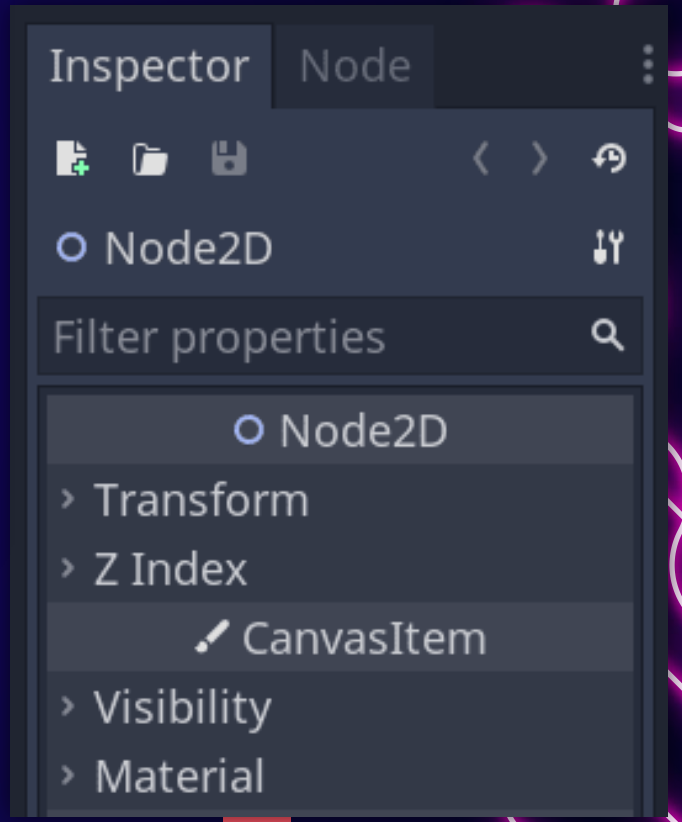
Barra de ferramentas



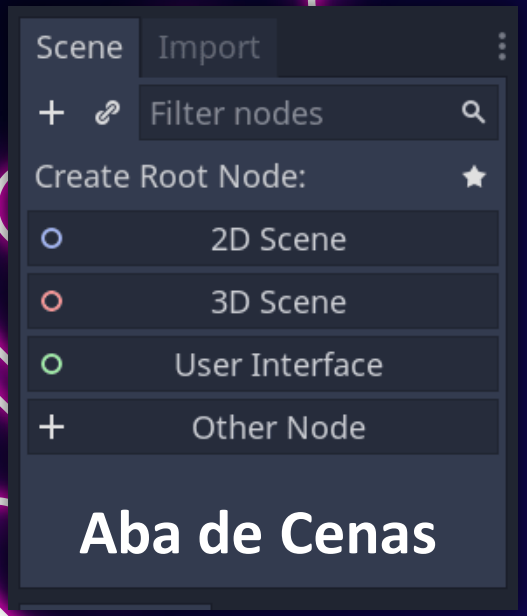
Tela do editor



Inspetor

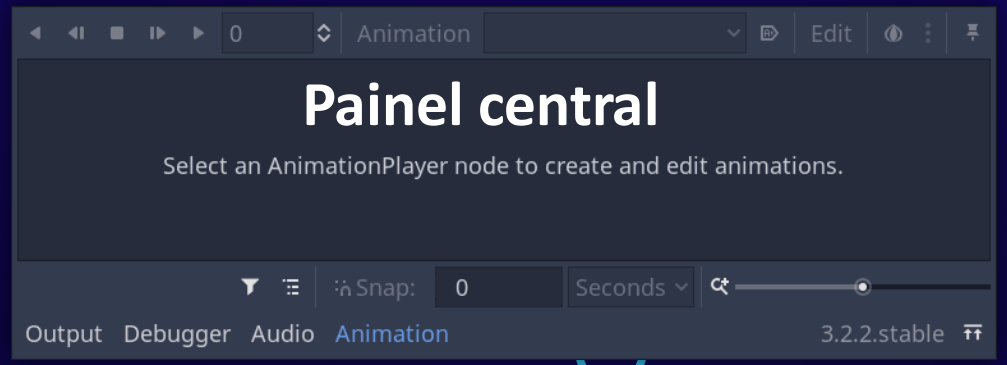


(aba de propriedades)

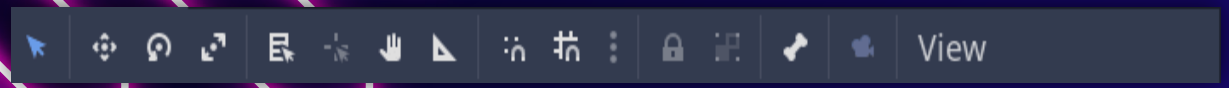


Aba de Cenas

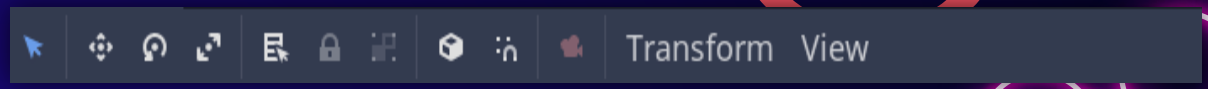
Arquivos



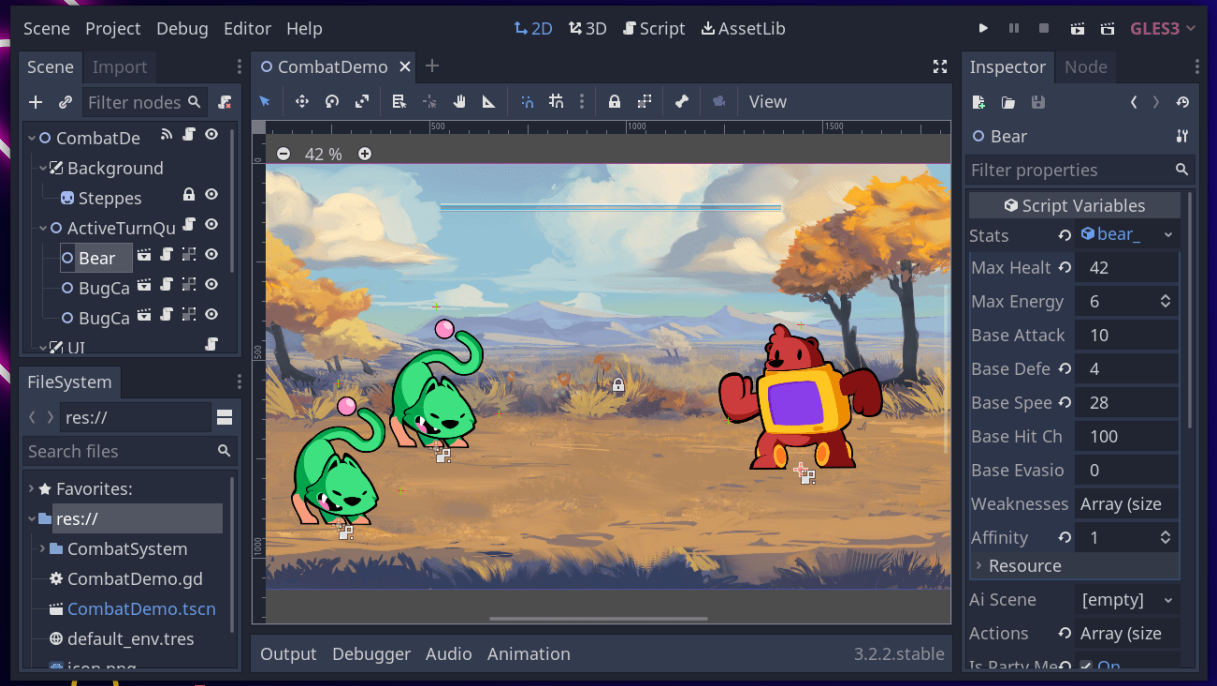
Barra de ferramentas 2D



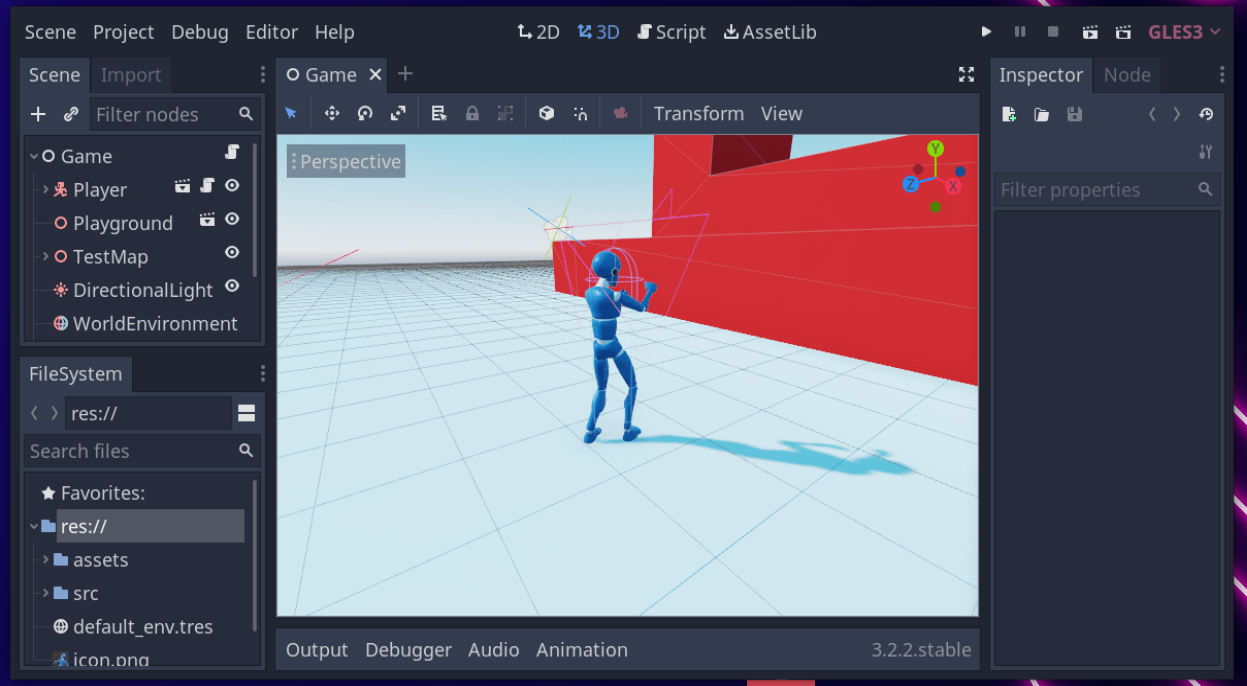
Barra de ferramentas 3D



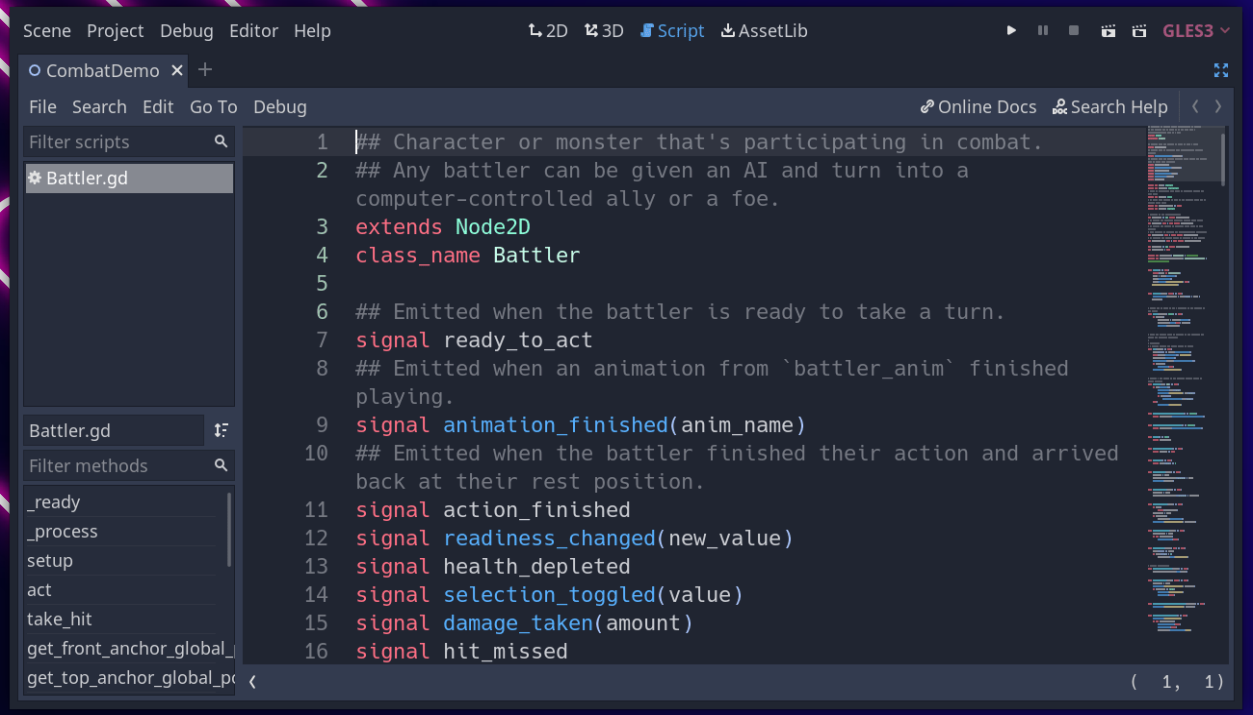
Editor 2D



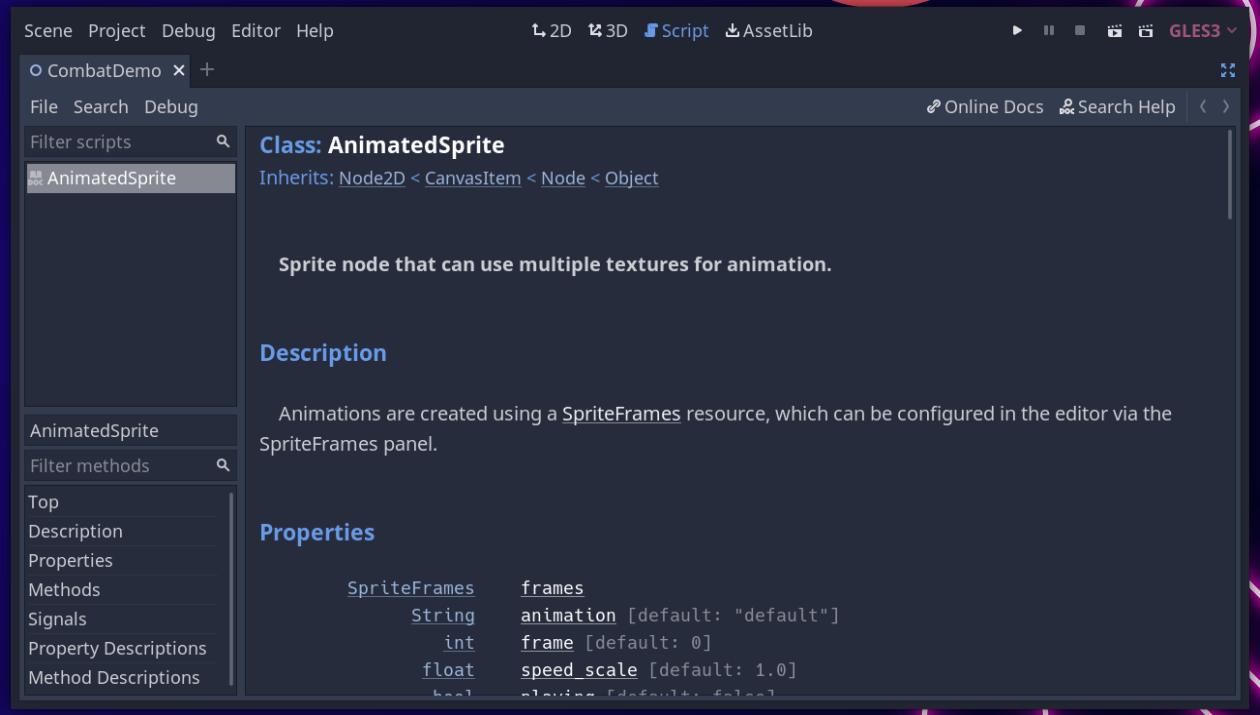
Editor 3D



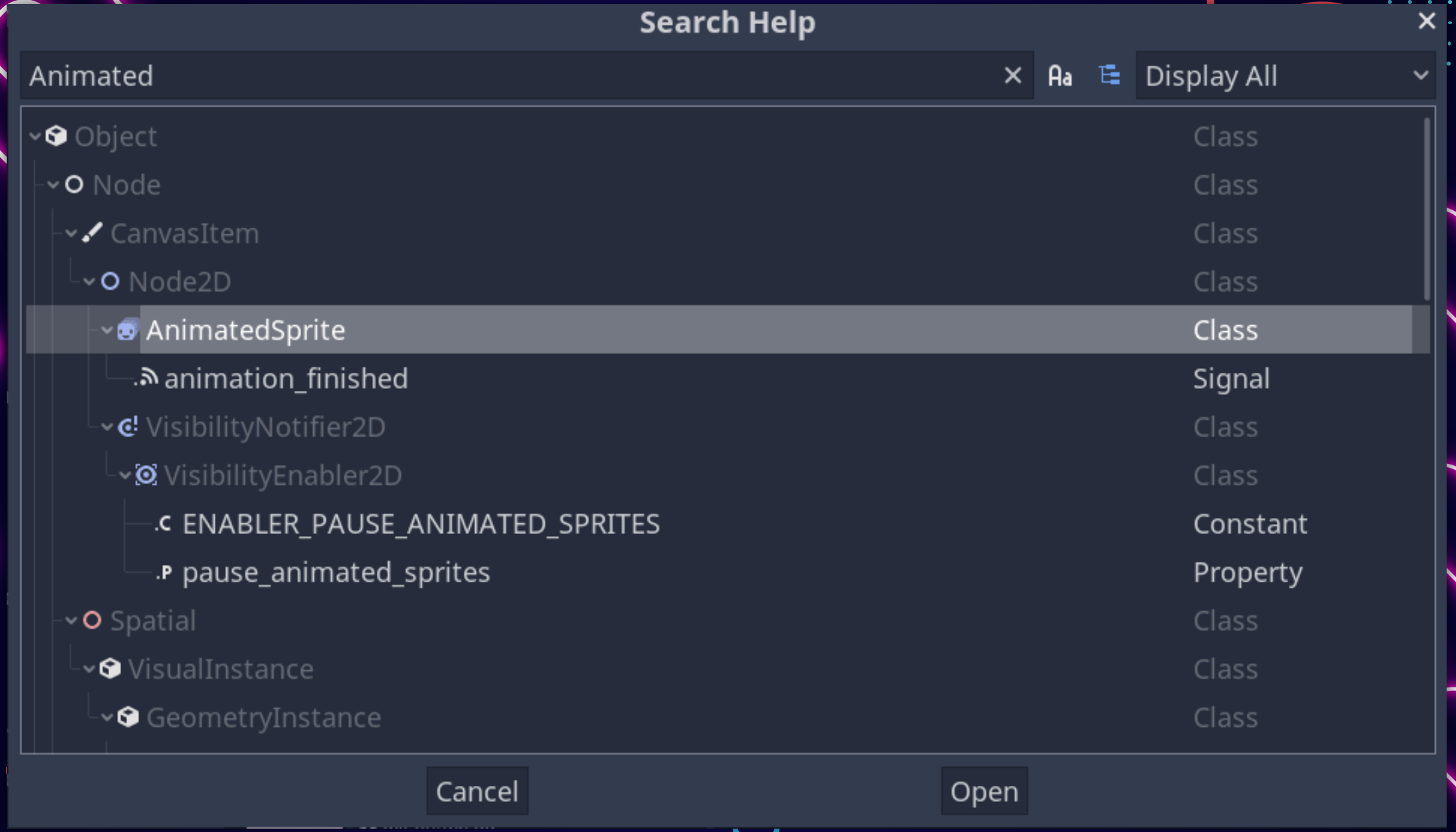
Editor de texto



Documentação



Janela de ajuda

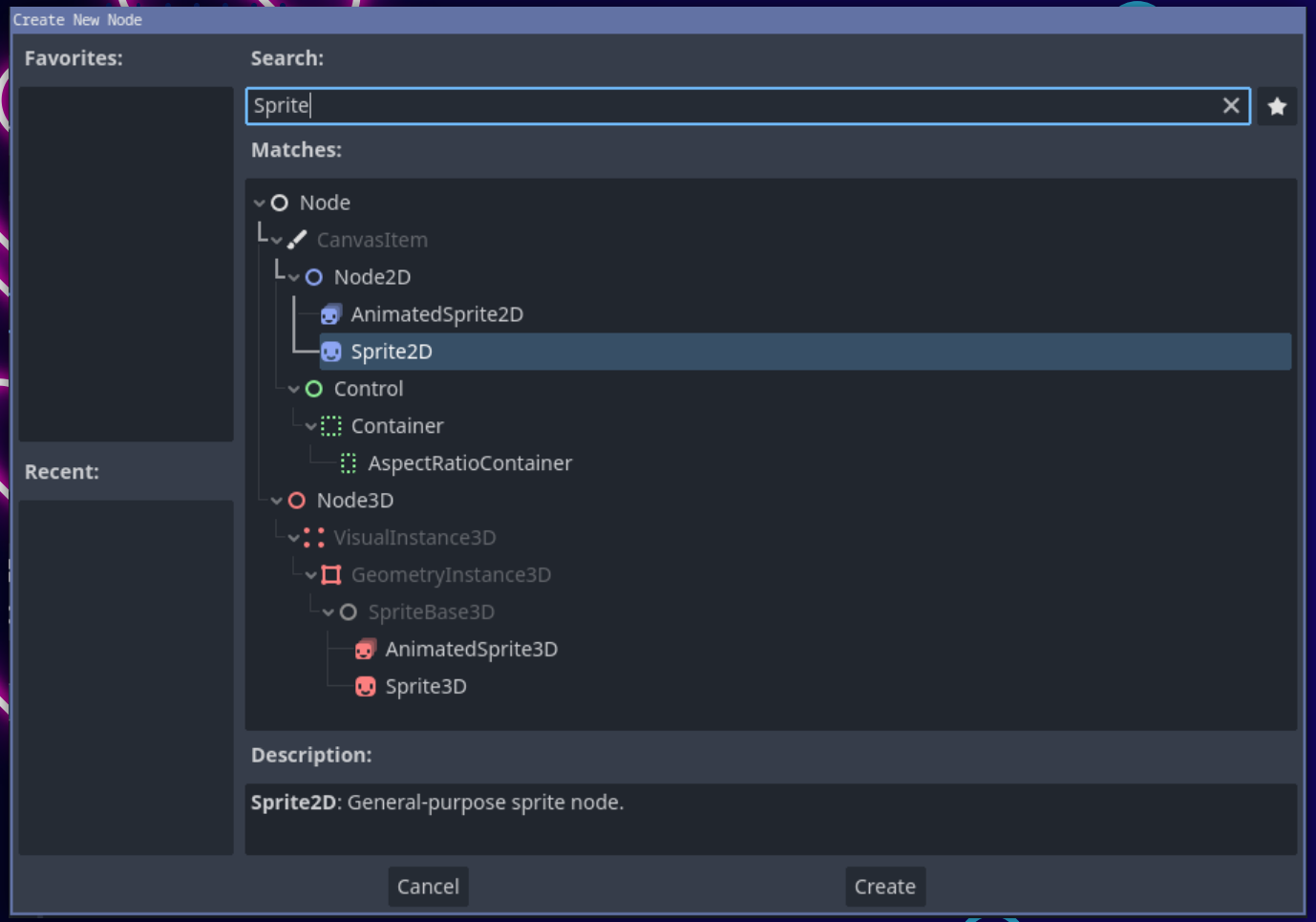


Imagens

Cenário

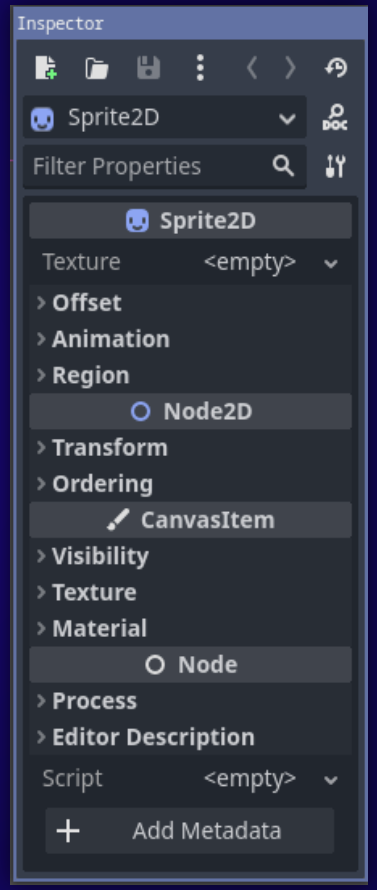
OpenGameArt.org

Demo

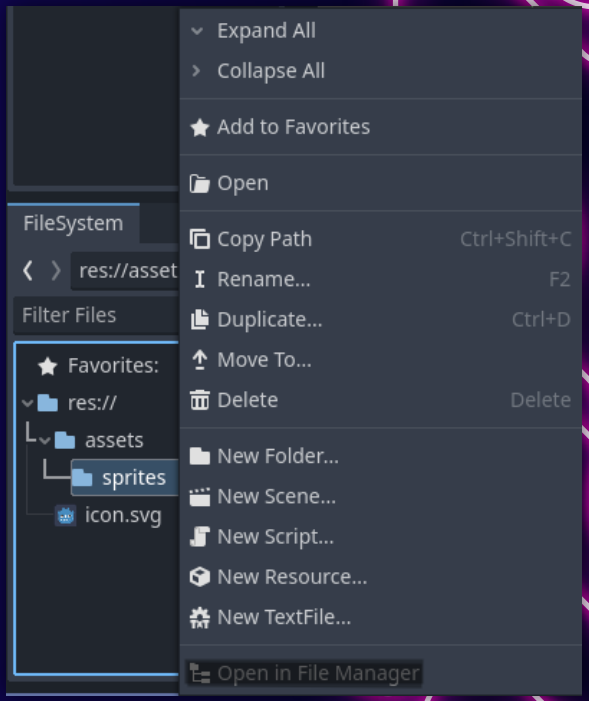


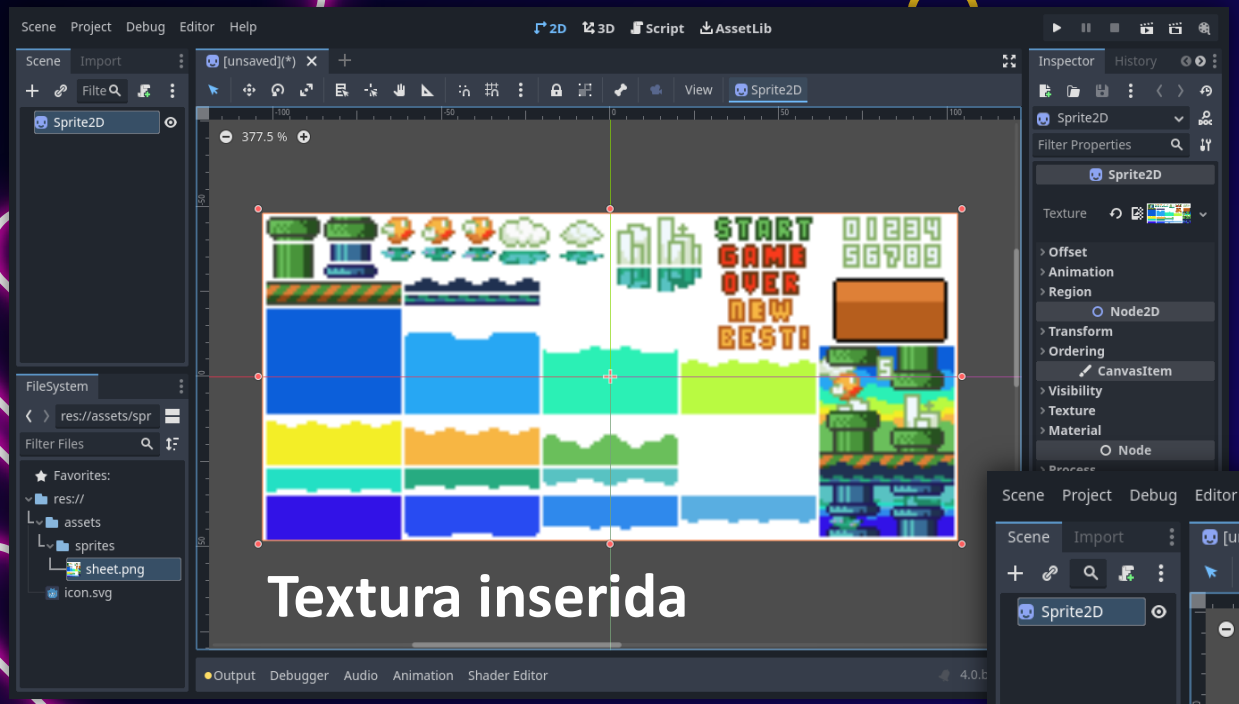
Selecionar nó "Sprite2D"

Inspetor



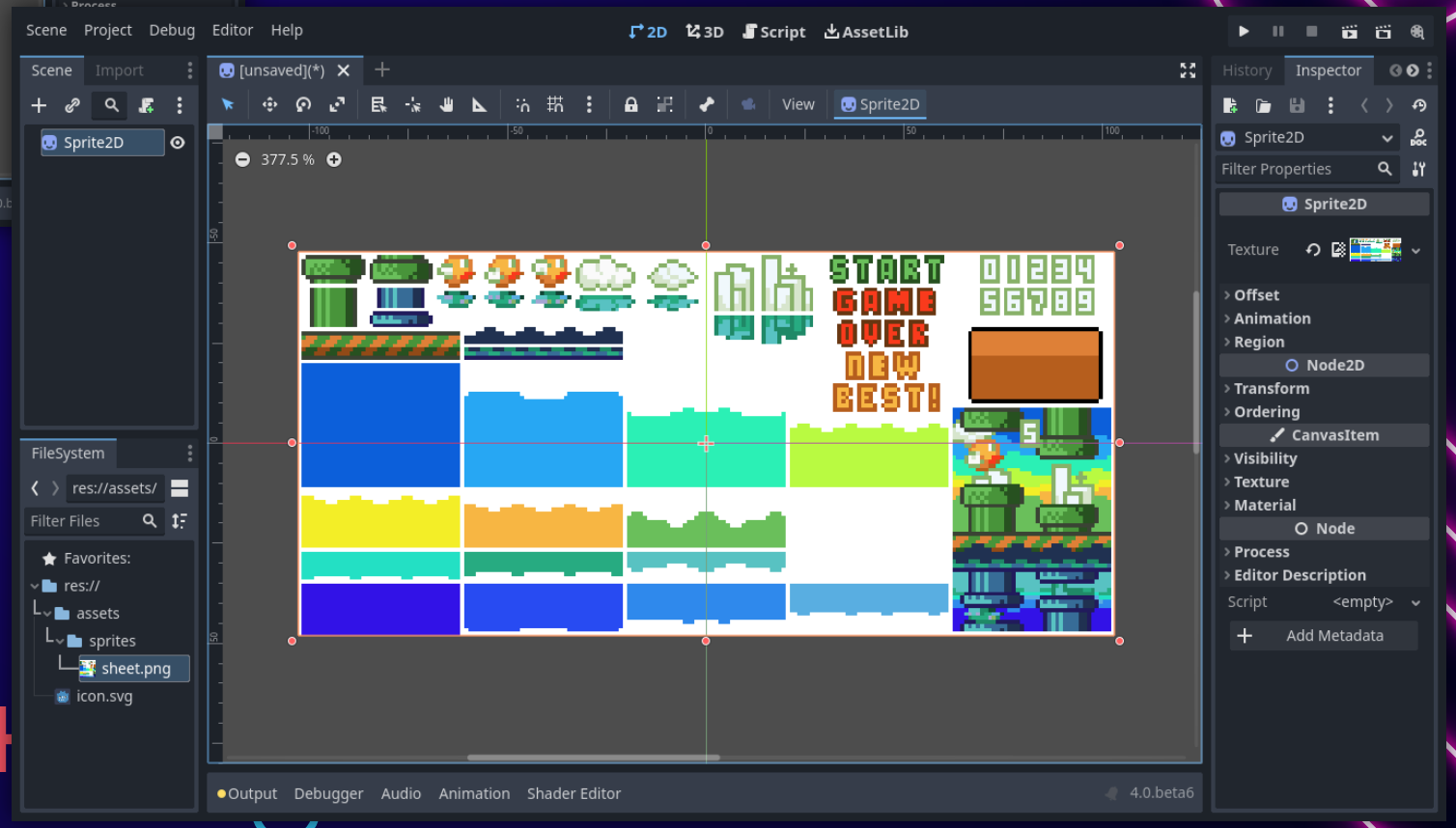
Mover textura

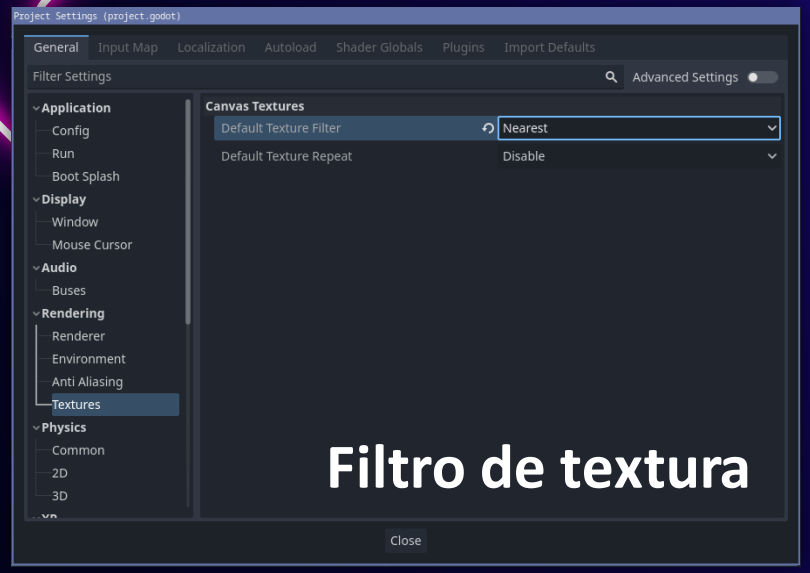




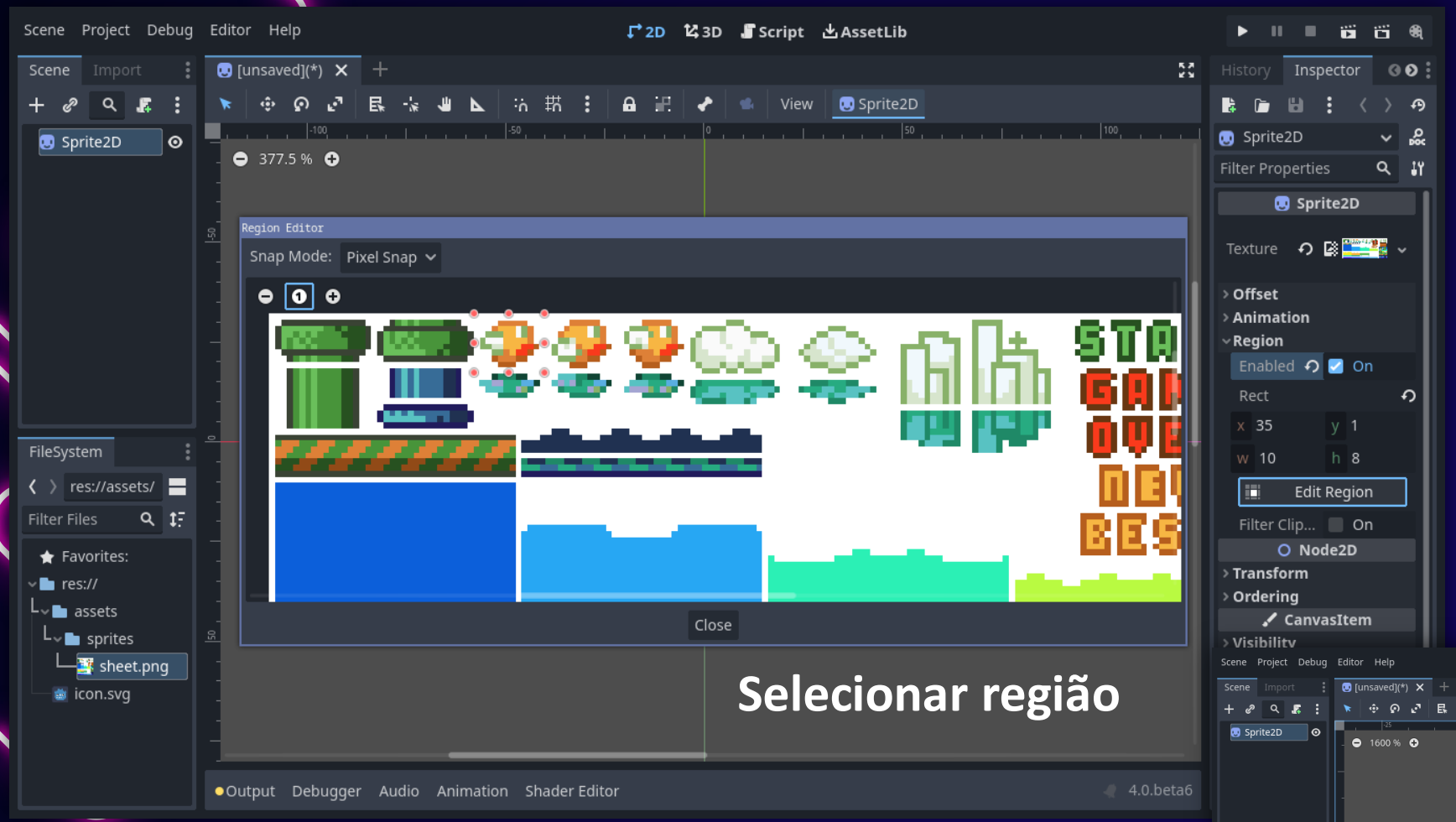
Textura inserida

"Nearest"

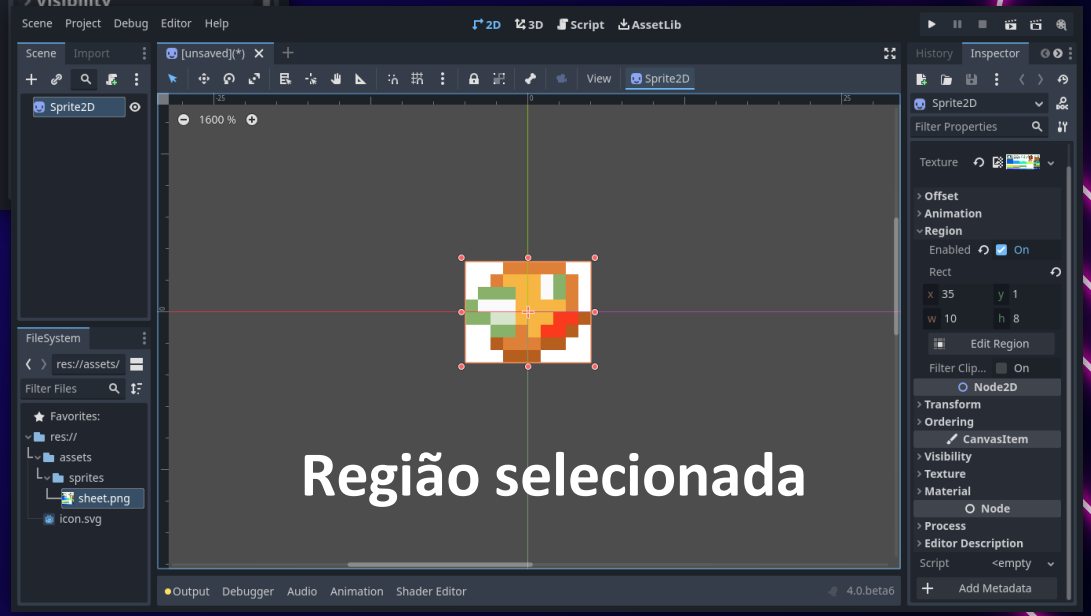




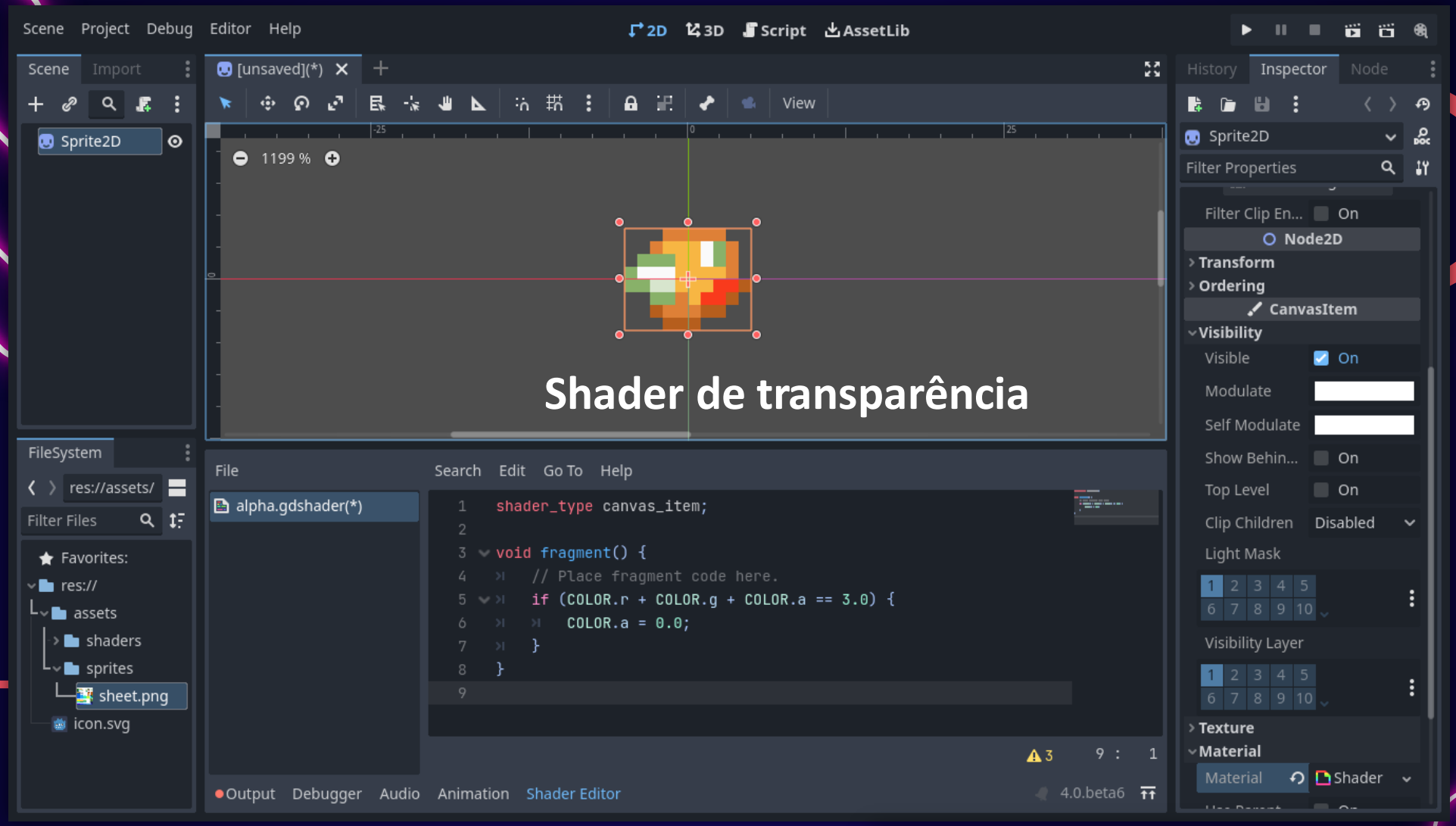
Filtro de textura



Selecionar região

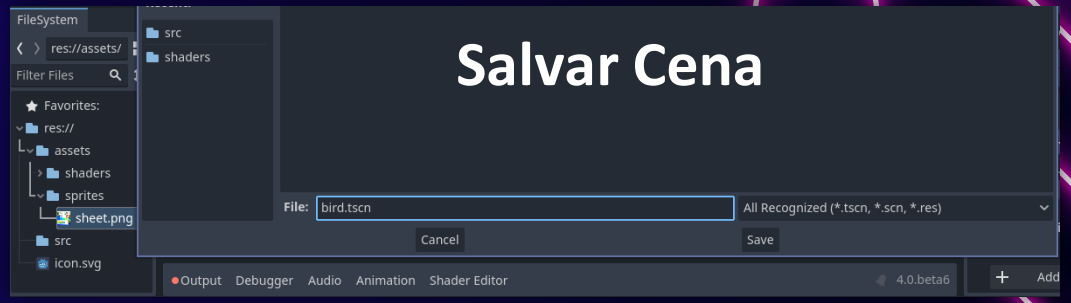


Região selecionada

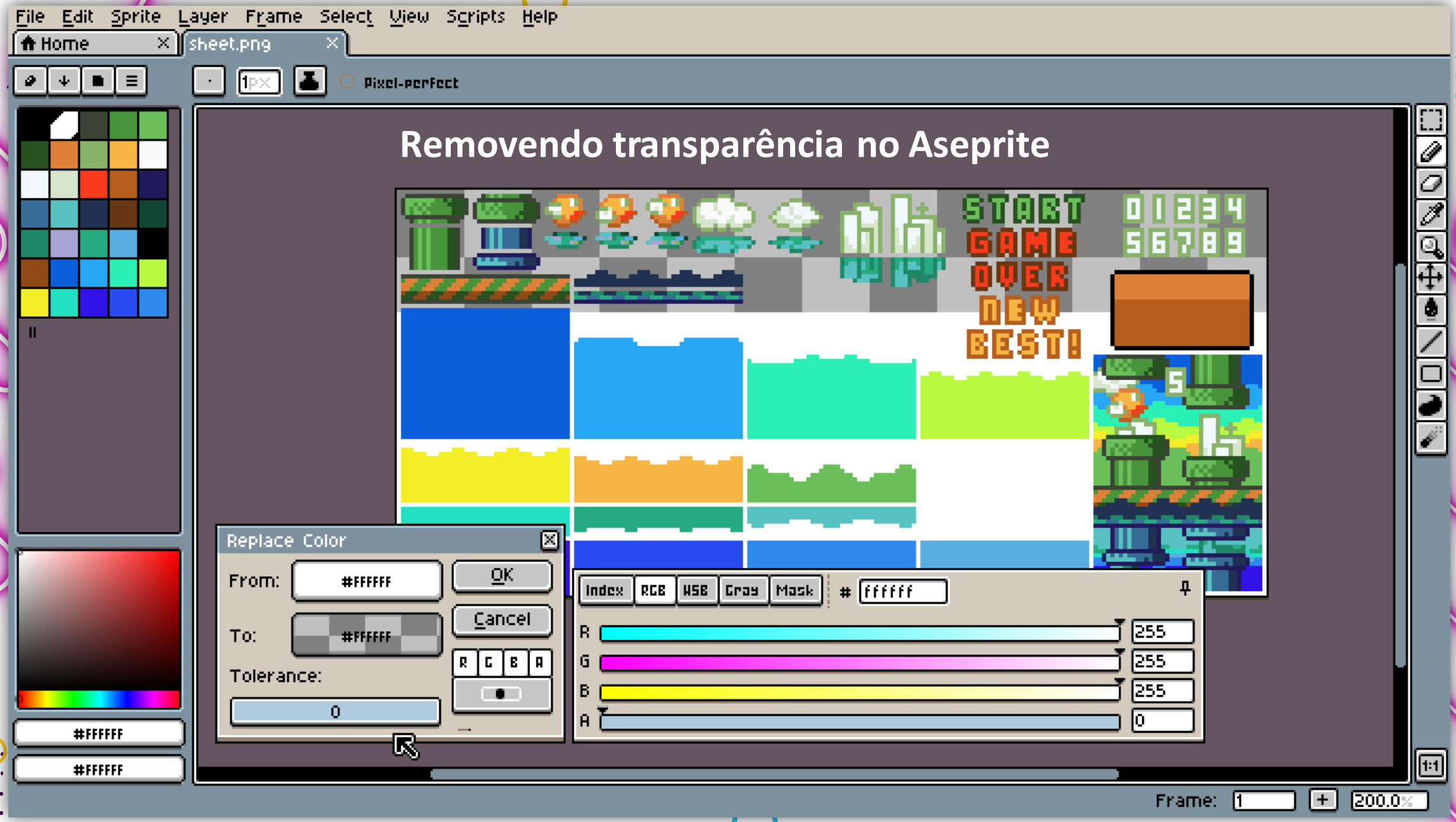


Renomear

Salvar Cena



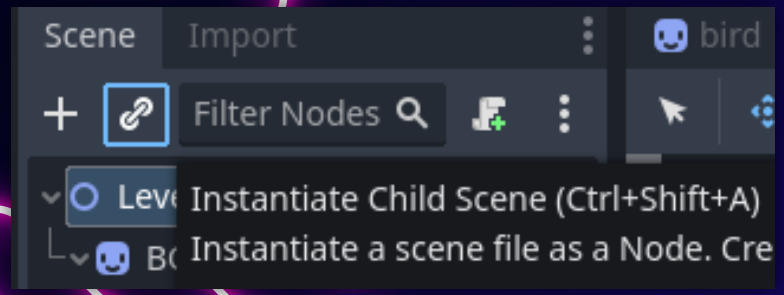
Removendo transparência no Aseprite



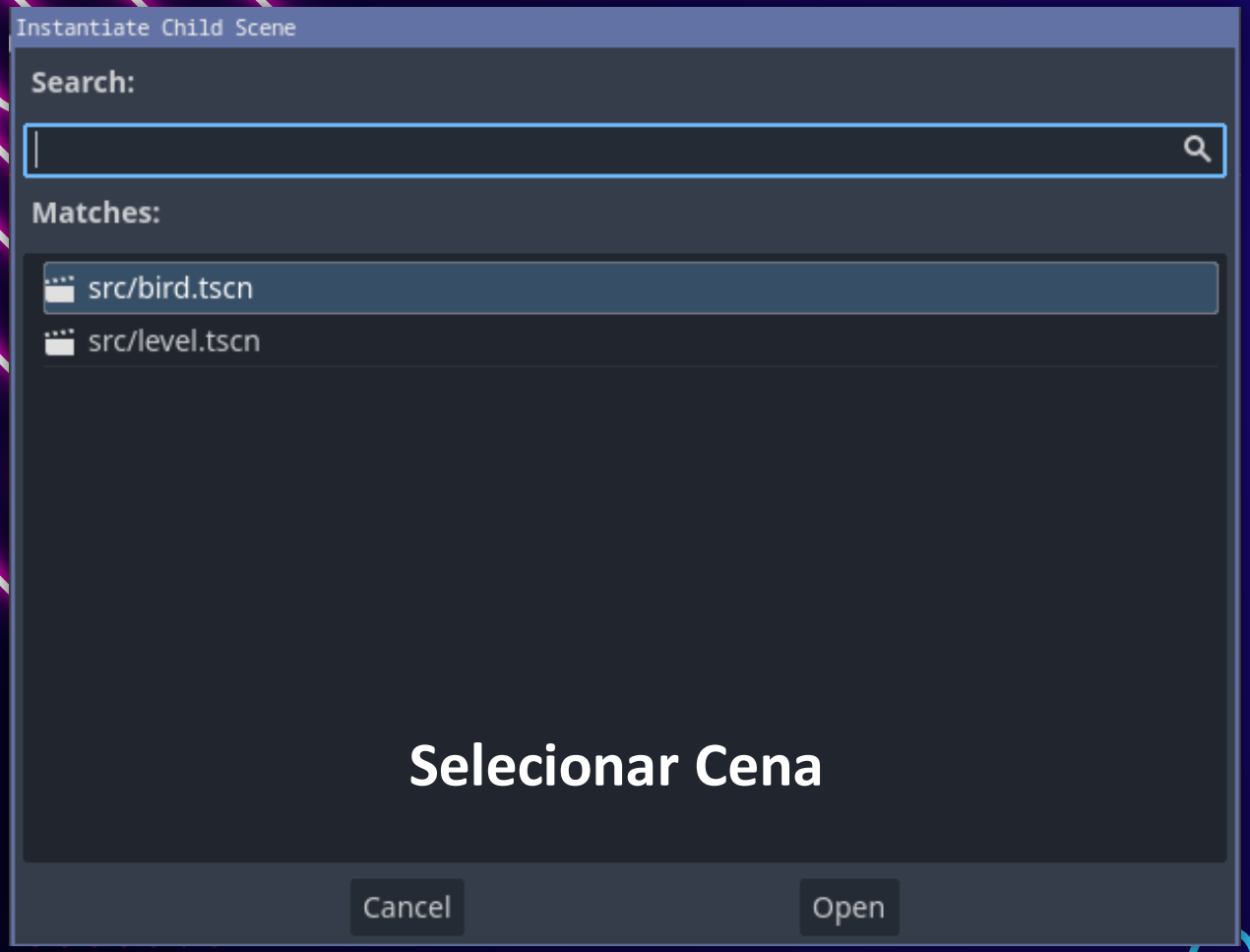
Nova Cena

Deslocar o centro

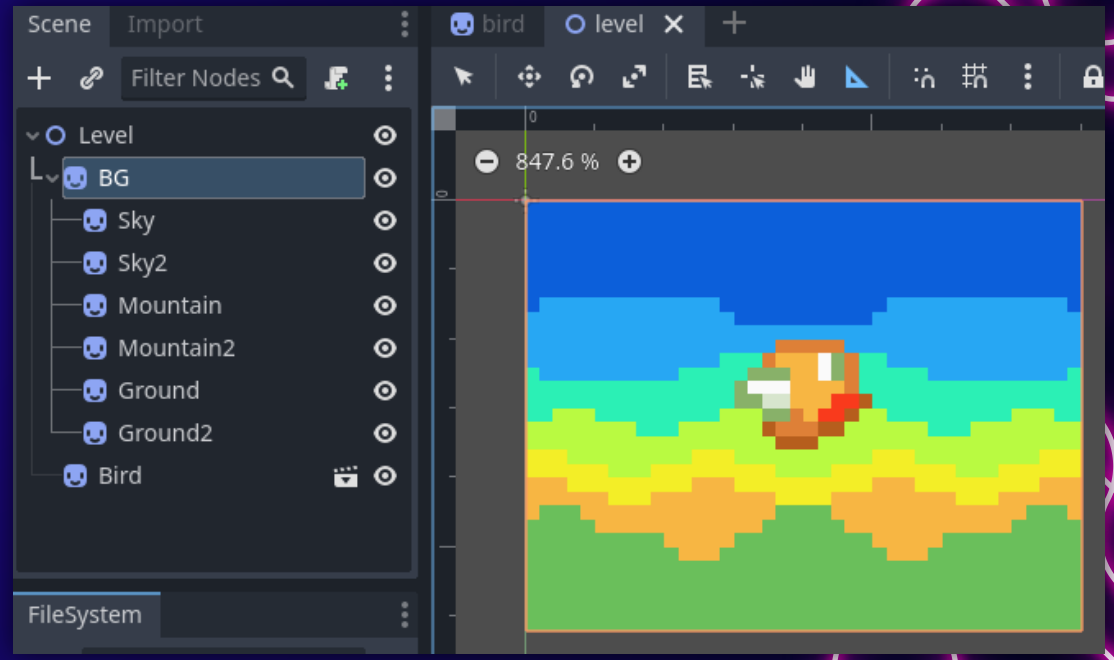
Background



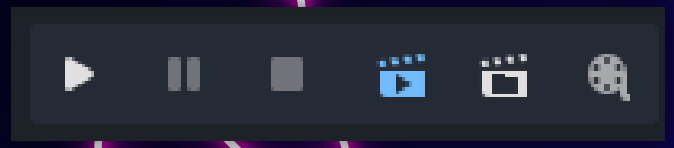
Instanciar cena como nó filho



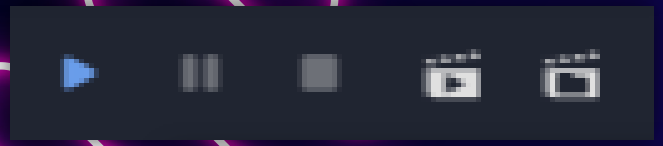
Selecionar Cena



Ajustar Cena

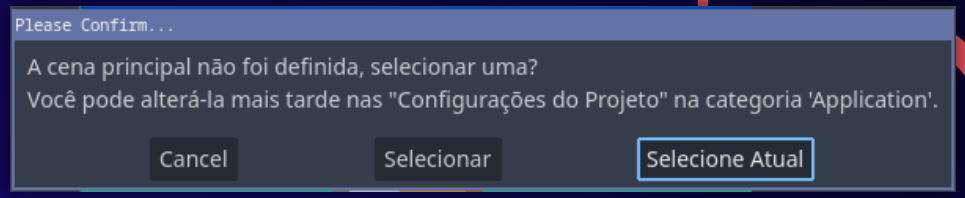


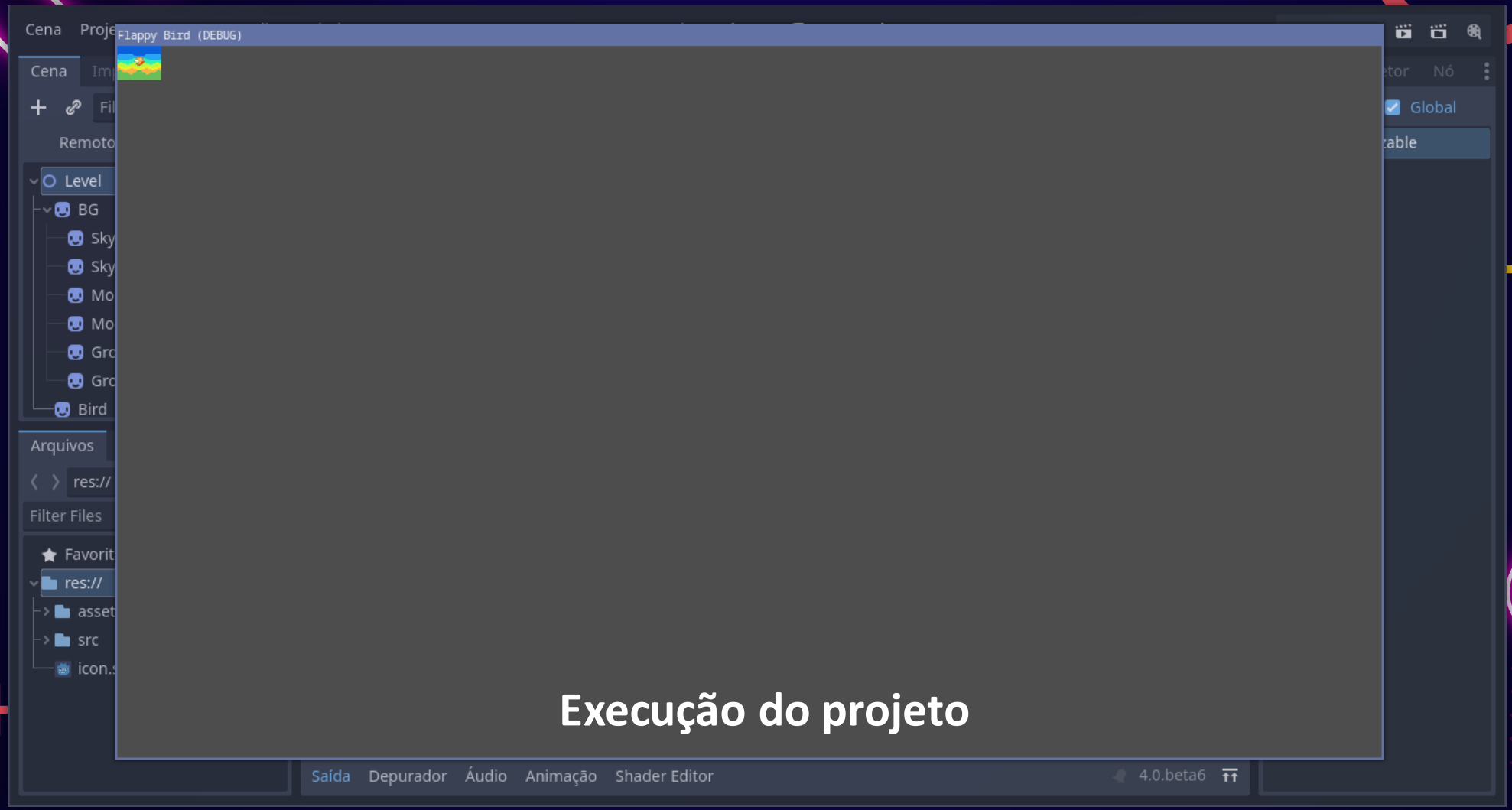
Executar cena



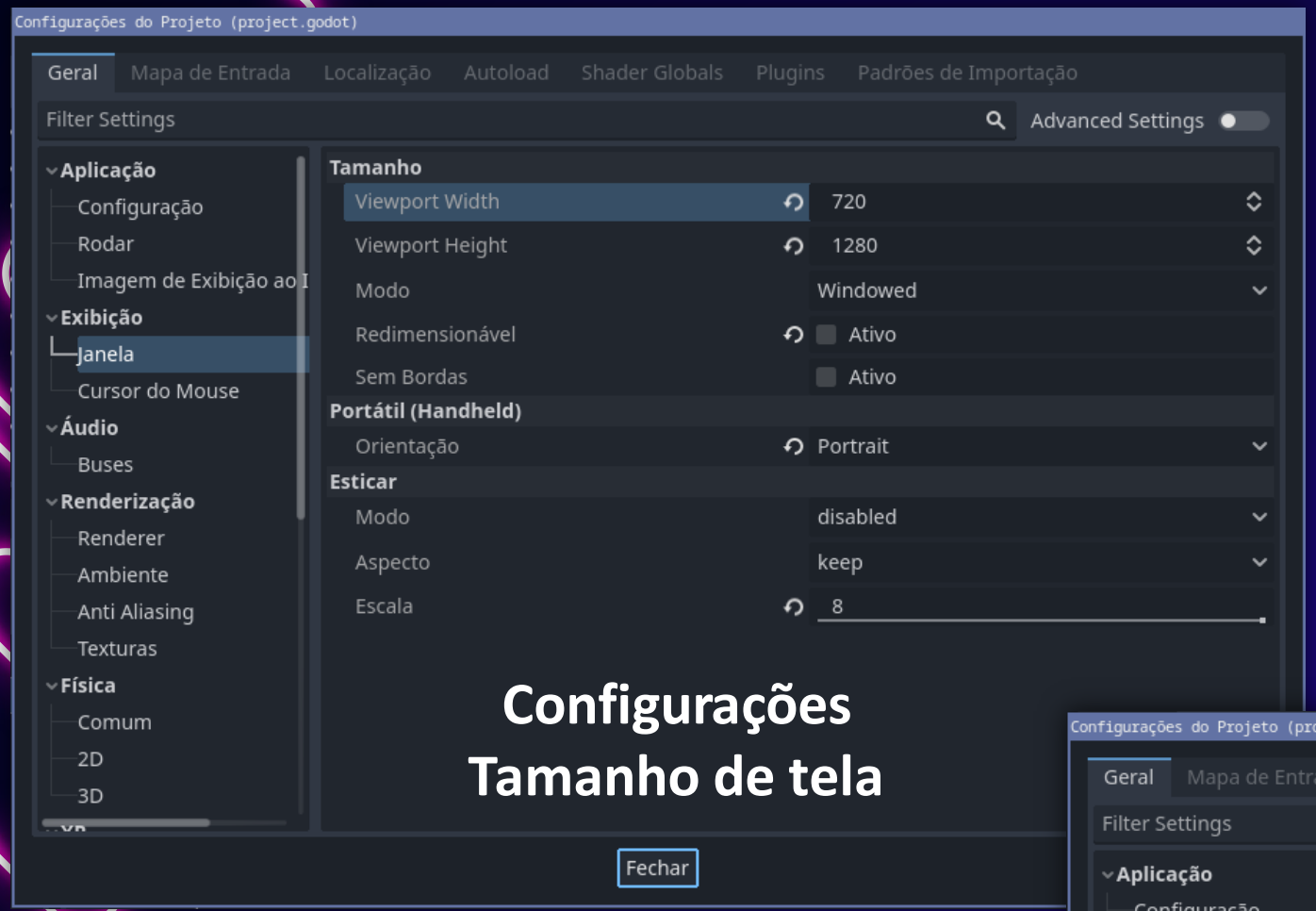
Executar projeto

Selecionar Cena principal

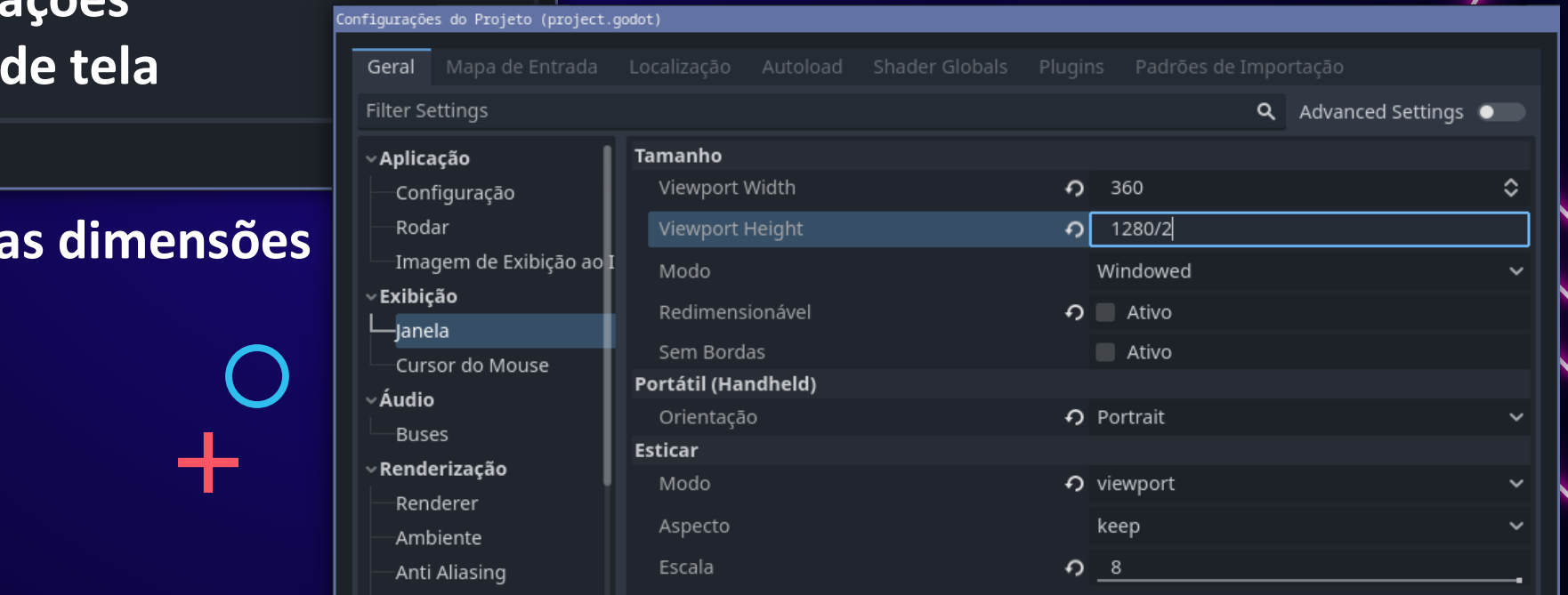




Execução do projeto



Configurações
Tamanho de tela



Novas dimensões

DeviceAtlas Products Solutions

The following table shows which screen resolution is most used in selected countries based on web traffic statistics for **Q2 2019**.

Country	Screen resolution	Share	Change (v Q1)
Argentina	720x1280	33.31%	-1.36%
Australia	750x1334	22.03%	-1.80%
Brazil	720x1280	22.52%	-6.86%
Canada	750x1334	24.77%	-1.33%
Colombia	1080x1920	19.10%	+0.18%
Denmark	750x1334	28.01%	-3.64%
Egypt	720x1280	21.36%	-5.68%
Finland	1080x1920	24.54%	+2.21%
France	750x1334	19.54%	-2.94%
Germany	1080x1920	17.89%	-1.74%

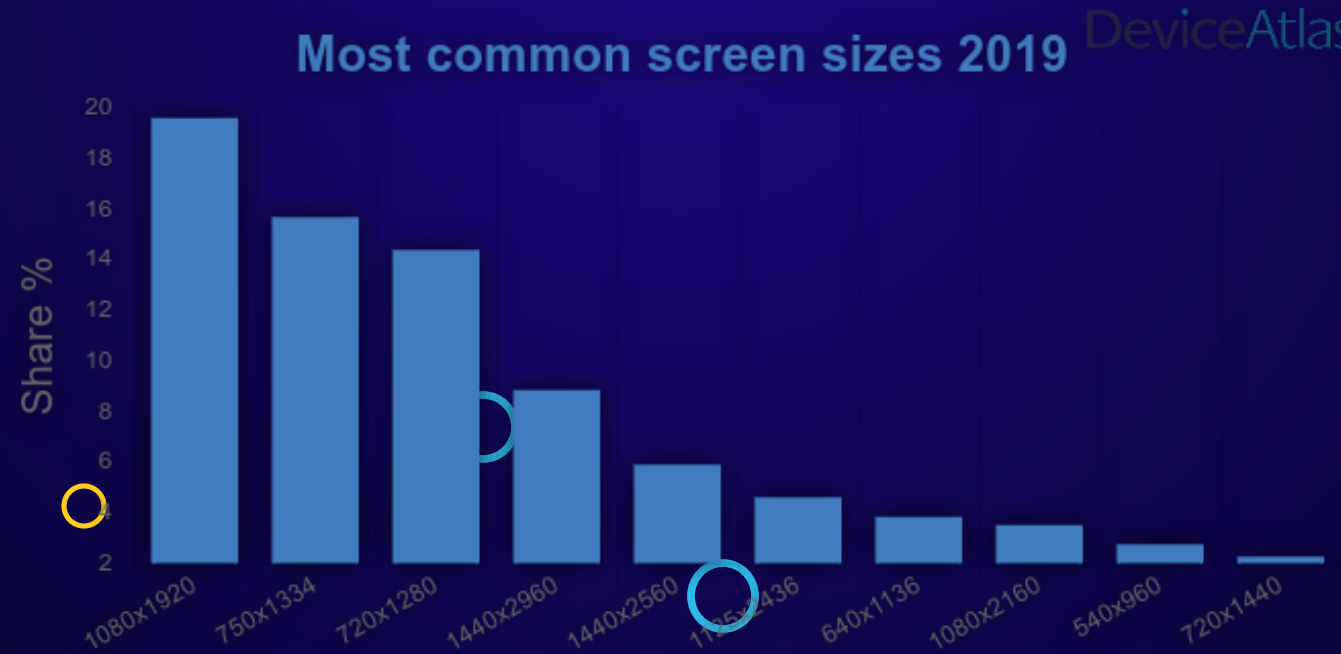
Resoluções por país

DeviceAtlas Products Solutions

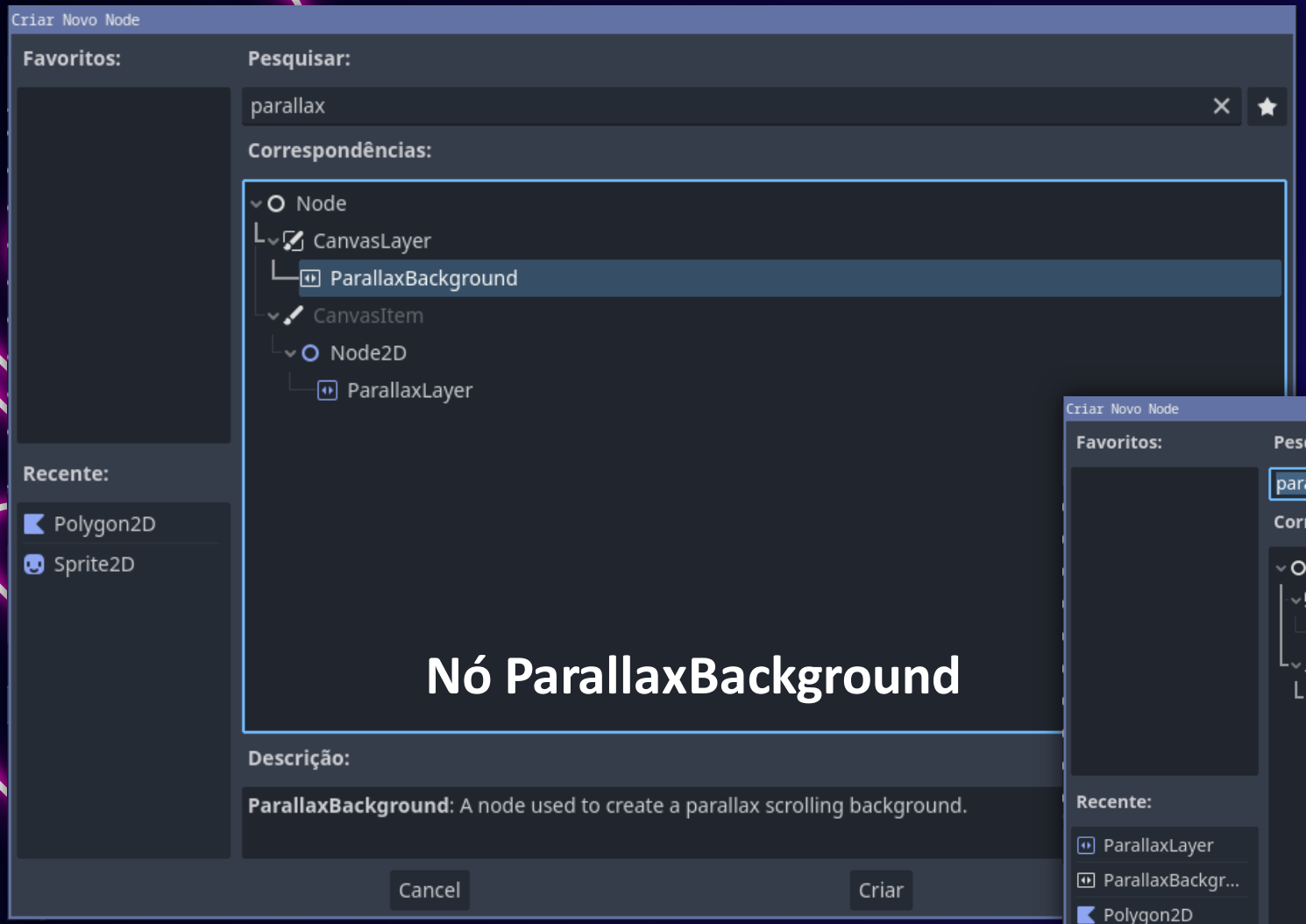
The following table shows which screen resolution is most used in selected countries based on web traffic statistics for **Q2 2019**.

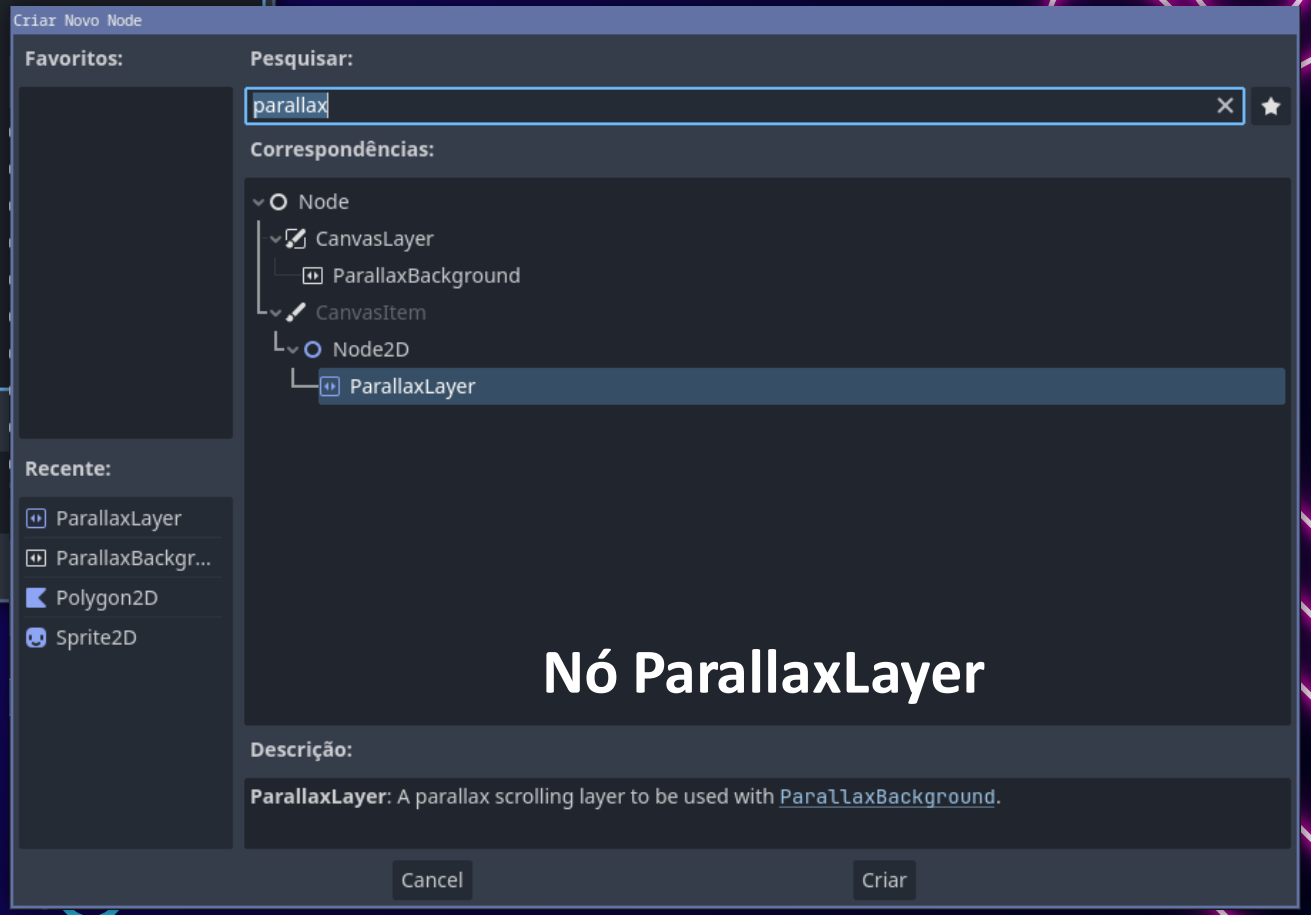
Country	Screen resolution	Share	Change (v Q1)
Argentina	720x1280	33.31%	-1.36%
Australia	750x1334	22.03%	-1.80%
Brazil	720x1280	22.52%	-6.86%
Canada	750x1334	24.77%	-1.33%
Colombia	1080x1920	19.10%	+0.18%
Denmark	750x1334	28.01%	-3.64%
Egypt	720x1280	21.36%	-5.68%
Finland	1080x1920	24.54%	+2.21%
France	750x1334	19.54%	-2.94%
Germany	1080x1920	17.89%	-1.74%

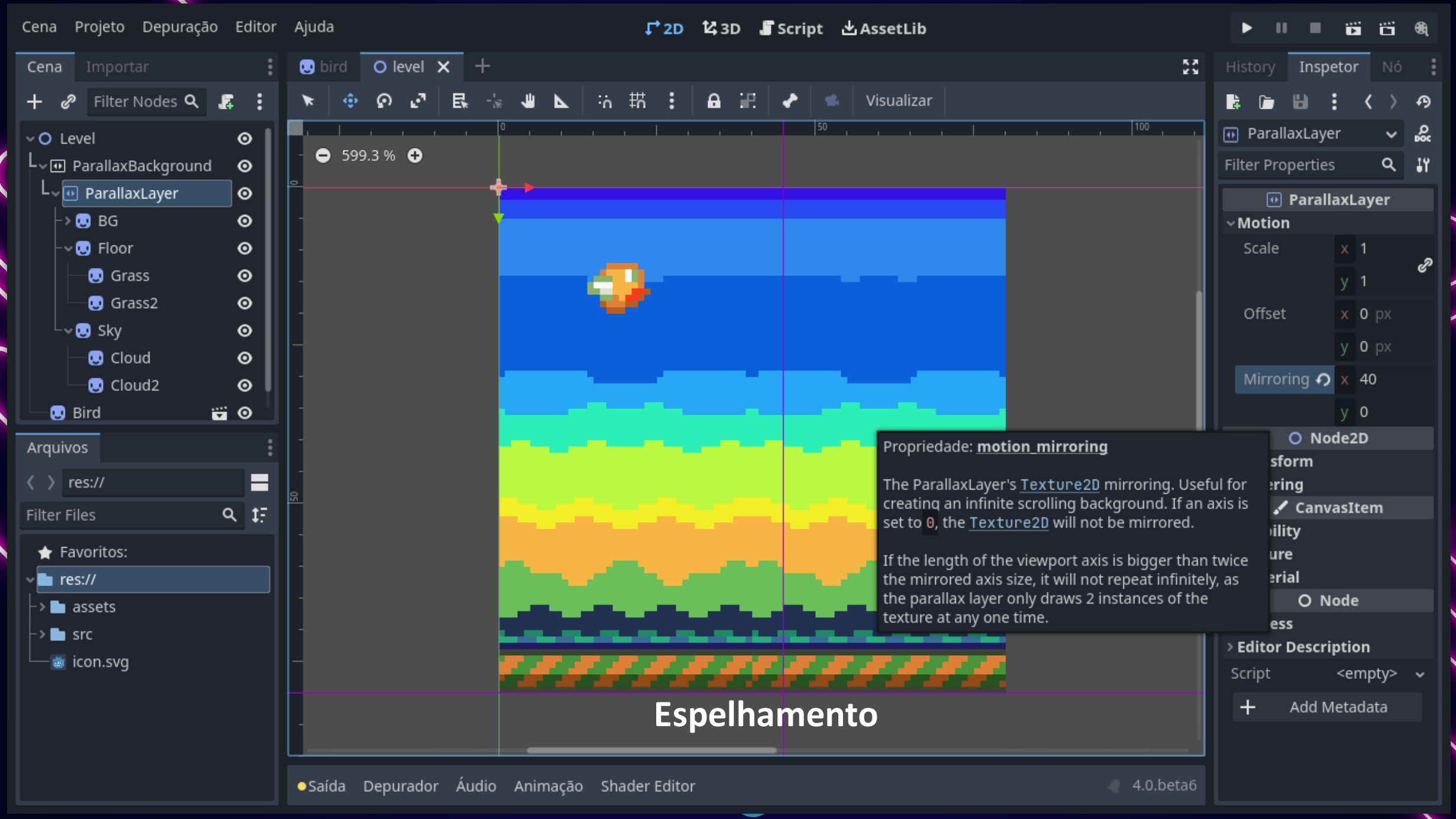
Resoluções no Brasil

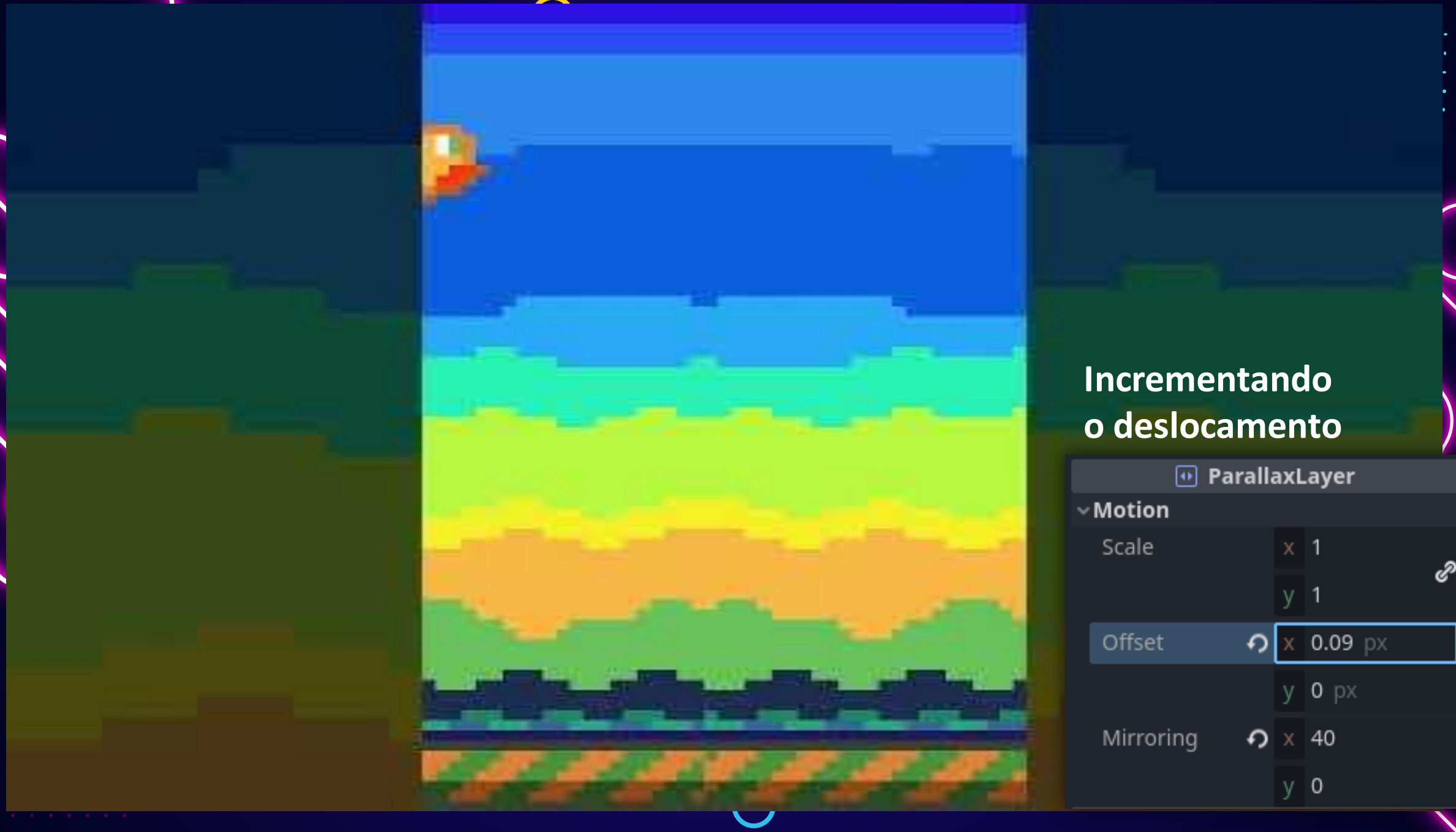


Parallax

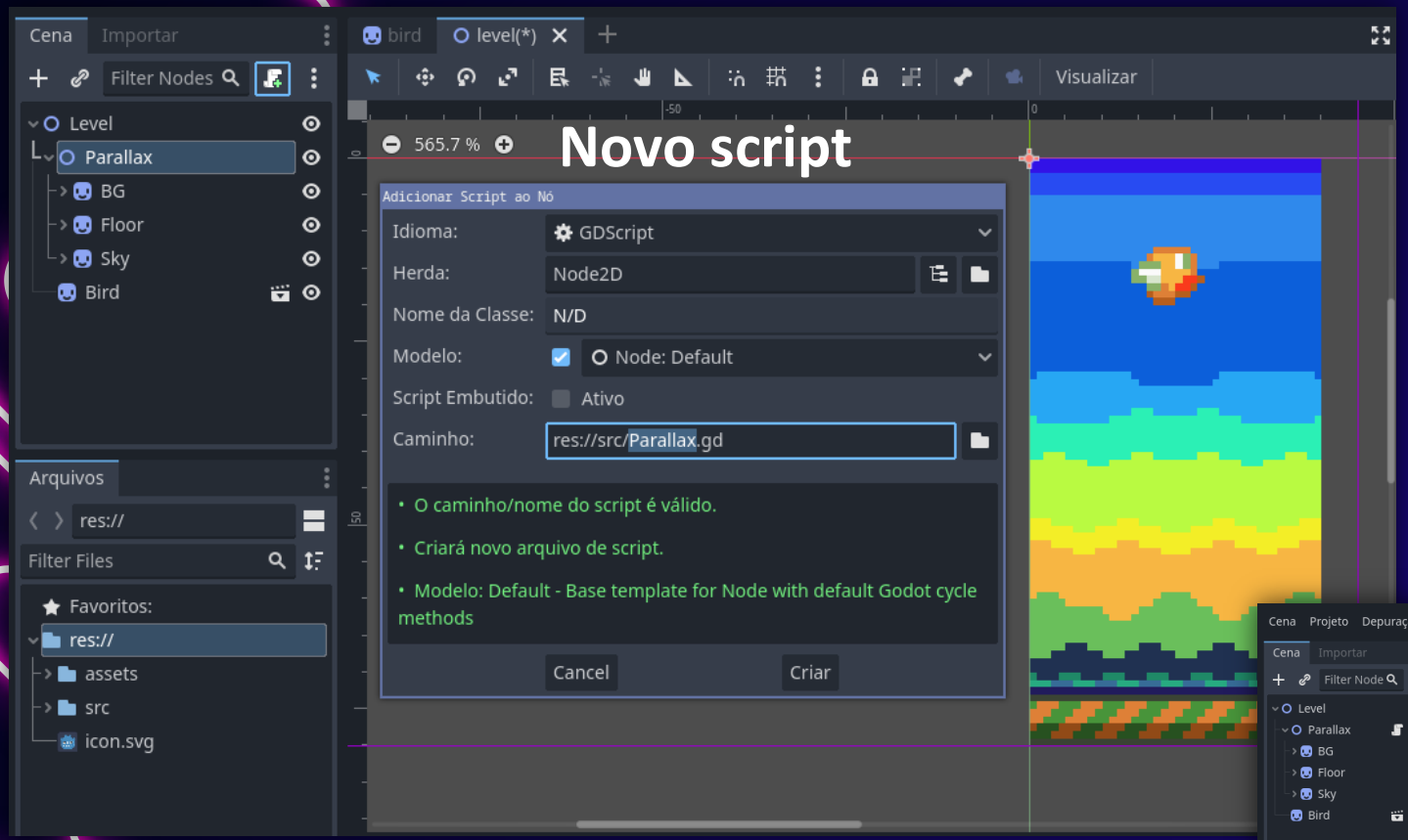

Criar Novo Node
Favoritos:
Pesquisar:
parallax
Correspondências:
Node
CanvasLayer
ParallaxBackground
CanvasItem
Node2D
ParallaxLayer
Recente:
Polygon2D
Sprite2D
Nó ParallaxBackground
Descrição:
ParallaxBackground: A node used to create a parallax scrolling background.
Cancel
Criar


Criar Novo Node
Favoritos:
Pesquisar:
parallax
Correspondências:
Node
CanvasLayer
ParallaxBackground
CanvasItem
Node2D
ParallaxLayer
Recente:
ParallaxLayer
ParallaxBackgr...
Polygon2D
Sprite2D
Nó ParallaxLayer
Descrição:
ParallaxLayer: A parallax scrolling layer to be used with ParallaxBackground.
Cancel
Criar

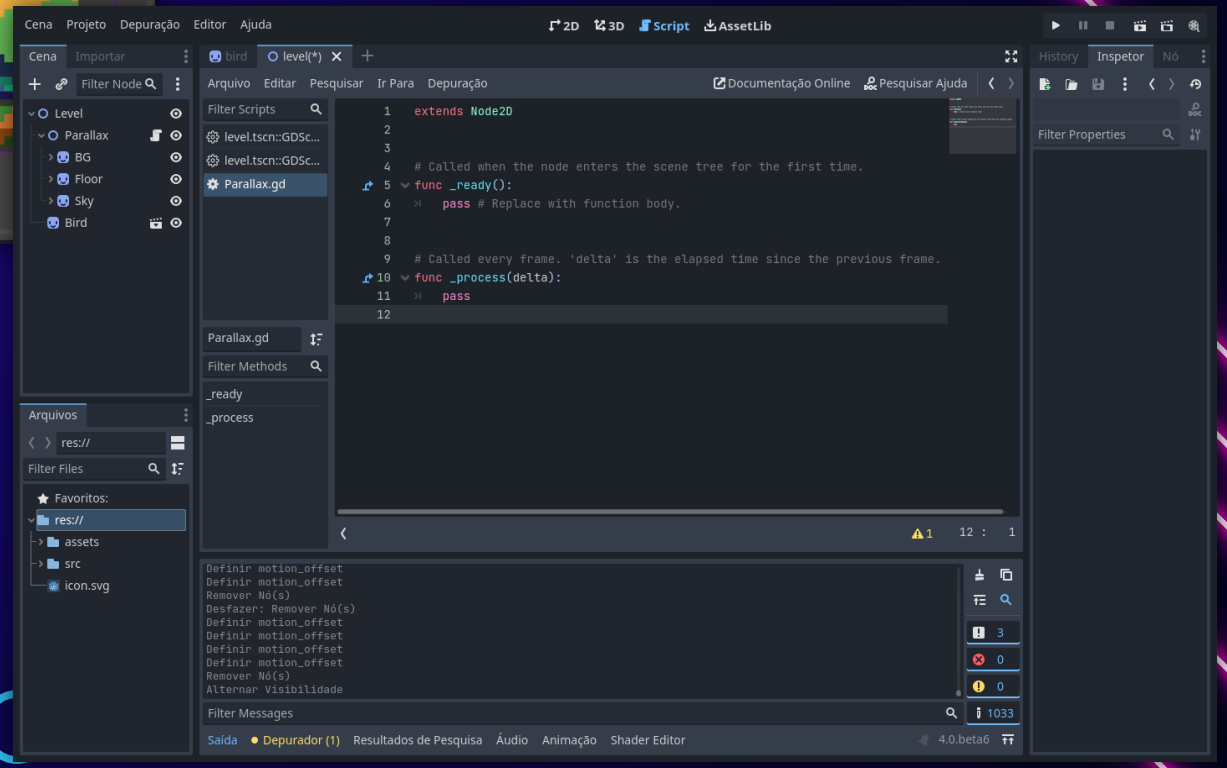

Cena Projeto Depuração Editor Ajuda
2D 3D Script AssetLib
Cena Importar
Filter Nodes
Level
ParallaxBackground
ParallaxLayer
BG
Floor
Grass
Grass2
Sky
Cloud
Cloud2
Bird
Arquivos
res://
Filter Files
Favoritos:
res://
assets
src
icon.svg
bird
level
Visualizar
599.3 %
Propriedade: motion_mirroring
The ParallaxLayer's Texture2D mirroring. Useful for creating an infinite scrolling background. If an axis is set to 0, the Texture2D will not be mirrored.
If the length of the viewport axis is bigger than twice the mirrored axis size, it will not repeat infinitely, as the parallax layer only draws 2 instances of the texture at any one time.
Espelhamento
Saída Depurador Áudio Animação Shader Editor
4.0.beta6
History Inspetor Nó
ParallaxLayer
Filter Properties
ParallaxLayer
Motion
Scale x 1 y 1
Offset x 0 px y 0 px
Mirroring x 40 y 0
Node2D
CanvasItem
Node
Editor Description
Script <empty>
Add Metadata


Incrementando
o deslocamento
ParallaxLayer
Motion
Scale
x 1
y 1
Offset
x 0.09 px
y 0 px
Mirroring
x 40
y 0

Parallax: Uma abordagem diferente



Editor de texto




```gdscript
extends Node2D

var bg
var floor
var sky

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg = get_node("BG") # Obtém um nó, via código, por nome
    floor = $Floor      # Atalho para `get_node`
    sky = get_child(2)  # Obtém um nó filho, por índice

```

Método `get_node()`

Ordem de precedência (exemplo com erro)

```gdscript
extends Node2D

var bg
var floor
var sky

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg = get_node("BG") # Obtém um nó, via código, por nome
    floor = $Floor      # Atalho para `get_node`
    sky = get_child(2)  # Obtém um nó filho, por índice

var bg_copy = bg.duplicate() # duplica uma instância de Nó
var floor_copy = floor.duplicate()
var sky_copy = sky.duplicate()

```

@onready

```gdscript
extends Node2D

@onready var bg = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor = $Floor      # Atalho para `get_node`
@onready var sky = get_child(2)  # Obtém um nó filho, por índice

var bg_copy
var floor_copy
var sky_copy

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg_copy = bg.duplicate() # duplica uma instância de Nó
    floor_copy = floor.duplicate()
    sky_copy = sky.duplicate()

```

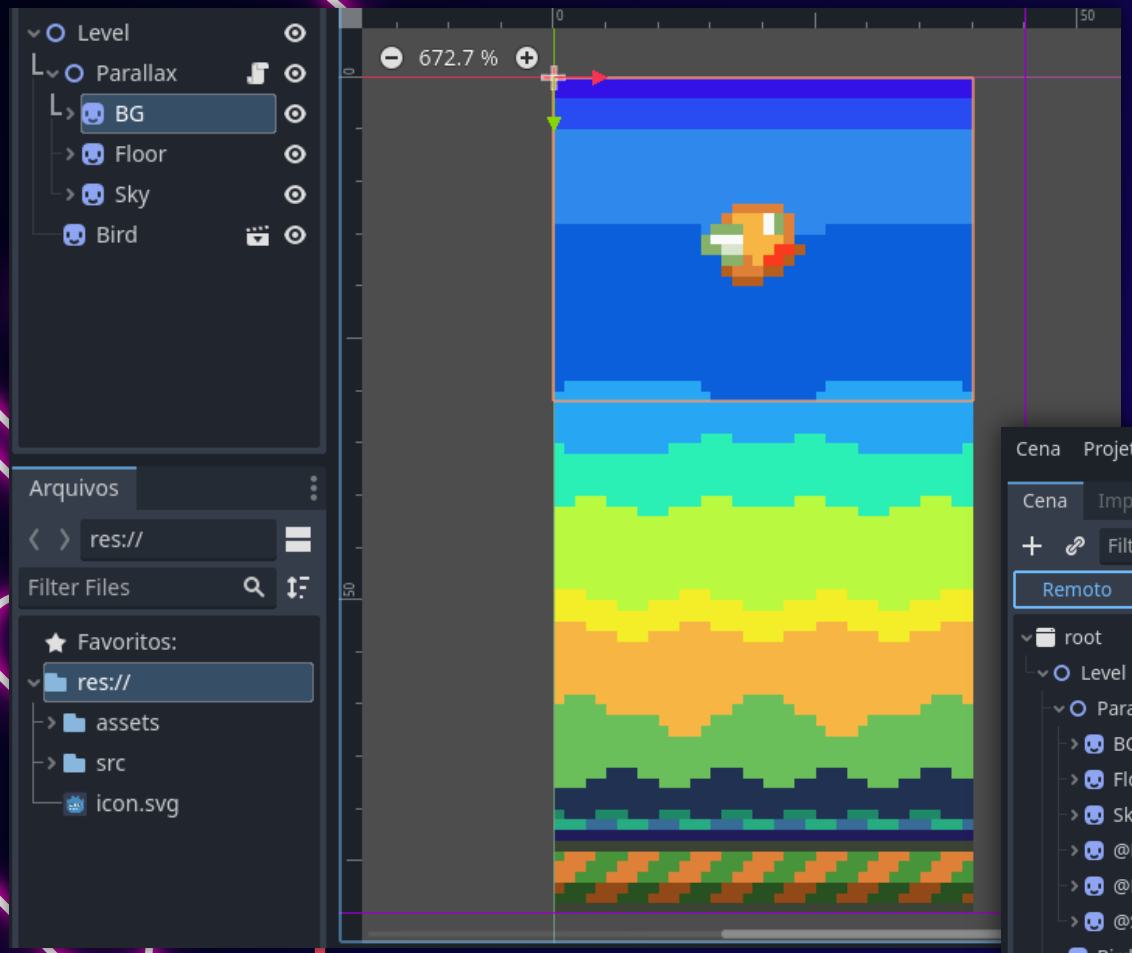
```gdscript
extends Node2D

@onready var bg = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor = $Floor      # Atalho para `get_node`
@onready var sky = get_child(2)  # Obtém um nó filho, por índice

var bg_copy
var floor_copy
var sky_copy

# _ready é chamado quando o nó entra na ár
func _ready():
    bg_copy = bg.duplicate() # duplica uma
    floor_copy = floor.duplicate()
    sky_copy = sky.duplicate()

    bg_copy.position.x = (bg as Sprite2D).reg

```

```
region_enabled
region_filter_clip_enabled
region_rect
frame_changed
texture_changed
draw_colored_polygon
draw_multiline_string
```

Intellisense

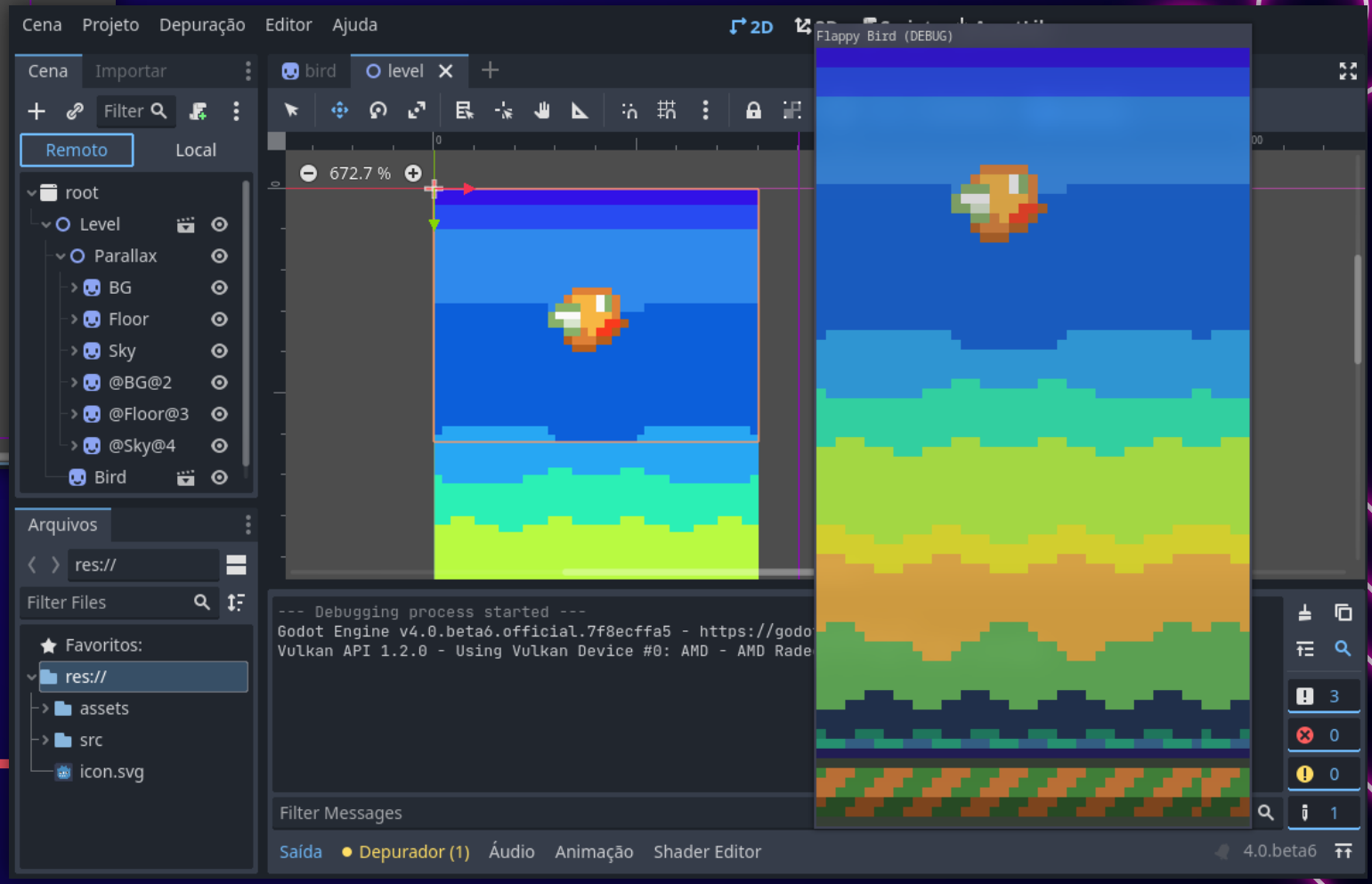
add_child()

```gdscript
extends Node2D

@onready var bg = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor = $Floor      # Atalho para `get_node`
@onready var sky = get_child(2)  # Obtém um nó filho, por índice

var bg_copy
var floor_copy
var sky_copy

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg_copy = bg.duplicate() # duplica uma instância de Nó
    floor_copy = floor.duplicate()
    sky_copy = sky.duplicate()

    bg_copy.position.x = (bg as Sprite2D).region_rect.size.x
    floor_copy.position.x = (floor as Sprite2D).region_rect.size.x
    sky_copy.position.x = (sky as Sprite2D).region_rect.size.x

    self.add_child(bg_copy) # Adiciona uma instância de Nó como filho
    add_child(floor_copy)   # o nó adicionado deve ser órfão antes da chamada
    add_child(sky_copy)

```



Estado atual

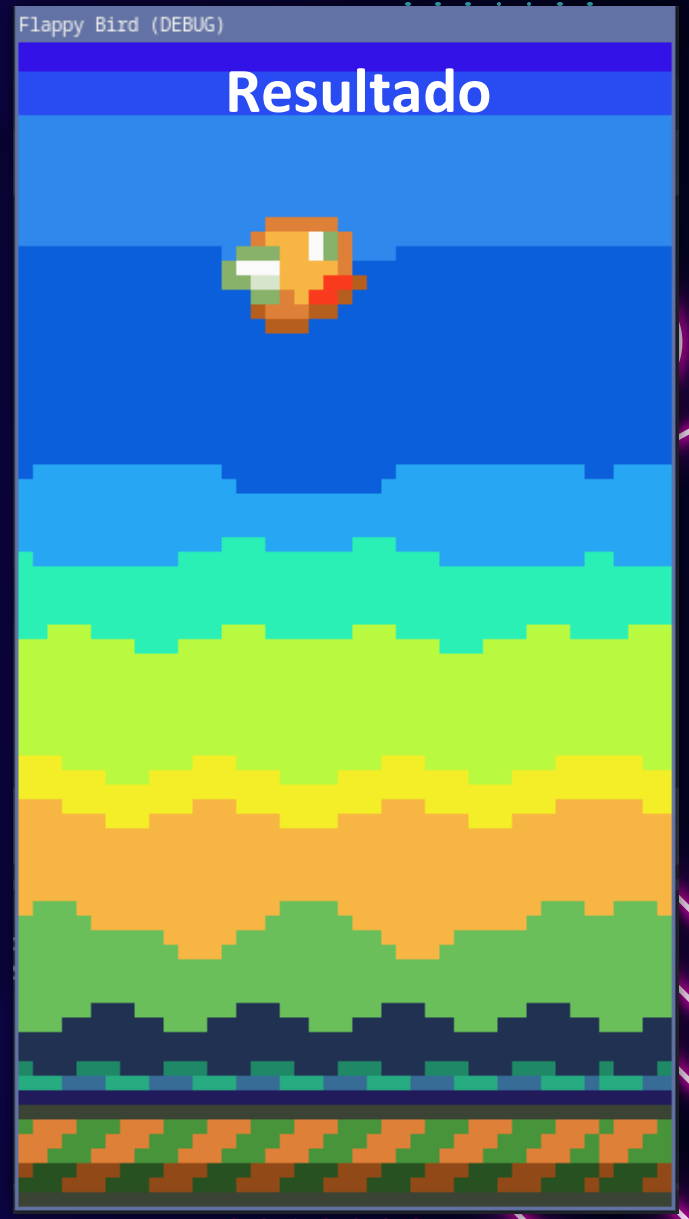
Instâncias em tempo real no modo remoto



movement

Movimentação

```gdscript
var speed = 1.0 # Taxa de velocidade (rapidez)

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    var velocity_x = speed * delta # Velocidade atual do movimento no eixo-x

    # Move os Sprites com base na velocidade atual
    bg.position.x = bg.position.x - velocity_x
    sky.position.x = sky.position.x - velocity_x
    floor.position.x = floor.position.x - velocity_x
    bg_copy.position.x -= velocity_x
    sky_copy.position.x -= velocity_x
    floor_copy.position.x -= velocity_x

    speed += delta # Aumenta a velocidade progressivamente

```



Ordem de atribuição

```gdscript
extends Node2D

@onready var bg = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor = $Floor      # Atalho para `get_node`
@onready var sky = get_child(2)  # Obtém um nó filho, por índice

@onready var bg_copy = bg.duplicate() # duplica uma instância de Nó
@onready var floor_copy = floor.duplicate()
@onready var sky_copy = sky.duplicate()

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg_copy.position.x = (bg as Sprite2D).region_rect.size.x
    floor_copy.position.x = (floor as Sprite2D).region_rect.size.x
    sky_copy.position.x = (sky as Sprite2D).region_rect.size.x

    self.add_child(bg_copy) # Adiciona uma instância de Nó como filho
    add_child(floor_copy)   # o nó adicionado deve ser órfão antes da chamada
    add_child(sky_copy)
```

Refatorando

```gdscript
@onready var bg: Sprite2D = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor: Sprite2D = $Floor      # Atalho para `get_node`
@onready var sky: Sprite2D = get_child(2)  # Obtém um nó filho, por índice

@onready var bg_copy: Sprite2D = bg.duplicate() # duplica uma instância de Nó
@onready var floor_copy: Sprite2D = floor.duplicate()
@onready var sky_copy: Sprite2D = sky.duplicate()

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg_copy.position.x = bg.region_rect.size.x
    floor_copy.position.x = floor.region_rect.size.x
    sky_copy.position.x = sky.region_rect.size.x

    self.add_child(bg_copy) # Adiciona uma instância de Nó como filho
    add_child(floor_copy)   # o nó adicionado deve ser órfão antes da chamada
    add_child(sky_copy)

var speed: float = 1.0 # Taxa de velocidade (rapidez)

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    var velocity_x: float # Velocidade atual do movimento no eixo-x
    velocity_x = speed * delta
```

Coerção de tipo

```gdscript
extends Node2D

@onready var bg = get_node("BG") # Obtém um nó, via código, por nome
@onready var floor = $Floor      # Atalho para `get_node`
@onready var sky = get_child(2)  # Obtém um nó filho, por índice

@onready var bg_copy = bg.duplicate() # duplica uma instância de Nó
@onready var floor_copy := (floor as Sprite2D).duplicate()
@onready var sky_copy: Sprite2D = sky.duplicate()

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    bg_copy.position.x = bg.region_rect.size.x
    floor_copy.position.x = floor.region_rect.size.x
    sky_copy.position.x = (sky as Sprite2D).region_rect.size.x

    self.add_child(bg_copy) # Adiciona uma instância de Nó como filho
    add_child(floor_copy)   # o nó adicionado deve ser órfão antes da chamada
    add_child(sky_copy)
```

```gdscript
# Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
func _process(delta):
    # Velocidade atual do movimento no eixo-x
    var velocity_x: float = speed * delta
    # Move os Sprites com base na velocidade atual
    bg.position.x -= velocity_x
    sky.position.x -= velocity_x
    floor.position.x -= velocity_x
    bg_copy.position.x -= velocity_x
    sky_copy.position.x -= velocity_x
    floor_copy.position.x -= velocity_x

    if bg.position.x + bg.region_rect.size.x < 0:
        bg.position.x = bg_copy.position.x + bg_copy.region_rect.size.x
    if sky.position.x + sky.region_rect.size.x < 0:
        sky.position.x = sky_copy.position.x + sky_copy.region_rect.size.x
    if floor.position.x + floor.region_rect.size.x < 0:
        floor.position.x = floor_copy.position.x + floor_copy.region_rect.size.x

    if bg_copy.position.x + bg_copy.region_rect.size.x < 0:
        bg_copy.position.x = bg.position.x + bg.region_rect.size.x
    if sky_copy.position.x + sky_copy.region_rect.size.x < 0:
        sky_copy.position.x = sky.position.x + sky.region_rect.size.x
    if floor_copy.position.x + floor_copy.region_rect.size.x < 0:
        floor_copy.position.x = floor.position.x + floor.region_rect.size.x

    speed += delta # Aumenta a velocidade progressivamente

```

Reposicionando BG (Back Ground)

Repositioning result

children array

```gdscript
extends Node2D

## Retorna a lista de filhos do nó
## Assume que todos os filhos são Sprites
@onready var children: Array[Sprite2D] = get_children()

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    var copies: Array[Sprite2D] = []

    for child in children:
        var copy: Sprite2D = child.duplicate()      # Duplica um filho
        child.position.x = child.region_rect.size.x # Posiciona a cópia
        add_child(copy) # Adiciona a cópia (uma instância de Sprite) como filho
        # o nó adicionado deve ser órfão antes da chamada

    children.append_array(copies) # Adiciona as cópias como filhos

```

Rafatorando

children processing

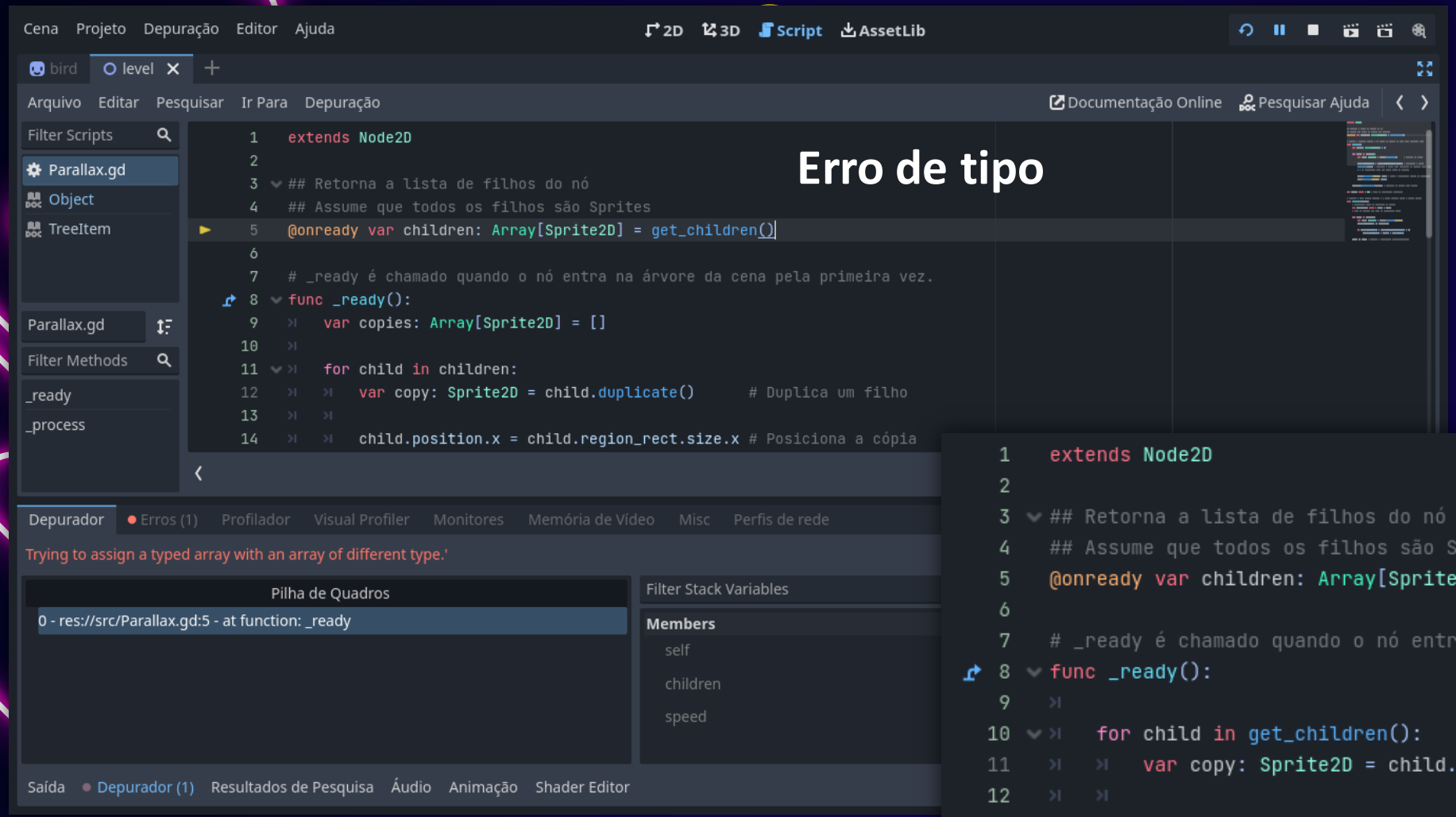
```gdscript
# Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
func _process(delta):
    # Velocidade atual do movimento no eixo-x
    var velocity_x: float = speed * delta
    # Move os Sprites com base na velocidade atual

    for child in children:
        child.position.x -= velocity_x

        if child.position.x + ???.region_rect.size.x < 0:
            child.position.x = ???.position.x + ???.region_rect.size.x

    speed += delta # Aumenta a velocidade progressivamente

```

```gdscript
var speed: float = 1.0 # Taxa de velocidade (rapidez)

# Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
func _process(delta):
    # Velocidade atual do movimento no eixo-x
    var velocity_x: float = speed * delta
    # Move os Sprites com base na velocidade atual

    for child in children:
        var twin: Sprite2D = child.get_meta("twin")
        child.position.x -= velocity_x

        if child.position.x + twin.region_rect.size.x < 0:
            child.position.x = twin.position.x + twin.region_rect.size.x

    speed += delta # Aumenta a velocidade progressivamente

```

Usando nó "twin" como metadado

Rafatorando

```gdscript
# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():
    var copies: Array[Sprite2D] = []

    for child in children:
        var copy: Sprite2D = child.duplicate()      # Duplica um filho

        child.position.x = child.region_rect.size.x # Posiciona a cópia
        add_child(copy) # Adiciona a cópia (uma instância de Sprite) como filho
        # o nó adicionado deve ser órfão antes da chamada

        child.set_meta("twin", copy) # Salva a "instância gêmea" em metadados
        copy.set_meta("twin", child)

```



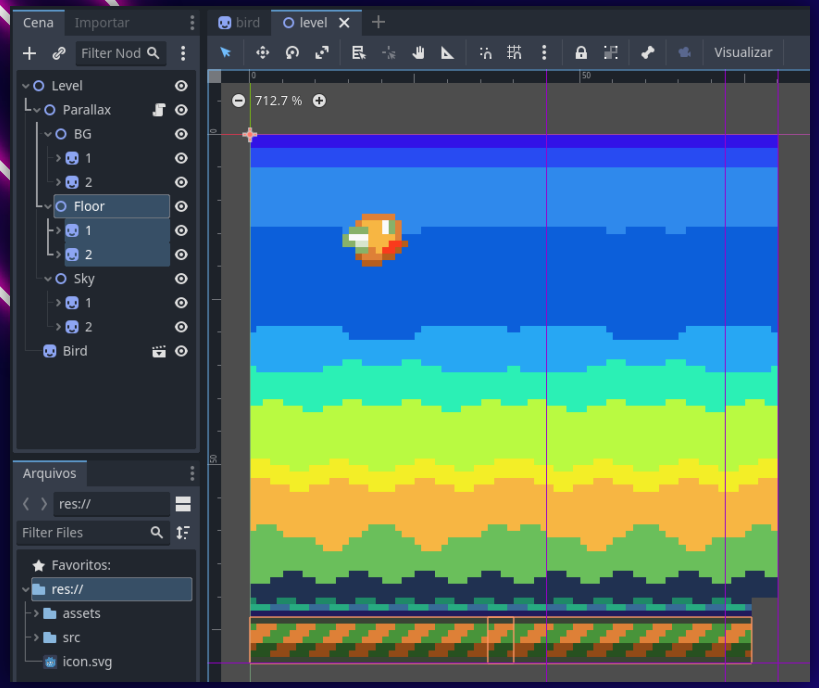
Erro de tipo

Consertando erro

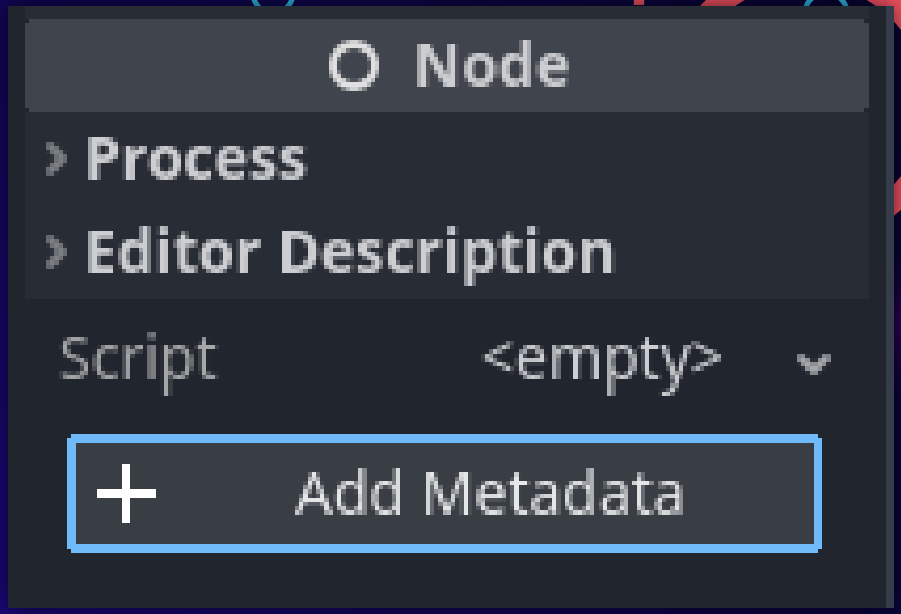
```gdscript
extends Node2D

## Retorna a lista de filhos do nó
## Assume que todos os filhos são Sprites
@onready var children: Array[Sprite2D] = []

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():

    for child in get_children():
        var copy: Sprite2D = child.duplicate()      # Duplica um filho

        child.position.x = child.region_rect.size.x # Posiciona a cópia
        add_child(copy) # Adiciona a cópia (uma instância de Sprite) como filho
        # o nó adicionado deve ser órfão antes da chamada

        child.set_meta("twin", copy) # Salva a "instância gêmea" em metadados
        copy.set_meta("twin", child)

        children.append(child) # Salva os filhos na lista `children`
        children.append(copy)
```

Refactor result

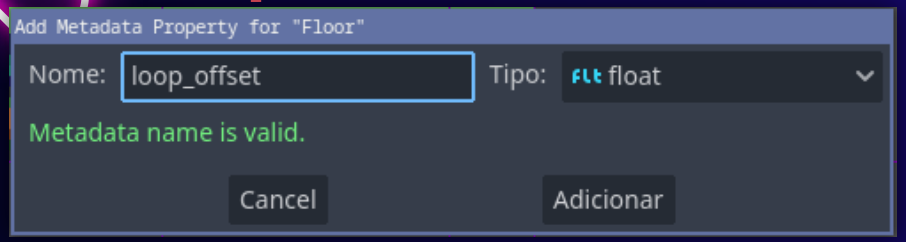
Duplicando



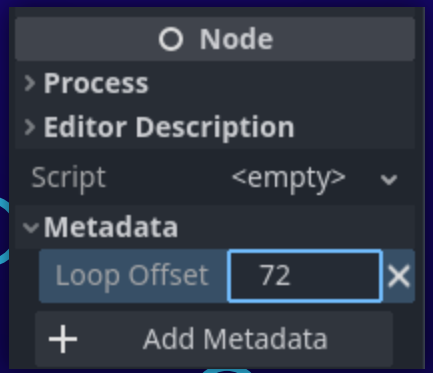
Add metadata



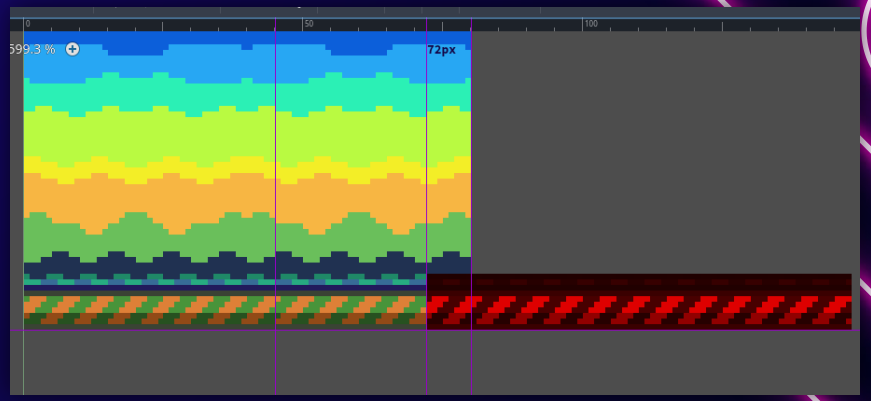
Metadata



Set offset



Loop offset



Refactor: ready

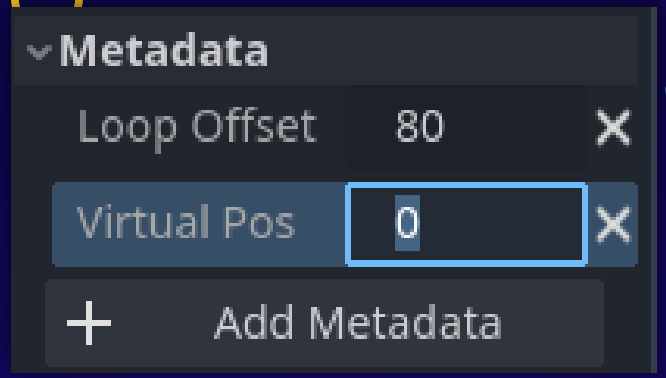
```gdscript
extends Node2D

## Retorna a lista de filhos do nó
## Assume que todos os filhos são Sprites
@onready var children: Array[Node2D] = []

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():

    for child in get_children():
        var copy: Node2D = child.duplicate()      # Duplica um filho

        copy.position.x = child.get_meta("loop_offset") # Posiciona a cópia
        add_child(copy) # Adiciona a cópia (uma instância de Sprite) como filho
        # o nó adicionado deve ser órfão antes da chamada

        #child.set_meta("twin", copy) # Salva a "instância gêmea" em metadados
        #copy.set_meta("twin", child)

        children.append(child) # Salva os filhos na lista `children`
        children.append(copy)

```

Refactor: process

```gdscript
var speed: float = 1.0 ## Taxa de velocidade (rapidez)

# Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
func _process(delta):
    # Velocidade atual do movimento no eixo-x
    var velocity_x: float = speed * delta
    # Move os Sprites com base na velocidade atual

    for child in children:
        child.position.x -= velocity_x

    for child in children:
        #var twin: Node2D = child.get_meta("twin")
        var offset: float = child.get_meta("loop_offset")
        var end_pos: float = child.position.x + offset

        if end_pos < 0:
            # end_pos será o quanto de espaço em x ultrapassou o eixo y
            child.position.x = offset + end_pos

    speed += delta # Aumenta a velocidade progressivamente

```

Fix gap

Virtual pos



Fix pixel precision

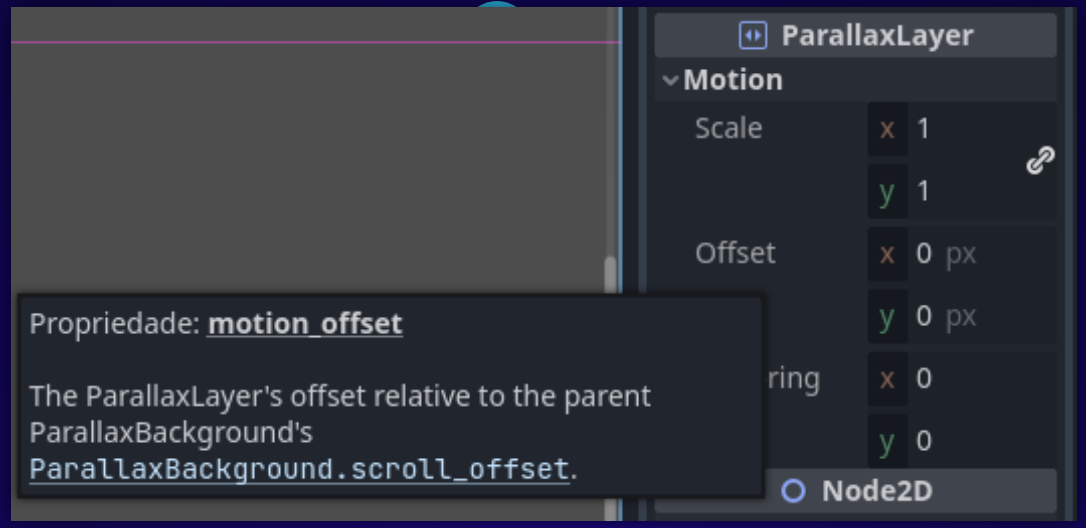
```gdscript
extends Node2D

## Retorna a lista de filhos do nó
## Assume que todos os filhos são Sprites
@onready var children: Array[Node2D] = []

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():

    for child in get_children():
        var copy: Node2D = child.duplicate()      # Duplica um filho
        var offset: float = child.get_meta("loop_offset")

        copy.set_meta("virtual_pos", offset) # Posição "virtual"
        copy.position.x = floorf(offset)     # Posiciona a cópia (fixed-pixel)

        add_child(copy) # Adiciona a cópia (uma instância de Sprite) como filho
        # o nó adicionado deve ser órfão antes da chamada

        #child.set_meta("twin", copy) # Salva a "instância gêmea" em metadados
        #copy.set_meta("twin", child)

        children.append(child) # Salva os filhos na lista `children`
        children.append(copy)

```

Fix position

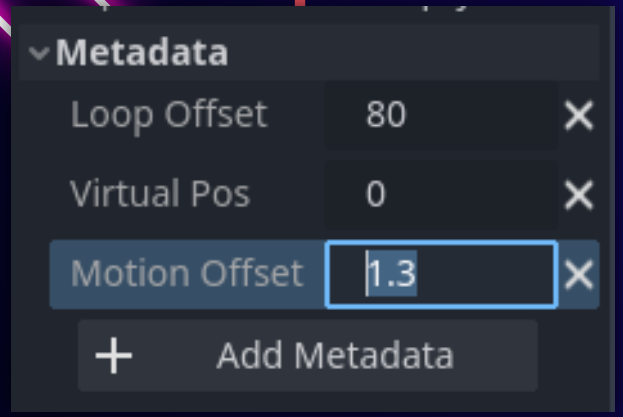
```gdscript
var speed: float = 1.0 ## Taxa de velocidade (rapidez)

# Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
func _process(delta):
    # Velocidade atual do movimento no eixo-x
    var velocity_x: float = speed * delta
    # Move os Sprites com base na velocidade atual

    for child in children:
        var new_pos := (child.get_meta("virtual_pos") as float) - velocity_x
        child.set_meta("virtual_pos", new_pos)
        child.position.x = roundf(new_pos)
        #child.position.x -= velocity_x

    for child in children:
        #var twin: Node2D = child.get_meta("twin")
        var offset: float = child.get_meta("loop_offset")
        var end_pos: float = (child.get_meta("virtual_pos") as float) + offset

        if end_pos < 0:
            # end_pos será o quanto de espaço em x ultrapassou o eixo y
            child.set_meta("virtual_pos", offset + end_pos)
            child.position.x = roundf(offset + end_pos)

    speed += delta # Aumenta a velocidade progressivamente

```

Fix gap final

Parallax offset



Motion offset

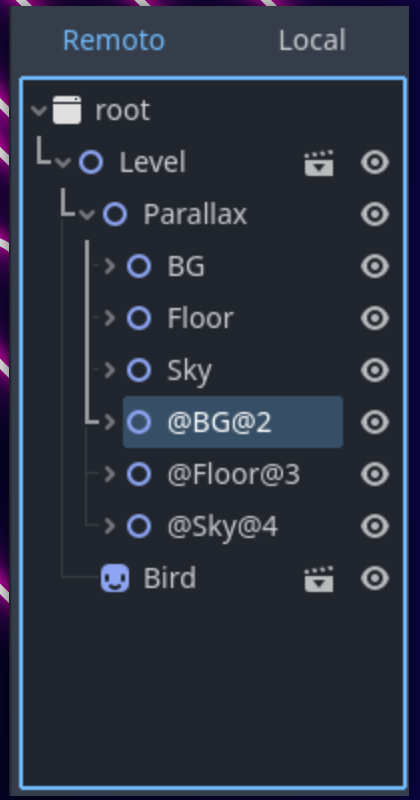


Apply offset

```
34  for child in children:
35      var new_pos := child.get_meta("virtual_pos") as float - \
36          velocity_x * child.get_meta("motion_offset") as float
37      child.set_meta("virtual_pos", new_pos)
38      child.position.x = roundf(new_pos)
39      #child.position.x -= velocity_x
40
```

Order problem

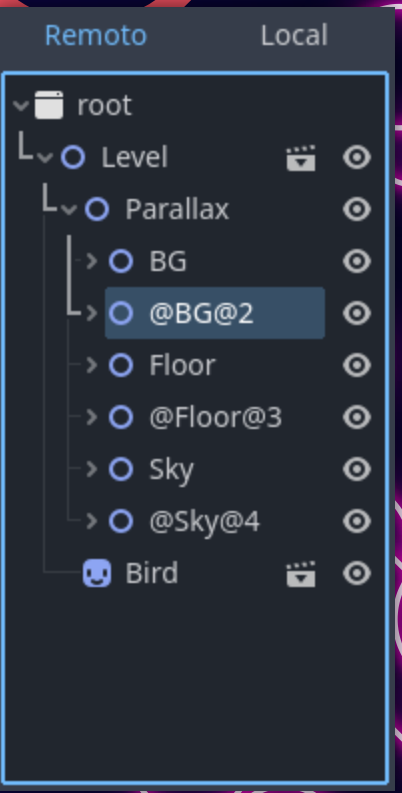
Remote state



Node as sibling

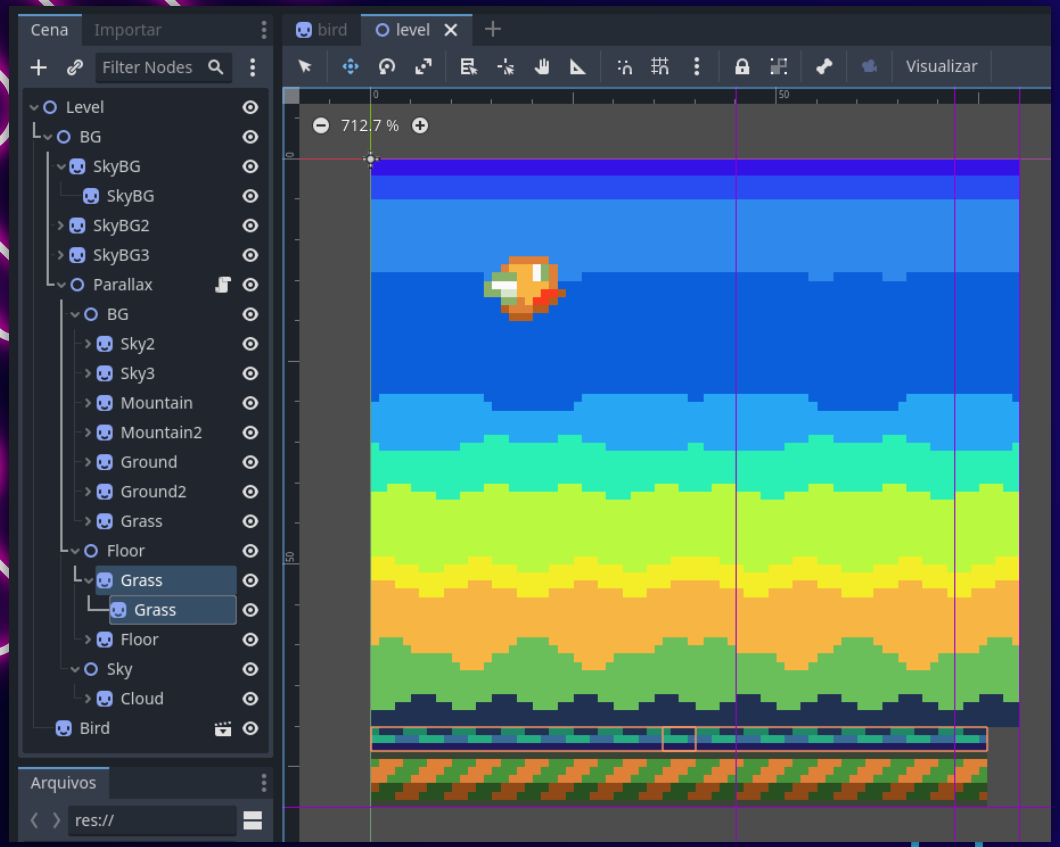
```gdscript
# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():

    for child in get_children():
        var copy: Node2D = child.duplicate()      # Duplica um filho
        var offset: float = child.get_meta("loop_offset")

        copy.set_meta("virtual_pos", offset) # Posição "virtual"
        copy.position.x = floorf(offset)     # Posiciona a cópia (fixed-pixel)

        child.add_sibling(copy) # Adiciona a cópia no mesmo nível do filho
        # o nó adicionado deve ser órfão antes da chamada
```

Remote fixed



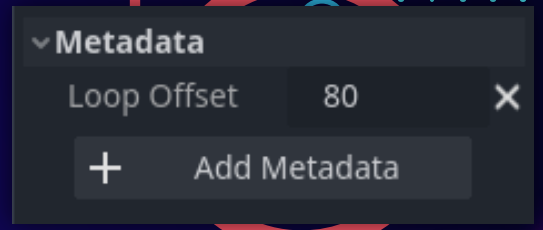
Order problem fixed

Nova hierarquia



Metadados internos

Remove metadata



Grandchildren

```gdscript
extends Node2D

## A lista de filhos do nó
## Assume que todos os filhos são Node2D
@onready var children: Array[Node2D] = []

# _ready é chamado quando o nó entra na árvore da cena pela primeira vez.
func _ready():

    for child in get_children(): # filhos
        var offset: float = child.get_meta("loop_offset")

        for grandchild in child.get_children(): # netos
            var copy: Node2D = grandchild.duplicate() # Duplica um neto

            grandchild.set_meta("virtual_posx", grandchild.position.x)
            copy.set_meta("virtual_posx", offset) # Posição "virtual"
            copy.position.x = floorf(offset)     # Posiciona a cópia (fixed-pxl)

            grandchild.add_sibling(copy)
            # Adiciona a cópia como um "irmão" logo abaixo do neto
            # o nó adicionado deve ser órfão antes da chamada

        children.append(child) # Salva os filhos na lista `children`
```

Refactor: loop

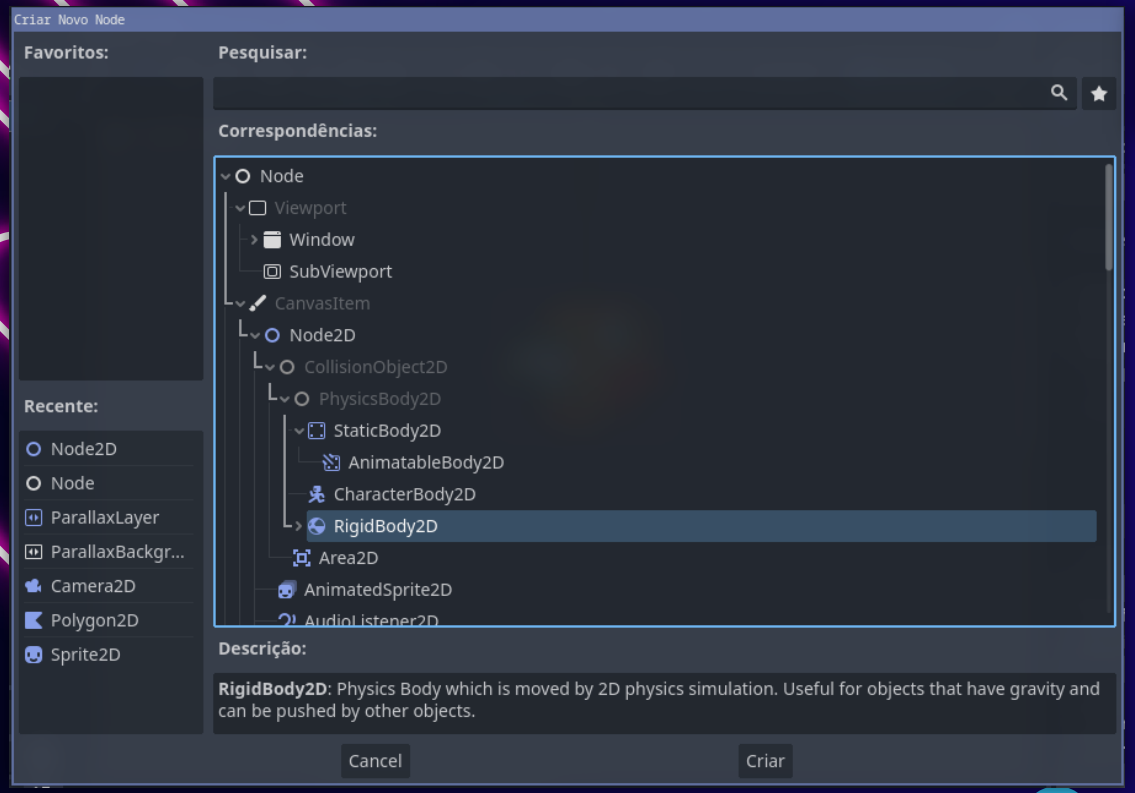
```
26  var speed: float = 1.0 ## Taxa de velocidade (rapidez)
27
28  # Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
29  func _process(delta):
30      # Velocidade atual do movimento no eixo-x
31      var velocity_x: float = speed * delta
32      # Move os Sprites com base na velocidade atual
33
34      for child in children:
35          var offset: float = child.get_meta("loop_offset")
36
37          for grandchild in child.get_children():
38              var new_pos := grandchild.get_meta("virtual_posx") as float - \
39                  velocity_x * grandchild.get_meta("motion_offset") as float
40              var end_pos := new_pos + offset
41
42              if end_pos < 0:
43                  # end_pos será o quanto de espaço em x ultrapassou o eixo y
44                  new_pos = offset + end_pos
45
46              grandchild.set_meta("virtual_posx", new_pos)
47              grandchild.position.x = roundf(new_pos)
48
49      speed += delta # Aumenta a velocidade progressivamente
50
```

Refactor: process

```
28  # Chamado a cada frame. 'delta' é o tempo passado desde o último frame.
29  func _process(delta):
30      # Velocidade atual do movimento no eixo-x
31      var velocity_x: float = speed * delta
32      # Move os Sprites com base na velocidade atual
33
34      for child in children:
35          for grandchild in child.get_children():
36              var new_pos := grandchild.get_meta("virtual_posx") as float - \
37                  velocity_x * grandchild.get_meta("motion_offset") as float
38              grandchild.set_meta("virtual_posx", new_pos)
39              grandchild.position.x = roundf(new_pos)
40
41      for child in children:
42          var offset: float = child.get_meta("loop_offset")
43
44          for grandchild in child.get_children():
45              var end_pos := (grandchild.get_meta("virtual_posx") as float) + offset
46
47              if end_pos < 0:
48                  # end_pos será o quanto de espaço em x ultrapassou o eixo y
49                  grandchild.set_meta("virtual_posx", offset + end_pos)
50                  grandchild.position.x = roundf(offset + end_pos)
51
52      speed += delta # Aumenta a velocidade progressivamente
```

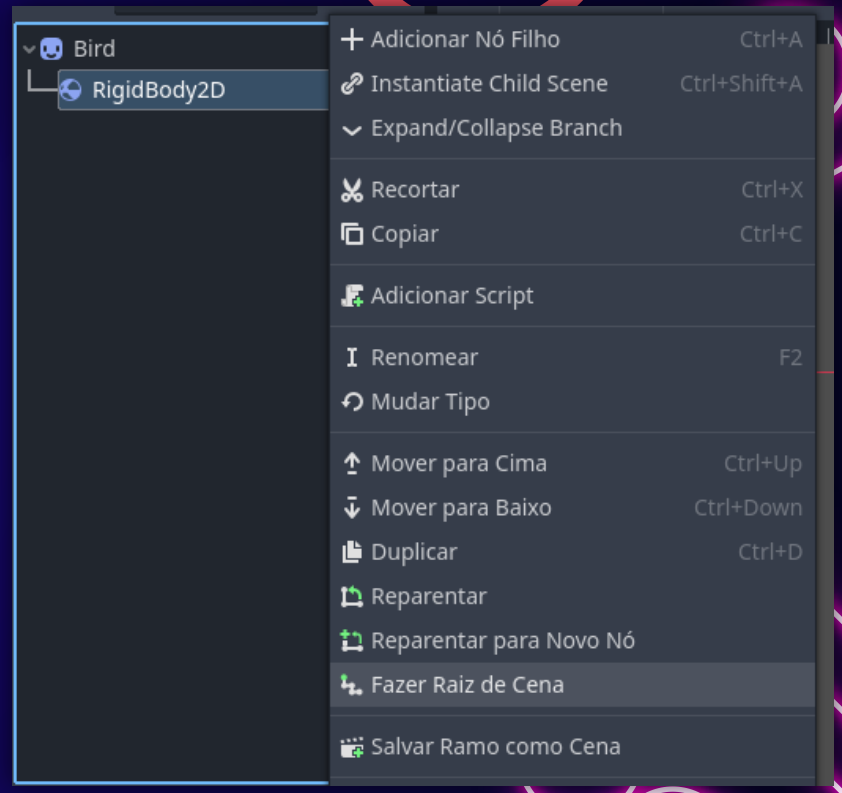
Gravidade

Add RigidBody2D

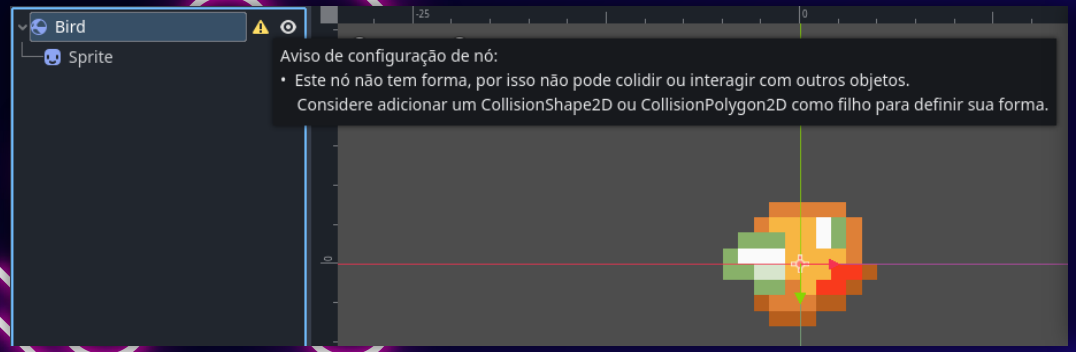


RigidBody2D tree

Tornar raiz

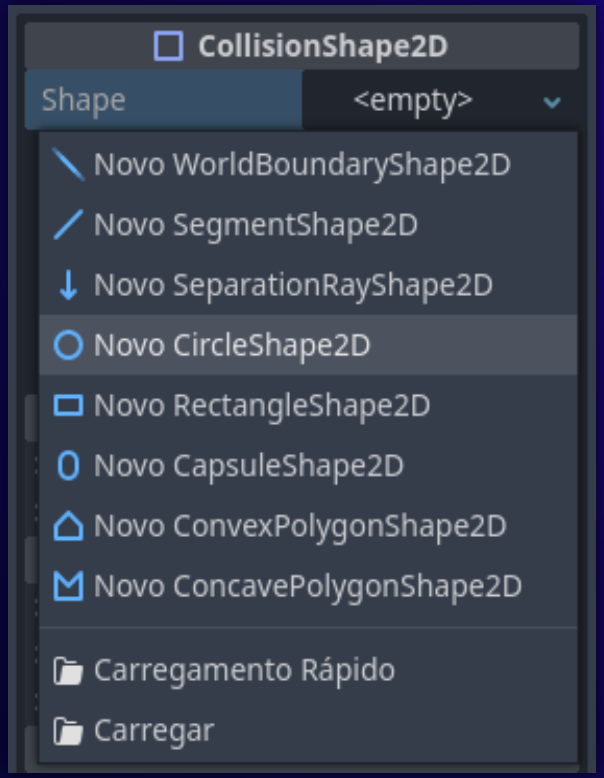


Warning

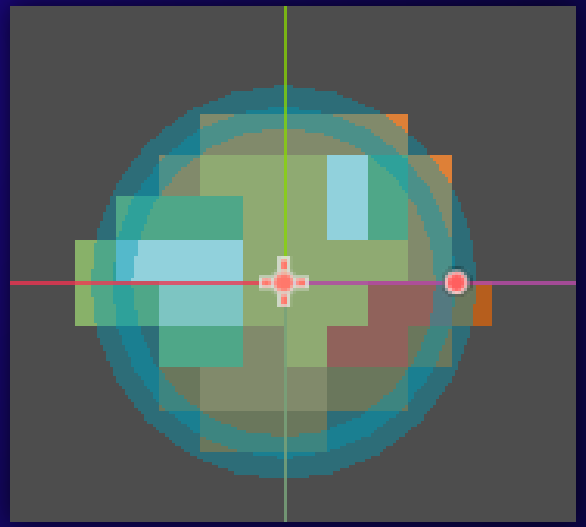


Collision shape

CircleShape

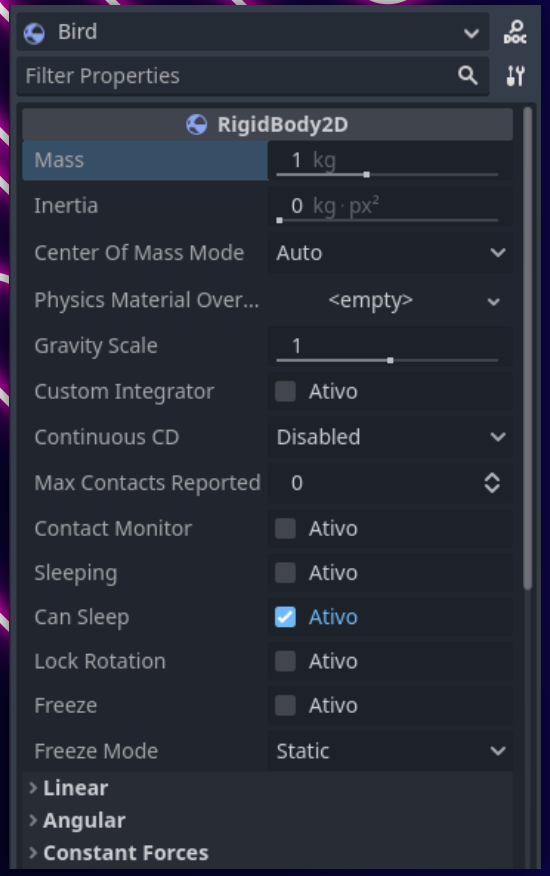


Fit shape

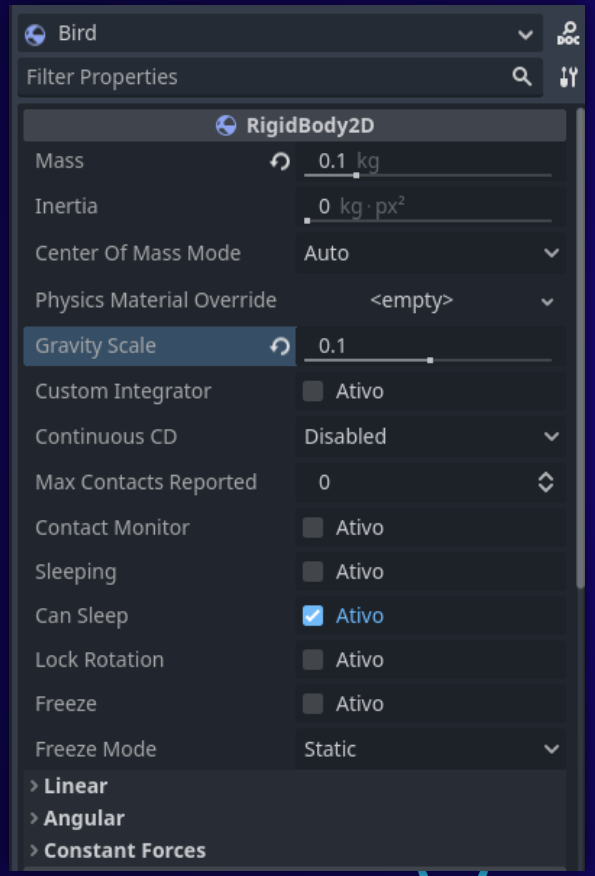


Gravity

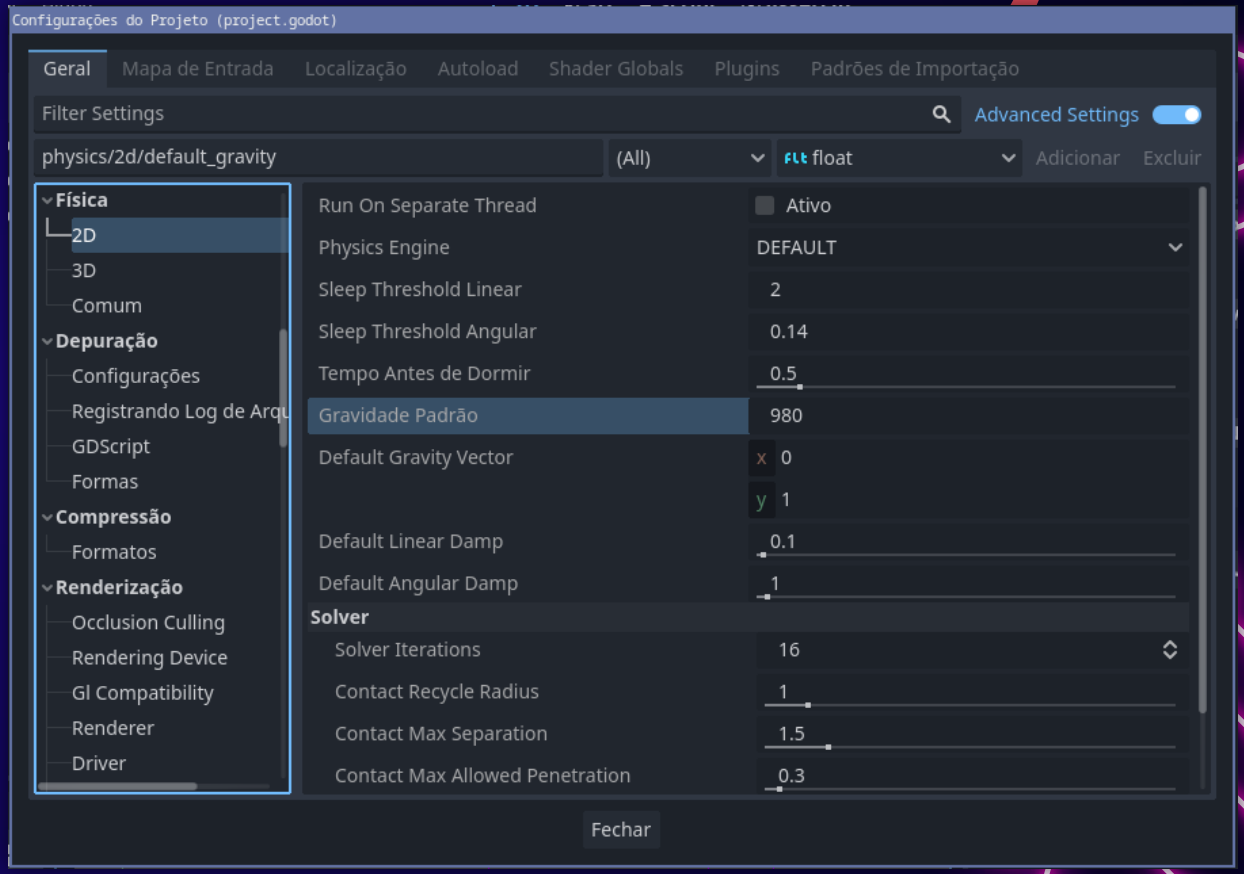
Set mass



Gravity scale

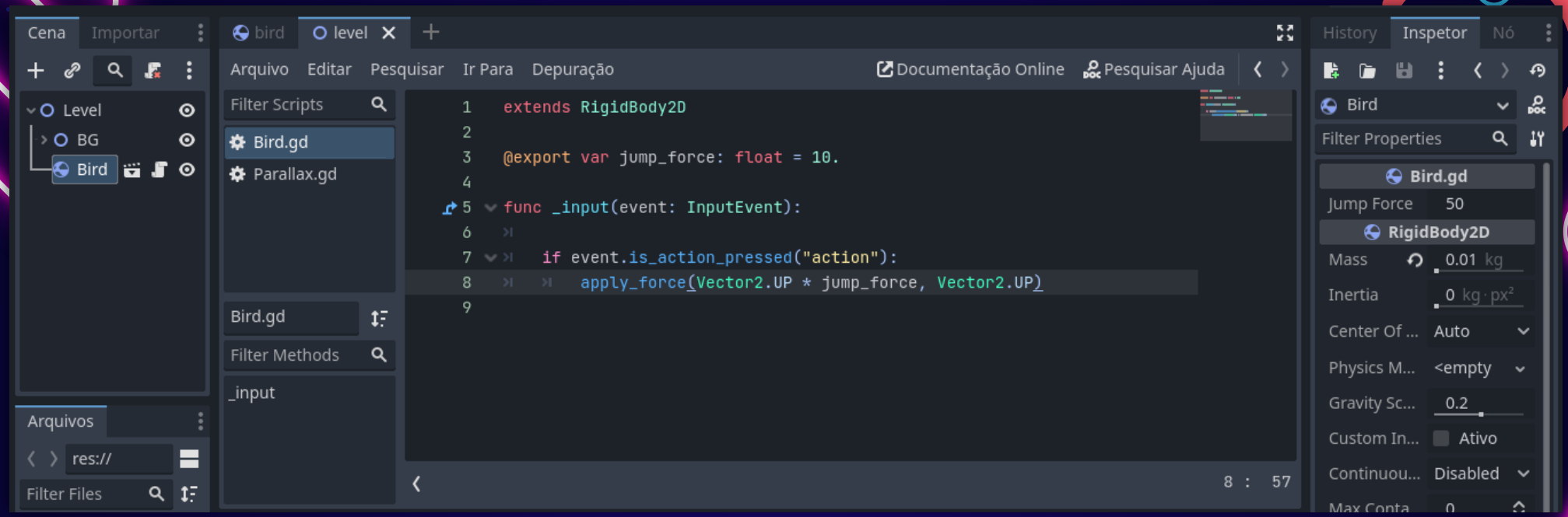


Defaults



Gravity fixed

Script do pássaro



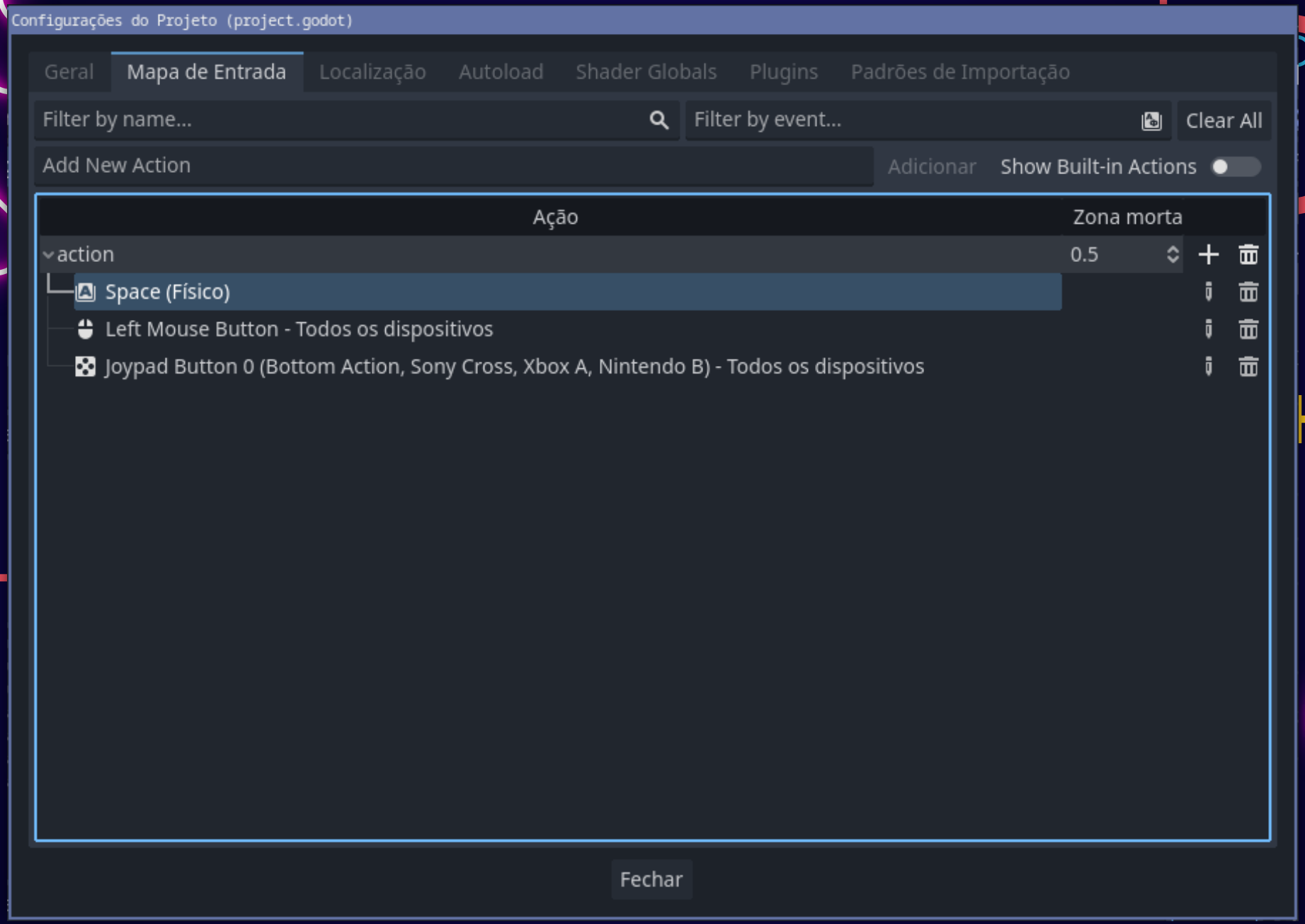
Apply force Docs

```
void apply_force(force: Vector2, position: Vector2 = Vector2(0, 0))
```

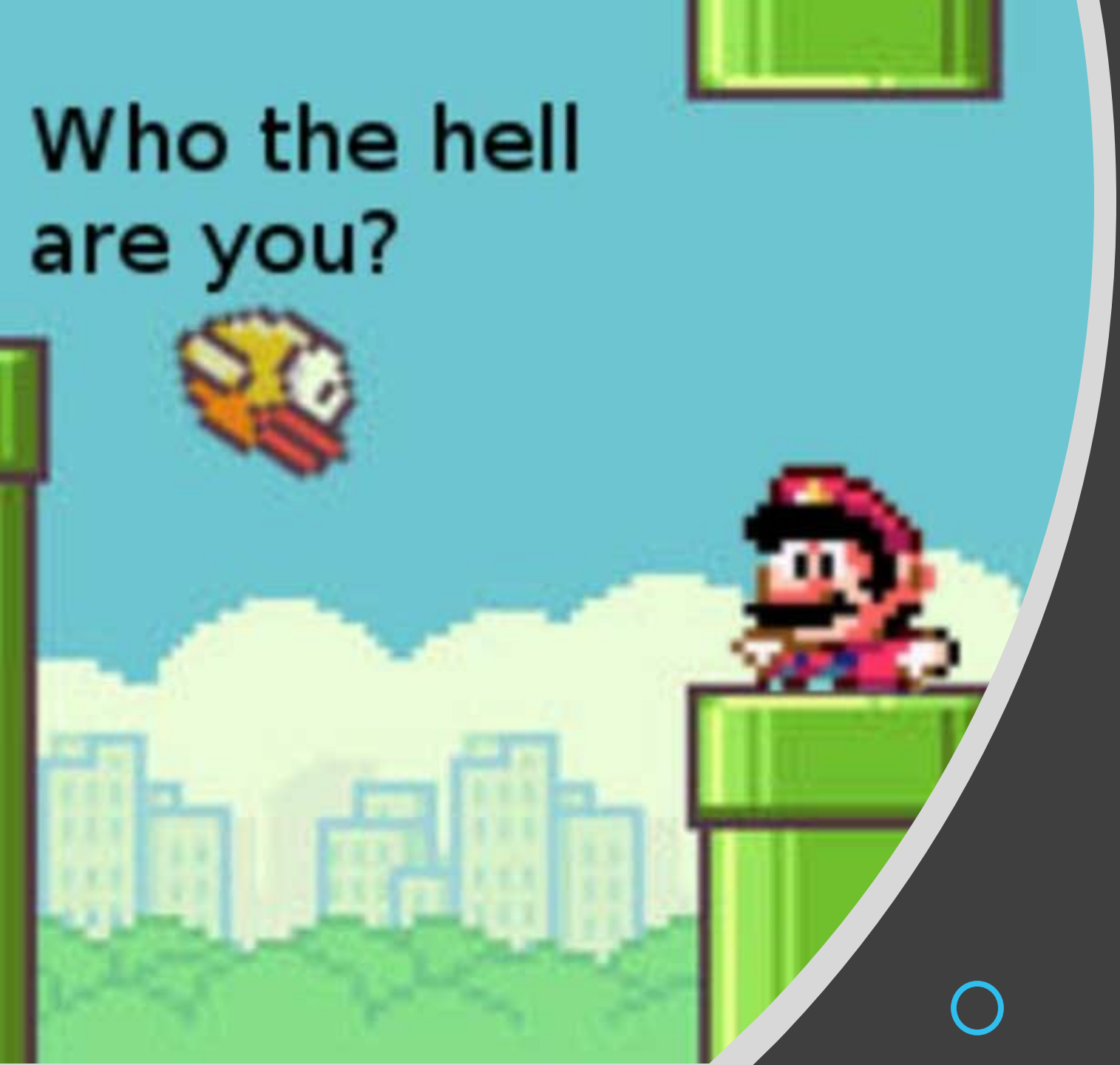
Applies a positioned force to the body. A force is time dependent and meant to be applied every physics update.

`position` is the offset from the body origin in global coordinates.

Mapa de entrada

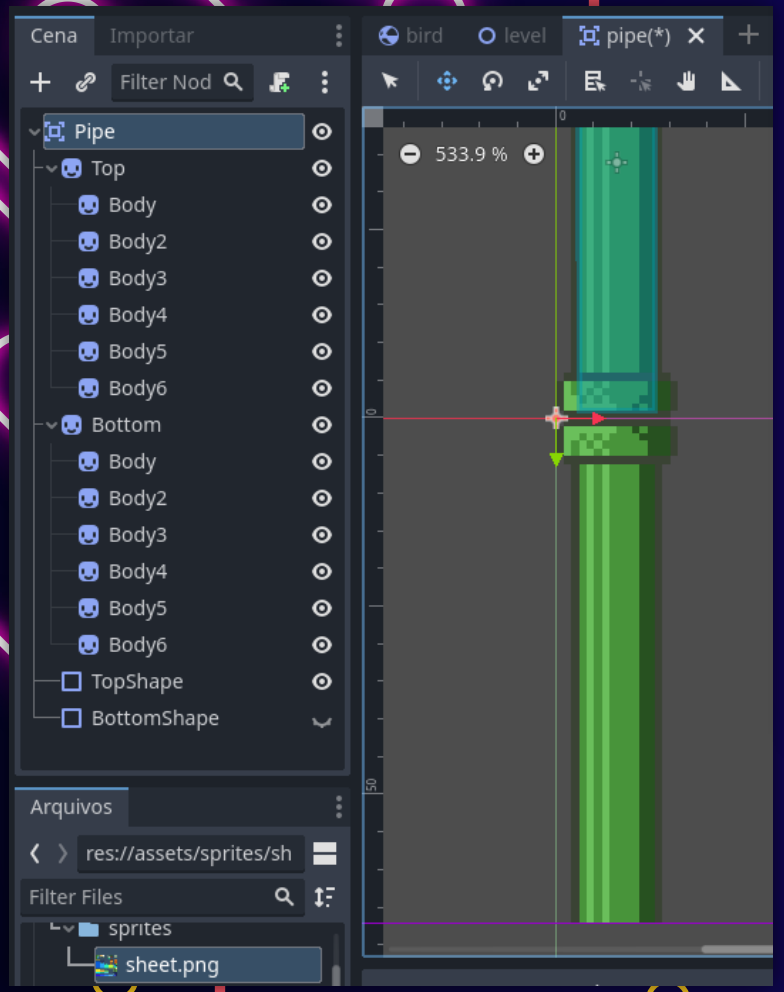


Fly



Canos

Cano



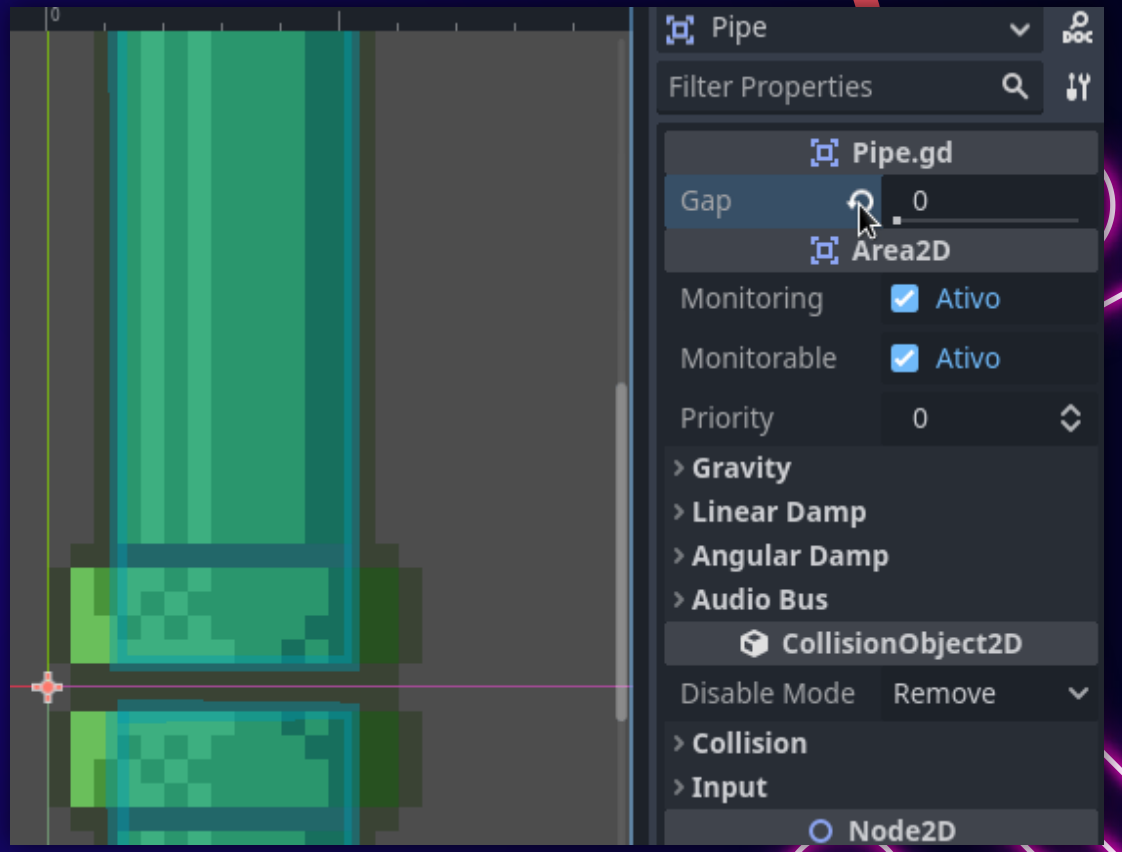
Script tool

```gdscript
@tool # Torna o código executável no editor
extends Area2D

## Distância entre canos
@export_range(0., 67.) var gap: float


func _ready():
    var top: Sprite2D = $Top
    var top_shape: CollisionShape2D = $TopShape

    top.position.y = -(gap + top.region_rect.size.y)
    top_shape.position.y = -(gap + top_shape.shape.size.y / 2 + 1)

```

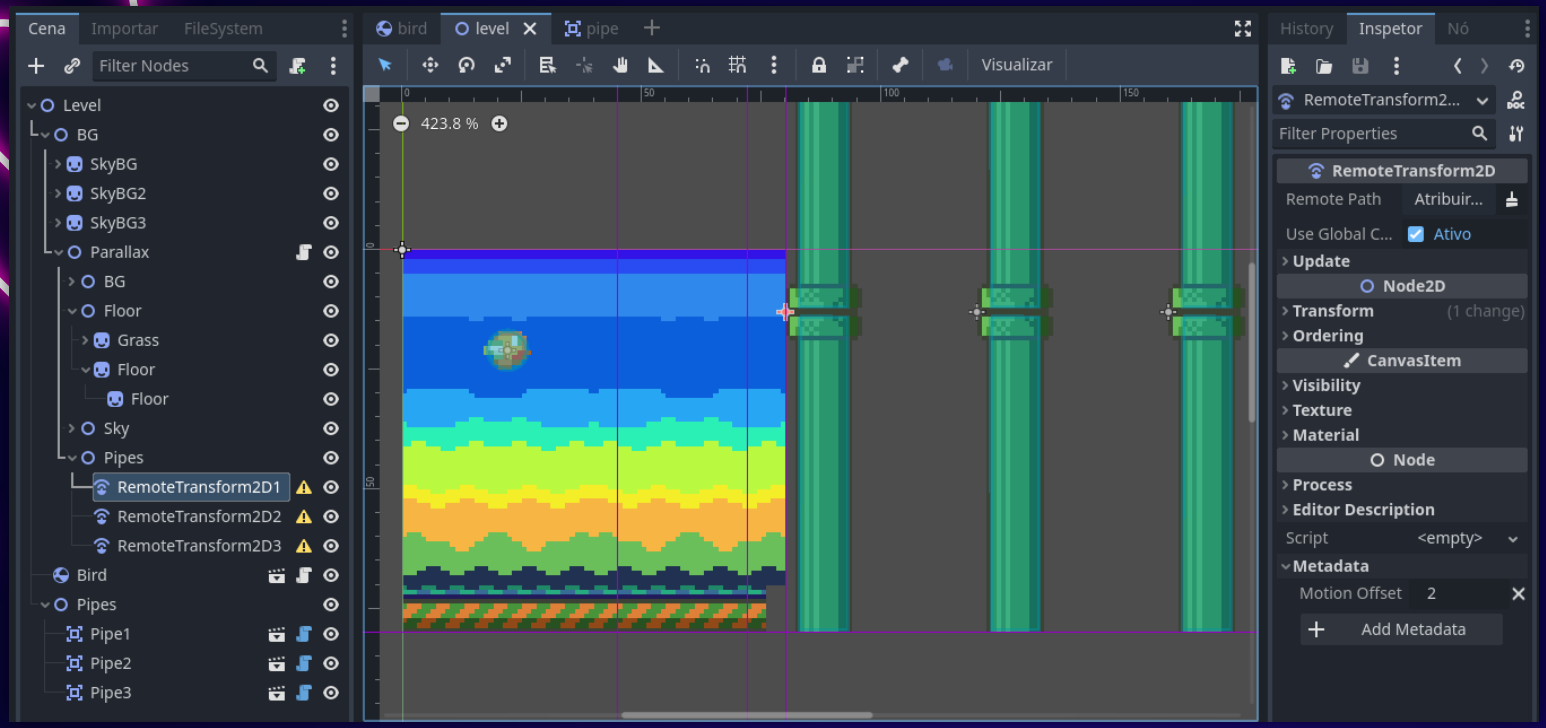
Setter

```gdscript
@tool # Torna o código executável no editor
extends Area2D

## Variável do comportamento que atualiza a cena com base no valor de `gap`
var _gap_update: Callable = func(): pass

## Distância entre canos
@export_range(0., 67.) var gap: float = 0:
    set(value):
        _gap_update.call(value)
        gap = value

@onready var top: Sprite2D = $Top
@onready var top_shape: CollisionShape2D = $TopShape


func _ready():
    _gap_update = update_gap
    update_gap(gap)


func update_gap(value: float):
    top.position.y = -(value + top.region_rect.size.y)
    top_shape.position.y = -(value + top_shape.shape.size.y / 2 + 1)

```

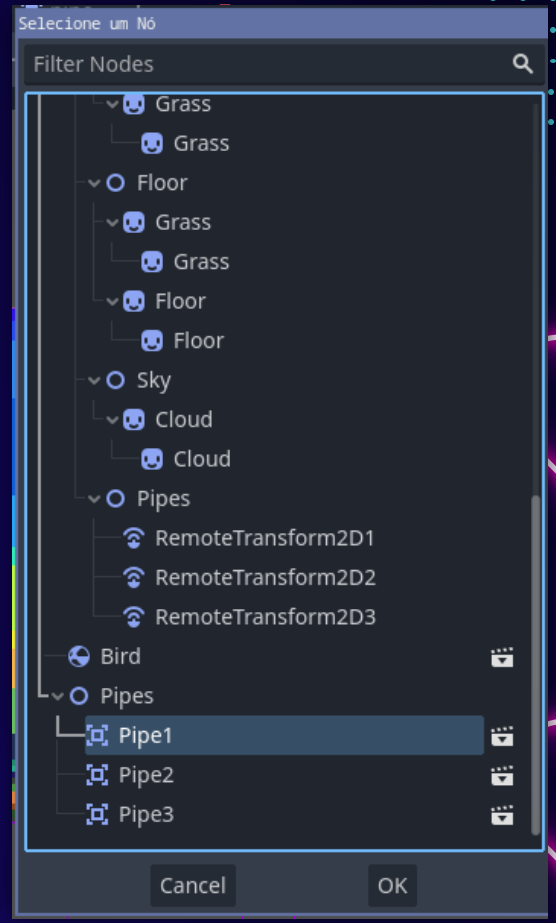
Gap



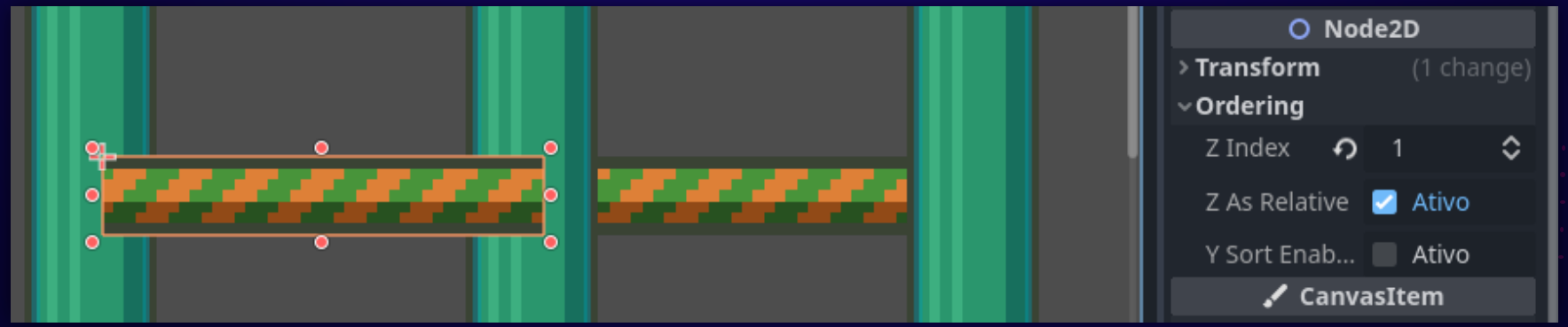
RemoteTransform2D



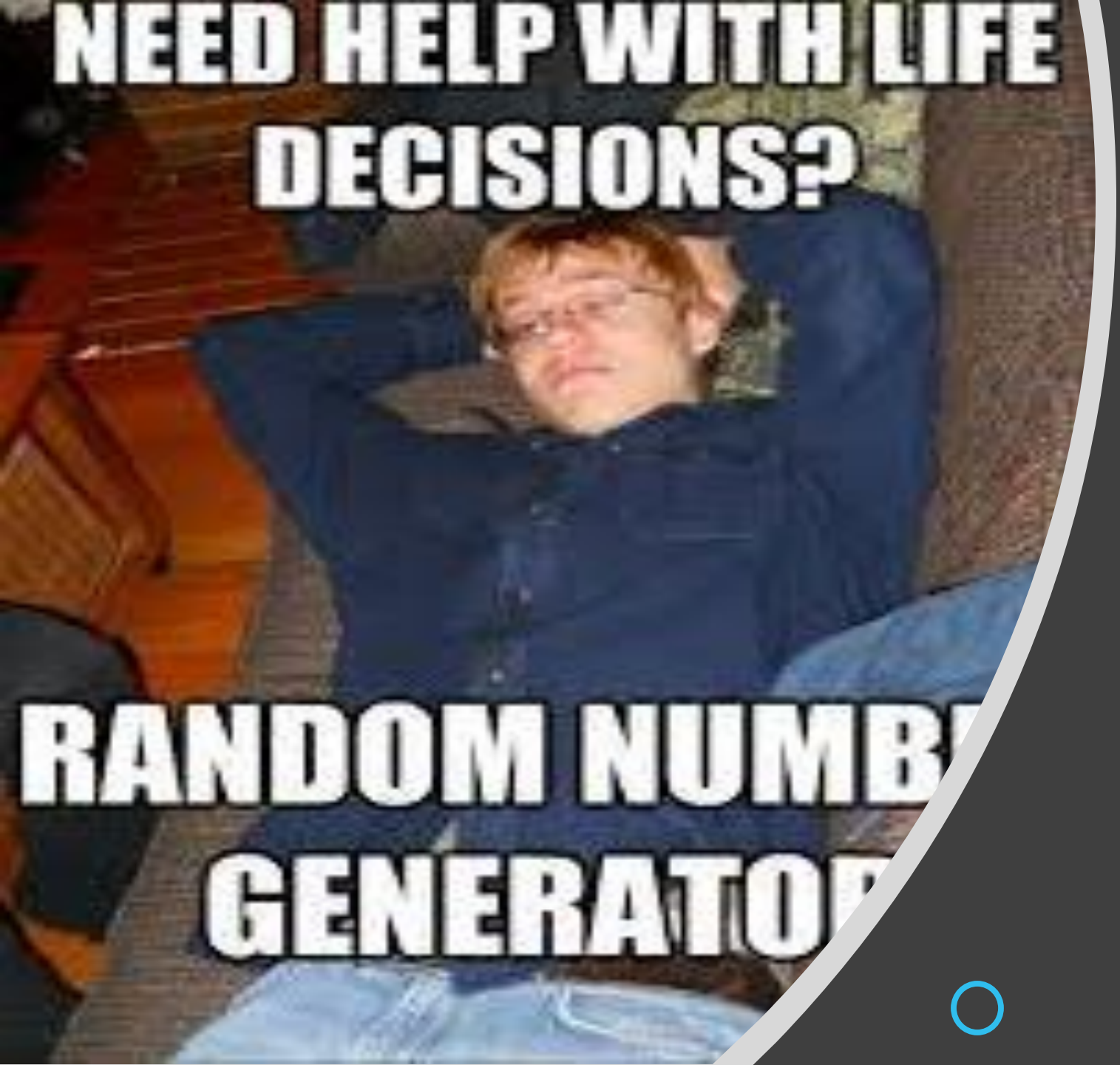
Remote path



Z index



Pipes



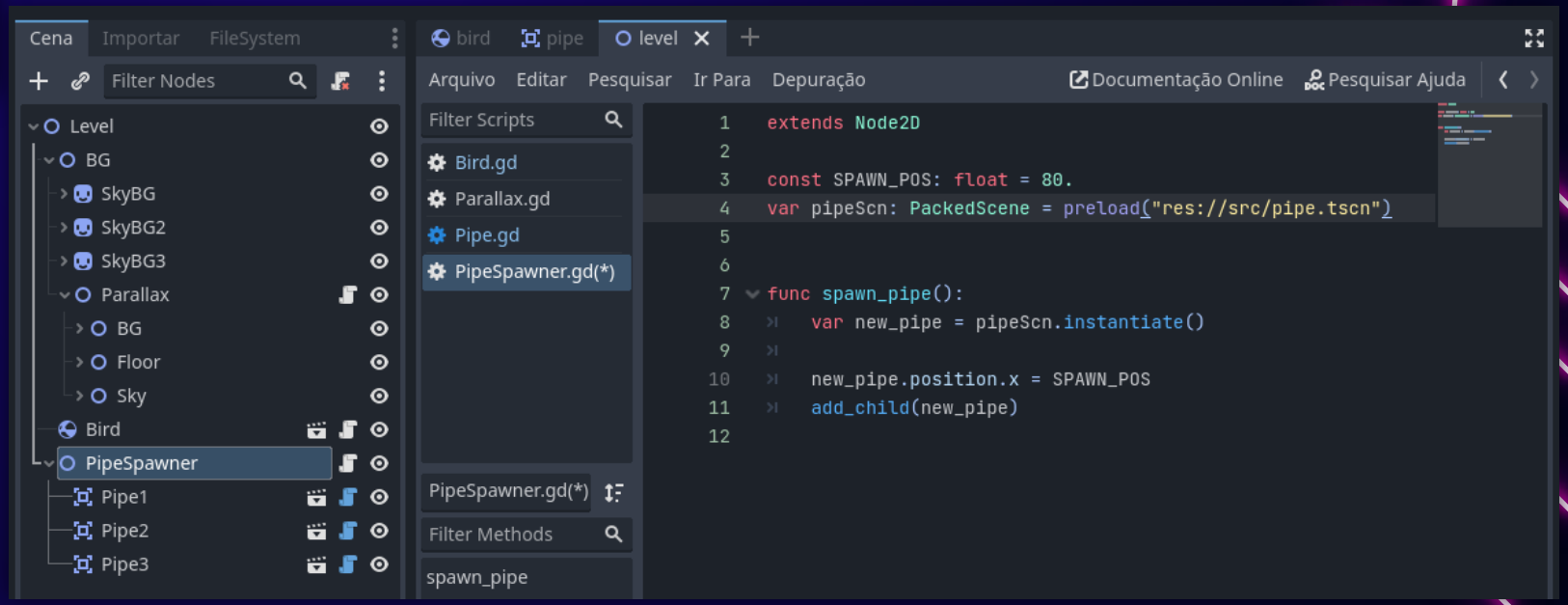
Geração aleatória

Result

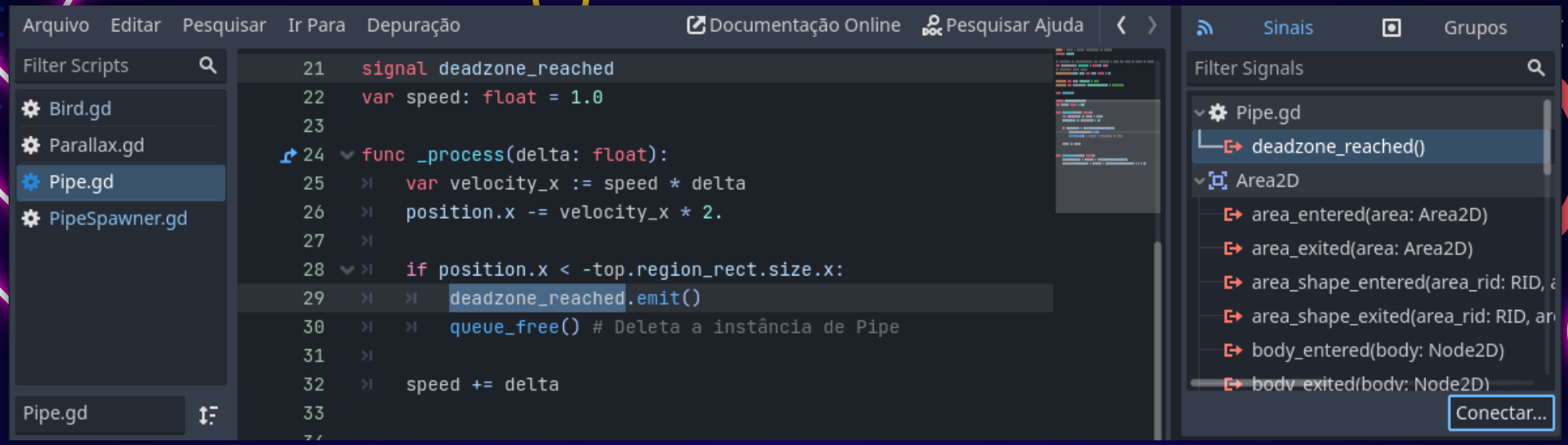
Individual logic sequence

```gdscript
15 func _ready():
16     _gap_update = update_gap
17     update_gap(gap)
18     # Desativa o processamento no Editor
19     set_process(not Engine.is_editor_hint())
20
21
22 const MOTION_OFFSET: float = 2.
23 var speed: float = 1.0
24 @onready var virtual_posx = position.x
25
26 func _process(delta: float):
27     var velocity_x := speed * delta * MOTION_OFFSET
28     virtual_posx -= velocity_x
29     position.x = floorf(virtual_posx)
30
31     if position.x < -top.region_rect.size.x:
32         queue_free() # Deleta a instância de Pipe
33
34     speed += delta
35
```

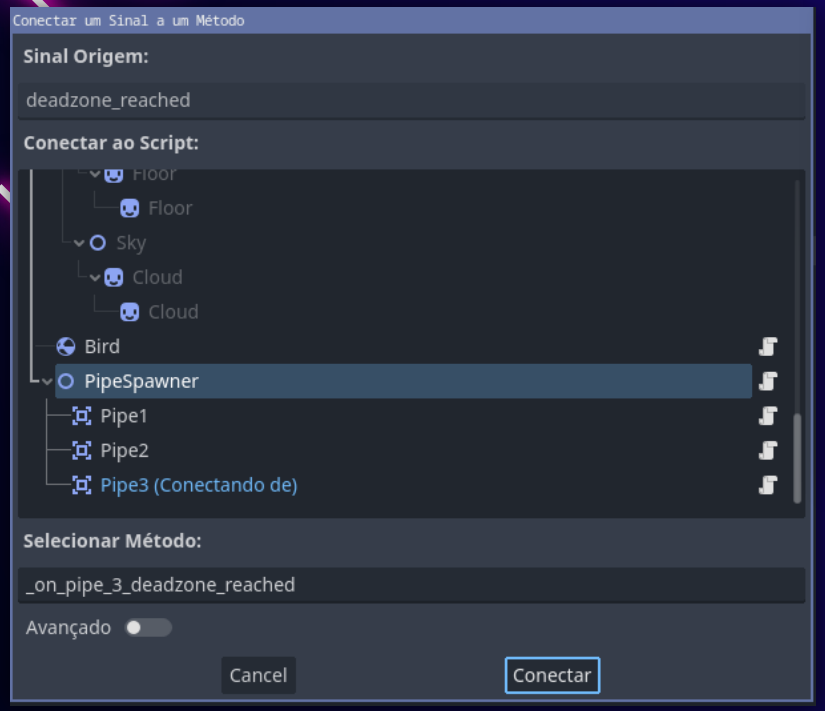
Spawner



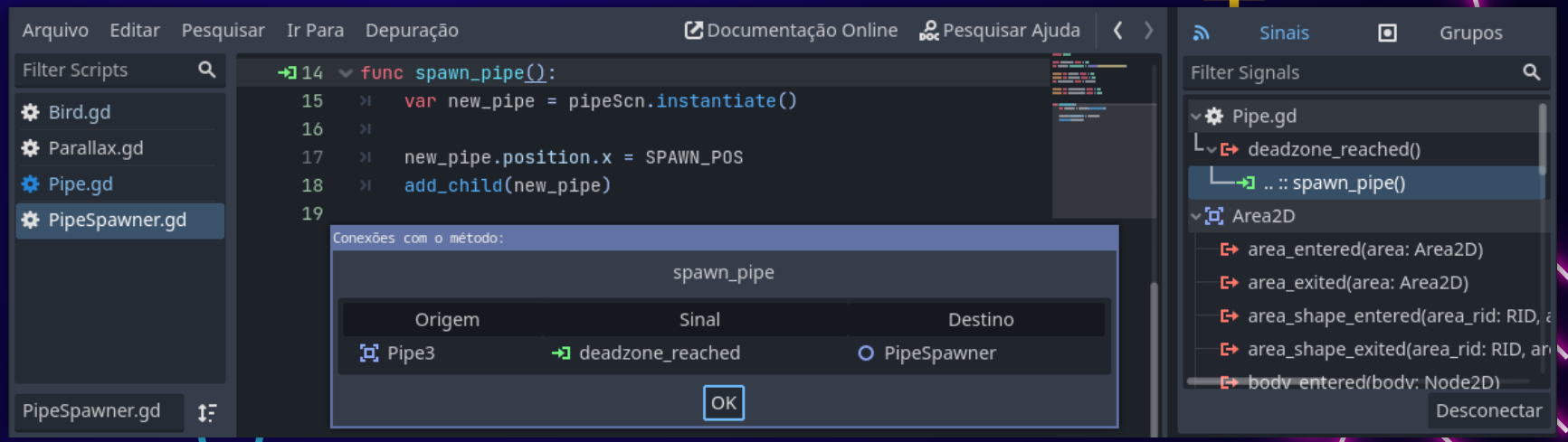
Sinal



Conectar



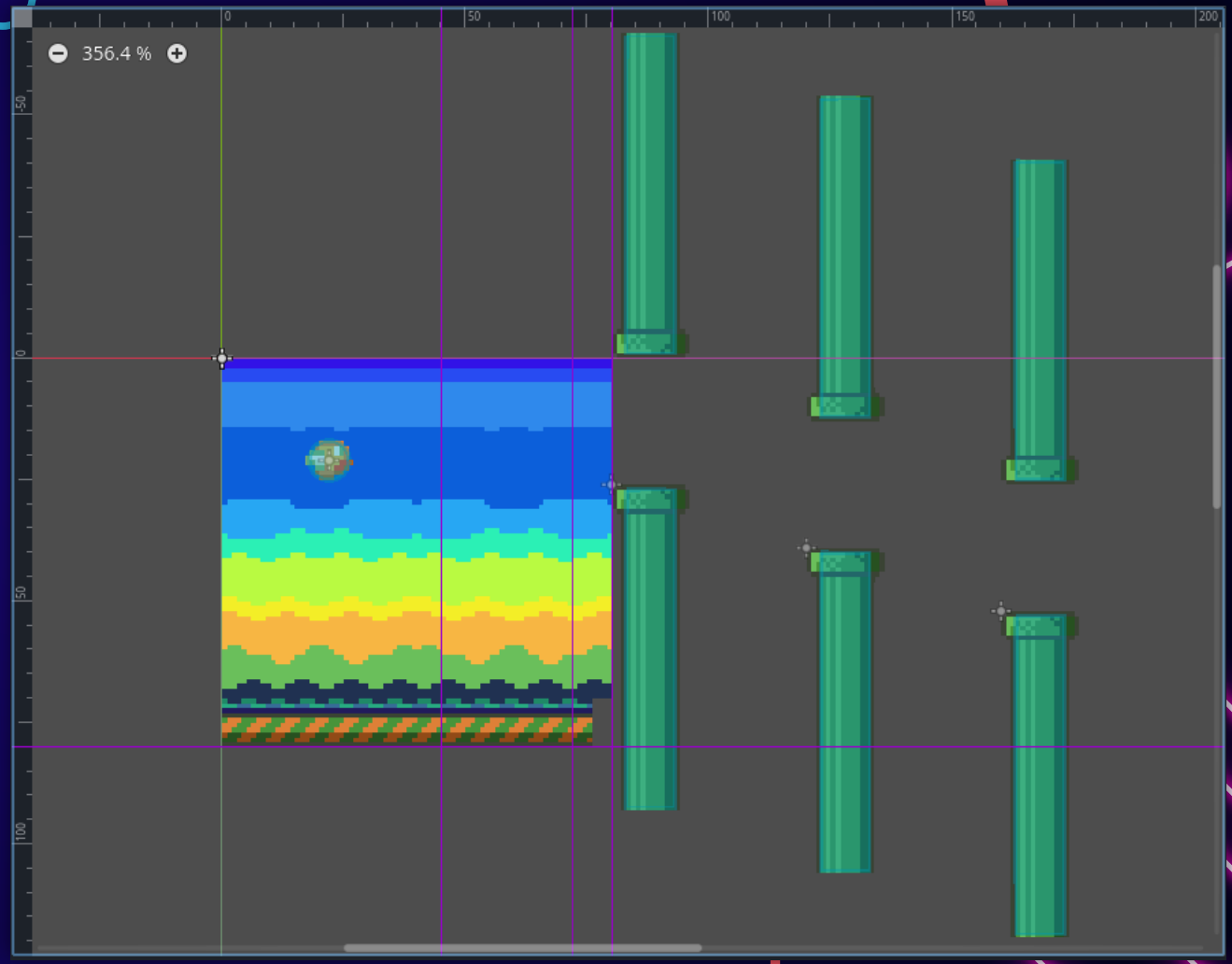
Conexão



Geração aleatória

```gdscript
extends Node2D

const SPAWN_POS: float = 80.
const Pipe = preload("res://src/Pipe.gd")
const PIPE_SCN: PackedScene = preload("res://src/pipe.tscn")

@export var gap_reduction: float = 0.1
@export var low_gap: float = 13.
@export var high_gap: float = 26.
@onready var current_gap: float = high_gap

@export var min_position: float = 26.
@export var max_position: float = 80. - 7. # Screen.height - Floor

func _ready():
    min_position = high_gap

func spawn_pipe():
    var new_pipe: Pipe = PIPE_SCN.instantiate()

    new_pipe.position.x = SPAWN_POS
    new_pipe.position.y = floorf(randf_range(min_position, max_position))
    new_pipe.gap = floorf(current_gap)

    # Update Gap
    current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)

    add_child(new_pipe)

```

Cena dos canos

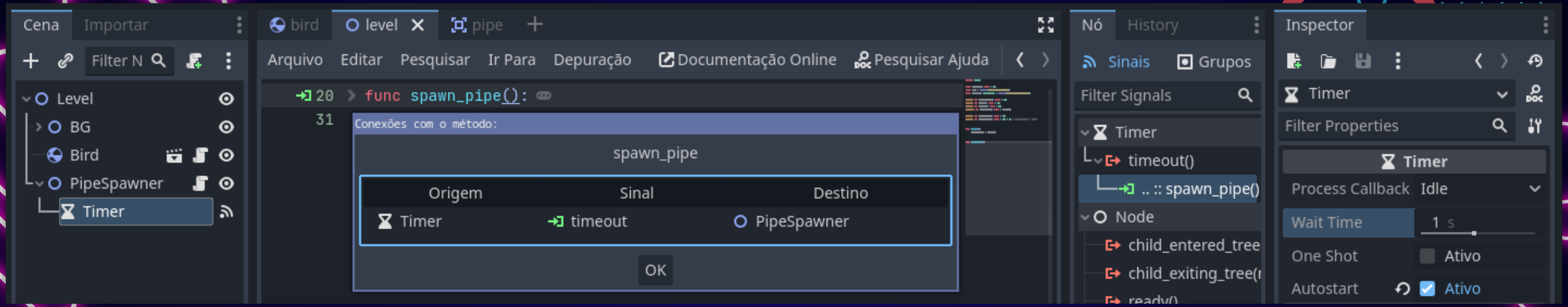


When you wake up at 6:59 and you set your timer at 7:00

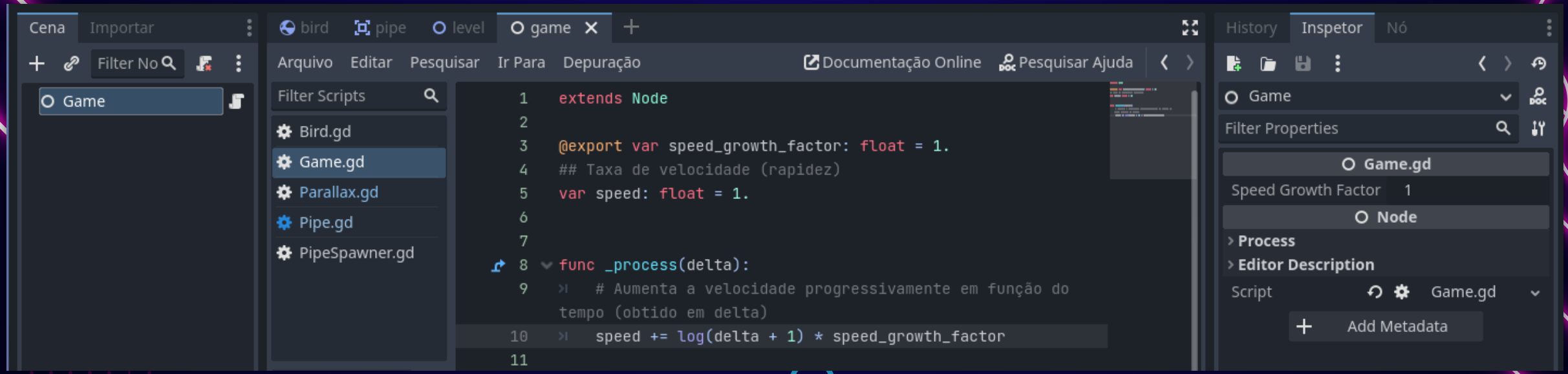
Timer

Result

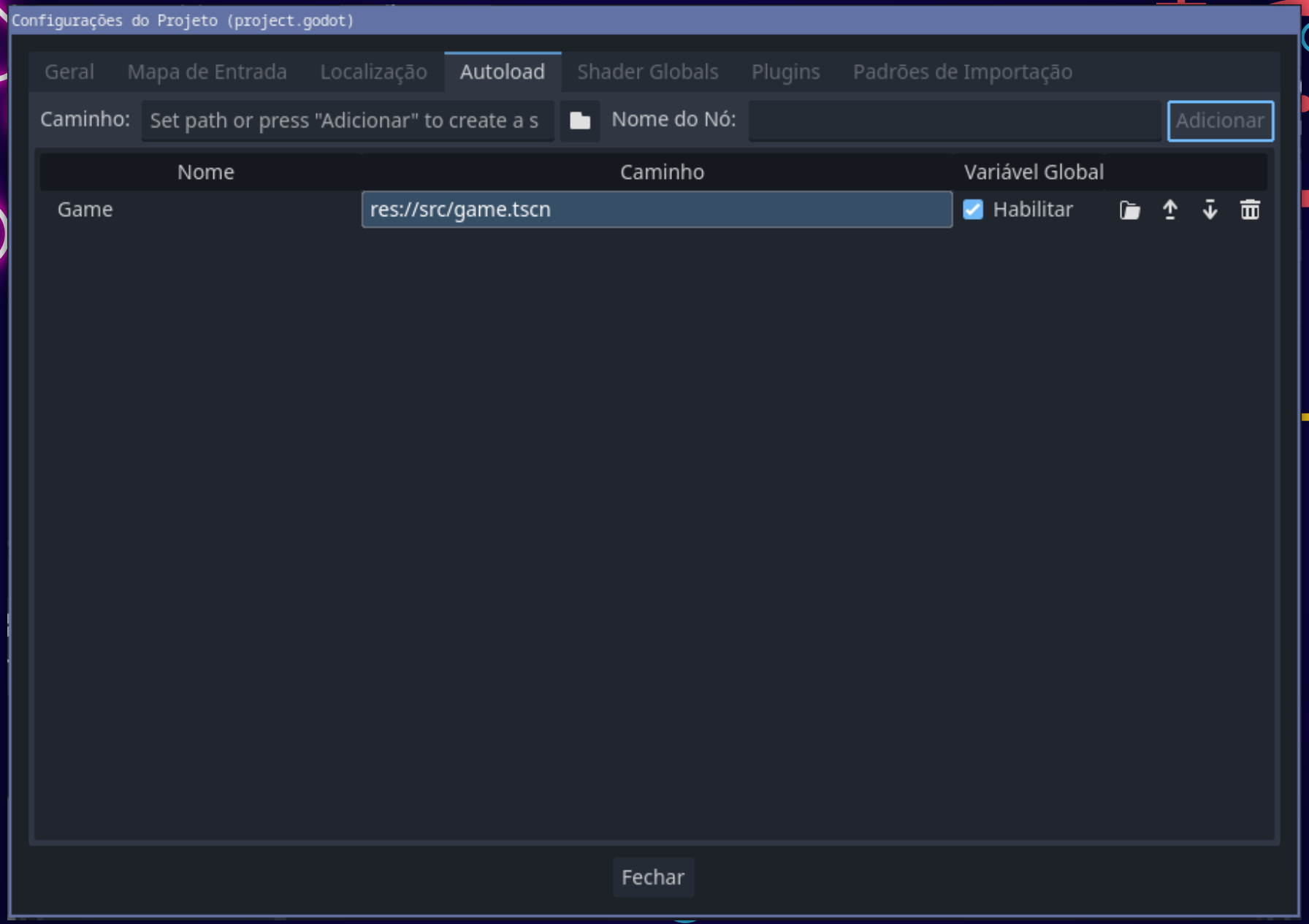
Timer connection



Script "Game"



Autoload


Configurações do Projeto (project.godot)
Geral
Mapa de Entrada
Localização
Autoload
Shader Globals
Plugins
Padrões de Importação
Caminho:
Set path or press "Adicionar" to create a s
Nome do Nó:
Adicionar
Nome
Caminho
Variável Global
Game
res://src/game.tscn
Habilitar
Fechar

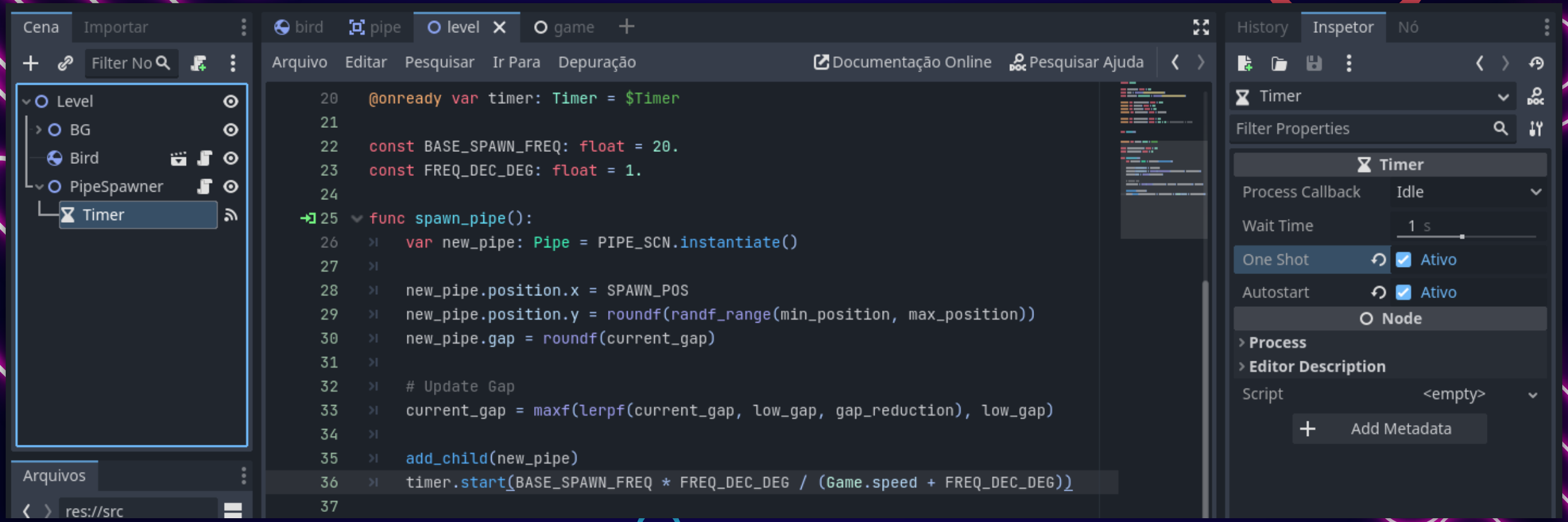
Refactor: pipe speed

```
22  const MOTION_OFFSET: float = 2.
23  @onready var virtual_posx = position.x
24
25  func _process(delta: float):
26      var velocity_x := Game.speed * delta * MOTION_OFFSET
27      # Velocidade atual do movimento no eixo-x
28
29      virtual_posx -= velocity_x
30      position.x = roundf(virtual_posx)
31
32      if position.x < -top.region_rect.size.x:
33          queue_free() # Deleta a instância de Pipe
34
```

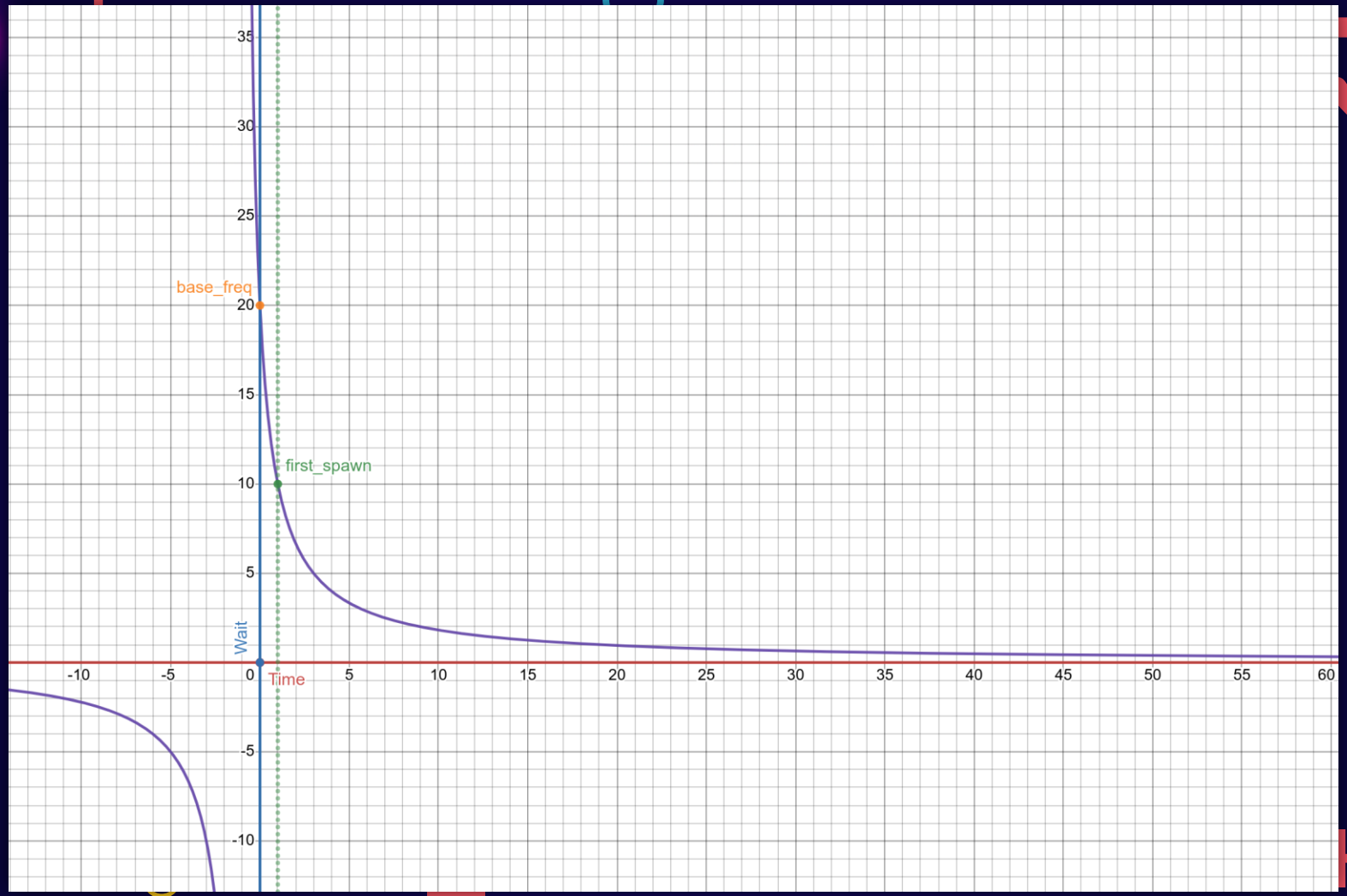
Refactor: parallax speed

```
28  func _process(delta):
29      # Velocidade atual do movimento no eixo-x
30      var velocity_x: float = Game.speed * delta
31
32      for child in children:
33          var offset: float = child.get_meta("loop_offset")
34
35          for grandchild in child.get_children():
36              # Move os Sprites com base na velocidade atual
37              var new_pos := grandchild.get_meta("virtual_posx") as float - \
38                  velocity_x * grandchild.get_meta("motion_offset") as float
39              var end_pos := new_pos + offset
40
41              if end_pos < 0:
42                  # end_pos será o quanto de espaço em x ultrapassou o eixo y
43                  new_pos = offset + end_pos
44
45              grandchild.set_meta("virtual_posx", new_pos)
46              grandchild.position.x = roundf(new_pos)
47
```

Timer restart



Wait time function

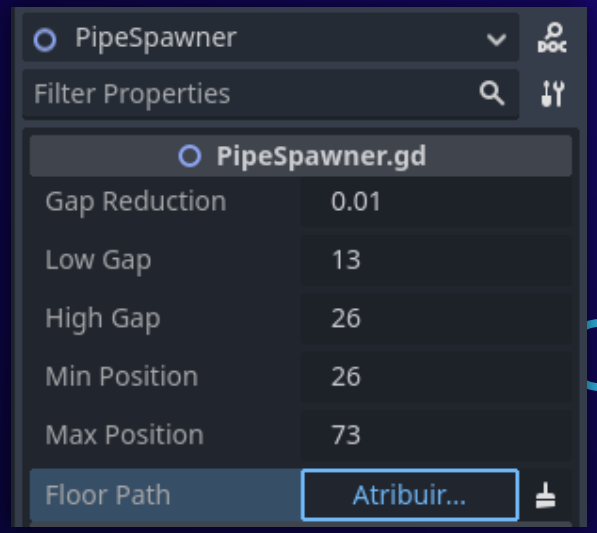


Timer restart

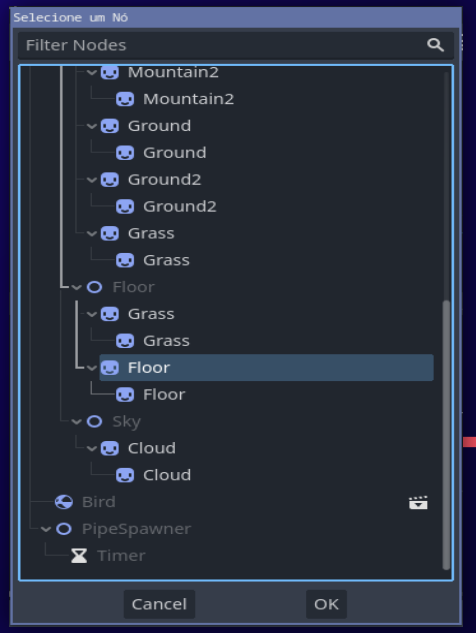
Fix flicker

```gdscript
@export_node_path(Sprite2D)
var floor_path: NodePath
@onready var floor := get_node(floor_path) as Sprite2D

func spawn_pipe():
    var new_pipe: Pipe = PIPE_SCN.instantiate()
    var floor_virtual_posx := floor.get_meta("virtual_posx") as float

    new_pipe.position.x = SPAWN_POS + \
            floor_virtual_posx - int(floor_virtual_posx)
    new_pipe.position.y = roundf(randf_range(min_position, max_position))
    new_pipe.gap = roundf(current_gap)

    # Update Gap
    current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)

    add_child(new_pipe)
    timer.start(BASE_SPAWN_FREQ * FREQ_DEC_DEG / (Game.speed + FREQ_DEC_DEG))

```

Path inspector



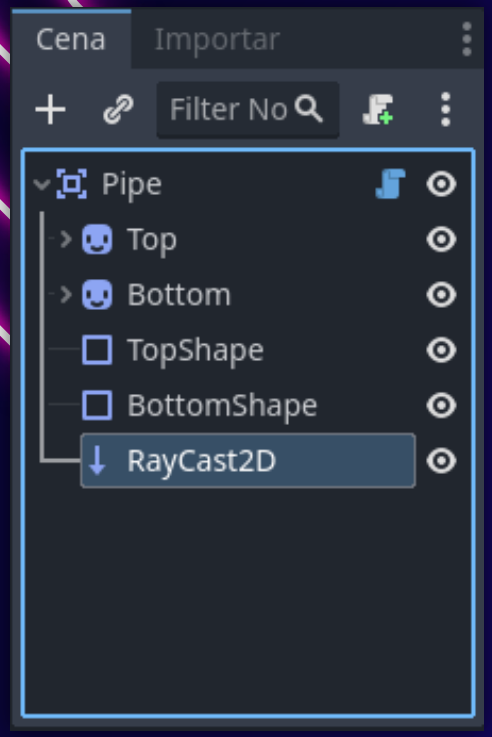
Path to floor



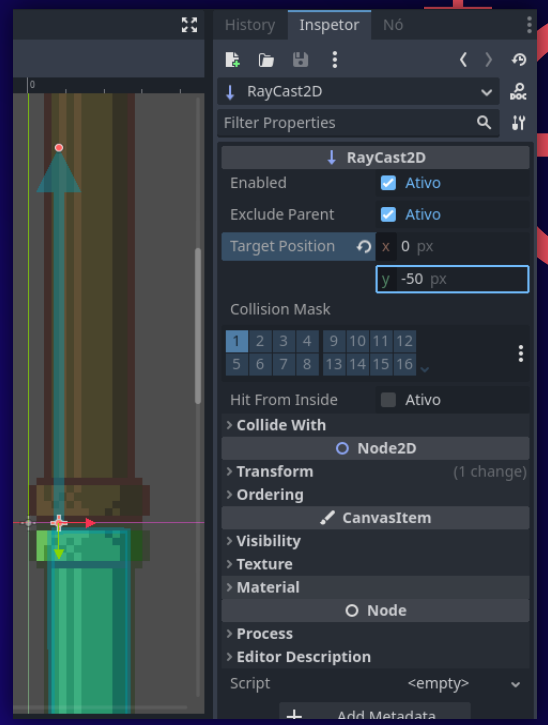
Final result

Score, Raycasting & HUD

Add RayCast2D



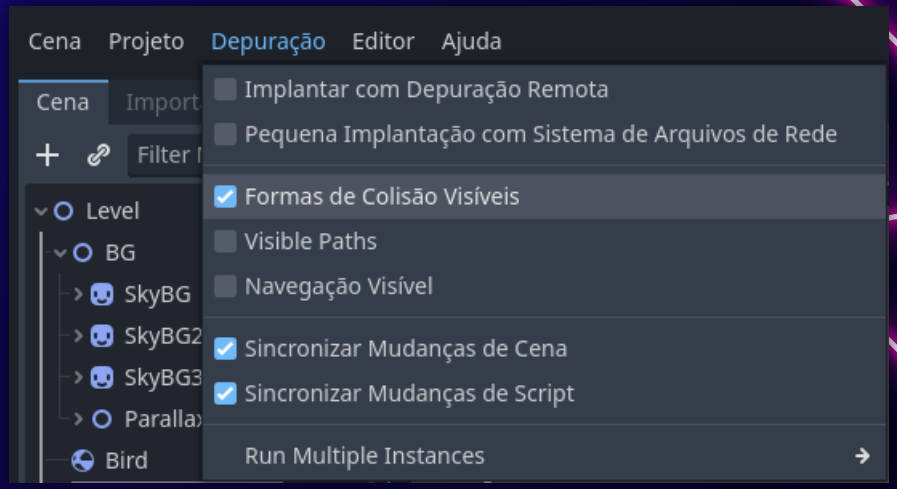
Target position



Pause

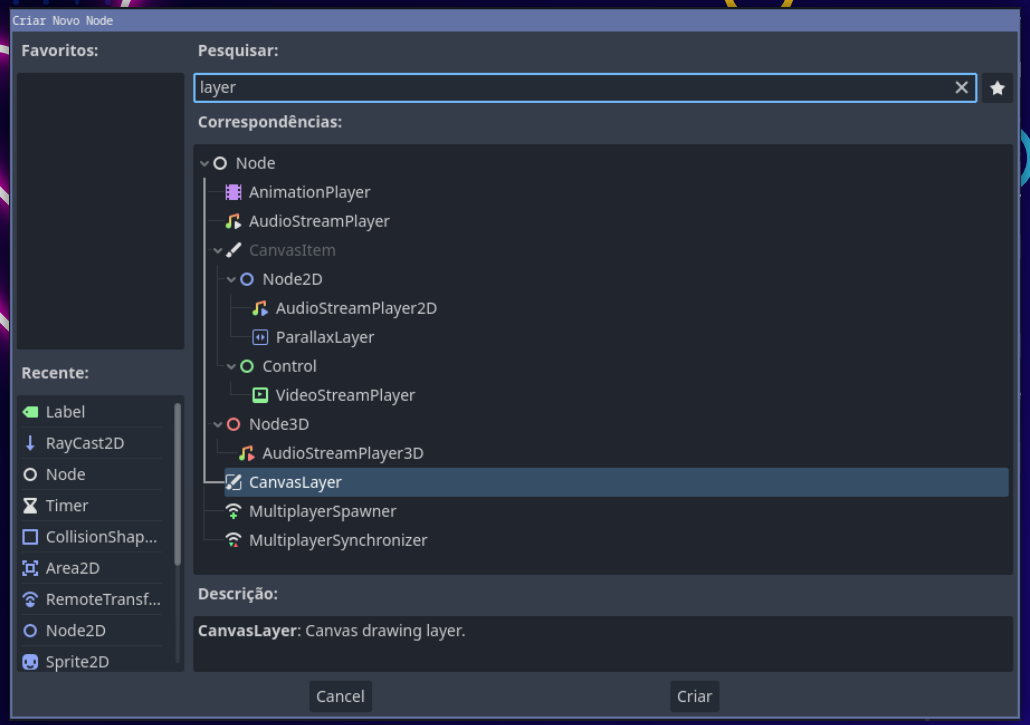
```gdscript
@onready var raycast := $RayCast2D as RayCast2D

func _physics_process(delta: float):
    if raycast.is_colliding():
        get_tree().paused = true # Pausa o jogo
```

Visible collisions

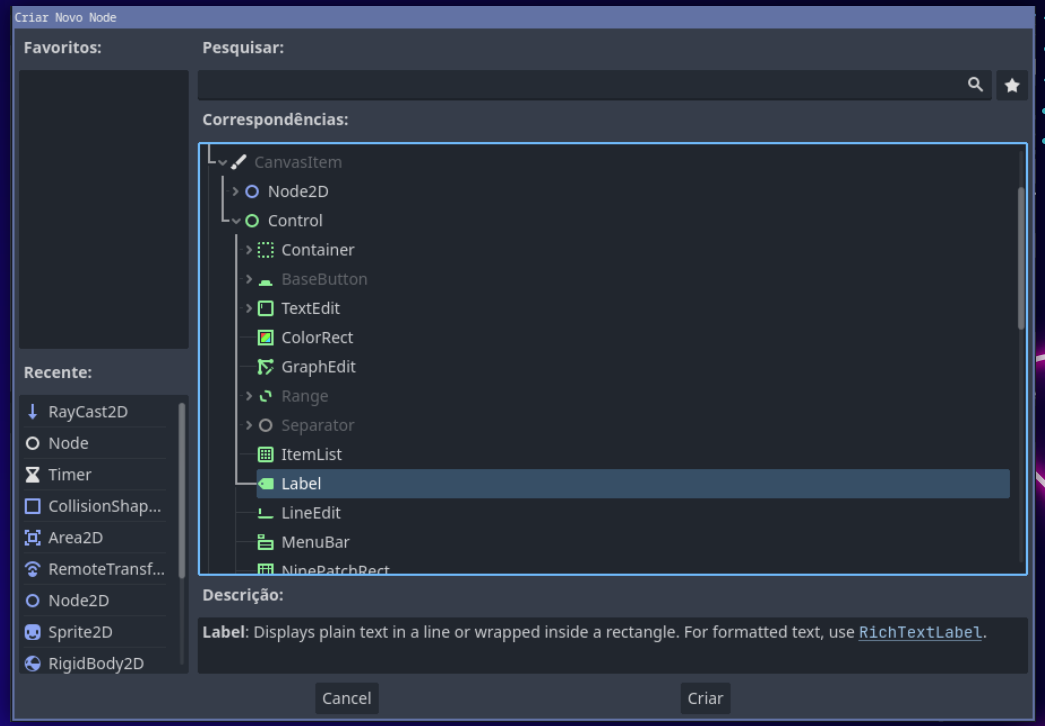


RayCast

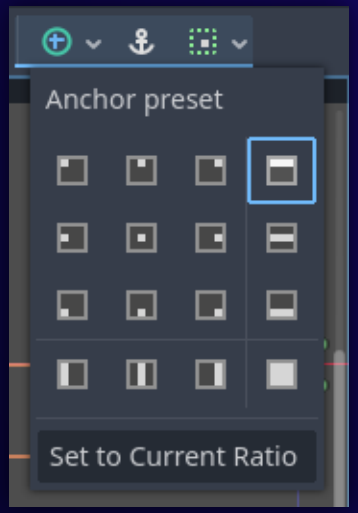
CanvasLayer



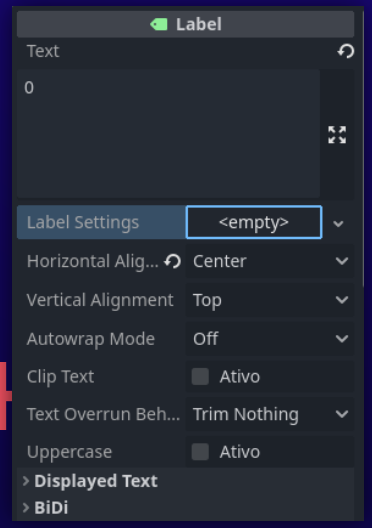
Label



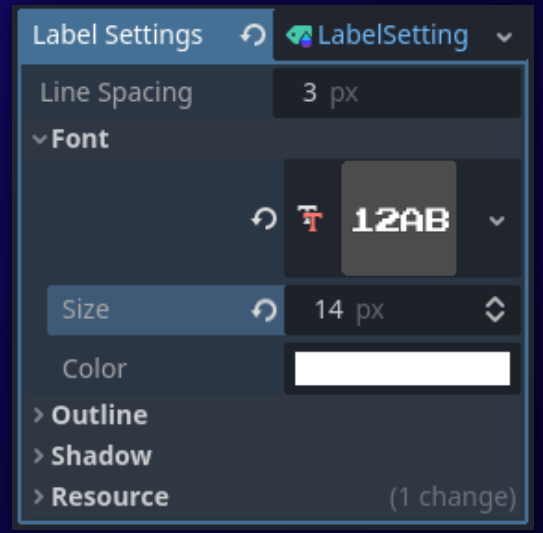
Anchors



Inspector



Settings



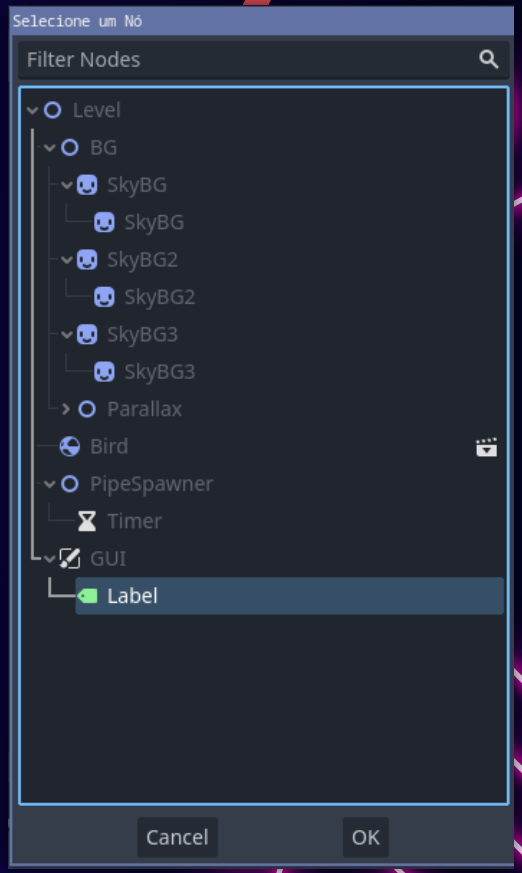
Scored signal

```gdscript
signal scored
@onready var raycast := $RayCast2D as RayCast2D

func _physics_process(delta: float):
    if raycast.is_colliding():
        scored.emit()
        raycast.queue_free()
        get_tree().paused = true # Pausa o jogo

```

Change score

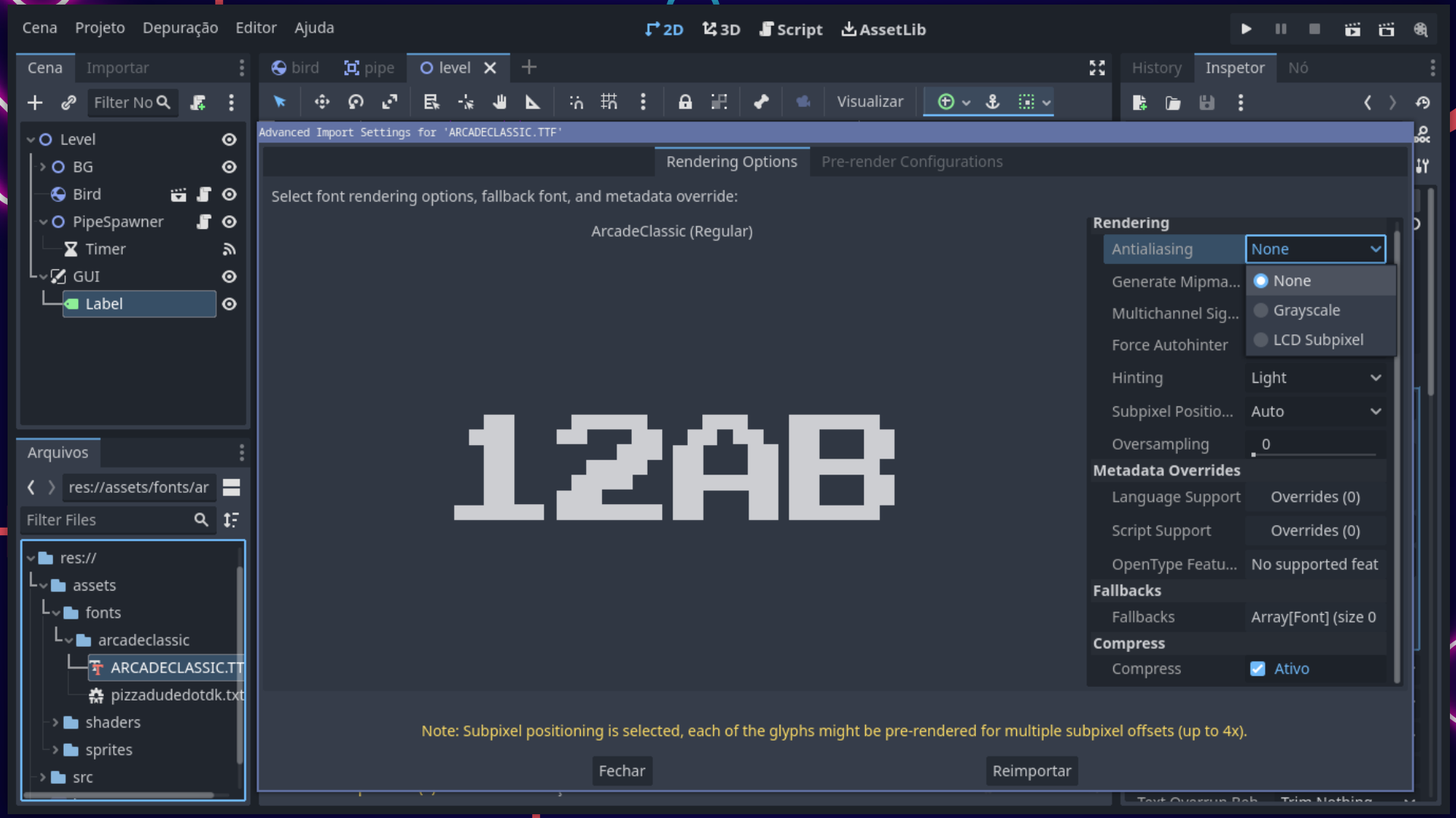
```gdscript
@export_node_path(Label)
var score_path: NodePath
@onready var score_label := get_node(score_path) as Label
var score: int = 0

func spawn_pipe():
    var new_pipe: Pipe = PIPE_SCN.instantiate()
    var floor_virtual_posx := floor.get_meta("virtual_posx") as float

    new_pipe.position.x = SPAWN_POS + \
            floor_virtual_posx - int(floor_virtual_posx)
    new_pipe.position.y = roundf(randf_range(min_position, max_position))
    new_pipe.gap = roundf(current_gap)
    # Update Gap
    current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)

    # Conecta-se ao evento "scored" pra atualizar a UI (Inteface do Usuário)
    new_pipe.scored.connect(func update_gui():
        score += 1
        score_label.text = str(score)
    )

    add_child(new_pipe)
    timer.start(BASE_SPAWN_FREQ * FREQ_DEC_DEG / (Game.speed + FREQ_DEC_DEG))

```

Connect



HUD

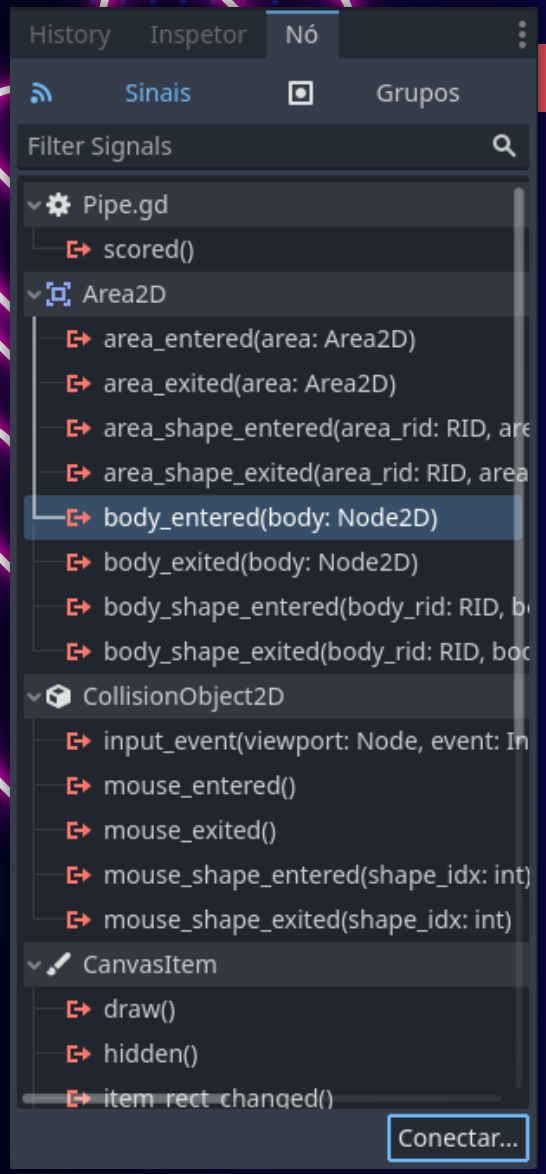
Fix aliasing



Final result

Colisões & Gameplay loop

Body entered



Export nodes

```gdscript
# No editor será exportado o path usado pra carregar a instância automaticamente
@export var score_label: Label
var score: int = 0

const Bird = preload("res://src/Bird.gd")
@export_node_path("PhysicsBody2D")
var bird_path: NodePath
@onready var bird := get_node(bird_path) as Bird

@export var freeze_node_paths: Array[NodePath]
@export var freeze_nodes: Array[Node] = []

func _ready():
    for path in freeze_node_paths:
        freeze_nodes.append(get_node(path))

    min_position = high_gap

```

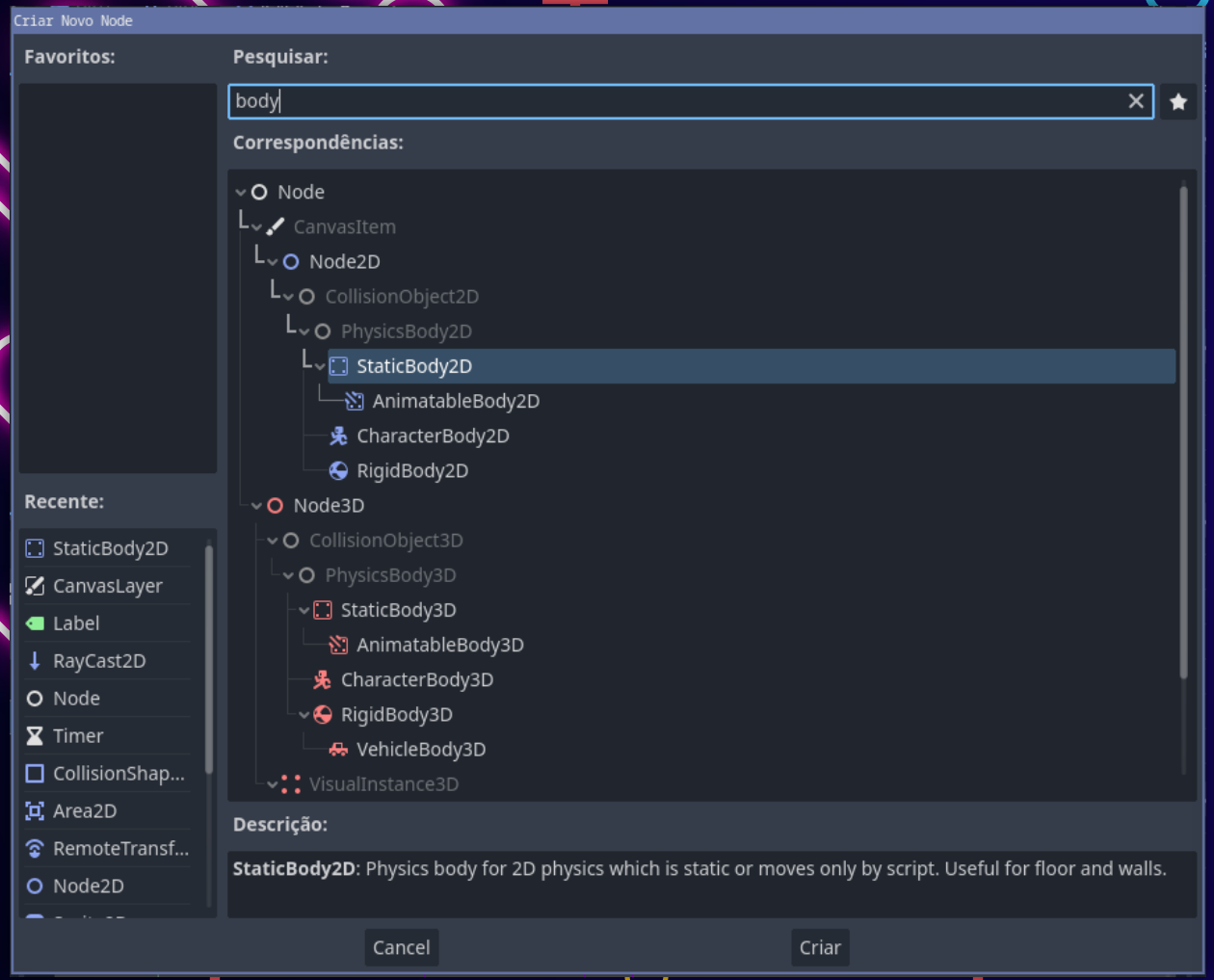
Freeze node

Pause nodes

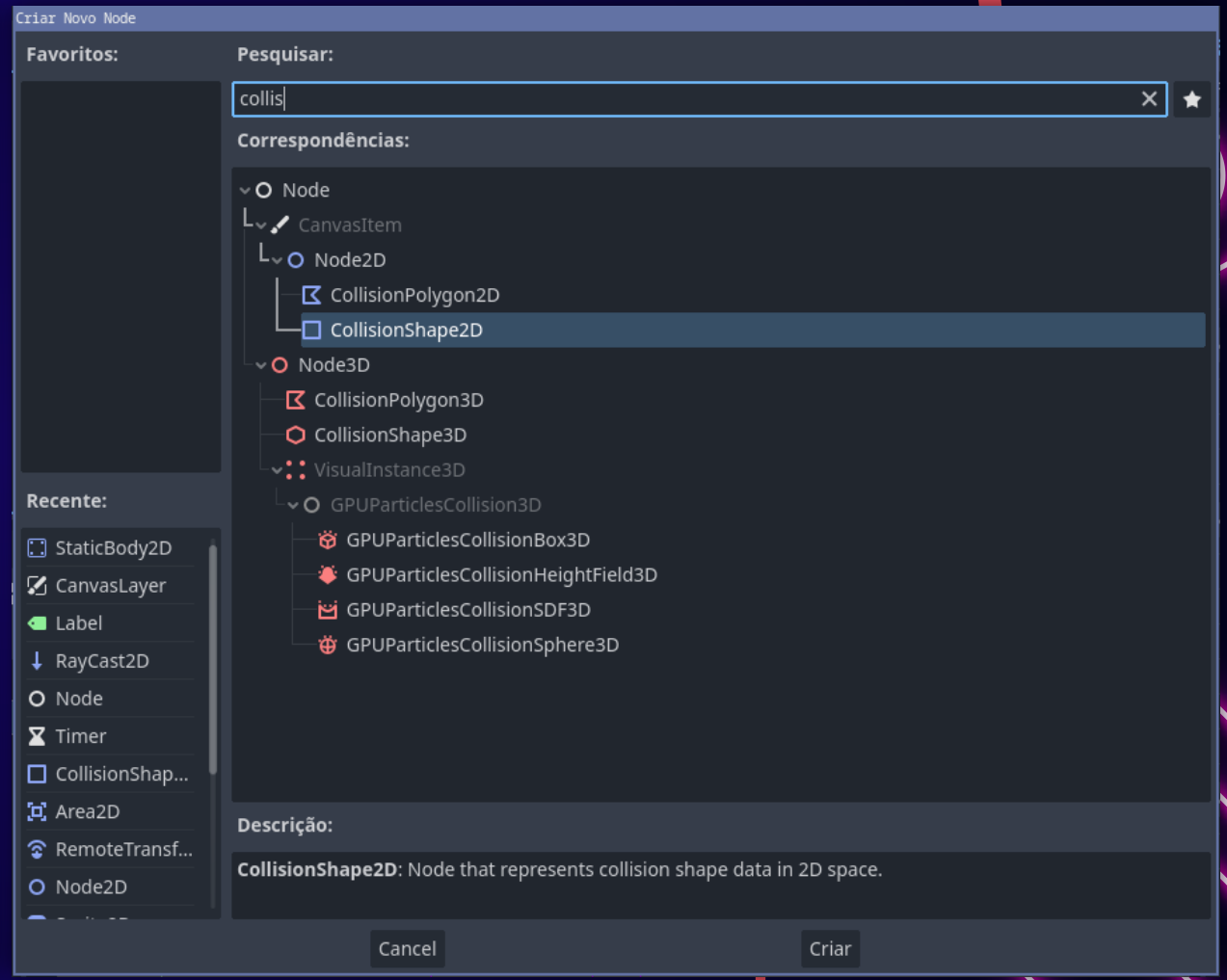
```gdscript
44 func spawn_pipe():
45     var new_pipe: Pipe = PIPE_SCN.instantiate()
46     var floor_virtual_posx := floor_sprite.get_meta("virtual_posx") as float
47     new_pipe.position.x = SPAWN_POS + \
49     new_pipe.position.y = roundf(randf_range(min_position, max_position))
50     new_pipe.gap = roundf(current_gap)
51     # Update Gap
52     current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)
53     # Conecta-se ao evento "scored" pra atualizar a UI (Inteface do Usuário)
54     new_pipe.scored.connect(func update_gui():
55         score += 1
56         score_label.text = str(score)
57     )
58     # Conecta-se ao evento "body entered" para resetar o Game Loop
59     new_pipe.body_entered.connect(func restart(_arg):
60         bird.set_process_input(false)
61         bird.apply_impulse(Vector2.LEFT * .2, Vector2.LEFT)
62 
63         await get_tree().physics_frame # Aguarda a física ser processada
64         for node in freeze_nodes:
65             node.process_mode = Node.PROCESS_MODE_DISABLED
66             # Para de processar o nó como se o jogo estivesse pausado para ele
67     )
68 
69     add_child(new_pipe)
70     timer.start(BASE_SPAWN_FREQ * FREQ_DEC_DEG / (Game.speed + FREQ_DEC_DEG))
71 
```

Collision

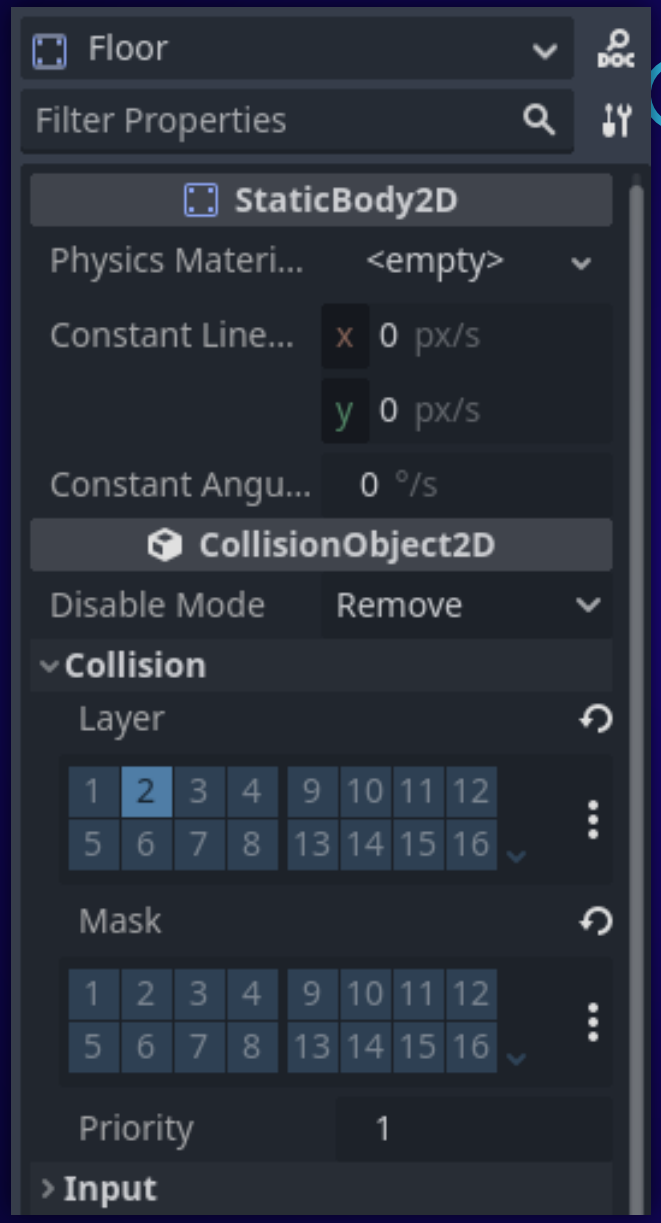
StaticBody2D



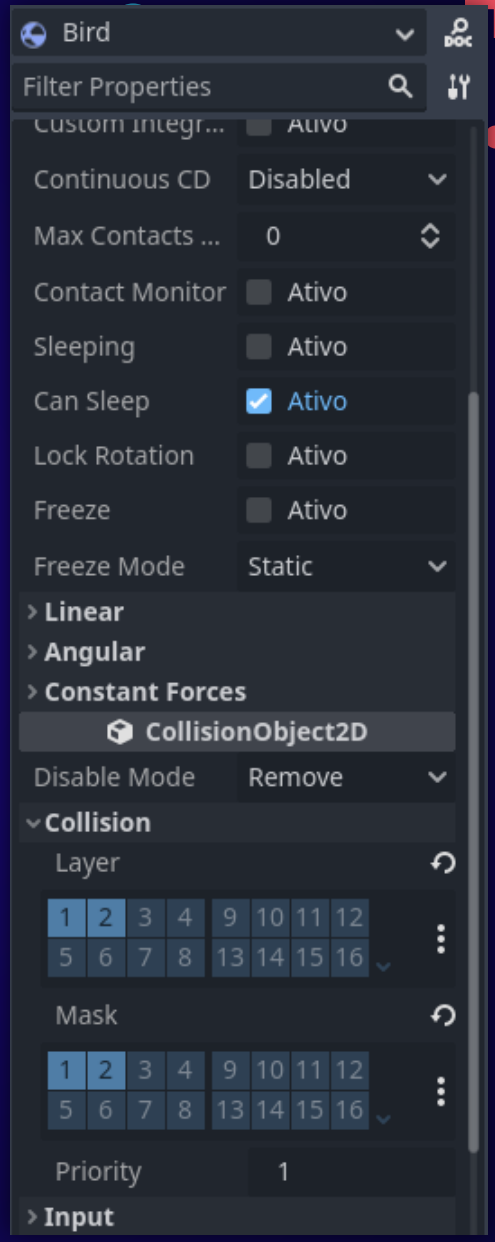
CollisionShape2D



Floor layer



Bird layer



Layered Collision

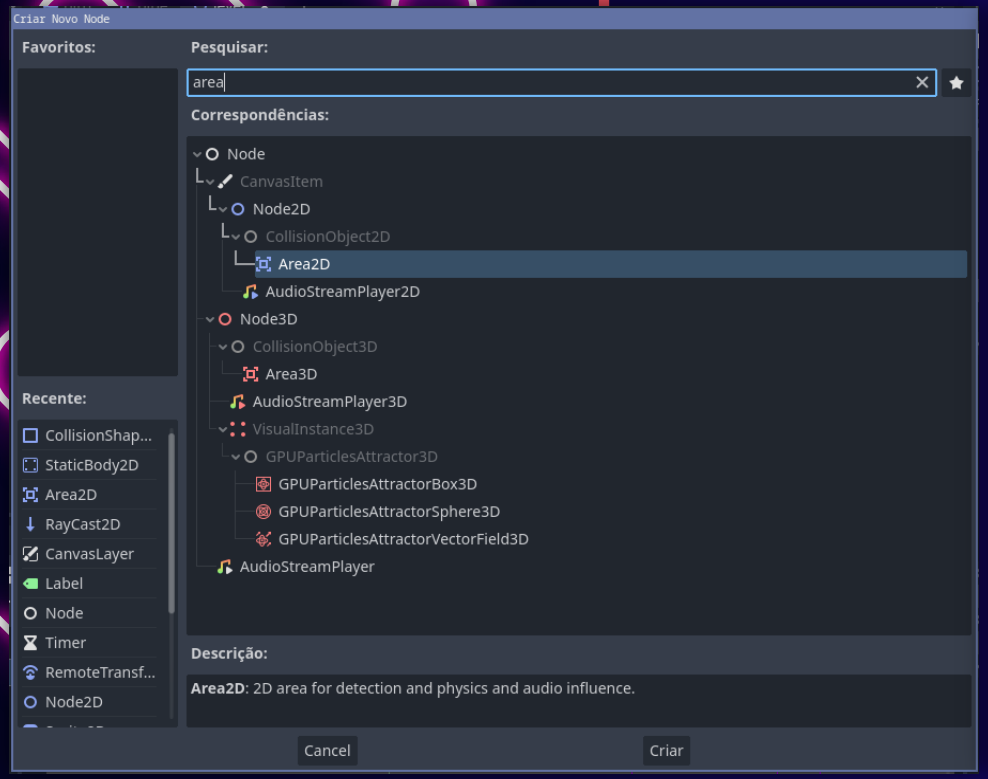
Floor Collision

Ceil Collision

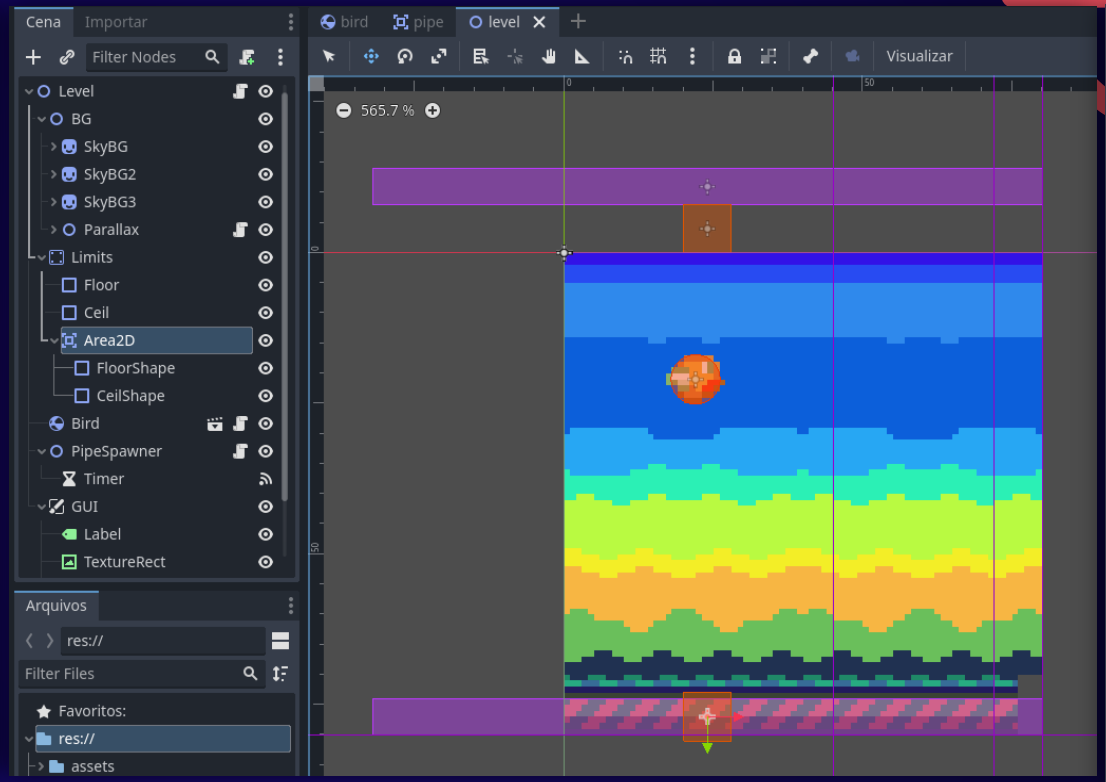
Pipe Collision

Pipe tip Collision

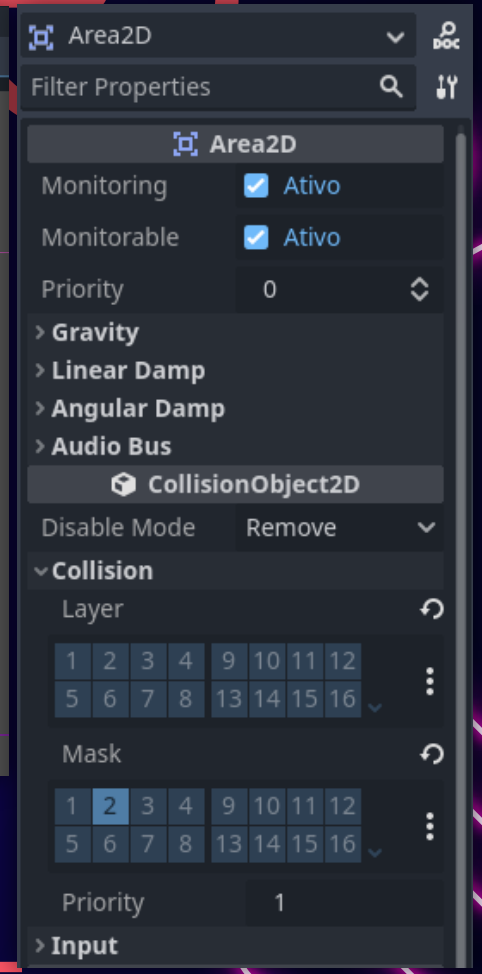
Area2D



Floor & ceiling



Mask



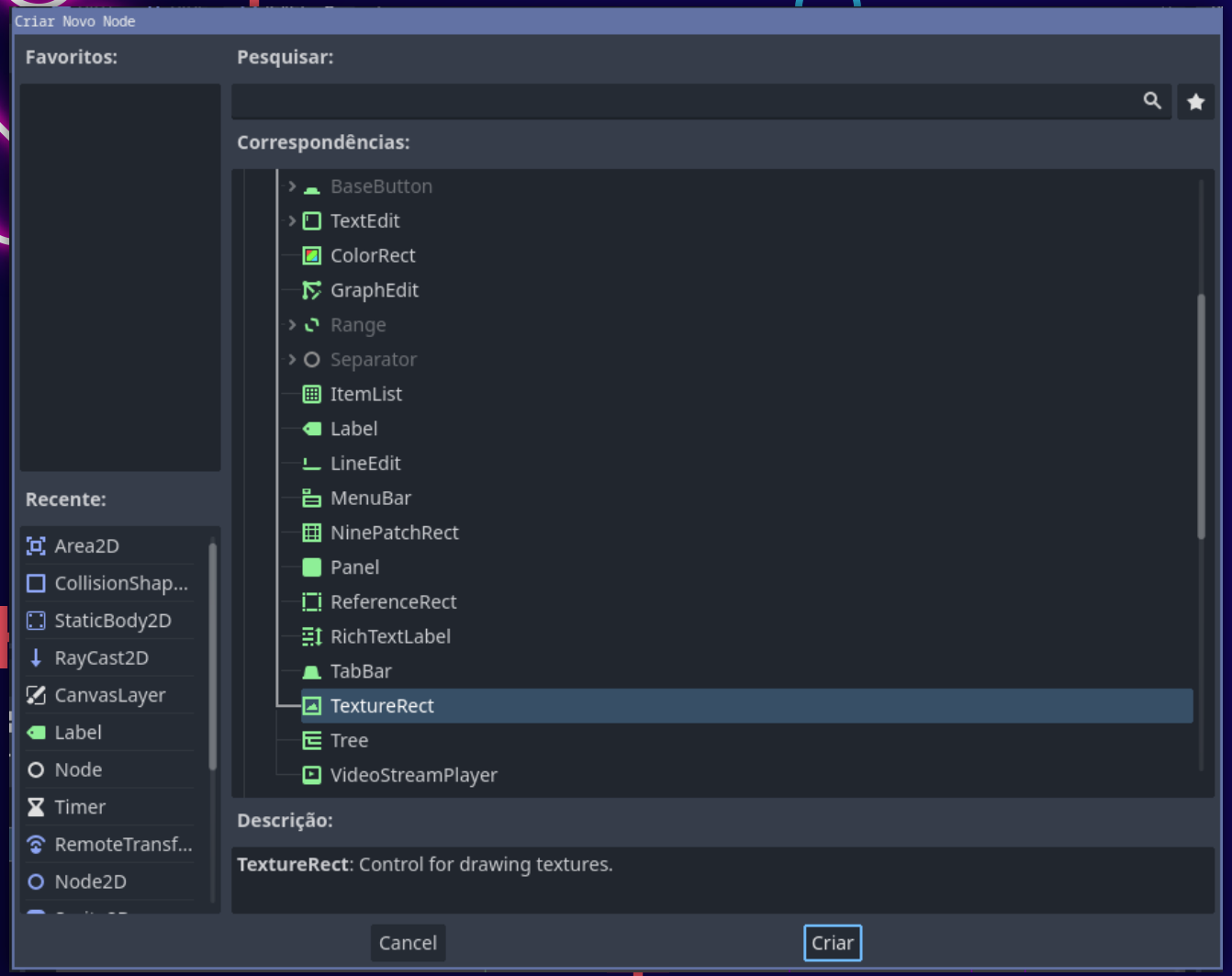
Function calls

```gdscript
@export var floor_area: Area2D

func spawn_pipe():
    var new_pipe: Pipe = PIPE_SCN.instantiate()
    var floor_virtual_posx := floor_sprite.get_meta("virtual_posx") as float
    new_pipe.position.x = SPAWN_POS + \
    new_pipe.position.y = roundf(randf_range(min_position, max_position))
    new_pipe.gap = roundf(current_gap)
    # Update Gap
    current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)
    # Conecta-se ao evento "scored" pra atualizar a UI (Inteface do Usuário)
    new_pipe.scored.connect(func update_gui():
        score += 1
        score_label.text = str(score)
    )
    # Conecta-se ao evento "body entered" para resetar o Game Loop
    new_pipe.body_entered.connect(func(_arg):
        bird.apply_impulse(Vector2.LEFT * .2, Vector2.LEFT)
        restart()
    )
    floor_area.body_entered.connect(func(_arg):
        bird.apply_impulse(Vector2.UP * .2, Vector2(randf_range(-1, 1), -1))
        bird.set_process_input(false)
        floor_area.queue_free()
        restart()
    )
    add_child(new_pipe)
    timer.start(BASE_SPAWN_FREQ * FREQ_DEC_DEG / (Game.speed + FREQ_DEC_DEG))

```

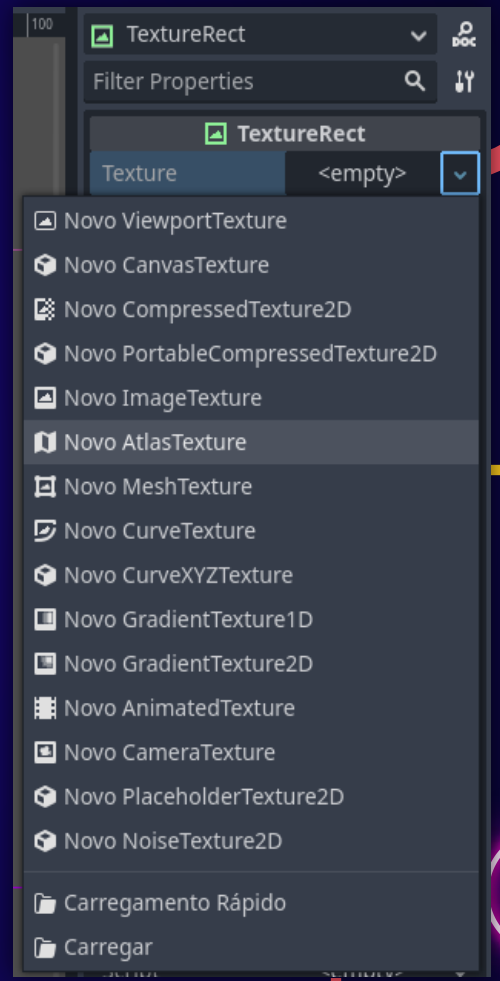
Restart function

```gdscript
func restart():
    bird.set_process_input(false)
    await get_tree().physics_frame # Aguarda a física ser processada

    for node in freeze_nodes:
        node.process_mode = Node.PROCESS_MODE_DISABLED
        # Para de processar o nó como se o jogo estivesse pausado para ele

```

TextureRect

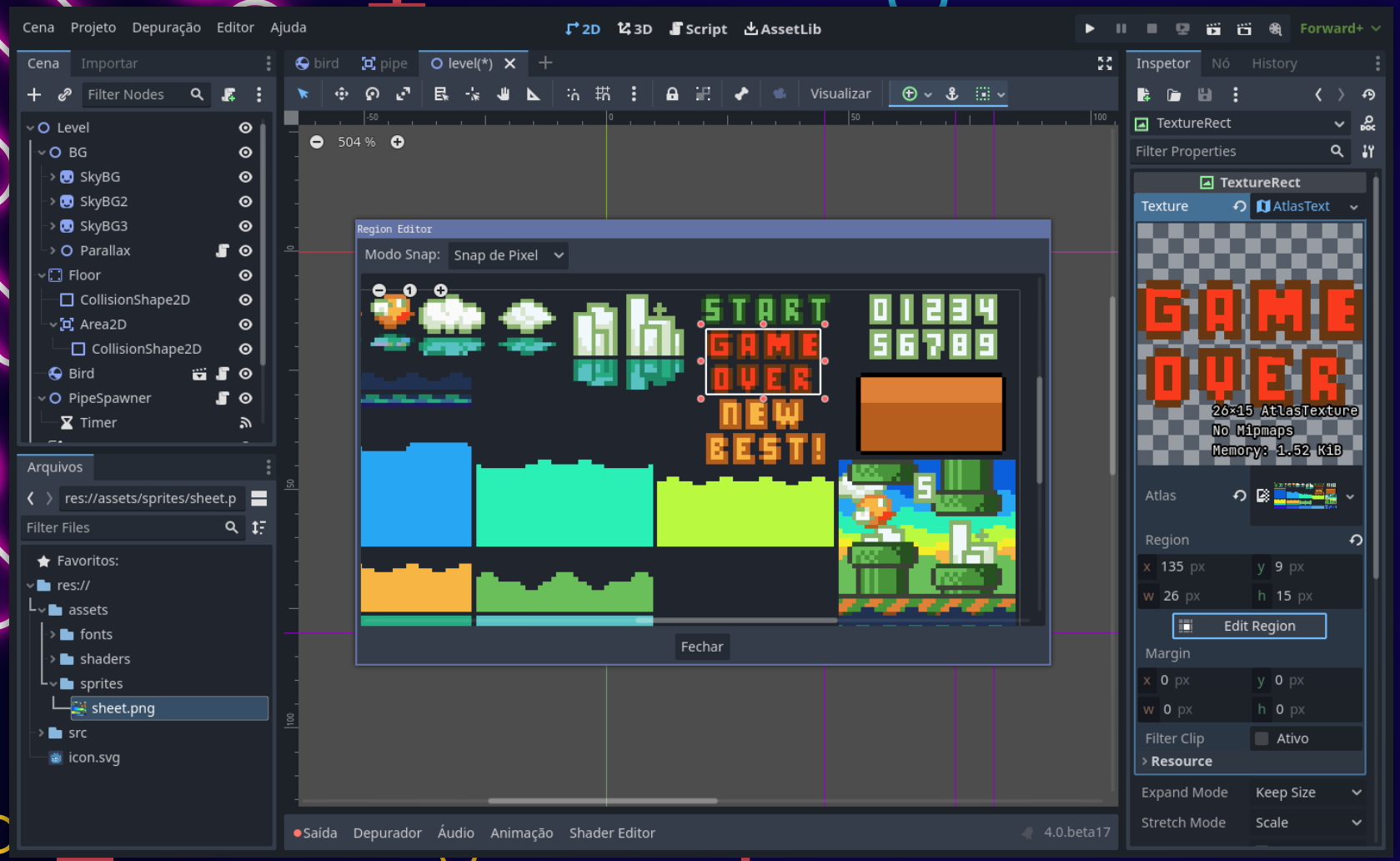


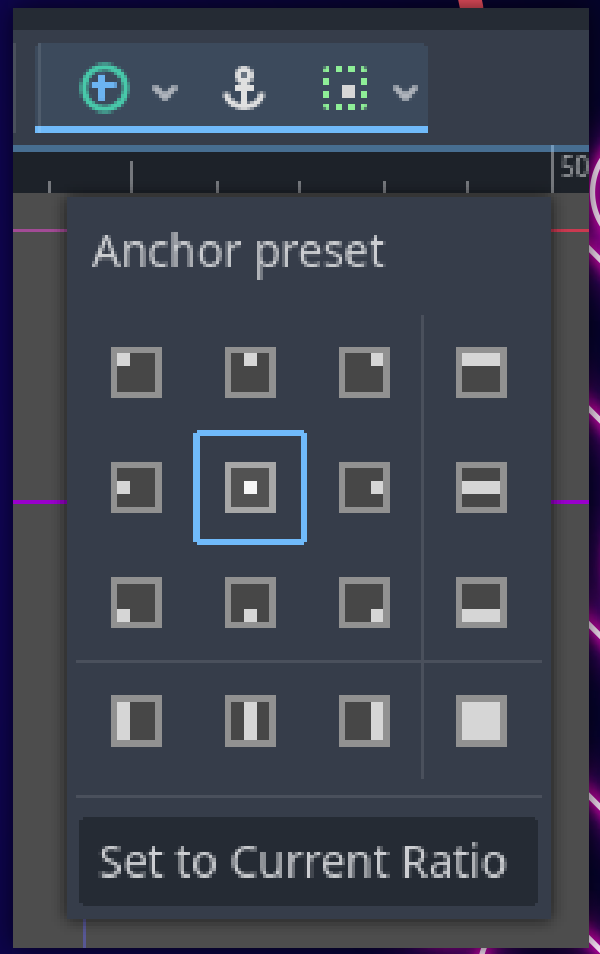
AtlasTexture



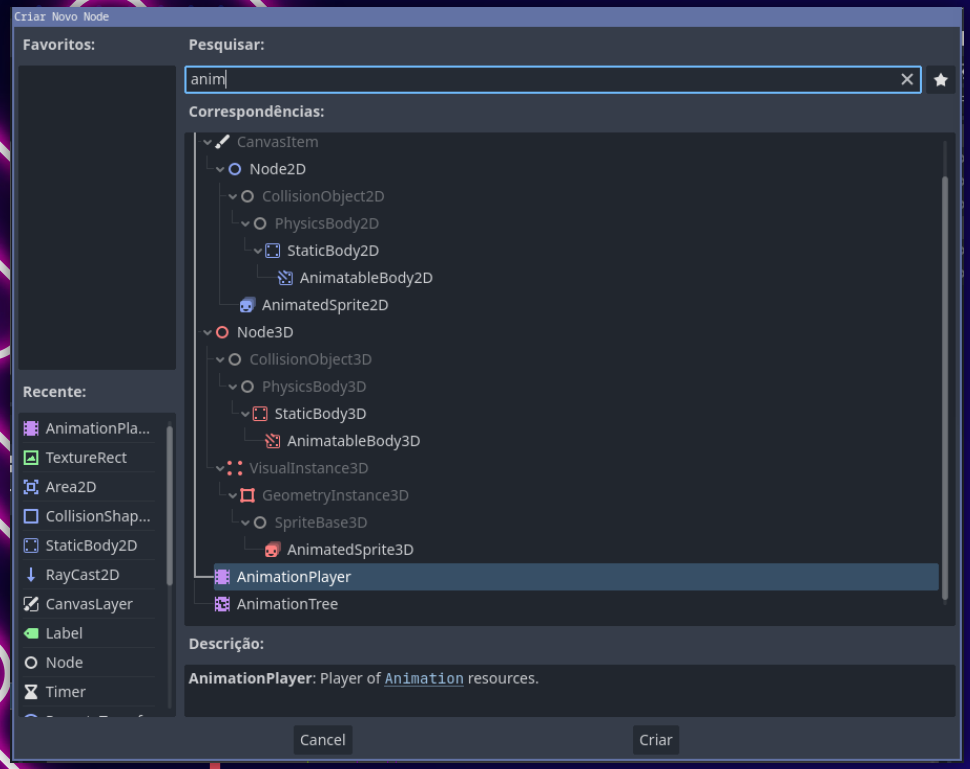
Edit Region

Center

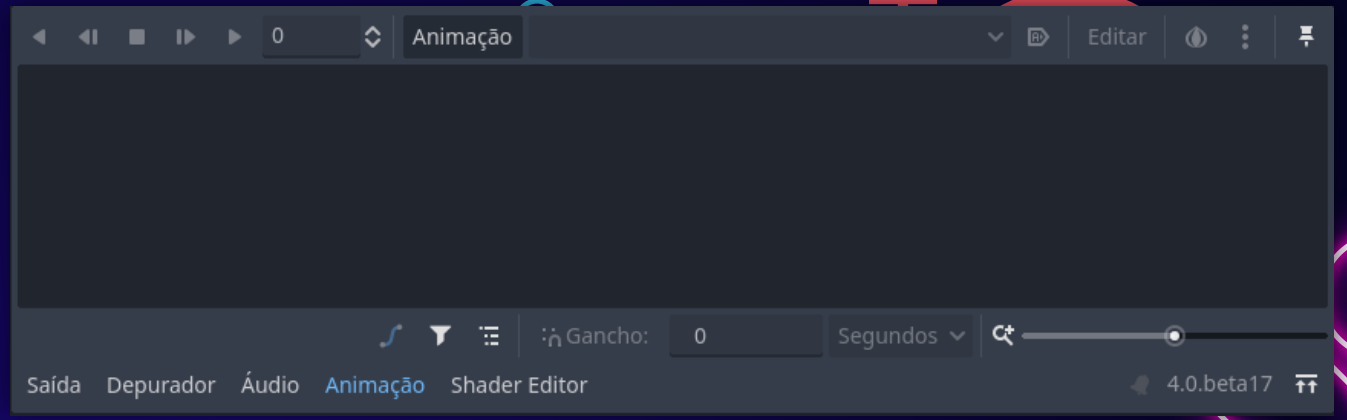




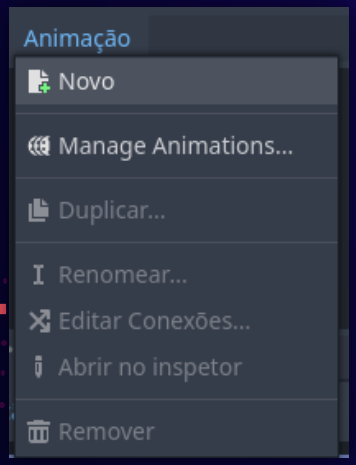
AnimationPlayer



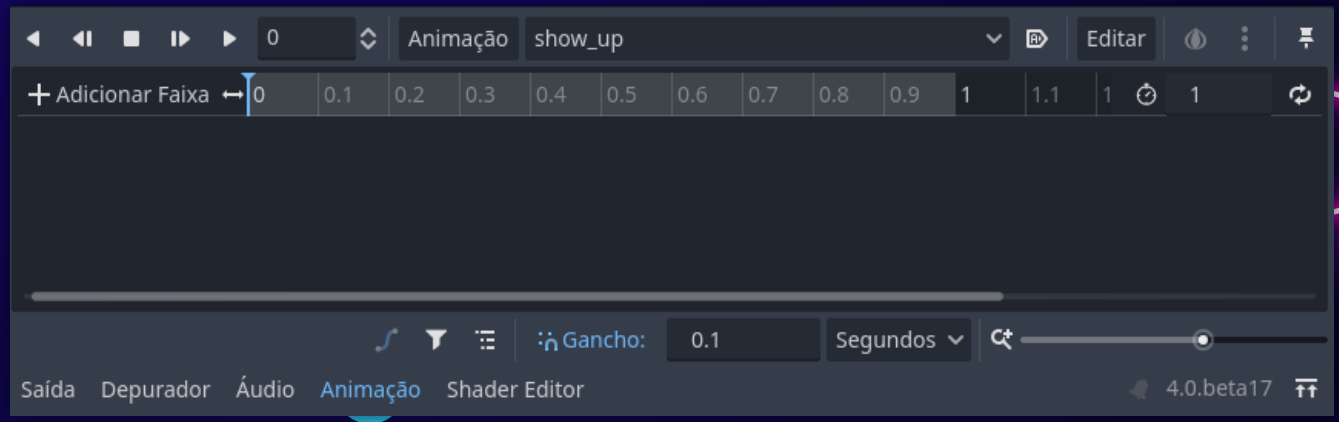
Animation dock



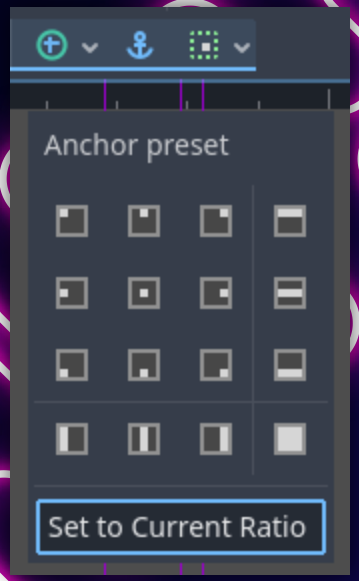
Nova animação



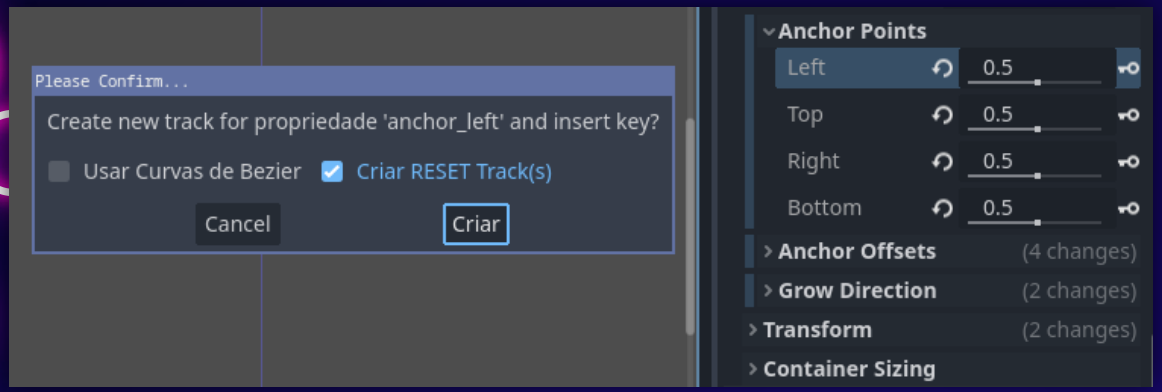
Animation track



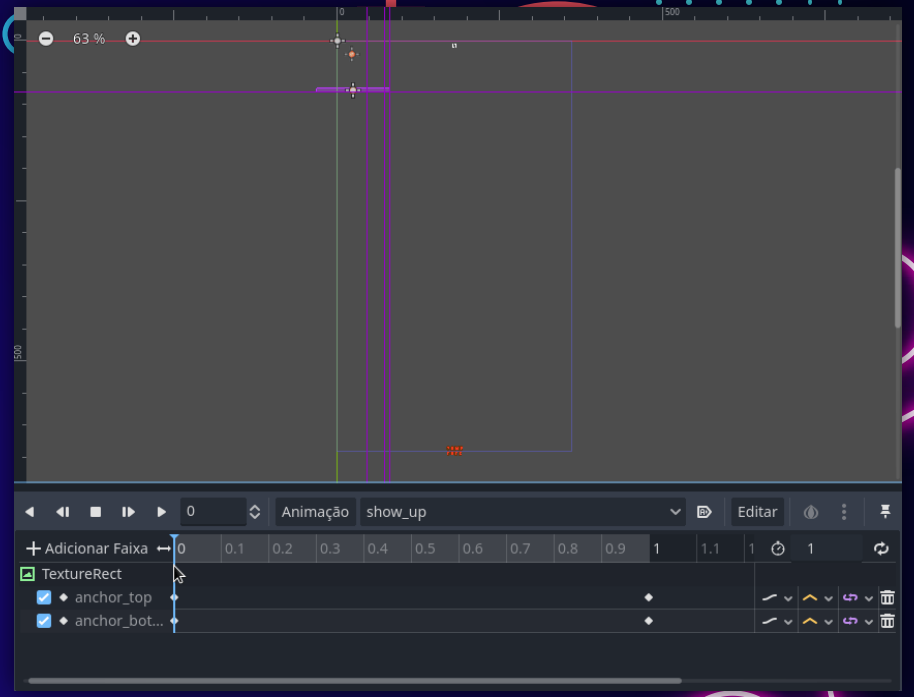
Anchors



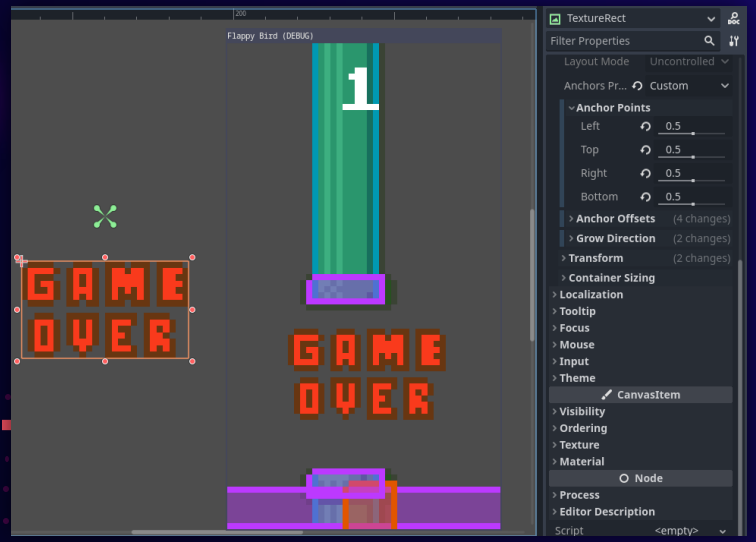
Keyframe



"Show up" animation



Offset

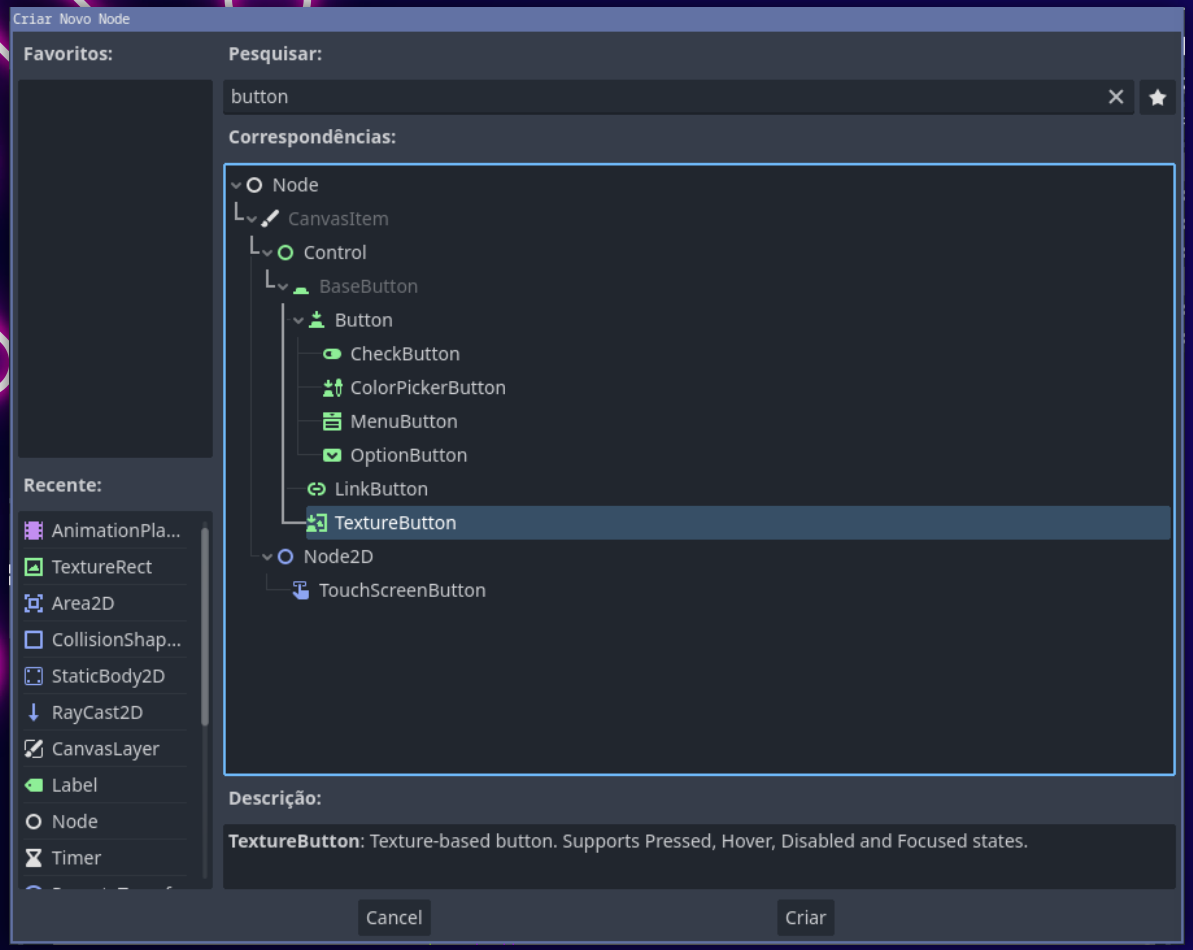


Play animation

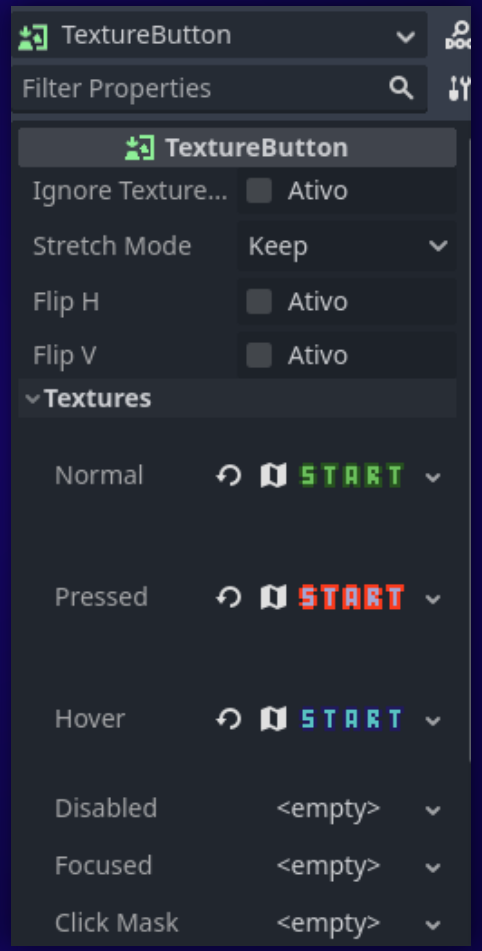
```
74  @export var anim_player: AnimationPlayer
75
76  func restart():
77      bird.set_process_input(false)
78      bird.jump_auto = func(_arg): pass
79
80      await get_tree().physics_frame # Aguarda a física ser processada
81
82      for node in freeze_nodes:
83          node.process_mode = Node.PROCESS_MODE_DISABLED
84          # Para de processar o nó como se o jogo estivesse pausado para ele
85
86      anim_player.play("show_up")
87
```

Gameover

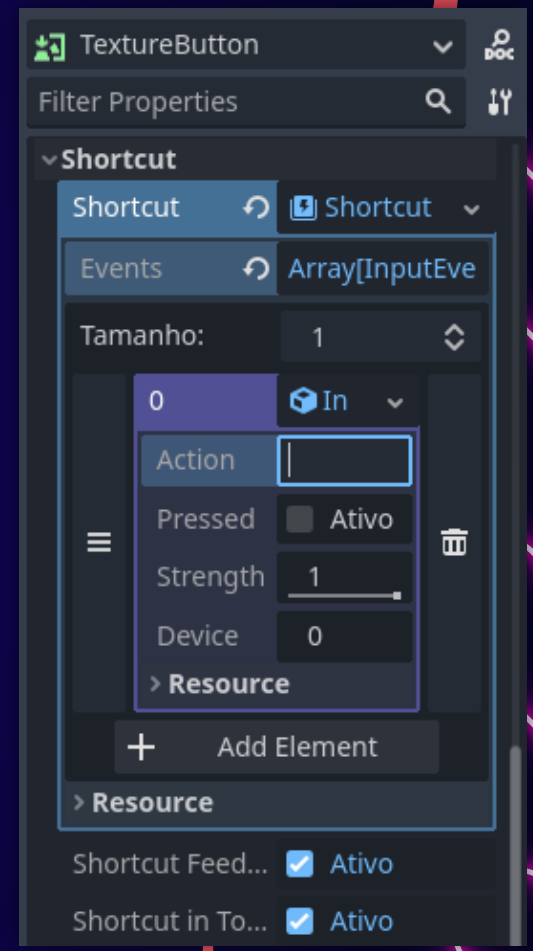
TextureButton



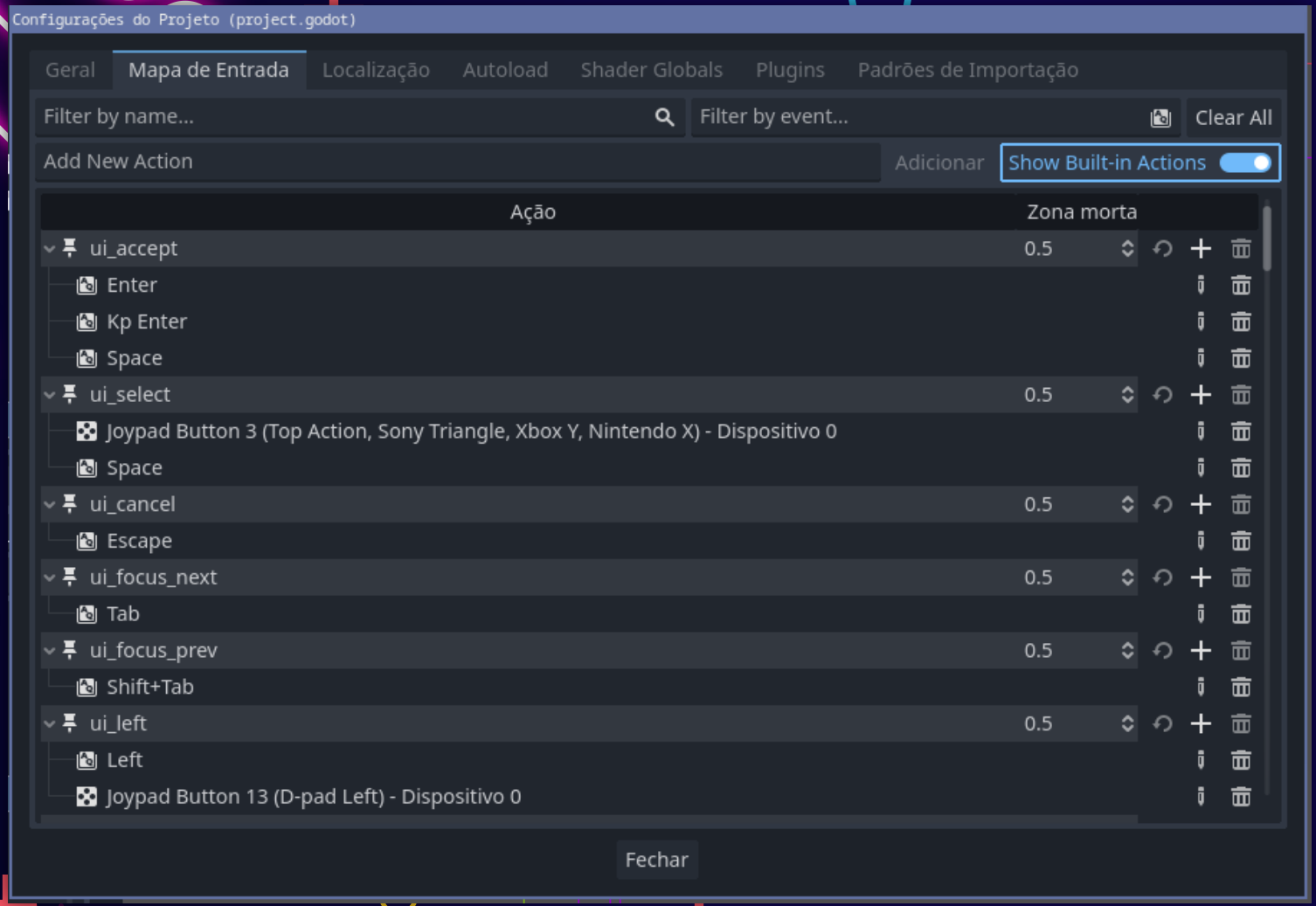
Texture



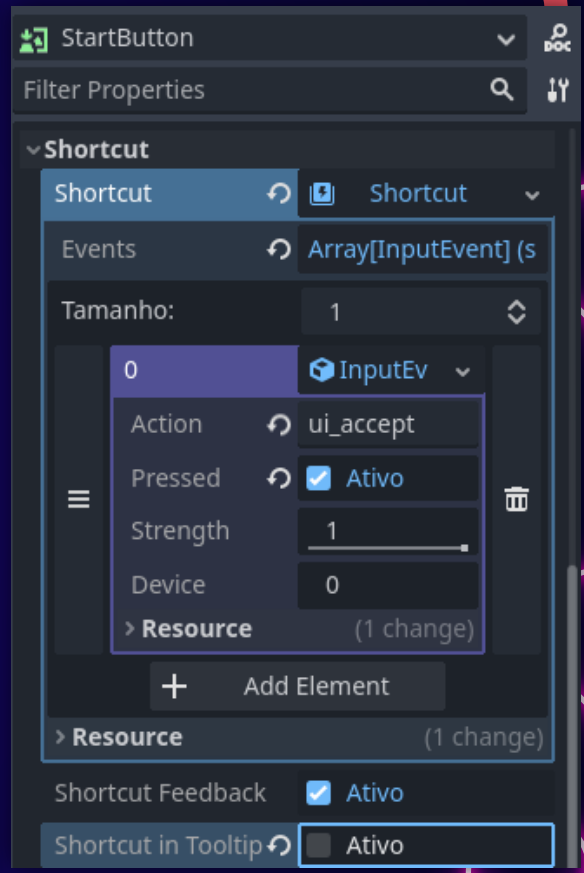
Shortcut



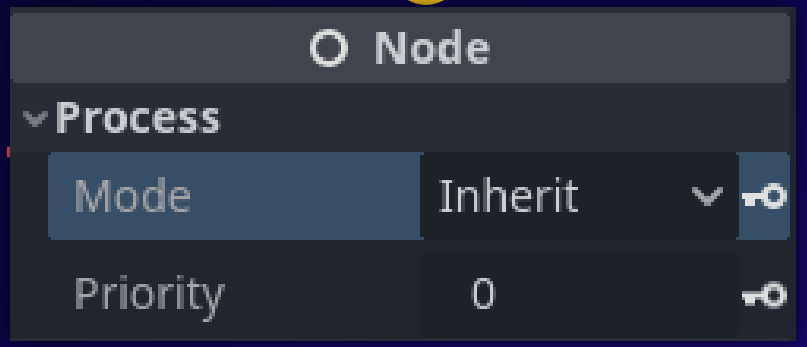
Input Actions



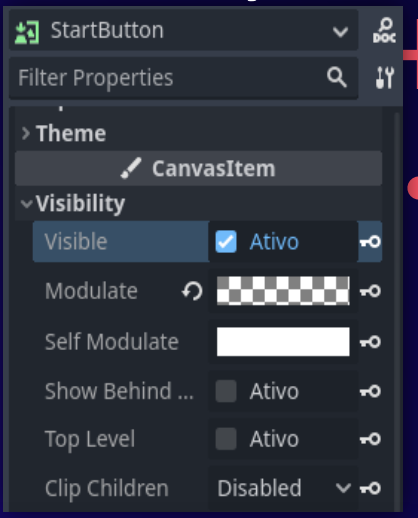
ui_accept



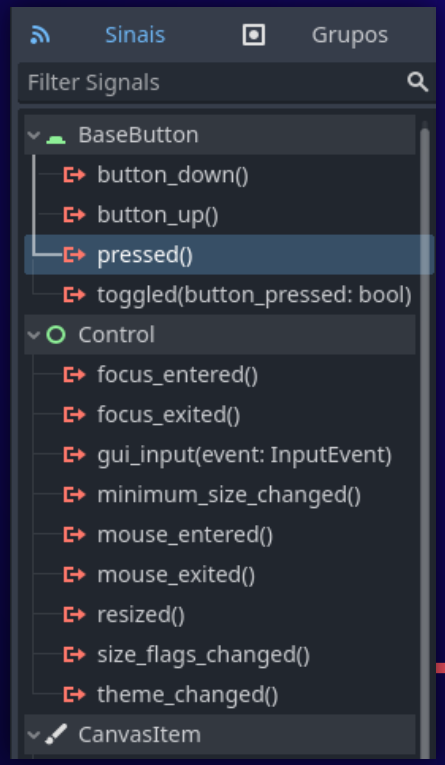
Process mode



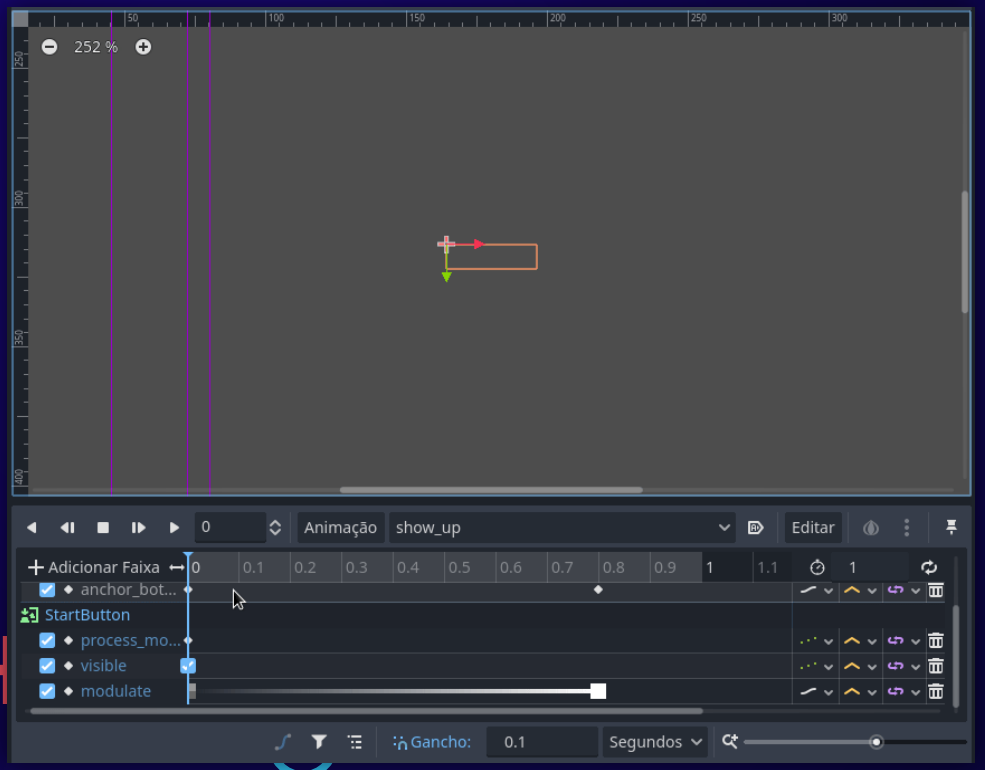
Visibility



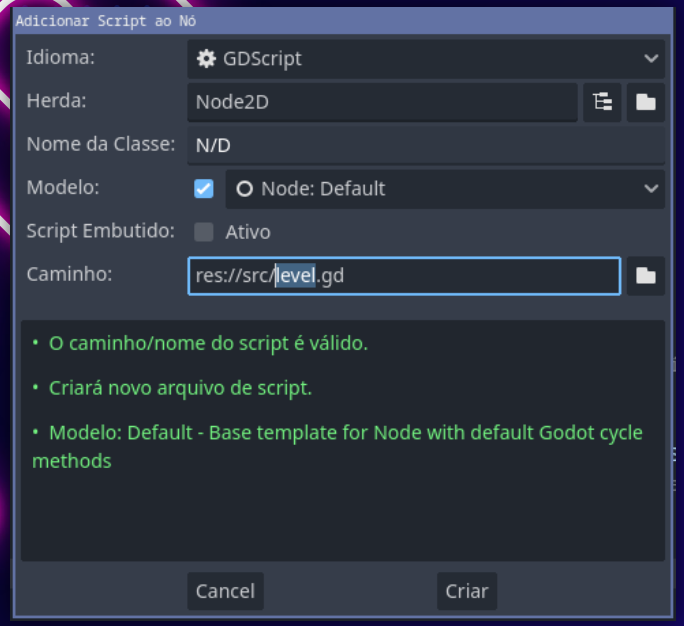
Signal



Animation



Level script



Game reset

```gdscript
extends Node

@export var speed_growth_factor: float = 1.
## Taxa de velocidade (rapidez)
@export var default_speed: float = 1.
@onready var speed: float = default_speed


func _process(delta):


## Reseta os parâmetros do jogo
func reset():
    speed = default_speed

```

New game

Best score

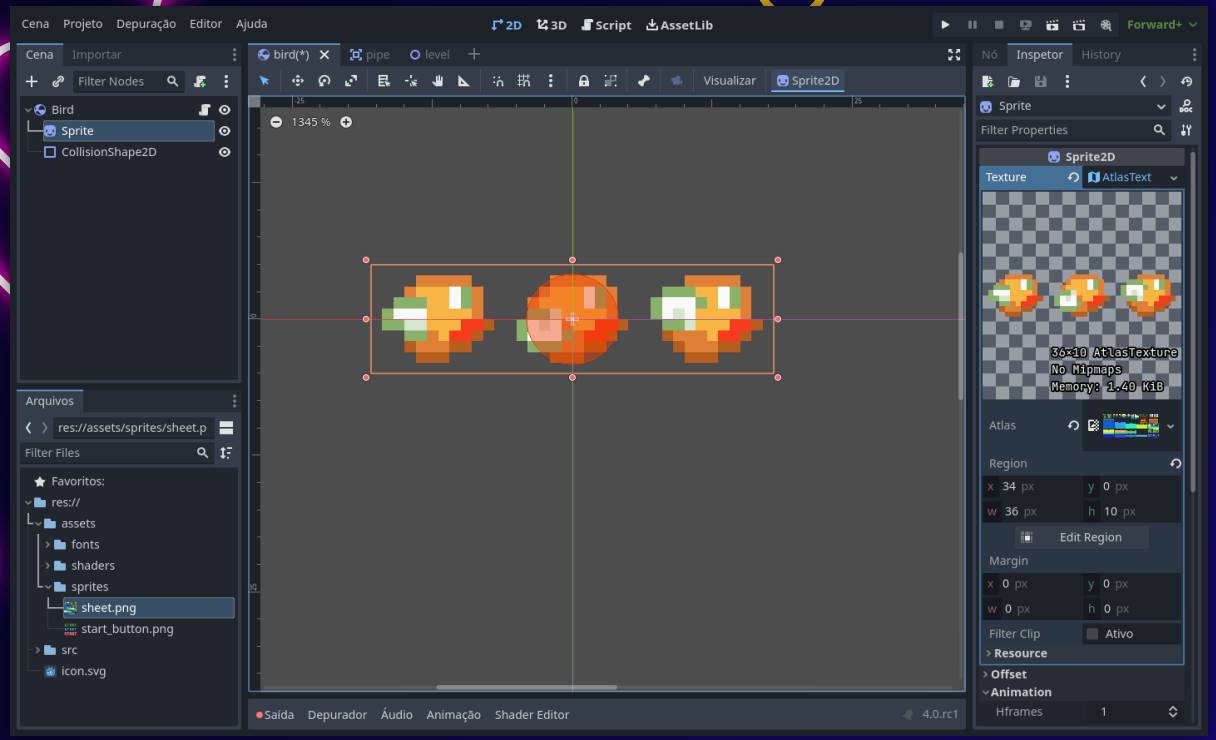
```gdscript
signal new_best_achieved
var best_score: int = 0
var score: int = 0:
    set(value):
        score = value
        if score > best_score:
            best_score = score
            new_best_achieved.emit()


## Reseta os parâmetros do jogo
func reset():
    speed = default_speed
    score = 0

```

Final result 1

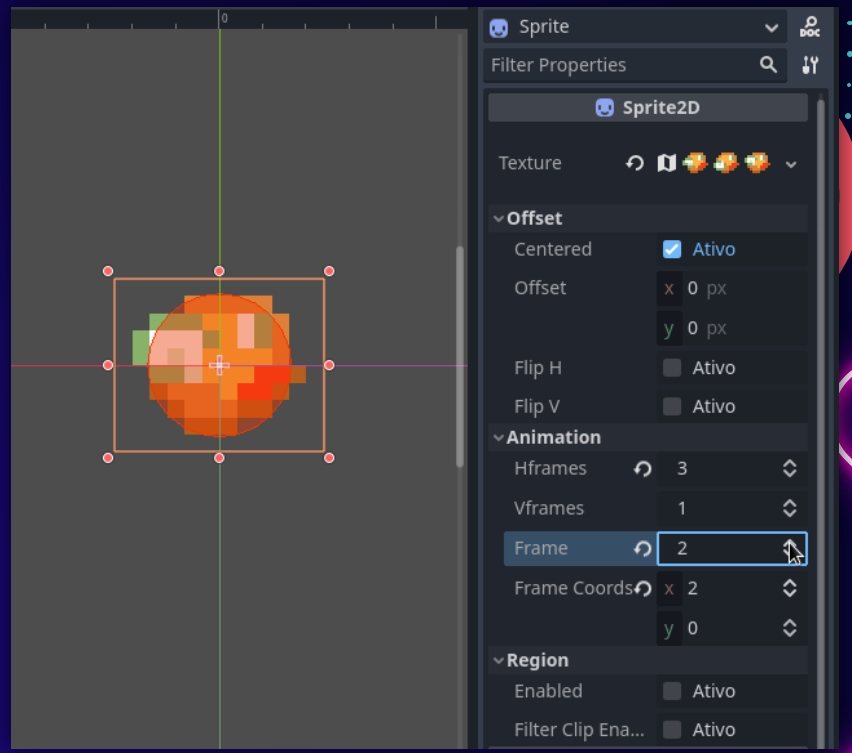
Final result 2

Máquina de estado finito & áudio

AtlasTexture



Animation frames

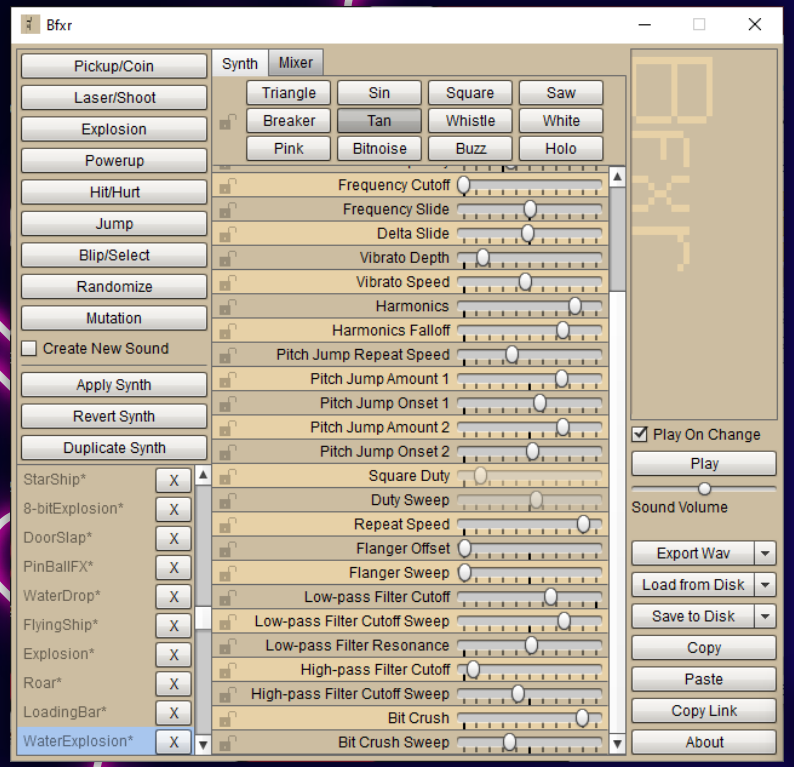


State machine

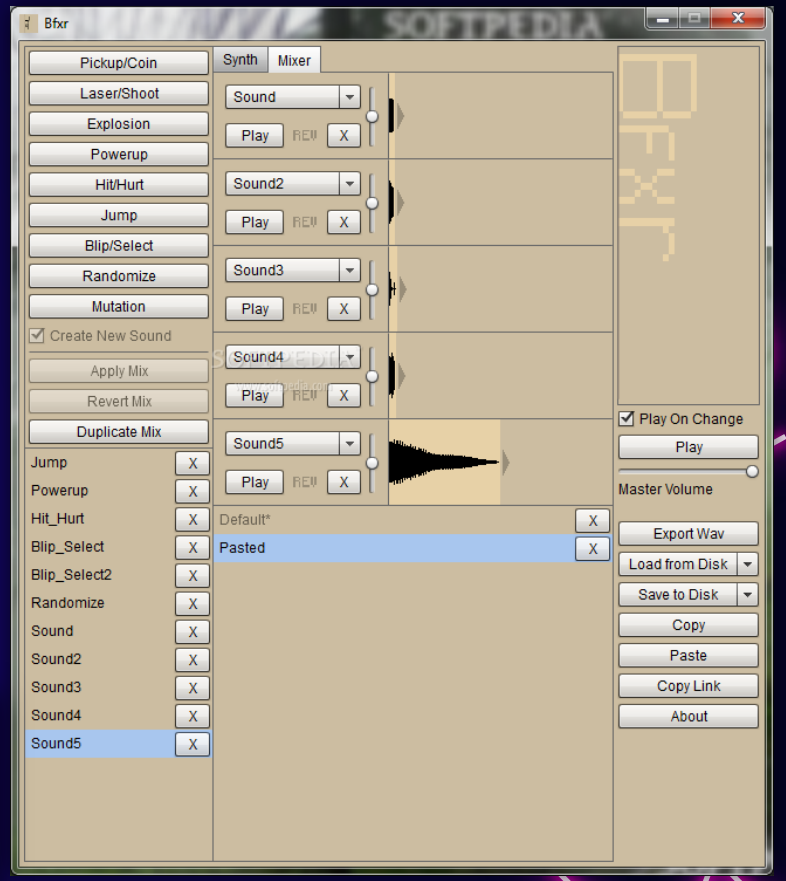
```gdscript
const ANIM_THRESHOLD: float = 3.
enum States { IDLE = 0, FALL, JUMP }
@onready var sprite := $Sprite as Sprite2D

func _physics_process(delta):
    jump_auto.call(delta)

    if linear_velocity.y < -ANIM_THRESHOLD:
        sprite.frame = int(States.FALL)
    elif linear_velocity.y > ANIM_THRESHOLD:
        sprite.frame = int(States.JUMP)
    else:
        sprite.frame = int(States.IDLE)
```

State machine

BFXR

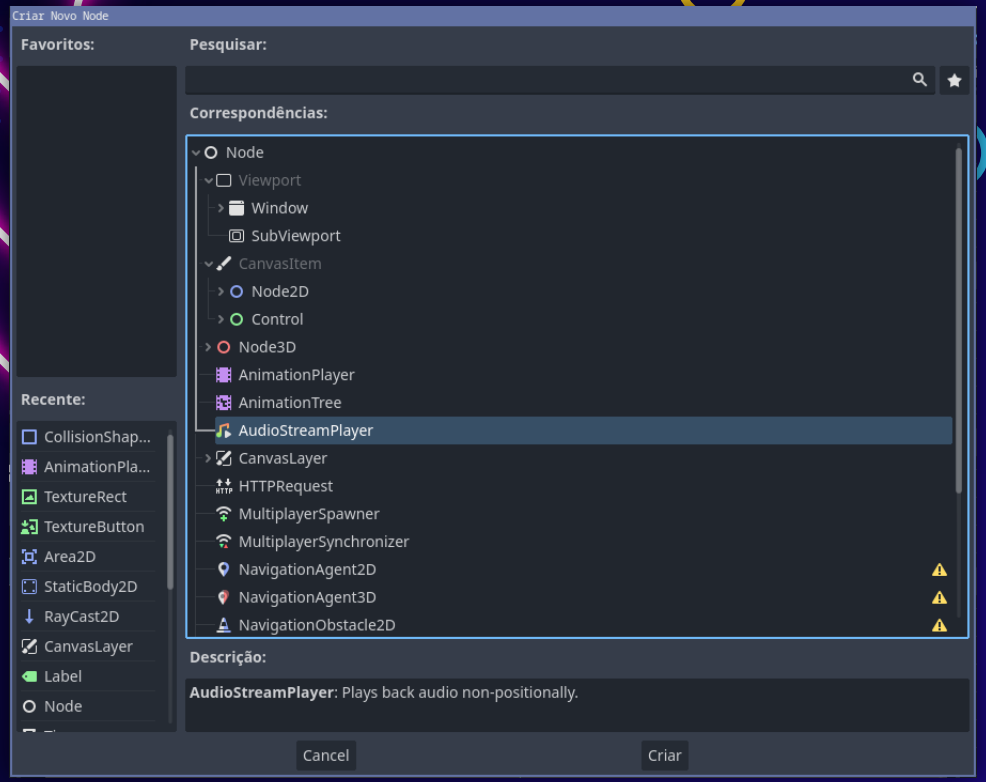


BFXR mixer



Chiptone

AudioStreamPlayer



Add audio

Play sfx

```gdscript
const MIN_PITCH: float = .75
const MAX_PITCH: float = 1.2

@export var jump_force: float = 10.
@onready var sfx_player := $SFX as AudioStreamPlayer

func _input(event: InputEvent):

    if event.is_action_pressed("action"): # Jump
        apply_force(Vector2.UP * jump_force, Vector2.UP)
        sfx_player.pitch_scale = randf_range(MIN_PITCH, MAX_PITCH)
        sfx_player.play()

```

Jump sfx

Collide sfxs

```
43  func spawn_pipe():
44      var new_pipe: Pipe = PIPE_SCN.instantiate()
45      var floor_virtual_posx := floor_sprite.get_meta("virtual_posx") as float
46
47      new_pipe.position.x = SPAWN_POS + \
49      new_pipe.position.y = roundf(randf_range(min_position, max_position))
50      new_pipe.gap = roundf(current_gap)
51
52      # Update Gap
53      current_gap = maxf(lerpf(current_gap, low_gap, gap_reduction), low_gap)
54
55      # Conecta-se ao evento "scored" pra atualizar a UI (Inteface do Usuário)
56      new_pipe.scored.connect(func update_gui():
59      )
60      # Conecta-se ao evento "body entered" para resetar o Game Loop
61      new_pipe.body_entered.connect(func(_arg):
62          bird.apply_impulse(Vector2.LEFT * .2, Vector2.LEFT)
63          play_sfx(pipe_collide_sfx)
64          bird_died.emit()
65      )
66      floor_area.body_entered.connect(func(_arg):
67          bird.apply_impulse(Vector2.UP * .2, Vector2(randf_range(-1, 1), -1))
68          bird.set_process_input(false)
69          floor_area.queue_free()
70          play_sfx(floor_collide_sfx)
71          bird_died.emit()
72      )
73
74      add_child(new_pipe)
75      timer.start(BASE_SPAWN_FREQ * FREQ_DEC_DEG / (Game.speed + FREQ_DEC_DEG))
76
```

Play stream

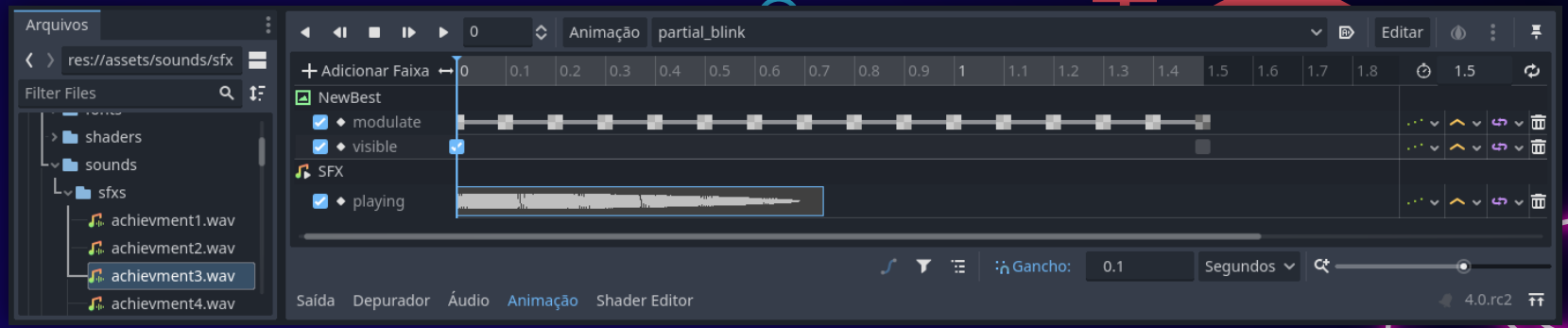
```
78  const MIN_PITCH: float = .75
79  const MAX_PITCH: float = 1.2
80
81  @onready var sfx_player := $SFX as AudioStreamPlayer
82
83  func play_sfx(stream: AudioStreamWAV):
84      sfx_player.stream = stream
85      sfx_player.pitch_scale = randf_range(MIN_PITCH, MAX_PITCH)
86      sfx_player.play()
87
```

Collide floor sfx

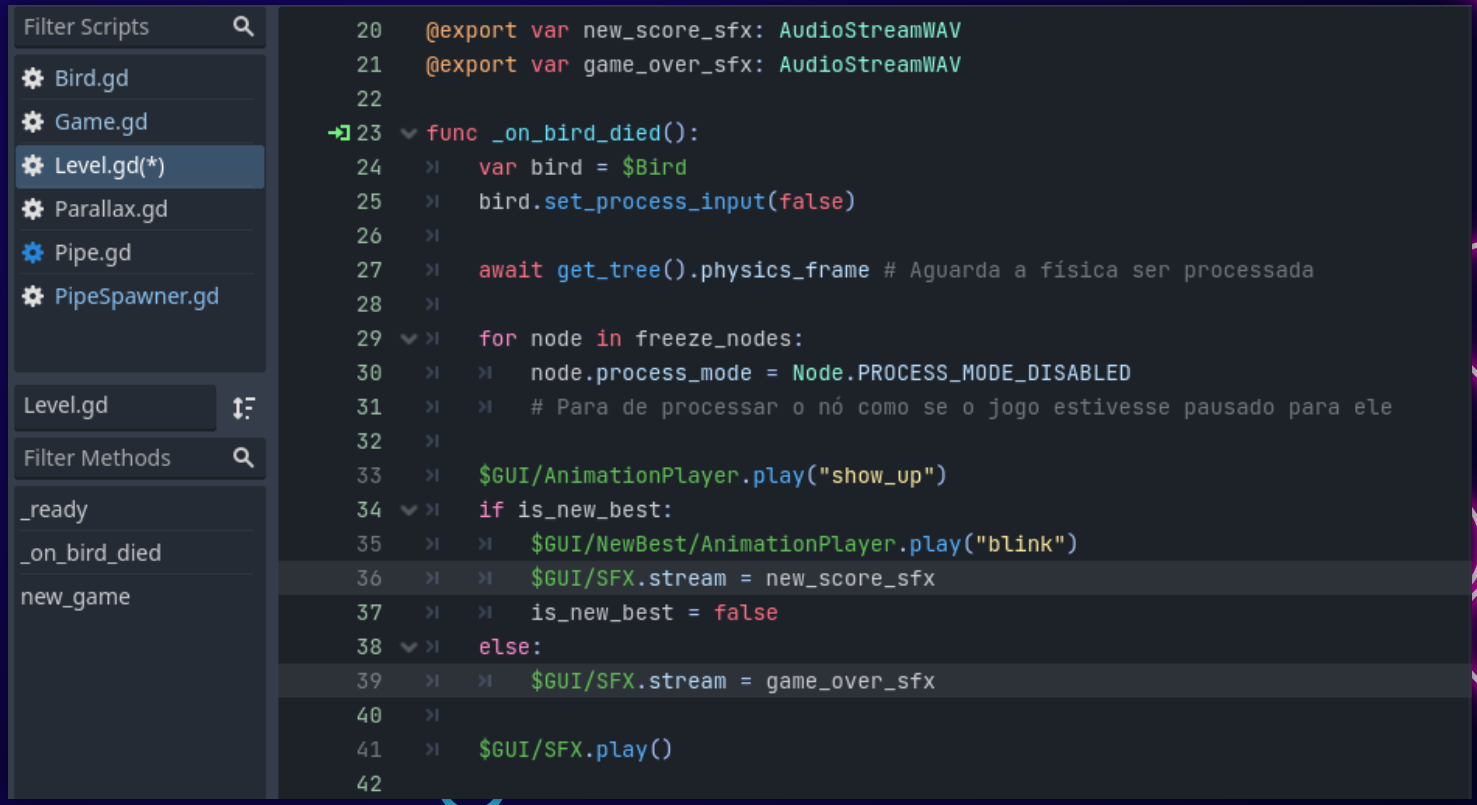
Collide pipe sfx

Add sfx to animation

Add achievment sfx

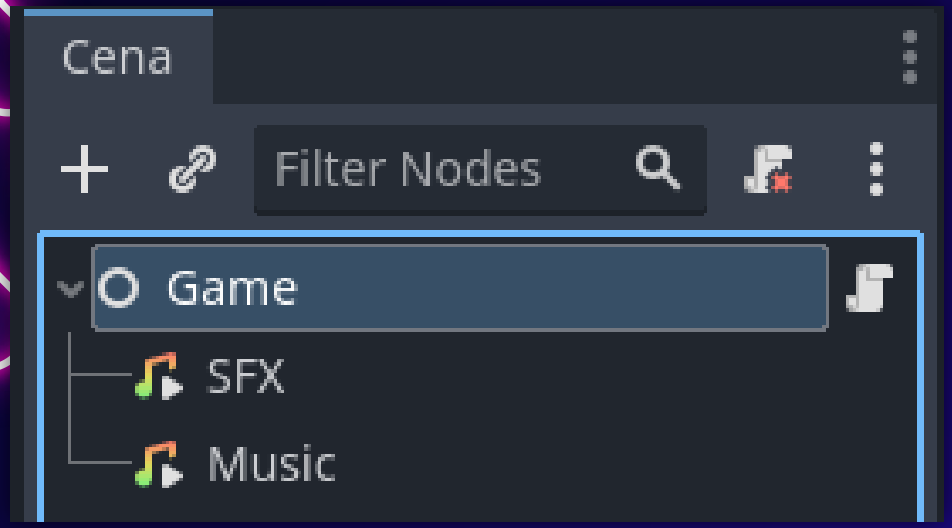


Play new score sfx

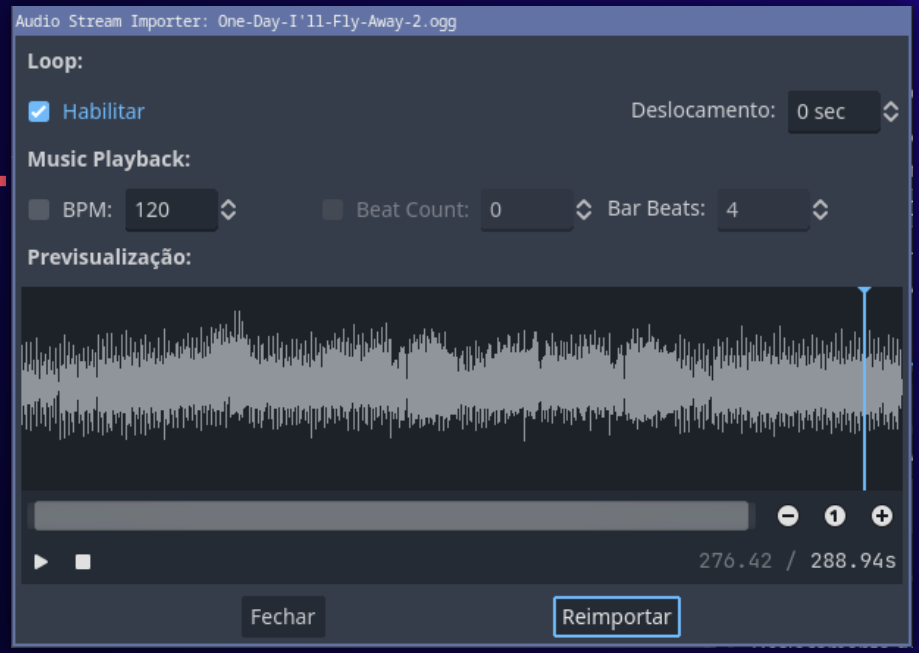


All sfx's

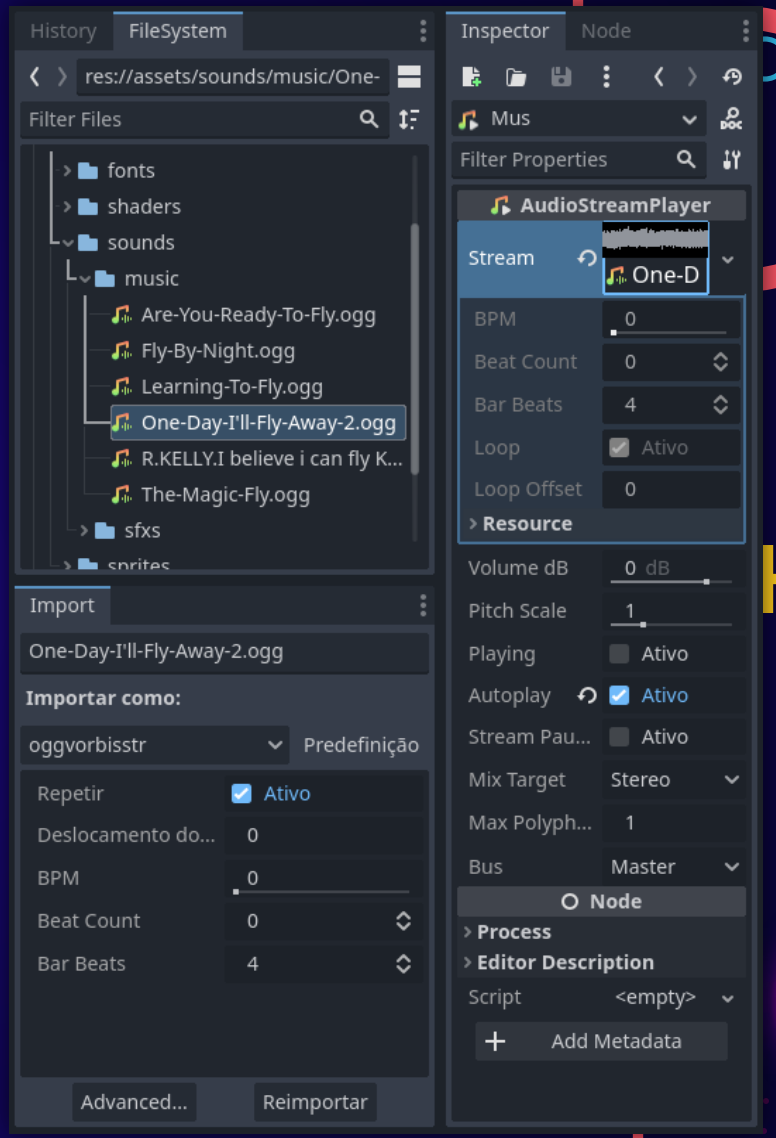
Game scene



Music loop



Music stream



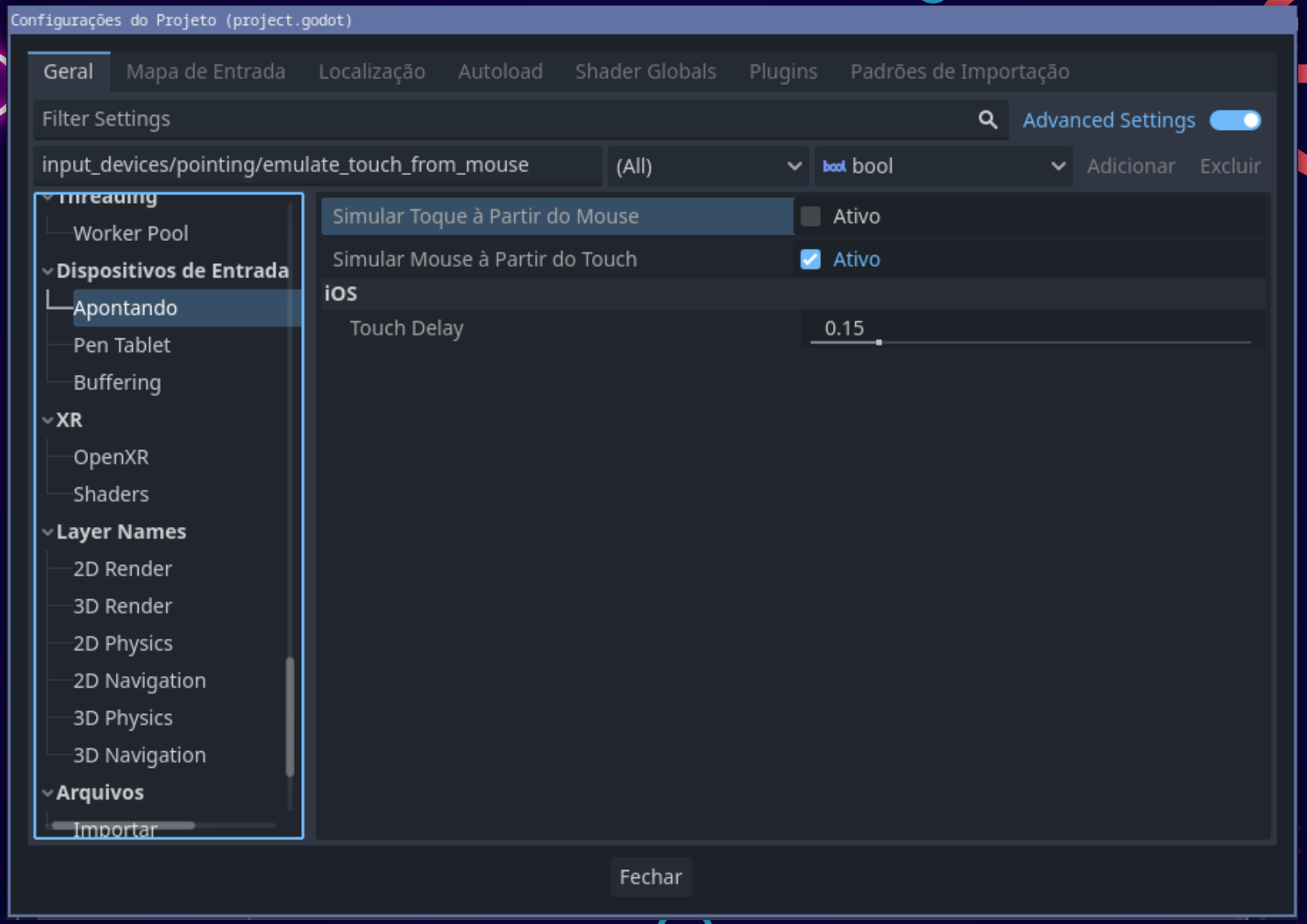
Play score sfx

```
14  signal new_best_achieved
15
16  const MIN_PITCH: float = 1.
17  const MAX_PITCH: float = 1.75
18
19  var best_score: int = 0
20  var score: int = 0
21  @onready var score_sfx := $SFX as AudioStreamPlayer
22
23  func score_up():
24      score += 1
25      score_sfx.pitch_scale = randf_range(MIN_PITCH, MAX_PITCH)
26      score_sfx.play()
27
28      if score > best_score:
29          best_score = score
30          new_best_achieved.emit()
31
```

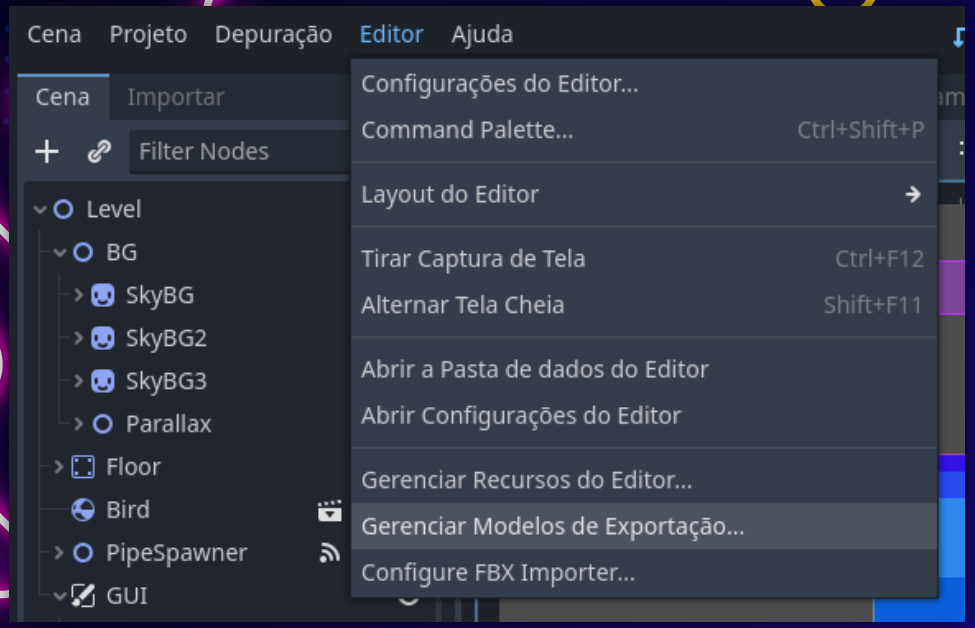
Gameplay

Exportando

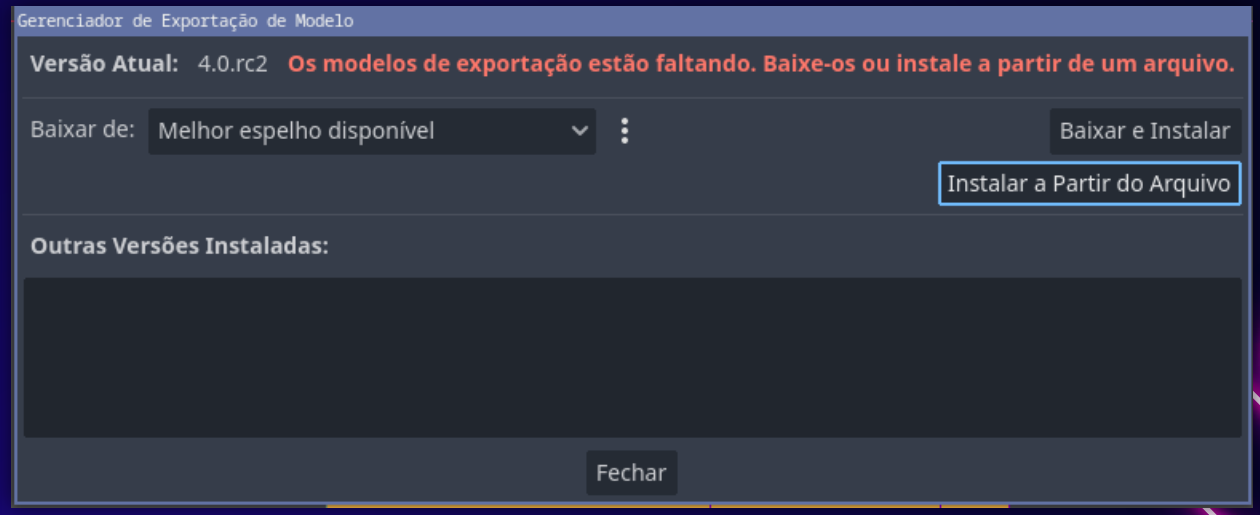
Simular touch



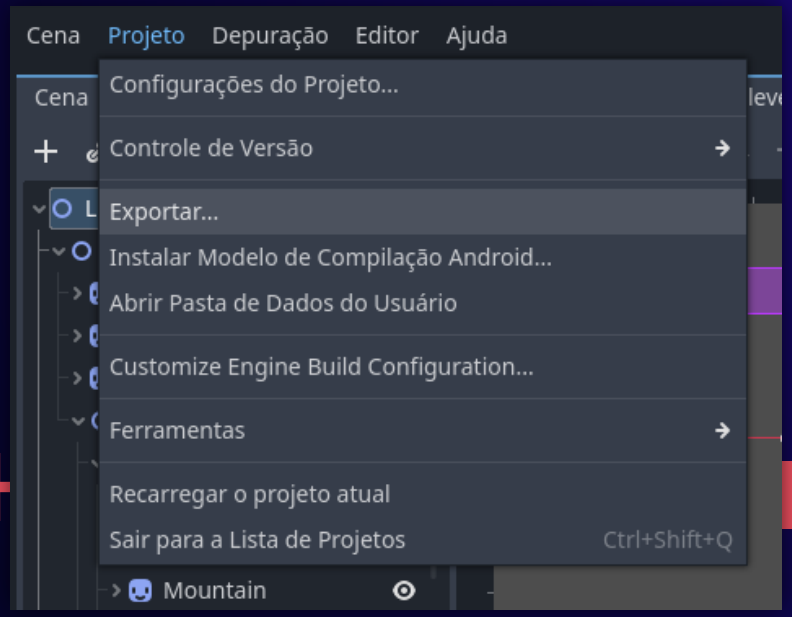
Modelos de exportação



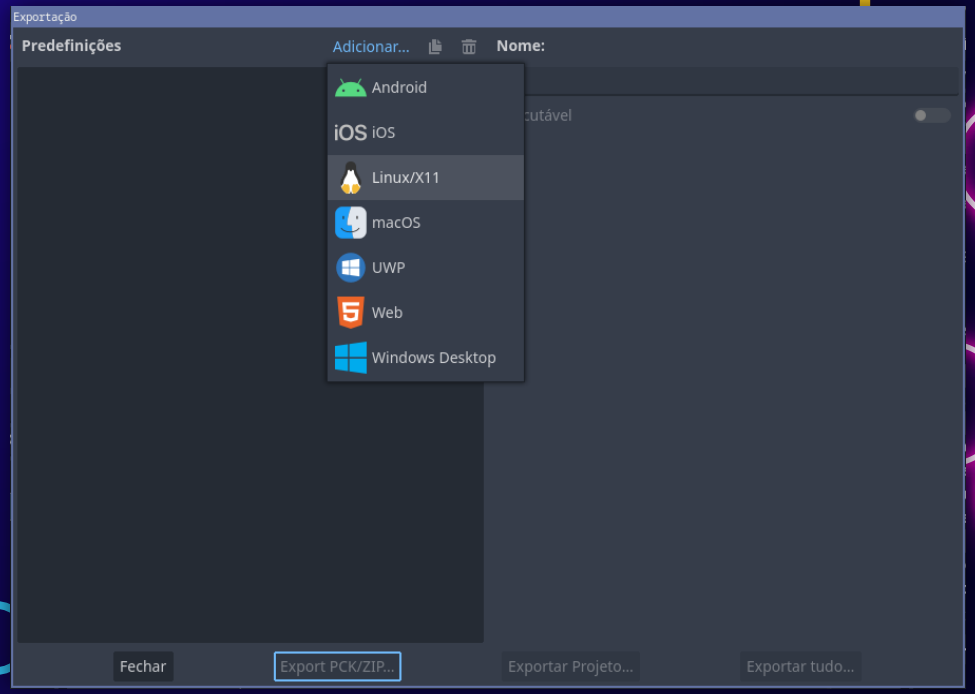
Gerenciador



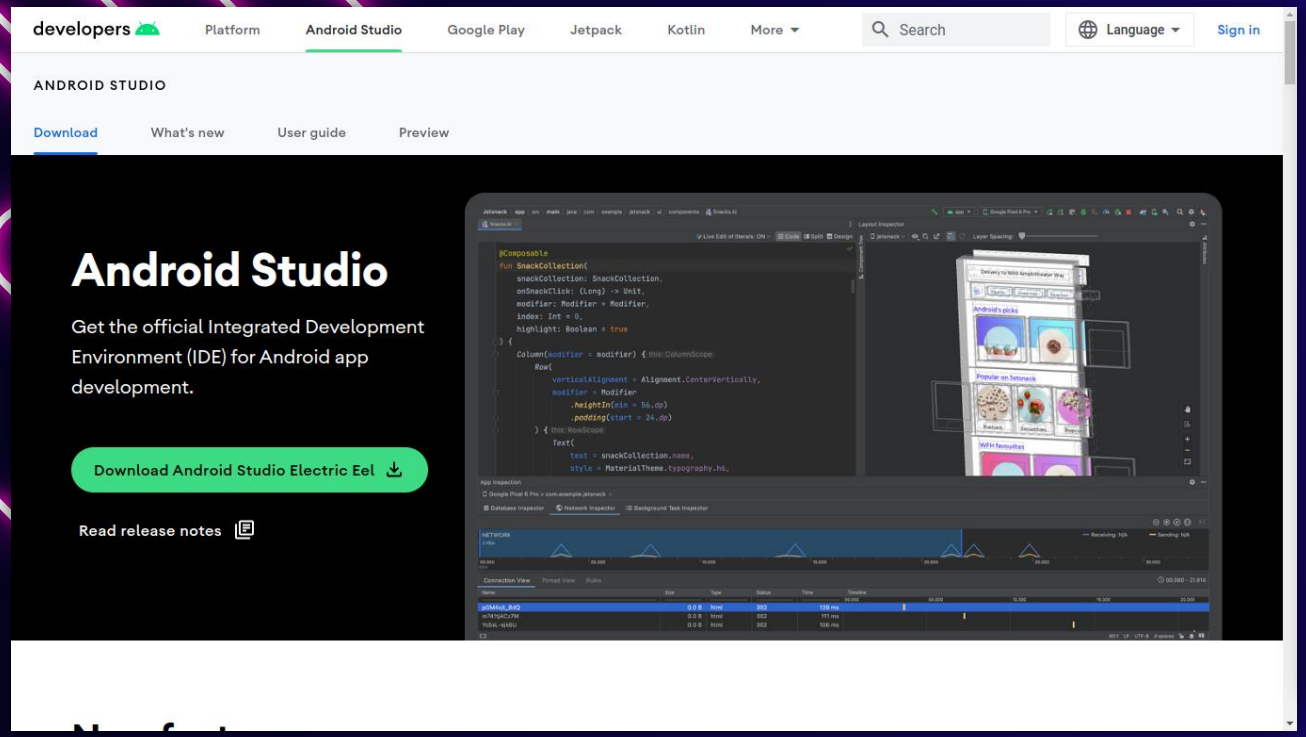
Exportar



Plataformas



Android SDK



Versão do Java

```
→  ~ java --version
openjdk 19.0.2 2023-01-17
OpenJDK Runtime Environment (build 19.0.2+7)
OpenJDK 64-Bit Server VM (build 19.0.2+7, mixed mode)
→  ~                                                    [1s520]
```

Exporting for Android

See also

This page describes how to export a Godot project to Android. If you're looking to compile export template binaries from source instead, read Compiling for Android.

Exporting for Android has fewer requirements than compiling Godot for Android. The following steps detail what is needed to set up the Android SDK and the engine.

Install OpenJDK 11

Download and install OpenJDK 11.

Download the Android SDK

Download and install the Android SDK.

- You can install it using Android Studio version 4.1 or later.
  - Run it once to complete the SDK setup using these instructions.
  - Ensure that the required packages are installed as well.
    - Android SDK Platform-Tools version 30.0.5 or later
    - Android SDK Build-Tools version 30.0.3
    - Android SDK Platform 31
    - Android SDK Command-line Tools (latest)
    - CMake version 3.10.2.4988404

Referências

Os links de materiais usados nesta apresentação estão disponíveis em github.com/GersonFeDutra/gd_start

Agradecimentos:

Daniel Nascimento

https://github.com/danielnasc

Miguel Reis de Araújo

https://github.com/reisaraujo-miguel