Использование сигналов

В этом уроке рассмотрим сигналы. Это сообщения, которые выдает узел при определенных событиях, например, при нажатии кнопки. Другие узлы могут получать этот сигнал и вызывать функции, соответствующие событию.

Сигналы — это механизм делегирования, встроенный в Godot, который позволяет одному игровому объекту реагировать на изменение другого, не ссылаясь друг на друга. Использование сигналов ограничивает связанность и делает ваш код гибким.

Например, у вас на экране может быть шкала жизни, которая соответствует здоровью игрока. Когда игрок получает урон или использует исцеляющее зелье, вы хотите, чтобы шкала реагировала на изменения. Чтобы сделать это в Godot, вы бы использовали сигналы.

Примечание

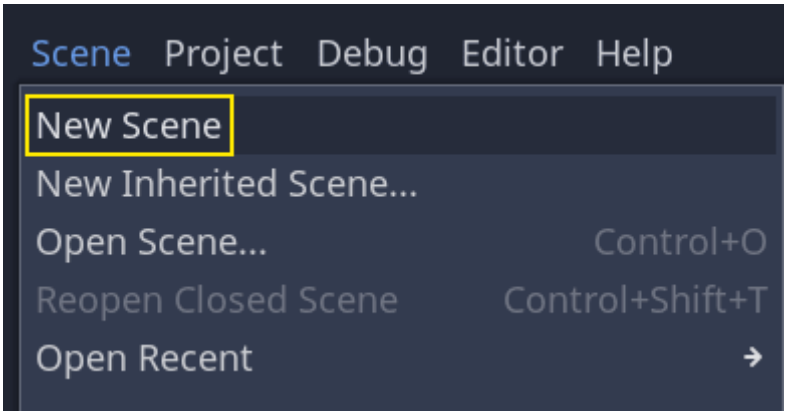
Как уже упоминалось во введении, сигналы - это версия шаблона наблюдателя в Godot. Подробнее об этом можно узнать здесь: https://gameprogrammingpatterns.com/observer.html

Теперь мы будем использовать сигнал, чтобы заставить нашу иконку Годо из предыдущего урока (Отслеживание ввода игрока) двигаться и останавливаться при нажатии кнопки.

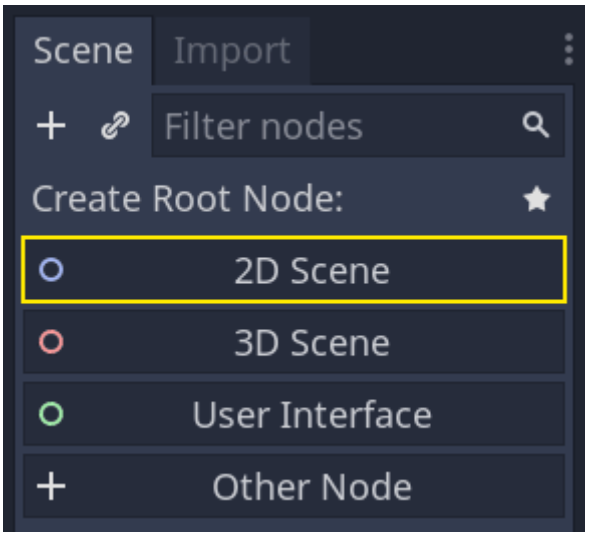
Настройка сцены

Чтобы добавить кнопку в нашу игру, мы создадим новую "главную" сцену, которая будет содержать как кнопку, так и сцену `Sprite.tscn`, которую мы вписали в скрипт в предыдущих уроках.

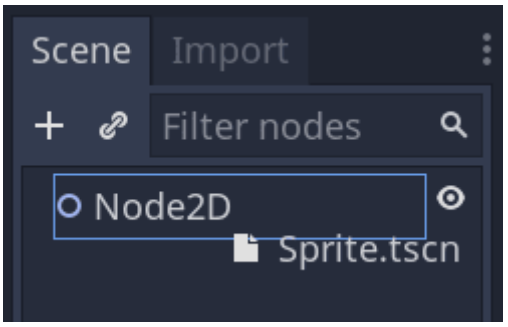
Создайте новую сцену через меню Сцена -> Новая сцена.



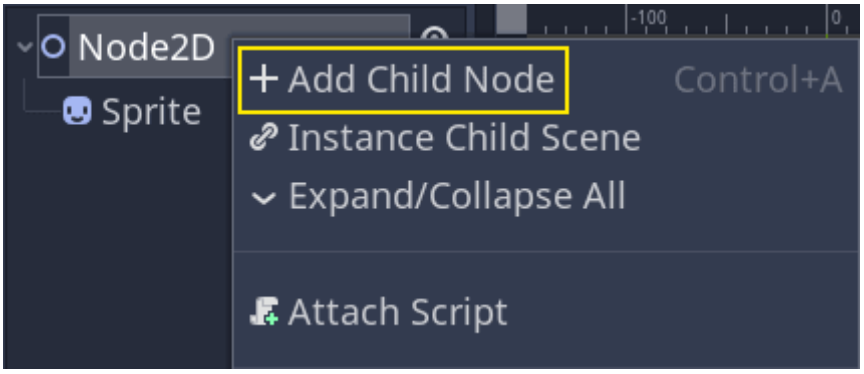
В доке Scene нажмите кнопку 2D Scene. Это добавит Node2D в качестве корня сцены.



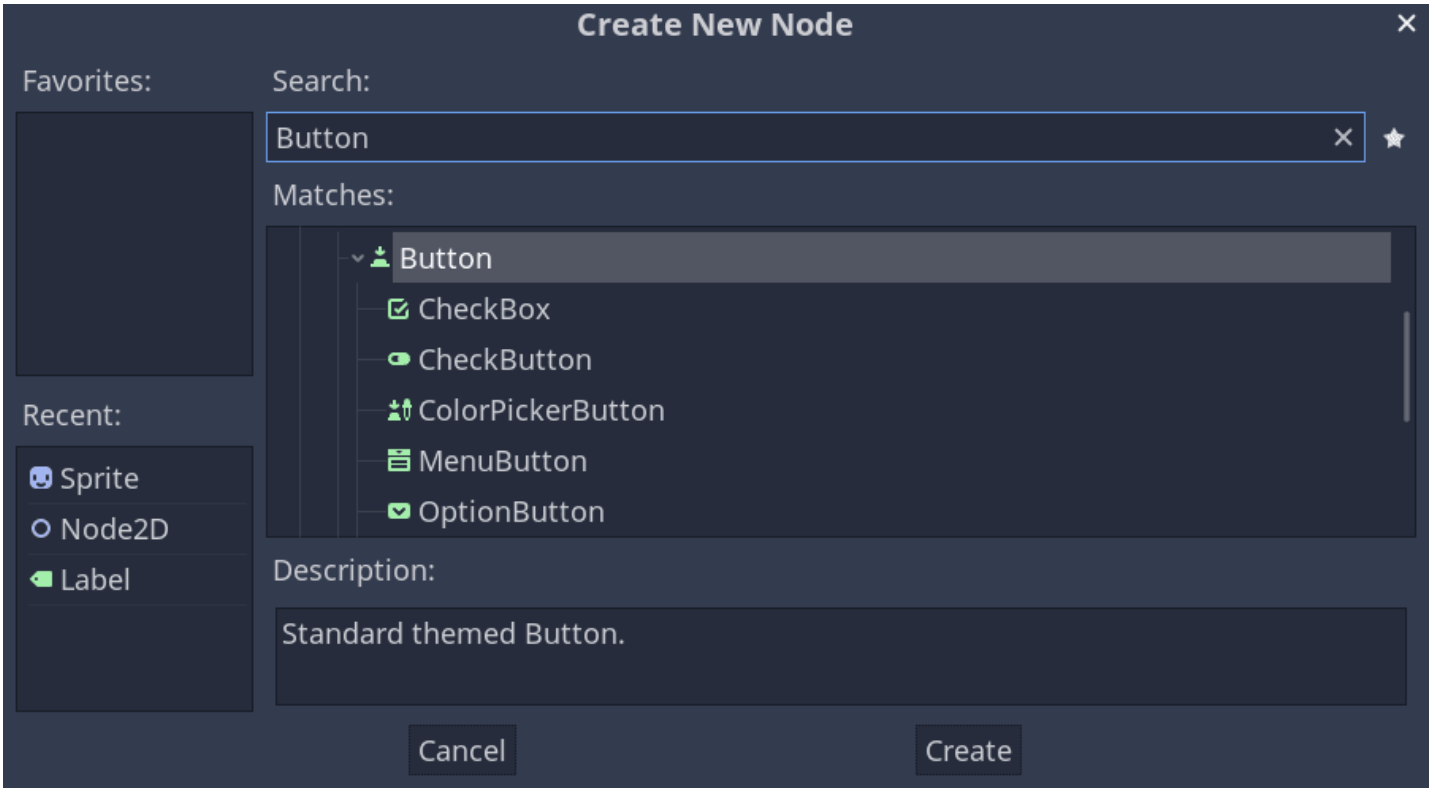
Нажмите и переместите файл `Sprite.tscn`, который Вы сохранили ранее, на панели задач файловой системы, на Node2D, чтобы создать экземпляр его класса.



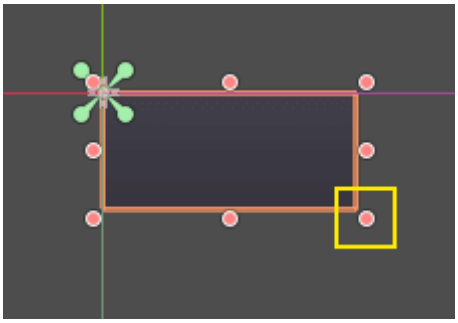
Мы хотим добавить другой узел, как соседний к спрайту. Чтобы сделать это, нажмите ПКМ на Node2D и выберите Добавить дочерний узел.



Выполните поиск узла типа Button и добавьте его.



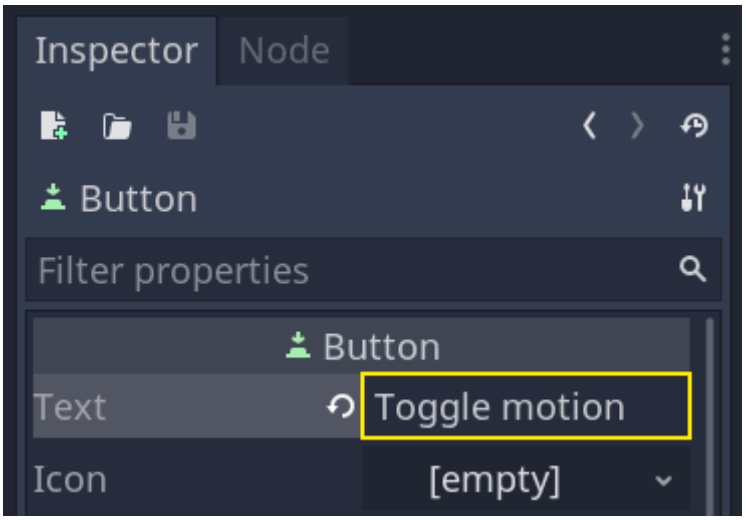
Узел является небольшим по умолчанию. Нажмите и потяните дескриптор кнопки в нижнем правом углу окна просмотра, чтобы изменить его масштаб.



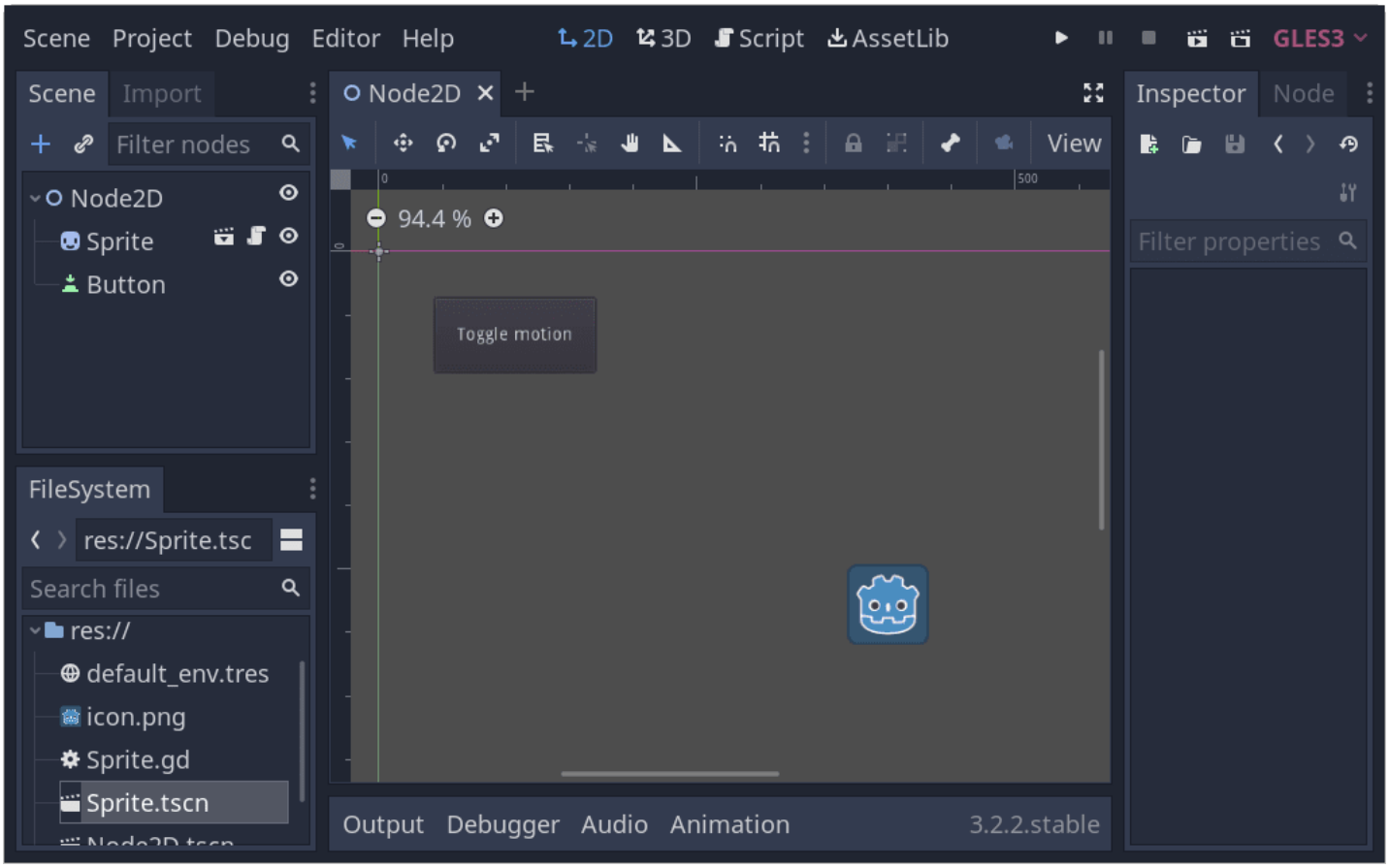
Если вы не видите ручки, убедитесь, что инструмент выделения активен на панели инструментов.

Нажмите на саму кнопку и перетащите курсор, чтобы приблизить её к спрайту.

Вы также можете написать метку на кнопке, отредактировав ее свойство Text в Инспекторе. Введите «Переключить движение».



Дерево сцены и область просмотра должны выглядеть следующим образом.

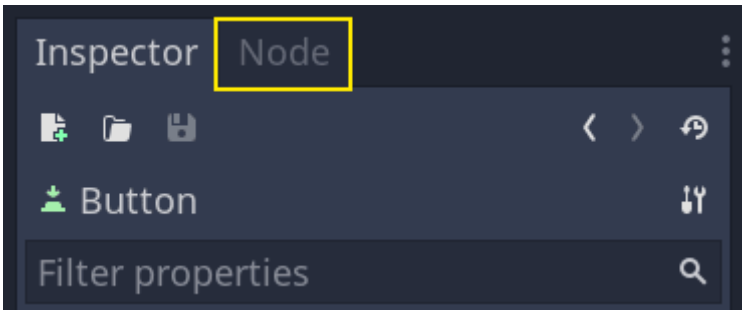


Сохраните только что созданную сцену. Затем вы можете запустить его с помощью F6. В данный момент кнопка будет видна, но если вы ее нажмете, ничего не произойдет.

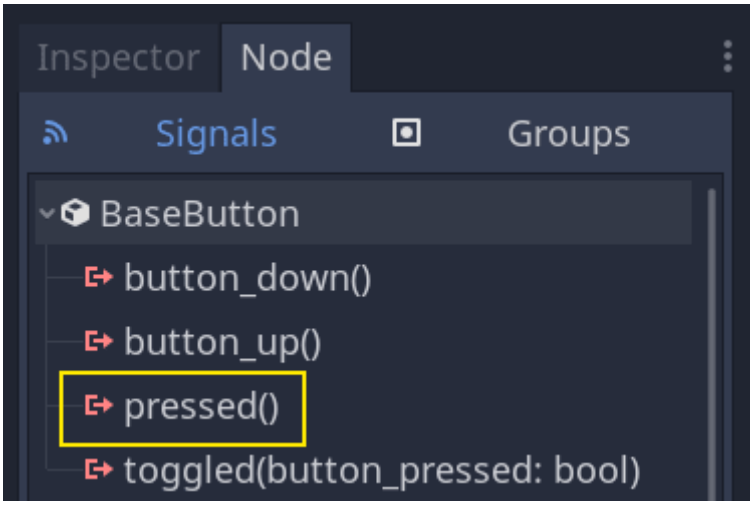
Подключение сигнала в редакторе

Здесь мы хотим присоединить сигнал "нажатие" Кнопки к нашему Спрайту, и мы хотим вызвать новую функцию, которая будет показывать и скрывать свои действия. Нам нужен скрипт, прикреплённый к узлу Спрайта, который мы напишем, исходя из предыдущего урока.

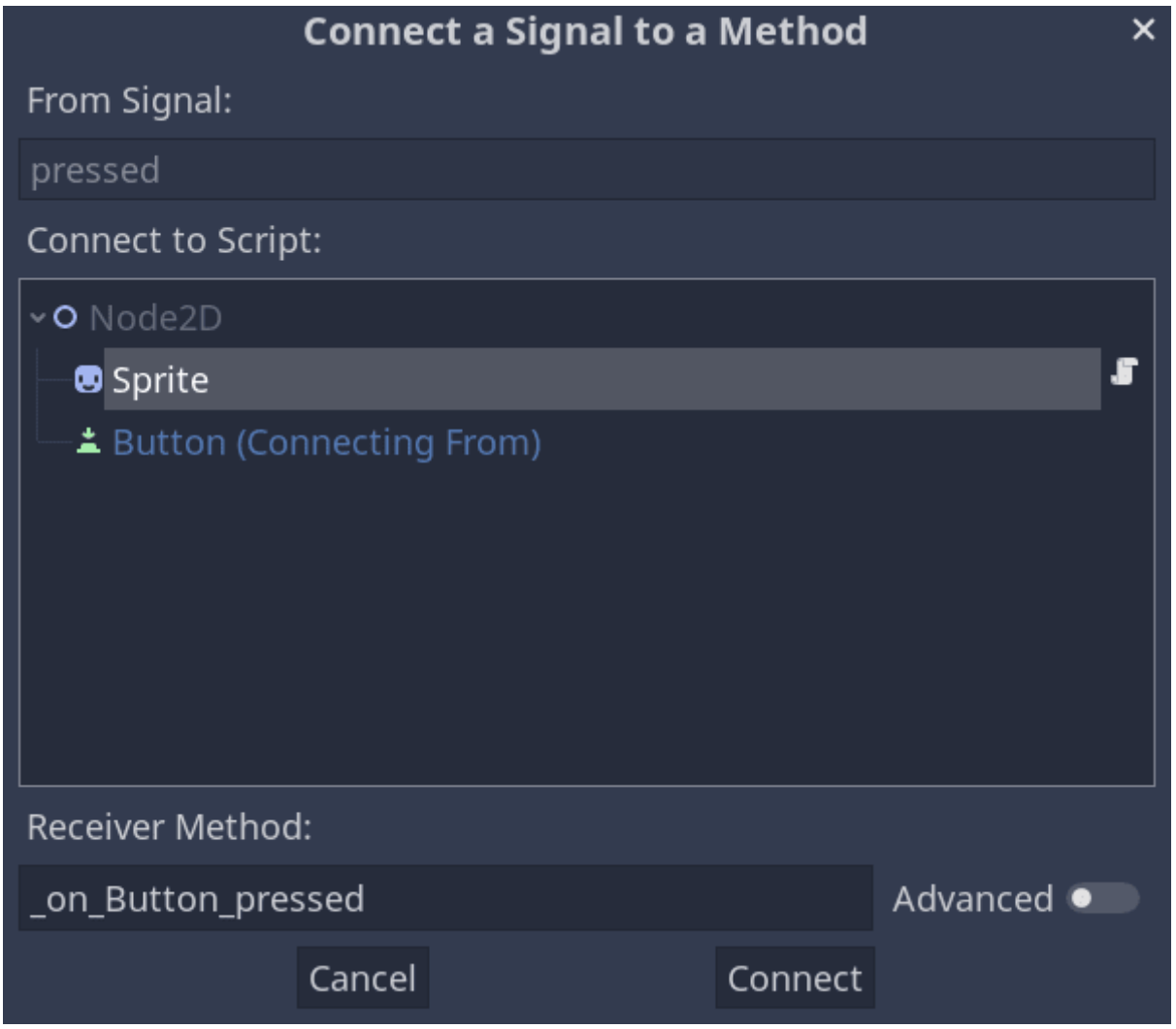
Вы можете подключить сигналы в панели Узел. Выберите узел Button и на правой стороне экрана нажмите вкладку "Узел" рядом с Инспектором.



На панели отображаются сигналы, доступные выбранному узлу.



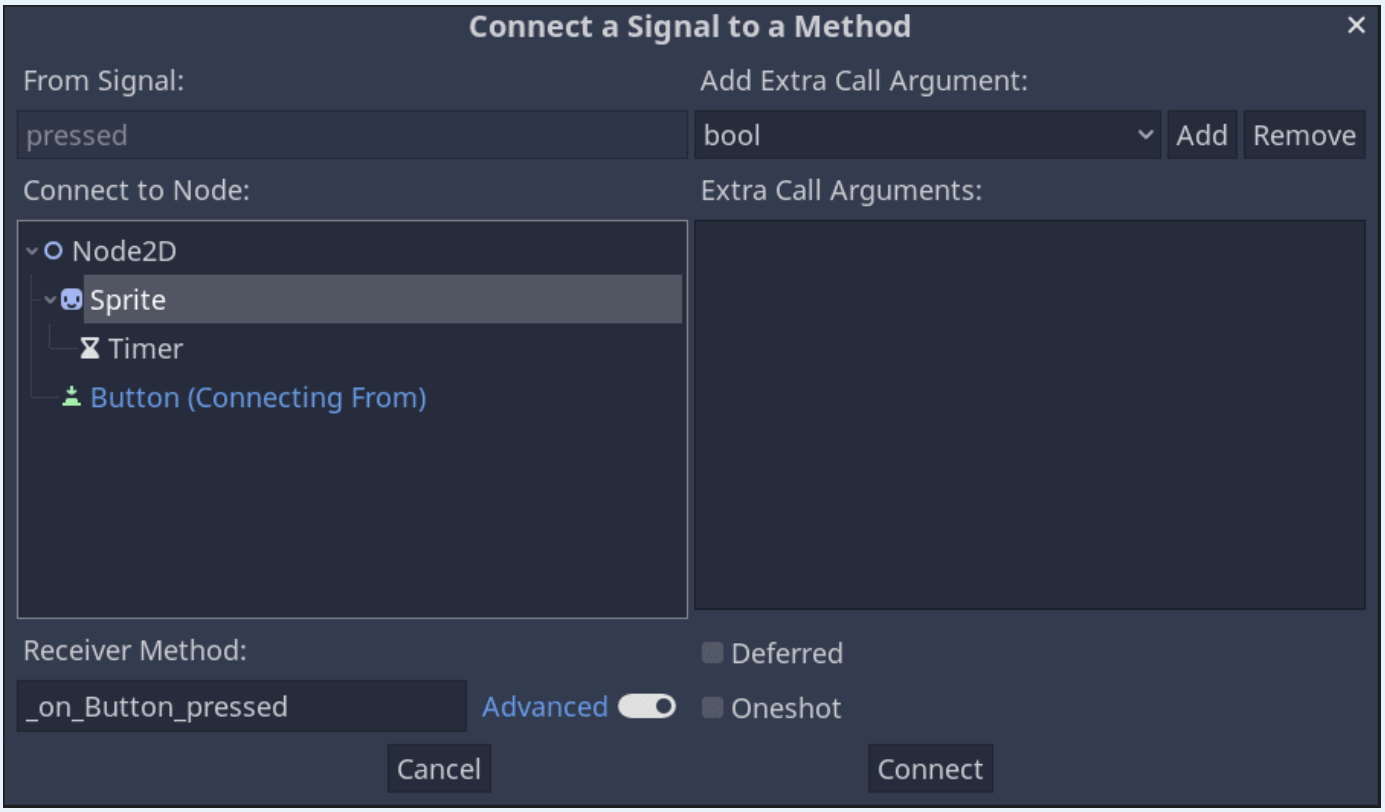
Дважды кликните сигнал "pressed", откроется окно присоединения узлов.



Там вы можете подключить сигнал к узлу Sprite. Узлу нужна функция, которую вызовет Godot, когда Button выдаст сигнал. Редактор самостоятельно создаст её. По соглашению, мы называем эти обратные вызовы "_on_NodeName_signal_name". У нас это будет "_on_Button_pressed".

Примечание

При подключении сигналов через вкладку Узел редактора, можно использовать два режима. Можно просто подключить сигнал к узлу, имеющему скрипт, и будет автоматически создана функция обратного вызова в нем.

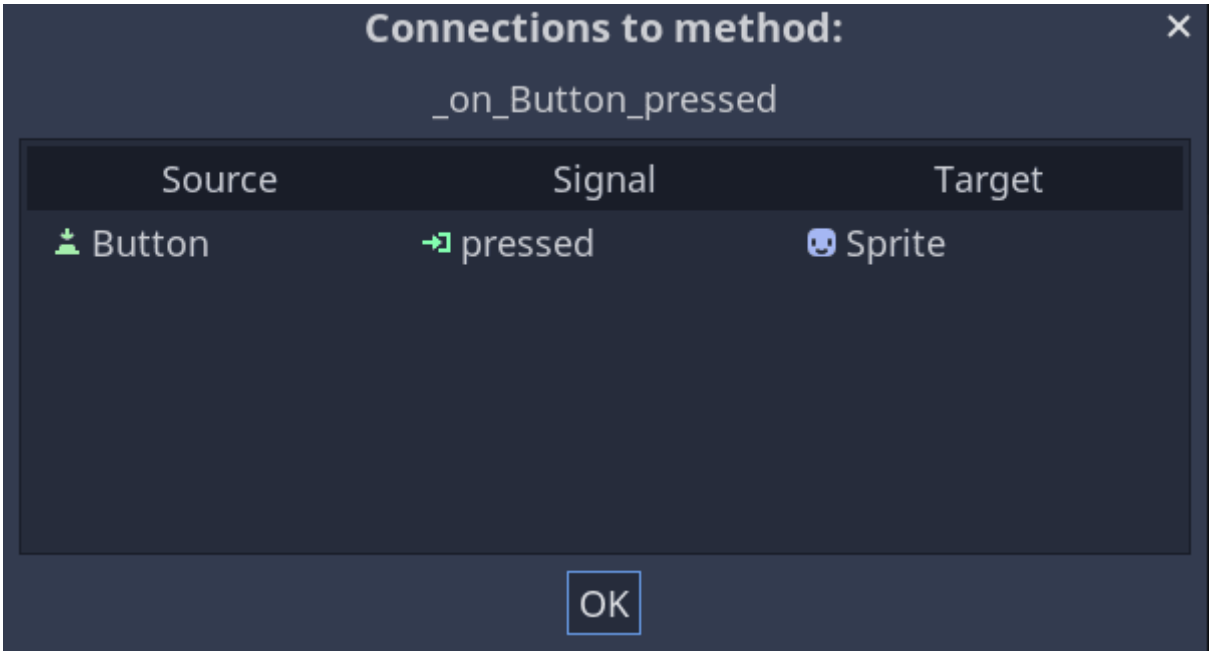


Расширенный вид позволяет подключаться к любому узлу и любой встроенной функции, добавлять аргументы к обратному вызову и устанавливать параметры. Вы можете переключить режим в правом нижнем углу окна, нажав кнопку «Дополнительно».

Нажмите кнопку Connect, чтобы завершить подключение сигнала и перейти в рабочую область Script. Вы должны увидеть новый метод со значком подключения в левом поле.

```
23 func _on_Button_pressed():
24     pass # Replace with function body.
25
```

Если нажать иконку, всплывёт окно с информацией о соединении. Это доступно только при присоединении узлов в редакторе.



Давайте заменим строку со словом `pass` кодом, который изменяет движение узла.

Наш спрайт движется благодаря коду функции `_process()`. Godot предоставляет метод для включения и выключения обработки: Node.set_process(). Другой метод класса Node, `is_processing()`, возвращает `true`, если обработка активна. Мы можем использовать ключевое слово ``not``для инвертирования значения.

GDScript

```
func _on_Button_pressed():
    set_process(not is_processing())
```

Эта функция будет переключать обработку и, в том числе, движение значка по нажатию кнопки.

Перед тем, как попробовать поиграть, нам необходимо упростить нашу функцию `_process()`, чтобы движение узла было автоматическим и не ожидало команд пользователя. Замените текущий код функции на тот, который мы видели два урока назад:

GDScript

```
func _process(delta):
    rotation += angular_speed * delta
```

```
    var velocity = Vector2.UP.rotated(rotation) * speed
    position += velocity * delta
```

Ваш полный код `Sprite.gd` должен выглядеть следующим образом.

GDScript

```
extends Sprite

var speed = 400
var angular_speed = PI


func _process(delta):
    rotation += angular_speed * delta
    var velocity = Vector2.UP.rotated(rotation) * speed
    position += velocity * delta


func _on_Button_pressed():
    set_process(not is_processing())
```

Запустите сцену и нажмите кнопку, чтобы увидеть запуск и остановку спрайта.

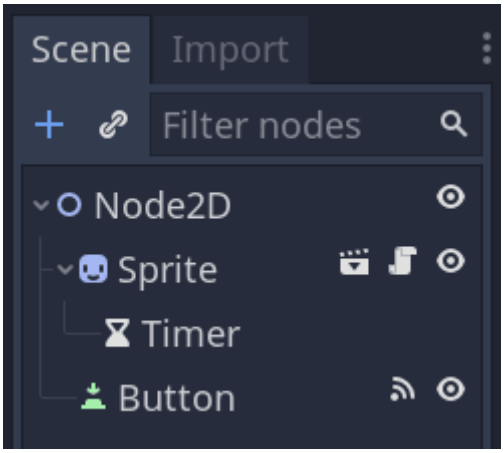
Подключение сигналов в коде

Вы можете присоединять сигналы в коде вместо использования редактора. Это нужно, когда узлы или элементы сцены создаются в скрипте.

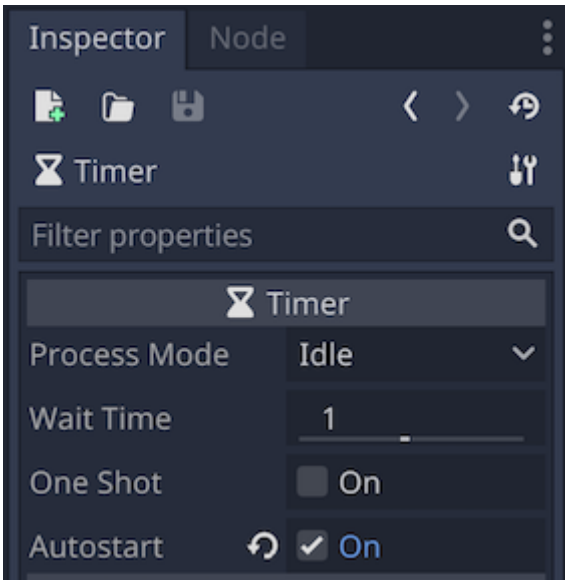
Давайте используем различные узлы здесь. У Godot есть узел Timer, который полезен для реализации задержки перезарядки способностей, перезарядки оружия и другого.

Вернёмся к рабочему пространству 2D. Для этого можно нажать "2D" вверху экрана, или на клавиатуре `Ctrl + F1` (`Alt + 1` для macOS).

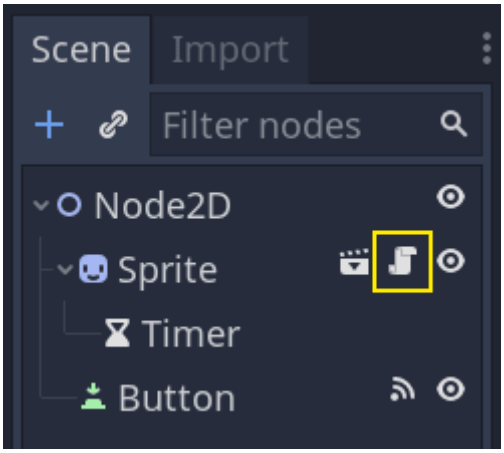
Нажмите правой кнопкой мыши на панели задач на узел Спрайт и добавьте дочерний узел. Найдите Таймер и добавьте соответствующий узел. Теперь ваша сцена должна выглядеть так.



Выбрав узел Таймера, откройте Инспектор и проверьте свойства **Autostart**.



Нажмите на иконку скрипта рядом со Спрайтом, чтобы вернуться к рабочей области скрипта.



Нам необходимо выполнить две операции для подключения узлов через код:

1. Получите ссылку на таймер из Спрайта.

2. Вызвать метод `connect()` Таймера.

Примечание

Для подключения к сигналу через код, Вам нужно использовать метод `connect()` узла, который вы хотите прослушать. В этом случае, мы хотим прослушать сигнал Таймера "тайм-аут".

Мы хотим подключить сигнал, когда сцена инстанцируется, и мы можем это сделать, используя встроенную функцию Node._ready(), которая автоматически вызывается движком, когда узел полностью инстанцирован.

Чтобы получить ссылку на узел, относящийся к действующему узлу, мы используем метод Node.get_node(). Мы можем сохранить ссылку в переменной.

GDScript

```
func _ready():
    var timer = get_node("Timer")
```

Функция . `get_node()``анализирует дочерний Спрайт и получает узлы по их имени. К примеру, если Вы переименуете в редакторе узел Таймера в "BlinkingTimer", то Вам нужно будет поменять вызов на ``get_node("BlinkingTimer")`

Сейчас мы можем подключить Таймер к Спрайту в функции `_ready()`.

GDScript

```
func _ready():
    var timer = get_node("Timer")
    timer.connect("timeout", self, "_on_Timer_timeout")
```

Строка читается так: мы подключаем сигнал Таймера "timeout" к узлу, к которому прикреплён скрипт (`self`). Когда Таймер испускает "timeout", мы хотим вызвать функцию

"_on_Timer_timeout", которую нам необходимо определить. Давайте добавим её в нижней части нашего скрипта и используем её для переключения видимости нашего спрайта.

GDScript

```
func _on_Timer_timeout():
    visible = not visible
```

Логическое свойство `visible` контролирует видимость нашего узла. Строка `visible = not visible` переключает значение. Если `visible` равно `true`, оно станет `false`, и наоборот.

Если вы запустите сцену сейчас, вы увидите, что спрайт мигает и гаснет с интервалом в одну секунду.

Готовый скрипт

Вот и все для нашей маленькой движущейся и мигающей иконки Годо! Вот полный файл `Sprite.gd` для справки.

GDScript

```
extends Sprite

var speed = 400
var angular_speed = PI


func _ready():
    var timer = get_node("Timer")
    timer.connect("timeout", self, "_on_Timer_timeout")


func _process(delta):
    rotation += angular_speed * delta
    var velocity = Vector2.UP.rotated(rotation) * speed
    position += velocity * delta


func _on_Button_pressed():
```

```
    set_process(not is_processing())


func _on_Timer_timeout():
    visible = not visible
```

Пользовательские сигналы

Примечание

Этот отдел является справкой о том, как обозначать и использовать Ваши собственные сигналы, и не опирается на проекты, созданные в предыдущих уроках.

Вы можете определить собственные сигналы в скрипте. Например, вы хотите вывести экран "Конец игры", когда здоровье игрока станет равным нулю. Для этого вы можете определить сигнал "died" или "health_depleted", и выдать его, когда здоровье игрока будет 0.

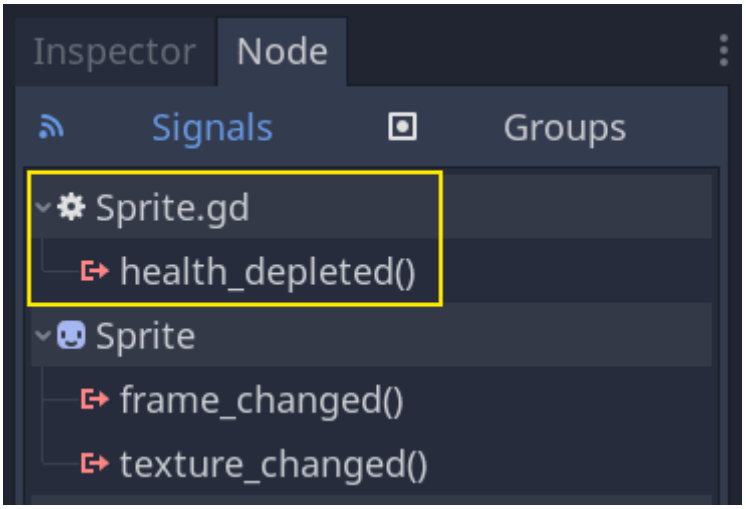
GDScript

```
extends Node2D

signal health_depleted

var health = 10
```

Примечание

Поскольку сигналы представляют собой события, которые только что произошли, мы обычно используем в их названиях глагол действия в прошедшем времени.

Ваши сигналы работают таким же образом, как и встроенные: они появляются во вкладке Узла, и Вы можете подключить к ним любые другие сигналы.



Чтобы излучать сигнал в скриптах, вызовите `emit_signal()`.

GDScript

```
func take_damage(amount):
    health -= amount
    if health <= 0:
        emit_signal("health_depleted")
```

Сигнал может дополнительно объявить один или несколько аргументов. Укажите имена аргументов между круглыми скобками:

GDScript

```
extends Node

signal health_changed(old_value, new_value)
```

Примечание

Эти аргументы показываются в док-узлах редактора, и Godot может использовать их, чтобы производить для вас функции обратного вызова. Однако, вы всё ещё можете отправлять любое число аргументов при отправке сигналов; отправка правильных значений зависит от вас.

Чтобы выдать значения вместе с сигналом, добавьте их в качестве дополнительных аргументов к функции `emit_signal()`:

GDScript

```
func take_damage(amount):
    var old_health = health
    health -= amount
    emit_signal("health_changed", old_health, health)
```

Подведение итогов

Любой узел в Godot излучает сигналы, когда с ним происходит что-то особенное, как например, нажатие на кнопку. Другие узлы могут подключиться к индивидуальным сигналам и среагировать на выбранные события.

Сигналы имеют множество применений. С их помощью Вы можете отреагировать на узел при выходе и входе в игровой мир, на столкновение, на персонажа, входящего или покидающего какую-либо область, на изменение в масштабе элемента интерфейса и на многое другое.

Например, Area2D, представляющий монету, испускает сигнал `body_entered` всякий раз, когда физическое тело игрока входит в его форму столкновения, что позволяет узнать, когда игрок подобрал монету.

В следующем разделе, Ваша первая 2D игра, вы создадите полноценную 2D-игру и примените на практике всё, чему научились до сих пор.