

**Внутренние детали движка,
или
Что скрывают от нас разработчики Construct 3**

О вещах, которые отталкивают при разработке на Construct 3.

О том, о чём не напишут разработчики в документации и гайдах.

Вся или часть предоставленной здесь информации может быть:

- неверной (автор мог сообщить неправильную информацию);
- неполной (автор мог что-то упустить);
- ложной (автор мог врать, чтобы убедить);
- необоснованной (автор не предоставил доказательств).

Если вы нашли ошибку либо имеете более достоверный источник информации - пишите на почту wilsonpercival13@gmail.com.

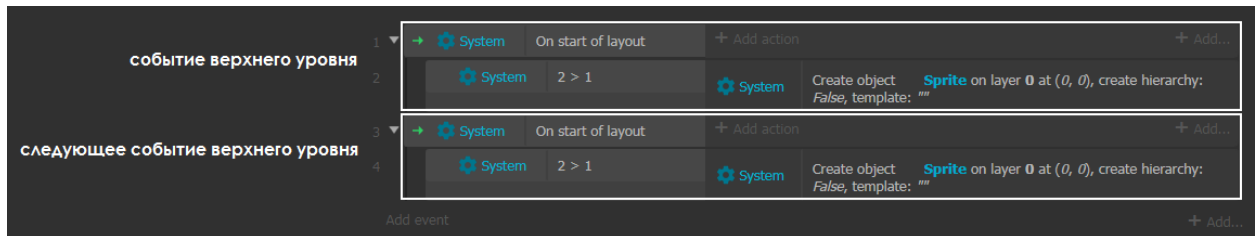
Оглавление

1. Терминология.....	4
2. Только что созданный экземпляр	5
1) Просто продублировать метод.....	7
2) Создать функцию, принимающую UID экземпляра.....	8
3) Использование блоков сценариев.	8
4) Использование ожидания.	9
5) Использование такого же события верхнего уровня.....	9
6) Запомнить соль.....	9
3. Ожидания. Полезные ожидания	11
1. Введение в полезные ожидания	11
2. Откладывает ли ожидание следующие действия на следующий тик?	11
3. Влияет ли место расположения ожидания на то, будут ли действия после него ожидать следующего тика или не будут ожидать?	13
4. Список мест, ожидающих и не ожидающих следующего тика	14
5. Ожидания, исправляющие проблемы.....	15
1) Установка скорости следующей анимации в триггере On animation finished.	15
2) Значение локальной переменной после вызова функции.....	16
6. Вызов функции при переходе на другой макет	18
4. Триггеры.....	20
1. Введение в триггеры.....	20
2. Важен ли порядок триггеров относительно других событий?	20
3. Триггер окончания твина	21
5. Ожидания. Опасные ожидания	22
1. Введение в опасные ожидания	22
2. Пример использования ожидания не по назначению при смене состояний ...	29
1) Создание локальной переменной	31
2) Блоки else	31
3) Инвертировать порядок событий	31
3. Промаргивание кадра	32
4. Экземпляры после уничтожения в файлах сценариев	32
5. Асинхронные JS-функции.....	33


1. Терминология

Соль (SOL, Selected object list) - список выбранных (пикнутых, отфильтрованных) объектов.

Событие верхнего уровня (Top-level event) – событие, которое не является под-событием. Если событие находится в группе – оно не будет считаться под-событием.



Событие верхнего уровня

Срабатывающее условие, Триггер (Trigger) – Срабатывающие условия выполняются (или запускают свои действия и тестируют под-события), когда происходит событие — например, клик мышкой. Срабатывающие условия обозначены специальным значком .

Оценка списка событий игнорирует эти события, поэтому их положение в списке событий не имеет значения (на самом деле ещё как имеет), за исключением в отношении других триггеров. Несколько идентичных триггеров (например, 5 событий **Mouse clicked (Клик мыши)**) срабатывают в порядке сверху вниз. В одном событии не может быть два триггера.

Оценивающее условие - Большинство условий являются оценивающими, это значит, что они тестируются раз в тик, когда запускается список событий. Например, **Mouse is over Sprite (Мышь наведена на Спрайт)** тестируется каждый тик. Действия в таком событии будут повторяться до тех пор, пока мышь будет на объекте, потому что каждый тик запускаются события, и условие оказывается истинным.

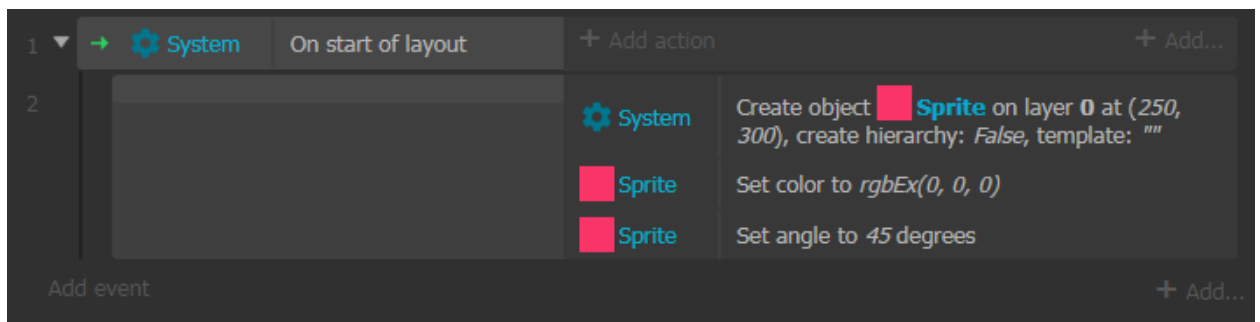
Гибрид – система написания игровой логики используя лист событий с блоками событий и блоками сценариев вперемешку.

2. Только что созданный экземпляр

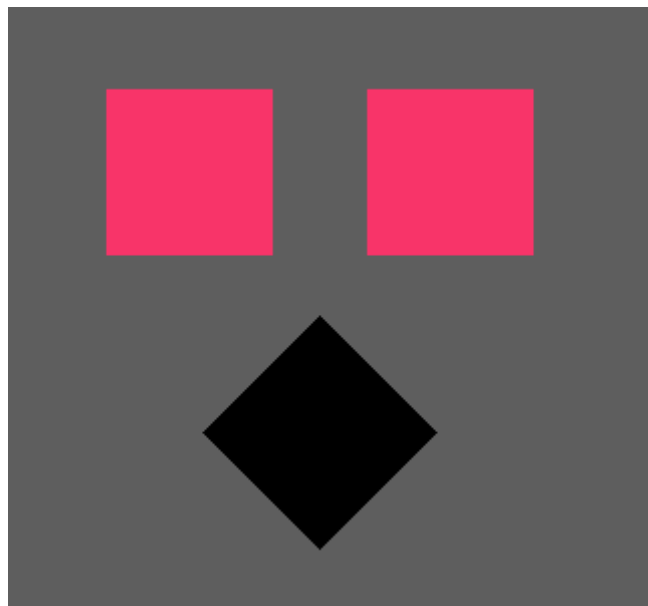
Если создать экземпляр объекта – система выберет только его. Все дальнейшие действия будут происходить только с этим экземпляром.

Есть случаи, когда нужно создать экземпляр объекта и применить какой-то метод ко всем экземплярам на макете.

Если сделать это сразу после создания экземпляра – метод отработает только на последнем созданном экземпляре.

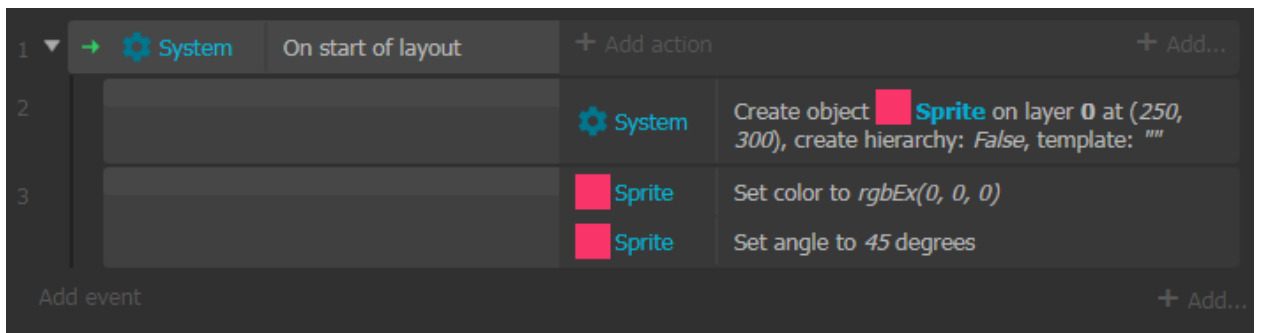


Установка метода после создания экземпляра



Экземпляры, которые уже присутствовали на макете (сверху) и только что созданный экземпляр (снизу)

Если вызвать метод параллельно блоку создания – метод вызовется у всех экземпляров, которые уже присутствовали на макете, кроме того, что был только что создан.



Установка метода в параллельном блоке



Экземпляры, которые уже присутствовали на макете (сверху) и только что созданный экземпляр (снизу)

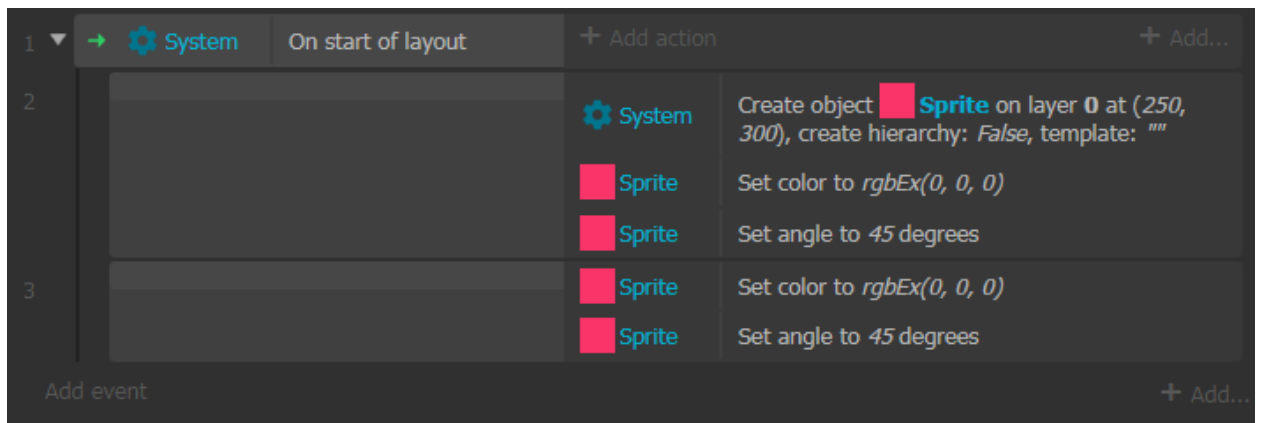
Пока не произойдёт переход на следующее событие верхнего уровня – не будет возможности обратиться к только что созданному экземпляру.

Суть задачи: применить метод ко всем экземплярам, которые уже были на макете и к тому, что только что был создан.

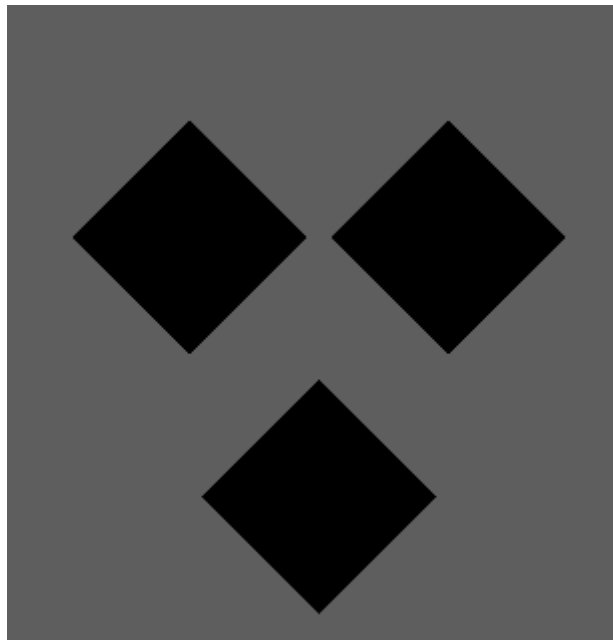
Проблема: метод срабатывает либо только на экземплярах, которые уже были на макете либо только на только что созданном экземпляре.

Пути решения:

1) Просто продублировать метод.



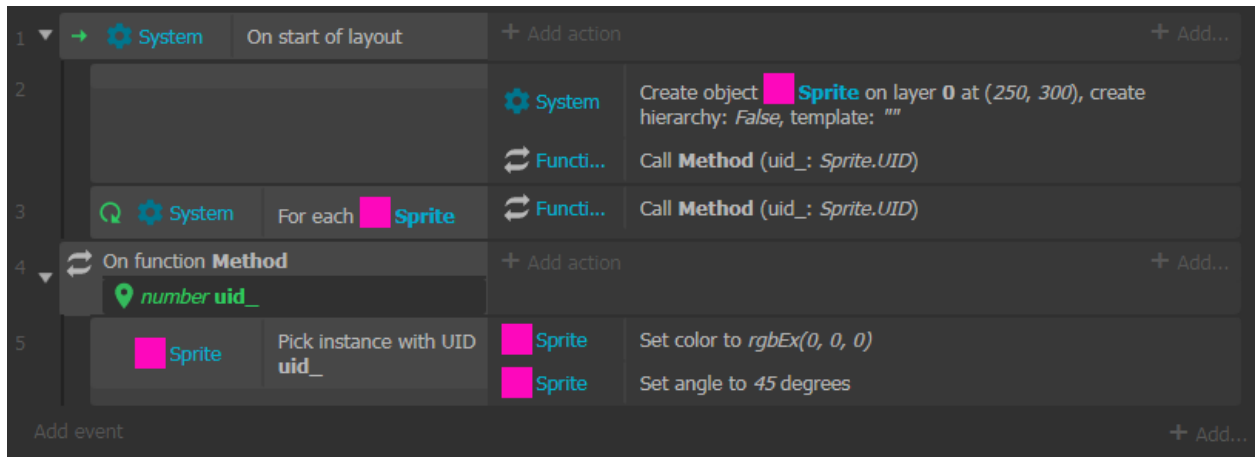
Дублирование метода



Экземпляры, которые уже присутствовали на макете (сверху) и только что созданный экземпляр (снизу)

Это решение нарушает принцип разработки программного обеспечения DRY (don't repeat yourself), поэтому настоятельно не рекомендую его использовать.

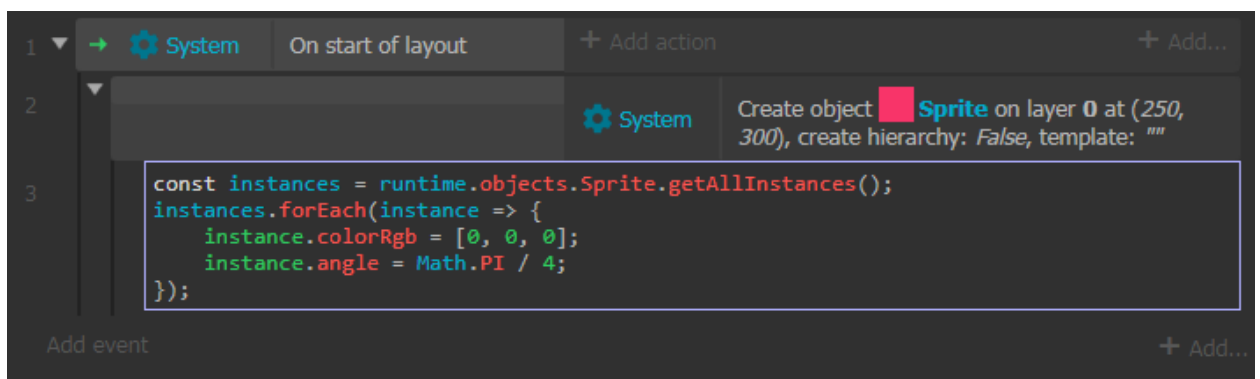
2) Создать функцию, принимающую UID экземпляра.



Функция, принимающая UID экземпляра

Это решение использует принудительный перебор всех объектов, используя цикл *For each*, что создаёт дополнительные расходы (доказательство в `benchmark_for_each.c3p`). По причине увеличения времени выполнения метода – это решение тоже не рекомендую.

3) Использование блоков сценариев.



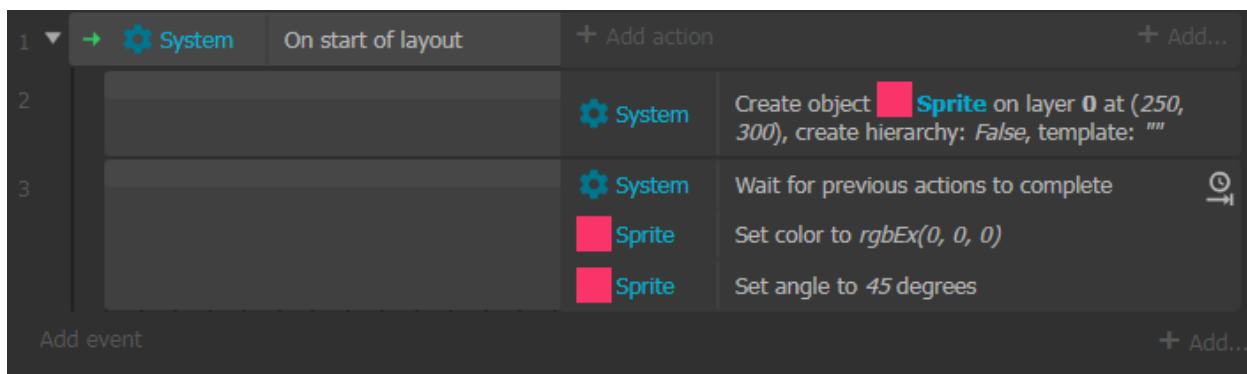
Использование блоков сценариев

До r282b сценарии работали так же само, как и события: после создания экземпляра метод `getAllInstances` возвращал все экземпляры, кроме самого последнего, который только что был создан. Я отрепортил это и на удивление это исправили:

<https://github.com/Scirra/Construct-3-bugs/issues/5439>

Я не уверен, что использование гибрида это хорошая идея. Использовать это решение или нет – оставляю на ваше усмотрение.

4) Использование ожидания.

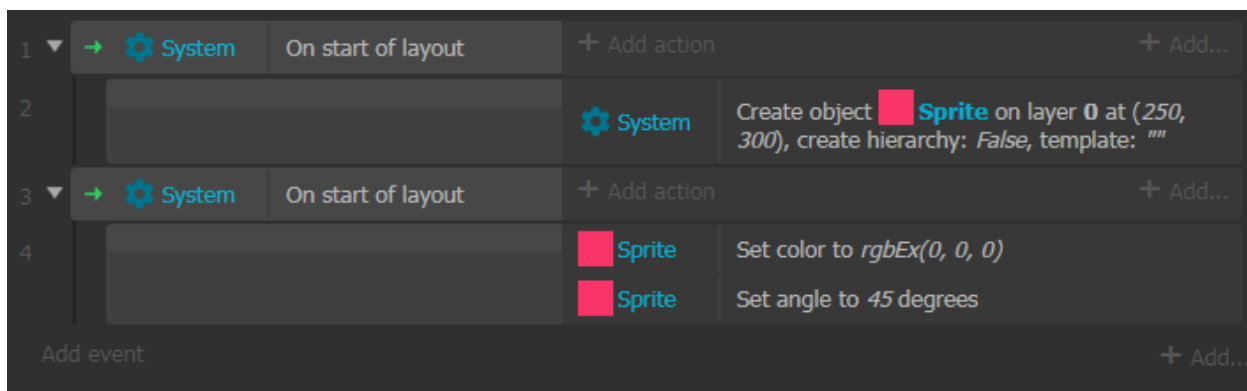


Использование Wait for previous actions

Более подробно об ожиданиях я расскажу в следующей части.

Я рекомендую это решение.

5) Использование такого же события верхнего уровня.



Дублирование события верхнего уровня

Если это решение кажется вам более подходящим, чем предыдущее – можете использовать его.

6) Запомнить соль.

Совсем недавно разработчики добавили возможность запоминать выбранные объекты когда вызывается функция. Это решение я так же рекомендую.


1	▼	→ System	On start of layout	+ Add action	+ Add...
2				System	Create object Sprite on layer 0 at (250, 300), create hierarchy: <i>False</i> , template: ""
				Funci...	Call Method
3				Funci...	Call Method
4	▼	↺	On function Method (copy picked)	+ Add action	+ Add...
5				Sprite	Set color to <i>rgbEx(0, 0, 0)</i>
				Sprite	Set angle to 45 degrees
Add event + Add...					

Copy picked

3. Ожидания. Полезные ожидания

1. Введение в полезные ожидания

Ожидание в Construct 3 является корнем всех зол. Это та штука, которая может починить, но может и поломать проект.

Изначальное предназначение: действие *Wait for previous actions* нужно ставить после асинхронных действий (напротив асинхронных действий стоит такой значок ). С этим всё просто и понятно.

Я хочу обсудить моменты, когда мы используем действие *Wait for previous actions* без асинхронных действий. Как я понял, действия, которые выполняются после *Wait for previous actions* выполняются будто в следующем событии верхнего уровня. Это объясняет, почему четвертое решение в предыдущей части работает так же как и пятое решение.

2. Откладывает ли ожидание следующие действия на следующий тик?

<https://www.construct.net/en/tutorials/system-wait-action-63>

One more trick: "Wait 0 seconds" postpones the following actions until the end of the event sheet.

Здесь написано, что действие *Wait 0 seconds* откладывает следующие действия до конца листа событий. Но это верно только для оценивающих условий.

1	System	Every tick	Browser	Log in console: "start: " & tickcount
2	Keybo...	On 1 pressed	System	Wait 0 seconds
			Browser	Log in console: "on key pressed: " & tickcount
3	Sprite	On collision with Sprite2	System	Wait 0 seconds
			Browser	Log in console: "on collision: " & tickcount
4	System	Every tick	System	Wait 0 seconds
			Browser	Log in console: "every tick: " & tickcount
5	System	Every tick	Browser	Log in console: "end: " & tickcount
Add event			+ Add...	

Действие Wait 0 seconds перед каждым действием

start: 33	runtime.js:14
end: 33	runtime.js:14
on key pressed: 34	runtime.js:14
start: 34	runtime.js:14
end: 34	runtime.js:14
start: 35	runtime.js:14
end: 35	runtime.js:14

Результат выполнения действий при нажатии клавиши

start: 27	runtime.js:14
end: 27	runtime.js:14
on collision: 28	runtime.js:14
start: 28	runtime.js:14
end: 28	runtime.js:14
start: 29	runtime.js:14
end: 29	runtime.js:14

Результат выполнения действий при столкновении

every tick: 12	runtime.js:14
start: 12	runtime.js:14
end: 12	runtime.js:14
every tick: 13	runtime.js:14
start: 13	runtime.js:14
end: 13	runtime.js:14
every tick: 14	runtime.js:14
start: 14	runtime.js:14
end: 14	runtime.js:14

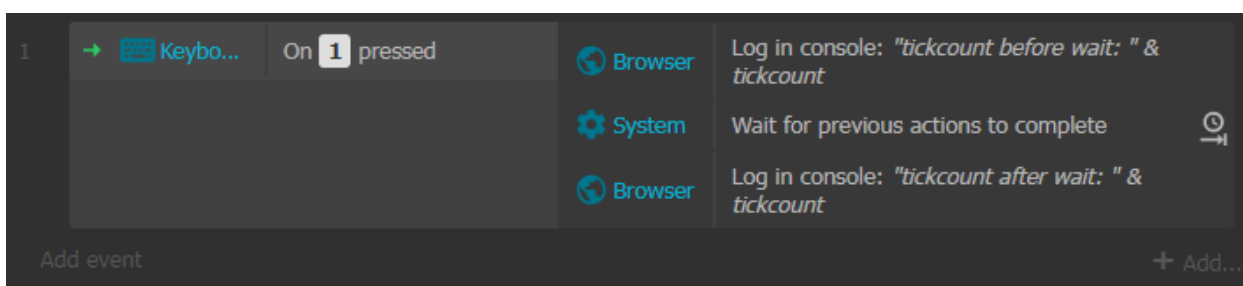
Результат выполнения действий в оценивающем условии

Видно, что если подождать после оценивающего условия – оно выполнится только на следующий тик.

Действие *Wait* со значением меньше чем $\frac{1}{60}$ секунд работает аналогично действию *Wait for previous actions*.

3. Влияет ли место расположения ожидания на то, будут ли действия после него ожидать следующего тика или не будут ожидать?

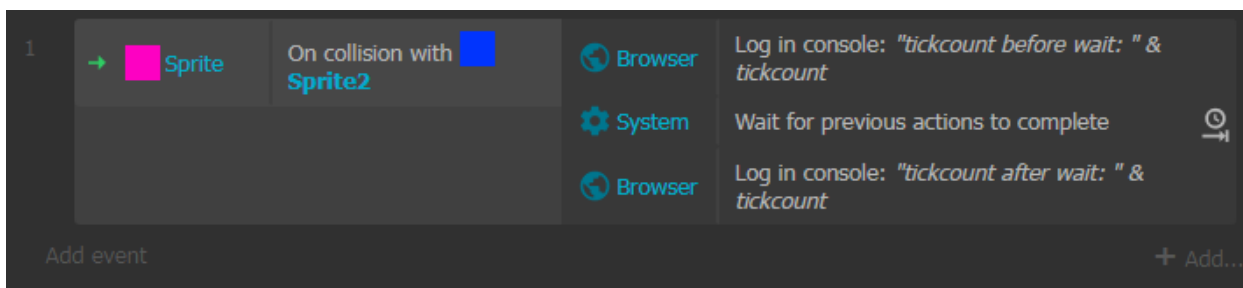
В зависимости от того, где находится действие *Wait for previous actions* – действия, которые стоят после него могут выполняться либо в этот же тик либо на следующий:



Wait for previous actions под триггером On key pressed

```
[C3 runtime] Hosted in worker, rendering with WebGL 1 [ANGLE runtime.js:22  
(Intel, Intel(R) HD Graphics Direct3D9Ex vs_3_0 ps_3_0, igdumdim64.dll)] (standard  
compositing)  
tickcount before wait: 42 runtime.js:14  
tickcount after wait: 42 runtime.js:14
```

Результат выполнения действий под триггером On key pressed



Wait for previous actions под триггером On collision

[C3 runtime] Hosted in worker, rendering with WebGL 1 [ANGLE (Intel, Intel(R) HD Graphics Direct3D9Ex vs_3_0 ps_3_0, igdumdim64.dll)] (standard compositing)	runtime.js:22
tickcount before wait: 89	runtime.js:14
tickcount after wait: 90	runtime.js:14

>

Результат выполнения действий под триггером On collision


4. Список мест, ожидающих и не ожидающих следующего тика

Неполный перечень триггеров, которые не ожидают следующего тика:

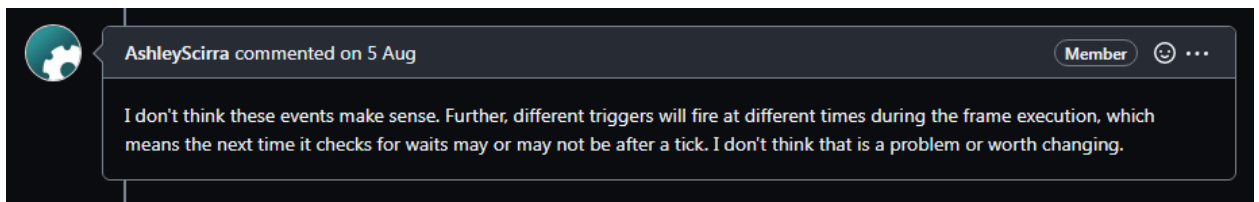
- On start of layout;
- On created;
- On key pressed;
- On ajax completed;
- On destroyed.

Неполный перечень условий, которые ожидают следующего тика (потенциально опасные места):

- On animation finished;
- On Tween finished;
- On Timer;
- On typewriter text finished;
- On collision;
- Every X seconds;
- оценивающие условия.

Every X seconds я считаю триггером, потому что вместе с ним нельзя поставить другой триггер. Перед ним нет зелёной стрелки () – мне кажется это из-за того, что начало его срабатывания зависит от начала срабатывания события, стоящего выше него. Ещё интересной особенностью является то, что если перед этим триггером нету событий – первый раз он сработает через X секунд. Но если он находится под другим событием, то первое его срабатывание произойдёт сразу же, как только событие выше станет истинным.

Баг-репорт: <https://github.com/Scirra/Construct-3-bugs/issues/5950>



ОТВЕТ [Ashley](#)

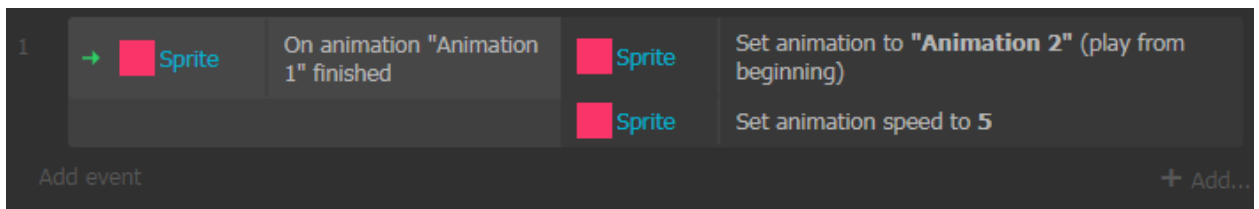
I don't think these events make sense. Further, different triggers will fire at different times during the frame execution, which means the next time it checks for waits may or may not be after a tick. I don't think that is a problem or worth changing.

5. Ожидания, исправляющие проблемы.

Ниже я приведу ещё несколько случаев, когда действие *Wait for previous actions* помогает исправить некоторые особенности Construct 3.

1) Установка скорости следующей анимации в триггере **On animation finished**.

Суть задачи: есть спрайт, у которого есть две анимации: Animation 1 и Animation 2. Обе анимации не анимации не зациклены. Animation 2 имеет скорость 0. После окончания проигрывания Animation 1 запустить Animation 2, выставив ей скорость 5.



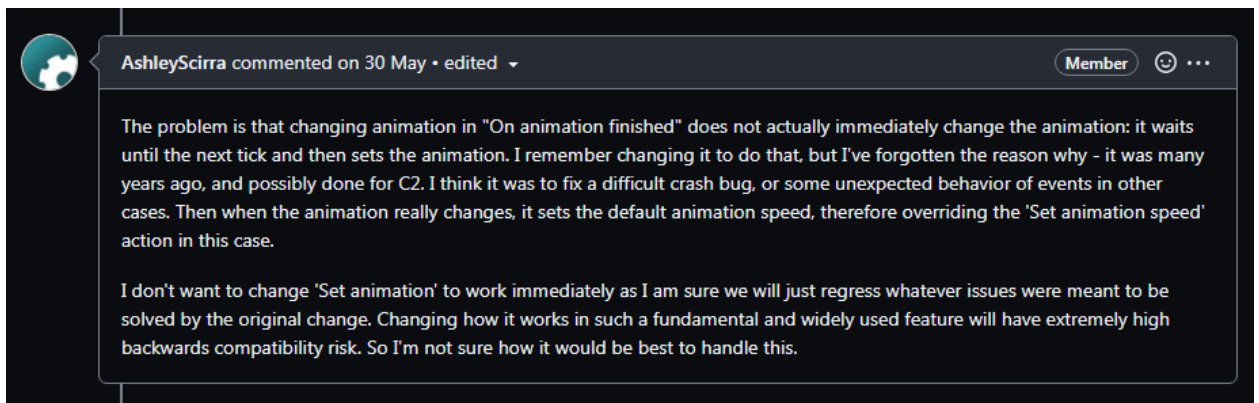
Установка скорости анимации

Проблема: видно, что проигралась Animation 1, но Animation 2 остановилась на первом кадре и не проигрывается.

Это происходит потому, что скорость Animation 2 равна 0. Действие, которое должно было установить скорость Animation 2 на 5 установило её для Animation 1. Происходит это из-за особенностей Construct 3: после окончания Animation 1 система всё ещё ссылается на анимацию, которая только что завершилась, даже если мы сменим анимацию.

Баг-репорт:

<https://github.com/Scirra/Construct-3-bugs/issues/5746>

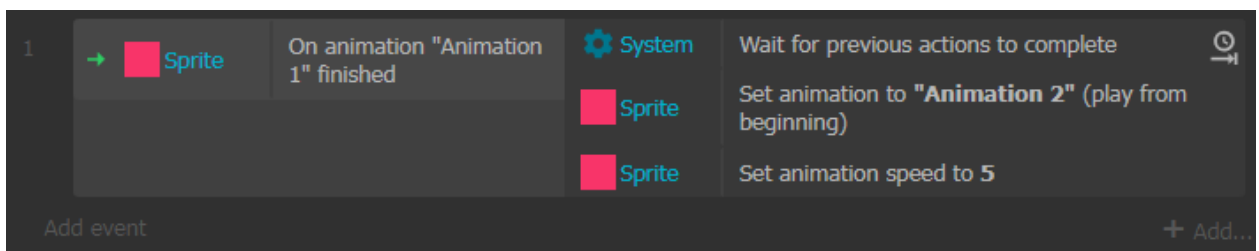


ОТБЕТ Ashley

The problem is that changing animation in "On animation finished" does not actually immediately change the animation: it waits until the next tick and then sets the animation. I remember changing it to do that, but I've forgotten the reason why - it was many years ago, and possibly done for C2. I think it was to fix a difficult crash bug, or some unexpected behavior of events in other cases. Then when the animation really changes, it sets the default animation speed, therefore overriding the 'Set animation speed' action in this case.

I don't want to change 'Set animation' to work immediately as I am sure we will just regress whatever issues were meant to be solved by the original change. Changing how it works in such a fundamental and widely used feature will have extremely high backwards compatibility risk. So I'm not sure how it would be best to handle this.

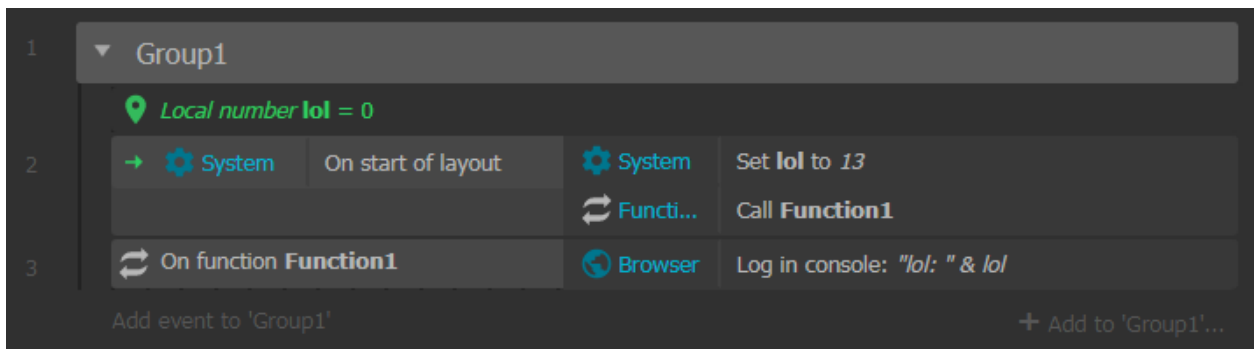
Решение: *Wait for previous actions.*



Установка скорости анимации после *Wait for previous actions*

2) Значение локальной переменной после вызова функции.

Суть задачи: есть группа, в ней локальная переменная. При старте макета мы устанавливаем ей значение, например, 13. После этого вызываем функцию, которая выводит значение переменной в консоль.



Вывод локальной переменной

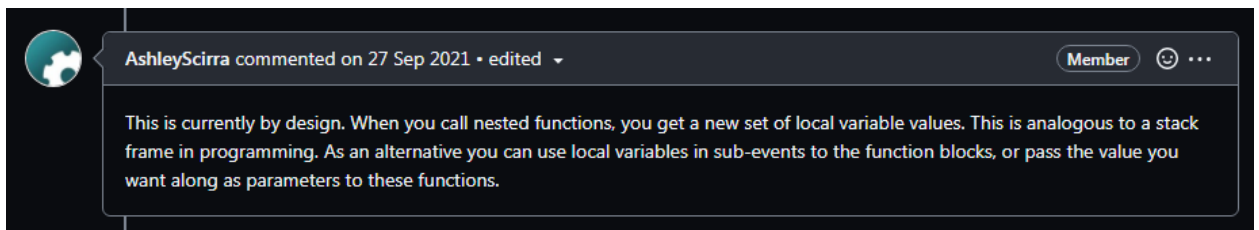
```
[C3 runtime] Hosted in worker, rendering with WebGL 1 [ANGLE runtime.js:22
(Intel, Intel(R) HD Graphics Direct3D9Ex vs_3_0 ps_3_0, igdumdim64.dll)] (standard
compositing)
lol: 0 runtime.js:14
>
```

Результат выполнения вывода локальной переменной

Проблема: мы установили значение переменной, но в консоли видим её значение по умолчанию.

Баг-репорт:

<https://github.com/Scirra/Construct-3-bugs/issues/5083>

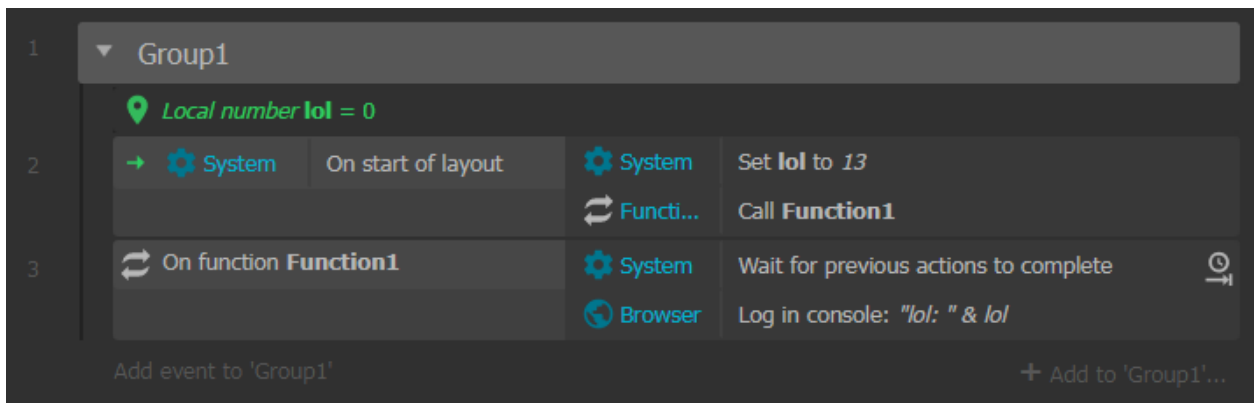


ОТВЕТ [Ashley](#)

This is currently by design. When you call nested functions, you get a new set of local variable values. This is analogous to a stack frame in programming. As an alternative you can use local variables in sub-events to the function blocks, or pass the value you want along as parameters to these functions.

Пути решения:

a) *Wait for previous actions.*

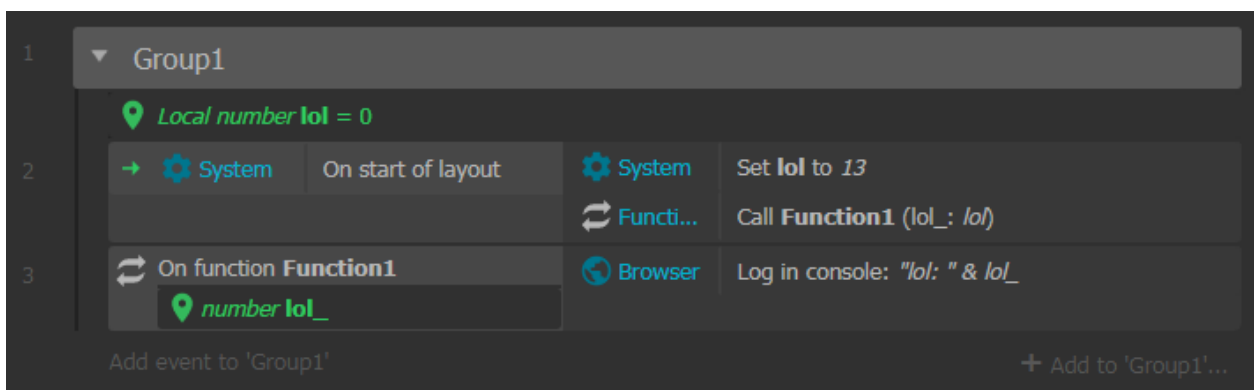


Вывод локальной переменной после *Wait for previous actions*

```
[C3 runtime] Hosted in worker, rendering with WebGL 1 [ANGLE runtime.js:22
(Intel, Intel(R) HD Graphics Direct3D9Ex vs_3_0 ps_3_0, igdumdim64.dll)] (standard
compositing)
lol: 13 runtime.js:14
```

Результат вывода локальной переменной после *Wait for previous actions*

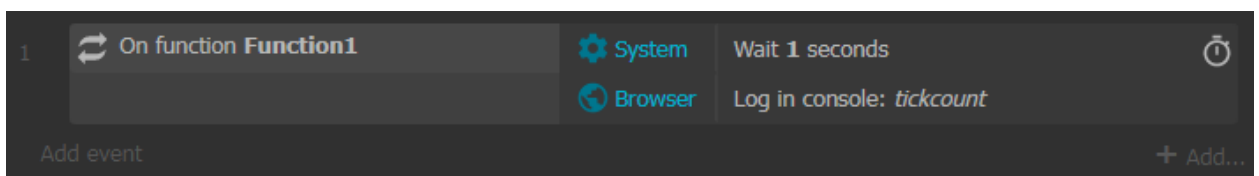
b) Решение, предложенное [Ahsley](#).



Передача переменной в параметрах функции

6. Вызов функции при переходе на другой макет

Если мы создадим функцию, внутри которой поставим ожидание, например, на одну секунду и вызовем её на одном макете, то при переходе на второй макет функция не выполнит действия, которые находились после ожидания.



Функция, чьи действия не выполнятся если перейти на следующий макет
после её вызова

Баг-репорт: <https://github.com/Scirra/Construct-3-bugs/issues/5775>

4. Триггеры

1. Введение в триггеры.

Триггеры можно поделить на две категории:

1. тип ожидания;
2. тип порядка.

По типу ожидания триггеры делятся на:

1. внутри которых действия после *Wait for previous actions* ожидают следующего тика;
2. внутри которых действия после *Wait for previous actions* не ожидают следующего тика.

По порядку в листе событий делятся на:

1. соблюдающие свой порядок в листе событий;
2. не соблюдающие свой порядок в листе событий.

В свою очередь, триггеры, внутри которых действия после *Wait for previous actions* ожидают следующего тика также делятся на:

1. выполняющиеся в начале списка событий;
2. выполняющиеся в конце списка событий.




Если триггер, который не соблюдает свой порядок в листе событий будет находиться под-событием под оценивающим условием – он всё равно выполнится в начале листа событий, если оценивающее условие, находящееся перед ним будет истинно.

2. Важен ли порядок триггеров относительно других событий?

Я нашёл противоречие в документации:

<https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/how-events-work>

However, triggers are an exception. See the green arrow to the left of Keyboard: On Space pressed from the previous example:

→  Keyboard	On Space pressed	 Monster	Destroy
 Player	Is PowerupEnabled	Add action	

This indicates the event is triggered. Rather than running once per tick, this event simply runs (or "fires") upon something actually happening. In this case, the event runs when the user hits the Spacebar key on the keyboard. It is never checked any other time. Since triggers run upon an event happening, they aren't

checked in top-to-bottom order like other events. This means the ordering of triggers relative to other events is not important (except relative to other triggers of the same type, since triggers still fire top-to-bottom).

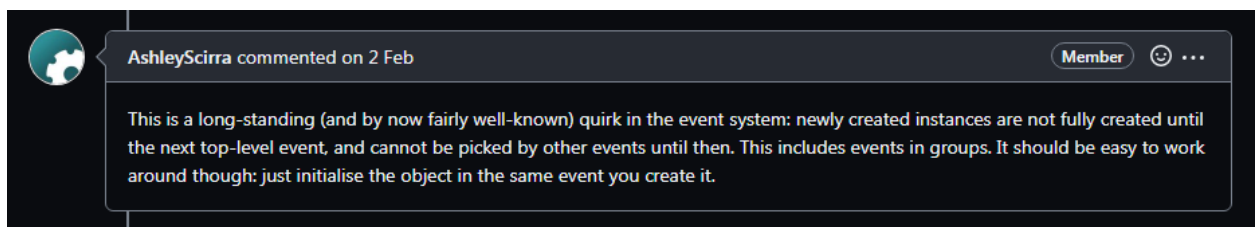
Тут сказано, что триггеры, помеченные зелёной стрелкой не учитывают порядок своего расположения в списке событий, но это не так: триггер On collision тому подтверждение.

3. Триггер окончания твина

В отличие от остальных, триггер окончания воспроизведения твин-анимации выполняется в конце списка событий. На моей памяти это единственный триггер с таким поведением. Если использовать ожидание в этом триггере, то действия, следующие за ожиданием выполнятся на следующий тик, что нивелирует проблему промаргивания кадра.

Баг-репорт: <https://github.com/Scirra/Construct-3-bugs/issues/5435>

Ashley ошибочно предположил, что это связано с тем, что к нужному моменту у меня нет доступа к экземпляру объекта до следующего события верхнего уровня, сославшись на мой другой баг-репорт <https://github.com/Scirra/Construct-3-bugs/issues/5434>.



Ответ Ashley

This is a long-standing (and by now fairly well-known) quirk in the event system: newly created instances are not fully created until the next top-level event, and cannot be picked by other events until then. This includes events in groups. It should be easy to work around though: just initialise the object in the same event you create it.

Хотя дело не в этом, у меня есть доступ к экземплярам в проекте, который я прикрепил в одном из обсуждений этого же бага: <https://github.com/Scirra/Construct-3-bugs/issues/5947>

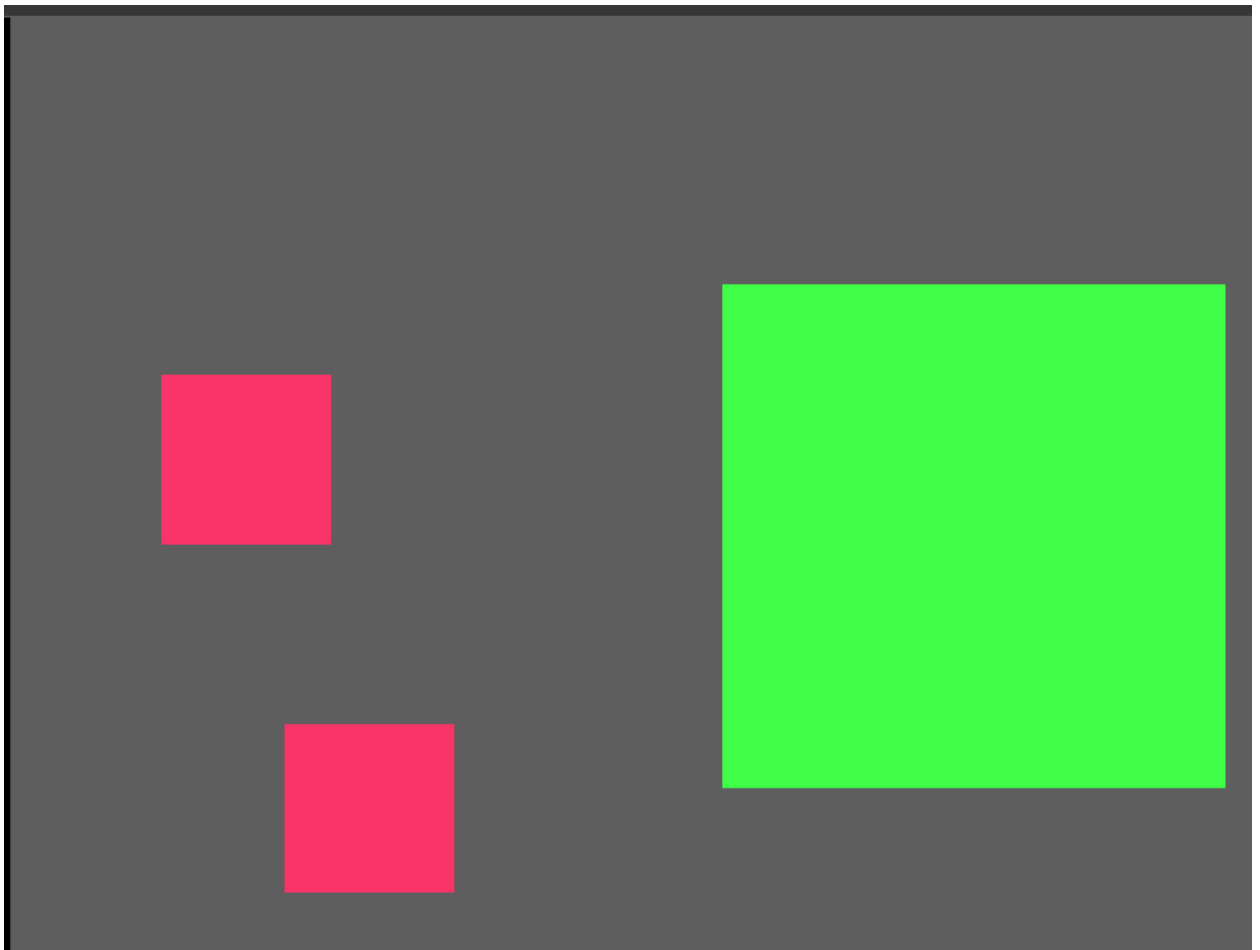
5. Ожидания. Опасные ожидания

1. Введение в опасные ожидания

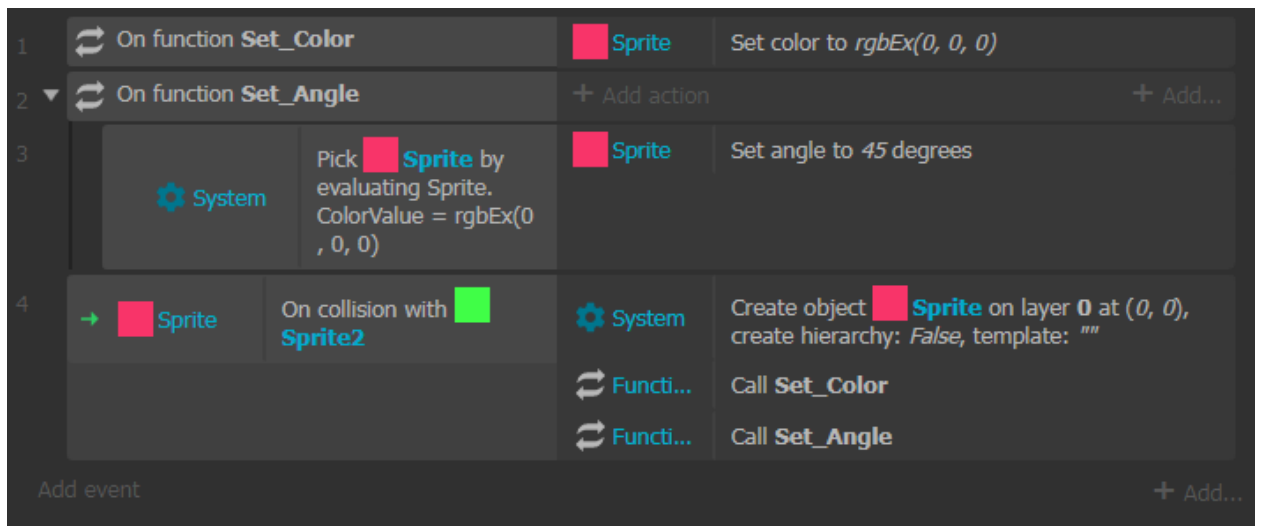
Ожидания можно использовать безболезненно, когда действия, стоящие после них выполняются в этот же тик. Когда же действия выполняются на следующий тик – случаются ужасные вещи. Рассмотрим одну из ситуаций.

Суть задачи: есть спрайт. Когда он сталкивается – вызываем две функции:

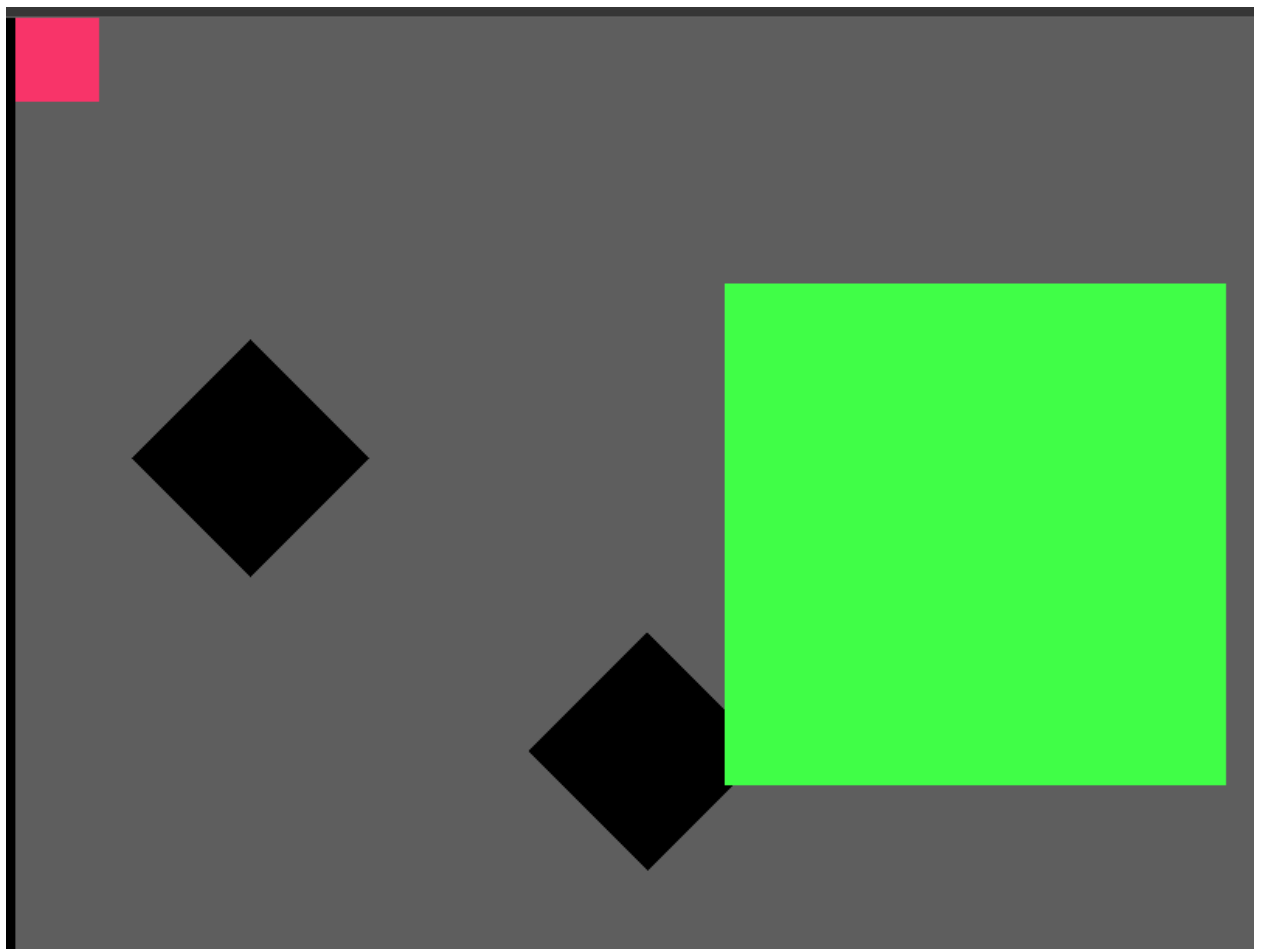
1. создать спрайт в координатах (0, 0);
2. покрасить все спрайты в чёрный цвет;
3. повернуть все спрайты, которые покрашены в чёрный цвет на 45 градусов.



Макет до коллизии

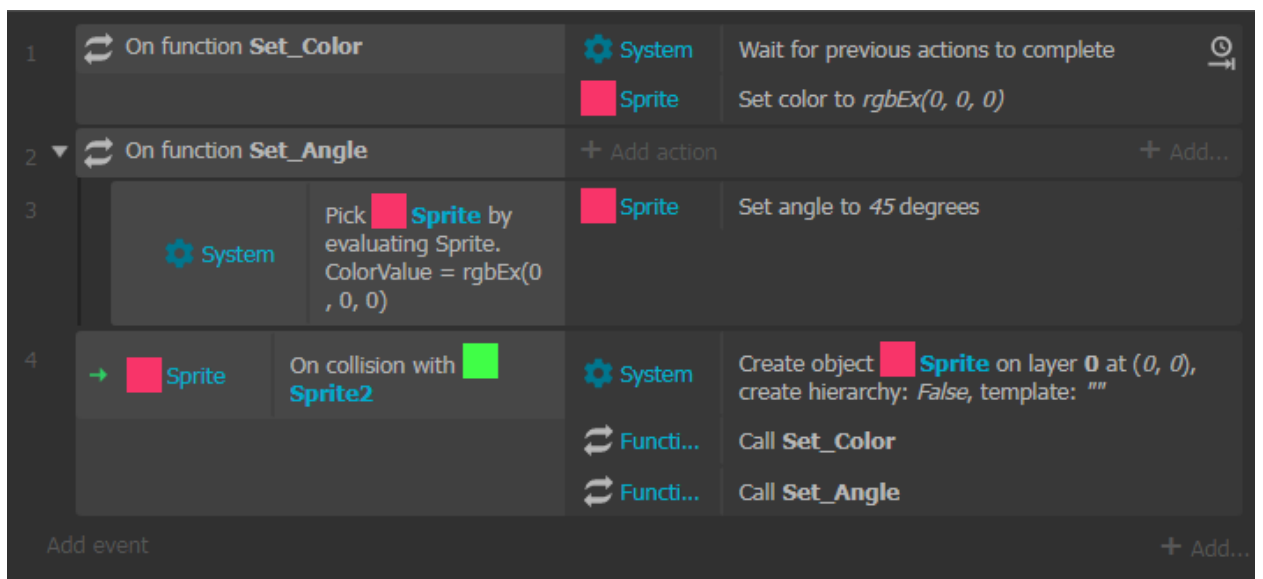


Вызов функций `Set_Color` и `Set_Angle` после создания спрайта

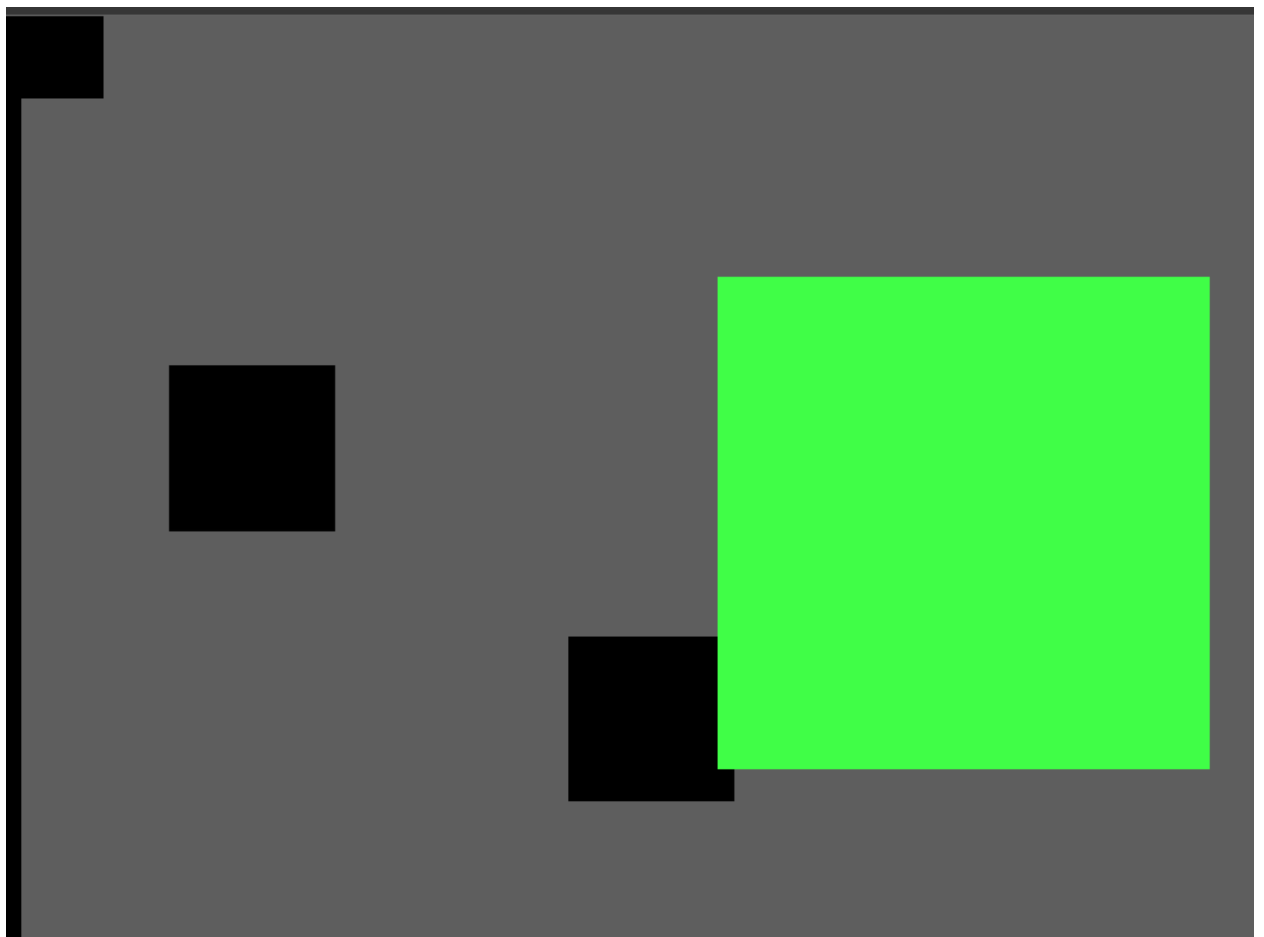


Макет после коллизии

Видно, что после коллизии создавался спрайт в углу, но почернели и изменили угол только те спрайты, которые были на макете до коллизии. Это произошло потому, что вызов функции `Set_Color` происходил в том же событии верхнего уровня, что и создание нового спрайта. Исправим это, добавив задержку в функции `Set_Color`:



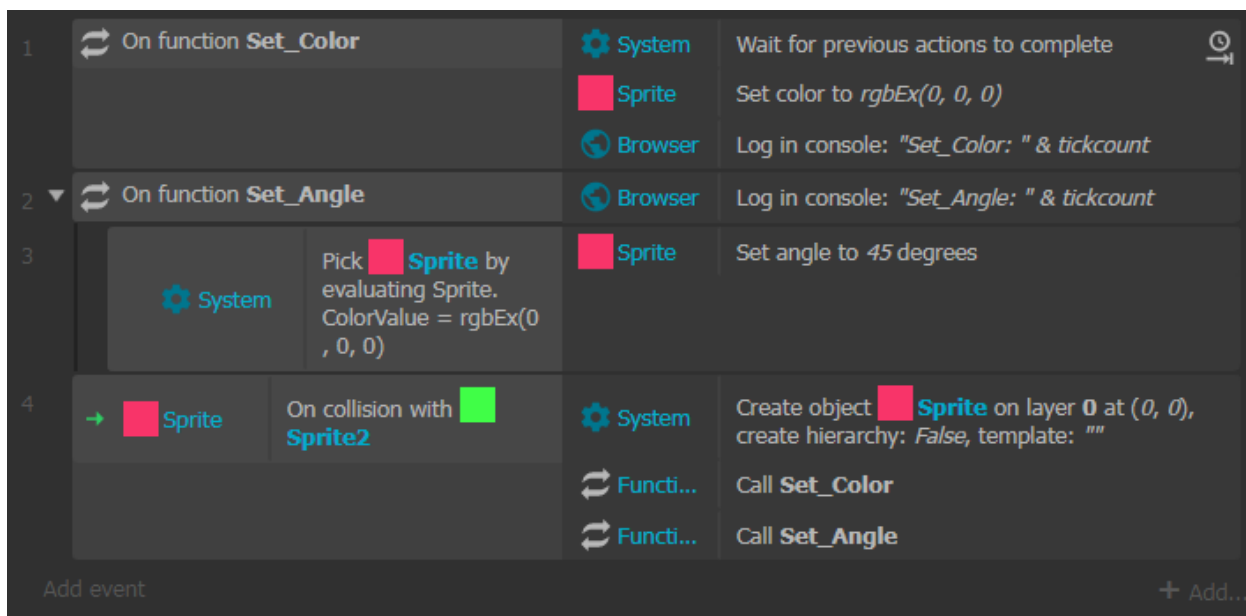
Добавление *Wait for previous actions* в функцию **Set_Color**



Макет после коллизии

Видно, что после коллизии создавался спрайт в углу и все спрайты покрасились в чёрный цвет. Но теперь ни один спрайт не повернут. Это произошло потому, что действия,

стоящие после *Wait for previous actions* в функции `Set_Color` выполнялись на следующий тик. Это произошло потому, что функция `Set_Color` вызвана под триггером `On collision`, внутри которого действия, стоящие после *Wait for previous actions* выполняются на следующий тик. Мы можем убедиться в этом, если выведем в консоль количество тиков.

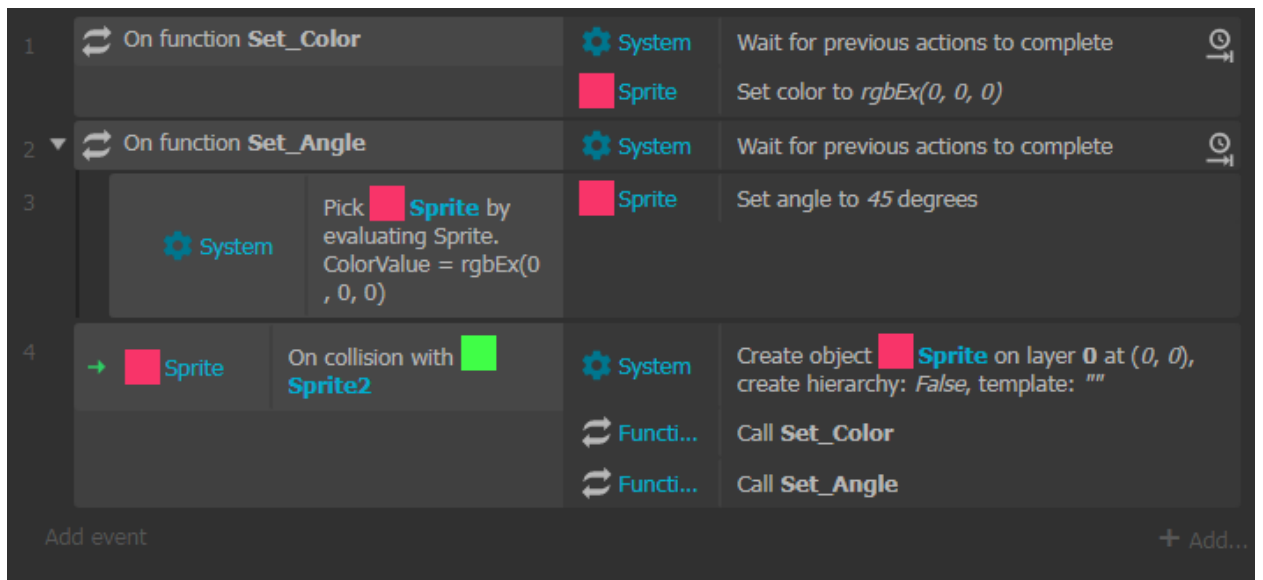


Выводим количество тиков в консоль

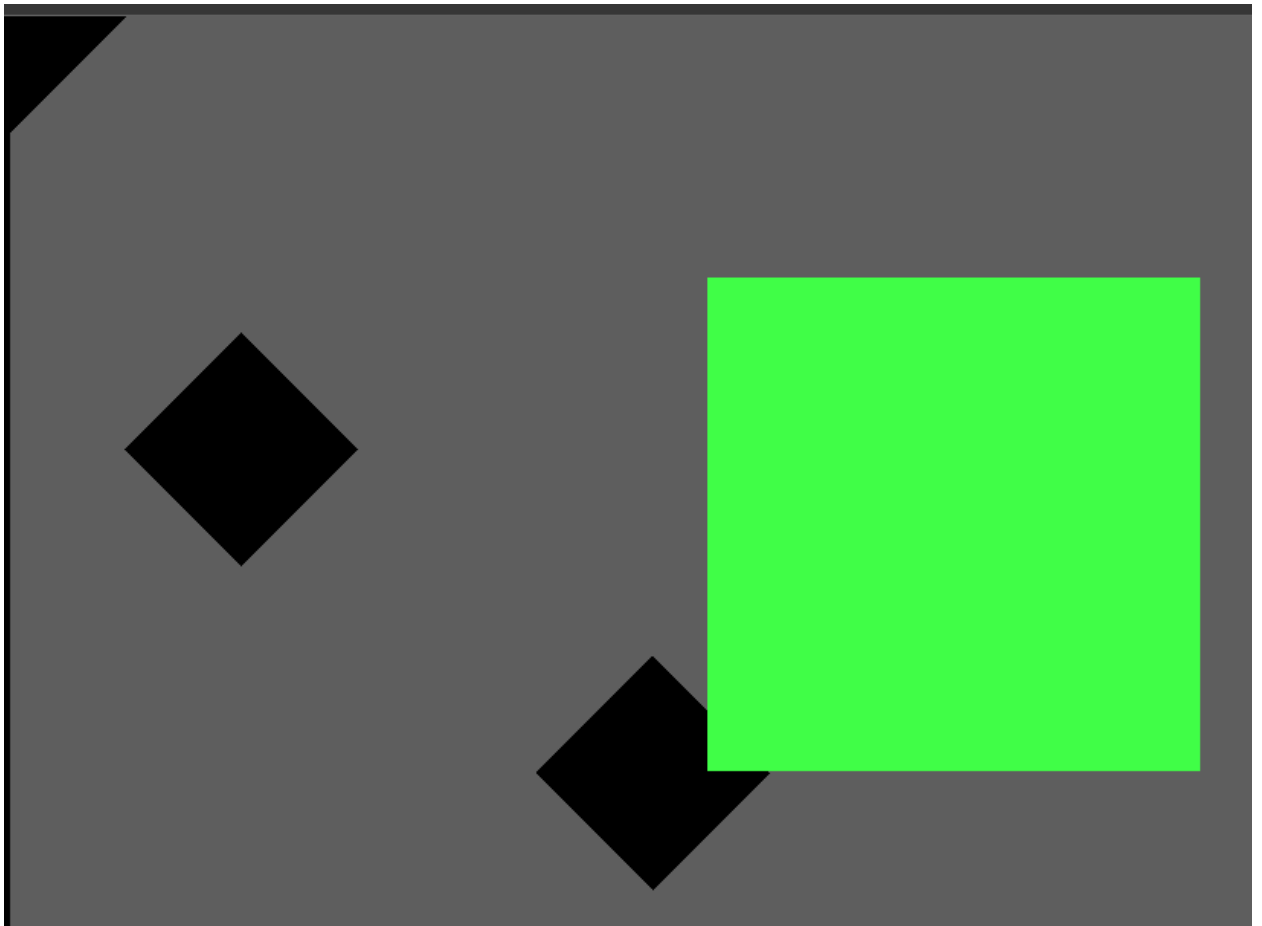
```
[C3 runtime] Hosted in worker, rendering with WebGL 1 [ANGLE runtime.js:22
(Intel, Intel(R) HD Graphics Direct3D9Ex vs_3_0 ps_3_0, igdumdim64.dll)] (standard
compositing)
Set_Angle: 56 runtime.js:14
Set_Color: 57 runtime.js:14
>
```

Функции и номер тика, в котором они вызвались

Видно, что функция `Set_Angle` вызвалась на один тик раньше, чем функция `Set_Color`. И вот тут-то и кроется зло. Большинство разработчиков просто установит ещё одно ожидание в функцию `Set_Angle`.



Установка второго *Wait for previous actions* в проекте



Макет после коллизии

После того как разработчик установил второй *Wait for previous actions* – где-то в проекте сломалось ещё что-то. Он долго ищет причину, не может её найти и не

придумывает ничего лучше, как добавить третий *Wait for previous actions*. Ну а дальше как снежный ком – чем больше растёт проект, тем больше ожиданий в нём появляется.

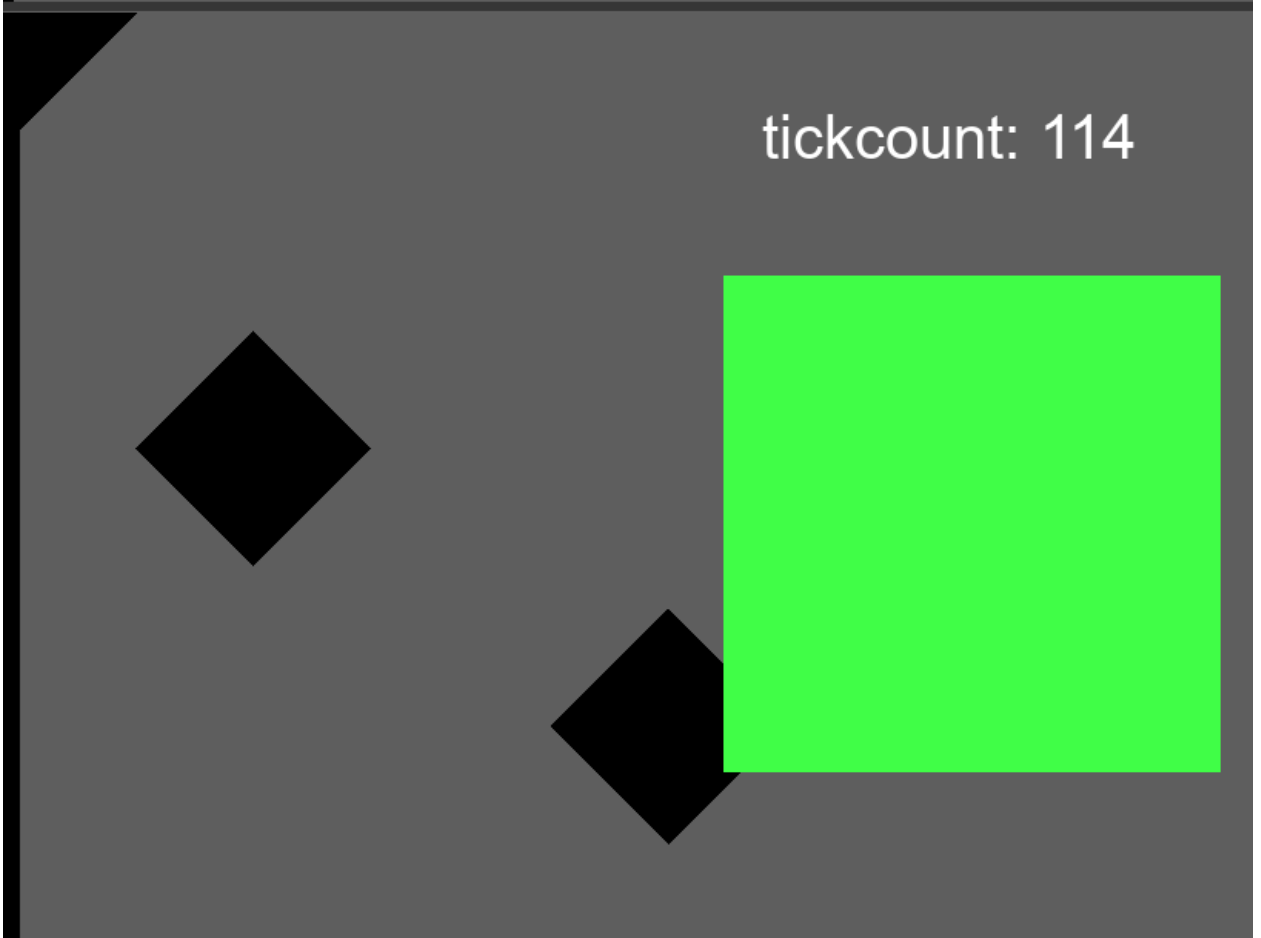
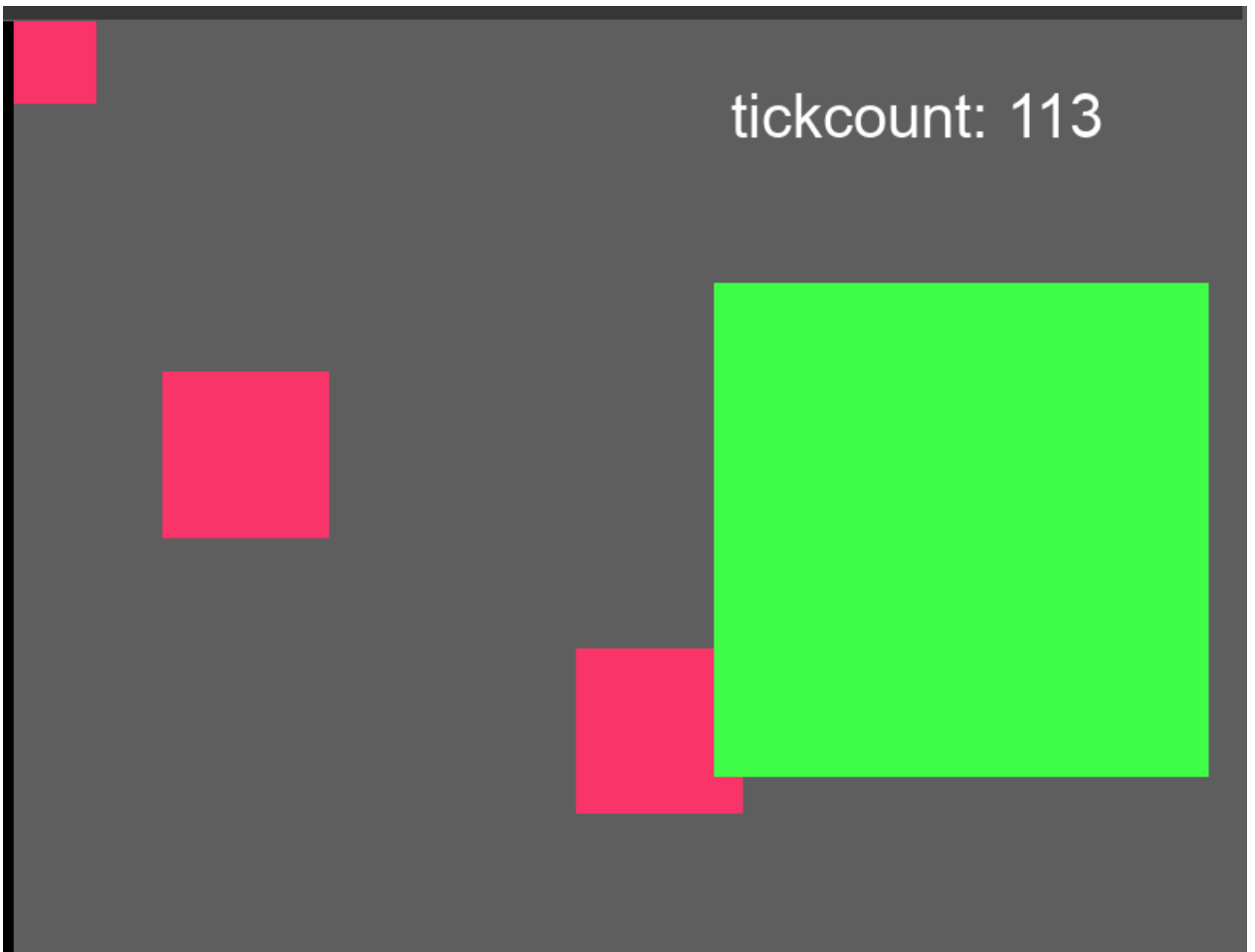
Решение: **а здесь нет нормального решения**. Если мы установим *Wait for previous actions* только после создания спрайта – спрайт создастся в текущем тике, а цвет поменяет и повернётся только на следующий. Из-за этого мы увидим неприятное мерцание. Мы должны были увидеть новый спрайт сразу чёрным и повернутым, но мы видим его сначала с настройками по умолчанию, а в следующем кадре уже чёрным и повернутым.

The screenshot shows the Godot engine's Event Editor with four events:

- Event 1:** On function *Set_Color* (Sprite) | Set color to *rgbEx(0, 0, 0)*
- Event 2:** On function *Set_Angle* (Sprite) | + Add action
- Event 3:** System | Pick *Sprite* by evaluating *Sprite*. *ColorValue = rgbEx(0, 0, 0)* | Sprite | Set angle to 45 degrees
- Event 4:** Sprite | On collision with *Sprite2* | System | Create object *Sprite* on layer 0 at (0, 0), create hierarchy: *False*, template: ""
System | Wait for previous actions to complete
Function | Call *Set_Color*
Function | Call *Set_Angle*

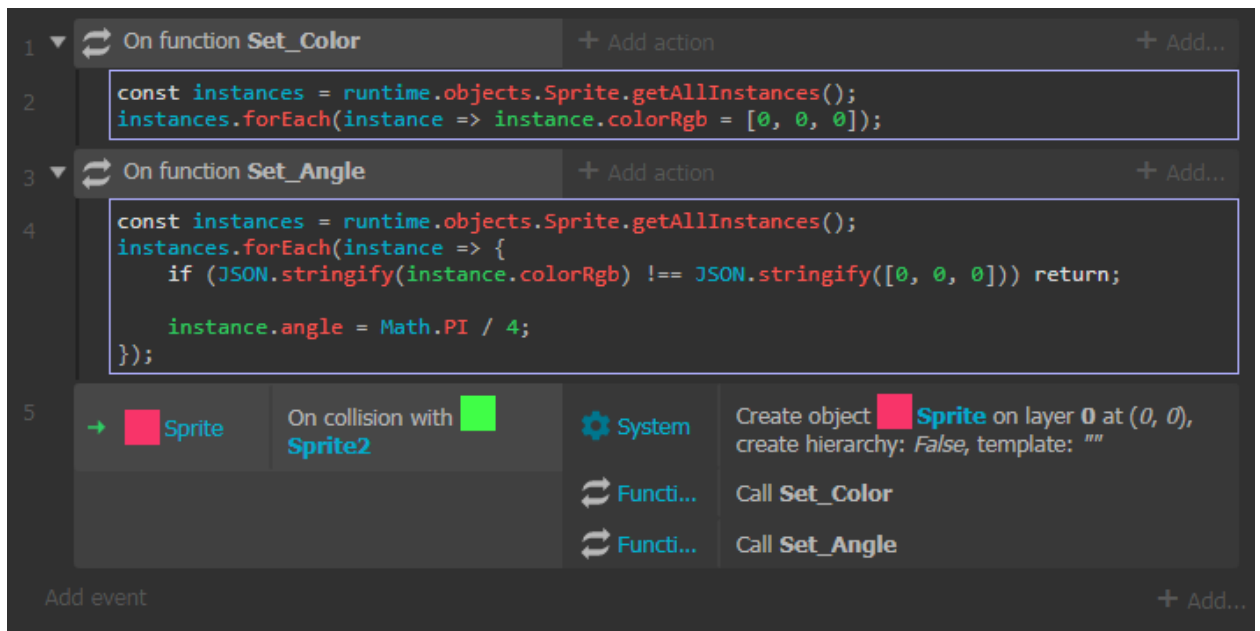
At the bottom, there is an "Add event" button and a "+ Add..." button.

Установка *Wait for previous actions* только после создания экземпляра



Появление экземпляра на макете в два кадра

Можно было бы использовать блоки сценариев – с ними вообще не нужны ожидания. Но я не считаю, что использовать гибриды это хорошее решение. Если пишете сценарии – пишите их в файлах сценариев.

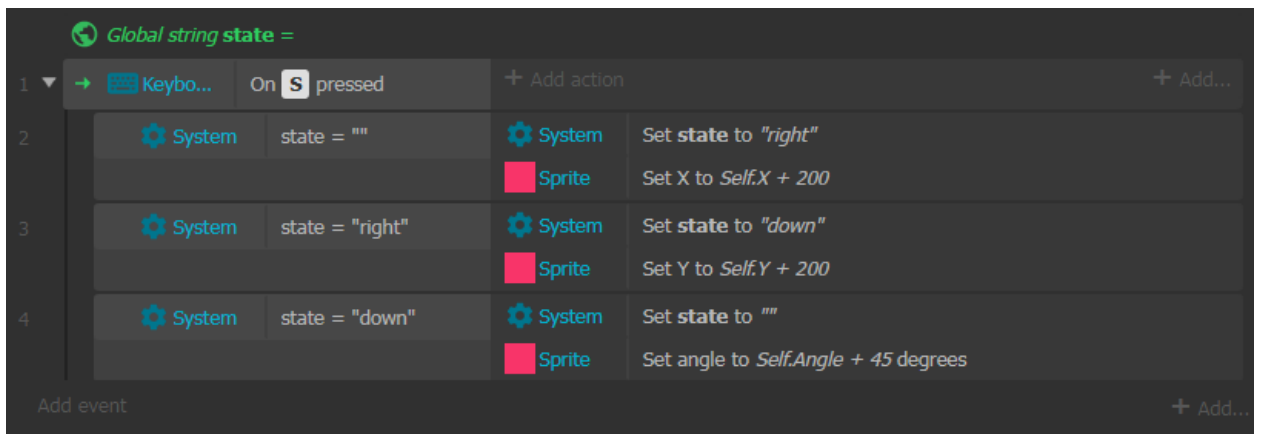


Блоки сценариев

Вы можете безопасно использовать ожидания, когда действия после него выполняются в этот же тик. Вы можете использовать ожидания по дизайну, когда следует явно подождать какое-то время. Использовать ожидание во всех остальных случаях я не рекомендую, потому что это может сломать ваш проект.

2. Пример использования ожидания не по назначению при смене состояний

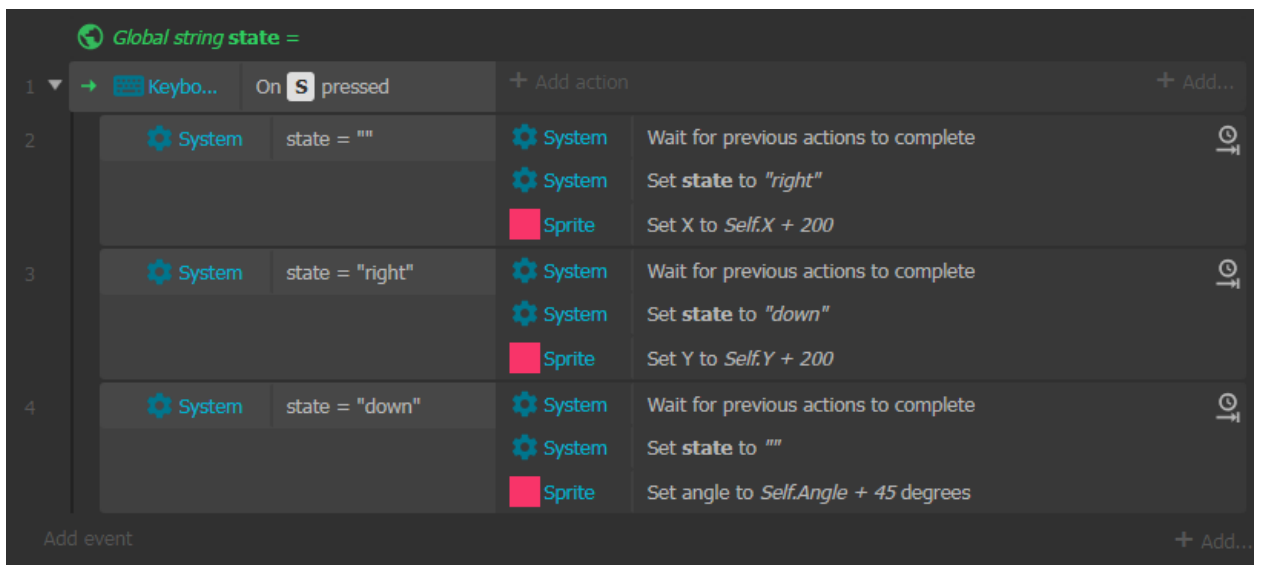
Рассмотрим случай, когда при нажатии кнопки нам нужно выполнить какое-то действие и переключиться на другое состояние.



Переключение состояний по нажатию кнопки

Если запустить проект и нажать на кнопку – за одно нажатие выполнятся сразу все действия. Это произойдёт потому, что во втором событии мы поменяли состояние и на проверке следующего события оно оказывается истинным, поэтому выполняет и его.

Чаще всего неопытные разработчики просто добавляют ожидание:



Добавление ожидания перед каждым действием

Это работает, потому что действие `Wait for previous actions to complete` откладывает выполнение всех действий в самый конец этого события: при нажатии клавиши у нас срабатывает условие во втором событии и действия откладываются к концу. Третье и четвёртое событие оказываются ложными и игнорируются. Затем выполнение программы возвращается к отложенным действиям в первом событии и выполняет их.

Какое решение лучше в этом моменте? У меня есть несколько вариантов.

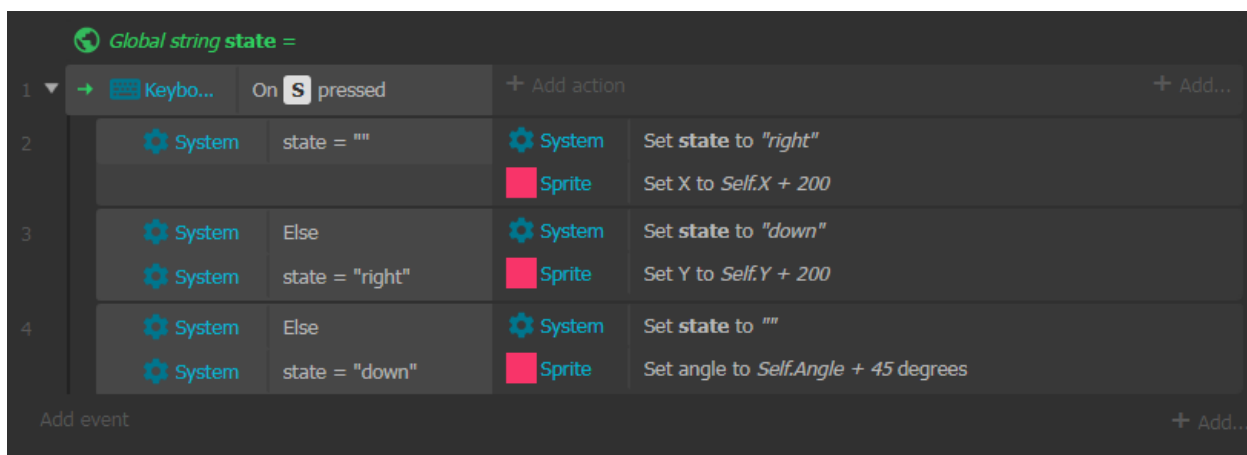
1) Создание локальной переменной



Локальная переменная

Мне больше всего нравится вариант, когда мы создаём локальную переменную, в которую записываем значение переменной состояния, а проверяем уже по значению локальной переменной. Тогда при изменении переменной состояния всё будет работать корректно.

2) Блоки else



Блоки else

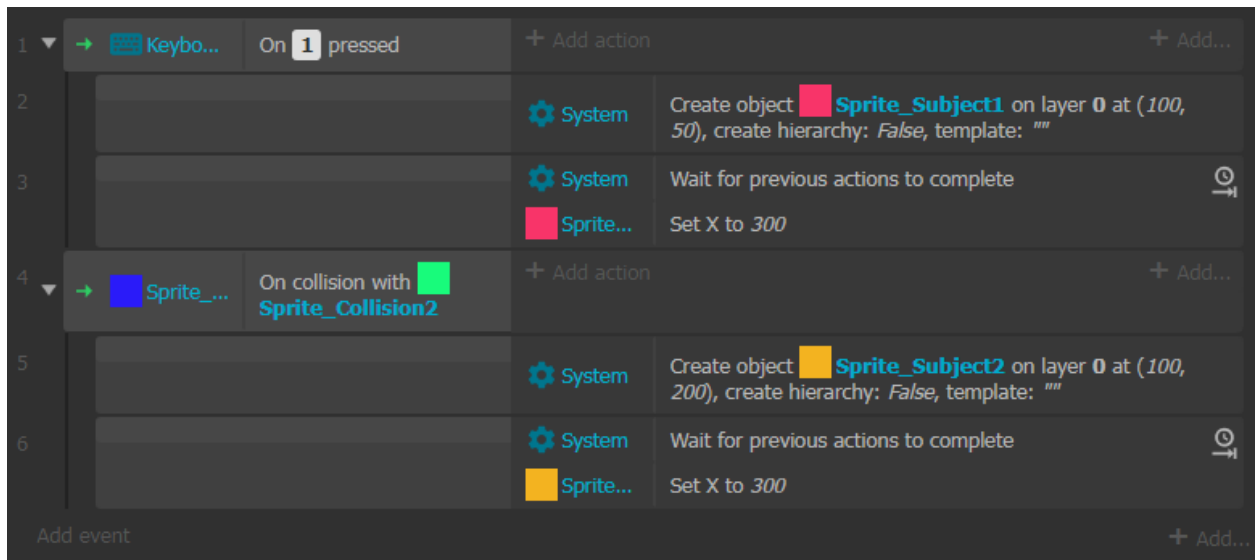
Есть ещё вариант с блоками `else`, но мне он не очень нравится, т.к. при увеличении количества состояний придётся увеличивать количество блоков `else`.

3) Инвертировать порядок событий

Ещё можно инвертировать порядок событий, но я даже не буду его рассматривать, потому что это самое ужасное решение и в некоторых нелинейных случаях переключения состояний может работать всё ещё плохо. В добавок это лишняя работа, чтобы следить за нужным порядком расположения условий – это не то, что должно заботить разработчика.

3. Промаргивание кадра

Проблему откладывания действий на следующий тик можно легко заметить.



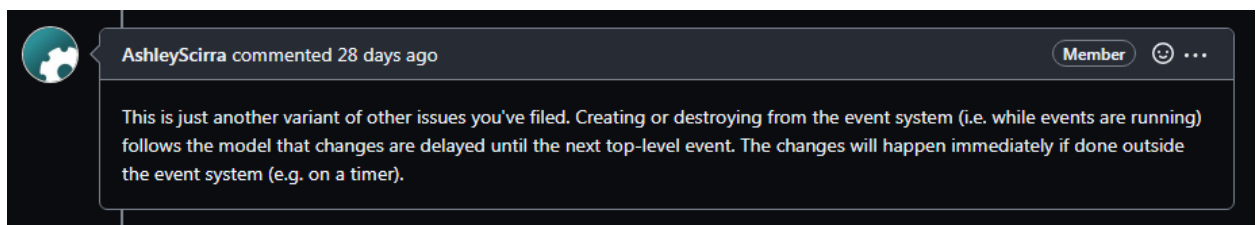
Промаргивание кадра

Нужно просто создать объект на одних координатах и после ожидания сдвинуть объект на другие координаты. Тогда, при соприкосновении мы заметим на один кадр спрайт на начальных координатах. Это происходит из-за того, что действие по перемещению выполняется на следующий кадр. Всё работает хорошо, если мы делаем точно такие же события под нажатием клавиши.

4. Экземпляры после уничтожения в файлах сценариев

Есть ещё одна проблема, связанная со сценариями. Как мы знаем, если удалить экземпляр объекта через события – он не удалится до начала следующего события верхнего уровня. Тоже самое произойдёт, если мы вызовем код, который удалит экземпляр и запросит все существующие экземпляры на макете – мы получим их все. Эта проблема так же решается ожиданием.

Баг-репорт: <https://github.com/Scirra/Construct-3-bugs/issues/6004>



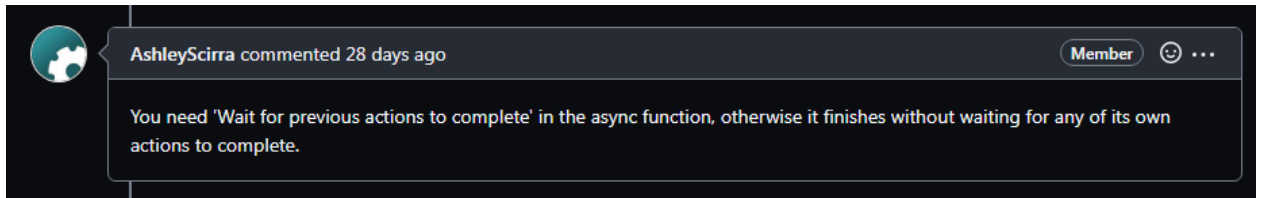
ОТВЕТ Ashley

This is just another variant of other issues you've filed. Creating or destroying from the event system (i.e. while events are running) follows the model that changes are delayed until the next top-level event. The changes will happen immediately if done outside the event system (e.g. on a timer).

5. Асинхронные JS-функции

Ещё один неочевидный момент, связанный с использованием ключевого слова `await` и асинхронных функций. Если мы не установим ожидание в конце асинхронной функции – `await` не будет ждать её завершения.

Баг-репорт: <https://github.com/Scirra/Construct-3-bugs/issues/6022>



ОТВЕТ [Ashley](#)

You need 'Wait for previous actions to complete' in the async function, otherwise it finishes without waiting for any of its own actions to complete.

