Использование контейнеров

Якоря — это эффективный способ обработки различных соотношений сторон для базовой обработки нескольких разрешений в графических интерфейсах.

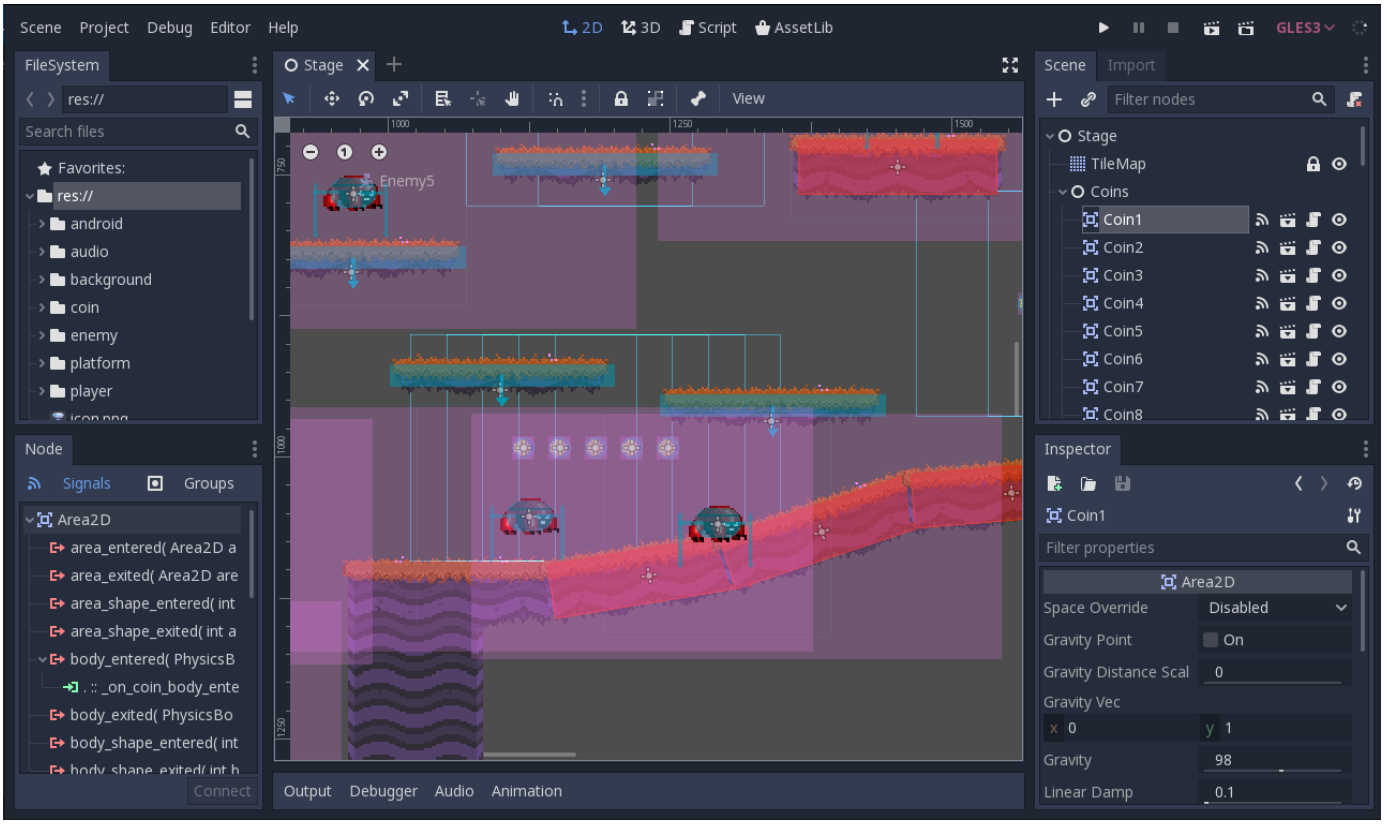
Для более сложных пользовательских интерфейсов они могут стать трудными в использовании.

Это часто бывает в играх, таких как ролевые игры, онлайн-чаты, магнаты или симуляторы. Еще один распространенный случай, когда могут потребоваться более сложные функции компоновки, — это внутриигровые инструменты (или просто инструменты).

Во всех этих ситуациях требуется более функциональный пользовательский интерфейс, похожий на ОС, с расширенным макетом и форматированием. Для этого контейнеры более полезны.

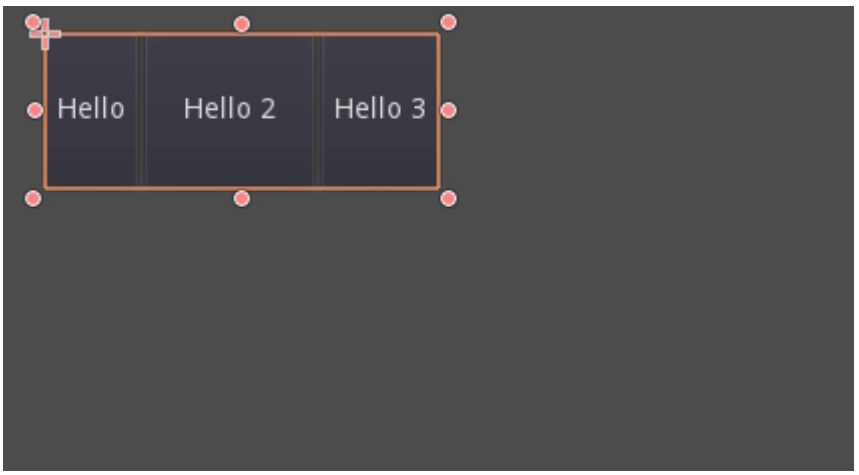
Компоновка контейнера

Контейнеры предоставляют огромные возможности макета (например, пользовательский интерфейс редактора Godot полностью выполнен с их использованием):



Когда используется узел, производный от Container , все дочерние узлы Control отказываются от своих собственных возможностей позиционирования. Это означает, что *Контейнер* будет контролировать их расположение, и любая попытка вручную изменить эти узлы будет либо проигнорирована, либо признана недействительной при следующем изменении размера их родителя.

Аналогичным образом, когда размер узла, производного от *контейнера* , изменяется, все его дочерние элементы будут перемещены в соответствии с ним с поведением, основанным на типе используемого контейнера:

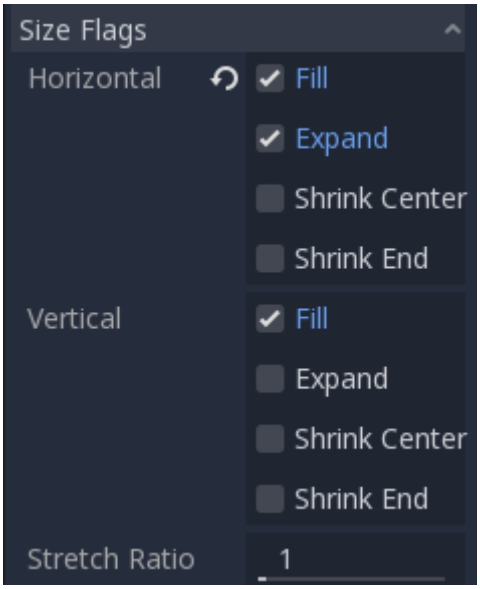


Пример изменения размера дочерних кнопок *HBoxContainer* .

Настоящая сила контейнеров заключается в том, что они могут быть вложены друг в друга (как узлы), что позволяет создавать очень сложные макеты, размеры которых легко изменяются.

Флаги размера (Size Flags)

При добавлении узла в контейнер то, как контейнер обрабатывает каждого потомка, в основном зависит от их *флагов размера* . Эти флаги можно найти, проверив любой элемент управления, который является дочерним элементом *Container* .



Флаги размера независимы для вертикального и горизонтального размера, и не все контейнеры используют их (но большинство):

- **Fill** : Гарантирует, что элемент управления *заполняет* указанную область внутри контейнера. Независимо от того, *расширяется* элемент управления или нет (см. ниже), он будет *заполнять* назначенную область только тогда, когда он включен (по умолчанию).
- **Expand** : пытается использовать как можно больше места в родительском контейнере (по каждой оси). Элементы управления, которые не расширяются, будут оттеснены теми, которые расширяются. Между расширяющимися элементами управления количество пространства, которое они занимают друг от друга, определяется соотношением *(* см. ниже).
- **Центр сжатия** При расширении (и если не заполнении) старайтесь оставаться в центре расширенной области (по умолчанию она остается слева или вверху).
- **Ratio** Простое соотношение того, насколько расширенные элементы управления занимают доступное пространство по отношению друг к другу. Элемент управления с «2» займет в два раза больше свободного места, чем элемент с «1».

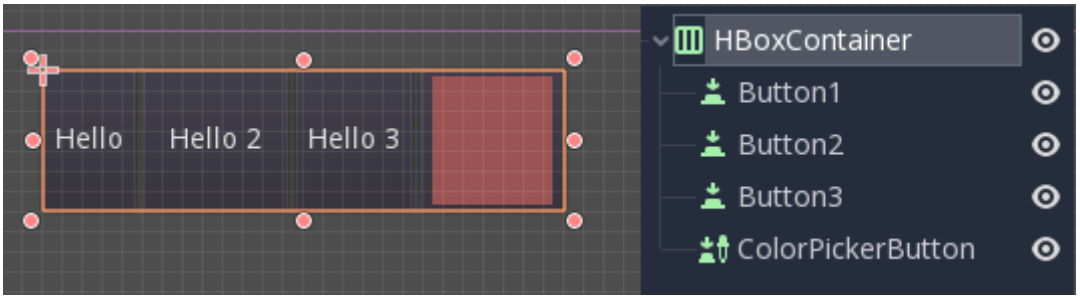
Рекомендуется поэкспериментировать с этими флагами и различными контейнерами, чтобы лучше понять, как они работают.

Типы контейнеров

Godot предлагает несколько типов контейнеров из коробки, поскольку они служат разным целям:

Коробка Контейнеры

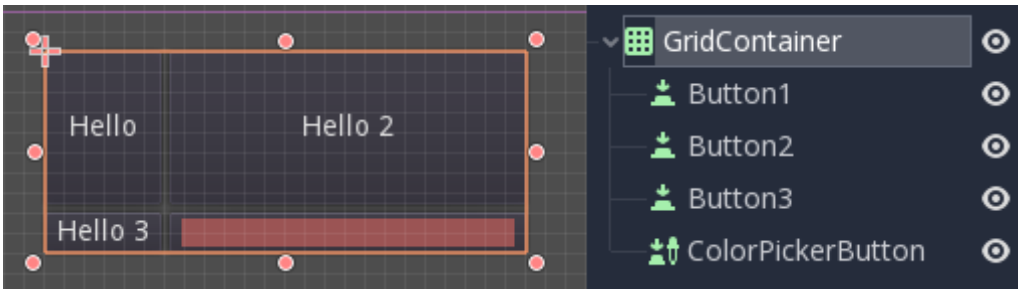
Упорядочивает дочерние элементы управления вертикально или горизонтально (через HBoxContainer и VBoxContainer). В направлении, противоположном указанному (например, вертикально для горизонтального контейнера), он просто расширяет дочерние элементы.



Эти контейнеры используют свойство *Ratio* для дочерних элементов с установленным флажком *Expand* .

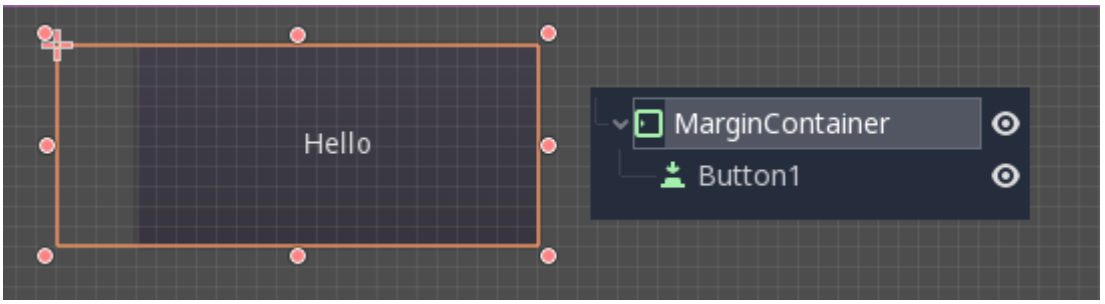
Контейнер сетки

Упорядочивает дочерние элементы управления в виде сетки (через GridContainer необходимо указать количество столбцов). Использует как вертикальные, так и горизонтальные флаги расширения.



Контейнер маржи

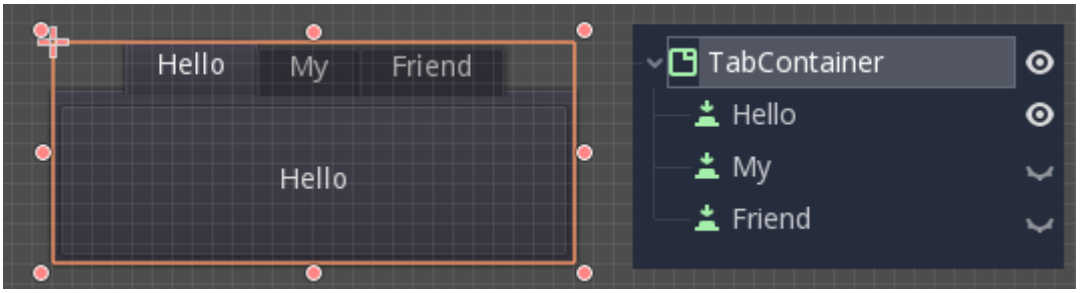
Дочерние элементы управления расширяются до границ этого элемента управления (через MarginContainer). Отступы будут добавлены на поля в зависимости от конфигурации темы.



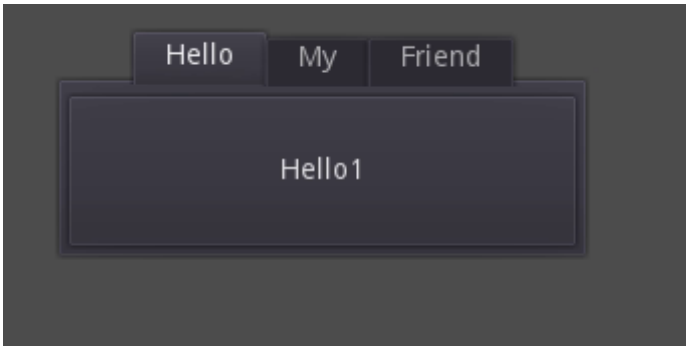
Опять же, имейте в виду, что поля являются значением *темы* , поэтому их необходимо редактировать в разделе переопределений констант каждого элемента управления:

Контейнер вкладок

Позволяет размещать несколько дочерних элементов управления друг над другом (через TabContainer), при этом видимым является только *текущий* элемент.



Изменение *текущего* осуществляется через вкладки, расположенные в верхней части контейнера, нажатием:

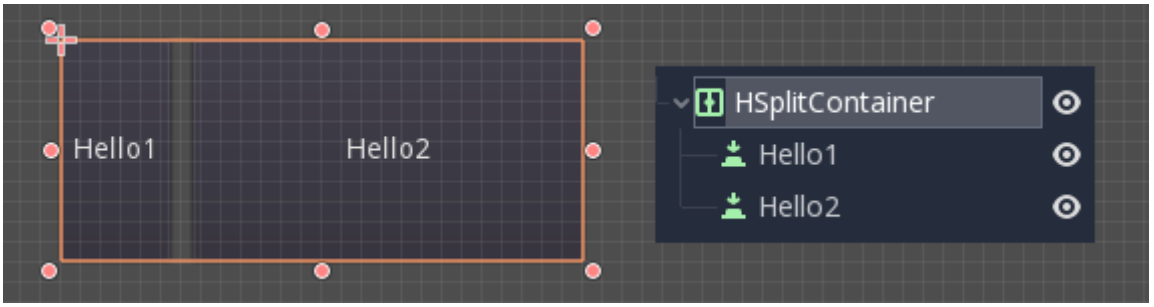


Заголовки генерируются из имен узлов по умолчанию (хотя их можно переопределить через *TabContainer* API).

Такие параметры, как размещение вкладок и *StyleBox* , можно изменить в переопределениях темы *TabContainer* .

Сплит-контейнер

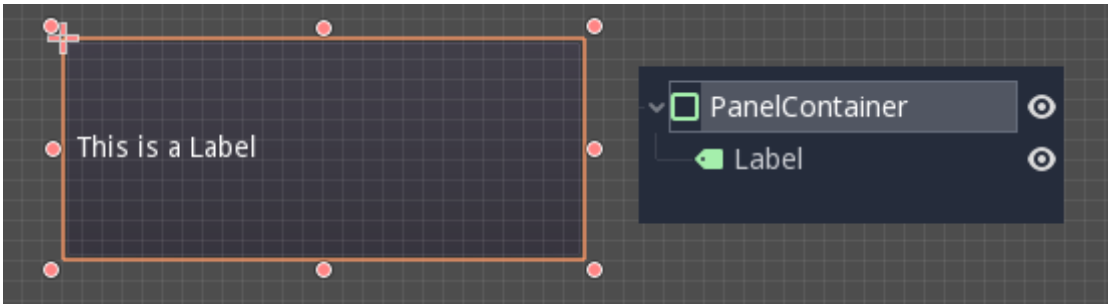
Принимает только один или два дочерних элемента управления, затем размещает их рядом с делителем (через HSplitContainer и VSplitContainer). Учитывает как горизонтальные, так и вертикальные флаги, а также *Ratio* .



Делитель можно перетаскивать, чтобы изменить соотношение размеров между обоими дочерними элементами:

ПанельКонтейнер

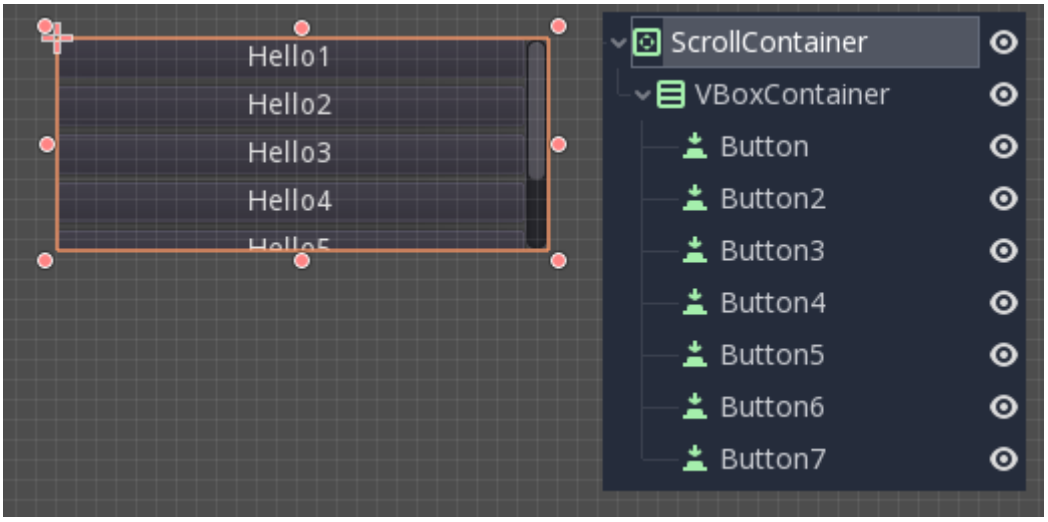
Простой контейнер, который рисует *StyleBox* , а затем расширяет дочерние элементы, чтобы покрыть всю его область (через PanelContainer , соблюдая поля *StyleBox*). Он учитывает как горизонтальные, так и вертикальные флаги размера.



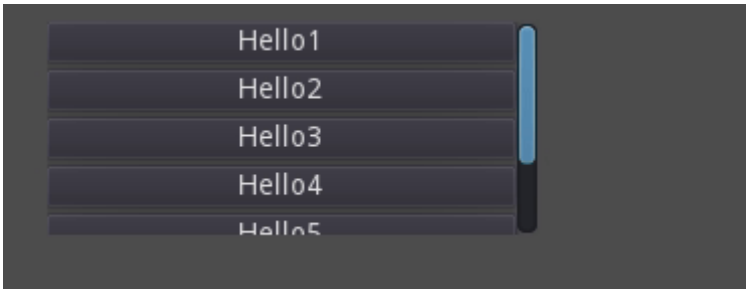
Этот контейнер полезен в качестве верхнего уровня или просто для добавления пользовательских фонов в разделы макета.

ScrollContainer

Принимает один дочерний узел. Если этот узел больше, чем контейнер, будут добавлены полосы прокрутки, позволяющие перемещать узел вокруг (через ScrollContainer). Учитываются флаги как вертикального, так и горизонтального размера, и поведение можно включить или отключить для каждой оси в свойствах.



Колесо мыши и сенсорное перетаскивание (когда доступно сенсорное управление) также являются допустимыми способами панорамирования дочернего элемента управления.



Как и в приведенном выше примере, один из наиболее распространенных способов использования этого контейнера — вместе с *VBoxContainer* в качестве дочернего элемента.

ViewportContainer

Это специальный элемент управления, который принимает только один узел *Viewport* в качестве дочернего и отображает его, как если бы это было изображение (через ViewportContainer).

Создание пользовательских контейнеров

С помощью скрипта можно легко создать собственный контейнер. Вот пример простого контейнера, который соответствует размеру дочерних элементов:

GDScript

```
extends Container

func _notification(what):
```

```
    if what == NOTIFICATION_SORT_CHILDREN:
        # Must re-sort the children
        for c in get_children():
            # Fit to own size
            fit_child_in_rect( c, Rect2( Vector2(), rect_size ) )

func set_some_setting():
    # Some setting changed, ask for children re-sort
    queue_sort()
```