Содержание

Полезные подсказки:

- В этой части книги мы будем регулярно экспериментировать с разными скриптами, которые, будут управлять одними и теми же **parts**. Например, со скриптами, управляющими камерой игрового мира Roblox. Камера одна, а программ управляющих ею будет несколько. Что делать? Стирать скопированные из этой книжки программы не нужно, они вам еще пригодятся. Поэтому, отключайте, скрипты с которыми вы уже разобрались, делая их ***неактивными***. Подсветите название скрипта в окне **Explorer** и, нажав правую кнопку мыши, выберите опцию **DisableScript**. Скрипт не будет удален, а вы получите возможность управлять этим же объектом при помощи новой программы. Для активации деактивированной программы выберите для нее **EnableScript** в том же меню;

- не ленитесь — не забывайте писать понятные названия скриптов и подробные комментарии к скриптам. Сохраняя скрипты в своих архивах, вы сэкономите себе массу времени, когда они вам понадобятся, ведь скопировать код из книжки гораздо труднее, чем скопировать электронный текст из файла;

- держите постоянно включенными окна **Output** и **Script Analysis** и контролируйте наличие в них рекомендуемых режимов. Если настройки режимов изменятся, вы можете не сразу заметить ошибки в своих скриптах.

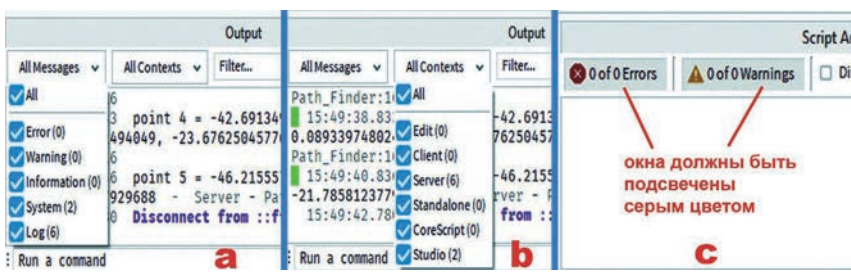

Рис. 0.1 Рекомендуемые установки режимов работы окон Output и cryptAnalysis

- если код, который вы написали не работает, и вы не понимаете почему, а **Script Analisis** не показывает ошибок, проверьте:
  1/ в тот ли раздел **Explorer** вы вставили ваш код;
  2/ не должен ли быть ваш код *локальным* или наоборот;
  3/ не *деактивирован* ли, случайно, этот скрипт;
  4/ используйте функцию **print**;

- Как использовать функцию **print:**
  1/ вставьте, например, **print(“Test”)** максимально близко к тому месту программы, которое не работает. Этим вы определите доходит ли исполнение программы до этого места скрипта или нет;

  2/ там, где вы используете оператор **if-then** или **while (условие)** вставьте перед этим оператором распечатку тех переменных, чье значение оператор проверяет. Например, у вас не работает код:

```
elseif event == "Up/Down" then
        if data == 1 and trigger == false and Height ==0 then
```

  а/ вставьте `print (event)` перед проверкой **event** — то есть, перед оператором **elseif**, а также в месте программы, где формируется значение **event**;

  б/ вставьте функцию **print** для проверки значений переменных **data, trigger** и **Height**, а именно:

```
elseif event == "Up/Down" then
        print (tostring(data), trigger, tostring (Height))
        if data == 1 and trigger == false and Height ==0 then
```

1. Кодовые панели

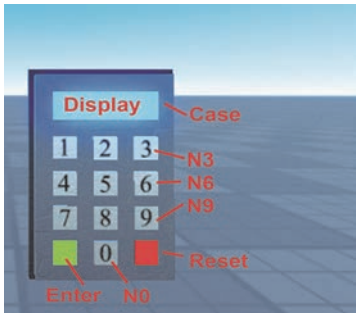

Рис. 1.1 Кодовая панель

Кодовая панель - штука интересная, ее можно много где использовать. Рассмотрим, пример программирования кодовой панели, созданной @Sakamotoo *www.roblox.com/users/2436160/profile.* Конструктивно его панель достаточно проста и включает в себя корпус (Case), дисплей (Display), номерные кнопки (N0…N9), кнопку ввода (Enter) и кнопку сброса (Reset).

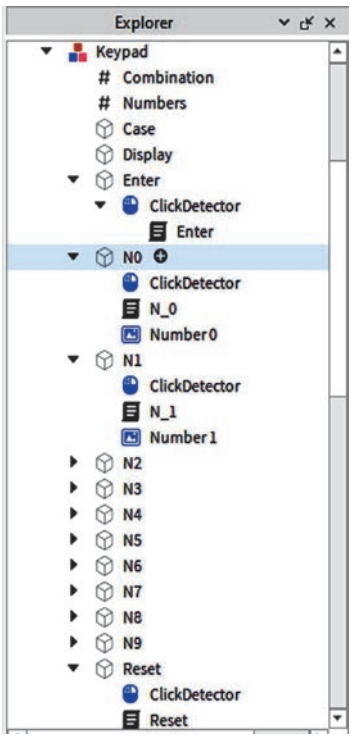

Рис. 1.2 Состав объекта Keypad

Объекты #**Combination** и #**Numbers** являются объектами типа конфигурация. Конфигурация — это контейнер, предназначенный для хранения переменных. Их использование необязательно, но хранящиеся в них значения могут легко считываться разными скриптами. В нашем случае, конфигурация #**Combination** хранит “правильную” комбинацию кода, а конфигурация #**Numbers** хранит набираемый игроком код.

В состав каждой кнопки панели входят: **ClickDetector** и скрипт кнопки. В номерные кнопки дополнительно входят декали с номерами кнопок, имеющие имена типа **Number N,** где **N** - номер кнопки. Скрипты номерных кнопок отличаются лишь именем **parts** кнопок в самой первой строке скриптов (N0 …N9).

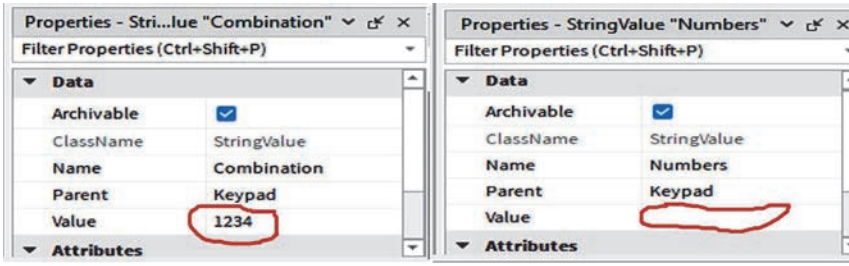

Рис. 1.3 Значения конфигураций Combination и Numbers

Ниже приведен скрипт номерной кнопки с декалем “2” (наберите его текст вместе с названием-комментарием, как показано ниже):

```
- - скрипт номерной кнопки с декалем “2” кодового устройства
1     local button = script.Parent.Parent.N2
2     script.Parent.ClickDetector.MouseClick:Connect(function()
3           script.Parent.Parent.Numbers.Value =
 script.Parent.Parent.Numbers.Value..string.sub(script.Parent.Name, 2)
4           print(script.Parent.Name)
5           print(script.Parent.Parent.Numbers.Value)
6           local audio = Instance.new("Sound")
7           audio.SoundId = "rbxassetid://5110339580"
8           audio.Looped = false
9           audio.Parent = script.Parent.Parent.Case
10          audio:Play()
11          button.Material = "Neon"
12          wait(0.6)
13          audio:Destroy()
14          button.Material = "Metal"
15    end)
```

— строка 1 определяет принадлежность переменной **button** объекту **N2** из состава **Keypad**. То есть, мы имеем дело со скриптом кнопки с декалем “2”, имеющей имя “**N2**”;
— строка 2 подключает **ClickDetector**, входящий в состав этой кнопки к функции без имени, которая при каждом нажатии этой кнопки *будет:*
— в строке 3 менять текущее значение кода, хранящееся в свойстве **Value** контейнера конфигурации #**Number**, *приписывая* правее к уже имеющемуся в нем значению кода значение, равное ***второму***

11. Бейджи в ролевых играх

Рис. 11.1 Ролевая игра

Ролевые игры очень популярны. Давайте, посмотрим, что можно сделать интересного для разработки таких игр.

Мы с вами научились разрабатывать **ScreenGui**, научились выдавать бейджи и контролировать наличие определенных бейджей у игроков. А что, если попробовать применить эти знания к созданию ролевых игр?

Например, в военной ролевой игре, вы можете организовать выдачу бейджей, которые будут соответствовать разным воинским званиям. По этим бейджам игроки будут видеть свои успехи в игре. А мы, чтобы повысить привлекательность игры, сделаем так, чтобы при входе в игру, ваша игра приветствовала каждого игрока в соответствии с его воинским званием, например: “***Здравия желаю, капитан!***”. А на экран выводилось изображение капитанских погон.

При этом, давайте используем все элементы **ScreenGui**, созданные нами в главе 3 в **StarterGui**:
— **TextButton** PLAY и RULES;
— **ScrollingFrame** с правилами игры;
— **TextBox** обратного отсчета.

Дополнительно нужно:
— разработать в **Photoshop** или в другом графическом редакторе дизайн бейджей, соответствующих разным воинским званиям;
— ввести бейджи в свою игру (конечно же, нужно игру опубликовать в Roblox) и получить идентификаторы бейджей;

— строка 12 подключает индикатор событий **Path.Blocked** к функции

IF_Blocked.
Еще один важный момент: вы вдруг можете увидеть, что ваш NPC передвигается, например, спиной вперед. Что с этим делать? Подскажем, введите в ваш скрипт еще пару строк:

```
1       local ori = workspace.Soldiers.Soldier_2.HumanoidRootPart
2       ori.Orientation = ori.Orientation + Vector3.new(0,180,0)
```

— в строке 1 мы присваиваем переменной **ori** значение **HumanoidRootPart** нашего солдата;
— в строке 2 меняем эту ориентацию так, как нам нужно.

Мы, например, развернули нашего **Soldier_2** на 180 градусов вокруг оси Y. Вам придется поворачивать NPC и, когда вы захотите, чтобы он возвращался по выбранному вами маршруту в исходную точку.

13. Лучи

Рис. 12.1 Луч

Вы уже знаете, что объект **Beam** соединяет два **Attachment**, рисуя между ними какие-нибудь текстуры. Лучи должны быть потомками **Workspace**, а их свойства **Beam.Attachment0** и **Beam.Attachment1** также должны быть соединены с потомками **Workspace**.

Ниже приведен скрипт **ServerScriptService**, рисующий приведенный на картинке луч:

```
- - скрипт рисования красивого луча
1       local att0 = Instance.new("Attachment")
2       local att1 = Instance.new("Attachment")
```

```
3   att0.Parent = workspace.Terrain
4   att1.Parent = workspace.Terrain
5   att0.Position = Vector3.new(-150, 15, -20)
6   att1.Position = Vector3.new(-150, 5, -30)
7   local beam = Instance.new("Beam")
8   beam.Attachment0 = att0
9   beam.Attachment1 = att1
10  beam.Color = ColorSequence.new({
            ColorSequenceKeypoint.new(0, Color3.fromRGB(255, 0, 0)),
            ColorSequenceKeypoint.new(0.5, Color3.fromRGB(0, 255, 0)),
            ColorSequenceKeypoint.new(1, Color3.fromRGB(0, 0, 255)), })
11  beam.LightEmission = 1 -- use additive blending
12  beam.LightInfluence = 1 -- beam not influenced by light
13  beam.Texture = "rbxasset://textures/particles/sparkles_main.dds"
14  beam.TextureMode = Enum.TextureMode.Wrap
15  beam.TextureLength = 1
16  beam.TextureSpeed = 1
17  beam.Transparency = NumberSequence.new({
            NumberSequenceKeypoint.new(0, 0),
            NumberSequenceKeypoint.new(0.8, 0),
            NumberSequenceKeypoint.new(1, 0), })
18  beam.CurveSize0 = 20
19  beam.CurveSize1 = -20
20  beam.FaceCamera = true
21  beam.Segments = 100
22  beam.Width0 = 4
23  beam.Width1 = 4
24  beam.Enabled = true
25  beam.Parent = att0
```

Поясним скрипт:
— строки 1 и 2 создают два объекта типа **Attachment**;
— строки 3 и 4 определяют Родителей созданных объектов;
— строки 5 и 6 задают координаты расположения объектов;
— строка 7 создает объект типа **Beam**;
— строки 8 и 9 определяют точки начала и конца объекта **Beam**;

— строка 11 обеспечивает повторный запрос на пересылку информации;
— в строке 13 (если передача информации прошла нормально) проводится проверка: какую информацию переслал сервис, и если сервис сообщил о наличии **gamepass** у игрока:
— в строке 14 программа печатает сообщение “Игрок с именем … обладает нужным пропуском”;
— в строке 15 (или в нескольких строках) должен быть вставлен код, присваивающий проверяемому игроку необходимые привилегии. Если же, вы разместили **gamepass-донат**, то здесь вы можете поместить код, который выведет **ScreenGui** с текстом вашей благодарности тому, кто оплатил донат;
— строка 18 подключает вошедшего в игру игрока к функции **check_Gamepass().**

Заключение

Вы дочитали нашу книгу. Но, это — не конец, а лишь начало вашего пути в мир программирования игр. На этом длинном пути вы, наверняка, будете сталкиваться с разочарованиями, неудачами и ошибками. Но, когда это будет происходить, пожалуйста, помните, что вы не одиноки, и у вас есть секретное оружие. Его название — сообщество программистов всего мира. Свяжитесь с этим сообществом, заведите друзей, которым также, как вам, интересен мир творчества, поддерживайте друг друга, делитесь информацией и опытом. Классные игры создаются только в сотрудничестве.

Примеры, приведенные в нашей книге, были довольно просты, это означает, что они, возможно, не представляют собой самые эффективные решения и существуют другие решения. Кроме того, мы оставили без внимания очень много действительно интересных вещей, которые можно реализовать в Roblox Studio. Мы сосредоточились на понимании основ создания различных элементов игр. Если вы действительно хотите разобраться с программированием игр — учитесь! Пользуйтесь сайтами автоматического перевода и изучайте английский язык, чтобы прочитать и разобраться со

объемной справочной документацией по Roblox Studio. Вызвать эту документацию на экран вашего компьютера очень просто — достаточно в Roblox Studio нажать кнопку F1.

Загружая что-либо из Toolbox обязательно исследуйте программы, входящие в состав того, что вы выбрали, экспериментируйте и будьте бесстрашны, пробуя разные решения — без этого нельзя стать виртуозными программистами. Удивляйте всех своими программами! Делайте вещи, которые интересуют и радуют вас самих! Самое главное — получать удовольствие от своих занятий! Пусть ваше хобби станет вашей работой, когда вы станете взрослыми! Удачи!

Владимир Рубочкин, Юрий Вербиченко

Азбука программирования игр в Roblox Studio 10+

Книга 2

Ответственный за выпуск: **В. Митин**
Верстка и обложка: **СОЛОН-Пресс**

ООО «СОЛОН-Пресс»
115487, г. Москва,
пр-кт Андропова, дом 38, помещение № 8, комната № 2.
Формат 60×88/16. Объем 7,75 п. л. Тираж 1000 экз.