

# Today

## ➊ Finish Animation Demo

Less tippy, guided drops.

## ➋ Autolayout

How to make device autorotation easy(er).

And make your View Controller work in different environments (i.e. with different bounds).

## ➌ Autolayout Demo

Making Attributor autorotate properly.

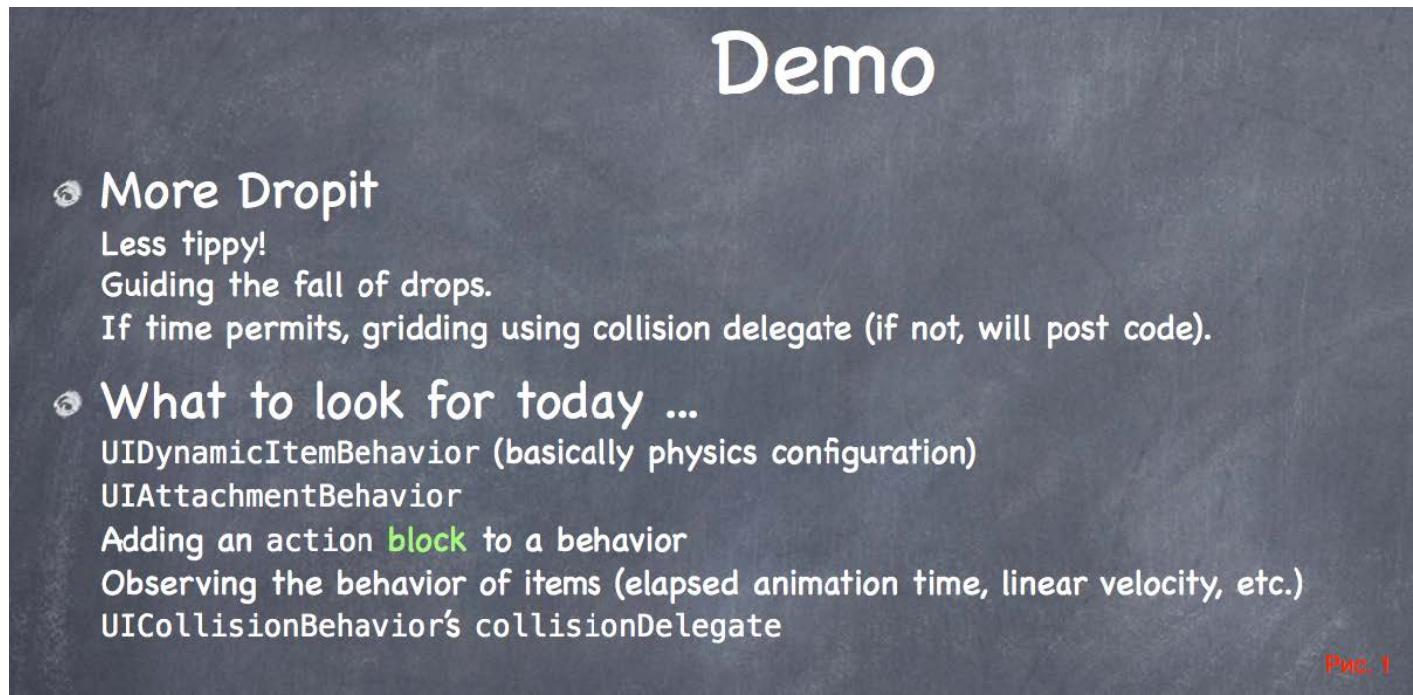
Сегодня мы закончим демонстрационный пример с анимацией, который мы начали в прошлый раз, а затем поговорим об Авторазметке (Autolayout). Это отличная вещь для адаптации пользовательского интерфейса (UI) к изменяющимся условиям. В конце лекции мы рассмотрим демонстрационный пример, в котором вы сможете посмотреть Autolayout в действии. Вернемся к демонстрационному примеру, с падающие вниз квадратами. Напомню - они у нас падают под действием “гравитации” и силы столкновений внизу экрана, которое мы оставили незаконченным в прошлый раз. Вы помните, что как только полностью заполняется строка, то квадраты “улетучивались”, поднимаясь вверх и исчезая за пределами экрана. Но это не всегда происходило и мы застряли в прошлый раз на том, что квадраты не “улетали”.

Часть причины этого состоит в том, что башни из квадратов были очень не устойчивы и продолжали немного шевелиться, никогда не успокаиваясь, поэтому анимация никогда не останавливалась. Эти неустойчивые “башни” продолжали немного перемещаться. И сегодня мы продолжим наше демо с того, что сделаем наши груды квадратов немного более жесткими, менее неустойчивыми. Мы достигнем этого тем, что не позволим квадратам вращаться.

Так как они не могут вращаться, то будут сидеть друг на друге более правильно, поэтому мы вернемся к значению ускорения “гравитации” =1.0, потому что в некоторой точке мы задали 0.9. Квадраты будут строго садится на вершины нижних.

Еще мы будем использовать `UIAttachmentBehavior` так, что когда квадрат падает, мы будем его захватывать, раскачивать и выбирать, в какой стэк его направить, и только затем отпустить его падать.

И последняя вещь, которую мы реализуем - это использование блока `action` для поведения `UIAttachmentBehavior`, чтобы показать “присоединение” во время анимации. Для этого мы будем рисовать линию в блоке `action`.



## Демонстрационный пример

- Продолжение Dropit

Меньше неустойчивости.

Направляем падение квадратов.

Если позволит время, разместим точно по сетке, используя делегата “столкновений” (collision) (если нет - пошлю код).

- **Что искать для сегодняшней демонстрации**

UIDynamicItemBehavior (преимущественно физические конфигурации).

UIAttachmentBehavior

Добавление action **блока** в поведение

Наблюдение за поведением элемента (истекшее время анимации, линейная скорость и т.д.)

Делегат “столкновений” collisionDelegate для UICollisionBehavior

Я не думаю, что у меня будет время показать работу делегата “столкновения” (collision delegate), но я вам пошлю код и вы поймете, что к чему.

Давайте вернемся к Xcode и демонстрационному примеру Dropit.

В прошлый раз мы переместили все поведение одного квадрата из DropitViewController.m в отдельный класс, который является subclass UIDynamicBehavior, и назвали его DropitBehavior. Мы разместили там “гравитацию” и “столкновения”.

Теперь мы добавим в наш пользовательский класс DropitBehavior еще одно поведение, которое позволяет нам управлять такими вещами, как “будут ли квадраты вращаться”. Кроме этого мы сможем управлять трением (friction), сопротивлением (resistance), плотностью (density) и другими подобными вещами. Поведение, которое управляет всеми этими вещами, называется UIDynamicItemBehavior, потому что оно определяет поведение именно этого элемента (item). Мы назовем это поведение animationOptions

```
@property (strong, nonatomic) UIDynamicItemBehavior*animationOptions;
```

Вы можете назвать его по-другому, если хотите, например, physicalCharacteristics.

Добавим код для отложенного получения (lazy instantiation) поведения animationOptions

```
-(UIDynamicItemBehavior*)animationOptions
{
    if (!_animationOptions) {
        _animationOptions = [[UIDynamicItemBehavior alloc] init];
        _animationOptions.allowsRotation = NO;
    }
    return _animationOptions;
}
```

Подсвеченная строка кода как раз и не позволяет квадратам вращаться. В этом фрагменте кода мы можем задавать и другие характеристики: friction, density и т.д.

Добавляем новое поведение в методы `addItem:` и `removeItem:`

```
- (void)addItem:(id <UIDynamicItem>)item
{
    [self.gravity addItem:item];
    [self.collider addItem:item];
    [self.animationOptions addItem:item];
}

-(void)removeItem:(id <UIDynamicItem>)item
{
    [self.gravity removeItem:item];
    [self.collider removeItem:item];
    [self.animationOptions removeItem:item];
}
```

Рис. 3

И наконец, добавляем новое поведение как дочернее к пользовательскому поведению

`DropitBehavior:`

```
- (instancetype)init {
    self = [super init];
    [self addChildBehavior:self.gravity];
    [self addChildBehavior:self.collider];
    [self addChildBehavior:self.animationOptions];

    return self;
}
```

Рис. 4

Поведение `animationOptions` действует также, как и любое другое поведение. Просто это

поведение определяет атрибуты вместо того, чтобы прикладывать какую-то силу или что-то типа силы, но в действительности вы можете думать об атрибутах также как о приложении сил, например, сил, удерживающих динамические элементы от вращения, или сил, которые делают трение больше.

ВОПРОС: Если у меня будет множество `UIDynamicItemBehavior`, которые имеют противоречие значения одних и тех же характеристик, например, `density` (плотность), то какое значение будет действовать?

ОТВЕТ: Прекрасный вопрос. Для этого существует правило: если у вас есть дерево поведений (*tree of behaviors*) и в нем есть несколько поведение `UIDynamicItemBehavior`, то существует правило “Last one wins”.

В документации точно определено это правило. Оно касается глубины предварительного порядка обхода дерева поведений и хорошо изложено в WWDC 2013 сессия 221. Если вы собираетесь

построить очень сложное дерево поведений, в котором некоторые `UIDynamicItemBehavior` будут встречаться дважды с различными поведениями более высокого порядка, то это совершенно законно, вы должны знать, что вы делаете, и какое `UIDynamicItemBehavior` (с какими значениями физических характеристик) будет выбрано. Обратитесь к документации, которая рассматривает более сложные поведения.

Запустим приложение и мы получим более устойчивые груды квадратов, хотя мы ничего не поменяли в поведении “гравитация”. Вы видите, что квадраты не крутятся, они немного отскакивают при приземлении, но остаются в том же столбце.

```

1 // DropitBehavior.m
2
3 #import "DropitBehavior.h"
4 @interface DropitBehavior()
5
6 @property (strong, nonatomic) UIGravityBehavior *gravity;
7 @property (strong, nonatomic) UICollisionBehavior *collider;
8 @property (strong, nonatomic) UIDynamicItemBehavior *animationOptions;
9
10 @end
11
12 @implementation DropitBehavior
13
14 - (UICollisionBehavior *)collider
15 {
16     if(!_collider) {
17         _collider = [[UICollisionBehavior alloc] init];
18         _collider.translatesReferenceBoundsIntoBoundary =YES;
19     }
20     return _collider;
21 }
22
23 -(UIGravityBehavior *)gravity
24 {
25     if(!_gravity) {
26         _gravity = [[UIGravityBehavior alloc] init];
27         _gravity.magnitude = 0.9;
28     }
29     return _gravity;
30 }
31
32 -(UIDynamicItemBehavior *)animationOptions
33 {
34     if (!animationOptions) {
35         _animationOptions = [[UIDynamicItemBehavior alloc] init];
36         _animationOptions.allowsRotation = NO;
37     }
38     return _animationOptions;
39 }
40
41 -(void)addItem:(id <UIDynamicItem>)item
42 {
43     [self.gravity addItem:item];
44     [self.collider addItem:item];
45     [self.animationOptions addItem:item];
46 }
47
48 -(void)removeItem:(id <UIDynamicItem>)item
49 {
50     [self.gravity removeItem:item];
51     [self.collider removeItem:item];
52     [self.animationOptions removeItem:item];
53 }
54

```

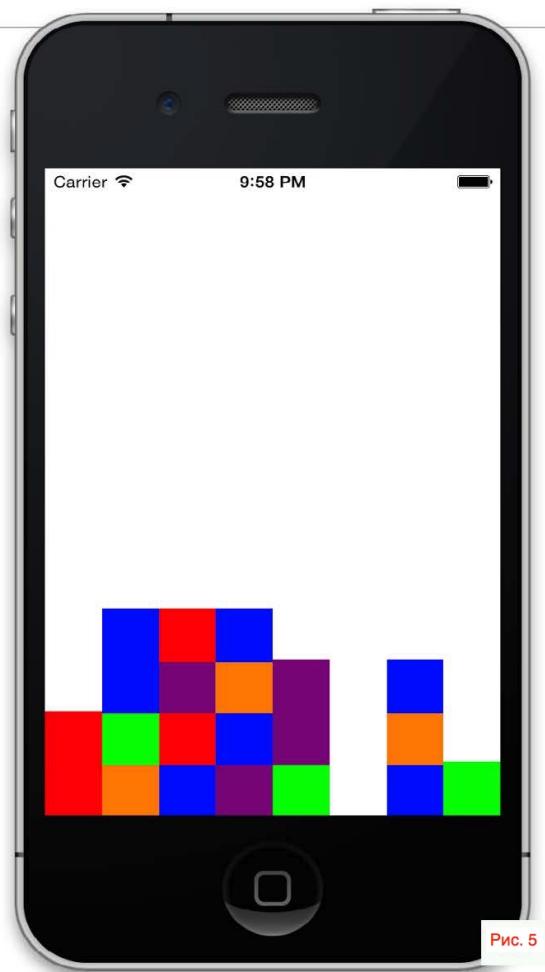


Рис. 5

Теперь квадраты “улетучиваются” из заполненной строки более надежно.

Вы должны понимать, что если у вас множество квадратов, анимирующих одновременно, то это происходит за счет снижения производительности, анимация не происходит даром. Если у вас есть приложение типа нашего демонстрационного примера, в котором вы бросаете квадраты, и разрешаете пользователю бросать множество квадратов, вам, возможно, придется делать некоторые дополнительные вещи. Например, вместо того, чтобы иметь границу столкновений и разрешить столкновение квадратов друг с другом, что если нам нарисовать границу вокруг верхних квадратов так, чтобы вновь прибывшие квадраты сталкивались с только что нарисованной границей? Тогда вы смогли бы удержать все нижние квадраты от расчета того, куда они переместятся. Такое решение примечательно тем, что после того, как квадраты “утрамбуются”, они в любом случае не будут отскакивать от нижней части экрана слишком много. Таким образом, вы могли бы совсем убрать отскок и разместить границу с помощью UIBezierPath. В этом случае она может проходить где угодно. Вы должны об этом думать, если начинают появляться экстремальные

случаи.

И последняя вещь, которую мы рассмотрим - это `UIAttachmentBehavior`. Это “присоединение” с помощью “стержня”, который может быть эластичным или абсолютно железным. Мы хотим, чтобы у нас был железный стержень, который не ломается и не растягивается. И мы будем использовать этот стержень, чтобы захватить один из падающих квадратов так, что мы сможем раскачать его и направить в нужный столбец, потому сейчас назначение столбца осуществляется очень строго, и сделать игру конкурентной очень тяжело.

Мы собираемся сделать пару вещей в нашем `DropitViewController`: очевидно, нам нужно добавить поведение “присоединение”, и мне также нужно знать, какой квадрат падает в данный момент, чтобы его захватить. Поэтому мы добавляем свойства:

```
@property (strong, nonatomic) UIAttachmentBehavior *attachment;
@property (strong, nonatomic) UIView *droppingView;
```

Так как падает множество `views` в нашей игре и все они собираются в кучи, то мы должен знать, какой `view` мы начали захватывать - этот `view` и есть `droppingView`, и мы должны отслеживать этот `view`. Отслеживать легко: достаточно поместить код в метод `drop`:

```
- (void)drop
{
    . . .
    CGRect frame;
    . . .
    [self.gameView addSubview:dropView];
    self.droppingView = dropView;

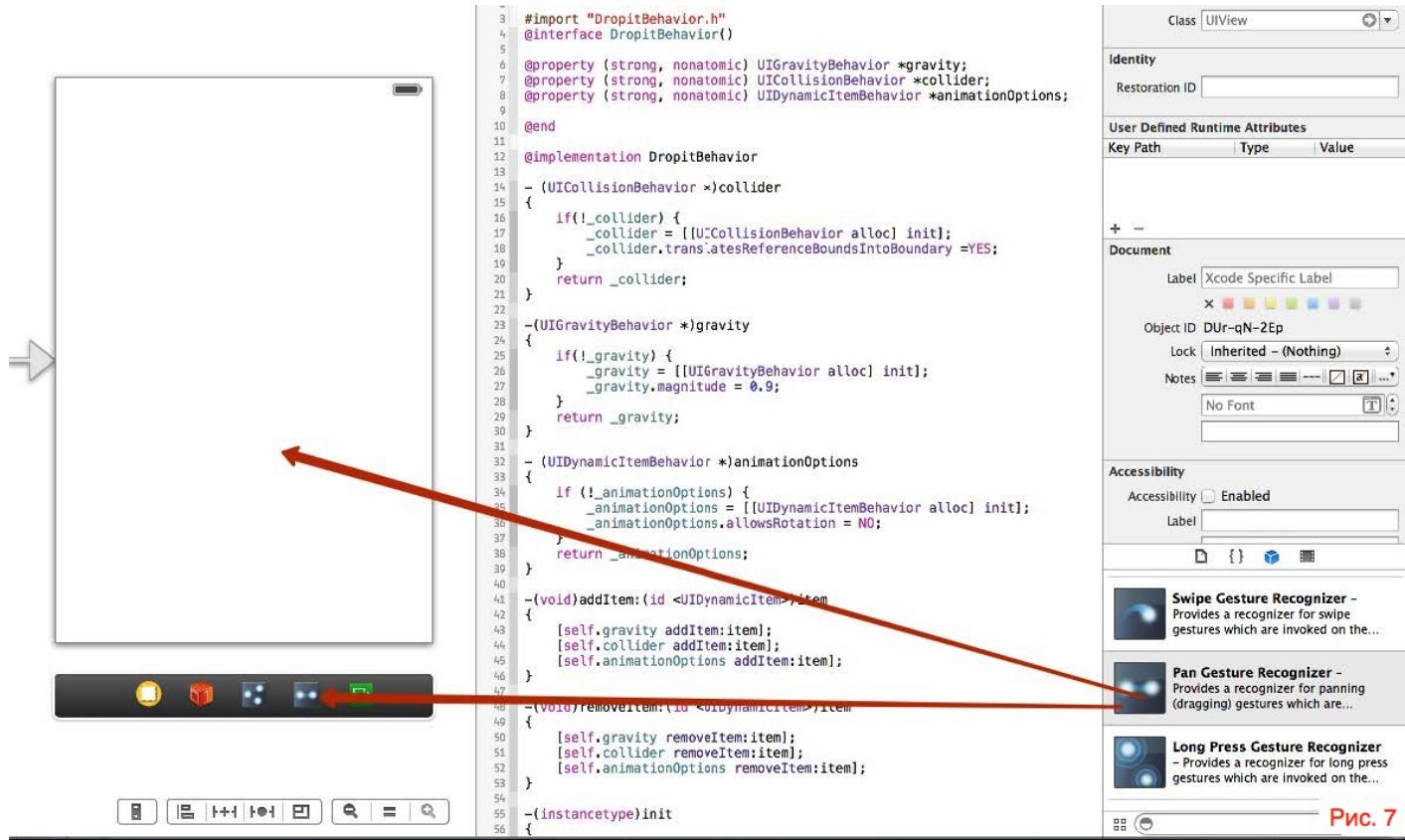
    [self.dropitBehavior addItem:dropView];
}
```

Рис. 6

Всякий раз, когда квадрат начинает падать, он может быть тем, который будет захвачен. И если падает множество квадратов, то это будет последний, если мы захватим его с помощью поведения “присоединения”.

Мы будем делать “присоединение” с помощью жеста `pan`, и здесь мы увидим другой распознаватель жестов, потому что мы будем захватывать квадрат, и двигать пальцы по экрану, поддерживая “присоединение”. Квадрат, который будет захвачен, будет следовать за нашими пальцами с помощью воображаемого стержня. Это интересно.

Мы будем устанавливать жест **pan** прямо на storyboard, взяв его из палитры объектов, и перетянув его на наш gameView.



Далее мы разместим жест **pan** прямо вслед за кодом для жеста **tap** и назовем распознаватель жеста **pan**:

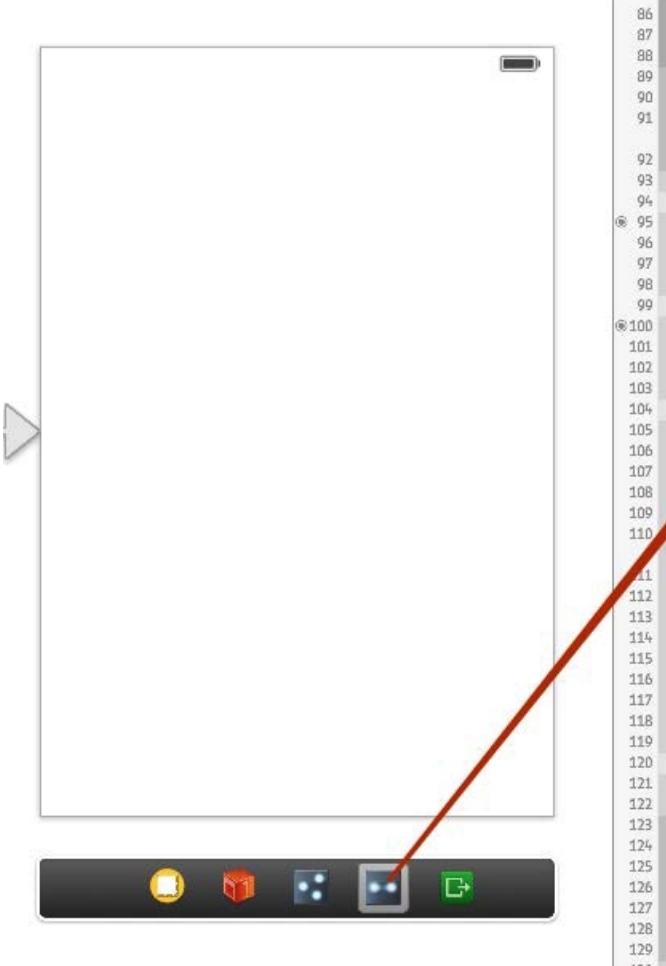


Рис. 8

Что же будет происходить в этом методе `pan:`?

Жест `pan`, также как и жест `pinch`, являются “продолжительными” жестами, поэтому по ходу действия происходит ряд событий, и у нас есть заготовленный код (snippet), который мы используем для такого рода жестов.

```
- (IBAction)pan:(UIPanGestureRecognizer *)sender
{
    if (sender.state == UIGestureRecognizerStateBegan) {

    } else if (sender.state == UIGestureRecognizerStateChanged) {

    } else if (sender.state == UIGestureRecognizerStateEnded) {

    }
}
```

Рис. 9

“Продолжительные” жесты проходят через одну и ту же последовательность событий: начало StateBegan, изменение позиции StateChanged по мере того, как вы водите пальцем по экрану, и окончание StateEnded. Что мы будем делать в этих трех состояниях с нашим жестом, если мы выполняем “присоединение”?

Первое, что мы собираемся сделать, и это действительно первый шаг, - определить, где на self.gameView произошел жест pan

```
CGPoint gesturePoint = [sender locationInView:self.gameView];
```

Мы находимся в методе, принадлежащем “распознавателю жестов” (**gesture recognizer**), которым в данном случае является sender, и он знает, где происходит жест pan согласно методу locationInView:. И каждый раз, когда вы водите пальцем по экрану, gesturePoint будет меняться на новую позицию.

Первое, что мы сделаем - “присоединим” droppingView к этой точке, точке жеста pan, в момент начала жеста pan, с помощью специального метода

```
CGPoint gesturePoint = [sender locationInView:self.gameView];
if (sender.state == UIGestureRecognizerStateBegan) {
    [self attachDroppingViewToPoint:gesturePoint];
} else if (sender.state == UIGestureRecognizerStateChanged) {
```

Рис. 10

и чтобы не забыть написать этот метод, сделаем заготовку

```
-(void)attachDroppingViewToPoint:(CGPoint)anchorPoint
{}
```

Когда жест pan меняется, то он передвигается на новую точку, и мы собираемся создать для своего жеста pan “якорную” точку (имеется в виду “присоединение” динамического элемента либо к другому динамическому элементу, либо к “якорной” точке”).

Мы создаем для поведения “присоединение” “якорную” точку, которая будет точкой жеста **pan**, и которая будет меняться при изменении жеста **pan** и, соответственно, точки **gesturePoint**

```
    . . . . .
} else if (sender.state == UIGestureRecognizerStateChanged) {
    self.attachment.anchorPoint = gesturePoint;
} else if (sender.state == UIGestureRecognizerStateEnded) {
```

После того как жест закончиться, мы просто уберем поведение “присоединения” из аниматора

```
} else if (sender.state == UIGestureRecognizerStateEnded) {
    [self.animator removeBehavior:self.attachment];
```

Рис. 11

и аниматор прекратит выполнять “присоединение”.

И это все.

Осталось выполнить начальное “присоединение” к **droppingView**.

Если у нас имеется **droppingView**, то инициализируем поведение “присоединение” одним из нескольких методов, а именно, методом инициализации **initWithItem: attachedToAnchor:** и добавляем это поведение к аниматору для немедленного начала анимации

```
- (void)attachDroppingViewToPoint:(CGPoint)anchorPoint
{
    if (self.droppingView) {
        self.attachment = [[UIAttachmentBehavior alloc]
                           initWithItem:self.droppingView
                           attachedToAnchor:anchorPoint];
        self.droppingView = nil;
        [self.animator addBehavior:self.attachment];
    }
}
```

Рис. 12

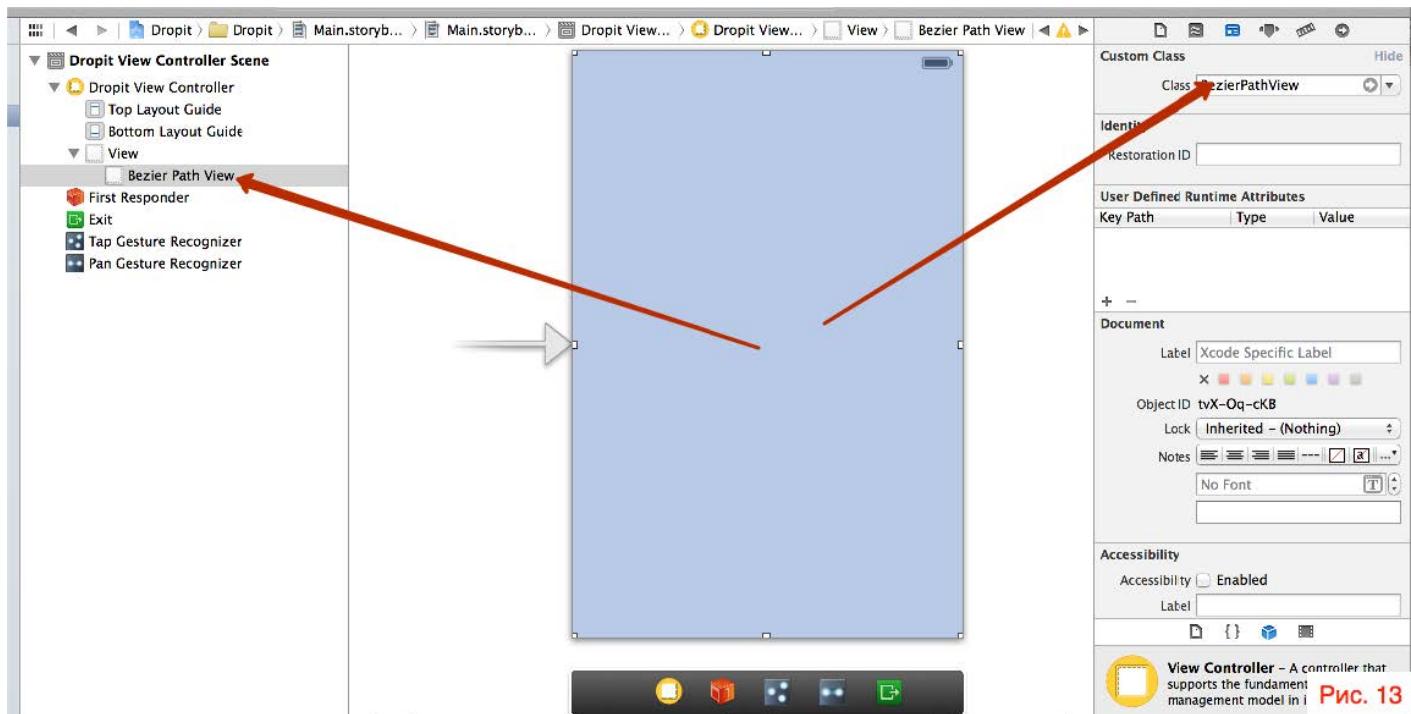
После того, как “присоединение” произошло, мы разрешим сделать это снова (уже к другому “падающему” квадрату), поэтому появилась строка кода

```
self.droppingView = nil;
```

Другими словами, вы “присоединяетесь” к какому-то, “падающему” в данный момент, квадрату, заставляете его следовать за вашим жестом, а после окончания жеста, вы сможете “присоединить” уже другой “падающий” квадрат.

Мы можем запустить приложение и проверить, что все работает.

Следующая вещь, которую мы рассмотрим - это рисование линии между точкой жеста **pan** и “падающим” квадратом, чтобы показать визуально как работает поведение “присоединения” (attachment) и как работает этот маленький **action** блок. Как мы будем рисовать? Мы сделем `gameView` на нашем storyboard view класса **BezierPathView**, предназначенным для рисования



Для этого мы создадим новый класс File -> New -> File -> Objective-C class с superclass `UIView` и назовем его `BezierPathView`. Это будет полностью обобщенный (generic) класс для рисования, и его public API будет единственное свойство `path`

```
@property (strong, nonatomic) UIBezierPath *path;
```

Единственное, что мы должны сделать в файле `BezierPathView.m`

это разместить единственную строку кода в drawRect:

```
- (void)drawRect:(CGRect)rect
{
    // Drawing code
    [self.path stroke];
}
```

Еще я должен добавить setter для свойства path

```
- (void)setPath:(UIBezierPath *)path
{
    _path = path;
    [self setNeedsDisplay];
}
```

Рис. 14

Всякий раз, когда кто-то изменяет path, он будет перерисовываться. Этот класс комбинирует все, что вы знаете о рисовании: drawRect:, UIBezierPath, stroke, setNeedsDisplay. После создания класса возвращаемся на storyboard и заменяем обобщенный класс UIView для нашего gameView на пользовательский класс BezierPathView, как показано на скриншоте вверху.

В файле DropitViewController.m также необходимо заменить класс свойства gameView с UIView \* на BezierPathView \*

```
@property(weak, nonatomic) IBOutlet BezierPathView *gameView;
```

добавив

```
#import "BezierPathView.h"
```

Теперь мы можем посыпать self.gameView сообщения класса UIBezierPath и наш gameView сможет отображать произвольную BezierPath.

Далее, чтобы показать “присоединение” (attachment) нам необходимо нарисовать BezierPath линию между “якорной” точкой и “падающим” квадратом droppingView, и каждый раз, как меняется droppingView или “якорной” точка, мы должны перерисовывать BezierPath линию. И здесь вступает в действие action block

```

- (void)attachDroppingViewToPoint:(CGPoint)anchorPoint
{
    if (self.droppingView) {

        self.attachment = [[UIAttachmentBehavior alloc]
                           initWithItem:self.droppingView
                           attachedToAnchor:anchorPoint];
        self.attachment.action = ^{
            UIBezierPath *path = [[UIBezierPath alloc] init];
            [path moveToPoint:self.attachment.anchorPoint];
            [path addLineToPoint:self.droppingView.center];
            self.gameView.path = path;
        };
        self.droppingView = nil;
        [self.animator addBehavior:self.attachment];
    }
}

```

Рис. 15

В `action` block мы создаем `UIBezierPath` с помощью `alloc/init` и перемещаемся в “якорную” точку, а затем добавляем линию, соединяющую “якорную” точку с центром “падающего” квадрата.

Полученную `UIBezierPath` мы присваиваем нашему `self.gameView.path`.

Еще одна вещь, которую нам необходимо сделать - это, когда жест закончится, мы в методе

```
- (IBAction)pan:(UIPanGestureRecognizer *)sender
```

убираем поведения “присоединение” (attachment) из аниматора, и должны сделать

```
self.gameView.path равной nil
```

```
    . . . . .
} else if (sender.state == UIGestureRecognizerStateEnded) {
    [self.animator removeBehavior:self.attachment];
    self.gameView.path = nil;
}
```

Рис. 16

Потому что блок `action` уже не будет исполняться, и нам не нужно последнее, что он нарисовал. Если мы будем раскачивать квадрат, а потом бросим с размаха, то квадраты будут сталкиваться, так как поведение “столкновения” (collide) все еще действует. И у нас возникает небольшая проблема. Квадраты не могут вращаться и не могут опрокидываться, потому что отсутствие вращения предотвращает их от этого. Но для игры это плохо, потому что как мы собираемся их выравнивать в нижней части экрана?

Когда происходят “столкновения”, поведение “столкновения” (collision) имеет делегата и в этом делегате находятся методы, которые вы можете выполнять когда происходят “столкновения”.

Там есть методы, которые говорят, когда “столкновения” начинаются и заканчиваются. И в методе “окончания” столкновений мы можем разместить динамические элементы в заданной сетке. Можно рассмотреть линейную скорость элемента, и если он движется направо, то разместить его правее, но строго по сетке, если он движется налево, то “подтолкнуть” его левее к узлу сетки.

## Итак, что такое Autolayout (Авторазметка)?

Autolayout (авторазметка) - это способ установки фреймов (frames) для всех ваших views, которые вы создаете в коде или перетаскиваете на storyboard. Мы собираемся решить, где расположить эти frames на экране, используя правила (rules), а не числа. Вместо того, чтобы говорить:

"Расположите этот frame в точке (20 , 20 ) шириной 180 и высотой 200", мы собираемся использовать правила, чтобы заставить их "идти" в нужное место. И эти правила очень гибкие и мощные, и все они, в основном, касаются взаимного расположения одного view по отношению к другому view или по отношению к superview.

Почему нам необходимо определить эти frames как правила, а не числа? Потому что иногда границы (bounds) прямоугольника, содержащего все ваши views, меняются.

Либо потому что прибор вращается, либо потому что мы можем находиться на длинном iPhone 5s или на коротком iPhone 4 или 4s, или мы вставлены внутрь UINavigationController или UITabBarController, или вы находитесь на iPad в некотором popover controller view или что-то подобное этому. Существует множество причин, по которым границы (bounds) меняются. Так что мы хотим построить наши views таким образом, чтобы все, что находится внутри них, подстраивалось автоматически.

Итак, у нас есть правила, которые мы называем **constraints** (ограничения), **layout constraints** (ограничения разметки), и именно об этих ограничениях мы будем говорить. Это правила для установки frames всех этих views. В iOS существует очень мощный программный интерфейс API для объектов и их **constraints**, но мы практически всегда будем задавать **Autolayout constraints** (ограничения разметки) в Xcode визуально, а не в коде. Конечно, в коде вы можете делать очень продвинутые вещи с **Autolayout**.

Но мы будем изучать **Autolayout** в Xcode и задавать **constraints** визуально графически, поэтому для лекции об **Autolayout** мы будем использовать screenshots.

# Autolayout

## • Setting UIView frames using rules rather than numbers

Why? Because many things affect the size of the area available to put views ...

Rotation

4 inch versus 3.5 inch iPhone

Embedding Controller's Views inside other Controllers (tab bars, navigation controllers, etc.)

We need these rules to put the views in their place no matter what bounds are available.

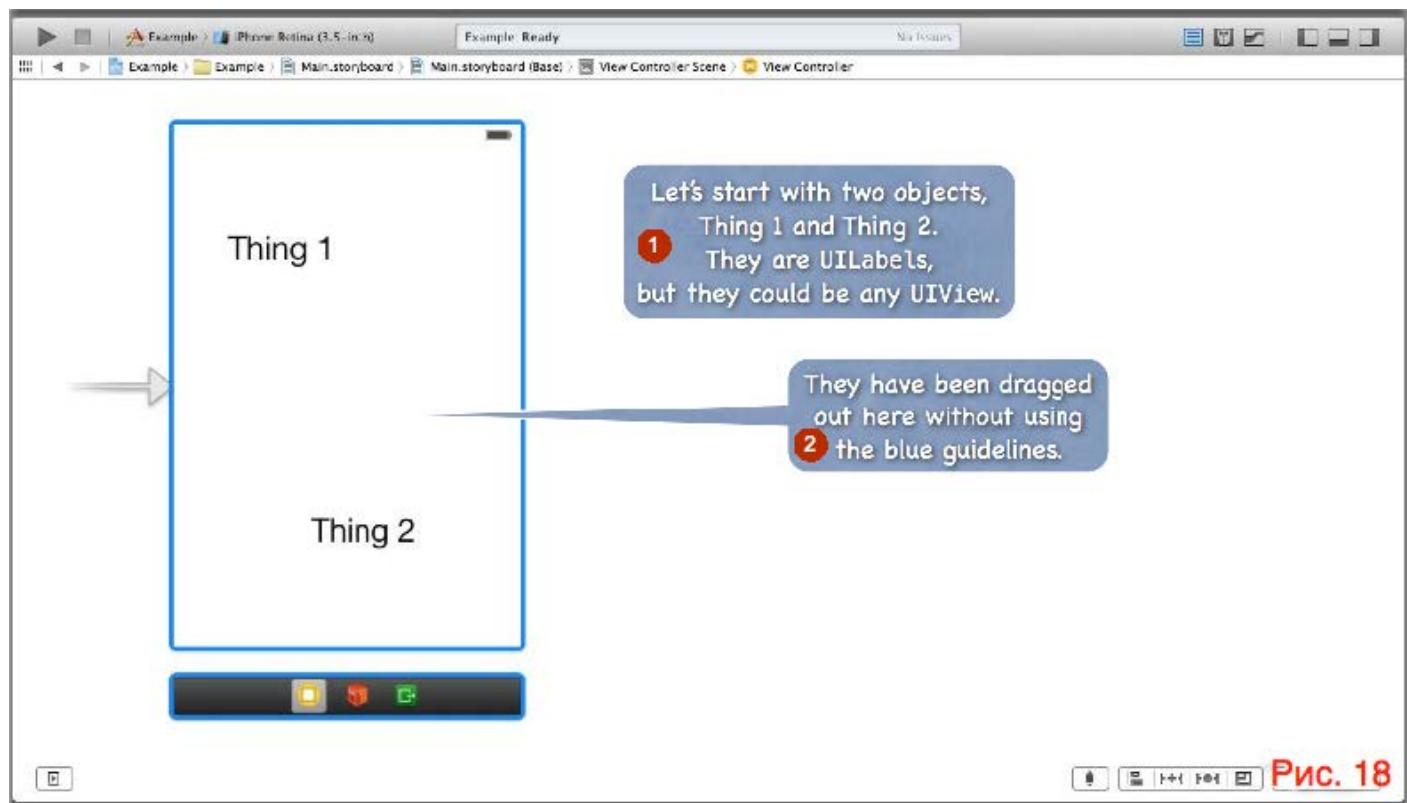
We call these rules "constraints".

There is a very powerful API (NSLayoutConstraint) for doing this, but ...

## • We almost always set up these rules in Xcode 5 graphically

So this is all best shown with some screen shots ...

Начнем с того, что разместим пару меток на ViewController: Thing 1 и Thing 2



Мы хотим, чтобы Thing 1 и Thing 2 оставались на правильных местах при изменении границ (bounds).

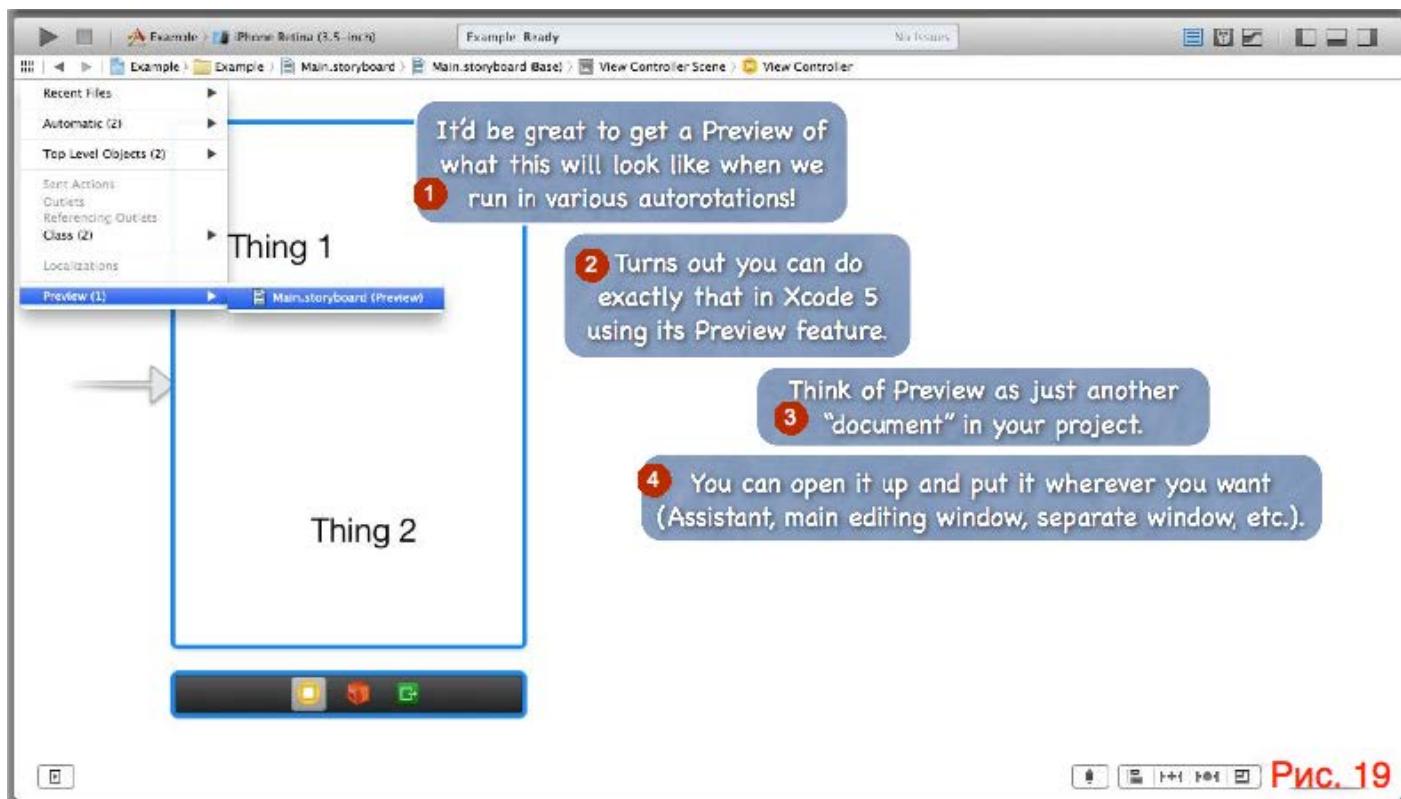


Рис. 19

1. Было бы здорово получить Preview того, как будет выглядеть ViewController, если мы запускаем приложение в различных положениях экрана (autorotation).
2. Оказывается, вы точно можете это делать в Xcode 5, используя возможность Preview.
3. Думайте о Preview просто как о другом “document” в вашем проекте.
4. Вы можете открыть Preview там, где вам захочется (в Ассистенте Редактора, в основном окне редактирования, в отдельном окне и т.д.).

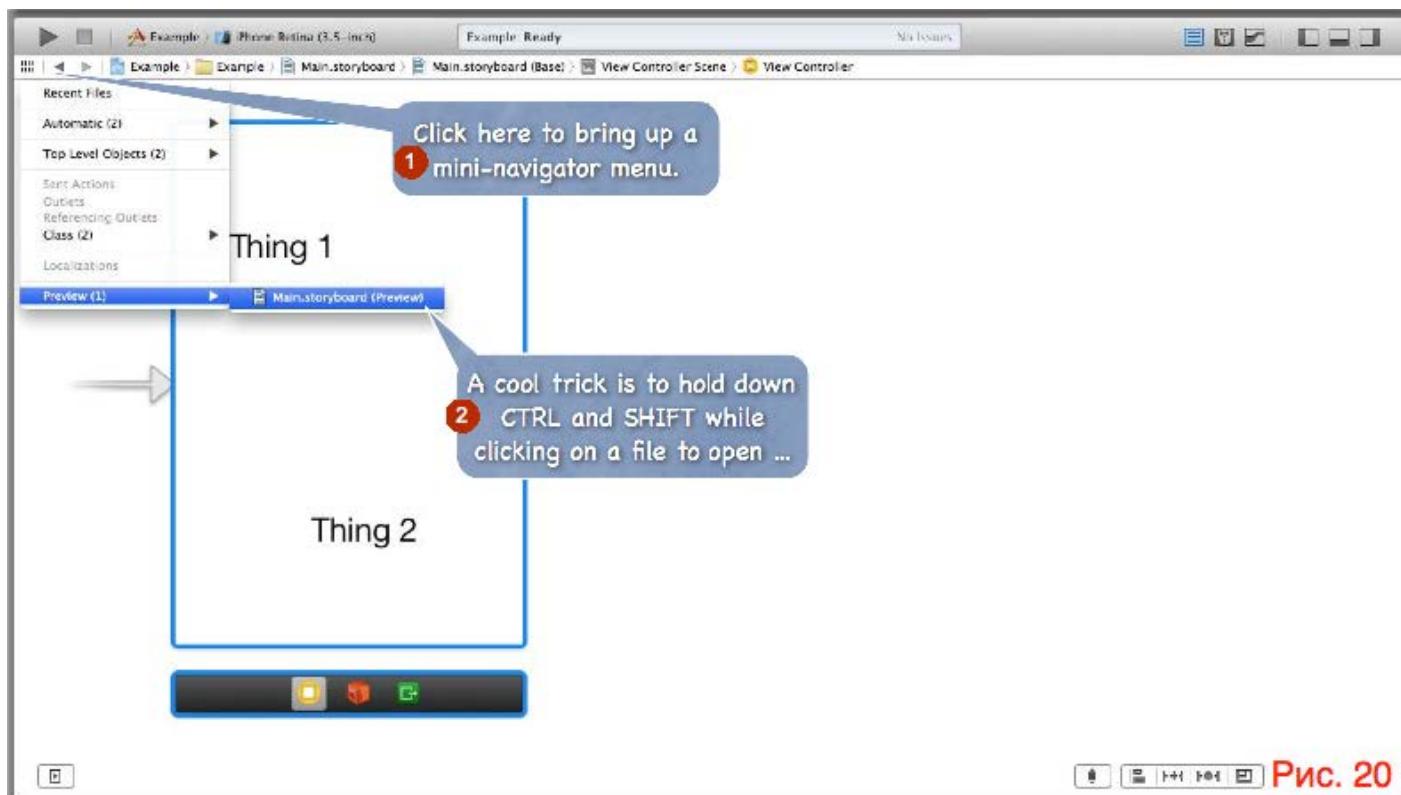
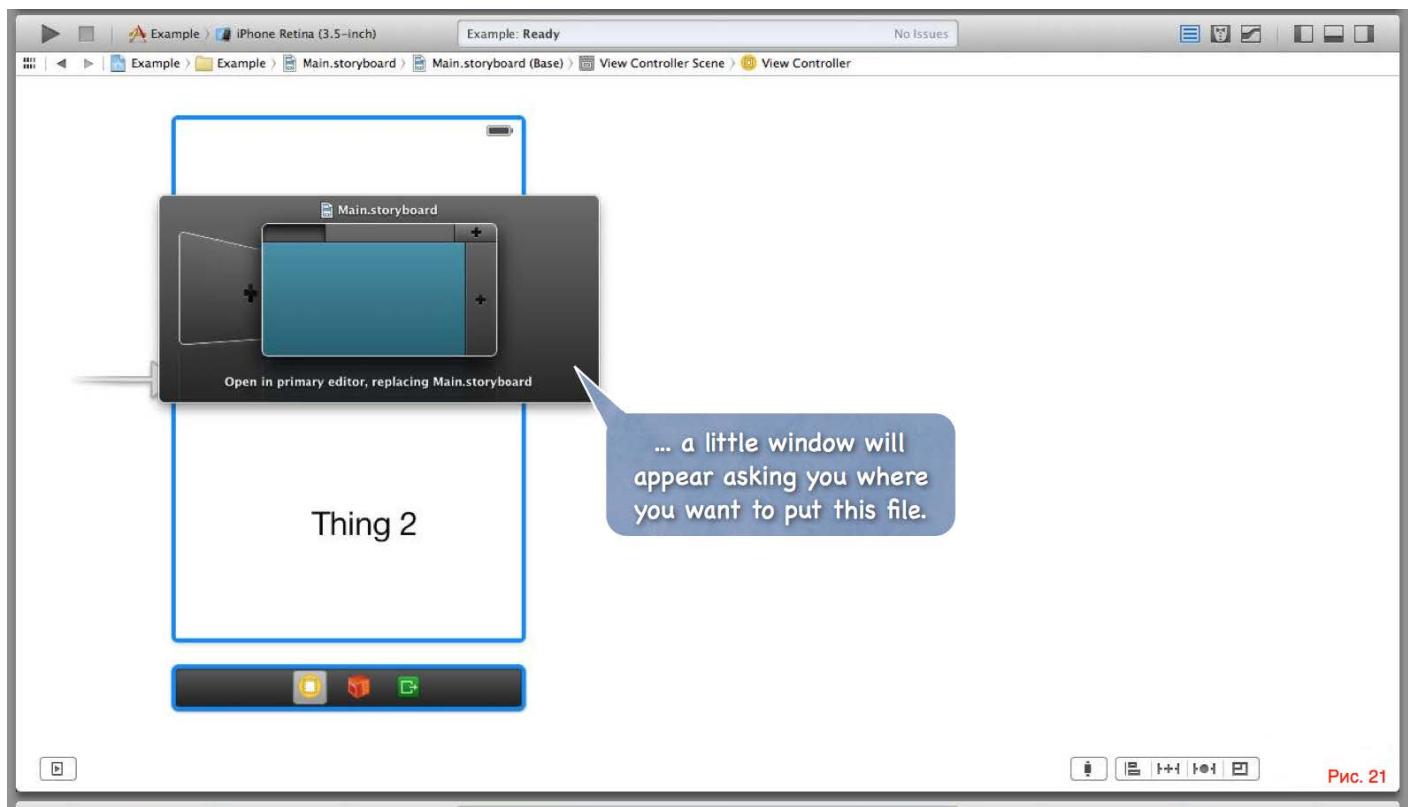


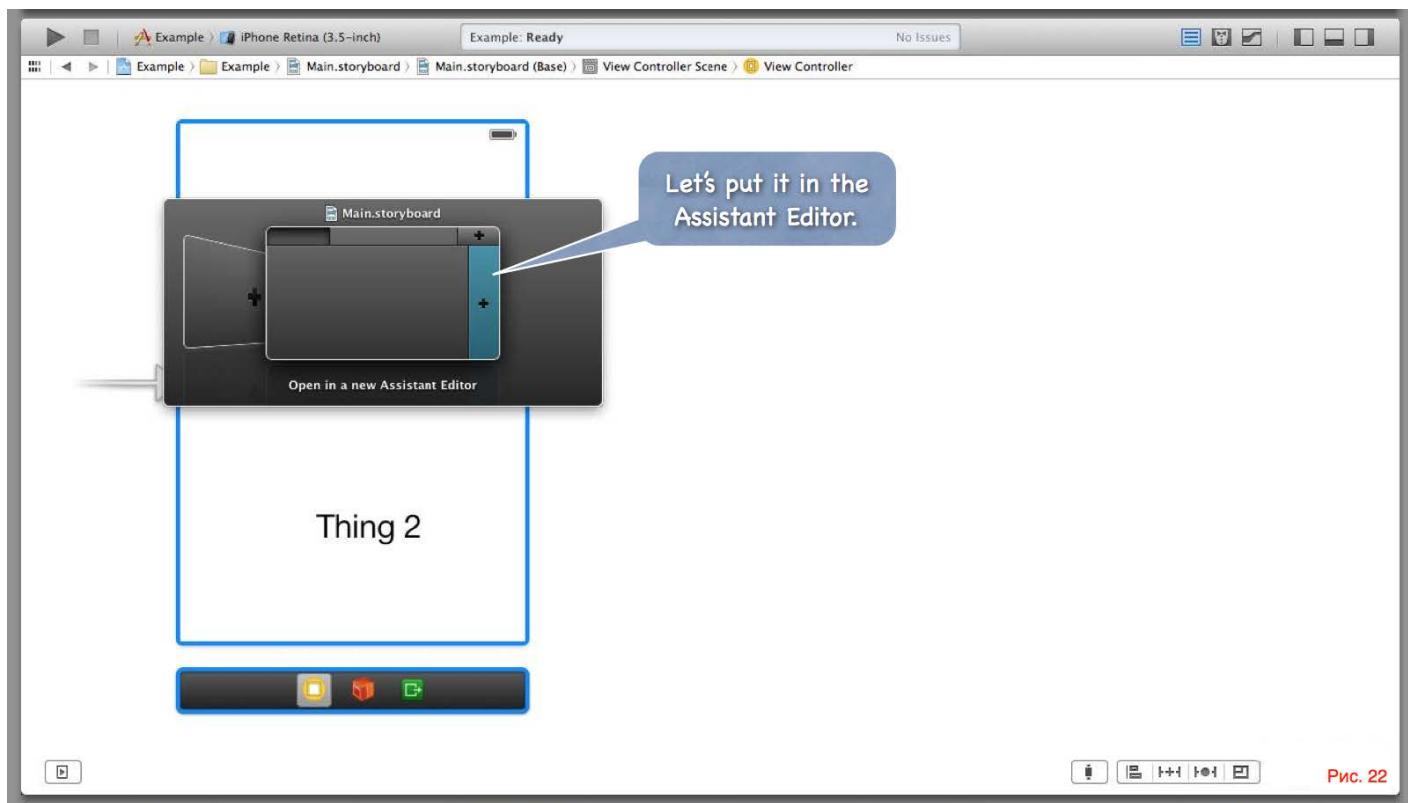
Рис. 20

1. Кликните здесь, чтобы появилось мини-навигационное меню.
2. Замечательная вещь происходит, если держать нажатыми OPTION (alt)\* и SHIFT клавиши во время открытия файла...



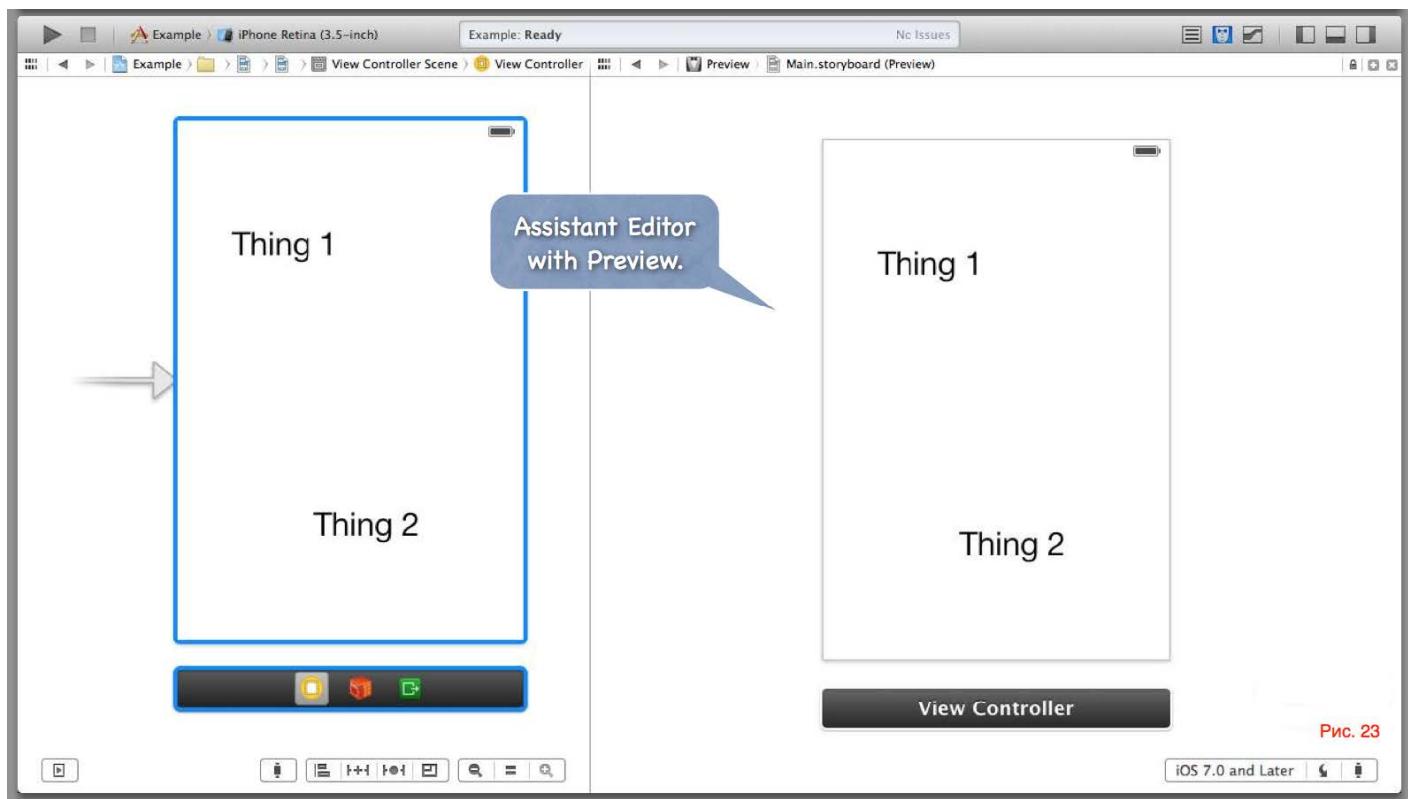
...появляется маленькое окно, в котором спрашивается, где вы хотите разместить Preview файл.

Рис. 21



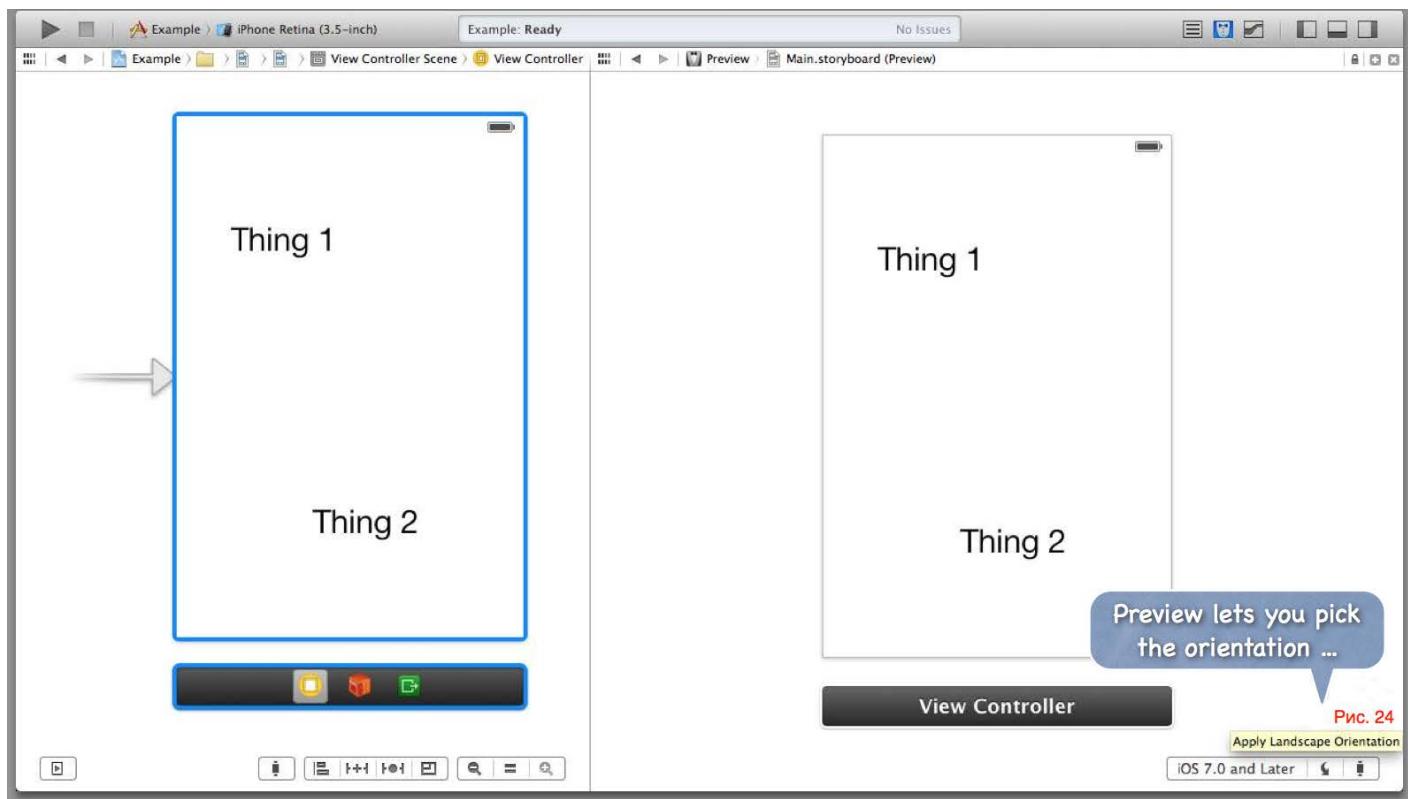
Давайте разместим его в Ассистенте Редактора.

Рис. 22



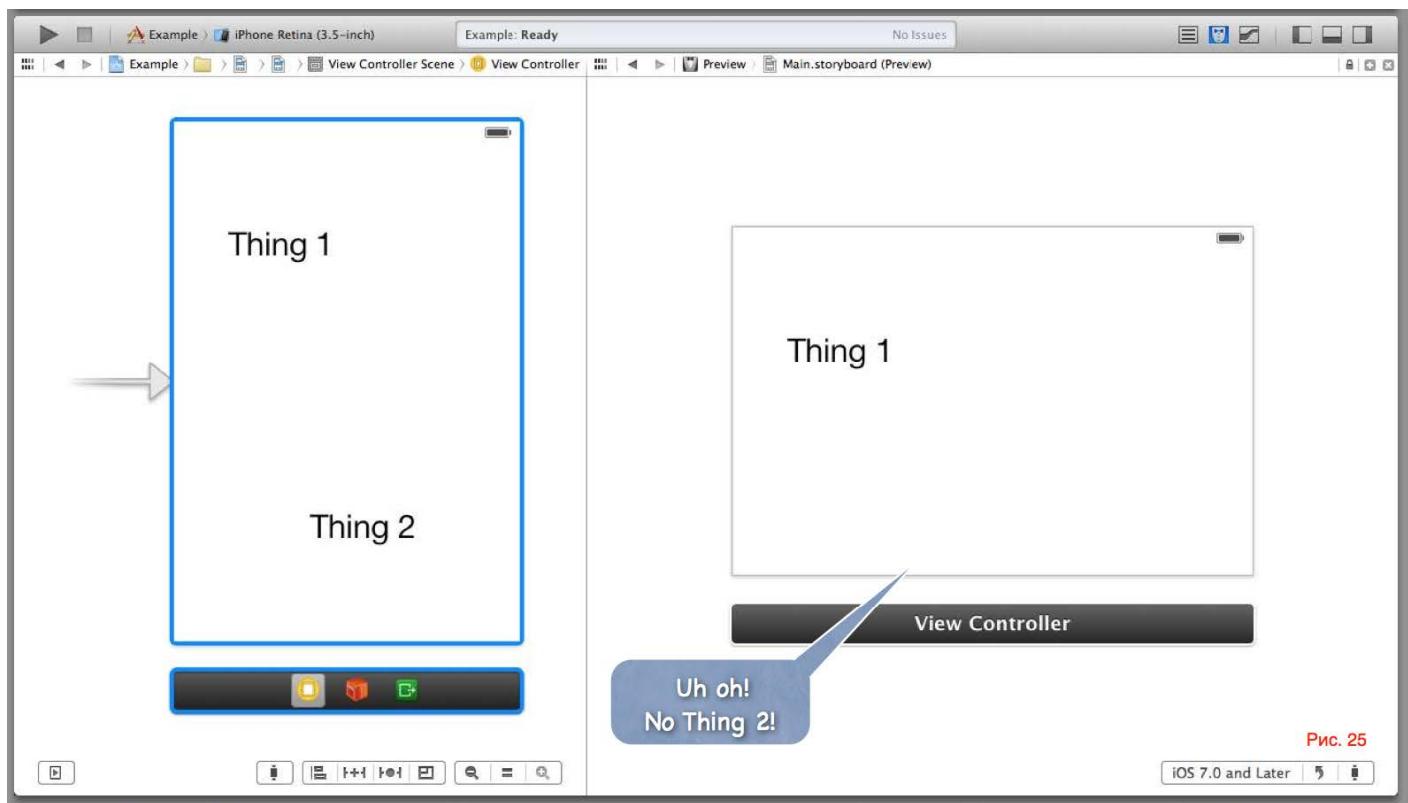
Ассистент Редактора с Preview.

Рис. 23



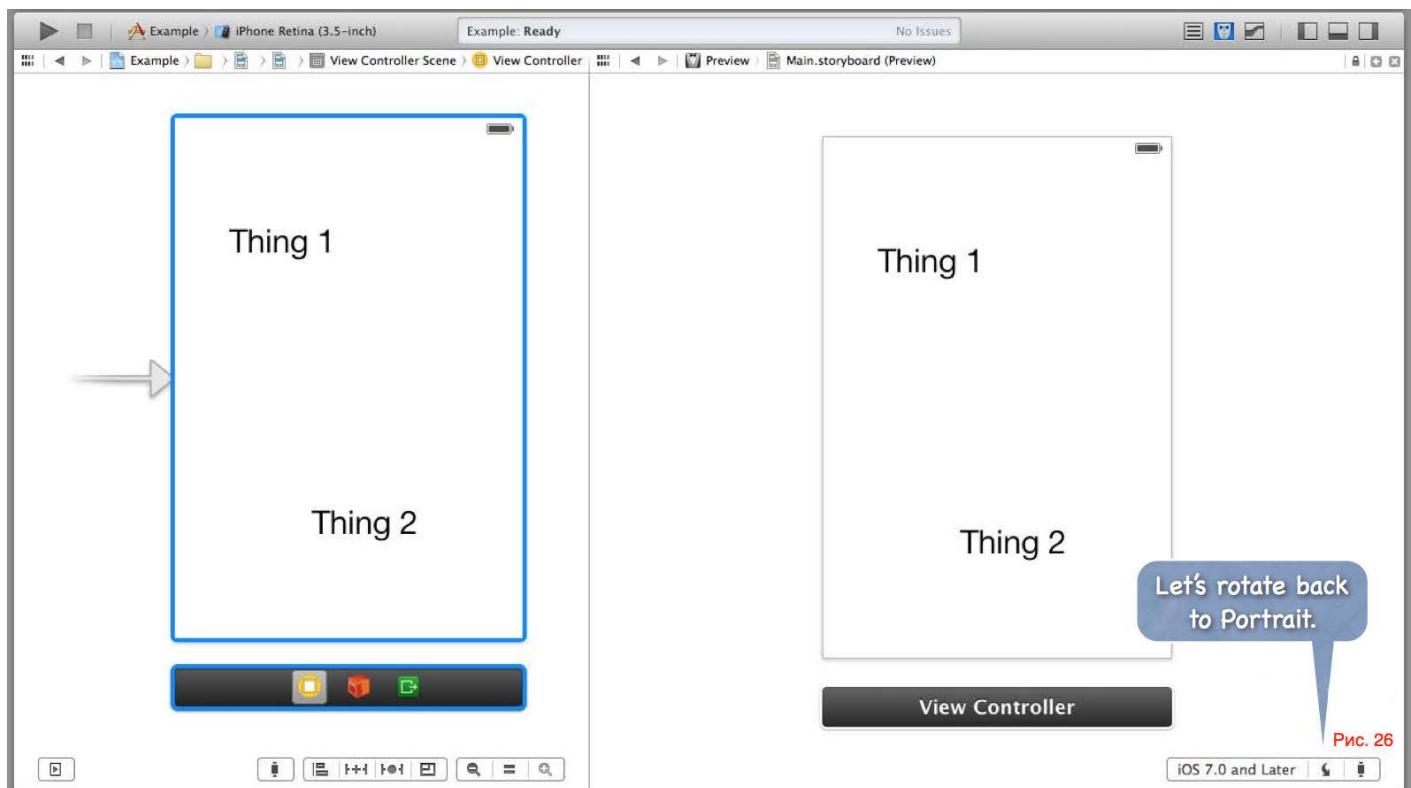
Preview позволяет вам выбирать ориентацию...

Рис. 24

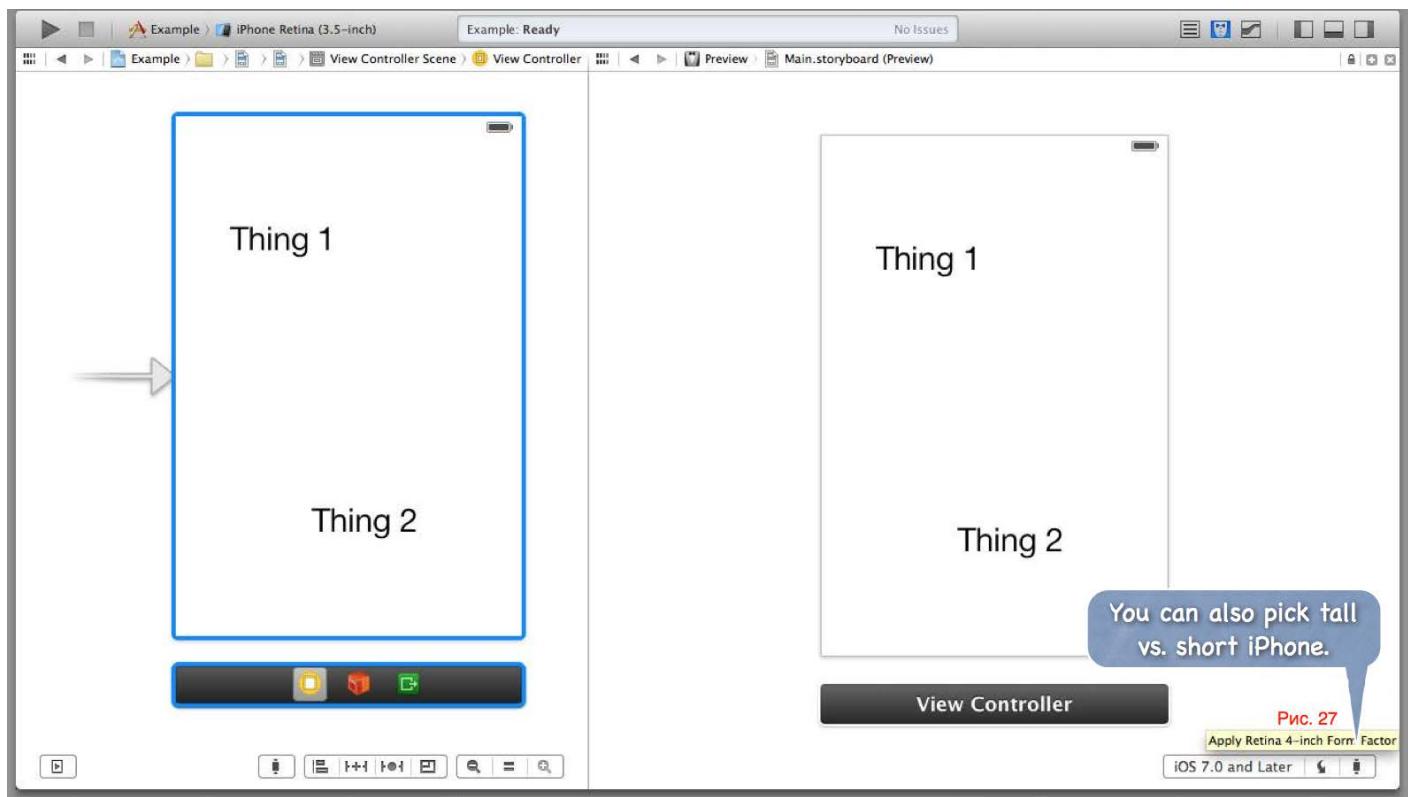


O! Нет Things 2!

Рис. 25



Давайте развернем обратно в положение Портрет.



Вы также можете выбрать длинный в противоположность короткому iPhone.

Рис. 27

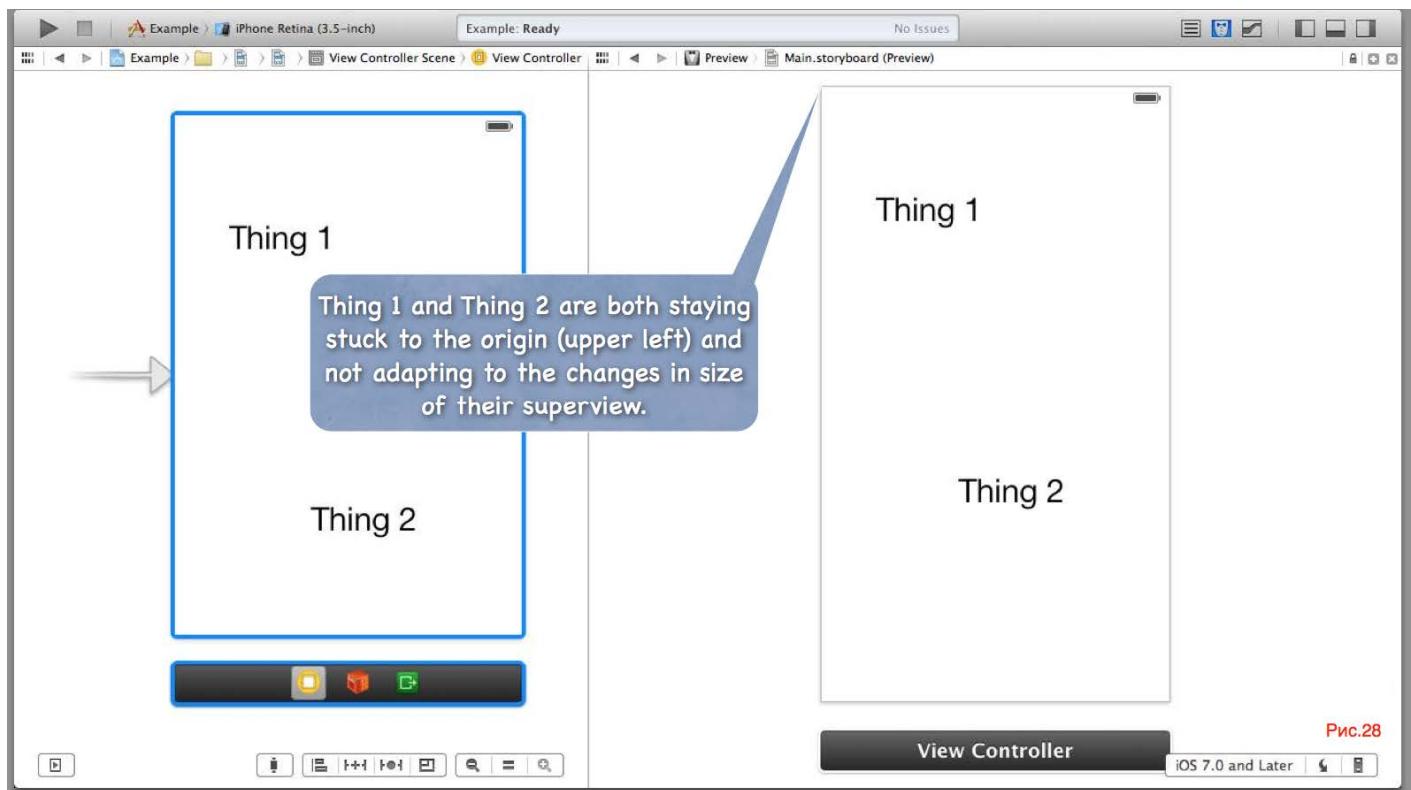
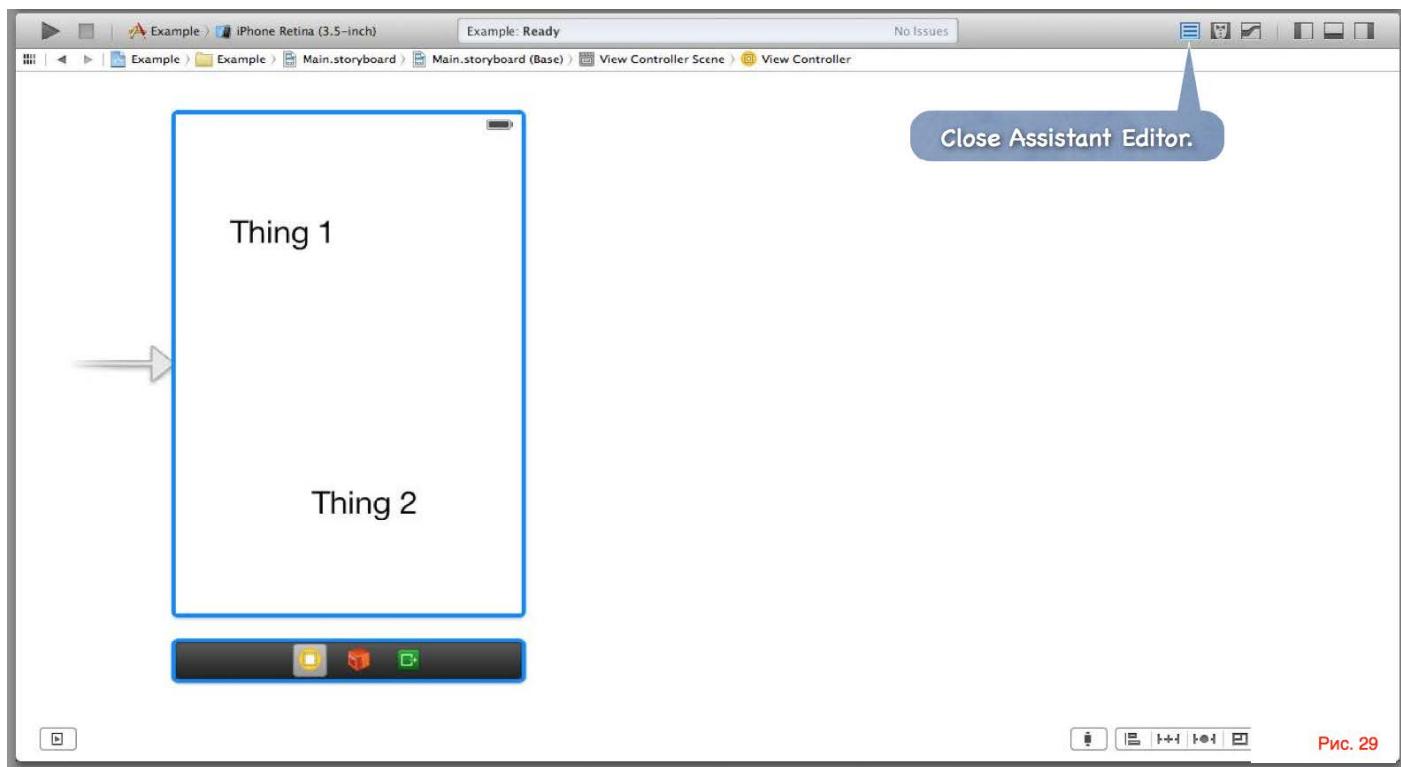
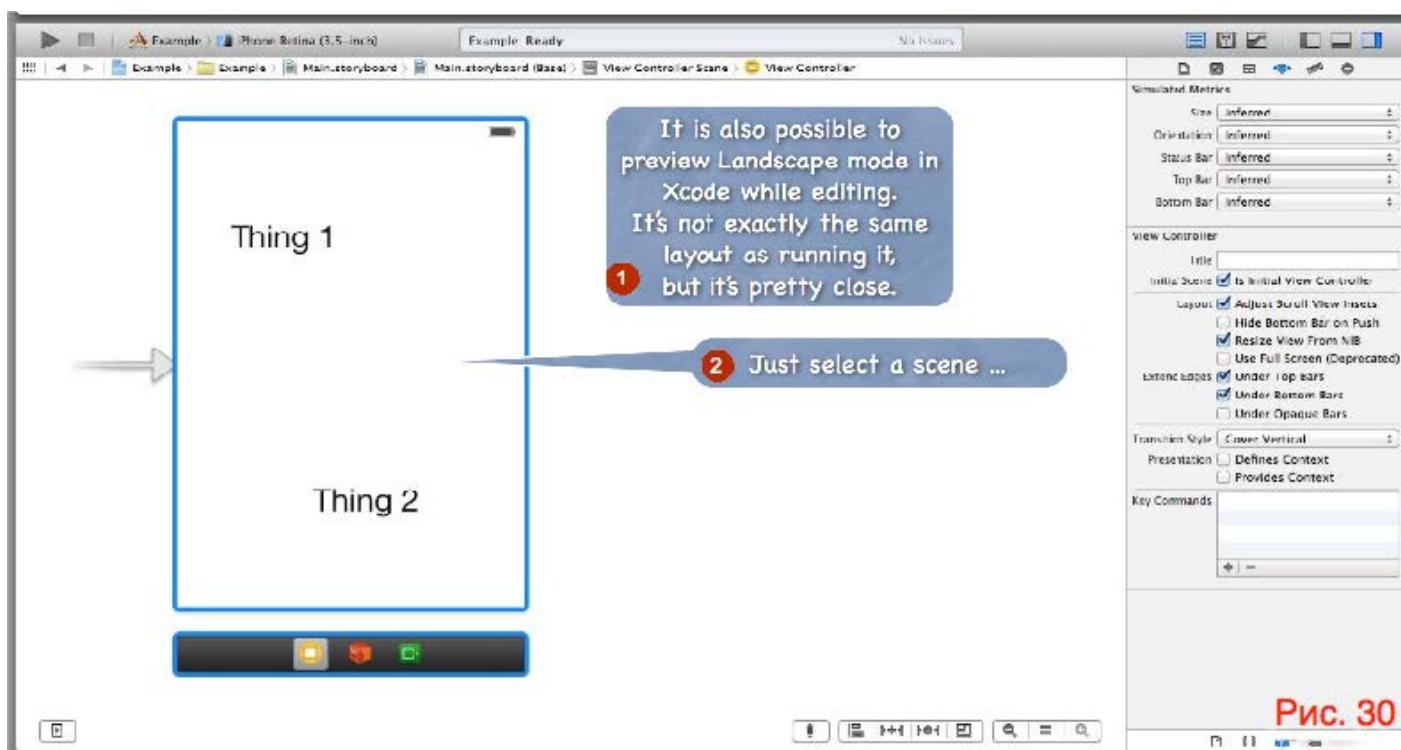


Рис.28

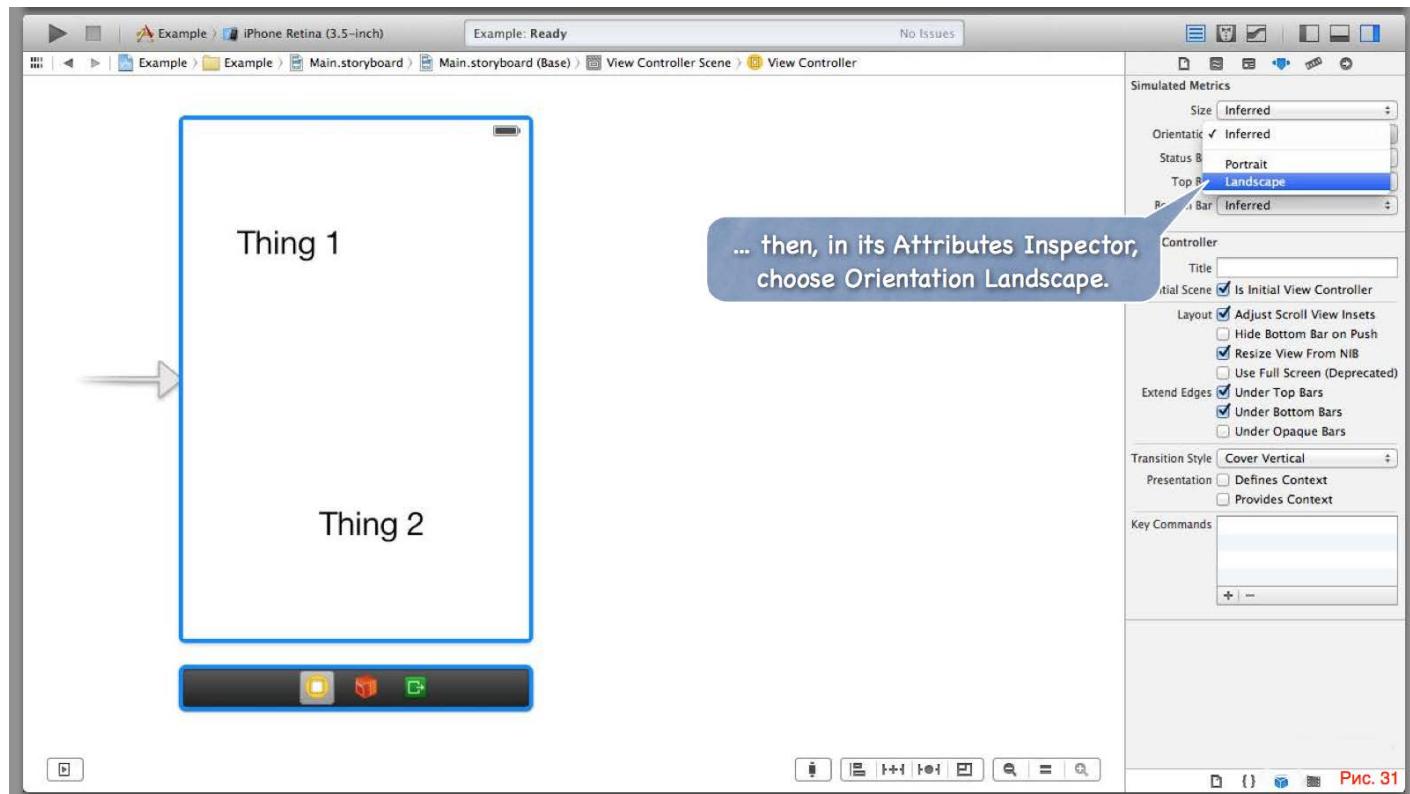
Thing 1 и Thing 2 остаются приклеенными к тем же местам по отношению к верхнему левому углу и не адаптируются к изменениям размеров superview, в котором они находятся.



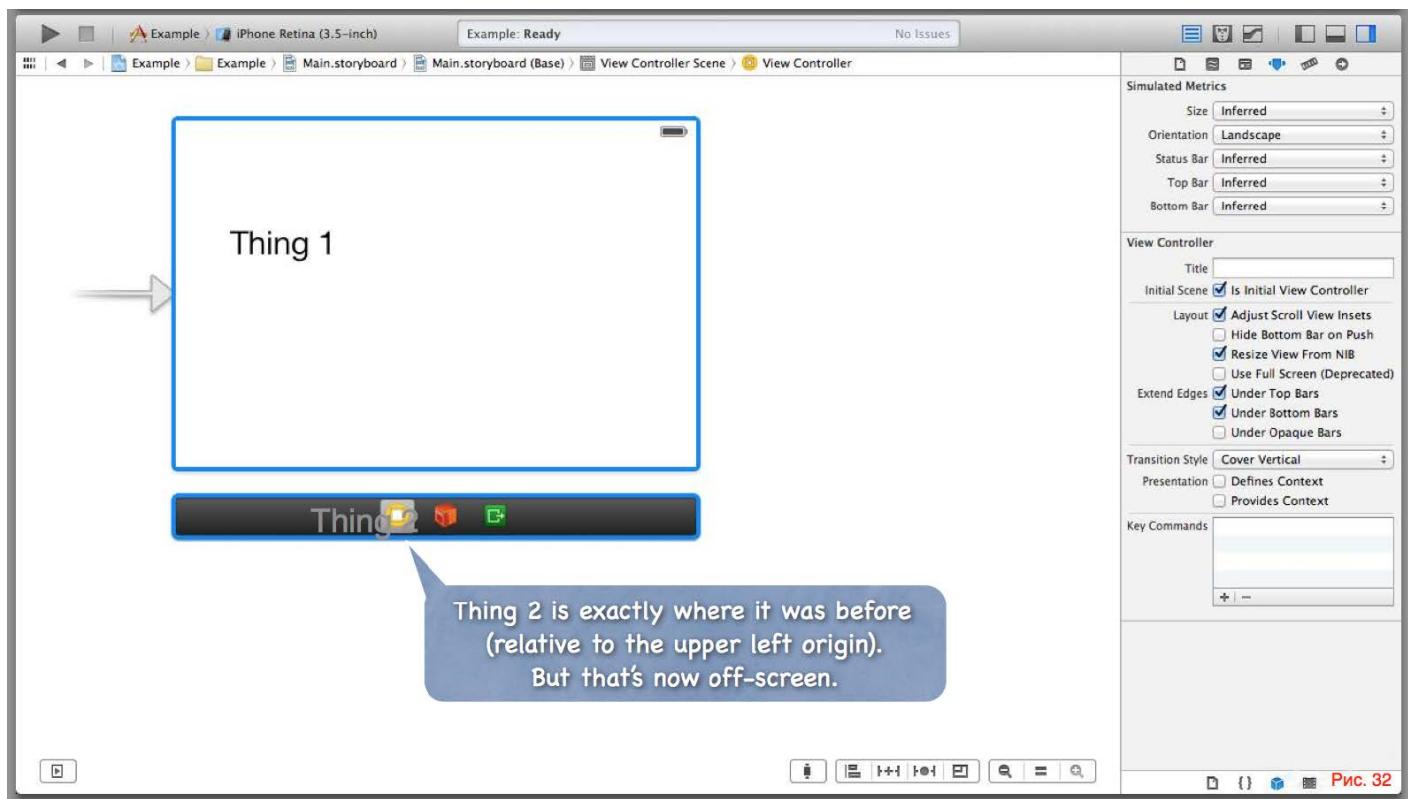
Закрываем Ассистент Редактора.



1. Также возможно предварительно просматривать режим “ландшафт” в Xcode во время редактирования. Это не точно такое же расположение, как при запуске приложения, но очень близкое к нему.
2. Просто выберите scene (сцену)...



... и затем, в Инспекторе Атрибутов, выберите Orientation Landscape.



Thing 2 в точности там, где она была перед этим (по отношению к верхнему левому углу). Но сейчас эта метка расположена за пределами экрана.

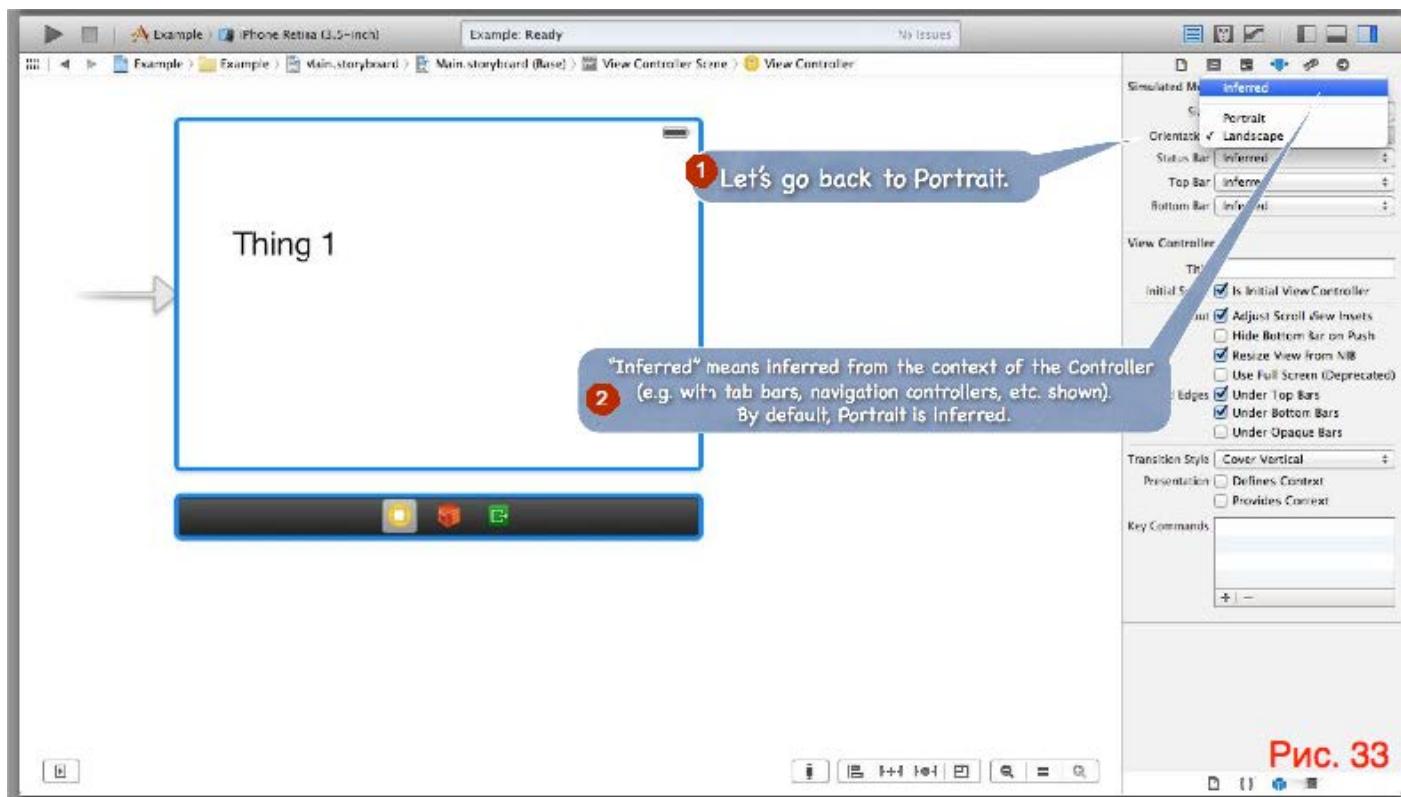
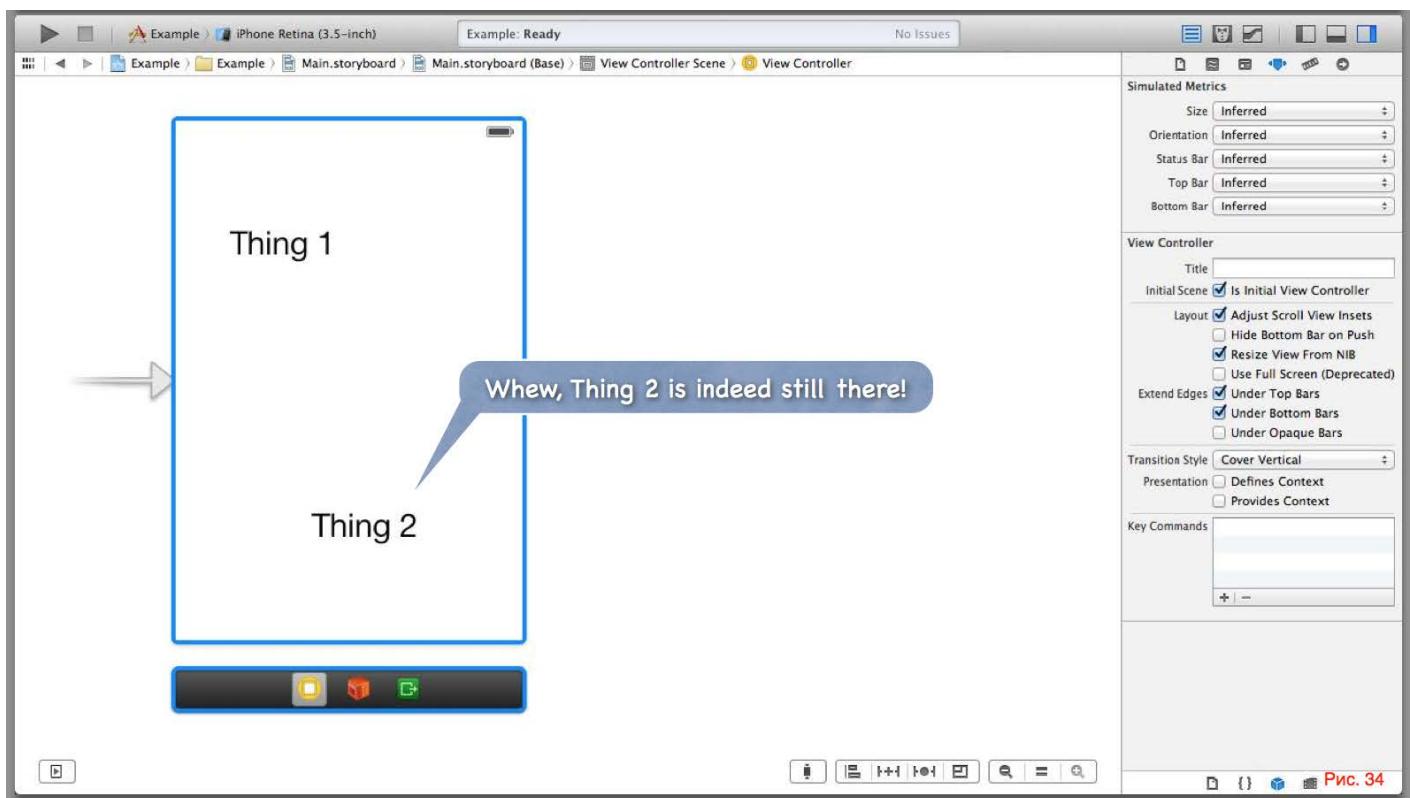


Рис. 33

1. Давайте вернемся к режиму Портрет.
2. “Inferred” означает выведенный из контекста Контроллера (например, с Tab Bars, Navigation Controller и т.д.). По умолчанию, режим Портрет является выведенным.



Да, Thing 2 в действительности все еще здесь.

Итак, у нас есть два способа предварительного просмотра.

Один - это настоящий Preview в Xcode 5, который будет показывать как выглядит пользовательский интерфейс при различных вращениях, различных iPhones, даже при iOS 7 или iOS 6, если вы пишите приложение, которое работает как в iOS 7, так и в iOS 6.

Второй способ предварительного просмотра - это инспектировать ваш Controller (Помните! Он должен быть обрамлен голубой линией) в Инспекторе Атрибутов, в верхней части которого расположены 5 элементов, называемых simulated metrics, с помощью которых вы сможете симулировать пользовательский интерфейс в различных ориентациях. Таким способом вы можете убедиться, что ваш UI работает с Autolayout.

Мы переключаем orientation (ориентацию) в режим landscape (ландшафт).

Мы уже видели, что в этом случае мы теряем Label Thing 2, хотя Thing 2 все еще находится на controller, но, к сожалению, за пределами экрана.

Если мы вернемся опять в режим Orientation inferred или Portrait, то у нас опять появилась Thing 2. Thing 1 и Thing 2 находятся сейчас в случайных местах и нам нужно разместить их в более

подходящих местах на экране, где мы сможем ими управлять : Thing 1 - в левый верхний угол, а Thing 2 - в правый нижний угол, и мы хотим, чтобы они находились именно в этих углах вне зависимости от того, куда эти углы будут перемещаться.

Если углы будут перемещаться вверх в случае режима “Ландшафт” или вниз в случае iPhone 5s, нам нужно, чтобы метки были приклейены к своим углам. Сейчас мы рассмотрим 3 различных способа выполнения этих правил.

Первым способом вы уже пользовались - это голубые направляющие линии, с помощью которых вы говорите Xcode, что вы хотите.

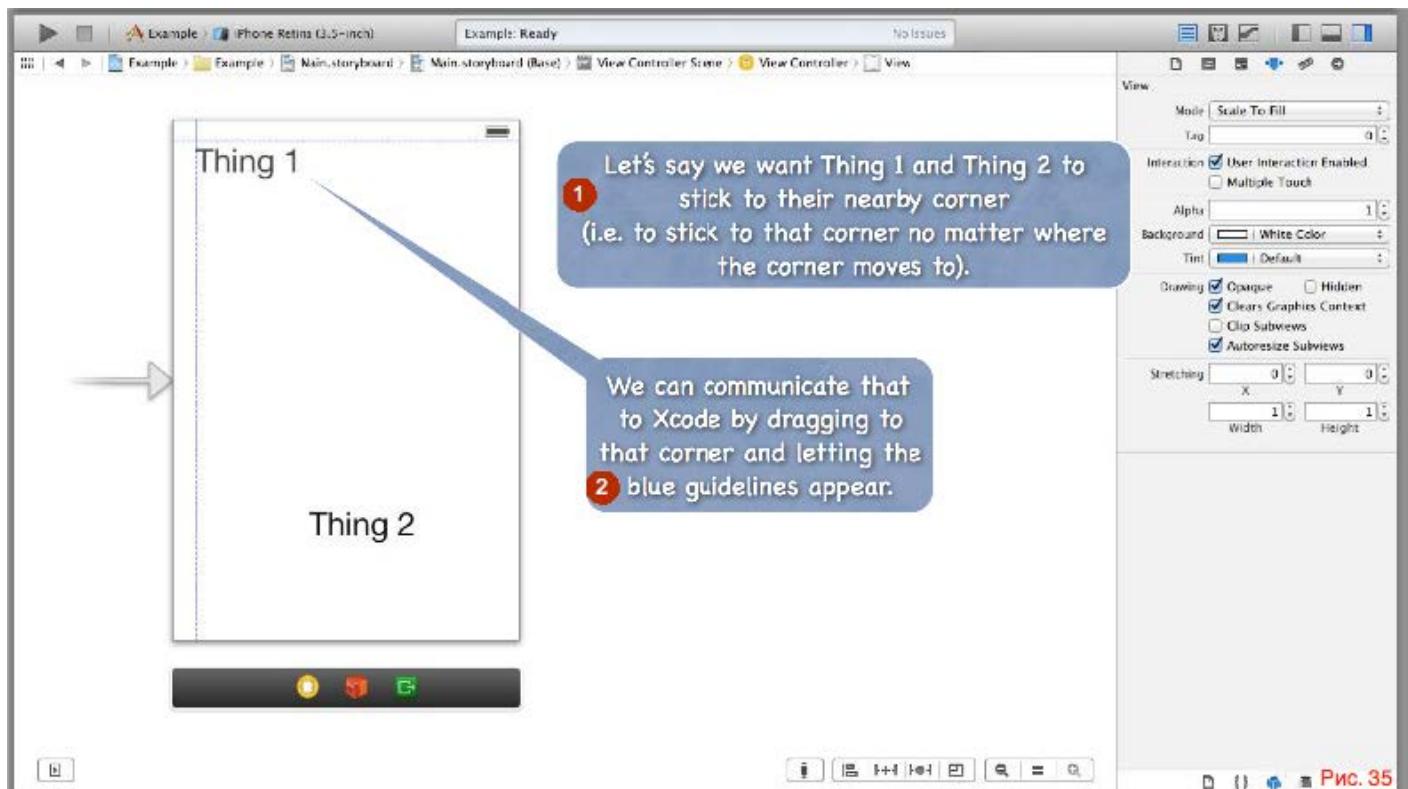
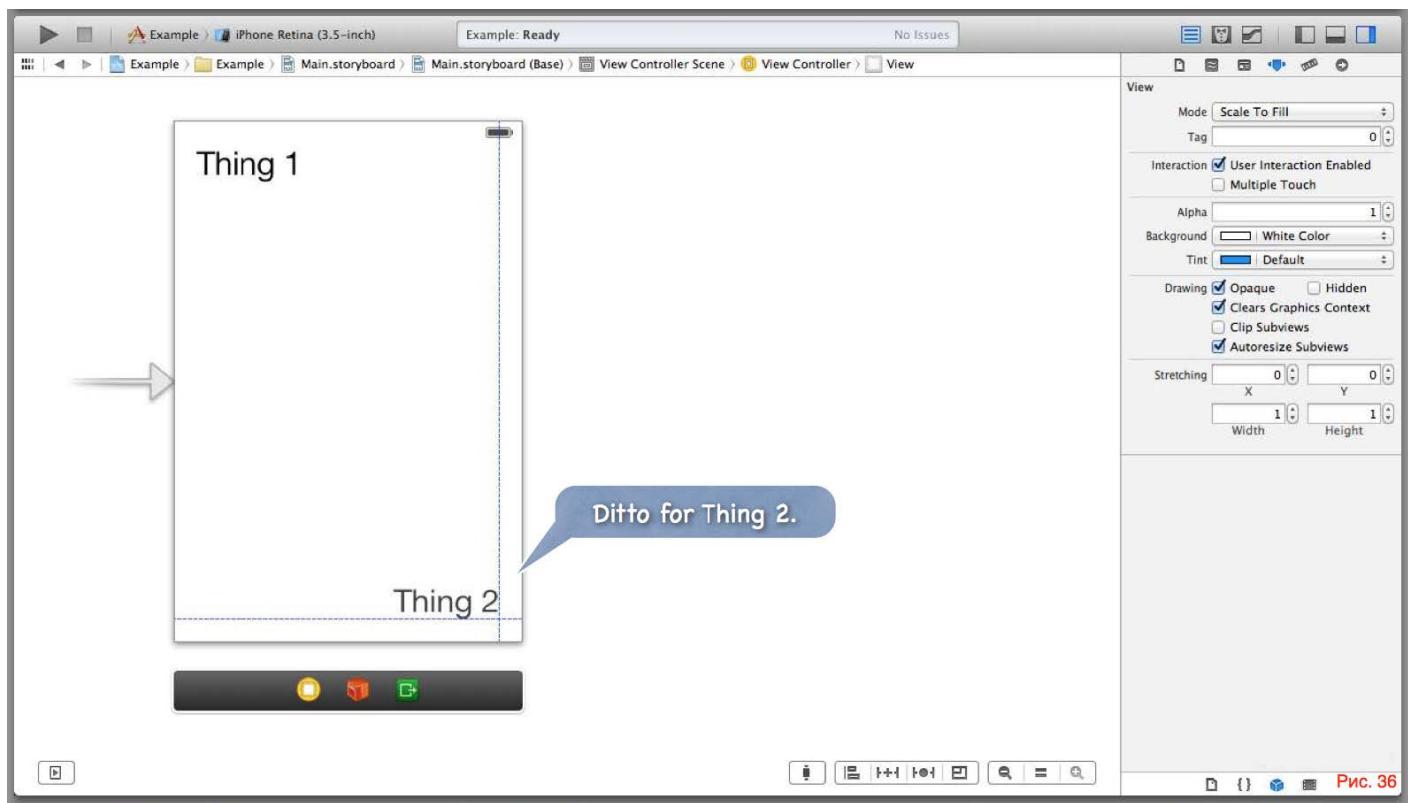
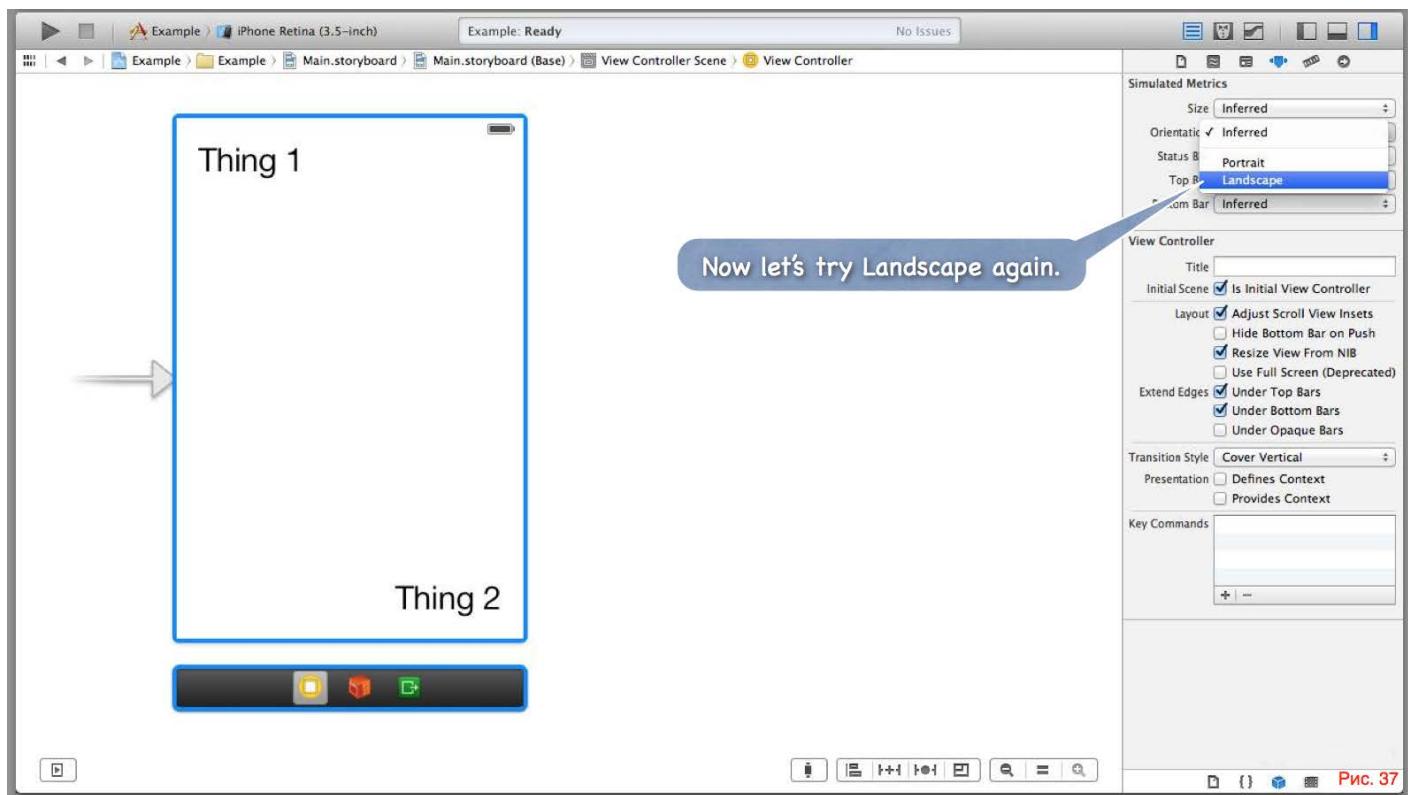


Рис. 35

1. Давайте скажем, что мы хотим, чтобы Thing 1 и Thing 2 были приклейены к своим ближайшим углам (приклейены к этим углам вне зависимости от того, куда эти углы движутся).
2. Мы можем взаимодействовать с Xcode путем перетаскивания этих меток в эти углы до тех пор, пока не появляются голубые линии.

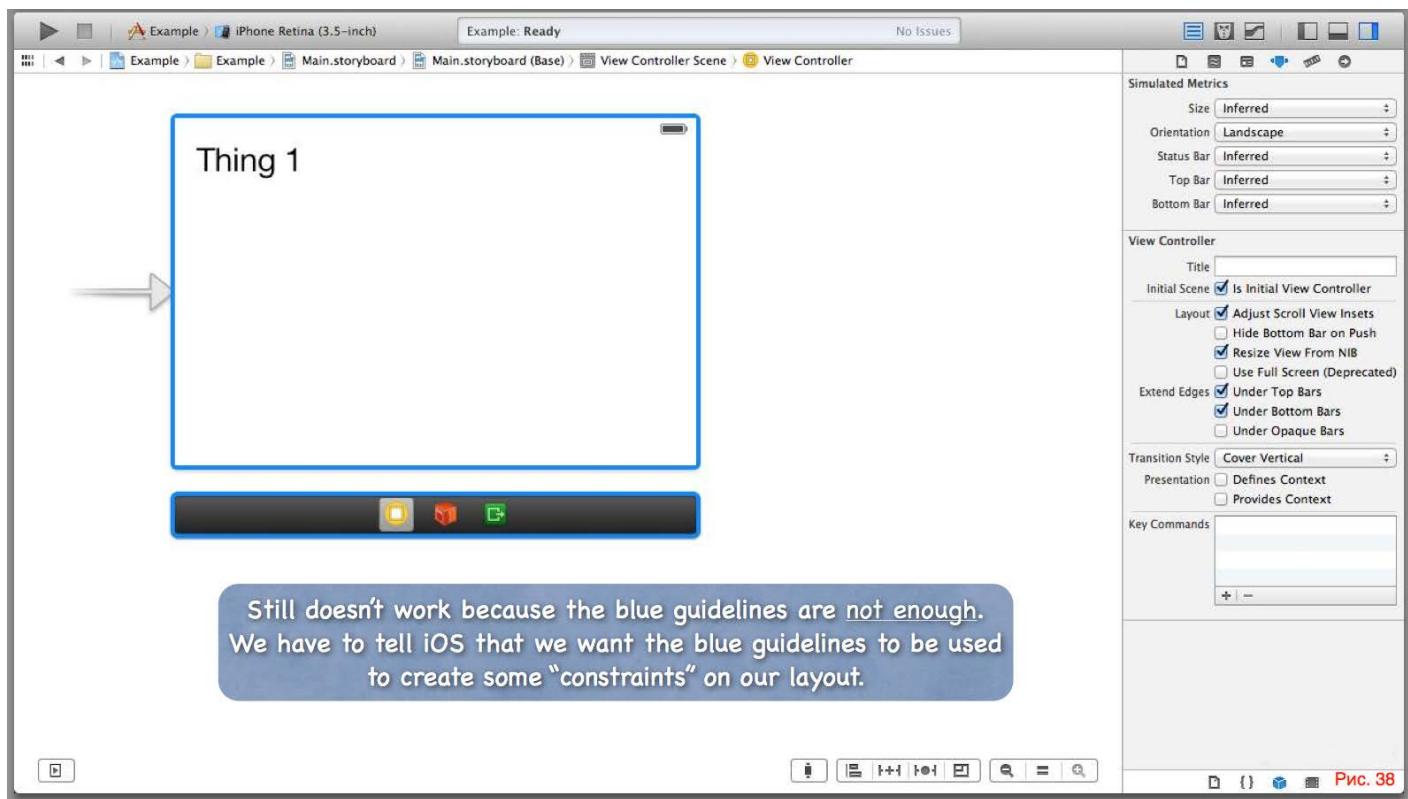


То же самое проделываем с Thing 2.

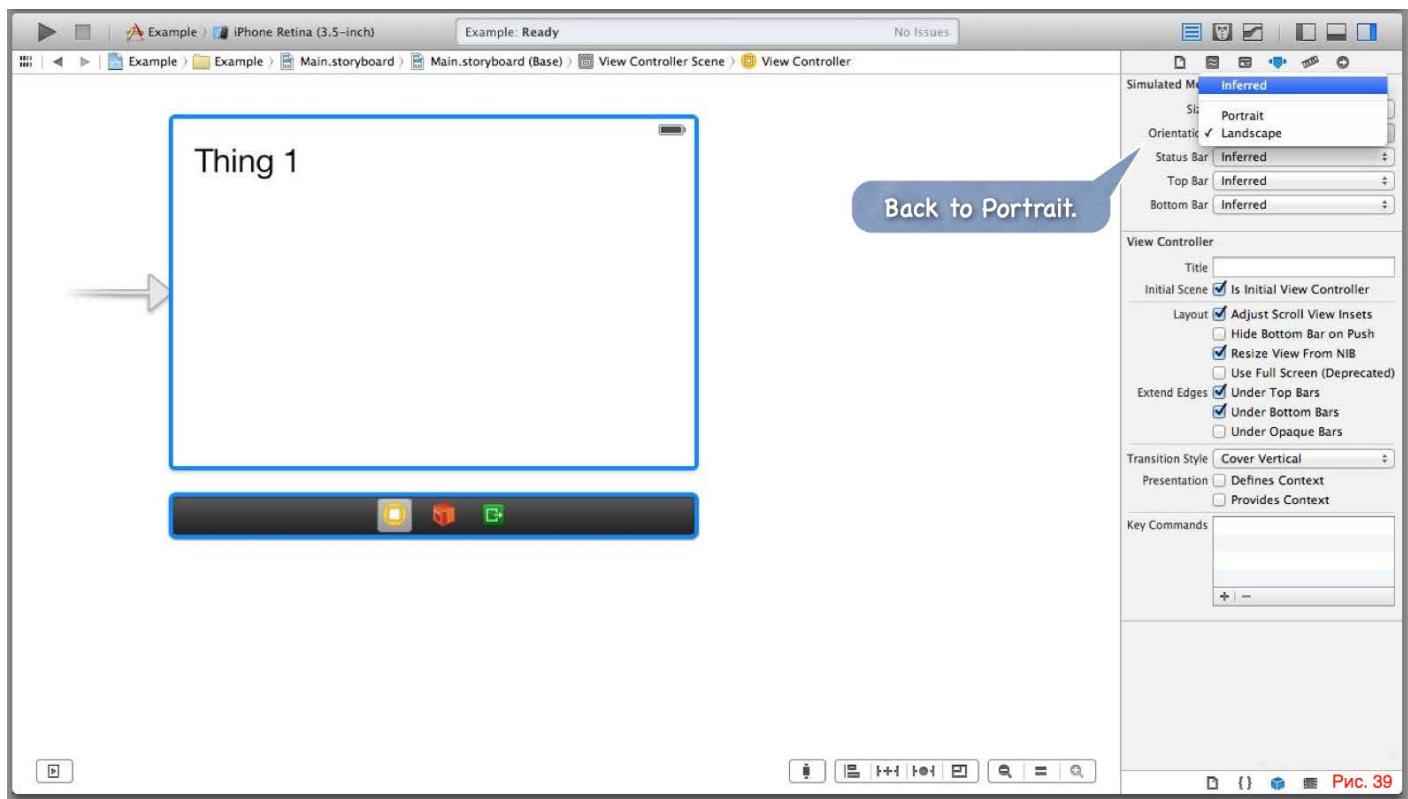


Давайте опять попробуем режим “Ландшафт”.

Рис. 37

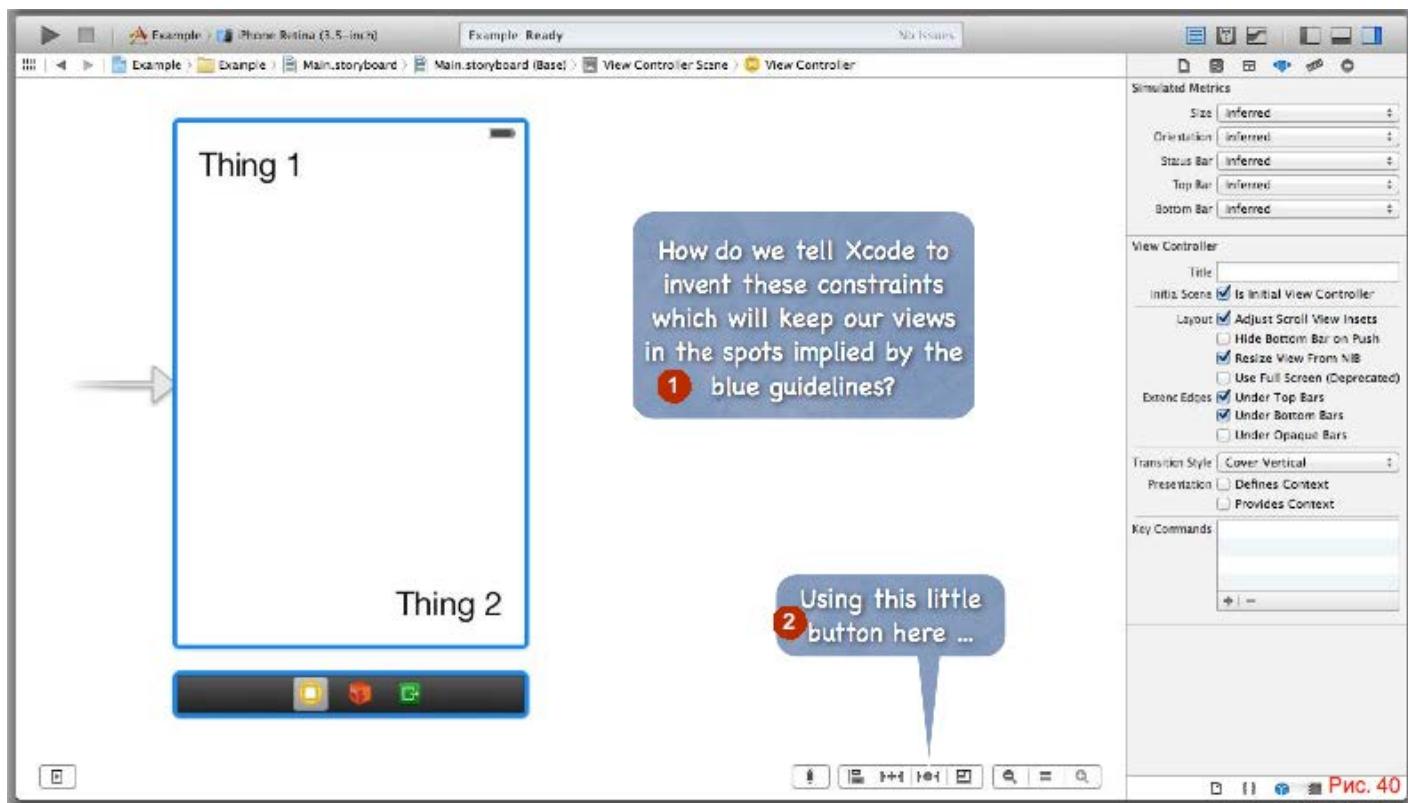


Все еще не работает, потому что голубых направляющих линий не достаточно. Мы должны сказать iOS, что мы хотим использовать голубые направляющие линии для создания некоторых "constraints" (ограничений) для взаимного расположения наших меток.



Вернемся в режим “Портрет”.

Рис. 39



1. Как мы скажем Xcode, чтобы он создал “constraints” (ограничения), которые удерживали бы наши views в местах, помеченных голубыми направляющими линиями?
2. Мы используем эту маленькую кнопочку...

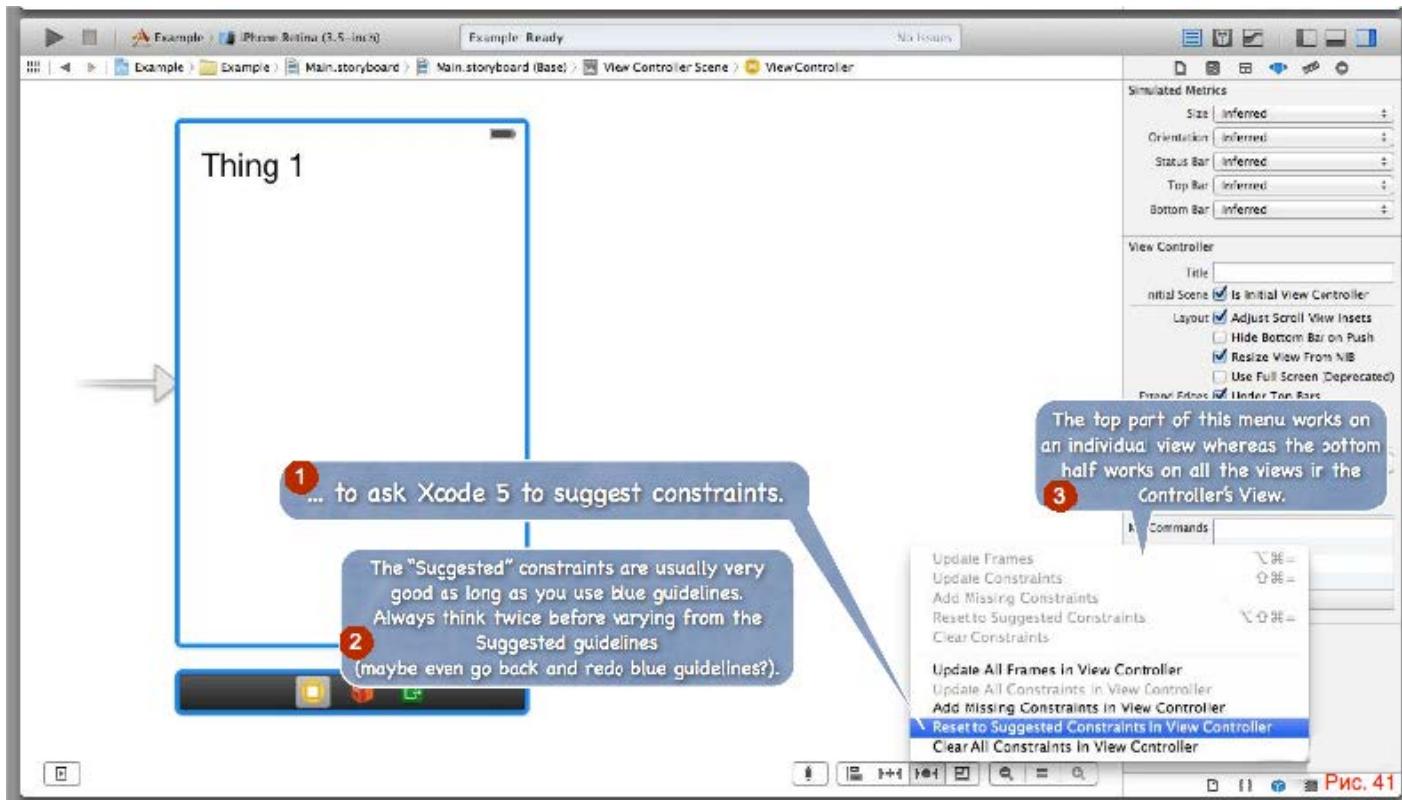
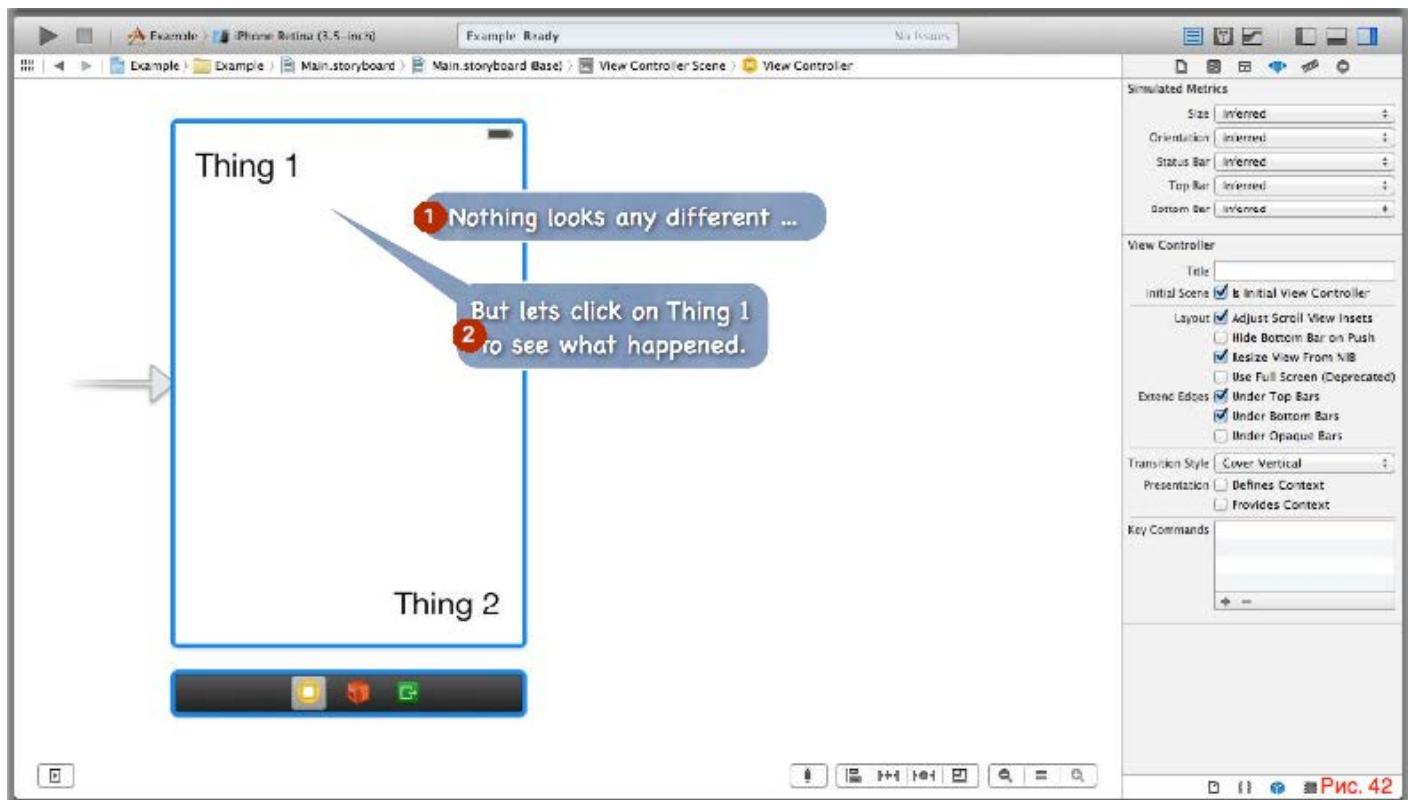


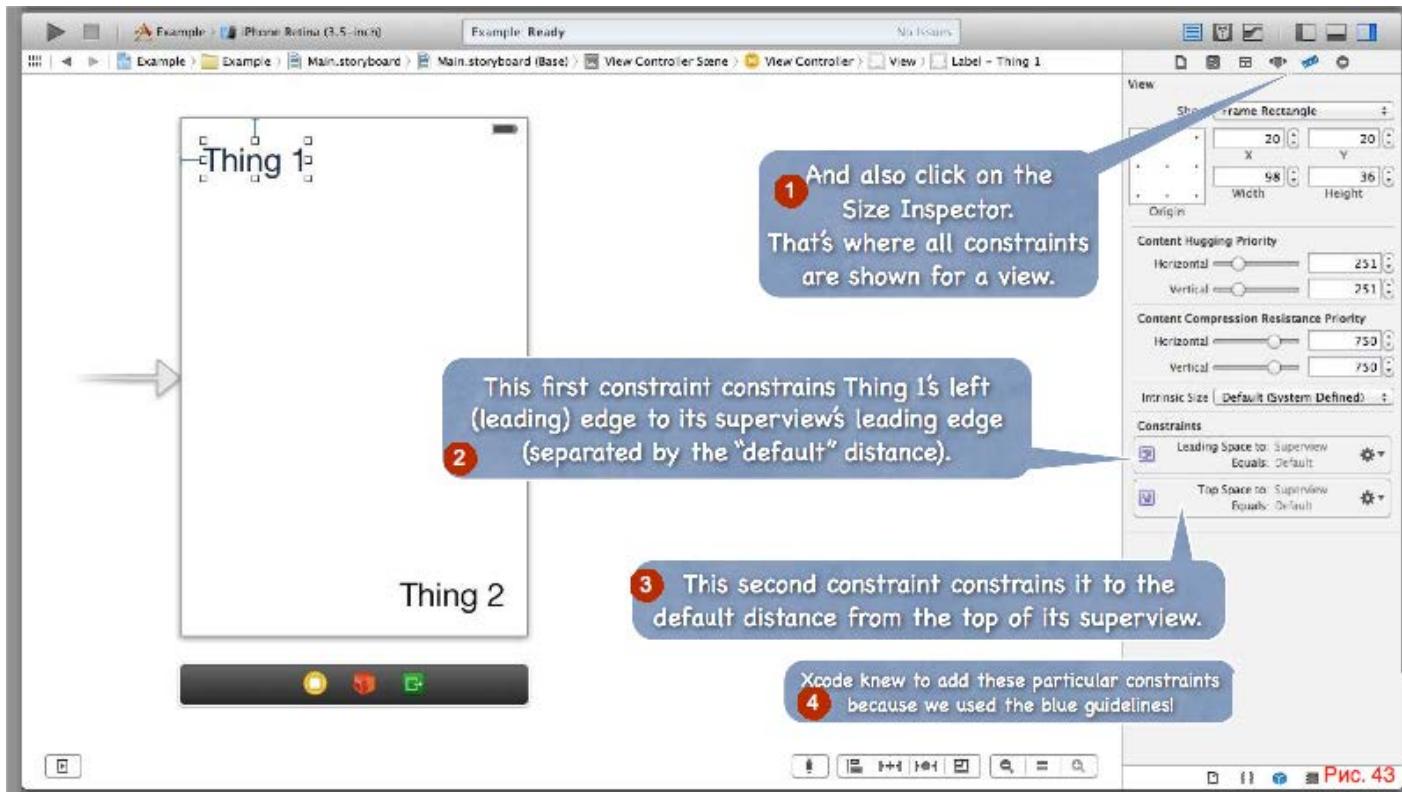
Рис. 41

1. ... попросить Xcode 5 предложить “constraints” (ограничения).
2. Предложенные (“suggested”) “constraints” (ограничения) обычно очень хорошие, если они опираются на голубые направляющие линии. Всегда нужно хорошо подумать, прежде чем изменять такие предложенные ограничения на основе направляющих линий .
3. Верхняя часть этого меню работает с индивидуальными view, в то время как нижняя половина работает со всеми views в ViewController.

Мы пройдем через несколько пунктов этого меню, но самый замечательный пункт - это “Reset to Suggested Constraints in View Controller”. Suggested Constraints означает, что Xcode собирается предложить наилучшие ограничения, и они действительно будут лучшими, если вы использовали голубые направляющие линии. Потому что если вы не будете использовать голубые направляющие линии, Xcode не будет знать что вам предложить. Xcode что-то предложит, но это необязательно будет хорошо. В нашем случае и Thing 1, и Thing 2 имеют голубые направляющие линии, поэтому когда мы используем “Reset to Suggested Constraints in View Controller”, это должно работать.



1. Ничего не выглядит иначе...
2. Но давайте кликнем на Thing 1, чтобы посмотреть, что происходит.



1. Кликнем также Инспектор Размера. Вот где все constraints (ограничения) показываются для view
2. Первое ограничение ограничивает Thing 1 левый (передний) край по отношению к переднему краю superview (разделяя их расстоянием “по умолчанию”).
3. Второе ограничение ограничивает Thing 1 расстоянием “по умолчанию” от верхнего края superview.
4. Xcode знает, что нужно добавить именно эти специфические ограничения, потому что мы использовали голубые направляющие линии.

Заметим, что первое ограничение называется **Leading Space**, а не Left Space, потому что в некоторых языках Leading означает правый (справа). Например, язык Иврит или ему подобные leading интерпретируют как правый. Если вы строите весь пользовательский интерфейс на Иврите, то весь ваш текст будет писаться справа налево, и **Leading Space** (лидирующее пространство) будет расположено справа. В любом случае, ограничение **Leading Space**, какую бы сторону это не означало, говорит о том, что мы хотим, чтобы наш view был привязан к superview, а точнее к краю нашего superview, и находился от него на некотором расстоянии, величина которого

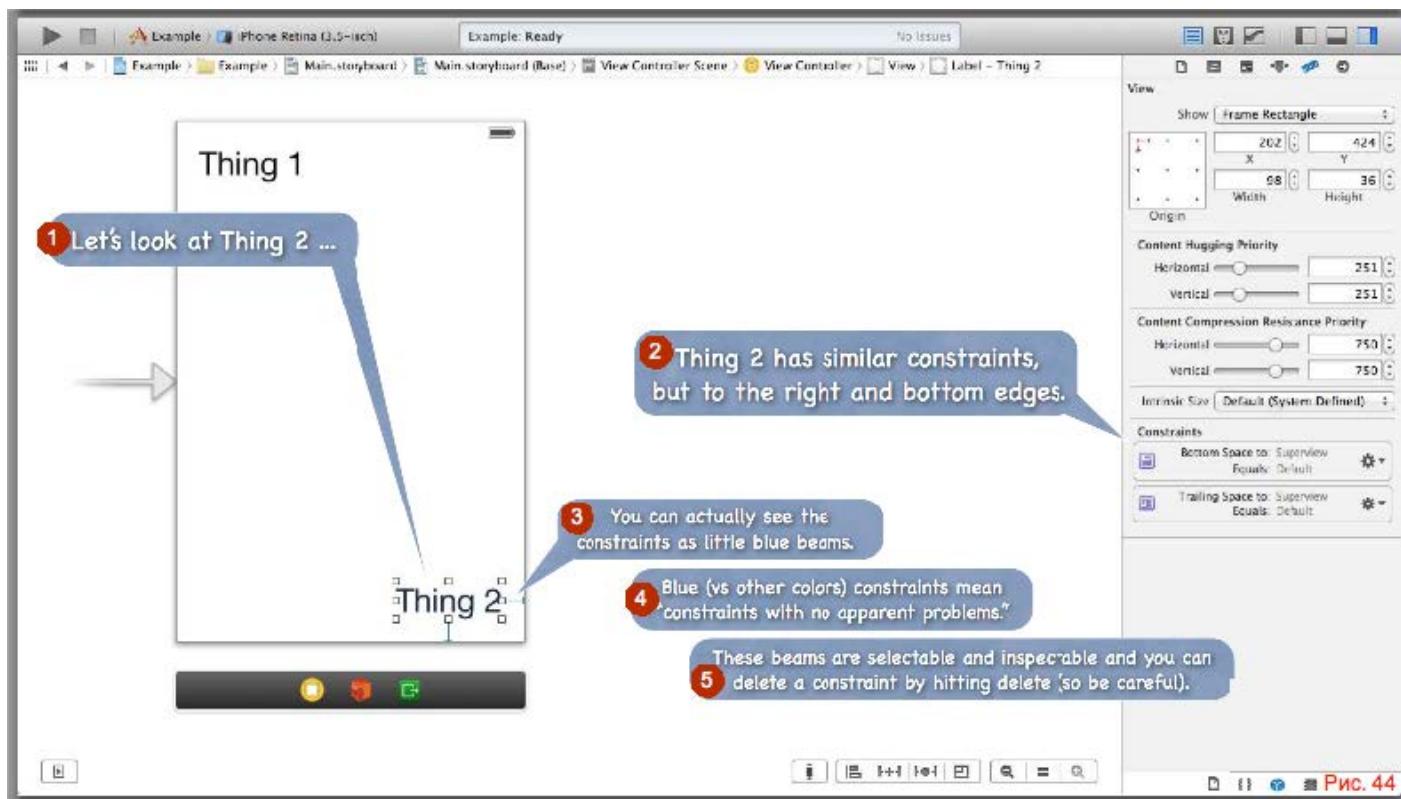
составляет значение “по умолчанию”. Это не какое-то “магическое” число, это значение “по умолчанию”, которое назначается самой системой.

Вторым ограничением является **Top Space**, которое привязано к верхнему краю вашего superview с расстояние “по умолчанию”.

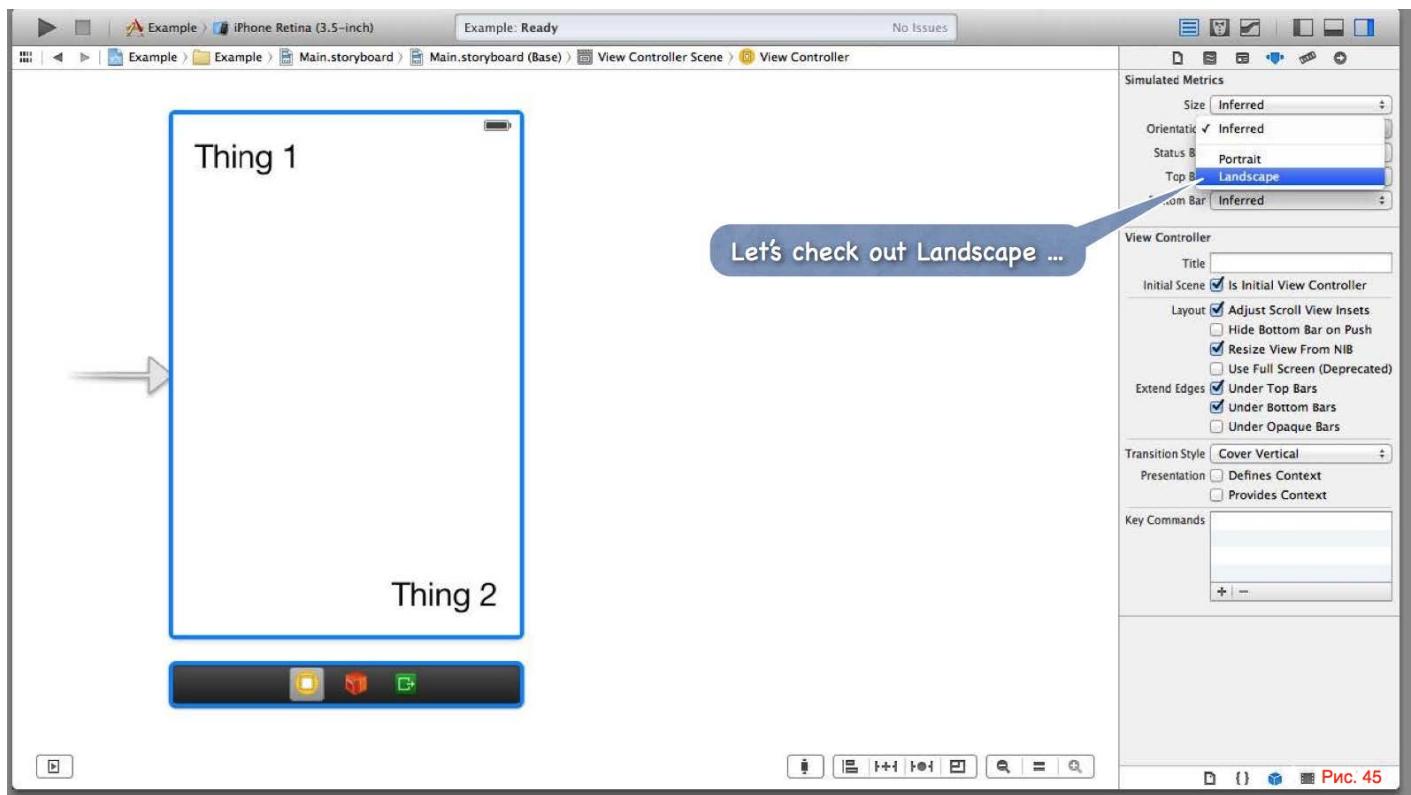
Два этих ограничения точно определяют, где будет располагаться верхний левый угол Thing 1 (то есть origin) на superview. А как насчет размеров нашей метки? Будет ли эта метка всегда одного и того же размера? Нет. Такие views, как метки, кнопки имеют свойство называемое *intrinsic size* (внутренний размер). Это размер, который определяется текстом метки или заголовком кнопки, и он может меняться, если вы меняете текст в метке или в заголовке кнопки. Если текст меняется, то вам необходимо, чтобы метка была шире или уже. Поэтому вы не можете наложить Autolayout ограничение на размер метки или кнопки. Если вы наложите эти ограничения, то метка застрянет на этом размере, что очень плохо.

Мы не должны накладывать ограничения на размер в общем случае на такие views, которые имеют свойство *intrinsic size*, то есть метки и кнопки. Итак, Thing 1 имеет два ограничения, которые полностью определяют его местоположение вне зависимости от того, что представляет собой superview.

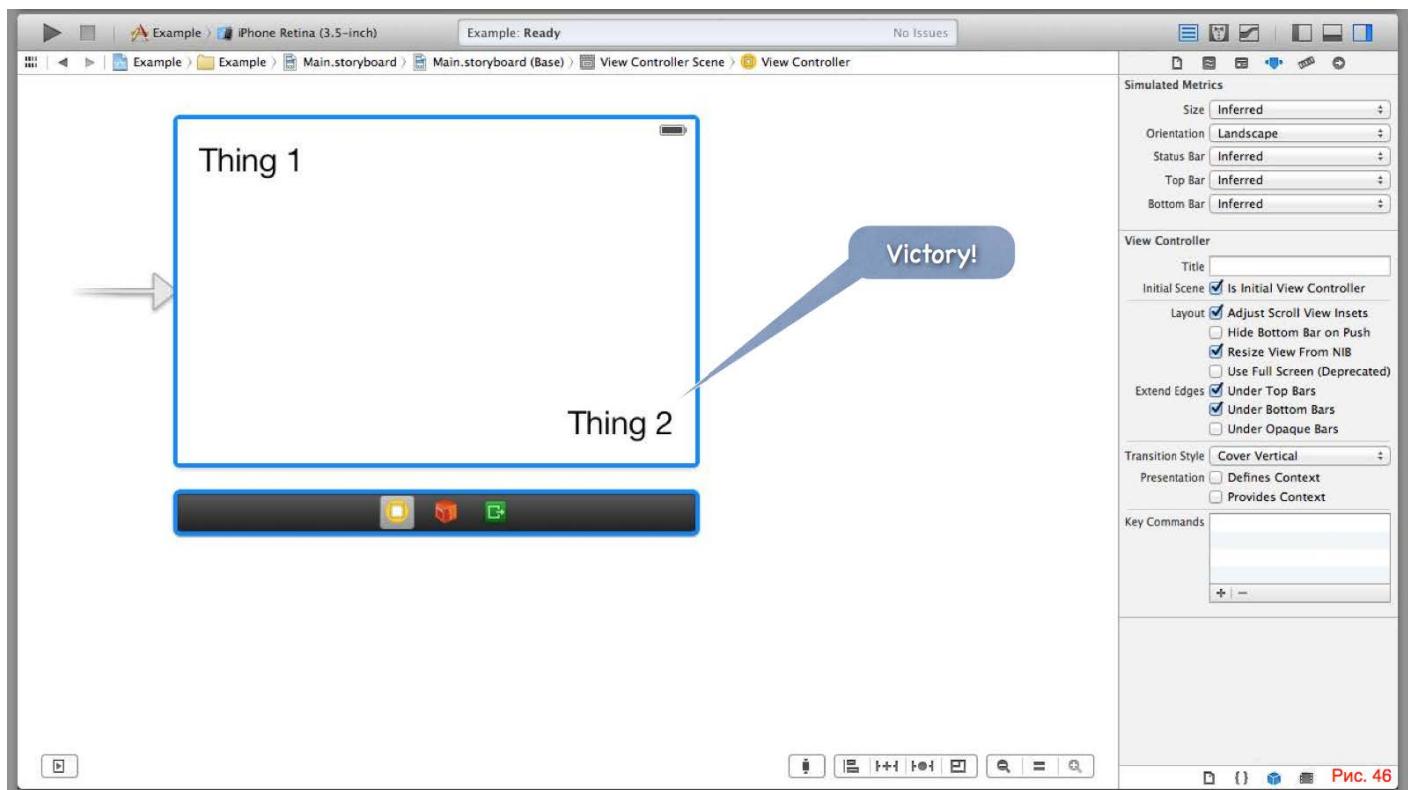
Схожая ситуация складывается с Thing 2.



1. Давайте посмотрим на Thing 2...
2. У Thing 2 похожие ограничения, но с правого и нижнего краев.
3. В действительности вы можете видеть ограничения в виде голубых “лучей”.
4. Голубая окраска (в противоположность другим цветам) ограничений означает “ограничения без явных проблем”.
5. Эти “лучи” можно выбрать и инспектировать и можно даже удалить ограничение, нажав “delete”.



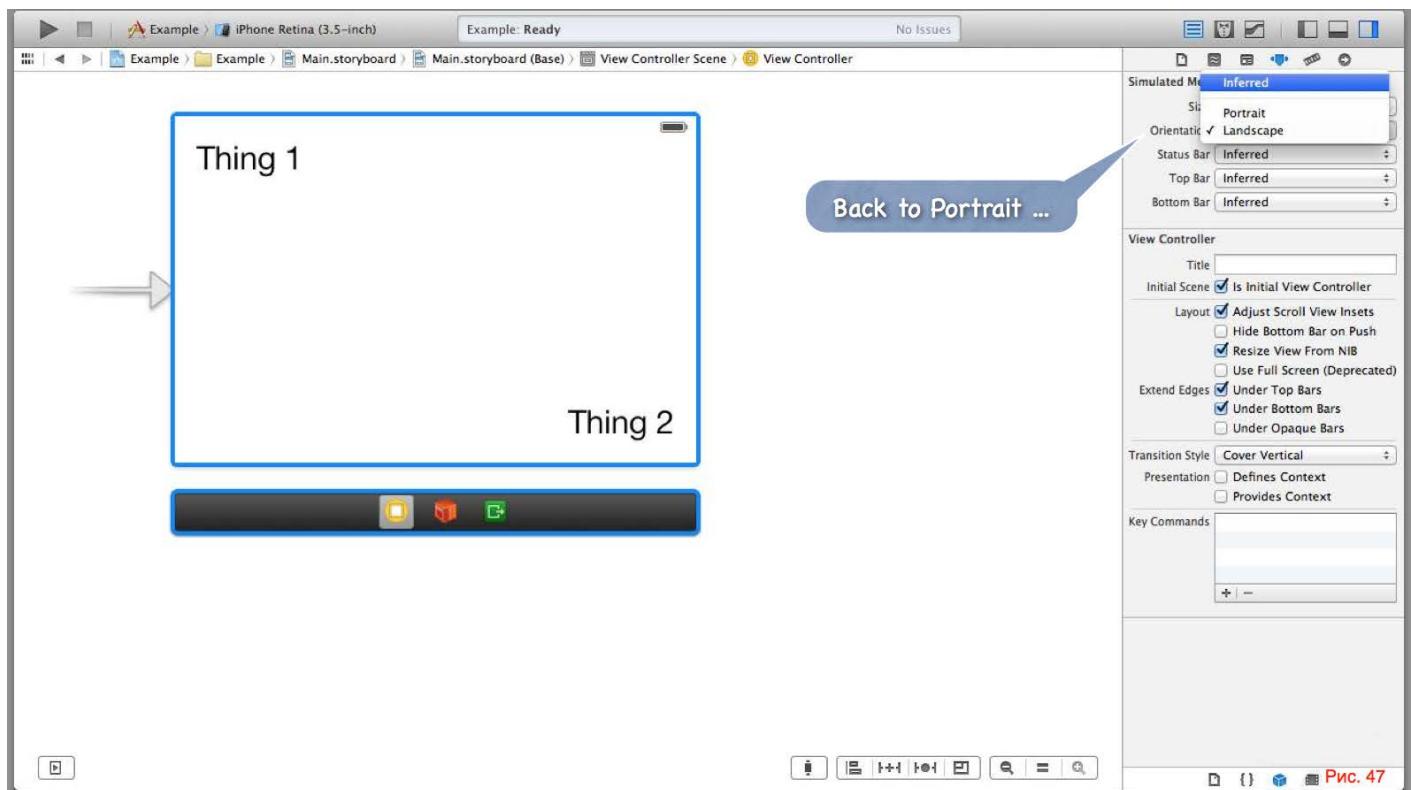
Давайте проверим наш режим “Ландшафт”...



Победа!

Вы видите, что метки приклеились к углам. Это именно то, что нам нужно. И все это благодаря полученным правилам - ограничениям.

Рис. 46



Вернемся в режим “Портрет”...

Рис. 47

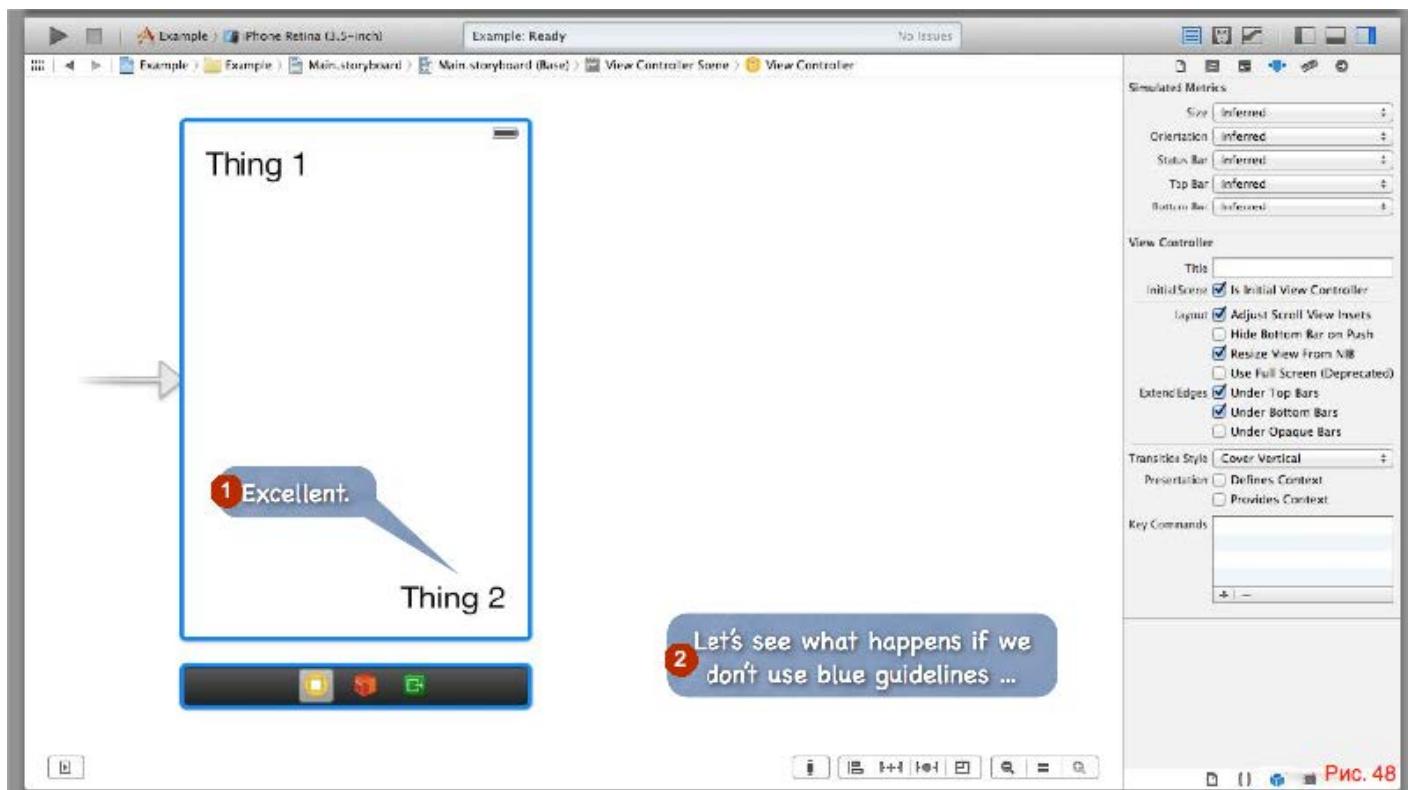
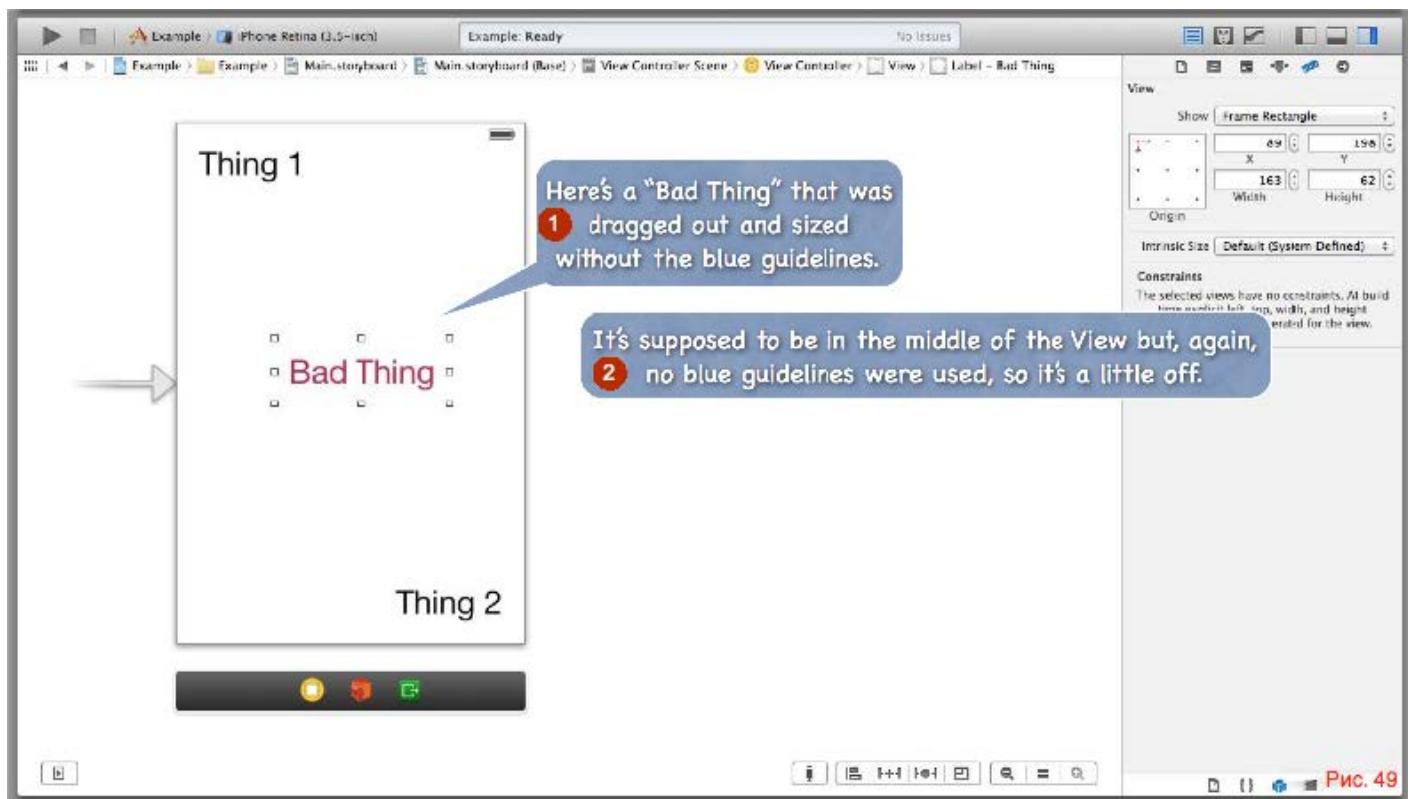


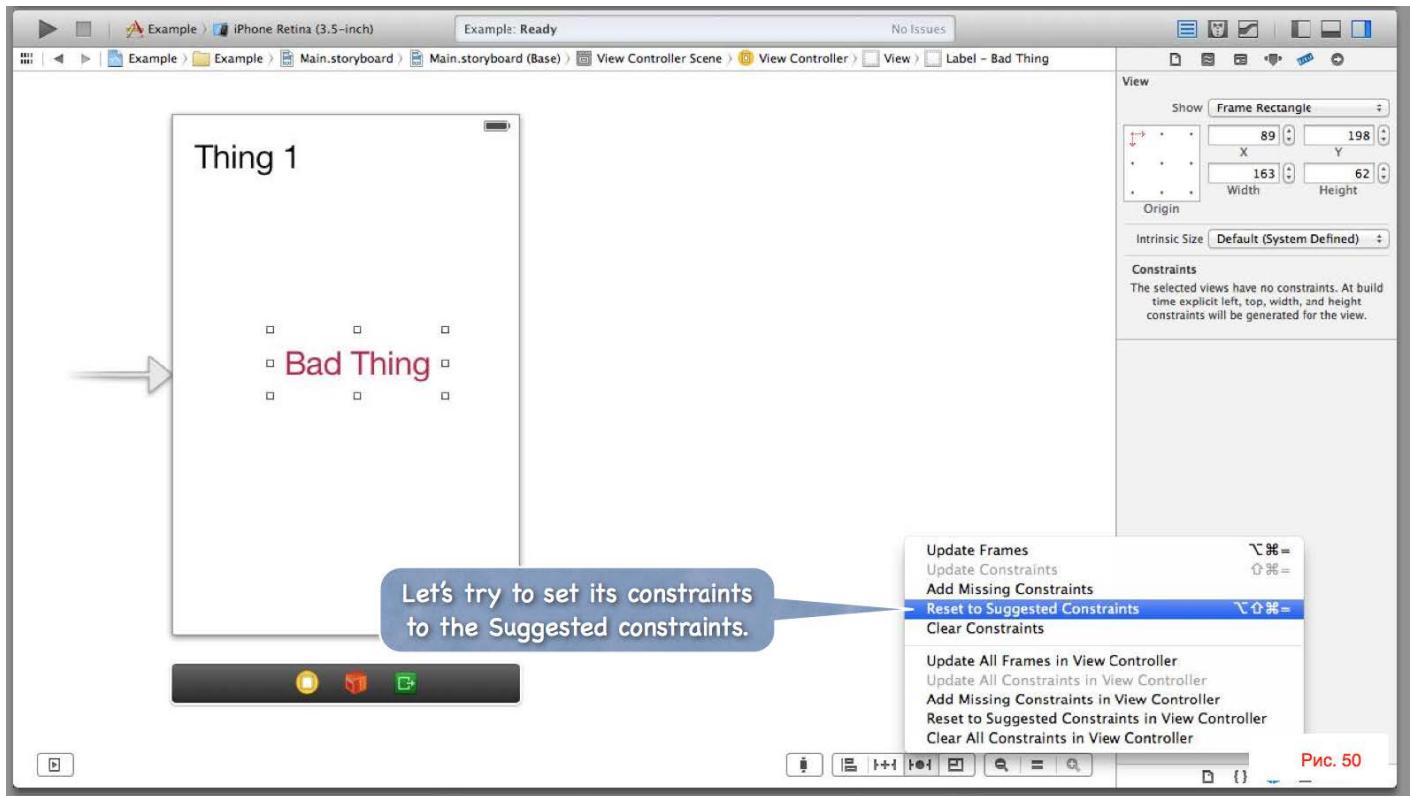
Рис. 48

1. Замечательно.
2. Давайте посмотрим, что будет происходить, если вы не будете использовать голубые направляющие линии...



1. Это метка “Bad Thing”, которую перетащили из палитры объектов и изменили размер без направляющих голубых линий.
2. Предполагалось, что метку разместили в центре view, но, опять, это было сделано без голубых направляющих линий, и, следовательно, есть небольшое смещение от центра.

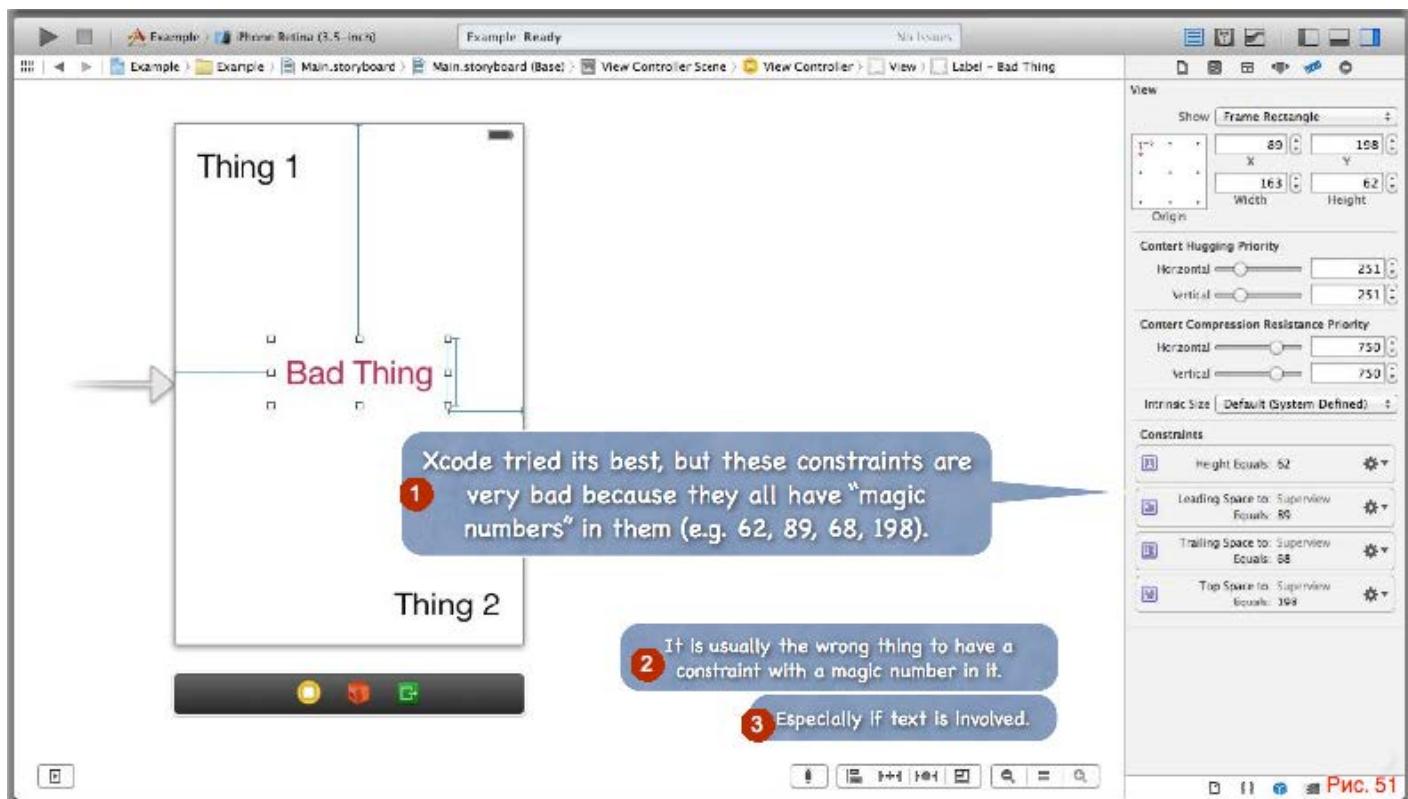
Мы изменем размер метки “Bad Thing”, так как он сделан слишком большим для нее, с помощью маленьких квадратиков для управления размером. Согласно свойству *intrinsic size* этот размер должен быть много меньше. У этой метки нет ограничений, так как мы только что перетащили ее из палитры объектов.



Давайте попробуем установить Suggested (предлагаемые) ограничения для метки.

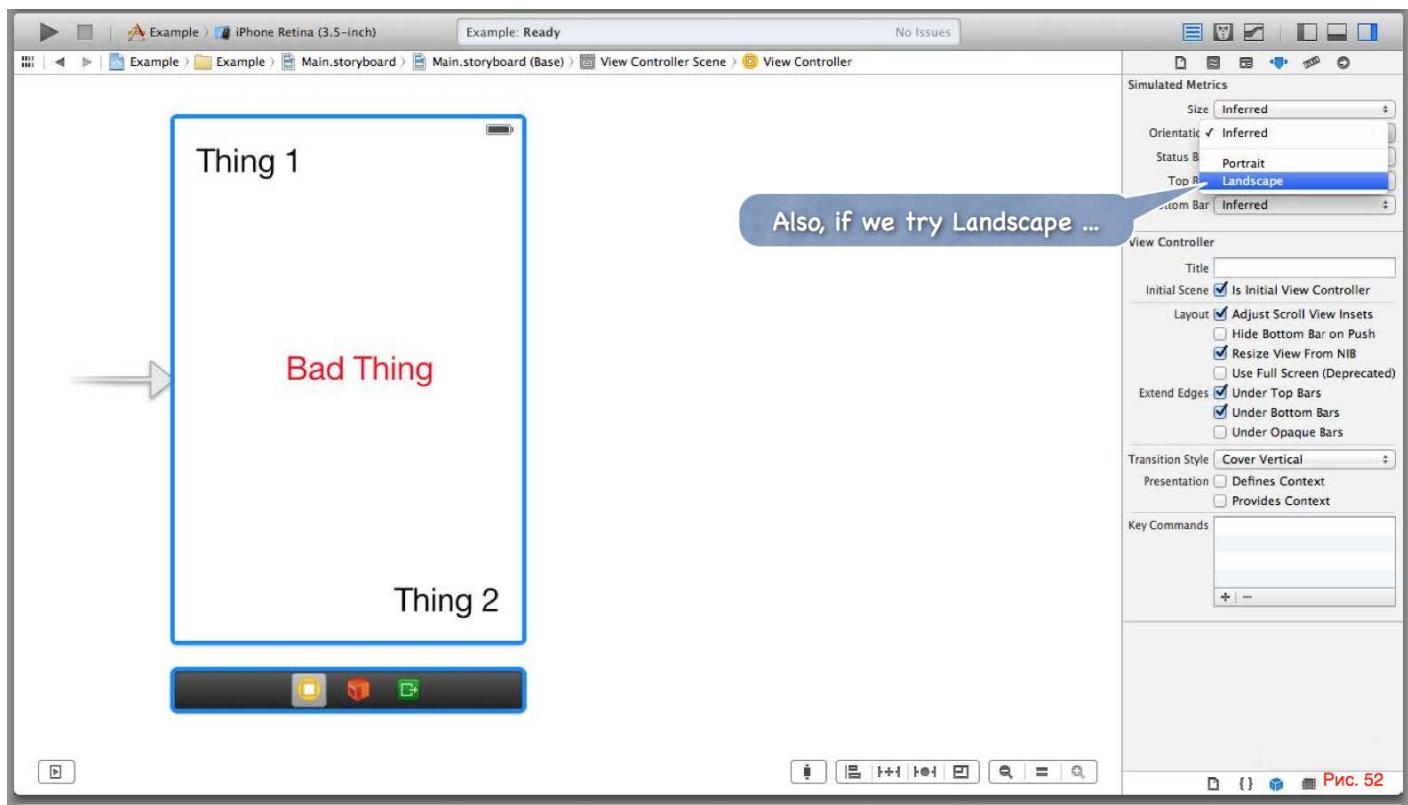
Помните? Я не использовал голубые направляющие линии при перетаскивании. Заметьте, я использую верхнюю часть меню, которая действует только для метки "Bad Thing", я не переустанавливаю ограничения для Thing 1 и Thing 2, только для метки "Bad Thing".

Рис. 50

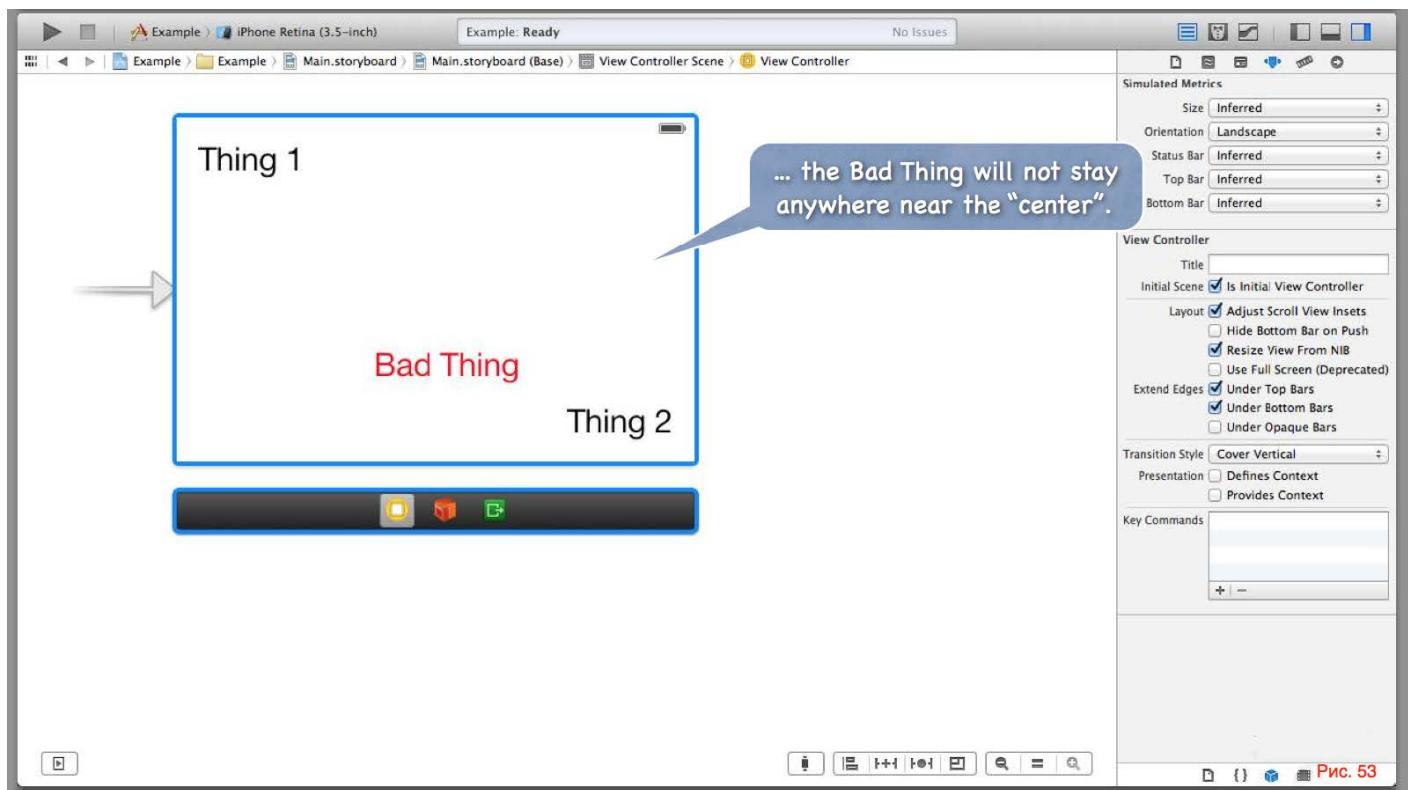


1. Xcode делает все, что возможно, но получаются очень плохие ограничения, потому что они содержат “магические числа” (62, 89, 68, 198).
2. Обычно очень плохо иметь ограничения с “магическими числами”.
3. Особенно, если в это вовлечен текст.

Ограничения с “магическими числами” выглядят ужасно, есть всего несколько случаев, когда это допустимо, но в большинстве случаев это ужасно.

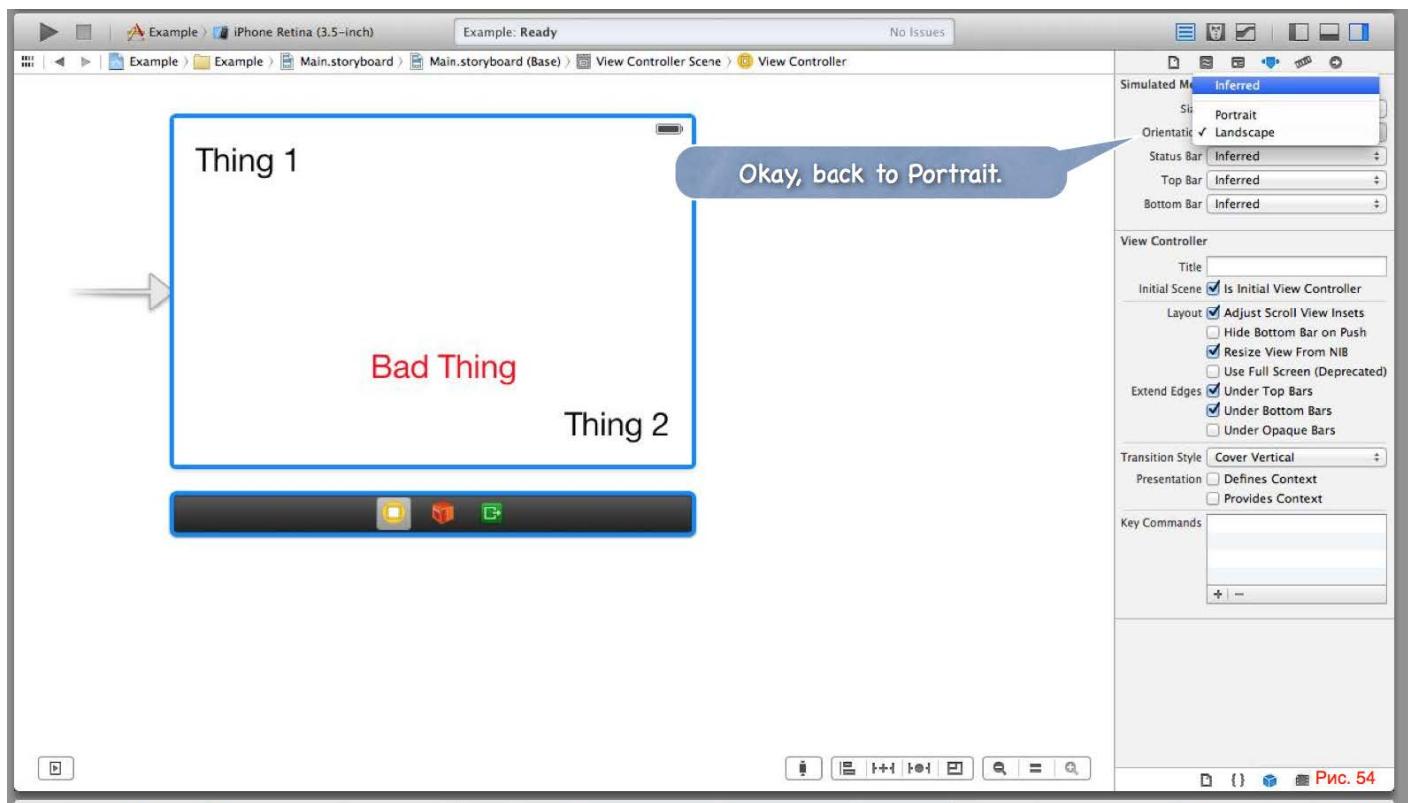


И действительно, если мы попытаемся переключиться в режим “Ландшафт”...



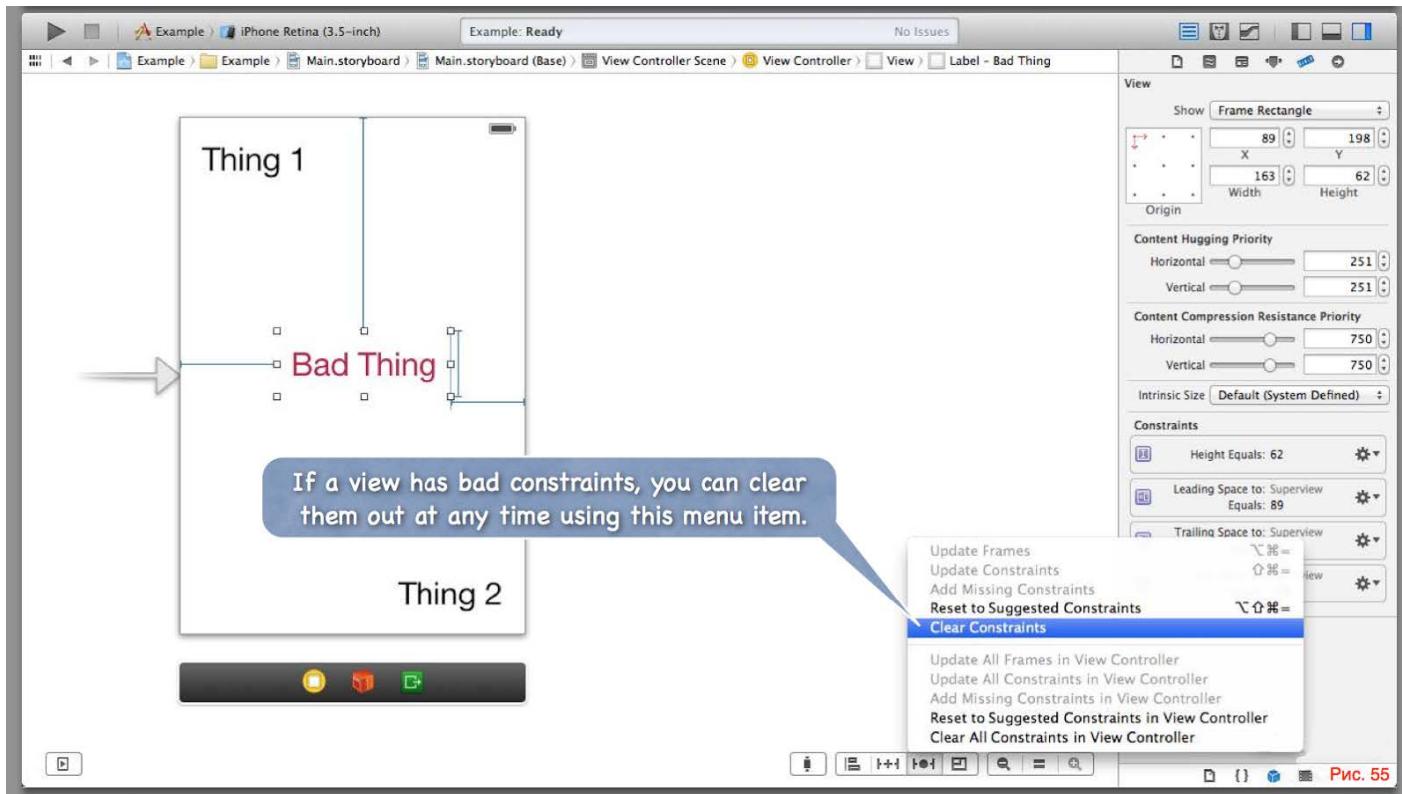
... “Bad Thing” не останется поблизости от центра.

Потому что остались те же “магические числа” (198 pixels сверху и т.д.). Это совсем не то, что нам нужно: мы пытались поместить метку “Bad Thing” в центр экрана, но у нас ничего не вышло. Таким образом, Suggested constraints (предлагаемые ограничения) не работают, если мы не используем голубые направляющие линии, так как Xcode не понимает, что нам нужно. Давайте избавимся от этих ограничений.



Вернемся в режим “Портрет”.

Рис. 54



Если view имеет плохие ограничения, мы можем их удалить в любое время в том же меню.

Мы идем в то же меню и выбираем другой элемент, который очистит все ограничения для этого view.

Заметьте, что все ограничения нарисованы - вы можете видеть их в виде голубых линий. Вы можете кликнуть на них и редактировать их. Большинство ограничений имеет атрибуты, для обсуждения которых у нас, к сожалению, нет времени, но ограничения (constraints) являются объектами первого класса на вашей storyboard.

Итак, очищаем предыдущие ограничения “Bad Thing” и разместим новые, но совершенно другим способом. Это будет способ № 2 добавления constraints (ограничений). **Способ № 1** - это использовать голубые направляющие линии и запросить предлагаемые (suggested) constraints (ограничения).

**Способ №2** - это использовать некоторые кнопки внизу экрана

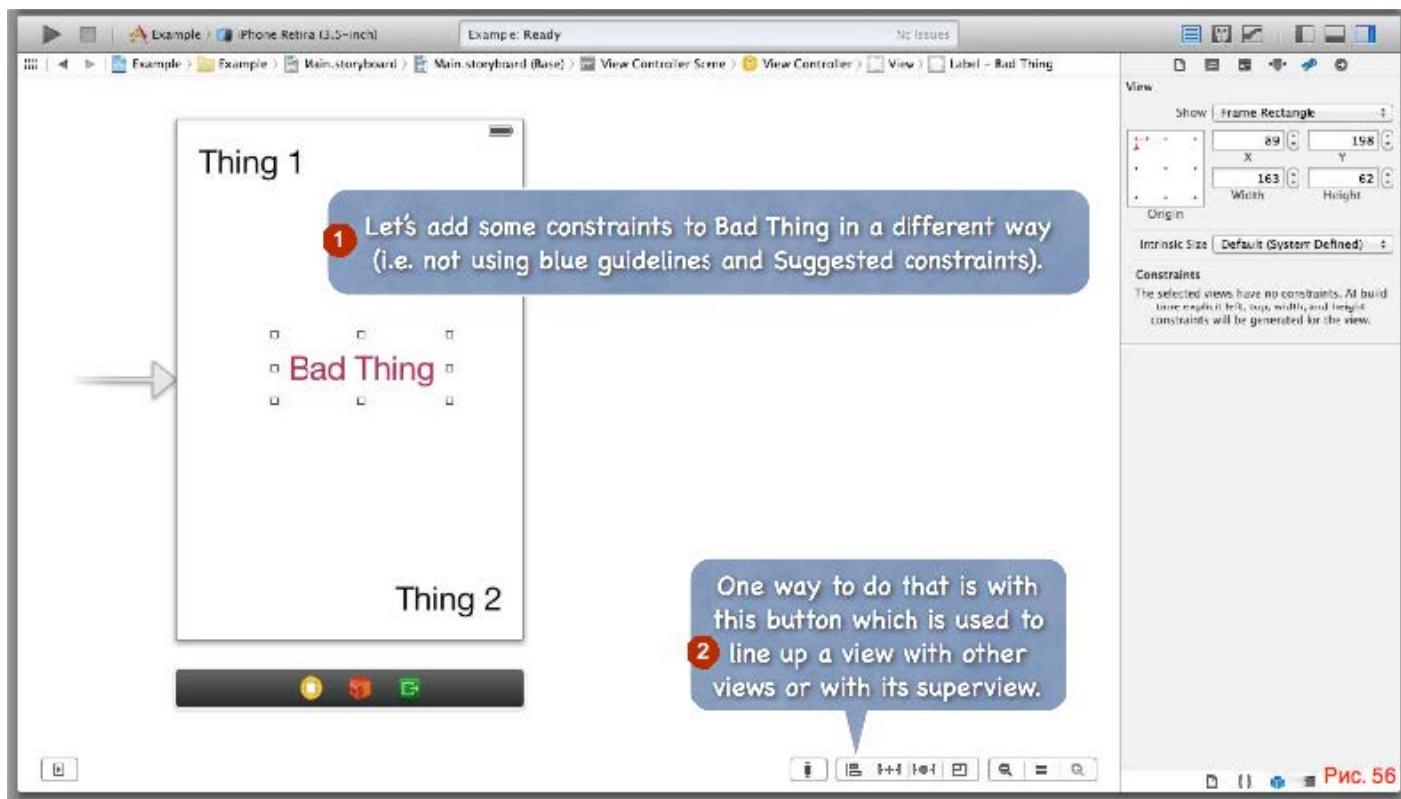
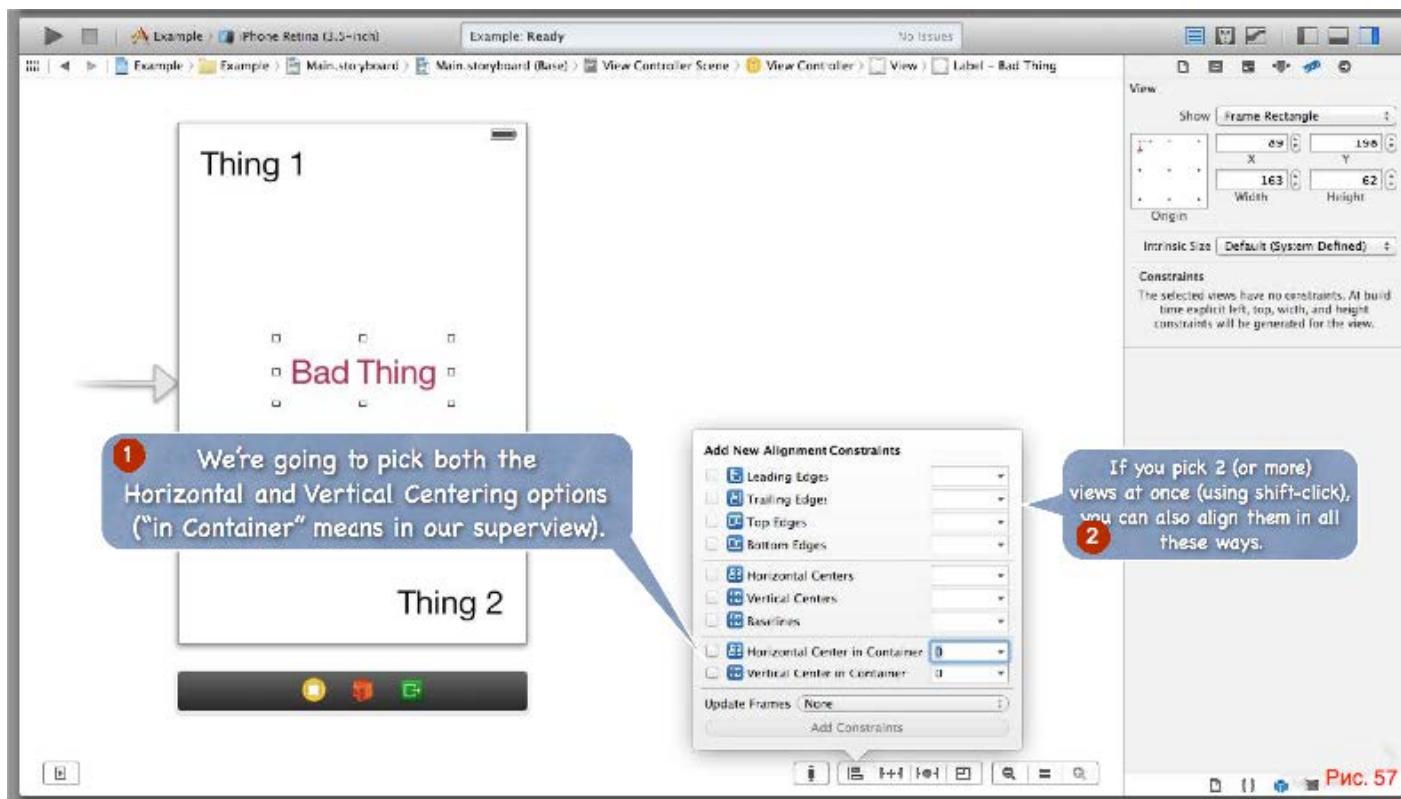


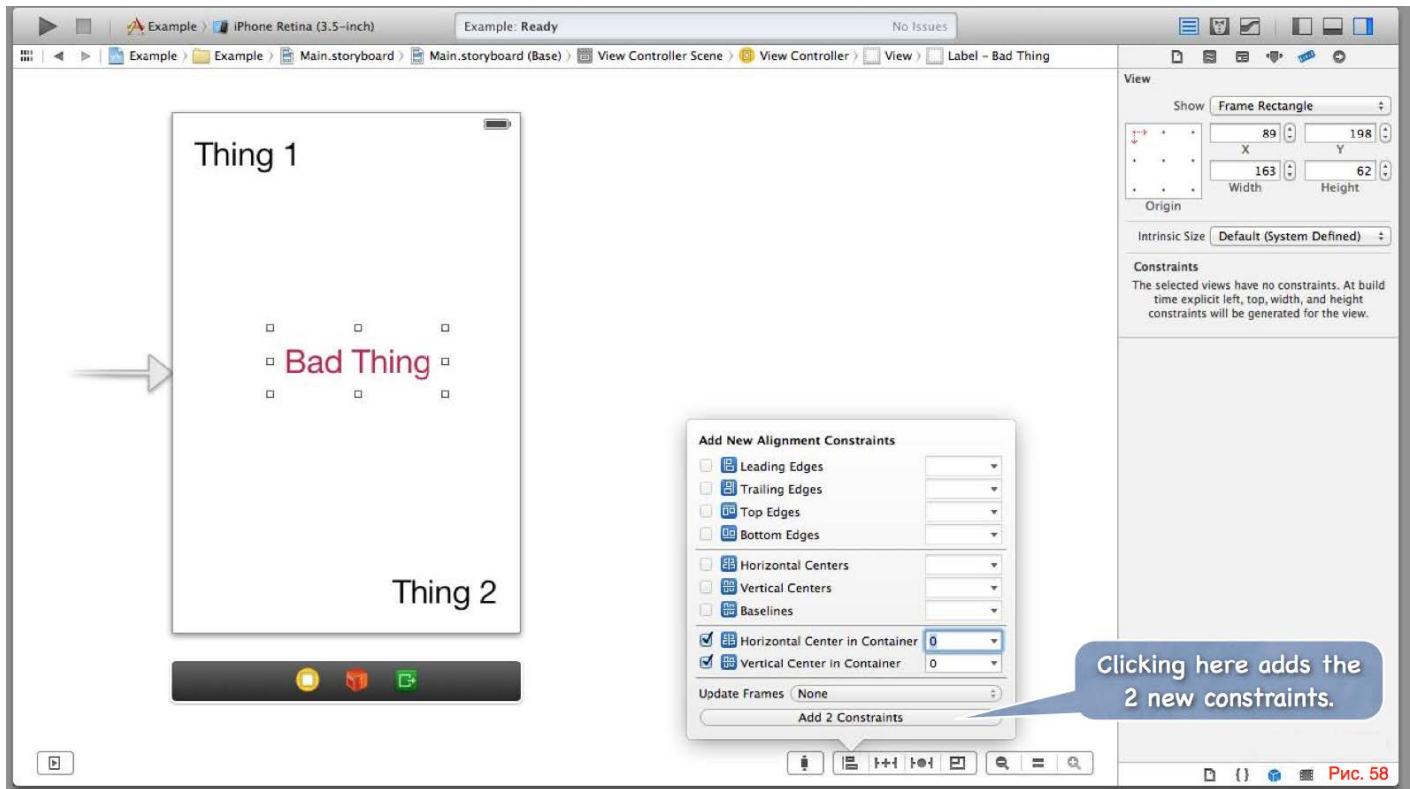
Рис. 56

1. Давайте добавим некоторые constraints (ограничения) другим способом ( не используя голубые направляющие линии и Suggested constraints).
2. Один из этих других способов - использовать кнопки для выравнивания view с другим view или с его superview.



1. Мы выберем обе опции: Horizontal и Vertical Center (“In Container” означает в вашем superview).
2. Если вы выберите 2 (или более) views одновременно (используя SHIFT-click), вы также сможете выравнивать их любым указанным здесь образом.

Мы выбираем две нижние опции, которые выравнивают view по горизонтальному и вертикальному центру в superview, и добавляем два новых ограничения: одно по горизонтали, другое - по вертикале.



Кликаем здесь, чтобы добавить 2 новых ограничения.

Мы также можем сместить view относительно центра : для этого надо указать числа справа от выбранных опций. Но в данный момент мы хотим расположить нашу метку точно по центру, поэтому оставляем предложенные нули.

Над кнопкой “Add 2 constraints” расположено поле “Update Frames” с выпадающим меню, и вы можете запросить перемещение вашего view с тем, чтобы удовлетворить добавленным ограничениям. Но мы не будем заставлять метку “Bad Thing” перемещаться, чтобы показать вам, что происходит, если вы этого не делаете.

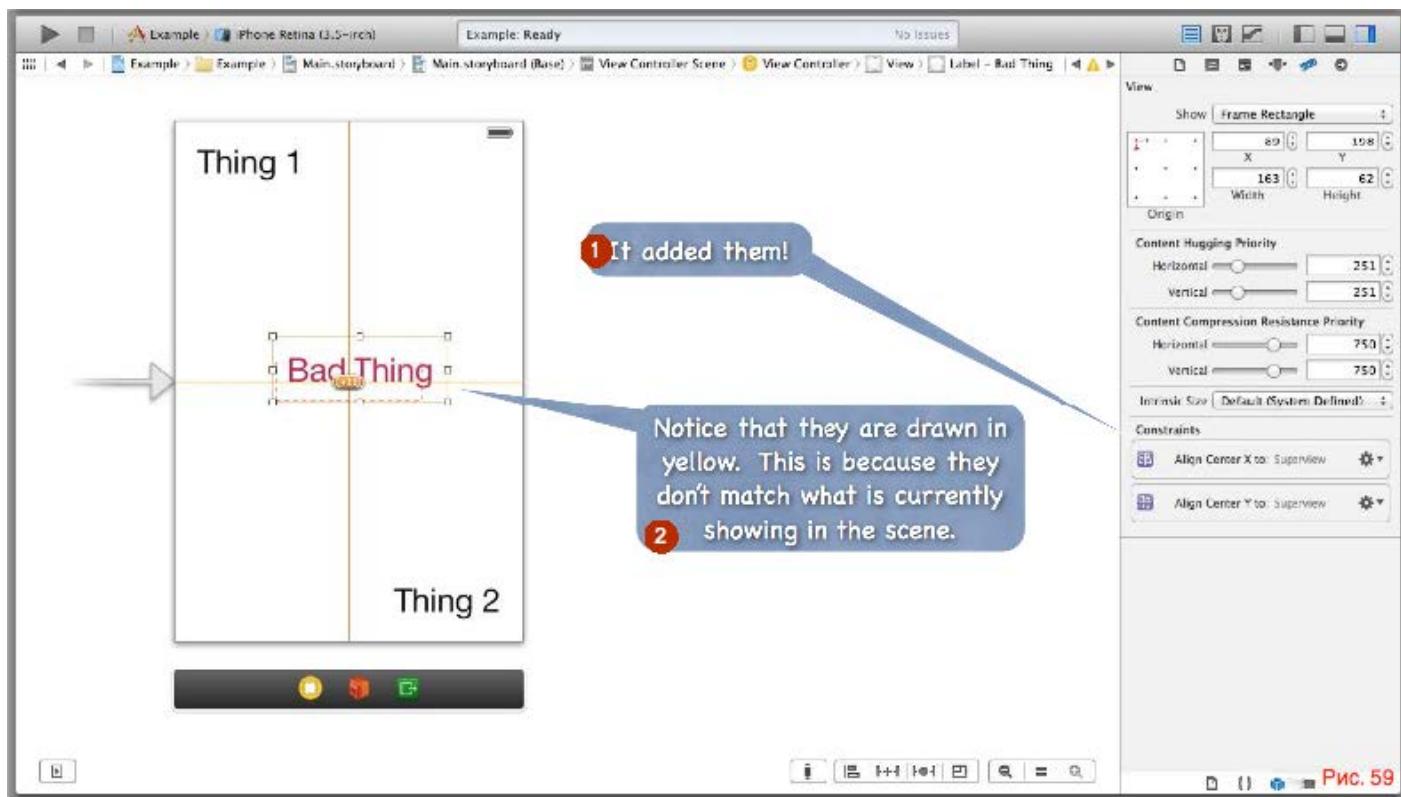


Рис. 59

1. 2 ограничения добавлены.
2. Заметьте, что они нарисованы желтым цветом. Потому что они не подходят тем, которые в данный момент находятся на экране.

Итак, добавлены 2 ограничения: одно Align Center X- для центрирования по горизонтали , другое Align Center Y- для центрирования по вертикале. Вы видите их на слайде. На экране ограничения показаны желтым цветом, что не очень хорошо - это как бы предупреждение. Голубой цвет - хорошо, желтый - предупреждение, красный - очень плохо, потому что есть конфликт. Похоже на то, что у вас есть два ограничения, которые задают противоположные вещи. И мы поговорим о том, как разрешить эти противоречия.

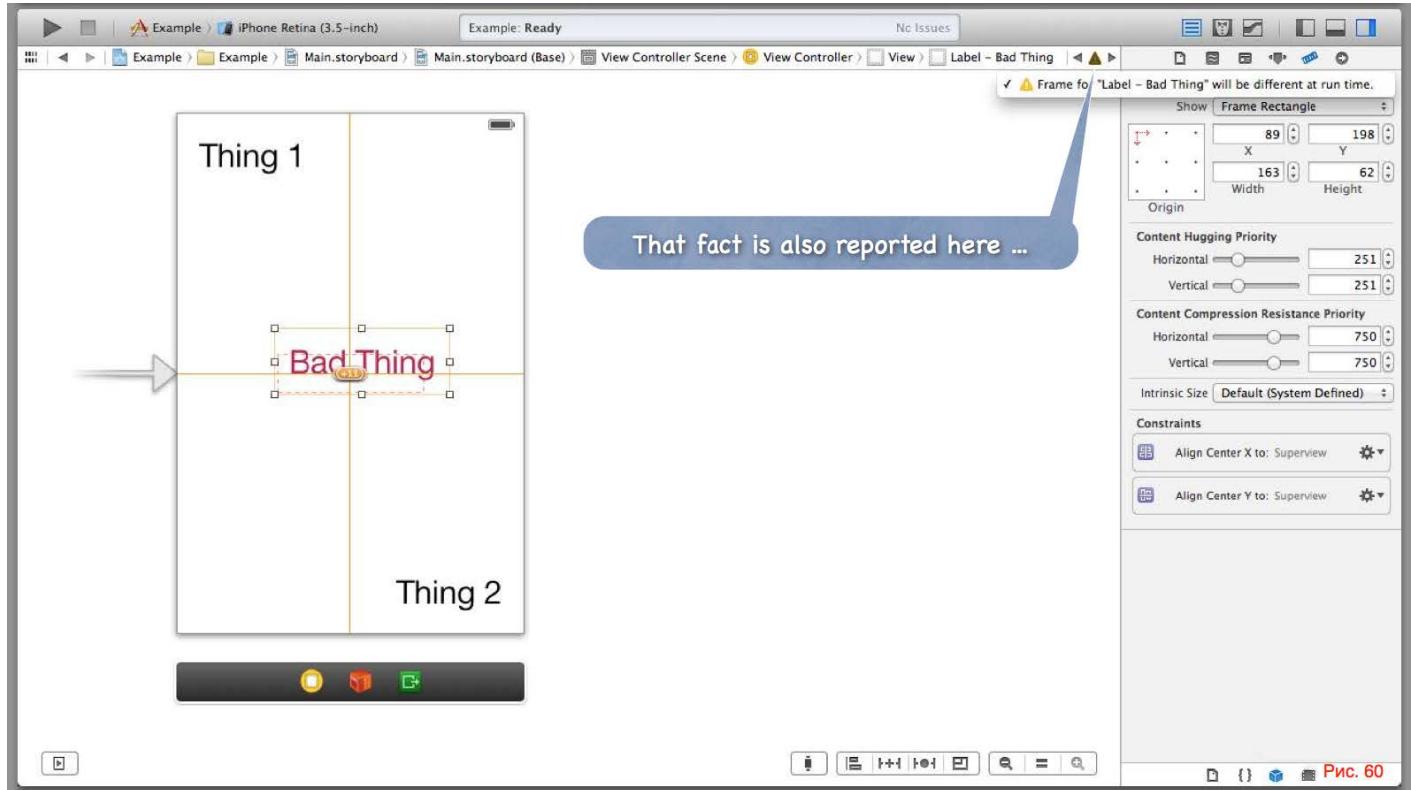
Итак, у вас желтая линия.

При внимательном рассмотрении вы увидите маленький прямоугольник, обозначенный пунктирной желтой линией, внутри большого. Эти желтые линии показывают, где должна располагаться метка "Bad things" согласно новым ограничениям. Но так как мы не позволили переместиться метке в правильное место - мы не кликнули поле "Update Frames" с выпадающим меню, - то наша метка "Bad Thing" осталась на месте, но пунктирная желтая линия показала ее правильное

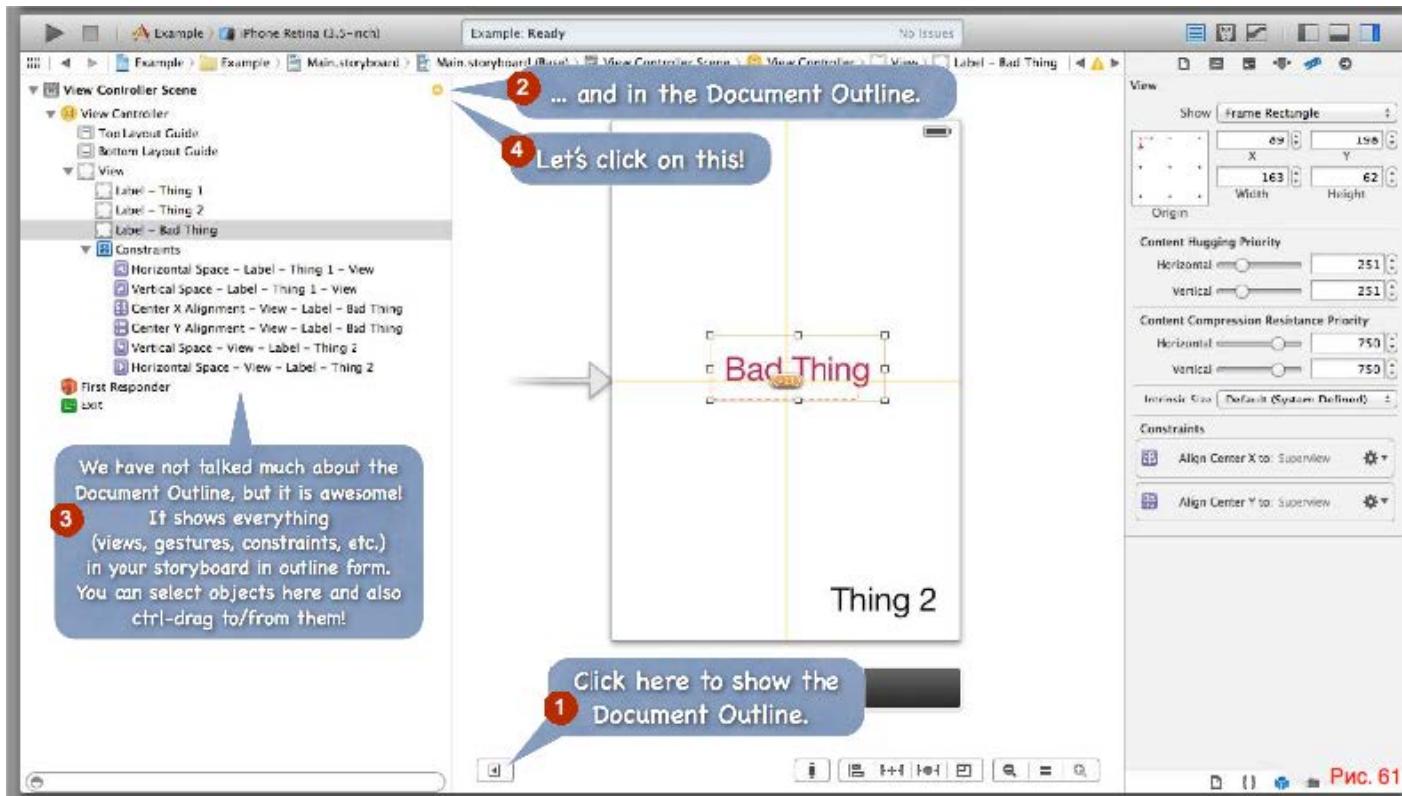
расположение. И поэтому выдается предупреждение.

Как я могу узнать, почему мои ограничения желтые?

Одно место - это маленький треугольник вверху, где обычно отображаются предупреждения компилятора

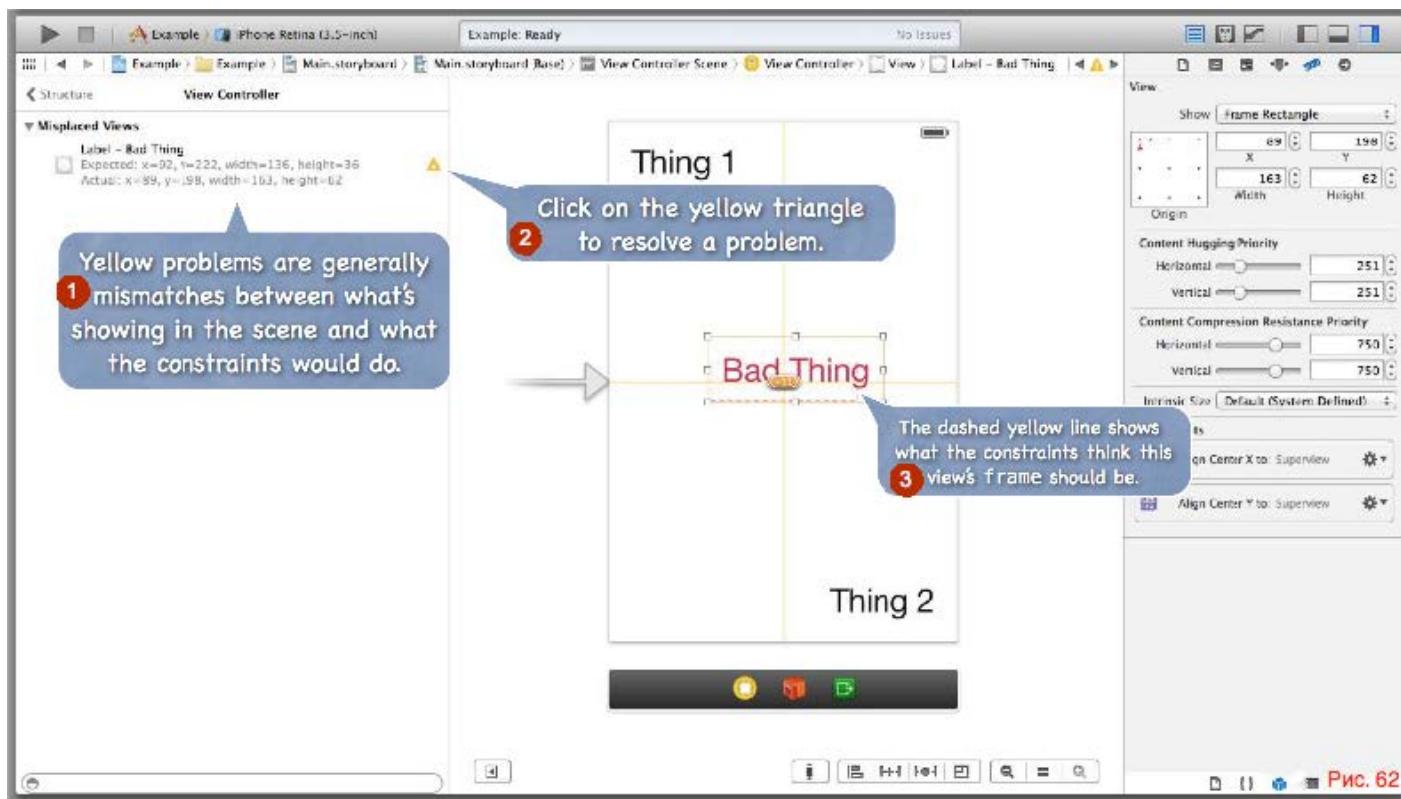


Другое место, где вы можете увидеть предупреждение относительно желтых ограничений - это Document Outline.



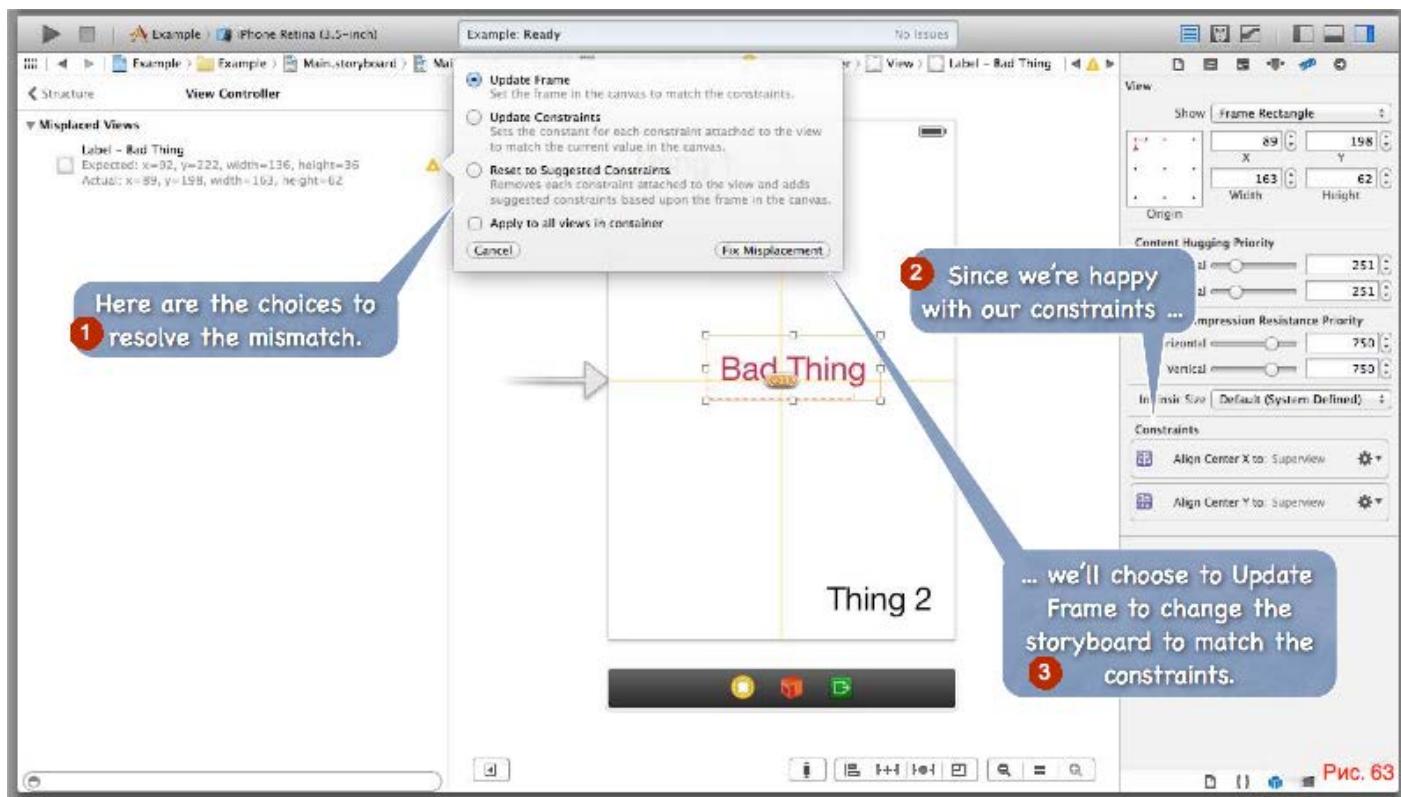
1. Кликните здесь, чтобы показать Document Outline.
2. ... и в Document Outline.
3. Мы не говорили много о Document Outline, но он замечательный! Он показывает все (views, жесты, ограничения (Constraints) и т.д.) на вашем storyboard в специальной outline форме. Здесь вы можете выбрать объект и CTRL - тянуть от него/ к нему.
4. Давайте кликнем на этой желтой стрелке.

Document Outline - прекрасное место для проведения операций CTRL-перетягивания. Но если у вас есть проблемы с Autolayout, то вы увидете маленький желтый или красный кружок со стрелкой в верхней части Document Outline вашего ViewController. Мы кликнем на этом желтом кружочке, чтобы посмотреть, какие проблемы с Autolayout в этом ViewController.



1. “Желтые” проблемы - это обычно несоответствие между тем, что показано на экране и тем, что должно быть, если бы действовали ограничения.
2. Кликните на желтом треугольнике, чтобы разрешить проблему.
3. Желтая пунктирная линия показывает, что с точки зрения ограничений должно было быть.

Как разрешить данную проблему?



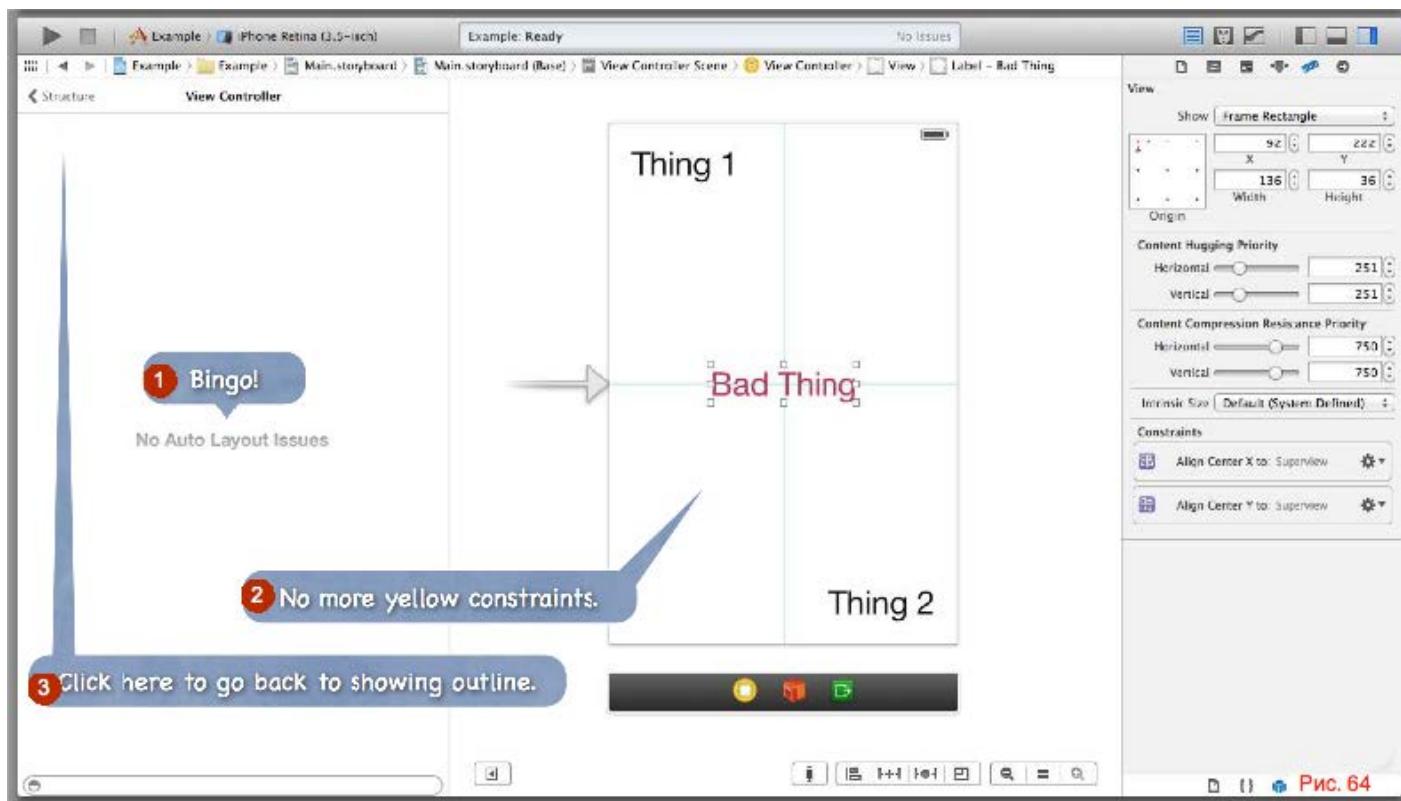
1. Здесь вы можете сделать выбор способа устранения несоответствия.
2. Чтобы все было в порядке с нашими ограничениями...
3. ... мы выберем “Update frame” для изменения storyboard с целью добиться соответствия ограничениям.

Существует несколько способов решения проблемы.

Способ “**Update frame**”. Вы могли бы изменить frame метки “Bad Thing” таким образом, чтобы ограничения удовлетворялись.

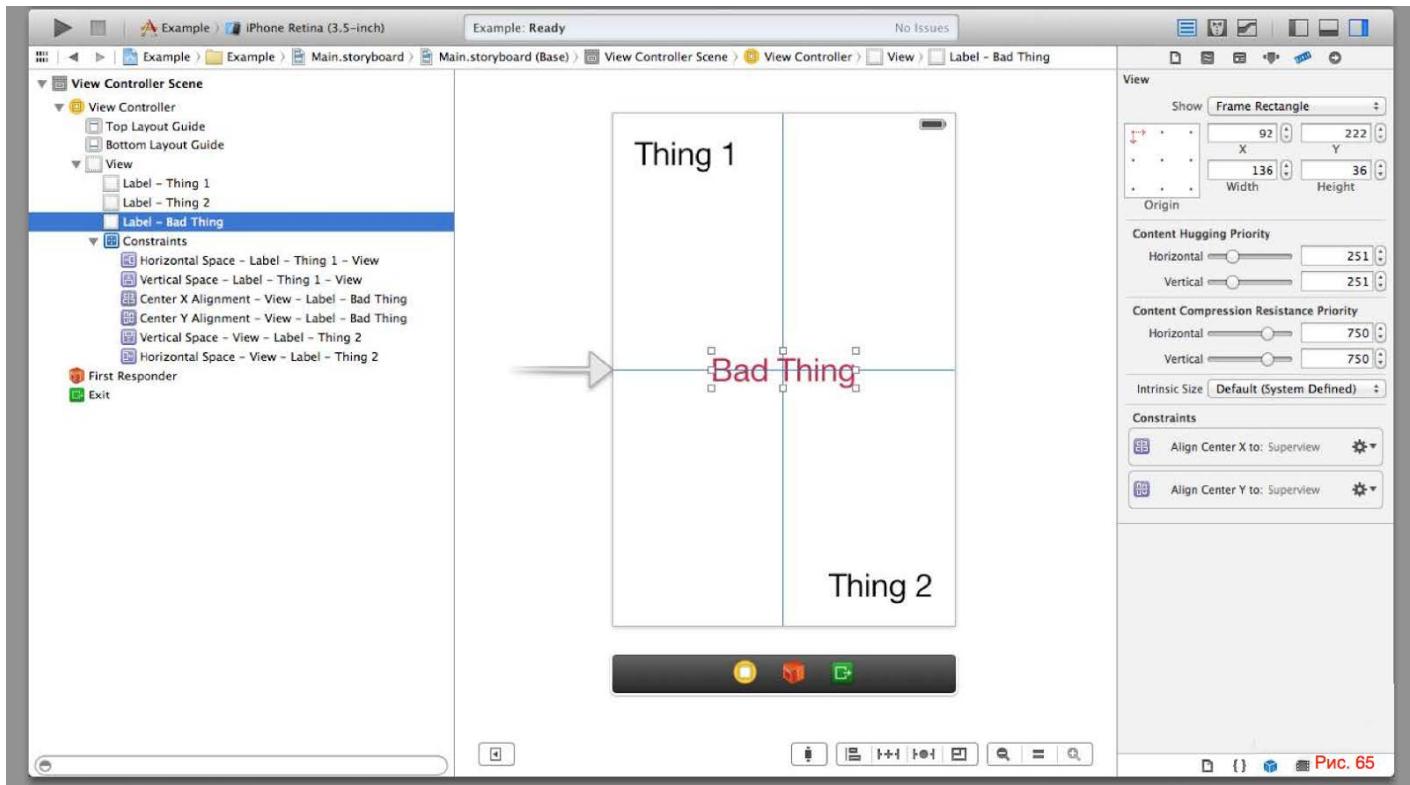
Способ “**Update Constraints**”. Вы могли бы изменить ограничения так, что вместо того, чтобы центрировать горизонтально или вертикально метку “Bad Thing”, мы могли бы выставить ограничения, чтобы работал существующий frame метки, или вернуться к Suggested ограничениям, которые мы знаем, что не работают.

Но мы выбираем “Update frame”, потому что нам нравятся наши ограничения, и кликаем на “Fix Misplacement” и смотрим, что происходит с нашей меткой “Bad Thing”. Она переместится.

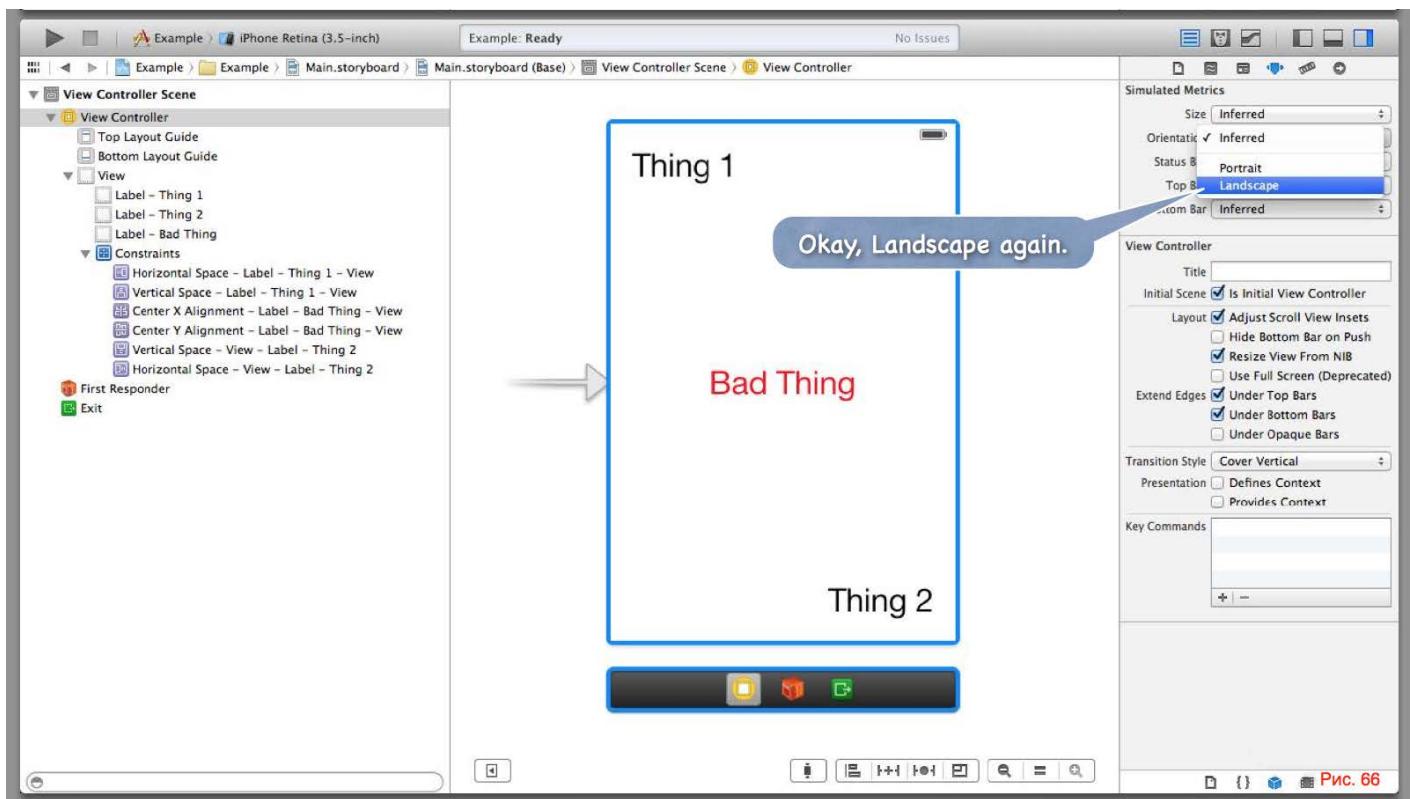


- 1 Победа!
- 2 Больше нет желтых ограничений.
- 3 Кликните здесь, чтобы вернуться в Document Outline.

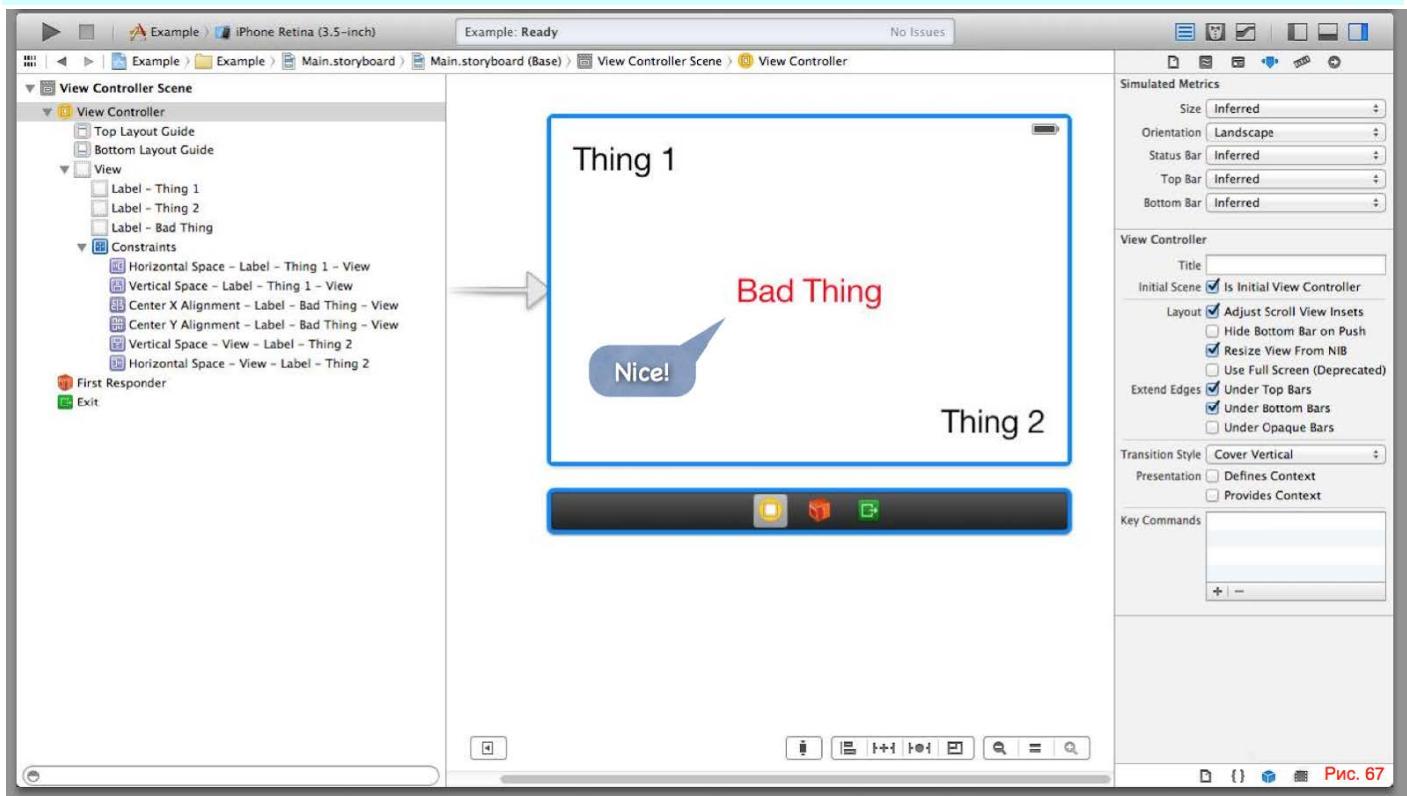
Frame метки “Bad Thing” установился туда, где требуют ограничения.



Теперь попробуем режим “Ландшафт”.

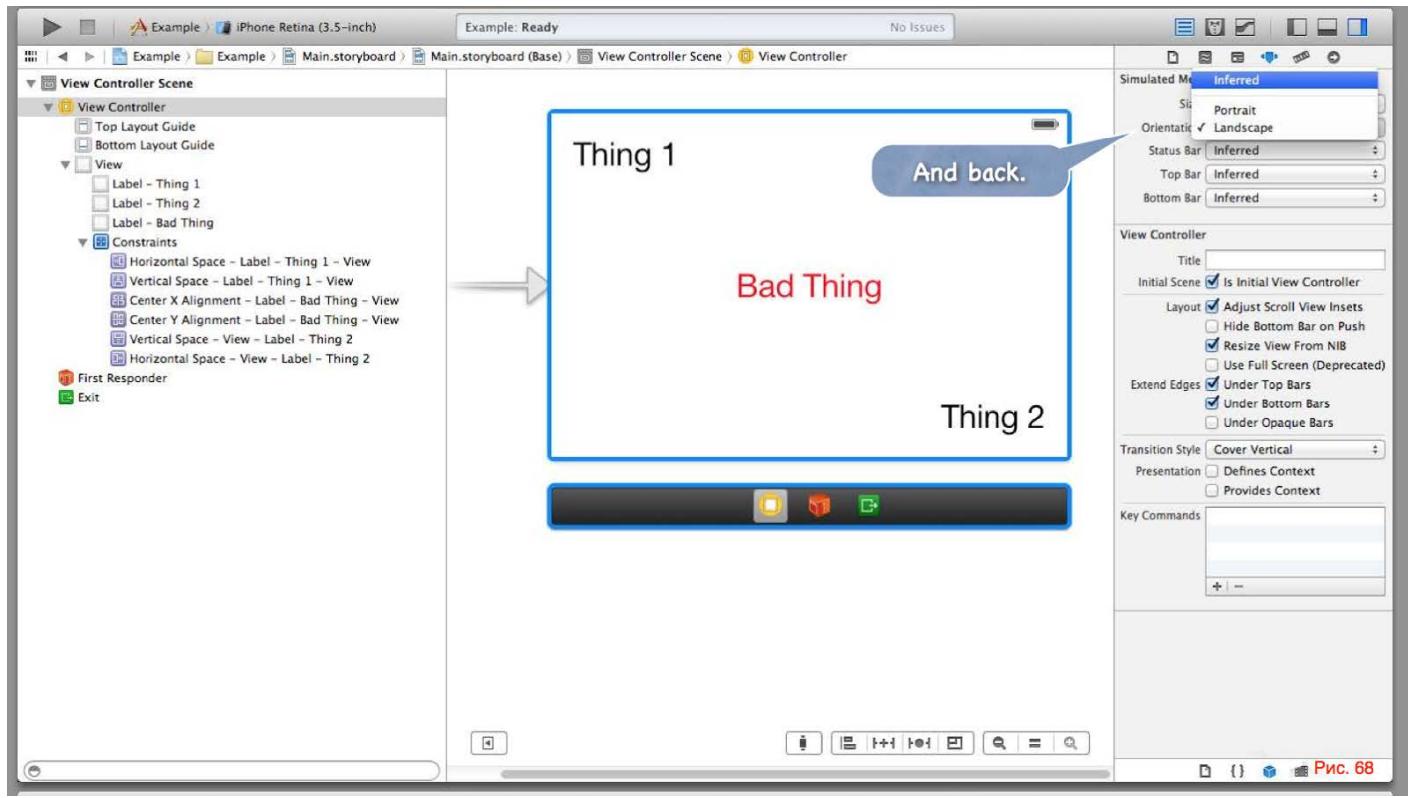


Итак, опять режим “Ландшафт”.



Прекрасно

Метка “Bad Thing” остается центрированной.



Вернемся назад.

Мы уже говорили о том, что вы можете кликнуть на любом ограничении как в Document Outline, так и непосредственно на самом ограничении.

Мы кликаем на этом вертикальном ограничении и оно будет подсвечиваться, а Инспектор Атрибутов будет показывать его параметры.

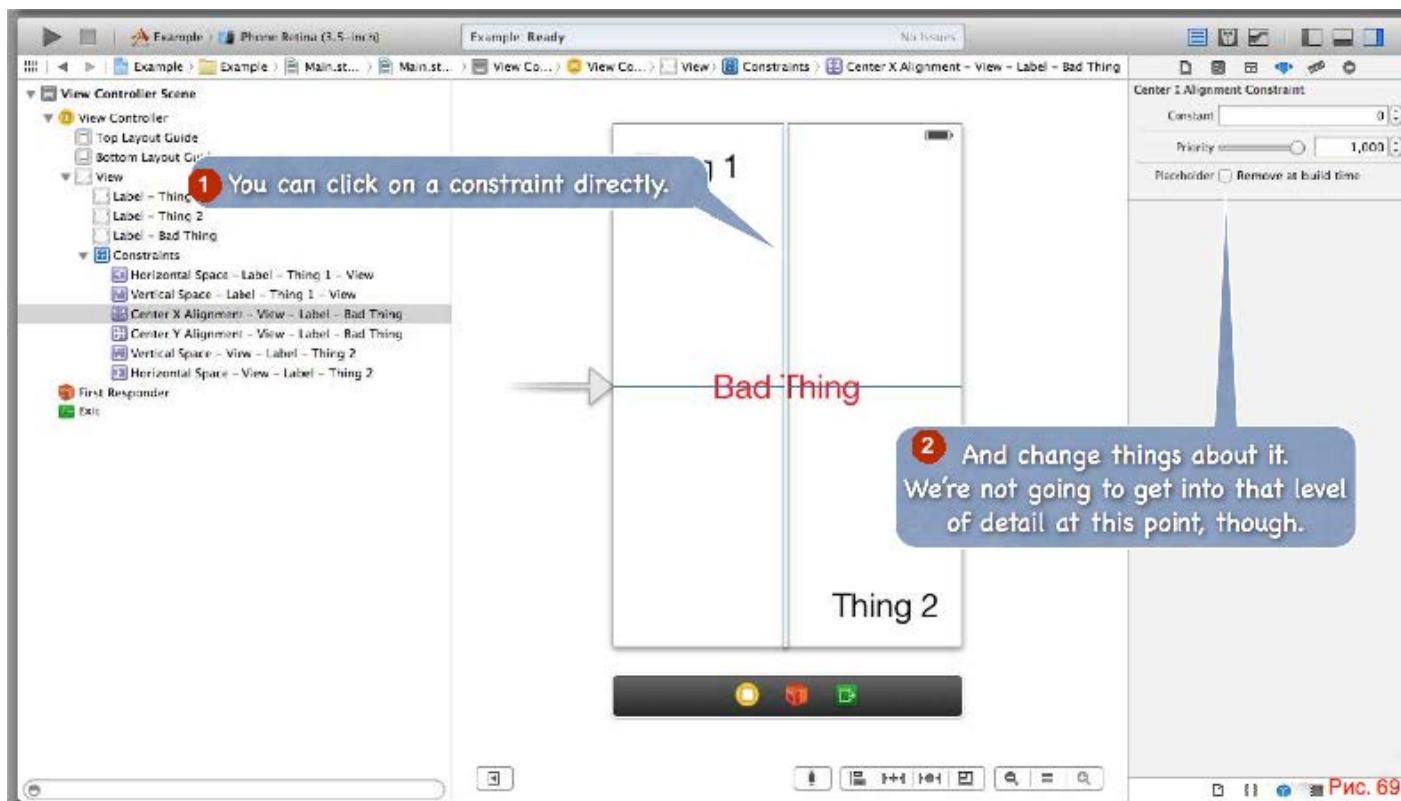


Рис. 69

Много параметров ограничения, которые вы можете изменить. Например, там есть priority (приоритет) , потому что у вас могут быть два ограничения в очень сложной разметке, которые конфликтуют. Например, вы хотите расположить некое view под другим view и в то же время посередине экрана. Но если экран становится реально очень маленьким, то вам важнее расположить первое view под вторым, чем оставаться посередине, и вы устанавливаете приоритеты для противоречивых ограничений. Приоритет принимает значение от 0 до 1000. Другая вещь, которую вы можете делать с ограничениями - это нажать клавишу DELETE. Тогда ограничение удалится.

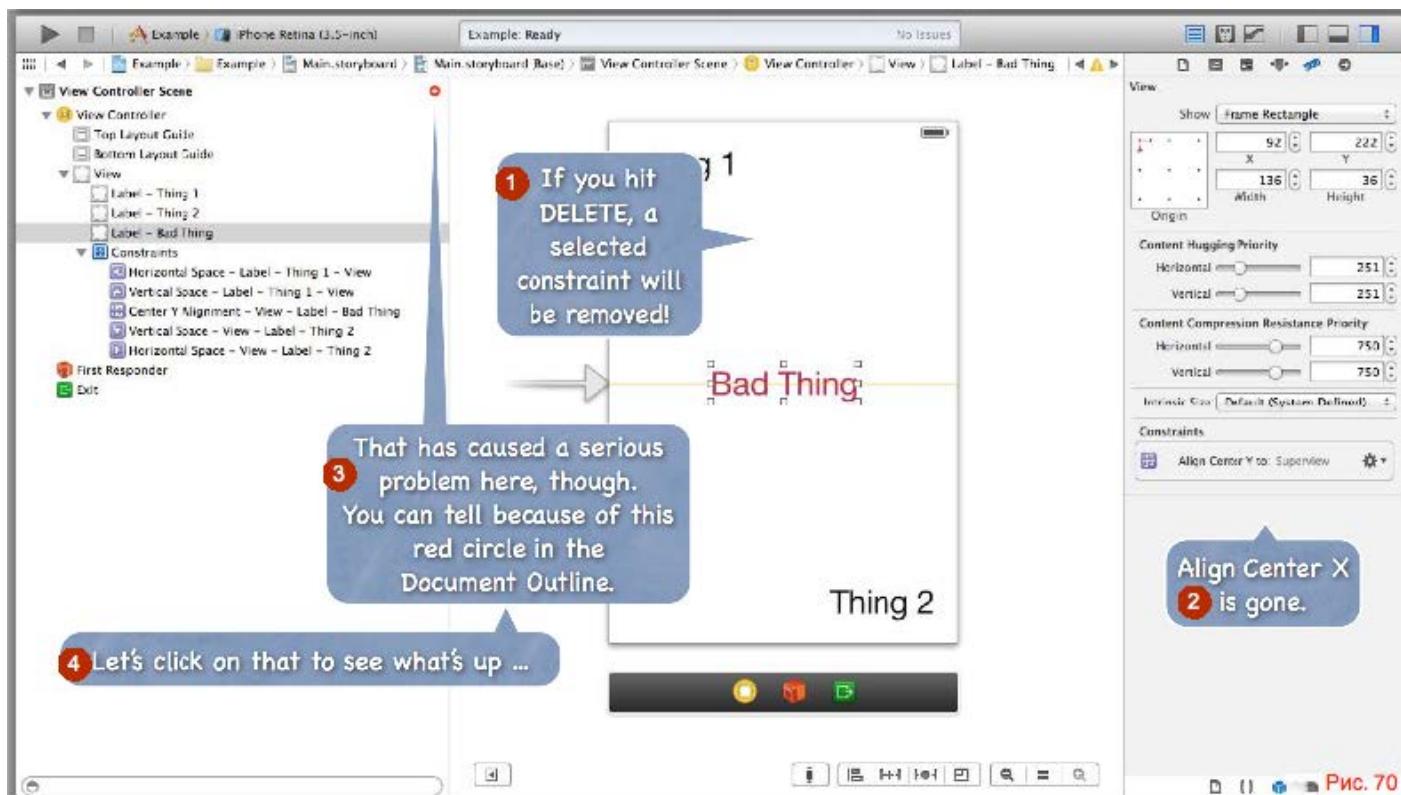
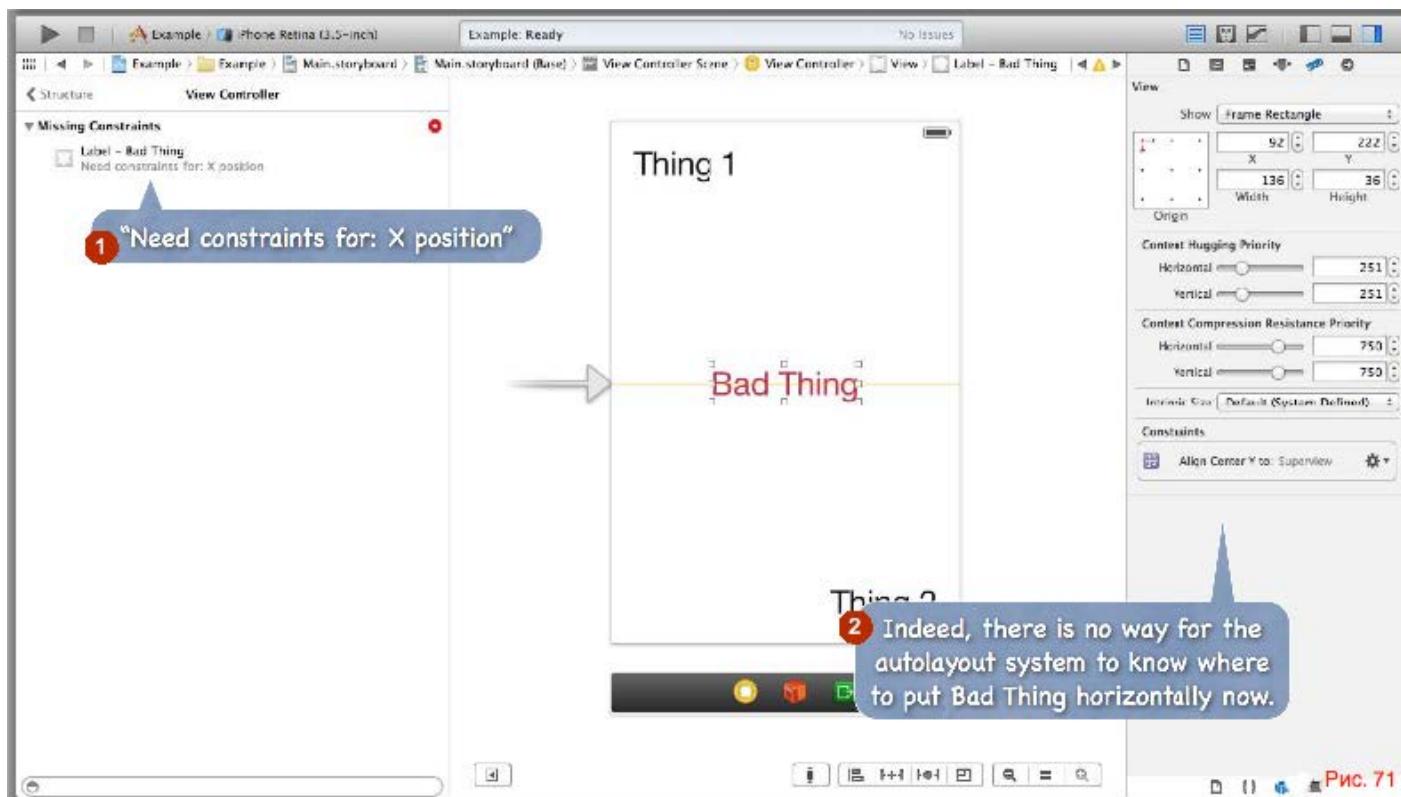


Рис. 70

1. Если вы нажмете DELETE, то выбранное ограничение удалится!.
2. Align Center X удалилось.
3. Это вызовет серьезные проблемы. Об этом говорит красный кружок в Document Outline.
4. Давайте кликнем на нем и посмотрим, что произойдет...

Теперь ничего не фиксирует положение метки “Bad Thing” по горизонтали, и это действительно плохо. Обычно красный кружок говорит о том, что ограничений недостаточно, чтобы зафиксировать положение view, то есть имеют место пропущенные ограничения (Missing Constraints).

Но если у нас будут два противоречивых ограничения, то кружочек в Document Outline тоже будет красным.



1. “Необходимо ограничение для: X позиции”
2. В действительности, нет способа у системы Autolayout узнать, где горизонтально должна располагаться метка “Bad Thing”.

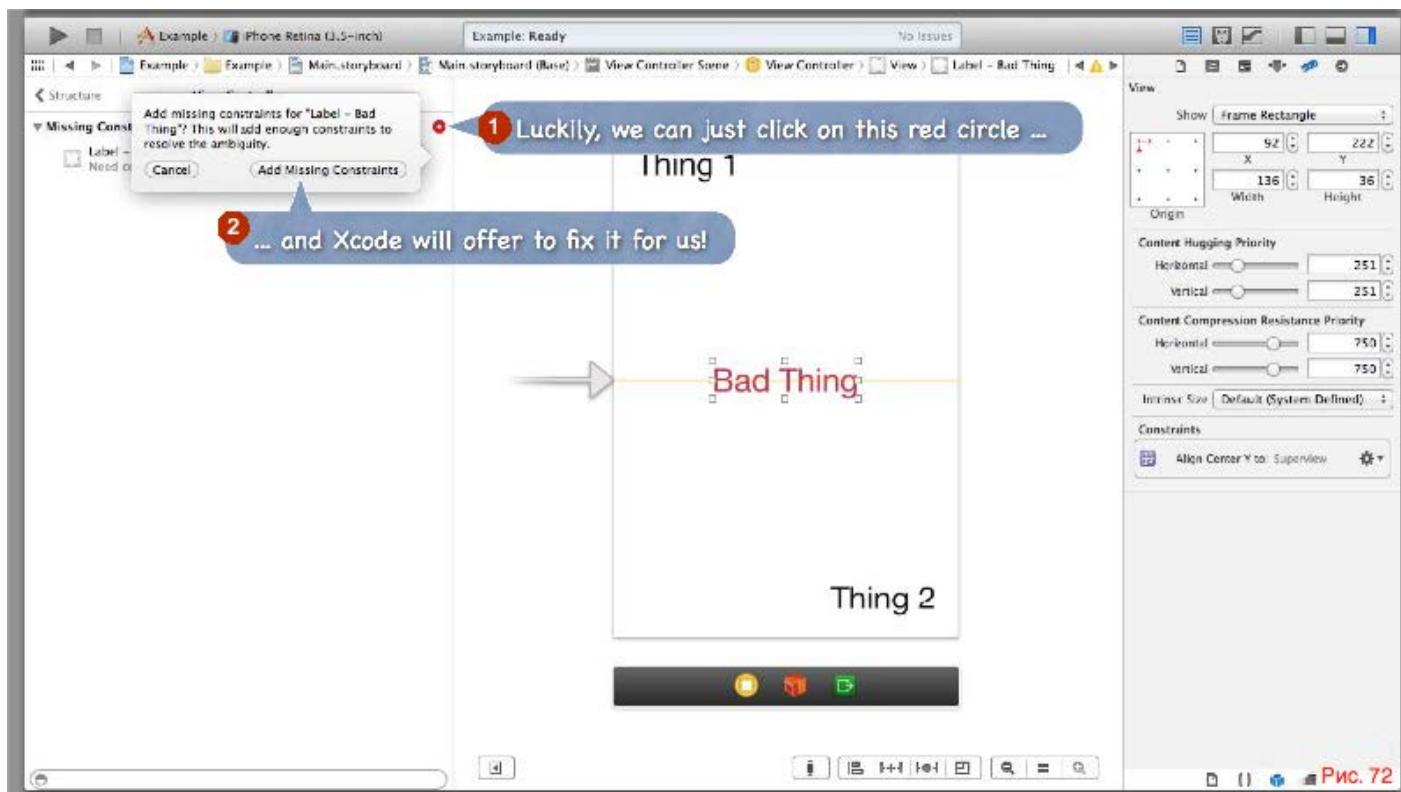


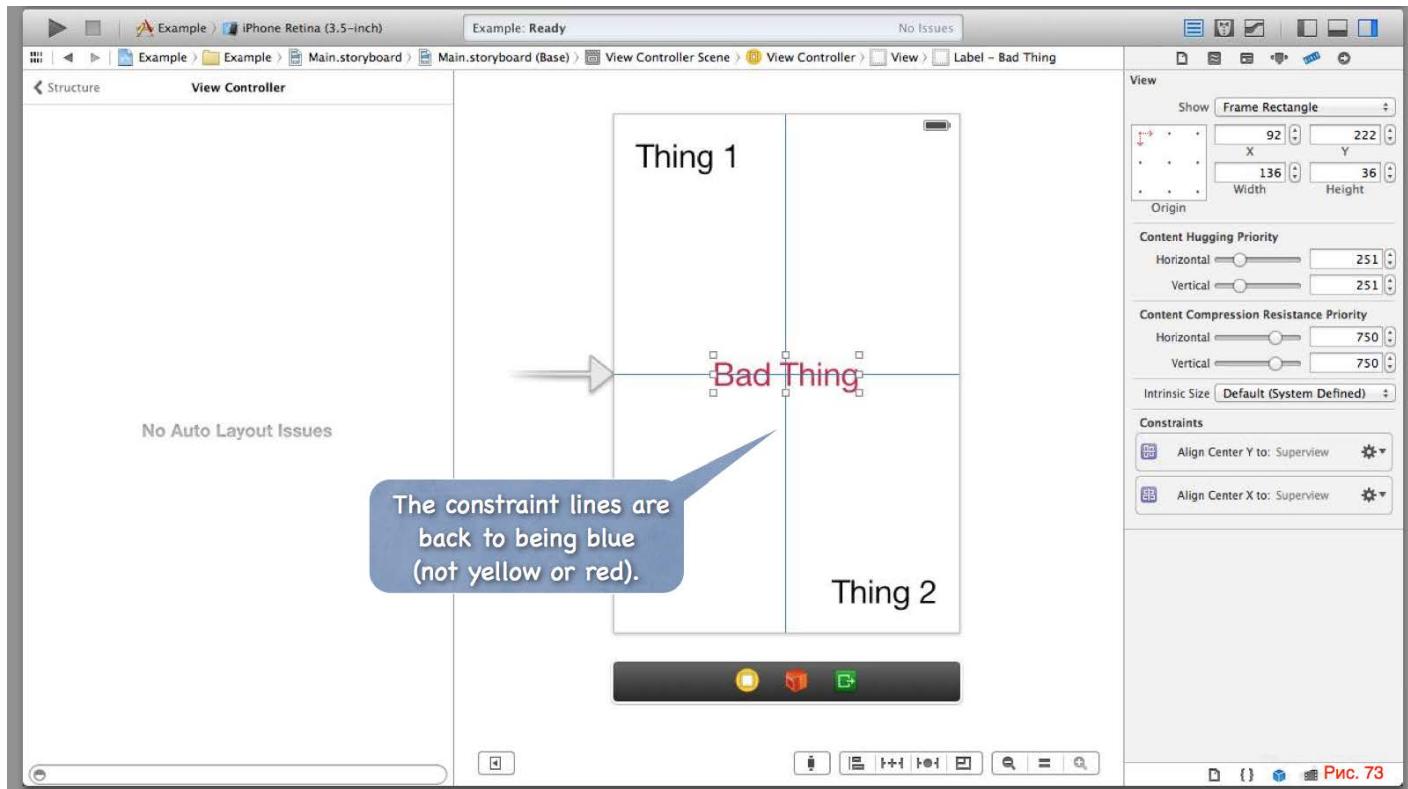
Рис. 72

1. К счастью, мы можем просто кликнуть на этом красном кружочке...
2. ... и Xcode предложит нам решение проблемы!

Xcode предложит добавить одно или несколько ограничений и сделает все возможное, чтобы решить нашу проблему. Иногда это может быть не те ограничения, которые вы бы хотели.

Возможно, будут предложены ограничения близкие к Suggested, но в любом случае это полностью определит местоположение и, если нужно, размер view.

В нашем случае, так как наша метка находится в середине экрана и зафиксирована голубыми направляющими линиями, то добавление ограничений просто вернет удаленное ограничение, так как оно лучше всего подходит текущему расположению метки “Bad Thing”.



Линии ограничений опять становятся голубыми (не желтыми или красными).

Система Autolayout очень мудрая относительно пропущенных ограничений. По большей части она старается использовать Suggested ограничения, но она значительно умнее.

Теперь у нас нет ошибок Autolayout.

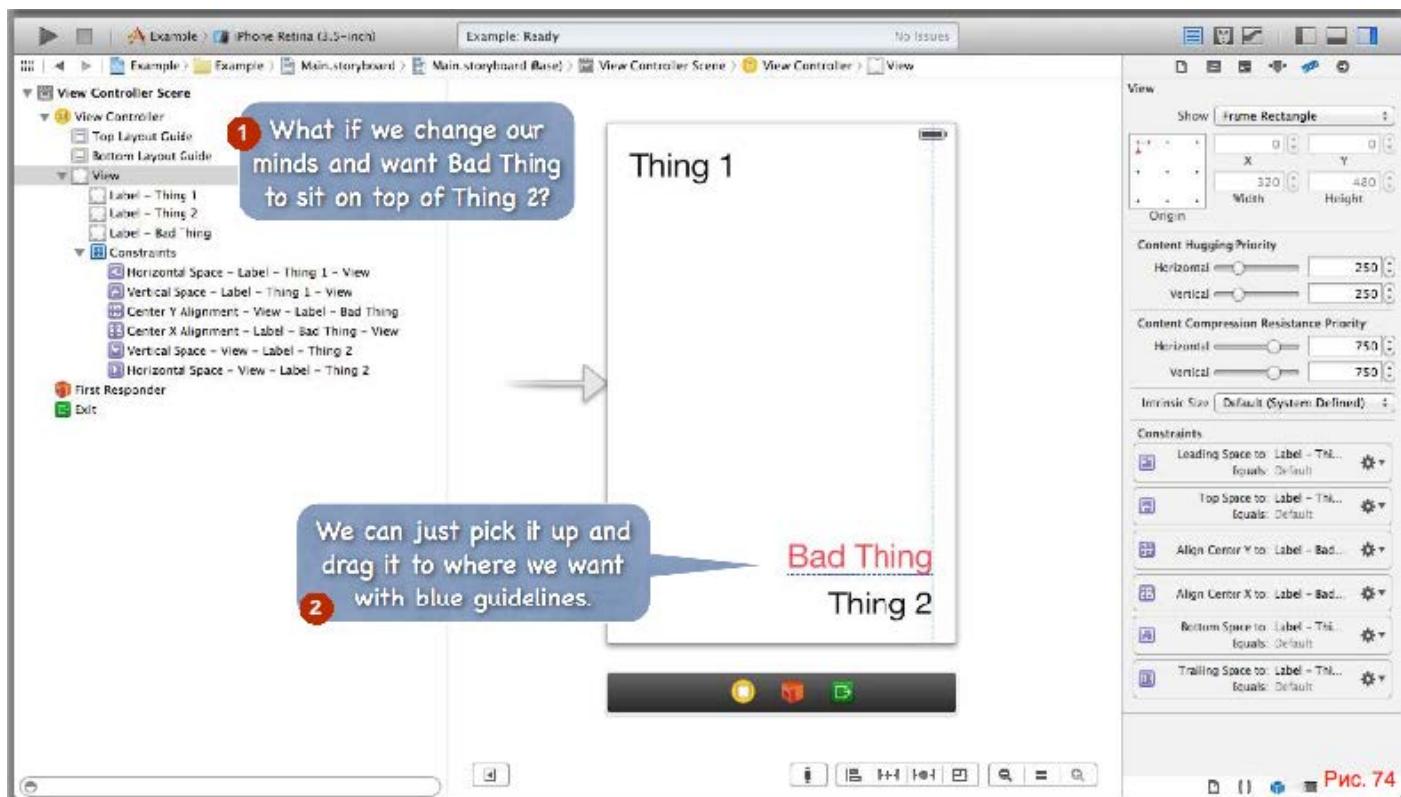
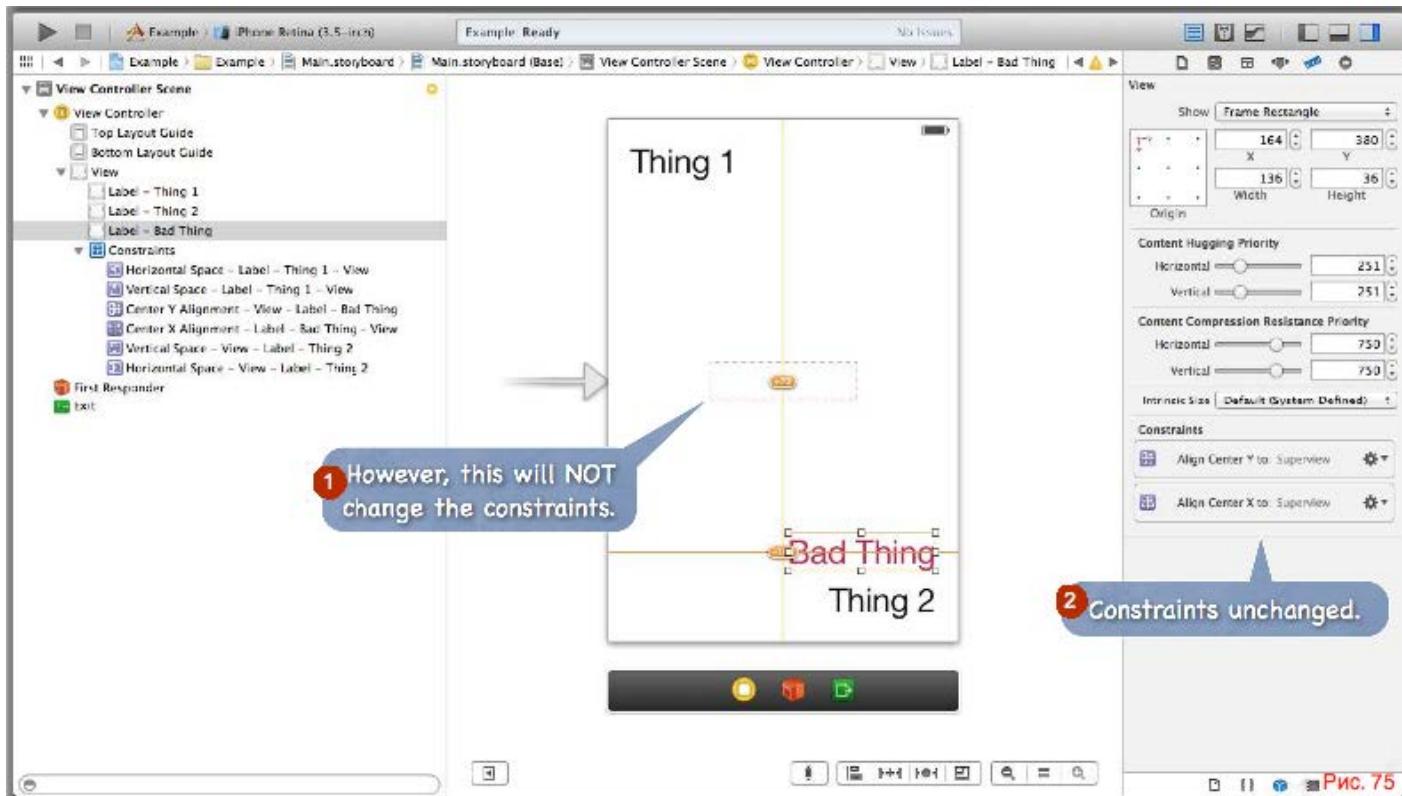


Рис. 74

1. А что если мы передумаем и захотим разместить “Bad Thing” над “Thing 2”?
2. Мы выберем метку и перетащим ее туда, куда мы хотим с голубыми направляющими линиями или нет.

После того, как мы бросим метку на новом месте, смотрите, что произойдет.



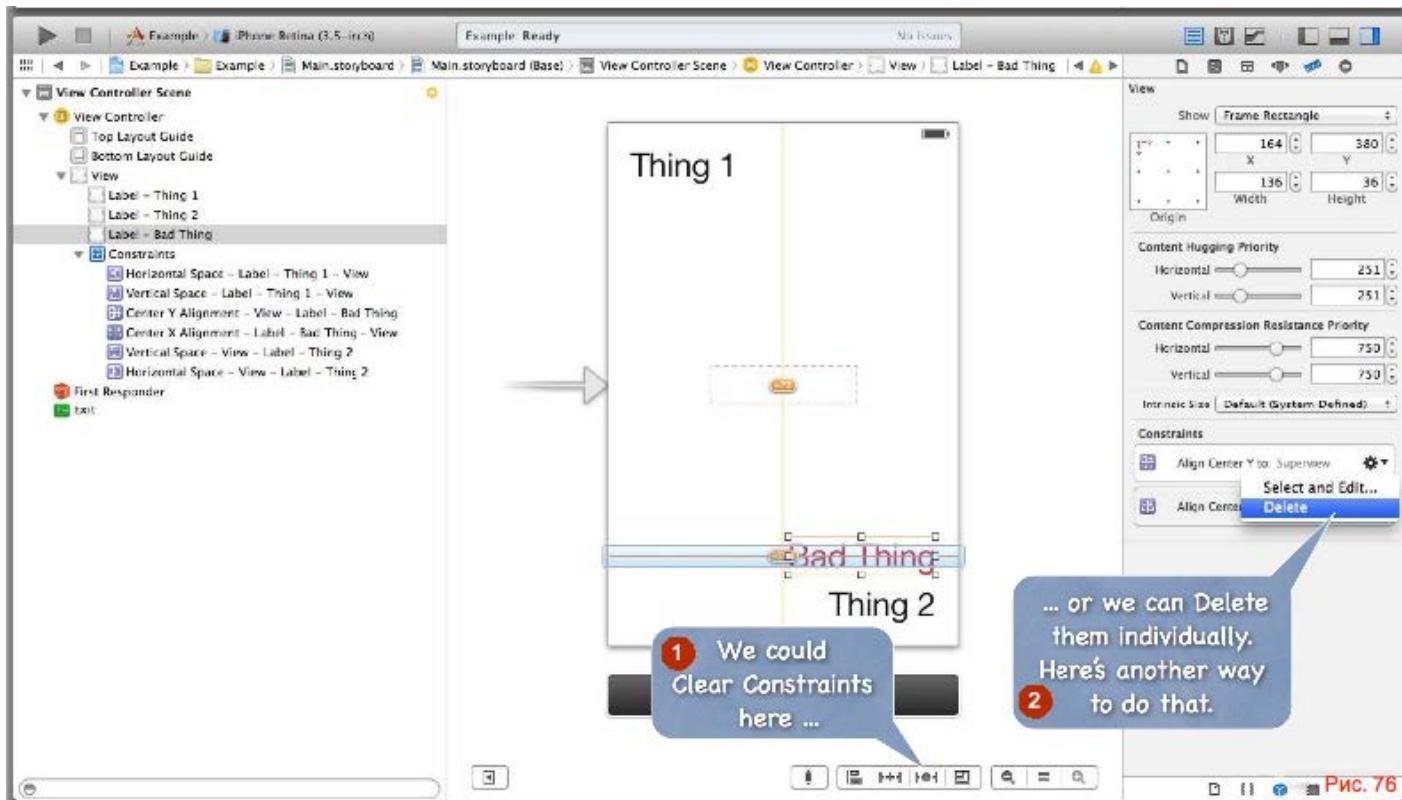
1. Наши действия **НЕ** изменили ограничения.
2. Ограничения остались неизменными.

Итак, мы опять получили желтые направляющие линии, потому что опять, “Bad Thing” расположена не на том месте. Когда мы перемещали метку на новую позицию, ограничения остались неизменными. Это очень ВАЖНО понимать: если мы что-то перемещаем на новое место, то ограничения остаются теми же самыми.

На слайде вы видите, что ограничения для “Bad Thing” те же: Align Center X и Align Center Y. Это не очень хорошо.

Что же мы можем сделать? Возможно нам следует удалить ограничения, так как они больше не работают, и мы смотрели ранее, как “очистить” ограничения (Clear Constraints), и как удалить их, если выбрать ограничение и нажать “delete”.

Но есть и еще один способ удаления ограничений: выбираете маленько колесико справа от ограничения в Инспекторе Размеров, появляется меню и вы можете выбрать либо “Select and Edit” (что тоже самое, что выбрать и в Инспекторе Атрибутов отредактировать ограничение) или “Delete”.

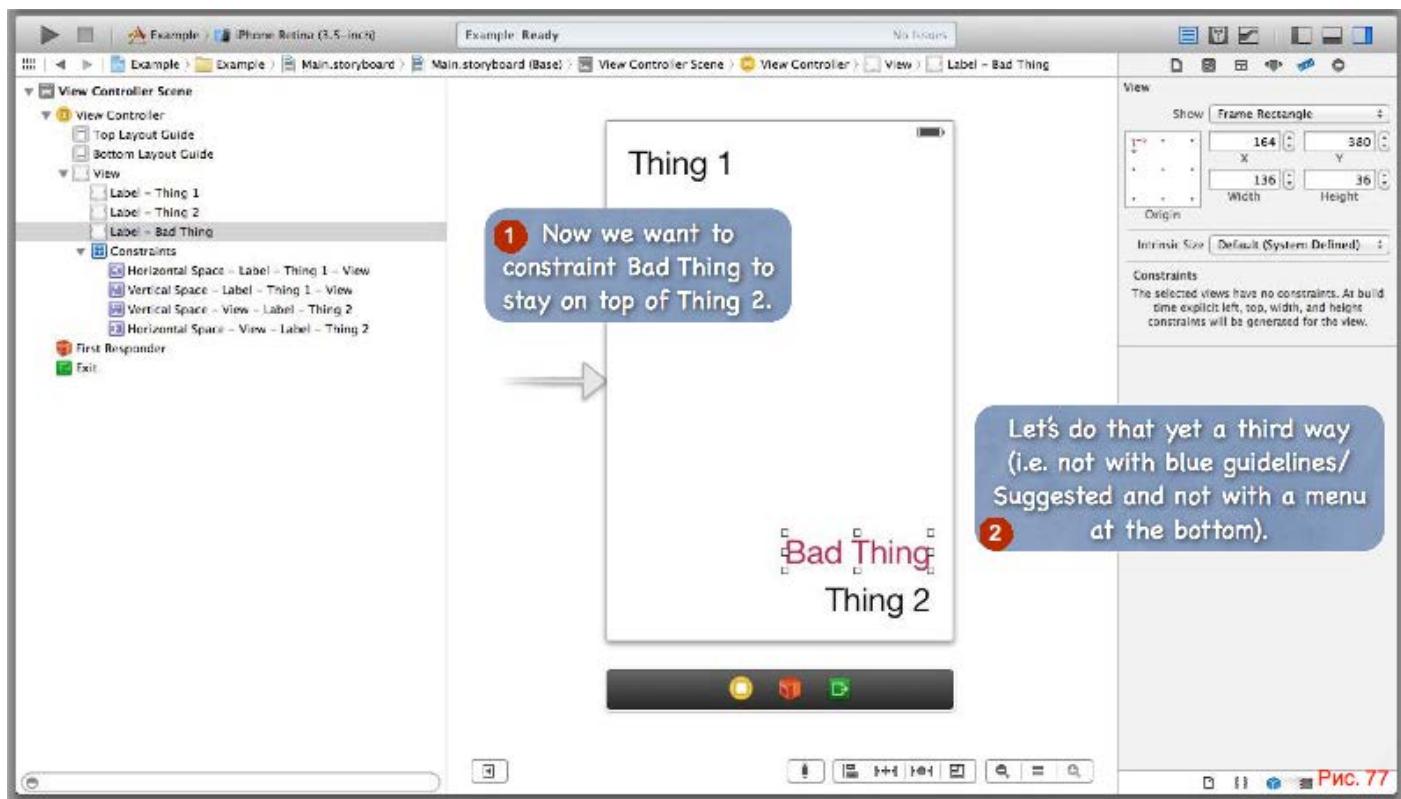


1. Мы можем выполнить Clear Constraints, нажимая эту кнопку...
2. ... или можем провести индивидуальное удаление. Это другой способ сделать это.

Удаляем оба ограничения и мы вернулись к ситуации, когда у “Bad Thing” нет ограничений.

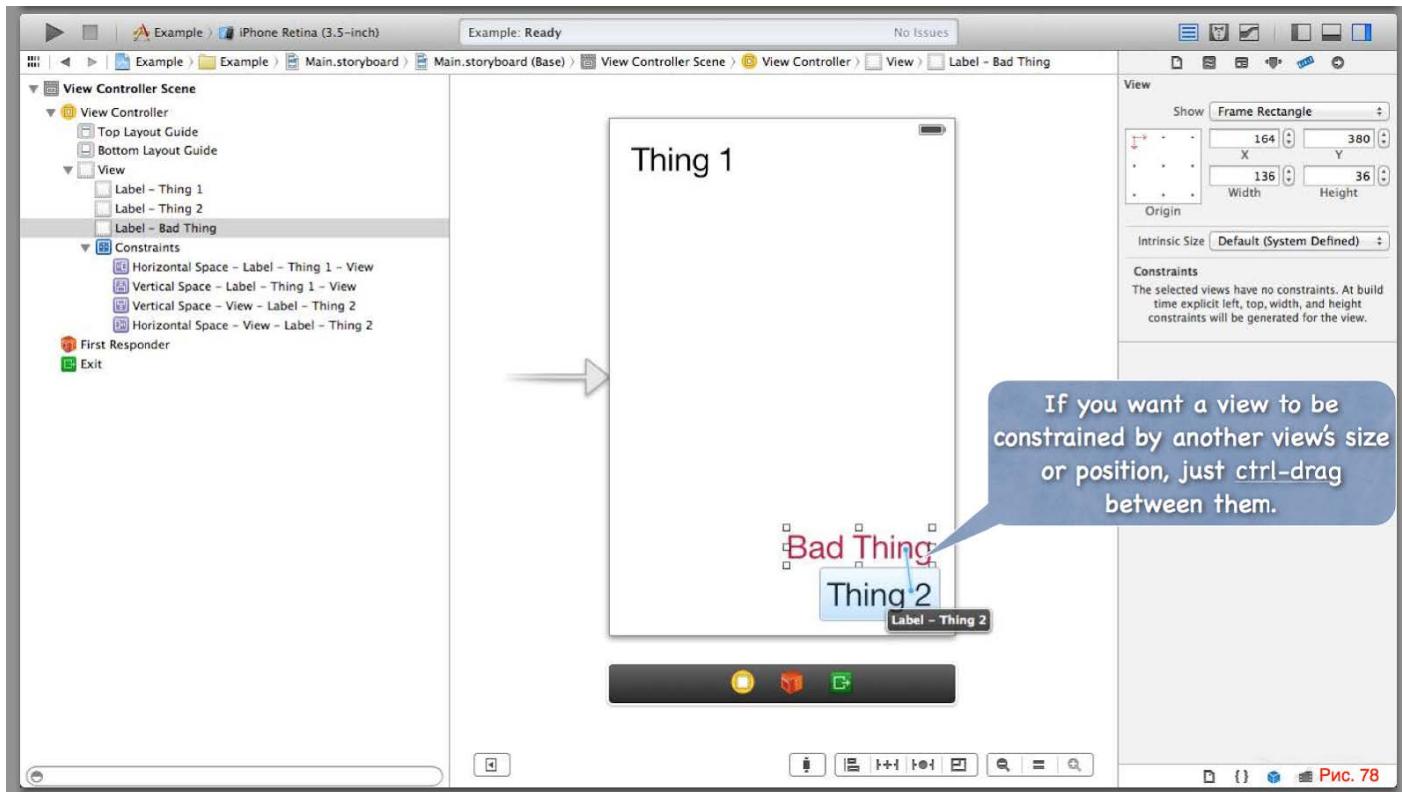
Это и есть способ №3.

Рис. 76



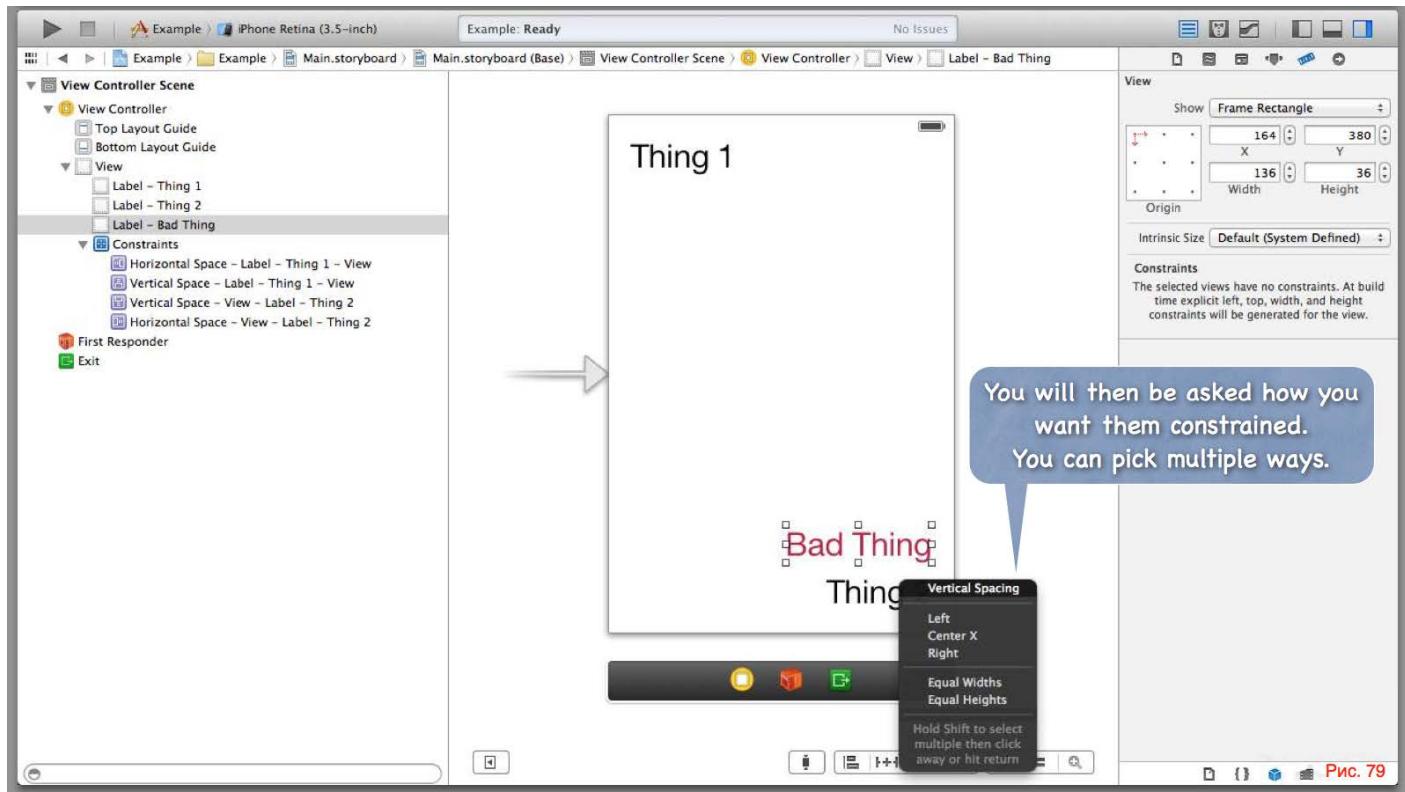
1. Теперь мы хотим ограничить “Bad Thing” так, чтобы она оставалась сверху метки “Thing 2”.
2. Давайте сделаем это 3-м способом ( то есть без голубых направляющих линий и/или Suggested ограничений и без меню в нижней части экрана).

**Способ № 3** - это использовать CTRL- перетаскивание (как это делали мы при привязки outlets) для того, чтобы установить ограничение типа взаимосвязи между двумя элементами пользовательского интерфейса.

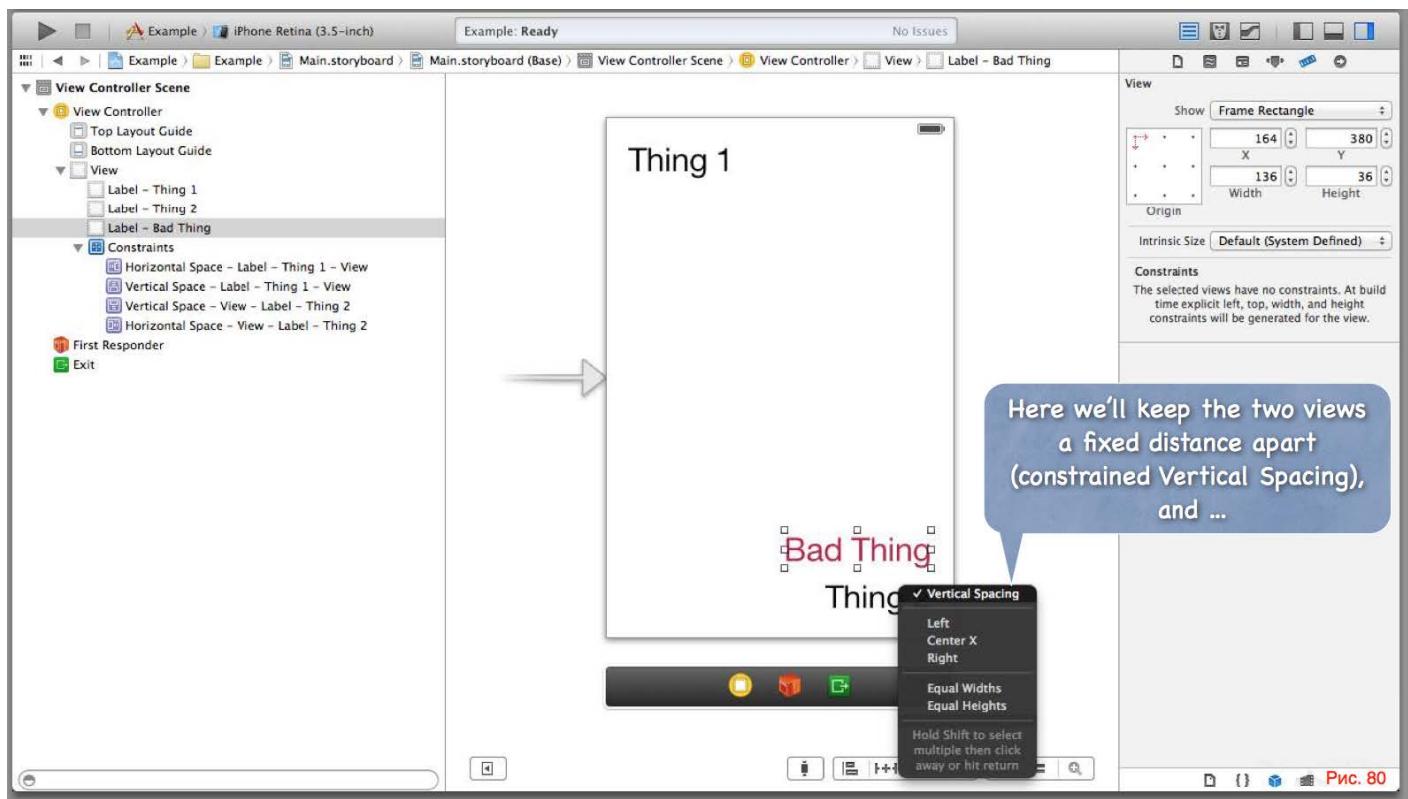


Если вы хотите, чтобы view было ограничено размерами или позицией другого view, сделайте **CTRL-перетаскивание** между ними.

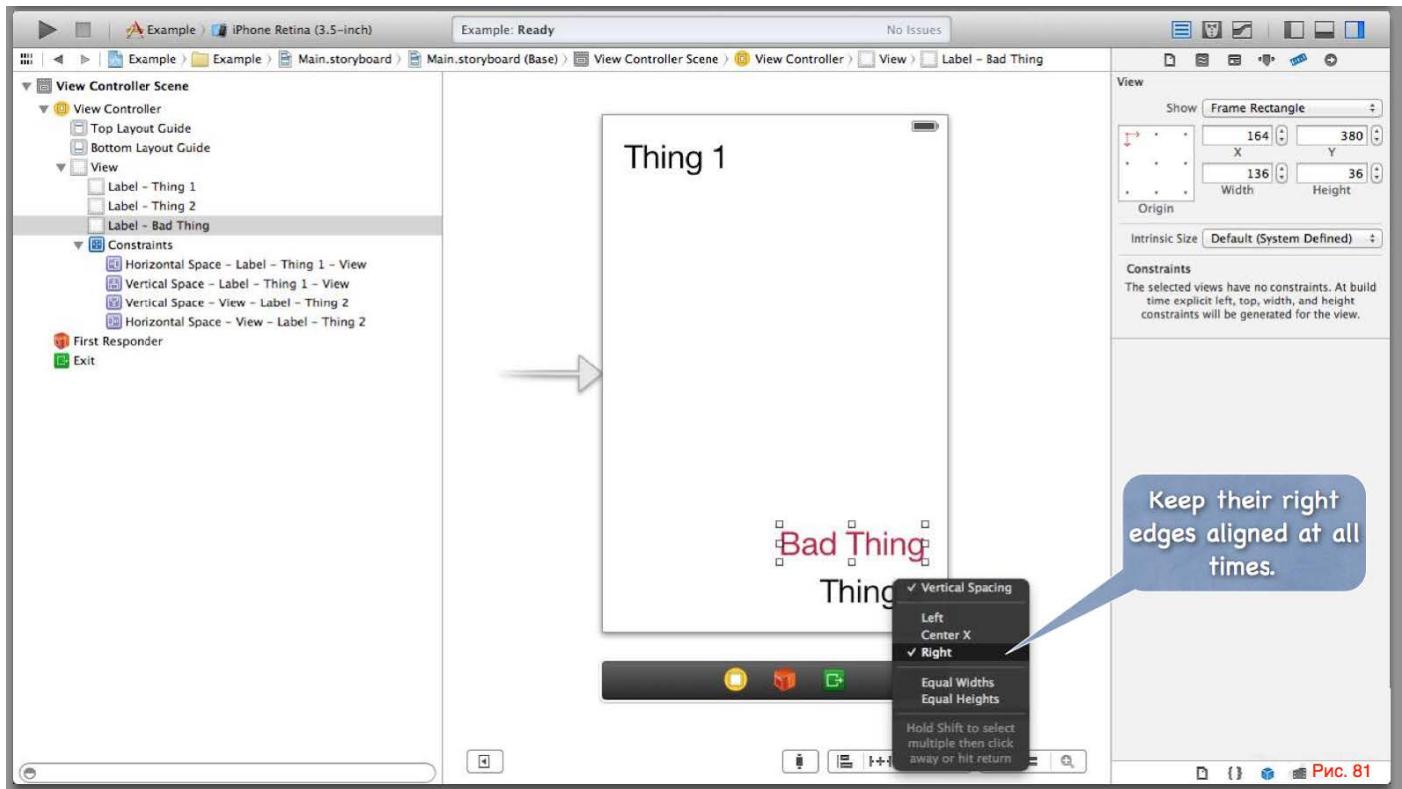
Мы будем делать **CTRL-перетаскивание** от “Bad Thing” к “Thing 2”. И когда мы сделаем это, появится маленький черный ящичек как при привязки outlets и нас спросят какого типа привязки мы хотим сделать между двумя элементами.



Вы можете выбрать множество опций, если будете держать нажатой клавишу SHIFT, о чем сказано в нижней части черного ящичка.

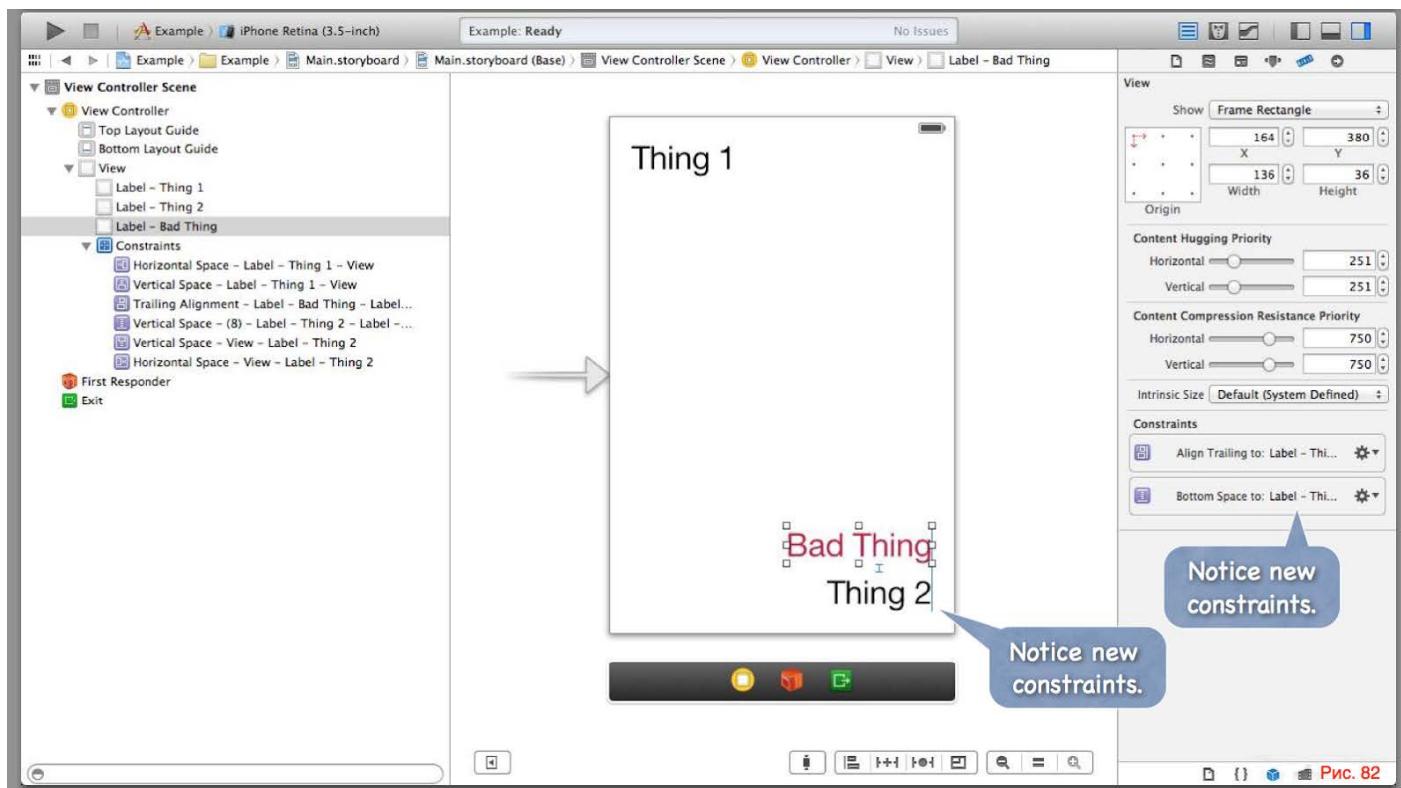


Мы будем держать два views на фиксированном расстоянии друг от друга (Vertical Spacing ограничение) и...



Мы всегда будем поддерживать выравнивание правых краев “Bad Thing” и “Thing 2”.

Хотя в черном ящичке говорится left и right, в действительности, это leading (лидирующие) и trailing (хвостовые) расстояния, но необходимо точно знать, что лево, а что право. В нашем случае нам нужно Right (право). Поэтому мы выбираем два ограничения: Vertical Spacing и Right.

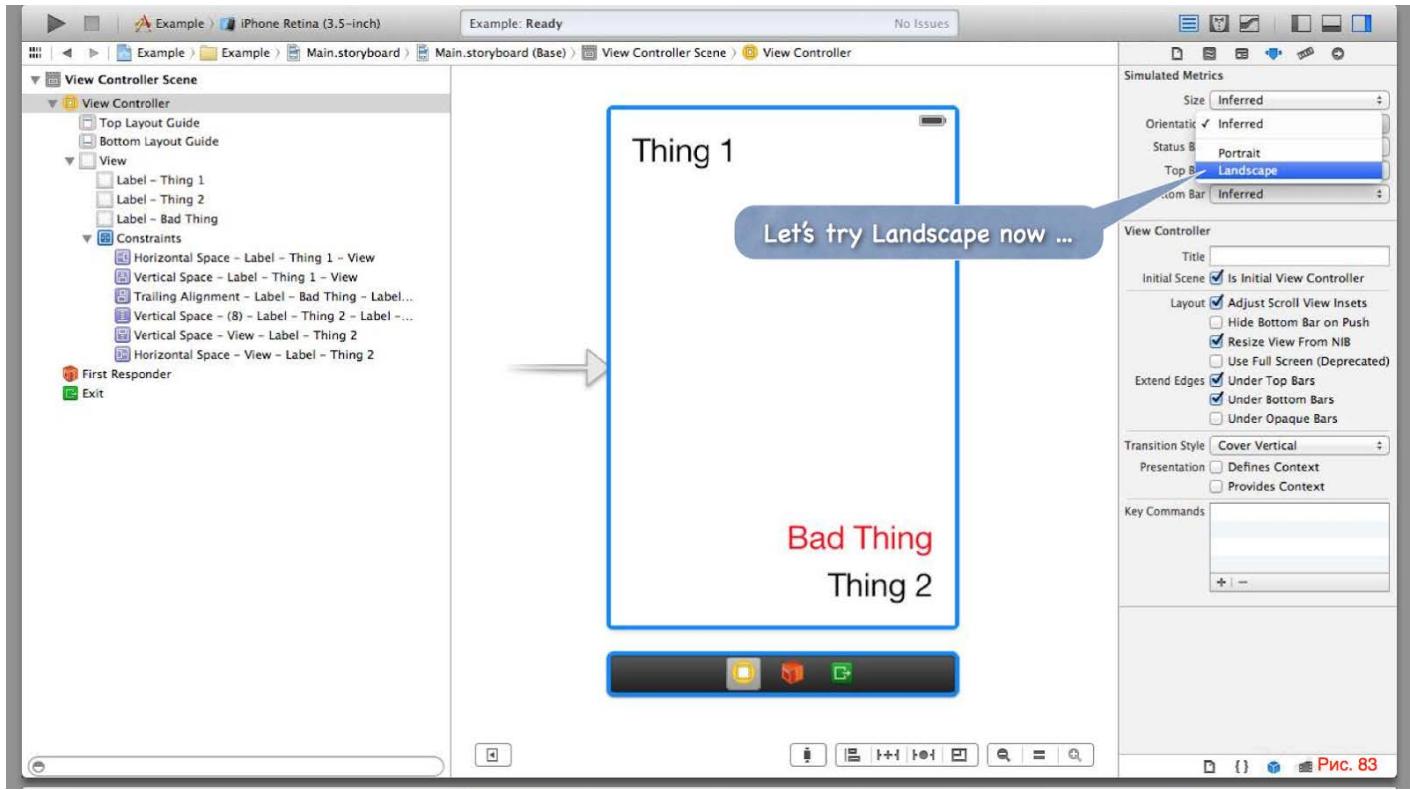


Обратите внимание на новые ограничения.

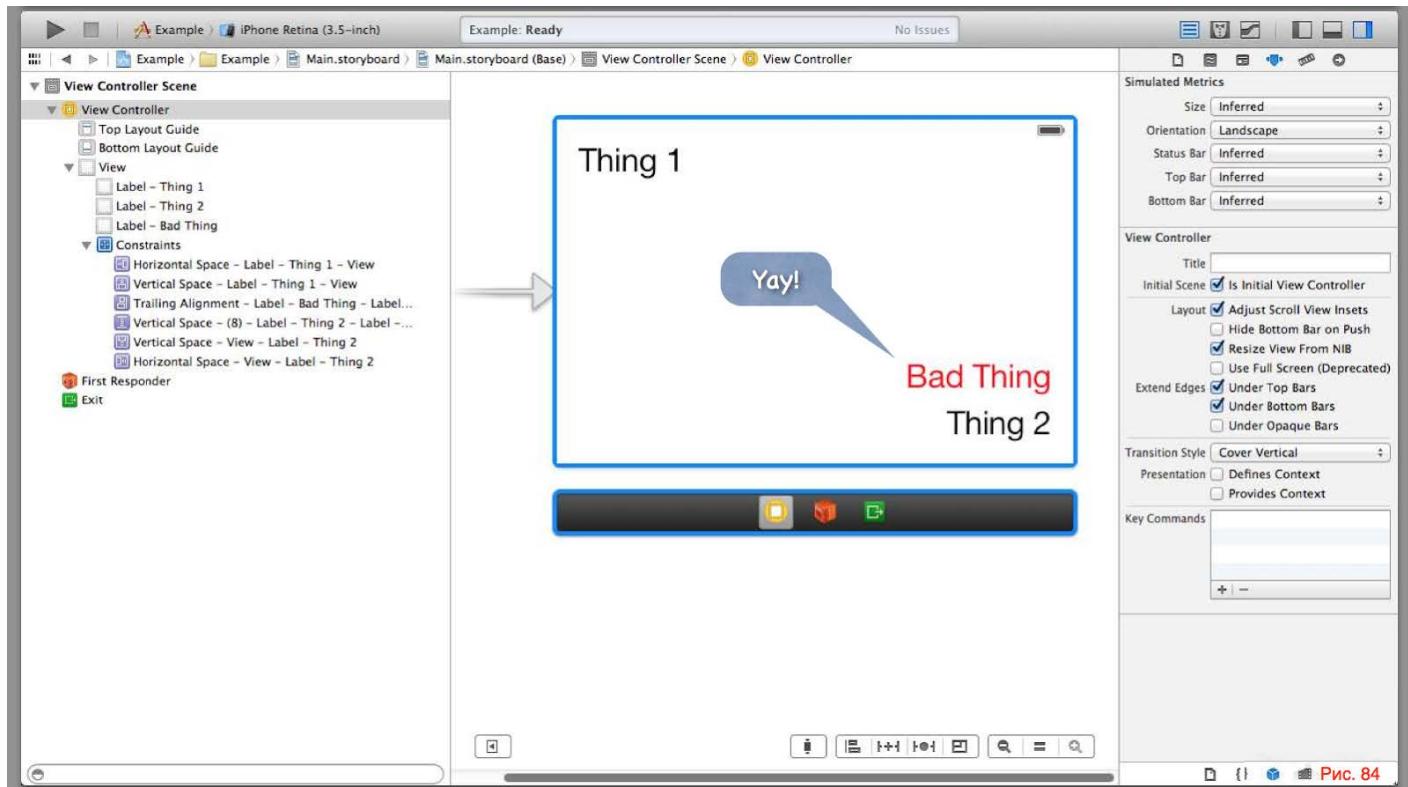
На экране вы видите две голубые линии между “Bad Thing” и “Thing 2”: они выравнены справа и между ними фиксируется расстояние, равное величине по умолчанию. Это получилось потому, что мы использовали голубые направляющие линии при размещении “Bad Thing”. Если бы мы не использовали голубые направляющие линии, и разместил бы 40 pixel вверх от “Thing 2”, то мы бы увидели в ограничениях “магическое число” 40, которое нам нужно было бы исправить. Это можно сделать, редактируя ограничение и выбирая constant “Use Standard Value”.

Теперь у нас есть два ограничения для “Bad Thing”, которые полностью фиксируют положение метки “Bad Thing”, так как метка “Thing 2” жестко приклеена к правому нижнему углу экрана, а метка “Bad Thing” привязана к правой стороне экрана и к верху метки “Thing 2”.

Рис. 82

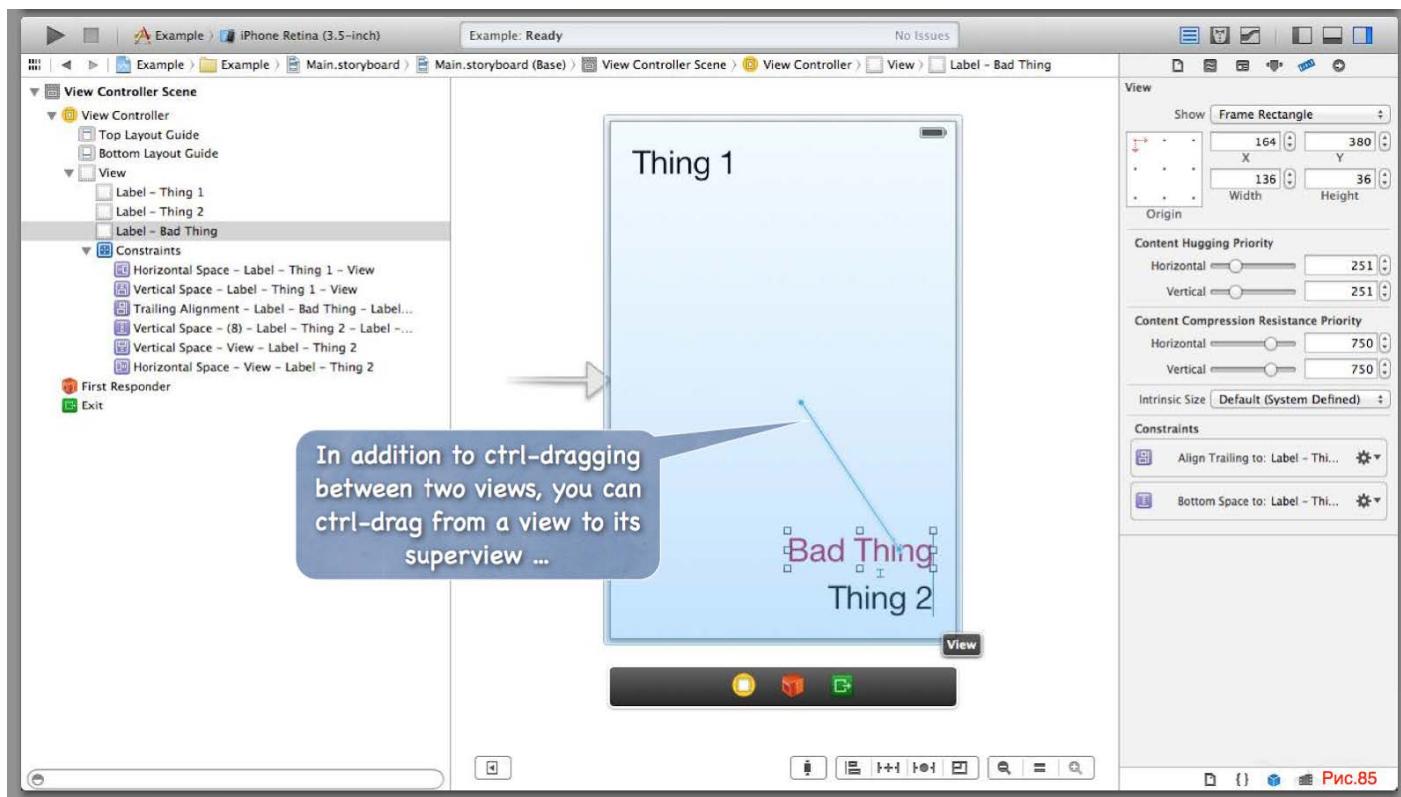


Давайте попробуем режим “Ландшафт”...



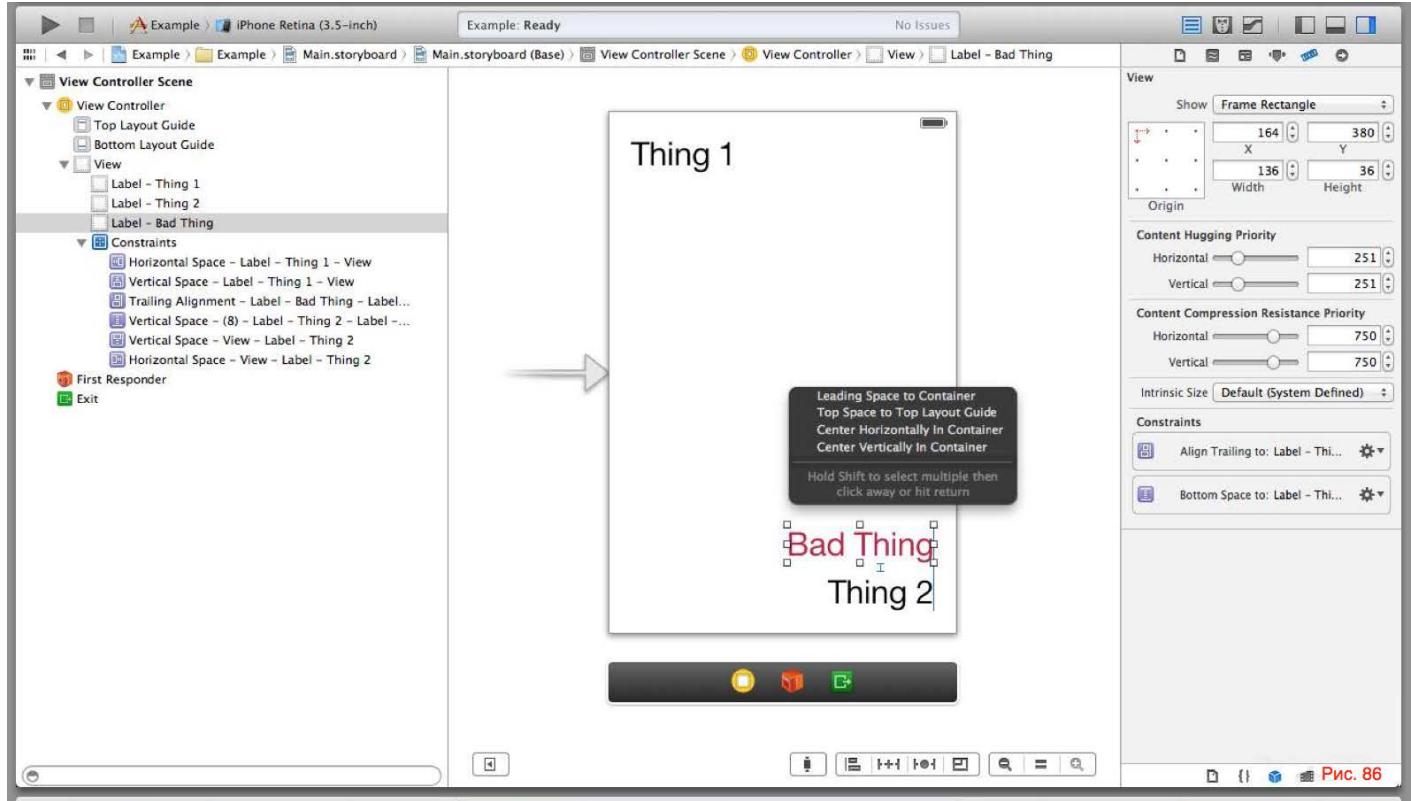
Да!

Итак, это Способ №3 привязывания views друг к другу с помощью CTRL-перетягивания.



В дополнение к CTRL- перетягиванию между двумя views, вы можете сделать CTRL-перетягивание от view к superview...

и вы получите другой перечень опций типа “Leading Space To Container”



Вы также можете сделать CTRL- перетягивание к самому себе, если захотите ограничить width (ширину) или height (высоту)

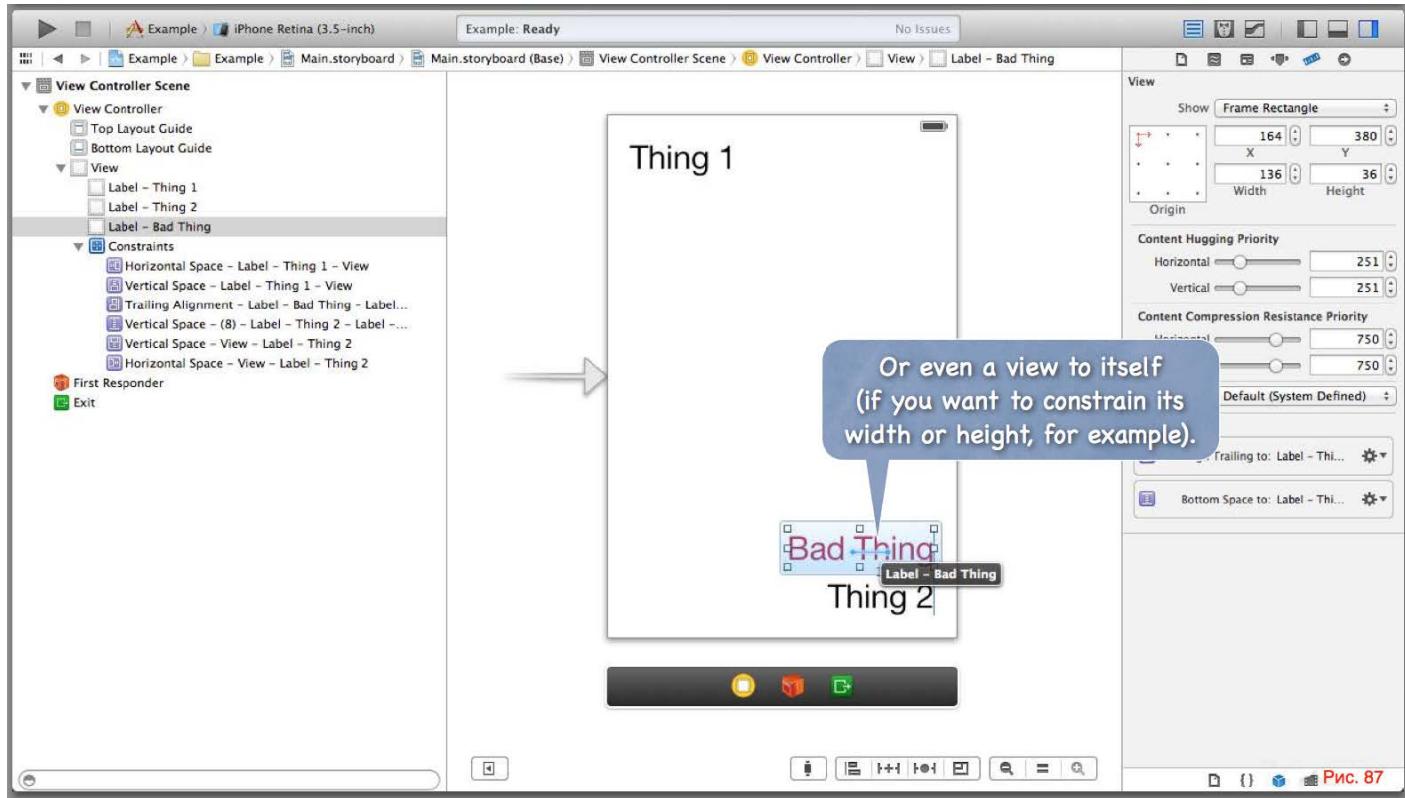
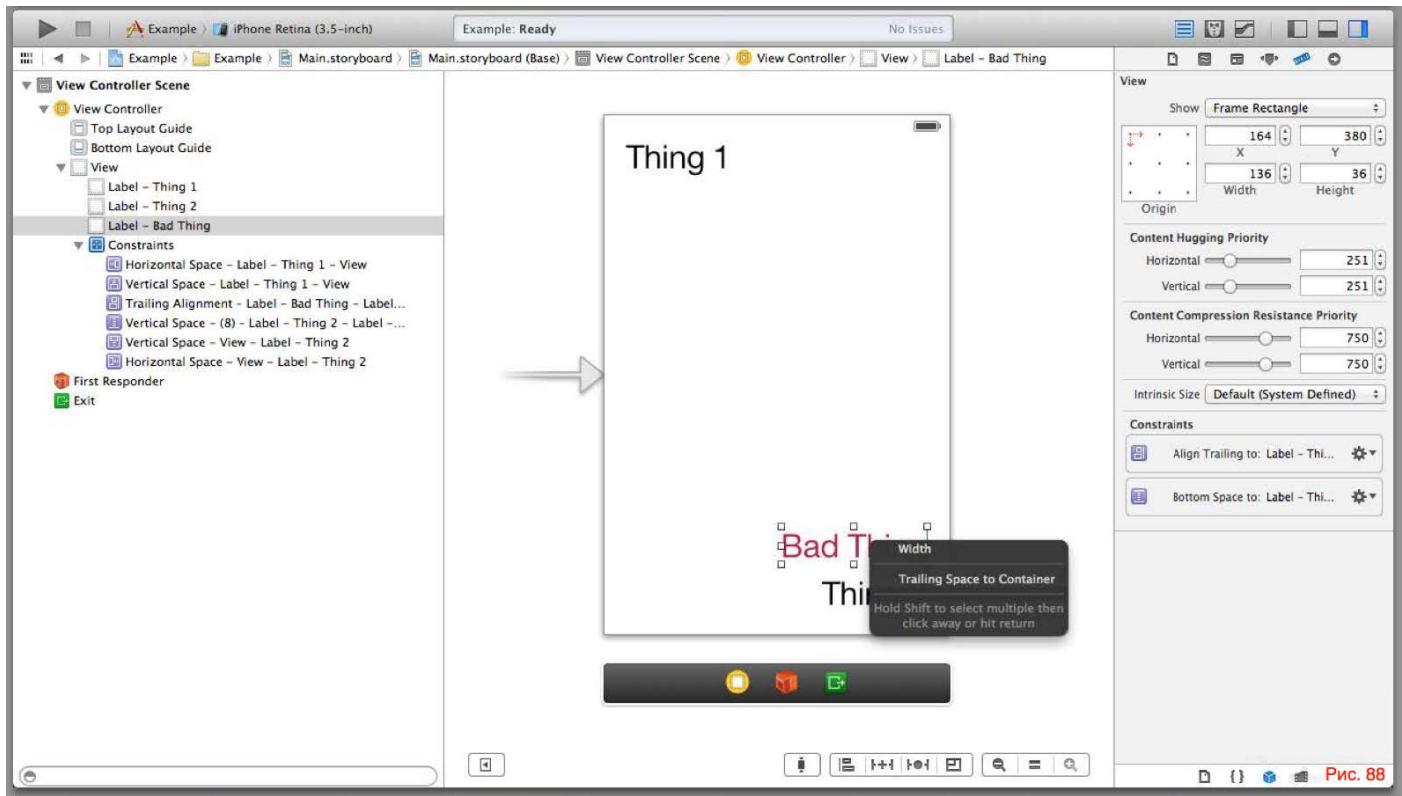


Рис. 87



Это все Способ №3 для задания ограничений.

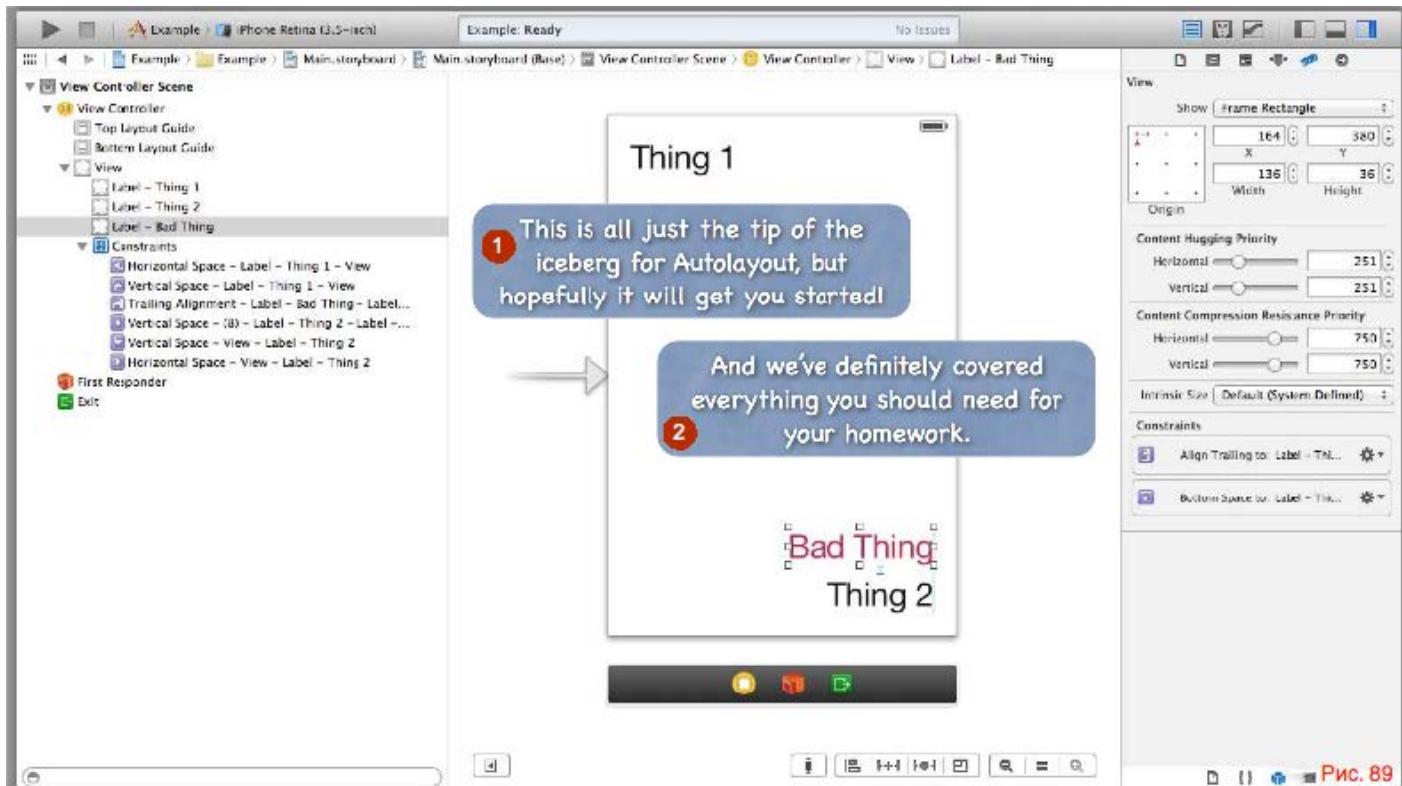


Рис. 88

Рис. 89

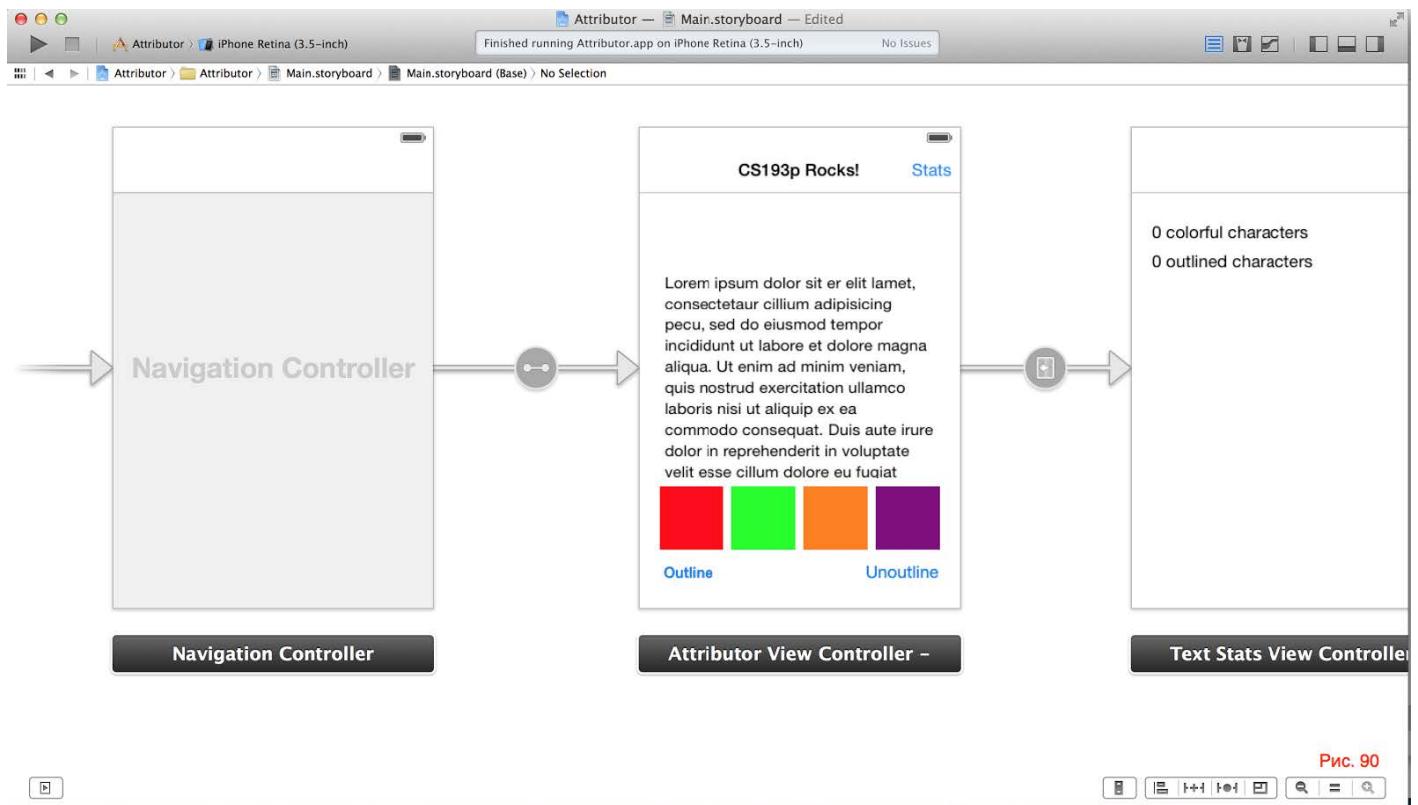
Мы рассмотрели основные приемы работы с системой Autolayout. Как кликнуть на ограничении, как инспектировать их, где искать ограничения, как использовать Document Outline для разрешения проблем, если они обозначены желтым или красным цветом.

Взгляните на Инспектор Размера на слайде. Вы увидите такие характеристики как “Content Hugging Priority” и “Content Compression Resistance Priority”. Иногда у нас есть views, которые делят одно и тоже вертикальное или горизонтальное пространство, и если это пространство исчезает, то они должны знать кто из них будет в большей степени использовать это пространство, а кому потребуется дополнительное, и какое view готово сжиматься, а какое - нет. Для этого надо познакомиться с документацией.

И последняя вещь, о которой мы поговорим сегодня - intrinsic size (внутренний размер). Опять, куча iOS элементов имеют intrinsic size, такие как метки или кнопки, но в действительности на вашем UI могут быть custom views (пользовательские views), которые имеют свой собственный intrinsic size. Для того, чтобы ваше пользовательское view имело свой собственный intrinsic size, необходимо реализовать маленький API, который будет сообщать свой intrinsic size, и тогда маленькое ниспадающее меню сможет рассказать Xcode как управлять intrinsic size.

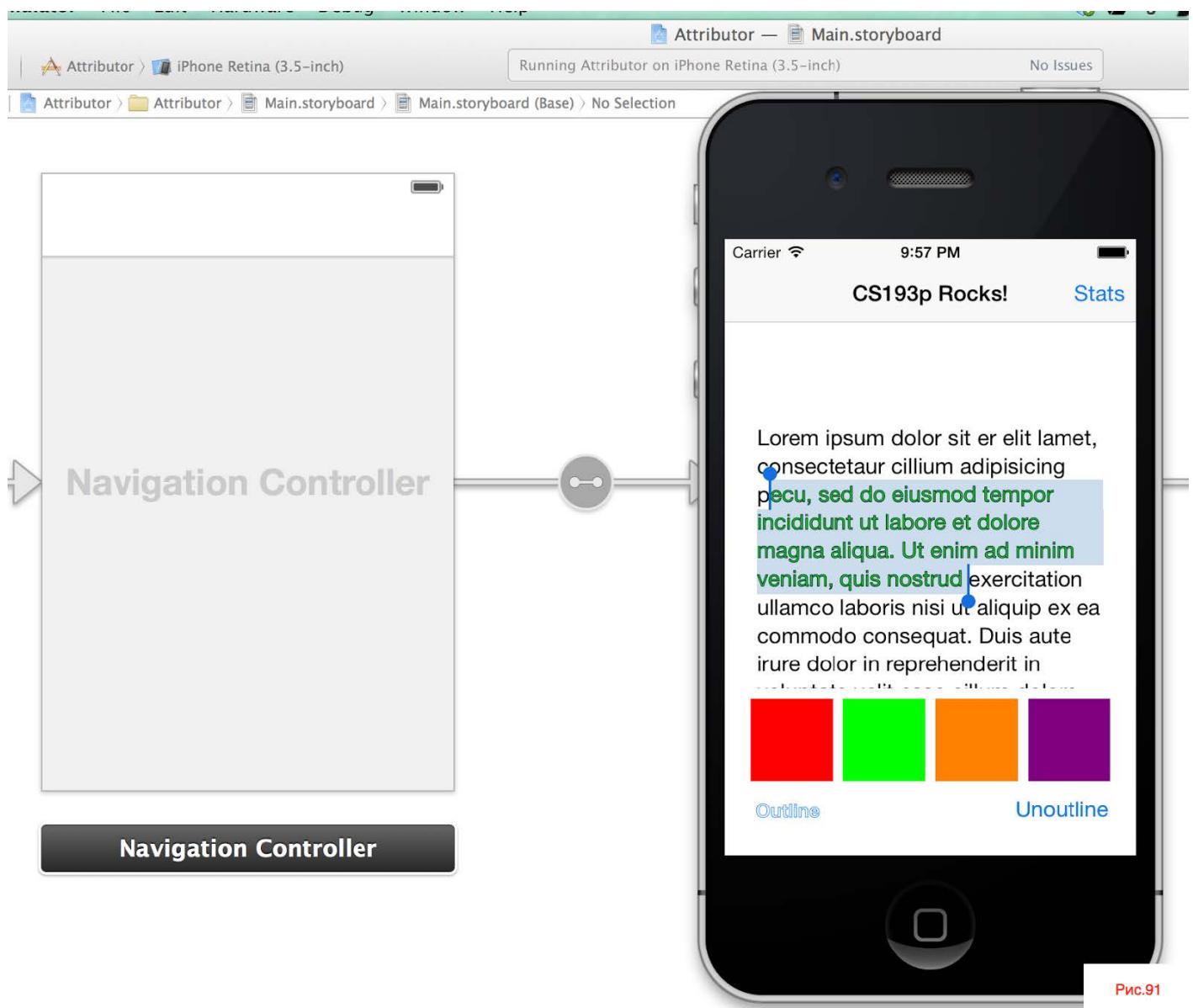
В качестве демонстрационного примера мы возьмем знакомое нам с лекции № 3 приложение Attributor. Мы собираемся заставить Attributor быть Autolayout.

Мы использовали голубые направляющие линии, поэтому выполнить нашу работу будет легко.



Теперь запустим Attributor и выберим некоторый текст

Рис. 90



Посмотрим статистику

Рис.91

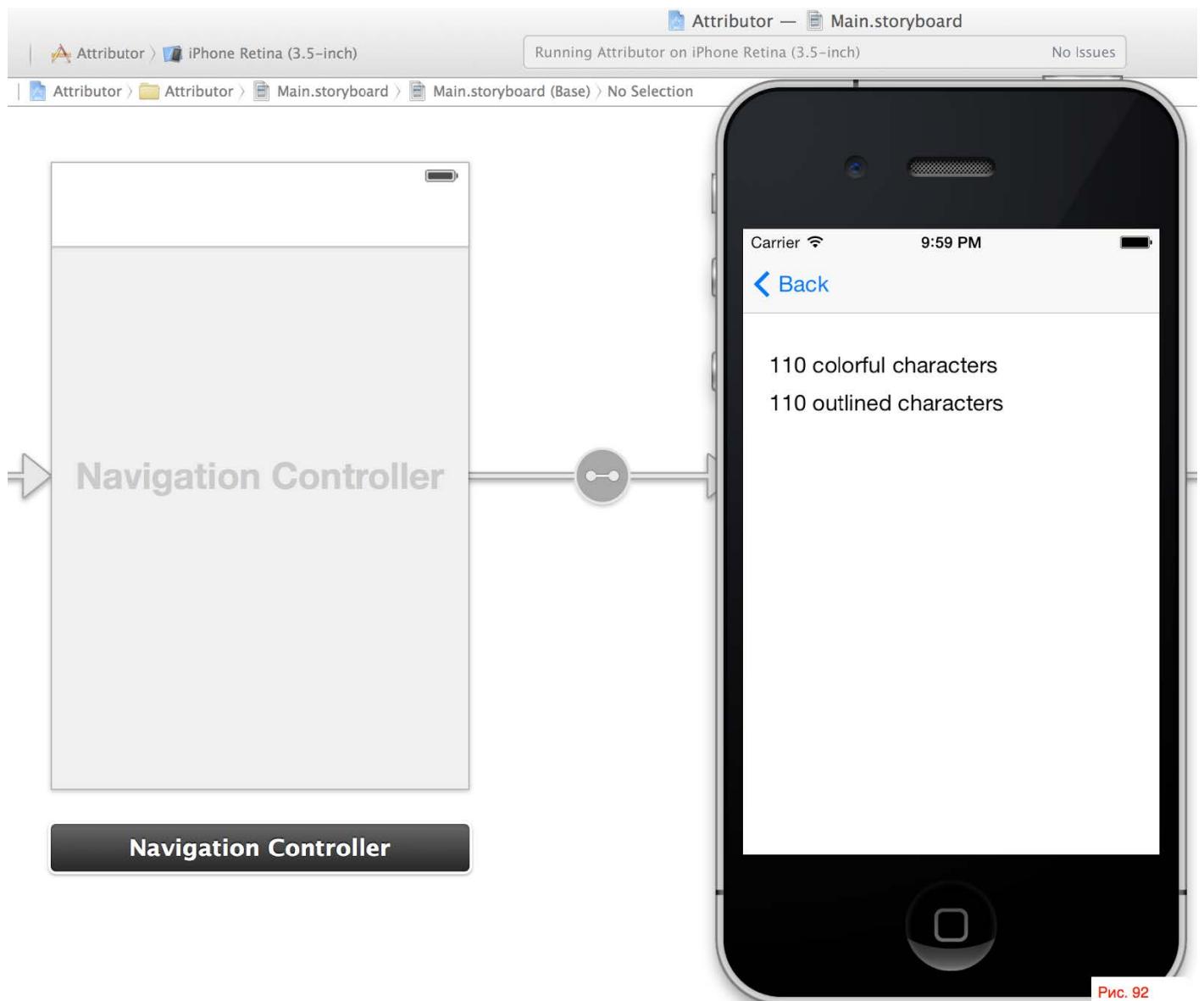
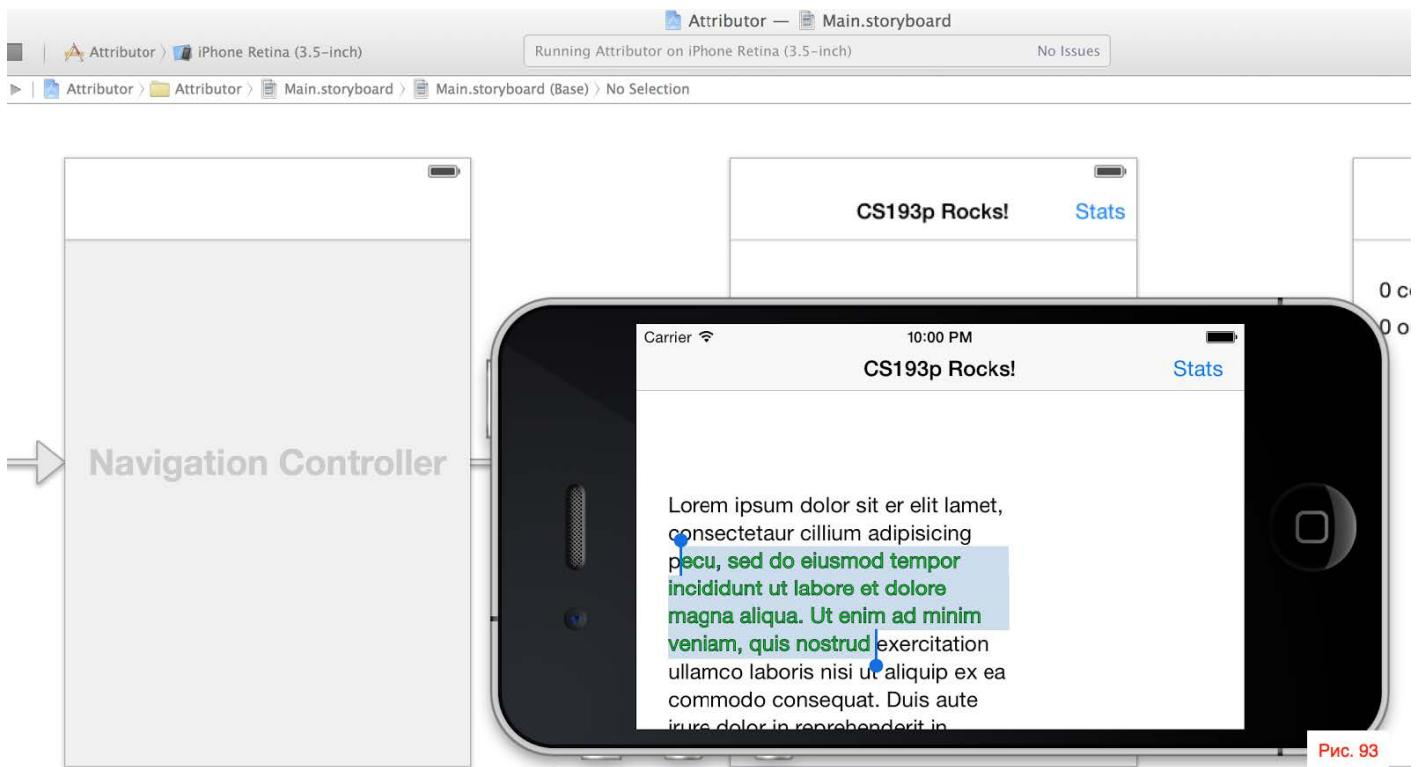


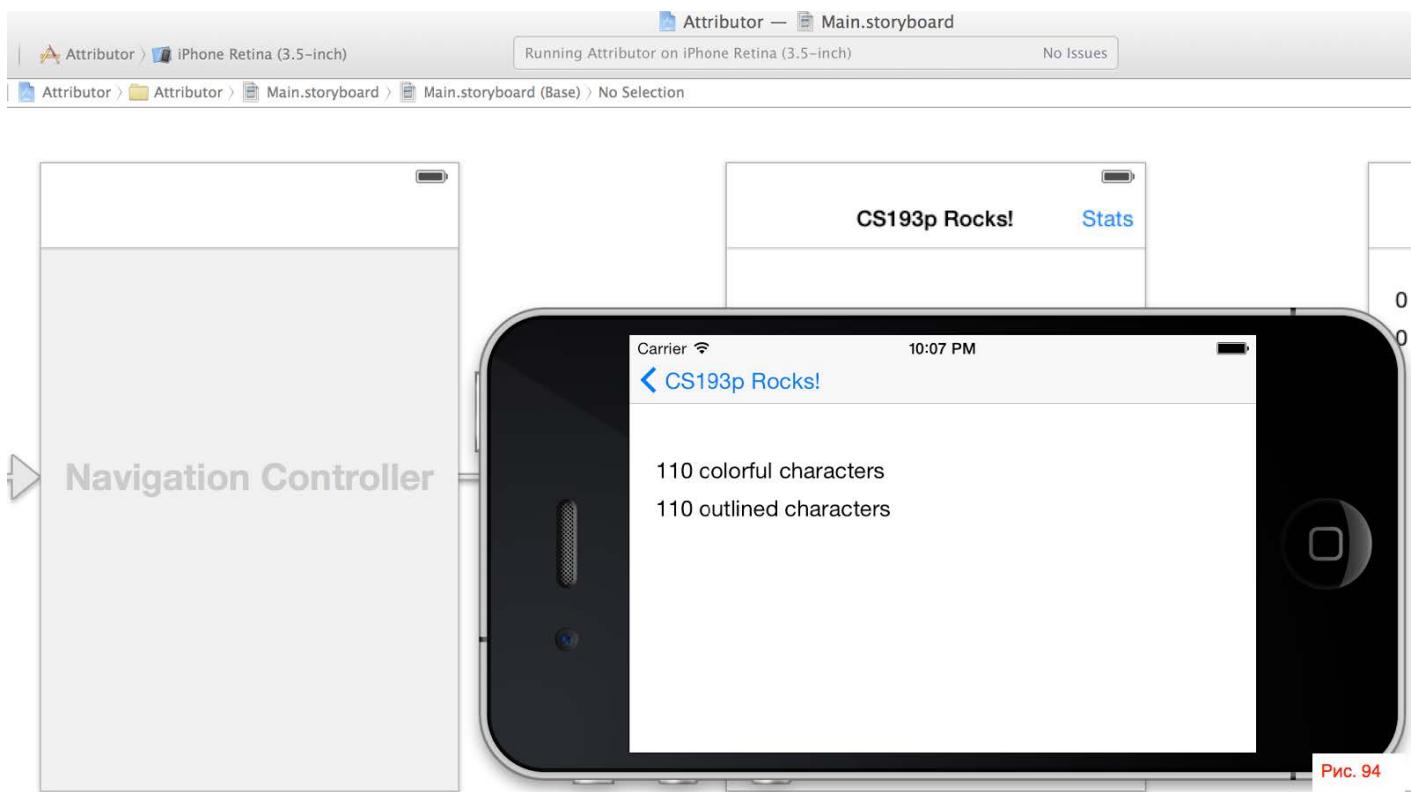
Рис. 92

Теперь повернем iPhone в режим “Ландшафт”. Для симулятора это можно сделать в меню Hardware -> Rotate Left или с помощью клавиш “Command” ←.



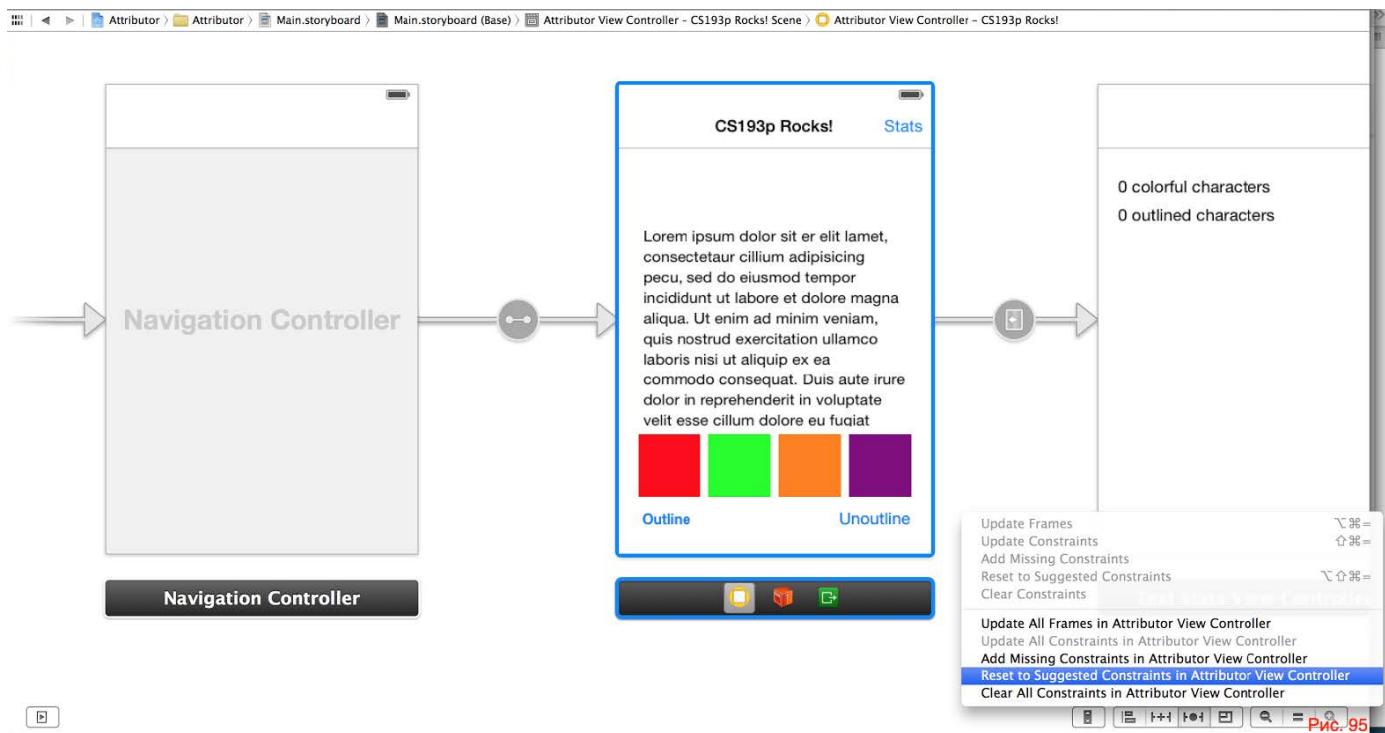
Это выглядит ужасно, потому что мы полностью потеряли управляемые кнопки с цветами, выделением текста и т.д.

При этом экран со статистикой выглядит неплохо.

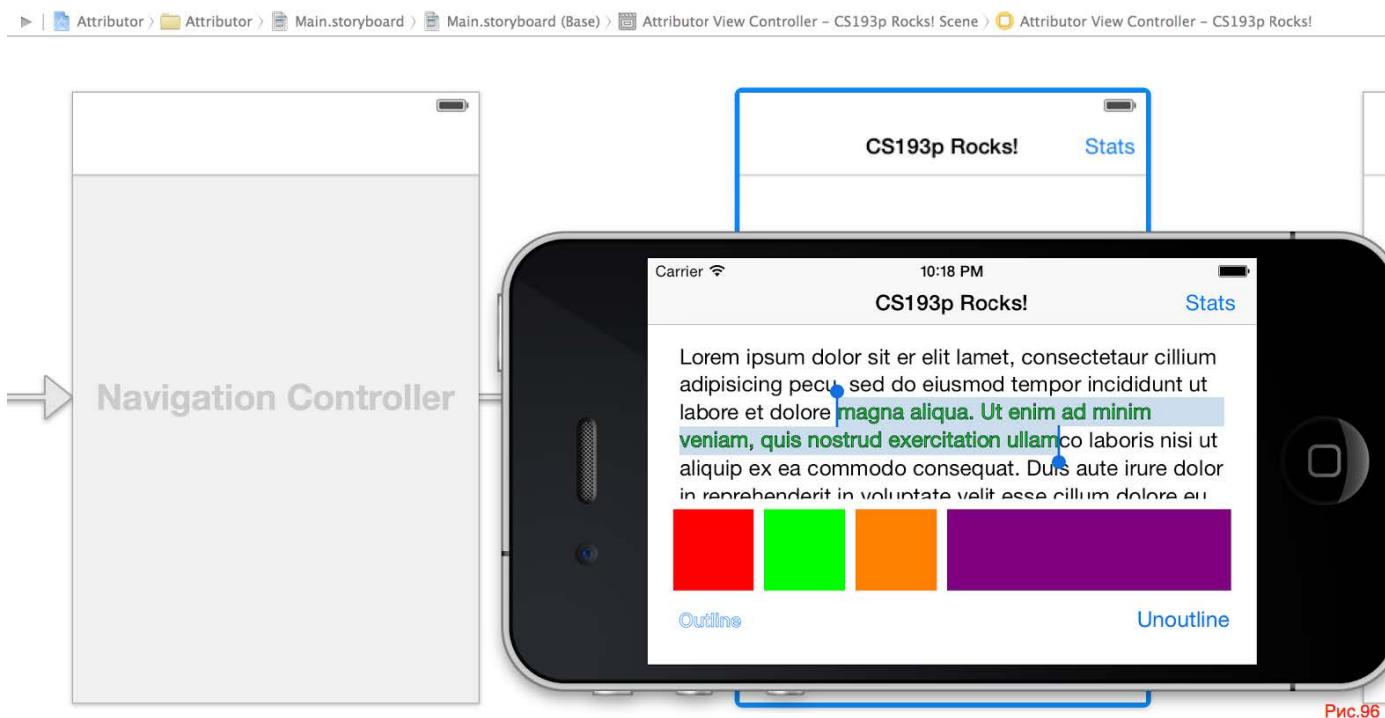


Но на основном экране у нас нет управления. Попробуем это исправить.

Так как при размещении мы использовали голубые направляющие линии, давайте попробуем применить “Reset To Suggested Constraints”



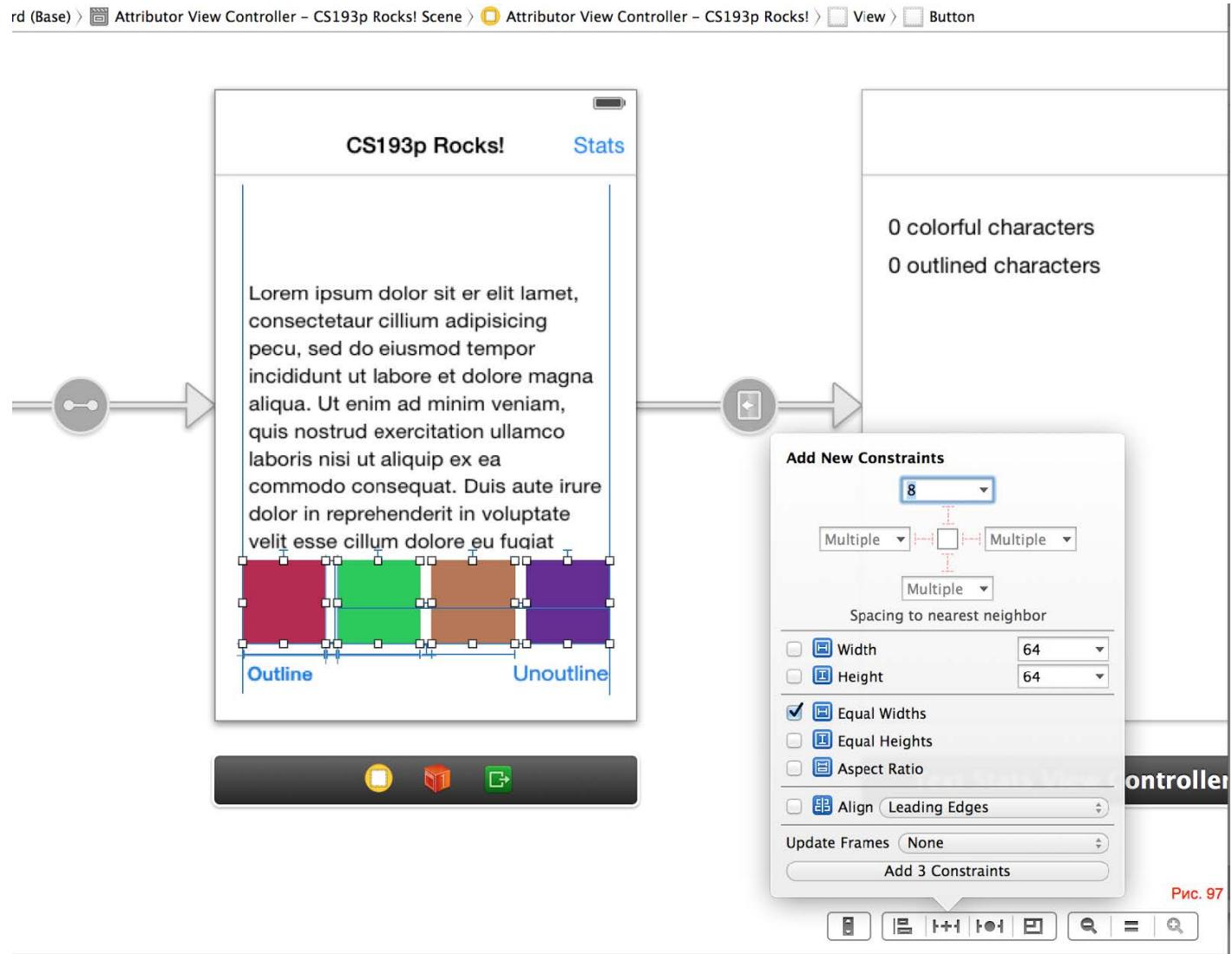
И посмотрим, что получится при запуске и вращении iPhone.



Результат значительно лучше: мы восстановили кнопки управления цветом и обводкой, но почему-то кнопка с фиолетовым цветом значительно шире других.

Мы должны сделать все кнопки с цветом одинакового размера. Как? Очень просто.

При размещении кнопок на storyboard мы ничего не сделали, чтобы подсказать Xcode, что кнопки с цветом все одинакового размера.



Я добавил 3 ограничения, нажав кнопку “Add 3 Constraints”

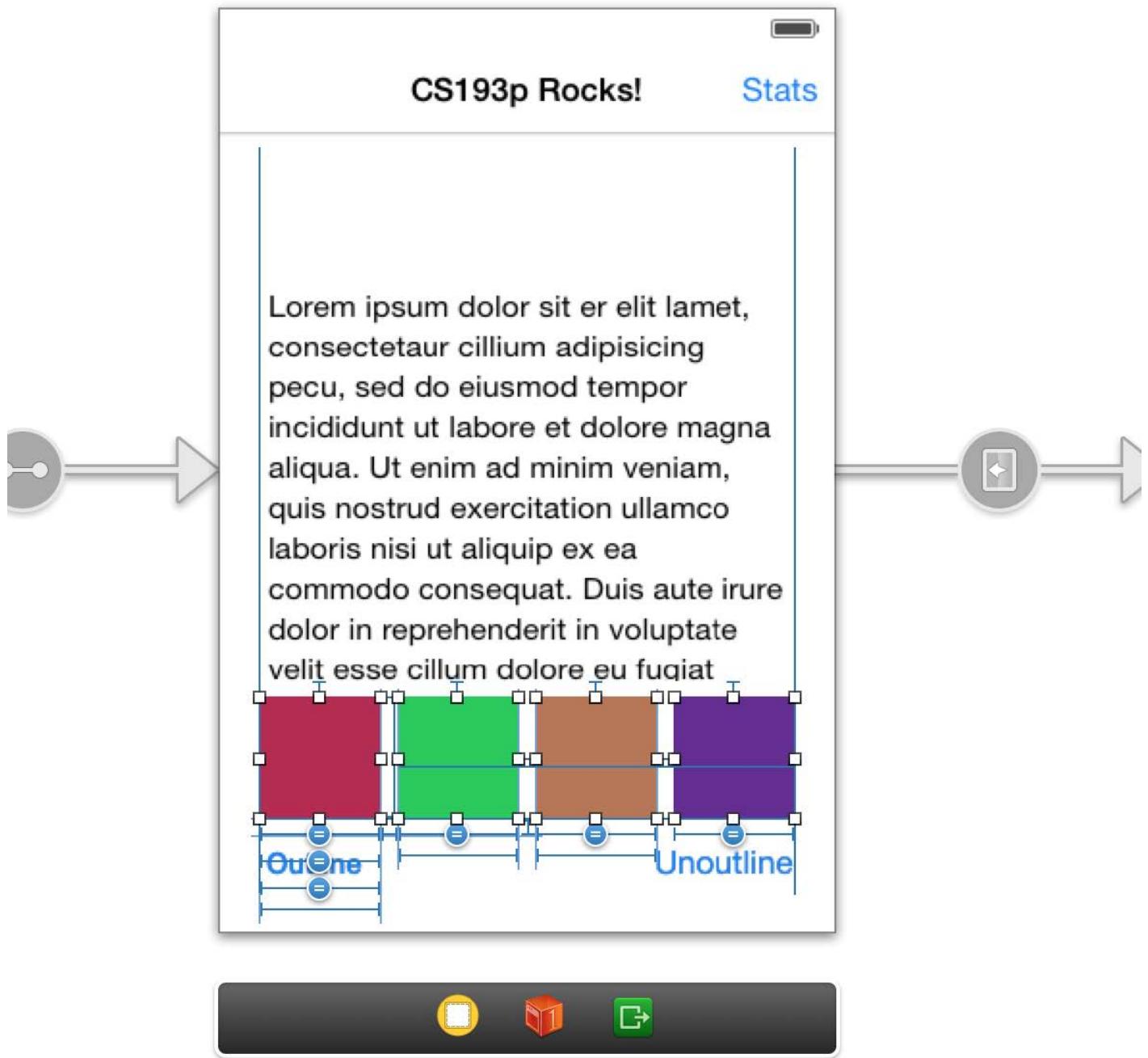


Рис. 98

Запускаем и смотрим в режиме “Ландшафт”

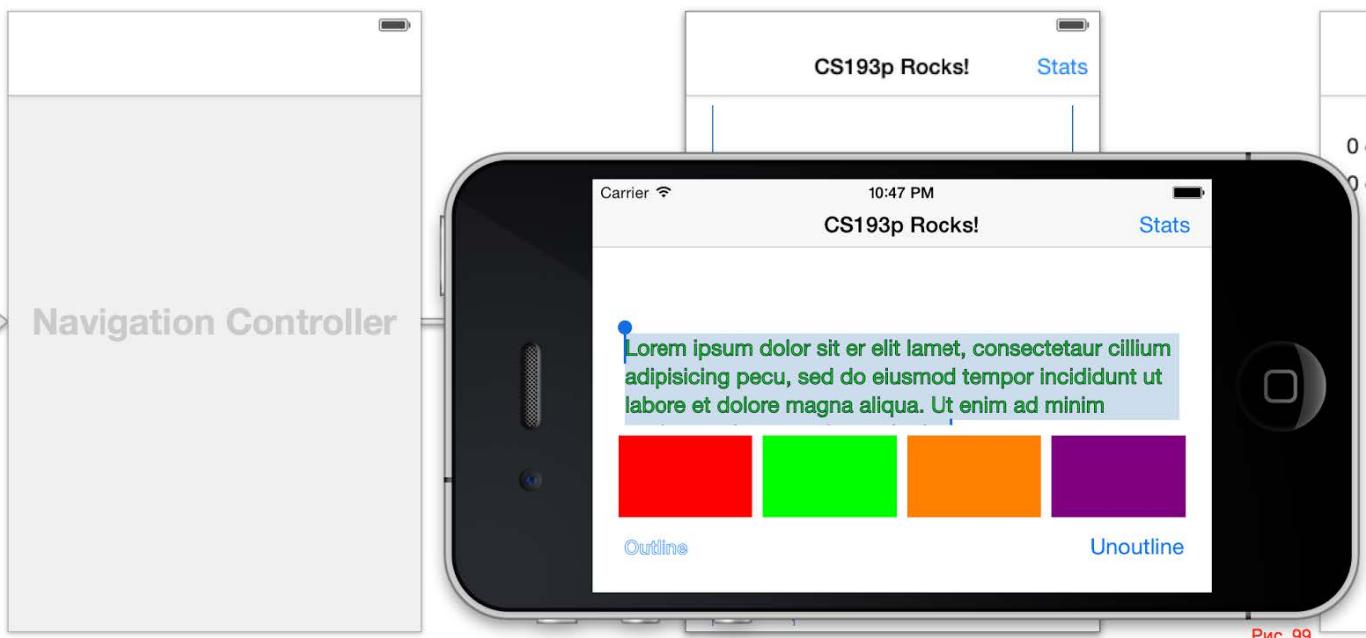


Рис. 99

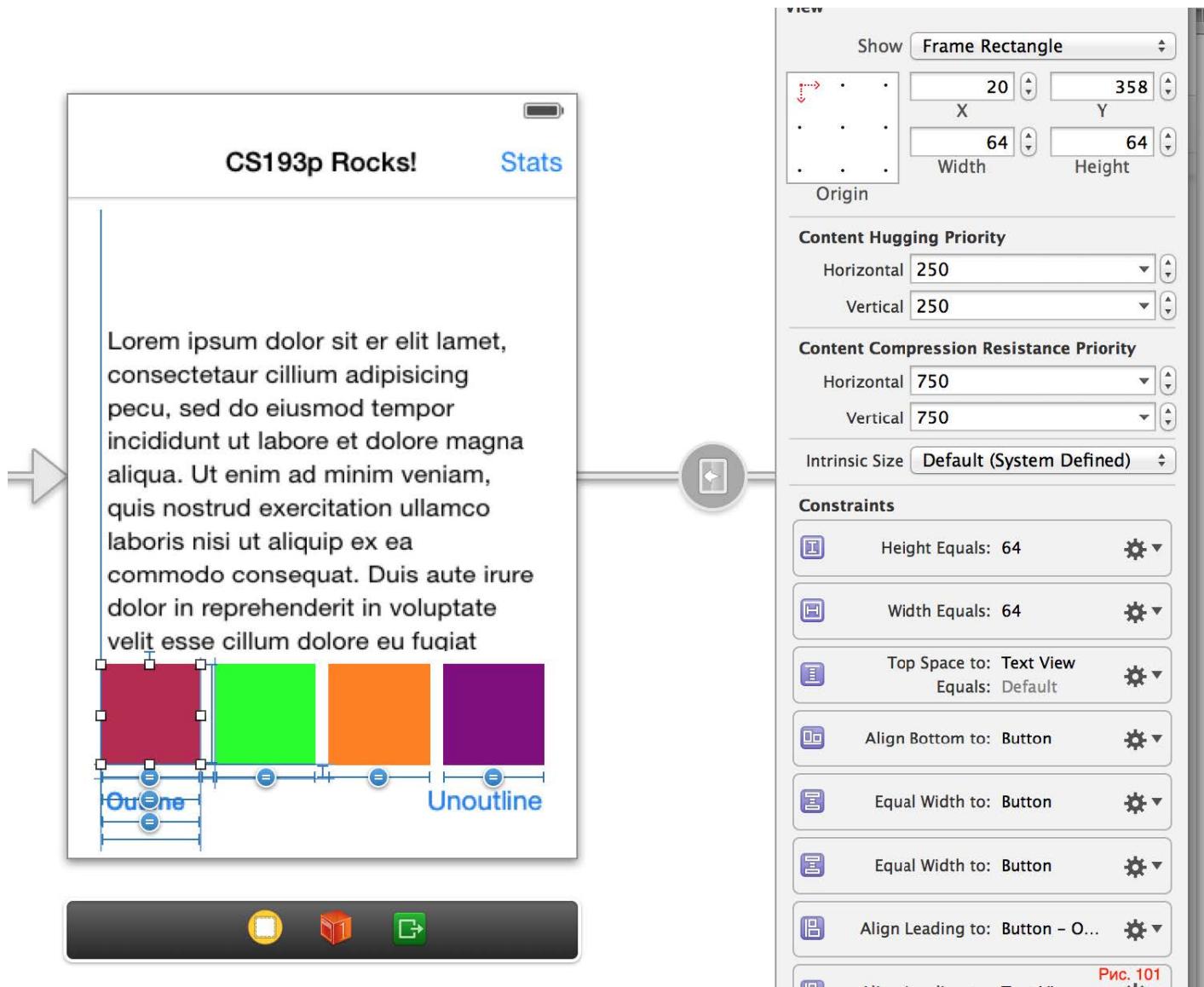
Все прекрасно, но в режиме вращения “upside down” вращения не происходит, потому в установках нашего проекта мы указали, что не поддерживаем режим “Портрет вверх ногами” “upside down”. Поэтому при вращении в это положение ничего не происходит. В других режимах все работает прекрасно, но если мы посмотрим на консоль, то обнаружим там множество ошибок и основная ошибка “**Unable to simultaneously satisfy constraints.**” (то есть невозможно одновременно удовлетворить всем ограничениям).

Но мы не видим никаких ни желтых, ни красных линий или кружочков. Нет никаких предупреждений и в Document Outline. Но появление на консоле сообщений говорит о том, что они избыточны. Если мы посмотрим на консоль, то там выдается много информации с описанием каждого ограничения

```
2014-05-30 22:47:19.731 Attributor[6072:60b] Unable to simultaneously satisfy constraints.
Probably at least one of the constraints in the following list is one you don't want. Try this: (1)
look at each constraint and try to figure out which you don't expect; (2) find the code that added the
unwanted constraint or constraints and fix it. (Note: If you're seeing
NSAutoresizingMaskLayoutConstraints that you don't understand, refer to the documentation for the
UIView property translatesAutoresizingMaskIntoConstraints)
(
    "<NSLayoutConstraint:0x8d45840 H:[UIButton:0x8d45670(64)]>",
    "<NSLayoutConstraint:0x8d461e0 H:[UIButton:0x8d46030(64)]>",
    "<NSLayoutConstraint:0x8d45eb0 H:[UIButton:0x8d45d00(64)]>",
    "<NSLayoutConstraint:0x9985be0 H:[UITextView:0x9411a00]-(NSSpace(20))-|  (Names: '|':UIView:
0x9985170 )>",
    "<NSLayoutConstraint:0x9985c60 H:|-(NSSpace(20))-[UITextView:0x9411a00] (Names: '|':UIView:
0x9985170 )>",
    "<NSLayoutConstraint:0x9985c90 UITextView:0x9411a00.trailing == UIButton:0x8d45140.trailing>",
    "<NSLayoutConstraint:0x9985db0 UIButton:0x8d45670.leading == UITextView:0x9411a00.leading>",
    "<NSLayoutConstraint:0x9985de0 UIButton:0x8d45670.width == UIButton:0x8d45140.width>",
    "<NSLayoutConstraint:0x9985e70 H:[UIButton:0x8d45670]-(NSSpace(8))-[UIButton:0x8d46030]>",
    "<NSLayoutConstraint:0x9985ed0 H:[UIButton:0x8d46030]-(NSSpace(8))-[UIButton:0x8d45d00]>",
    "<NSLayoutConstraint:0x9985ff0 H:[UIButton:0x8d45d00]-(NSSpace(8))-[UIButton:0x8d45140]>",
    "<NSAutoresizingMaskLayoutConstraint:0x9999df0 h==& v==& H:[UIView:0x9985170(480)]>"
)
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x8d45eb0 H:[UIButton:0x8d45d00(64)]>
```

Рис. 100

Такая ошибка связана с тем, что мы сделали наши кнопки одинаковой ширины. Давайте взглянем на ограничения

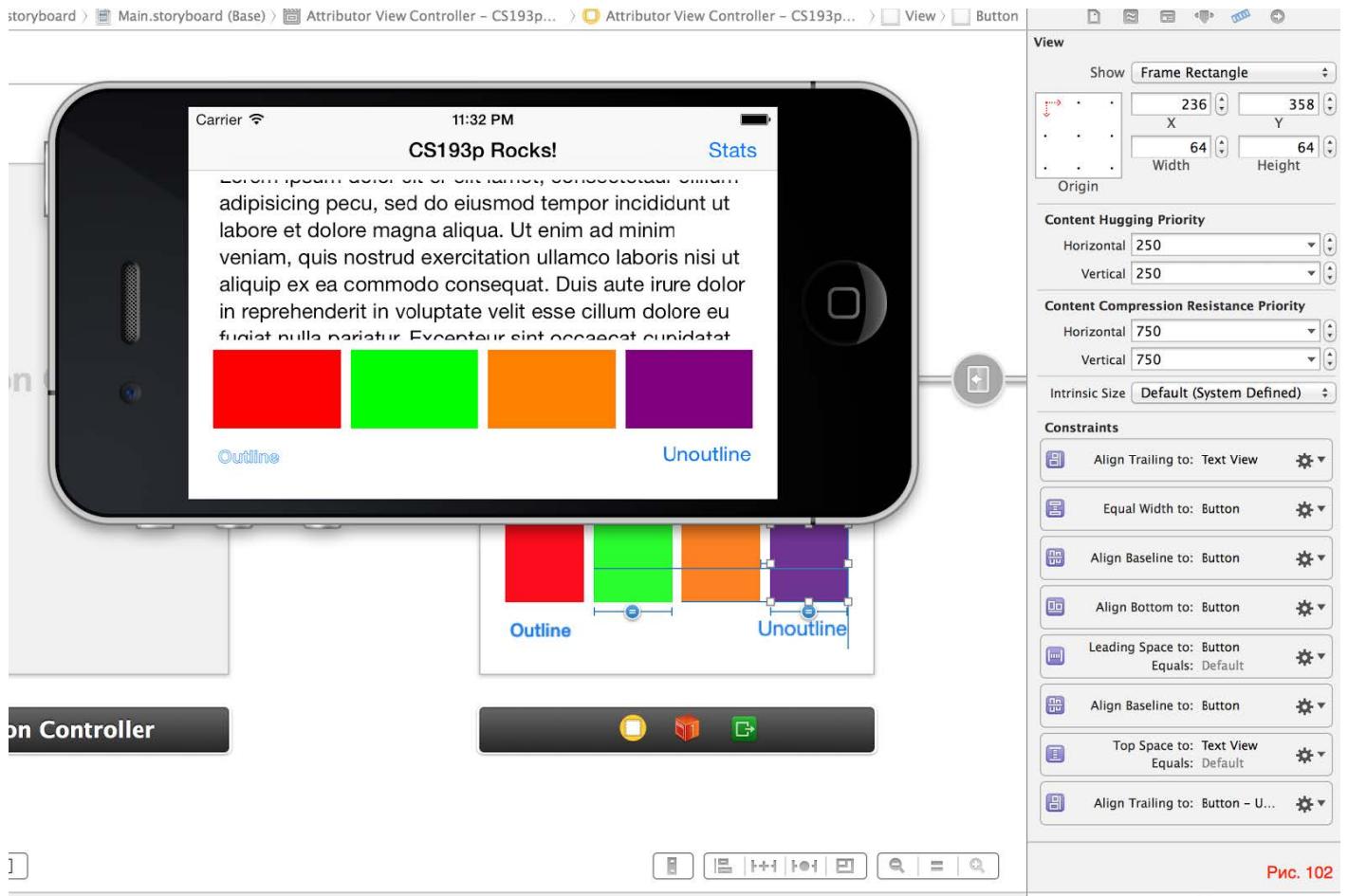


Мы видим здесь “магическое число” 64 и это плохо. Мы знаем, что в режиме “Ландшафт” ширина этих кнопок все равно не останется 64, а будет несколько больше и в этом проблема. Поэтому следует удалить все ограничения на width (ширина) с этим “магическим числом” 64.

У меня не появилось желтых линий или кружков - значит нет пропущенных ограничений, то есть все определено.

Запускаем и смотрим в режиме “Ландшафт”

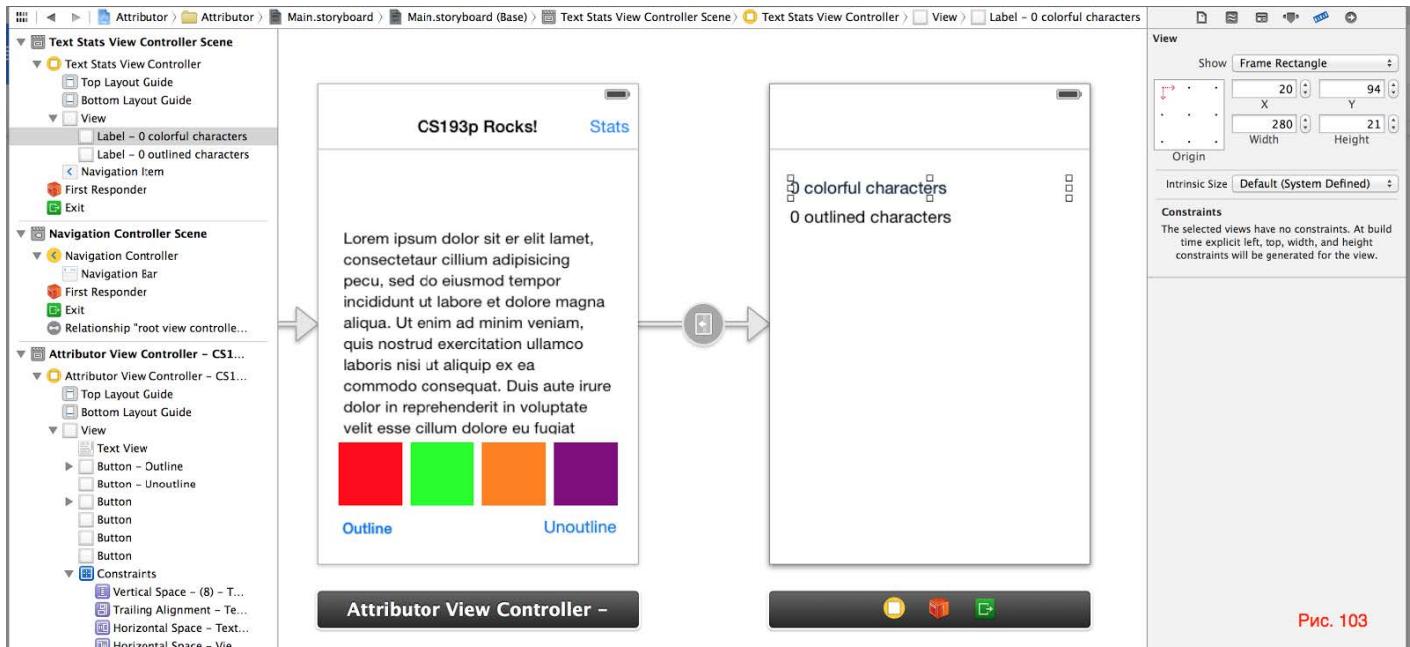
Рис. 101



Все работает и на консоле нет никаких сообщений.

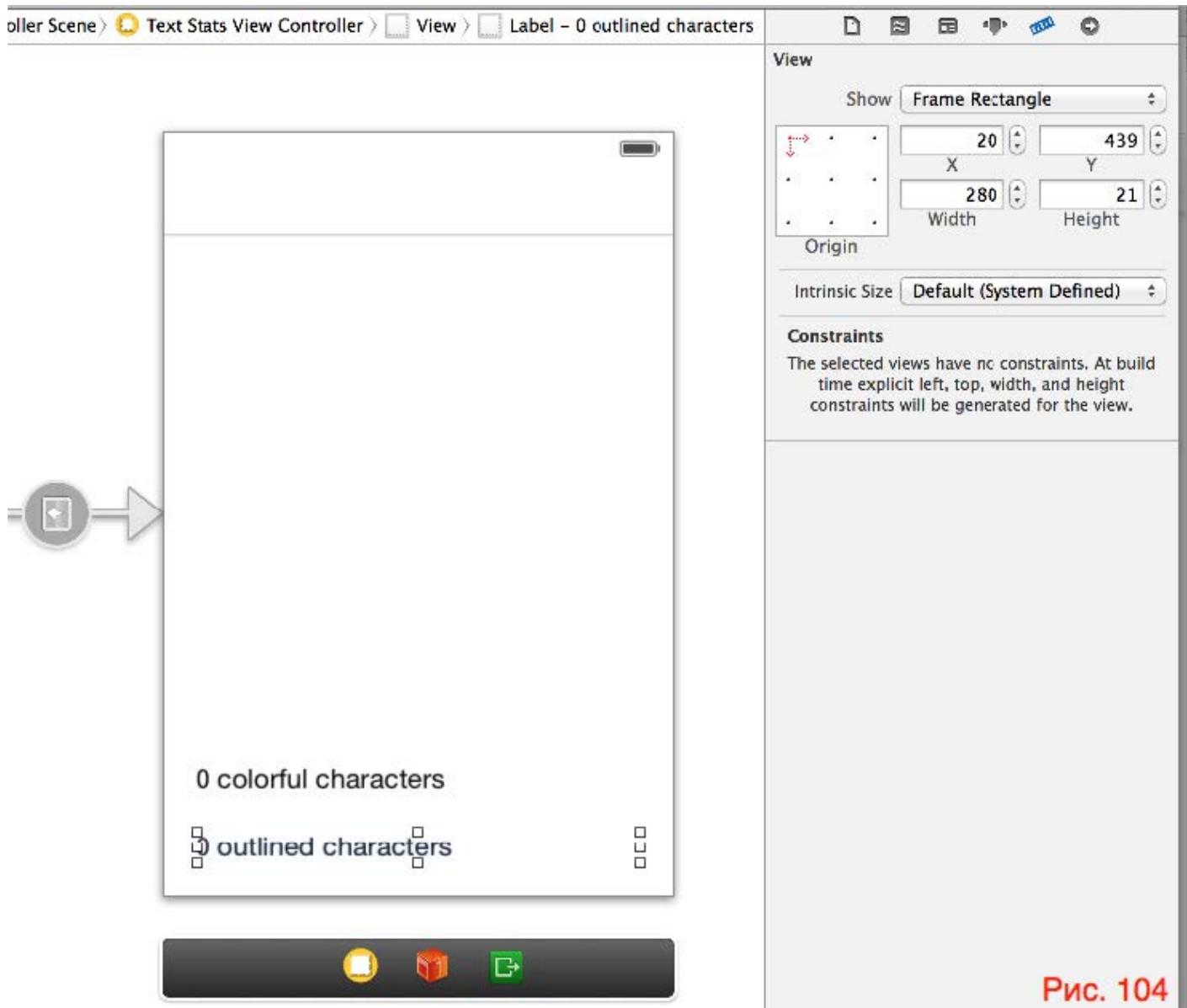
Иногда вы можете получить на консоле сообщение о том, что некоторые ограничения имеют проблемы в Xcode. Нужно внимательно смотреть на полученные описания конфликтных ограничений.

Еще одну вещь, которую мы посмотрим, - это экран со статистикой: мы помещаем две текстовые метки с количеством цветных символов и количеством обведенных символов. Мы поместили эти метки в случайные места без голубых направляющих линий и у них нет ограничений, что очень плохо



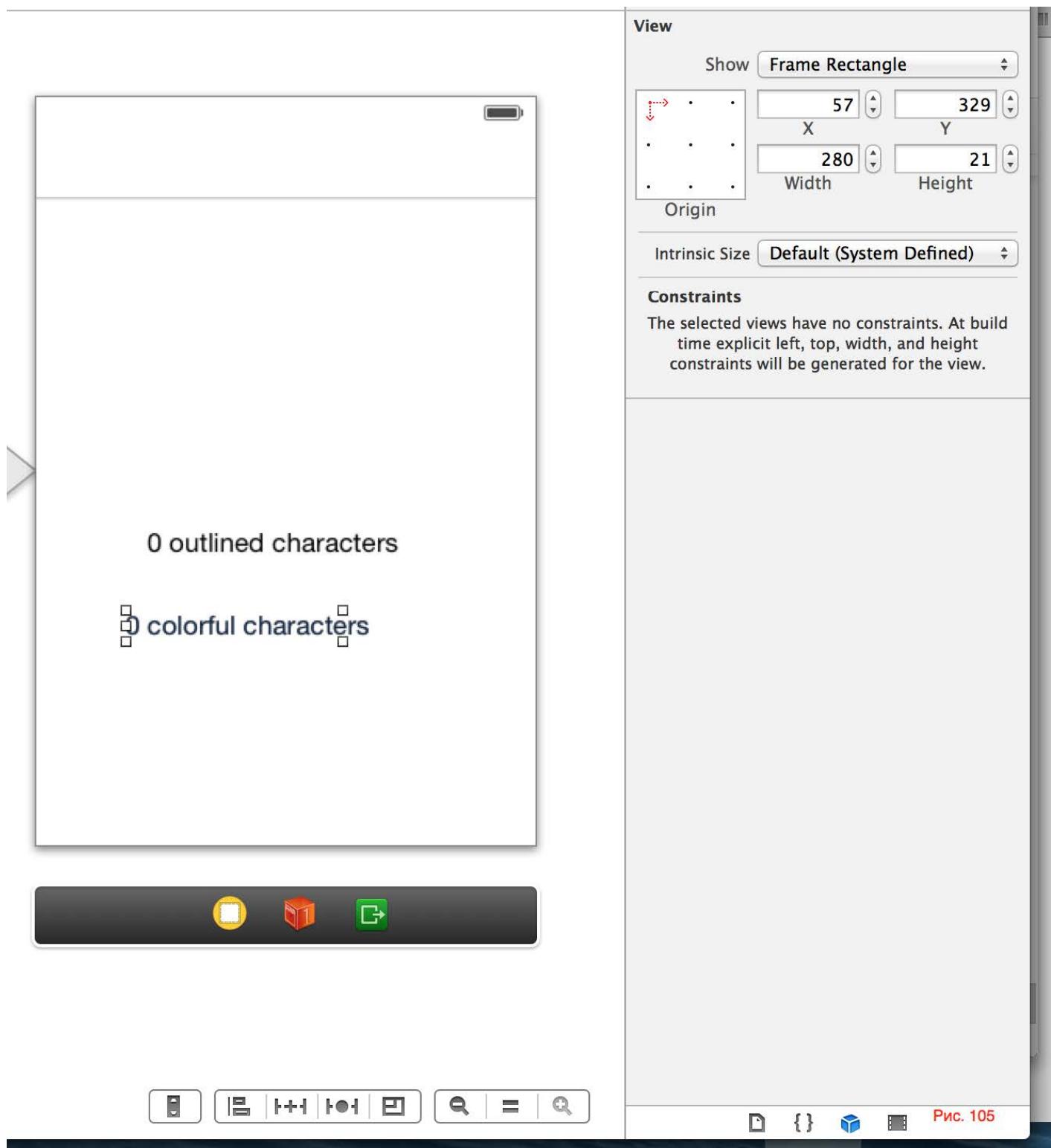
У нас есть шанс показать, как разместить эти метки на правильные места, не используя Suggested (предлагаемых) ограничений.

Мы рассмотрим два способа как это сделать, и оба способа не используют Suggested (предлагаемых) ограничений. Нам нужно, чтобы метки располагались следующим образом



Нам необходимо, чтобы метки располагались внизу: одна - четко в левом нижнем, а вторая строго над ней, но чтобы голубые направляющие линии не использовались, так как мы рассматриваем другой способ расположения объектов.

Для этого переместим метки на произвольное место



Итак метка с цветными символами не имеет в данный момент ограничений, но нам нужно, чтобы она находилась точно в левом нижнем углу, как если бы ее положение определялось голубыми направляющими линиями. Мы будем это делать с помощью маленькой кнопки внизу экрана для

привязки views и определять там два “луча” привязки нашей метки к views, находящимся вокруг. Этим view, находящимся вокруг, может быть superview или какое-нибудь другое view. Мы зафиксируем расстояние слева и снизу и выставим их значения равными Standard Value

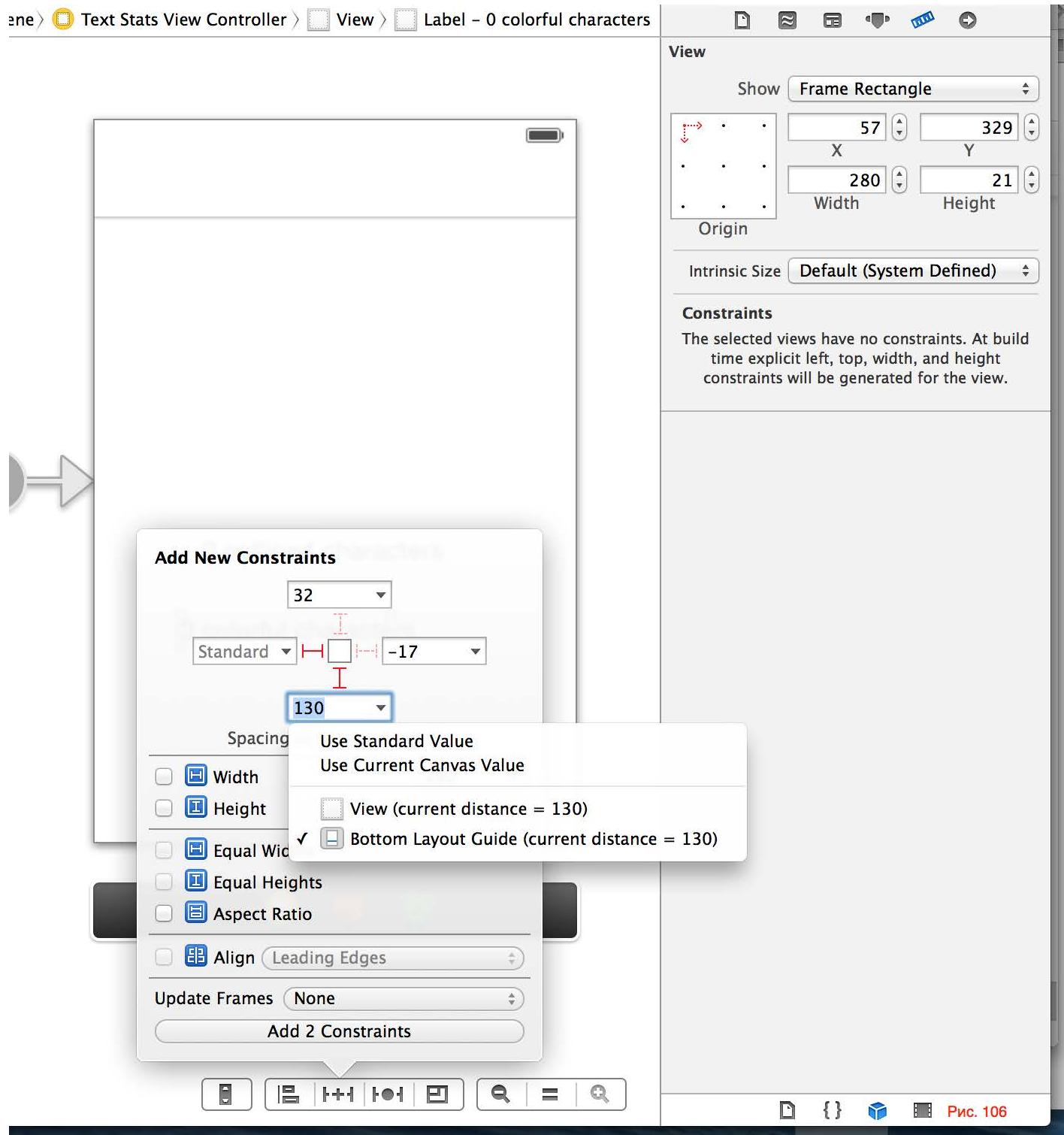
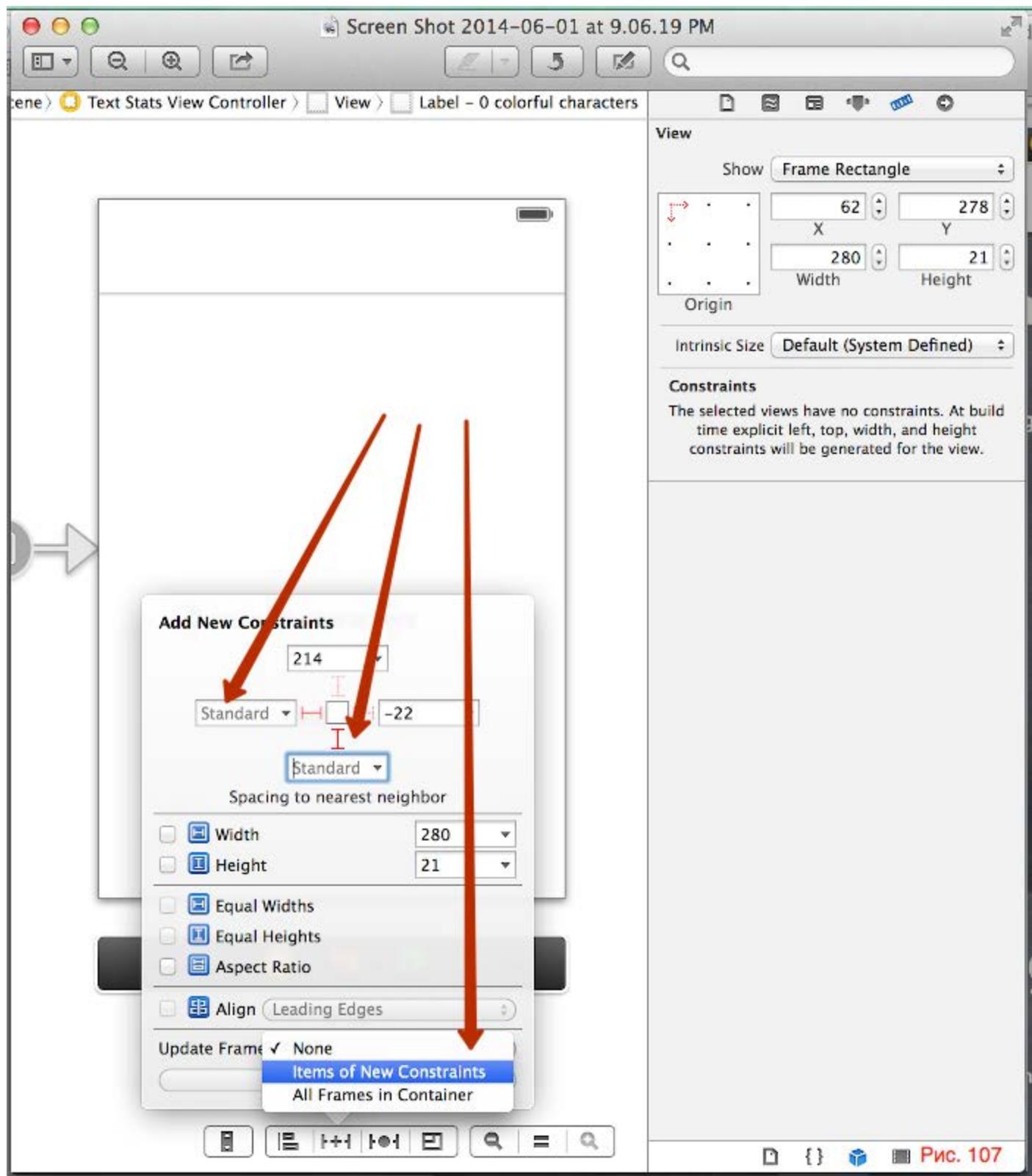
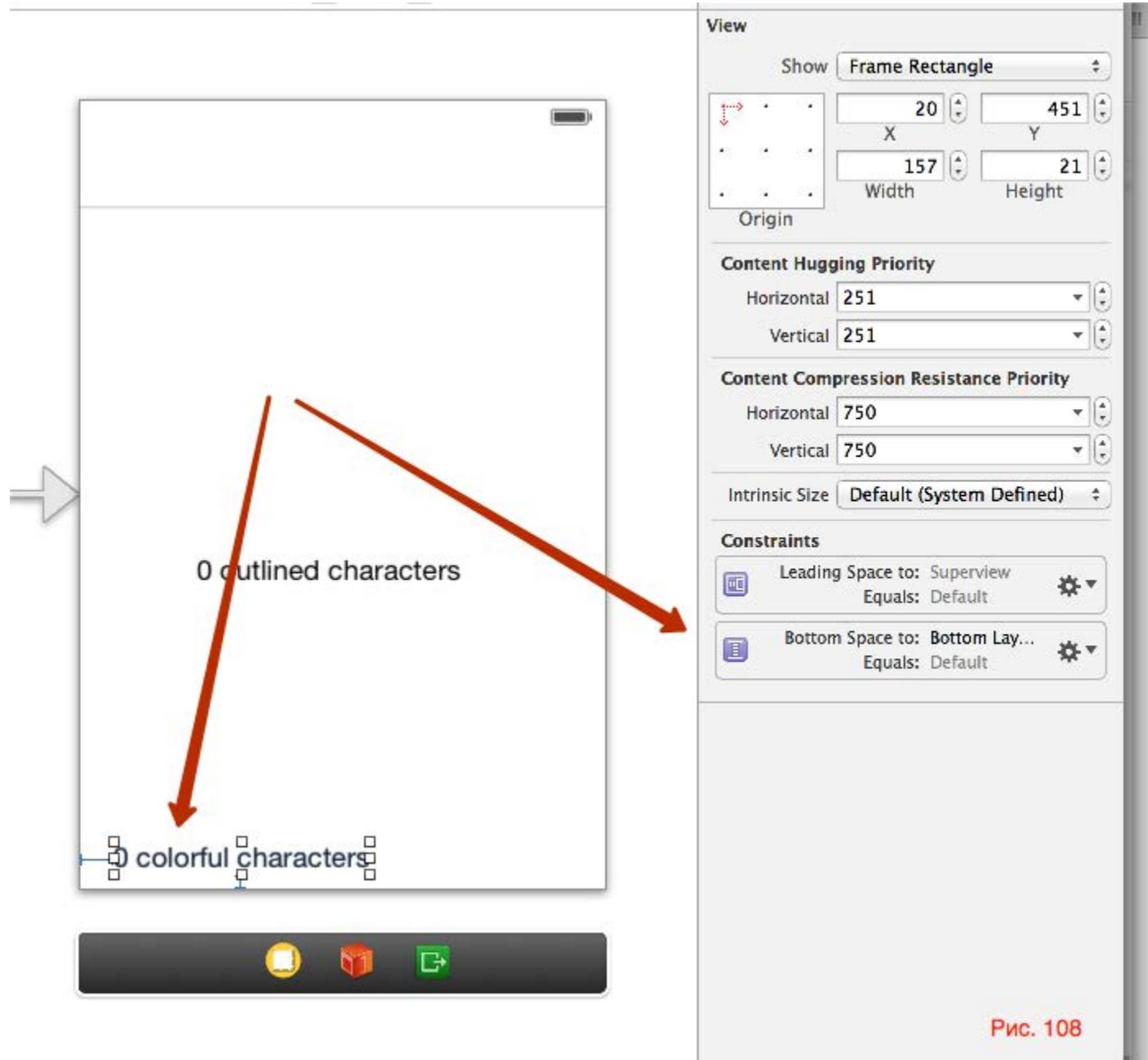


Рис. 106

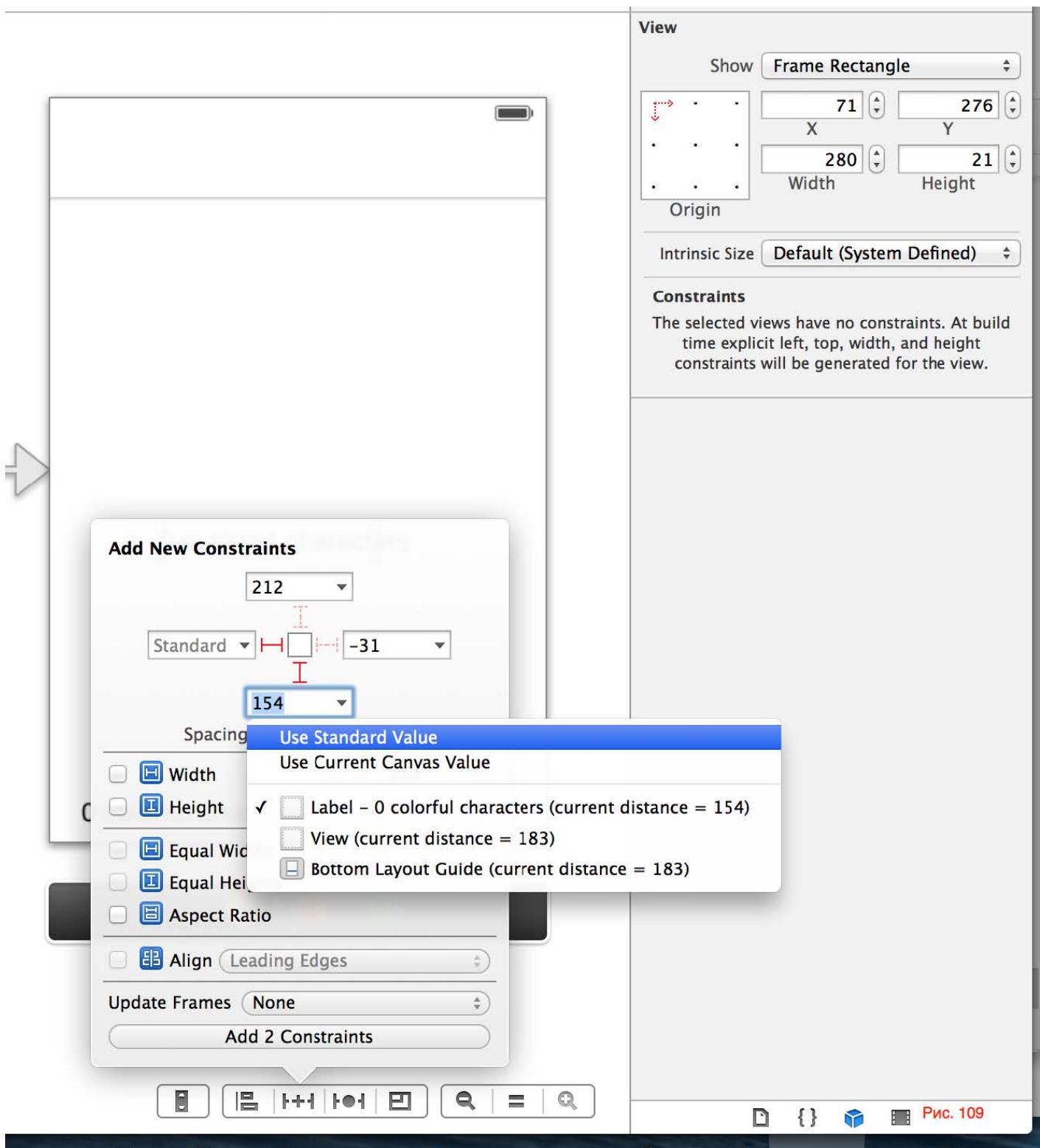
В результате получу следующее



и прежде, чем добавить два ограничения, мы используем опцию “Items of New Constraints” для поля выбора Update Frames, что означает перенос view (то есть нашей метки) на новую позицию. Вы можете перенести все views в этом контейнере при выборе опции “All Frames in Container” или только view, которое получило новые ограничения. Мы выбрали опцию “Items of New Constraints”, поэтому изменит свой frame только наша метка. Итак, мы нажимаем кнопку “Add 2 constraints” и смотрим, что будет.

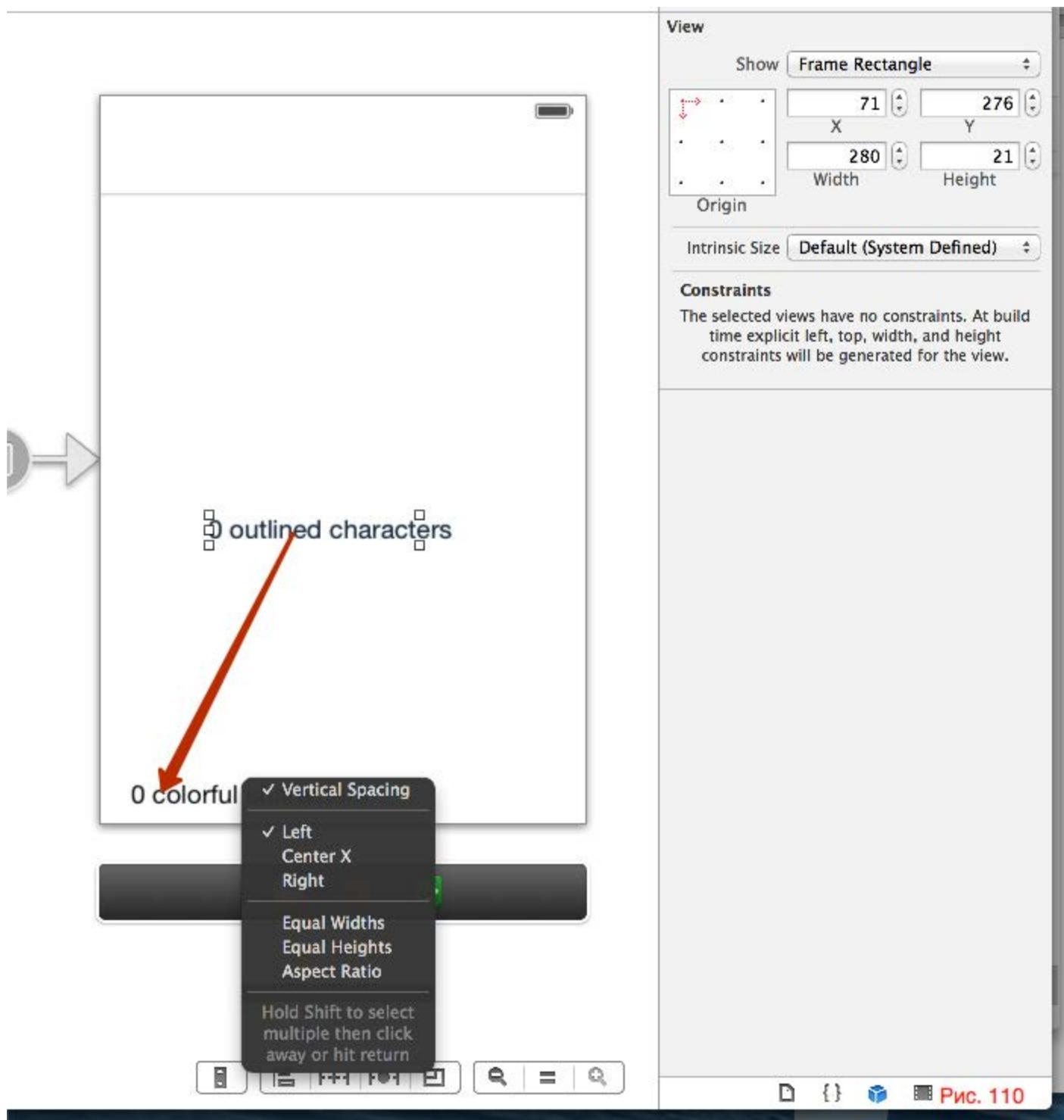


Метка “прыгнула” автоматически в нижний левый угол, как мы того и хотели, и добавились два ограничения. Это еще один способ установки ограничений, мы установили “стандартные” ограничения. Как насчет другой метки? Мы хотим, чтобы она располагалась над меткой с цветными символами. Мы могли бы выполнить это тем же способом, и задавая левый “лучик” по отношению к superview, нижний “лучик” по отношению к нашей первой метке

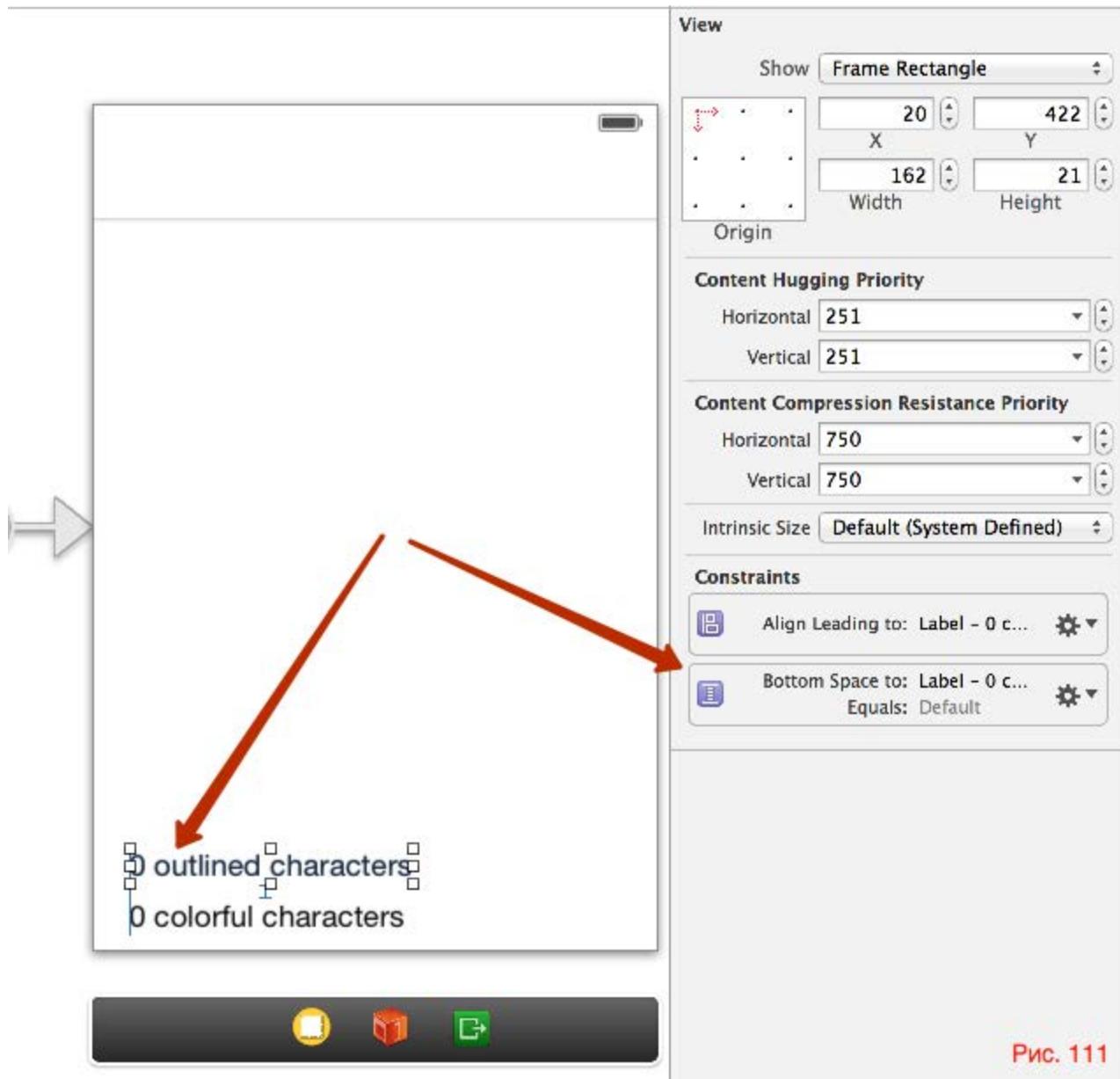


Для обоих “лучиков” нужно было бы выставить значение Standard.

Однако мы будем использовать CTRL- перетаскивание от нашей метки к первой метке.



Выбираем “Vertical Spacing” и “Left” и нажму Return.



Создалось два ограничения и теперь посмотрим, как это работает в режиме “Ландшафт”.

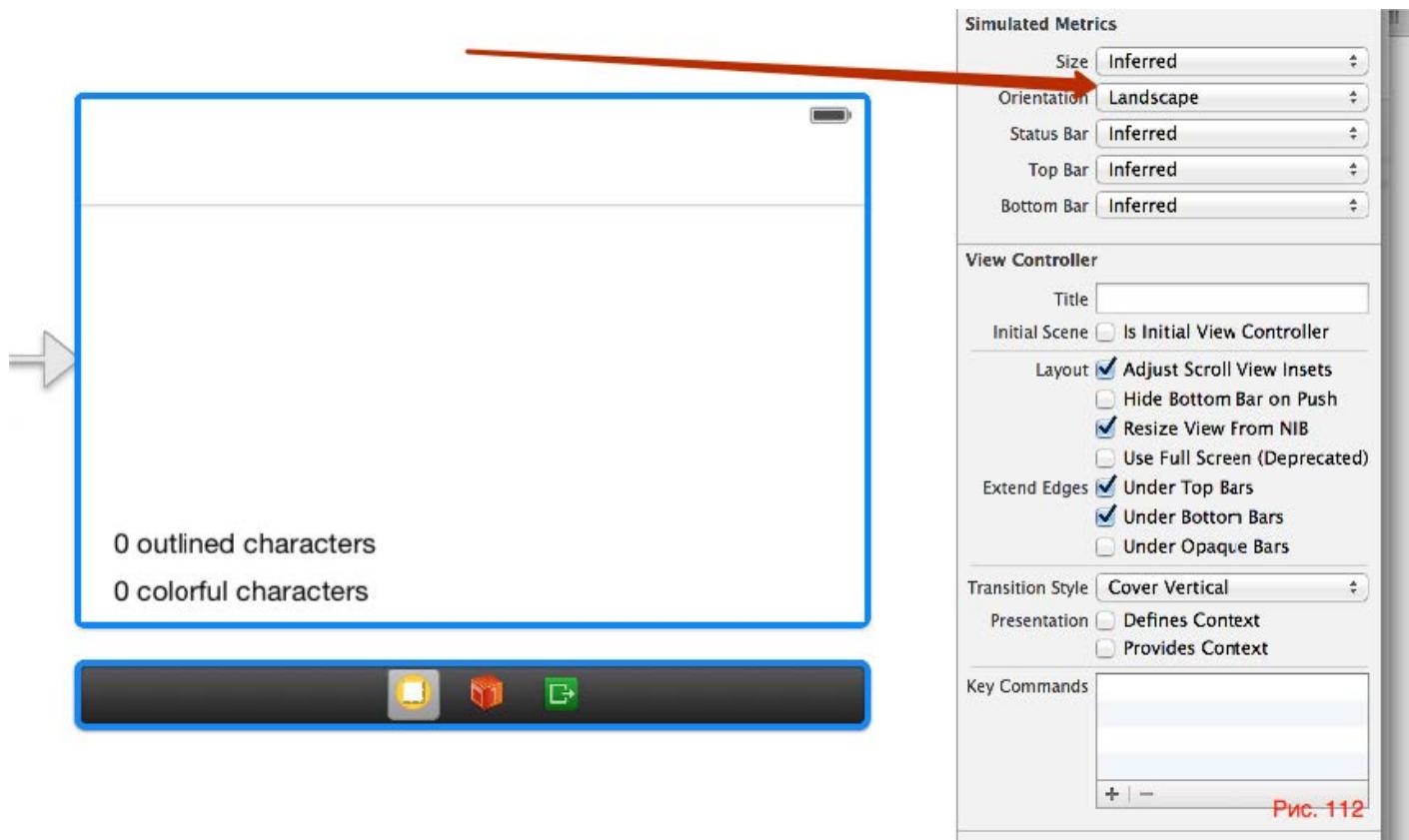
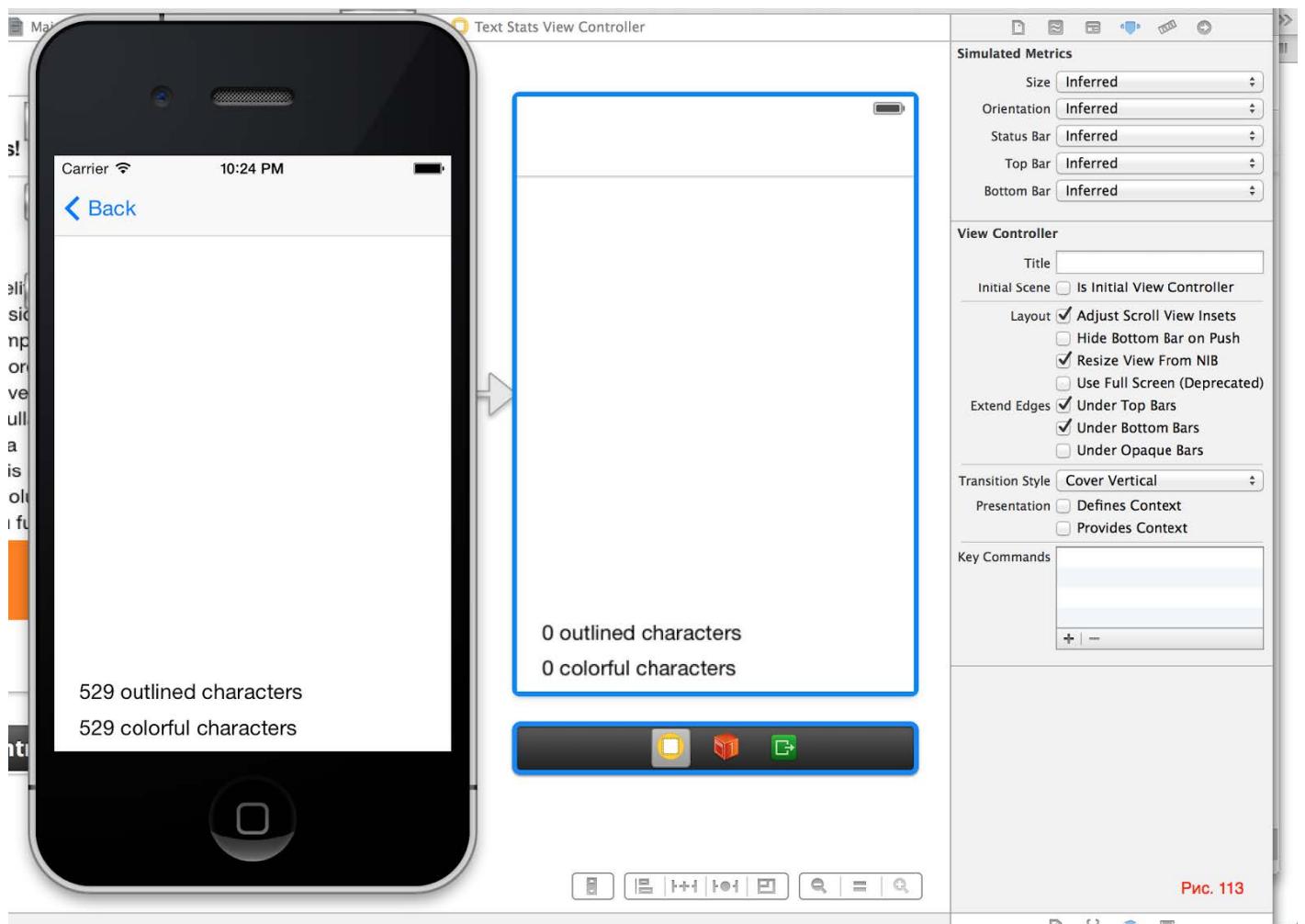


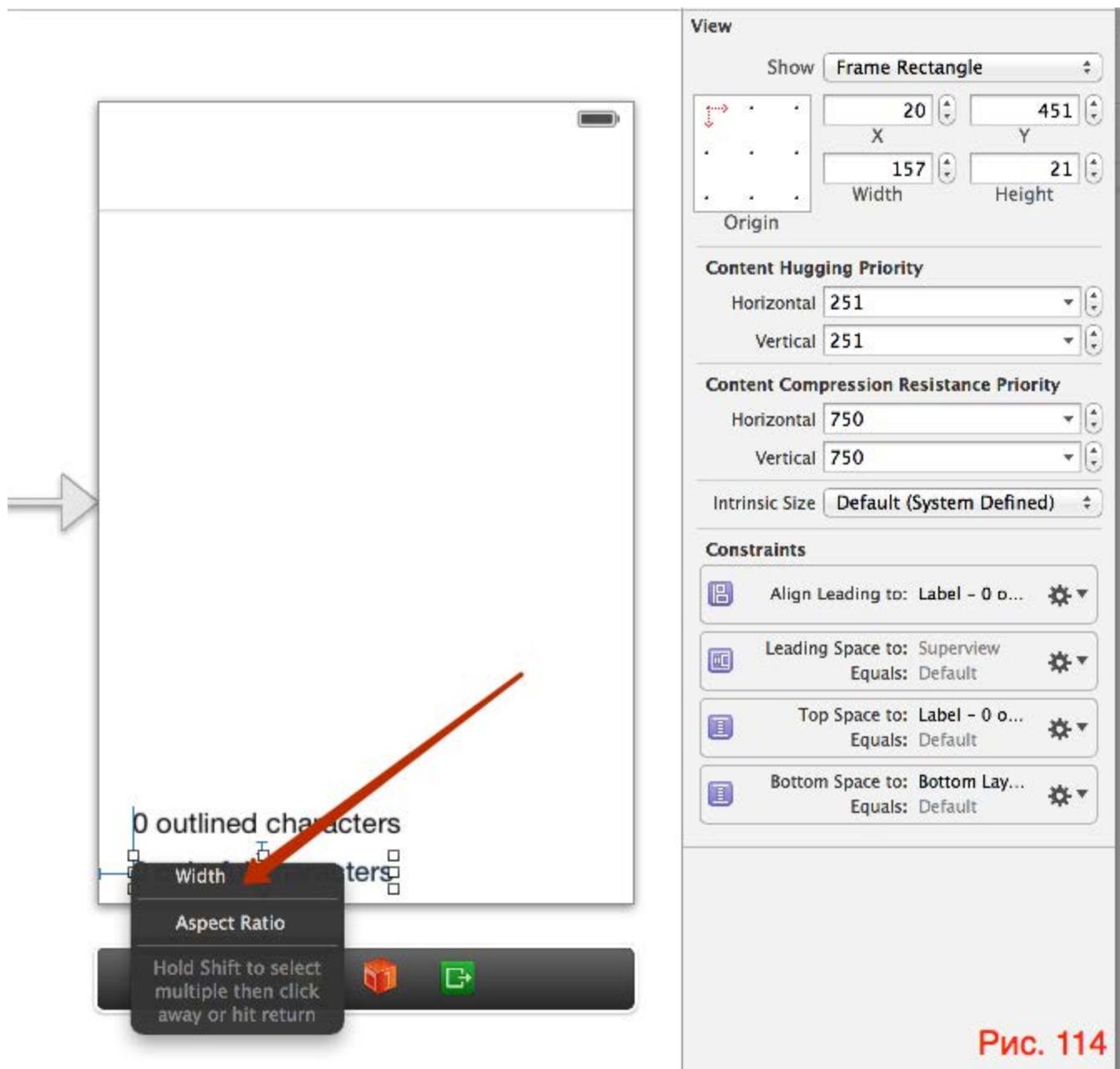
Рис. 112

Все работает прекрасно. Еще стоит посмотреть, как работает intrinsic size обоих меток. Давайте выберем большое количество символов, раскрасим их и обведем (порядка 200) и посмотрим экран Stats



Все работает прекрасно - метки не обрезаются даже, если размер текста увеличился за счет 529 символов. И это благодаря intrinsic size. Что произойдет, если зафиксировать размер одной из меток?

Давайте сделаем CTRL-перетаскивание нижней метки самой в себя и выберем width



Добавится еще одно ограничение с фиксированной шириной 157 символов, то есть текущая ширина метки

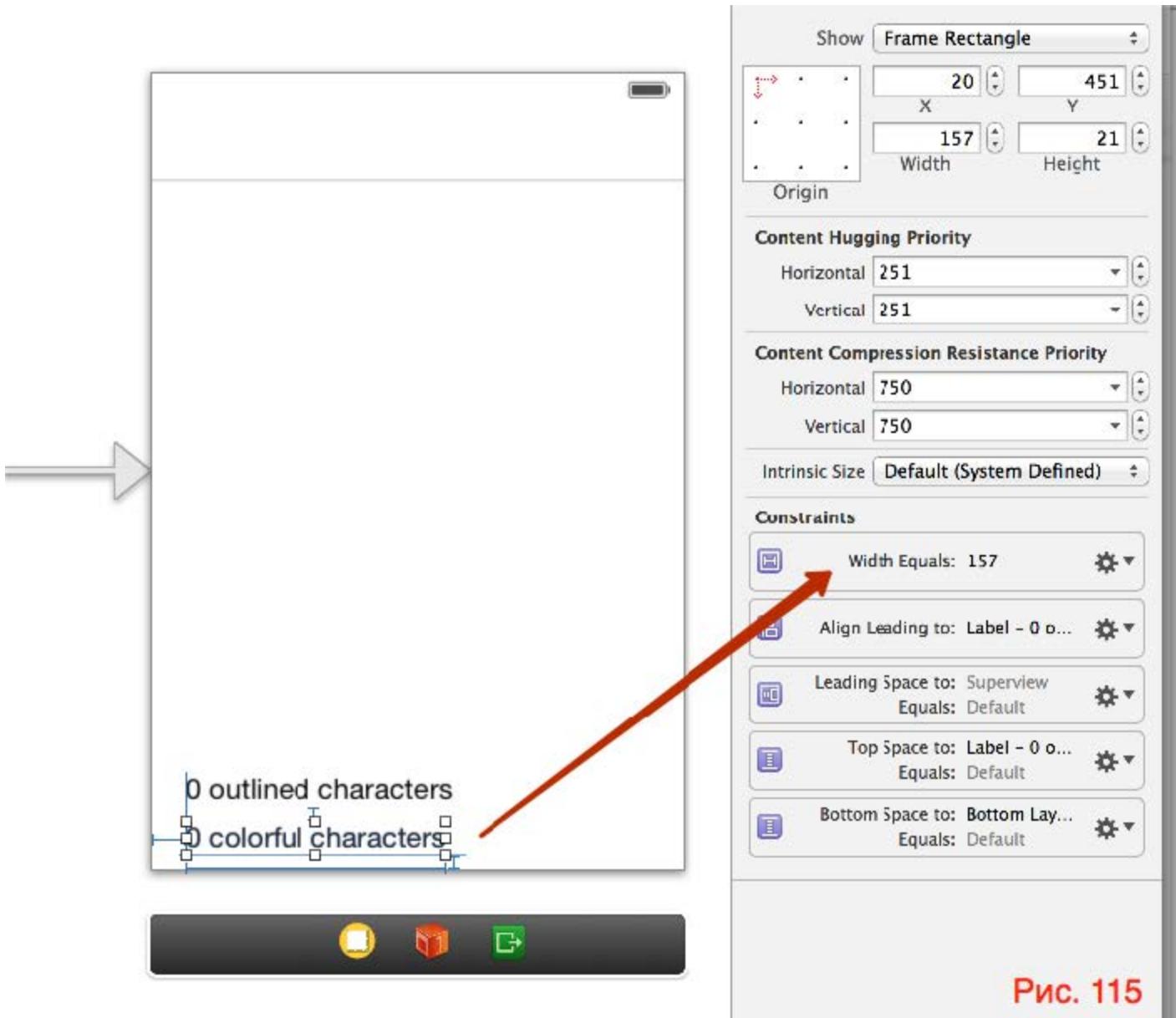


Рис. 115

И если мы повторим эксперименты с выделение, окрашиванием и обводкой большого количества символов, то получим усечение метки с числом окрашенных символов.

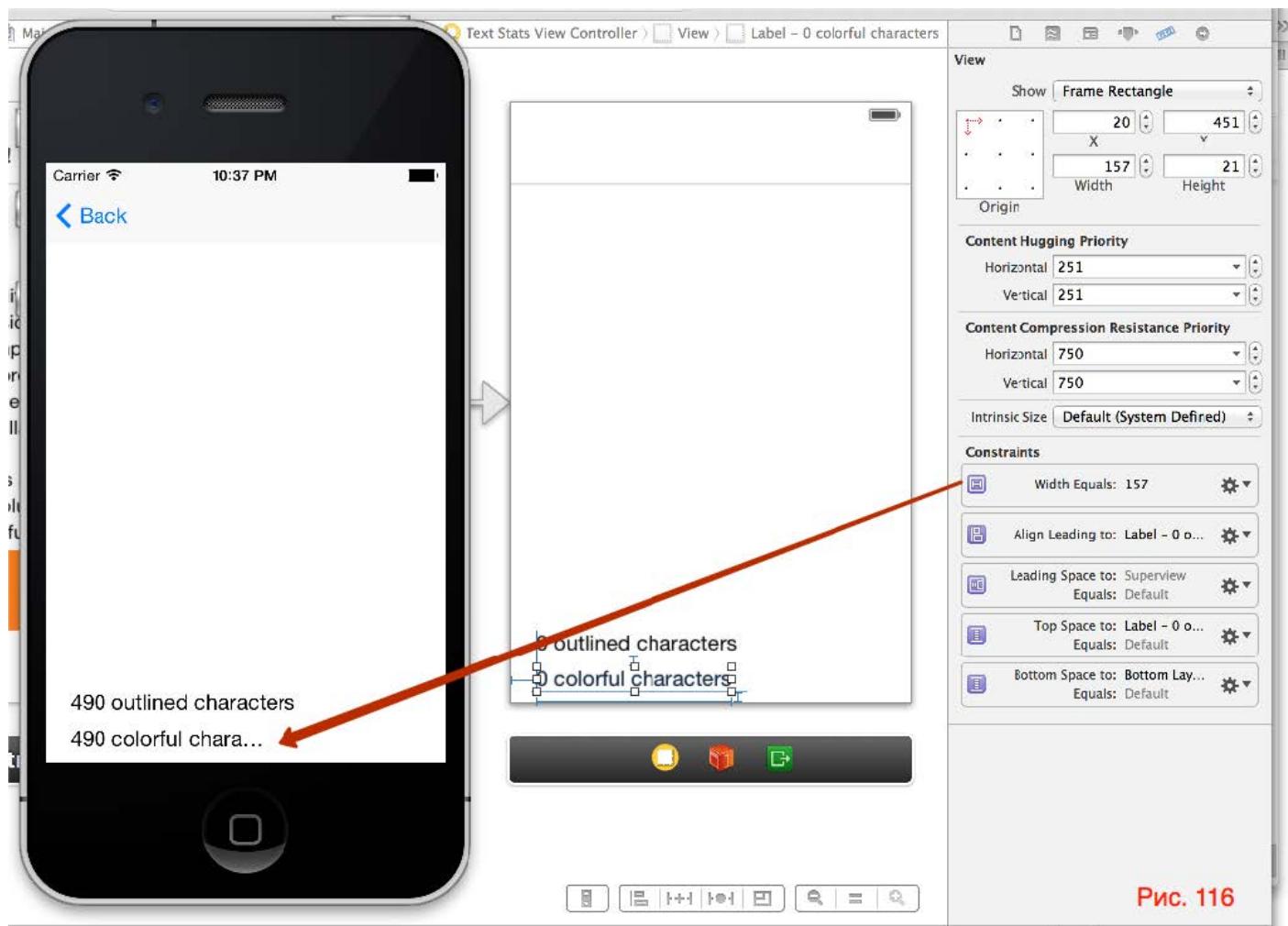


Рис. 116

При малом количестве символов, кажется, нет проблем, но если мы выберем больше 157 символов ( как на слайде), то установленная ширина не подходит и метка “усекается” и появляется с многоточием в конце.

Поэтому мы удаляем это рискованное ограничение, запускаем приложение и все восстанавливаются, даже при большом количестве выделенных букв.

И еще одну вещь стоит рассмотреть. Давайте у нашей метки “0 outlined characters” удалим ограничение “Align Leading to:”

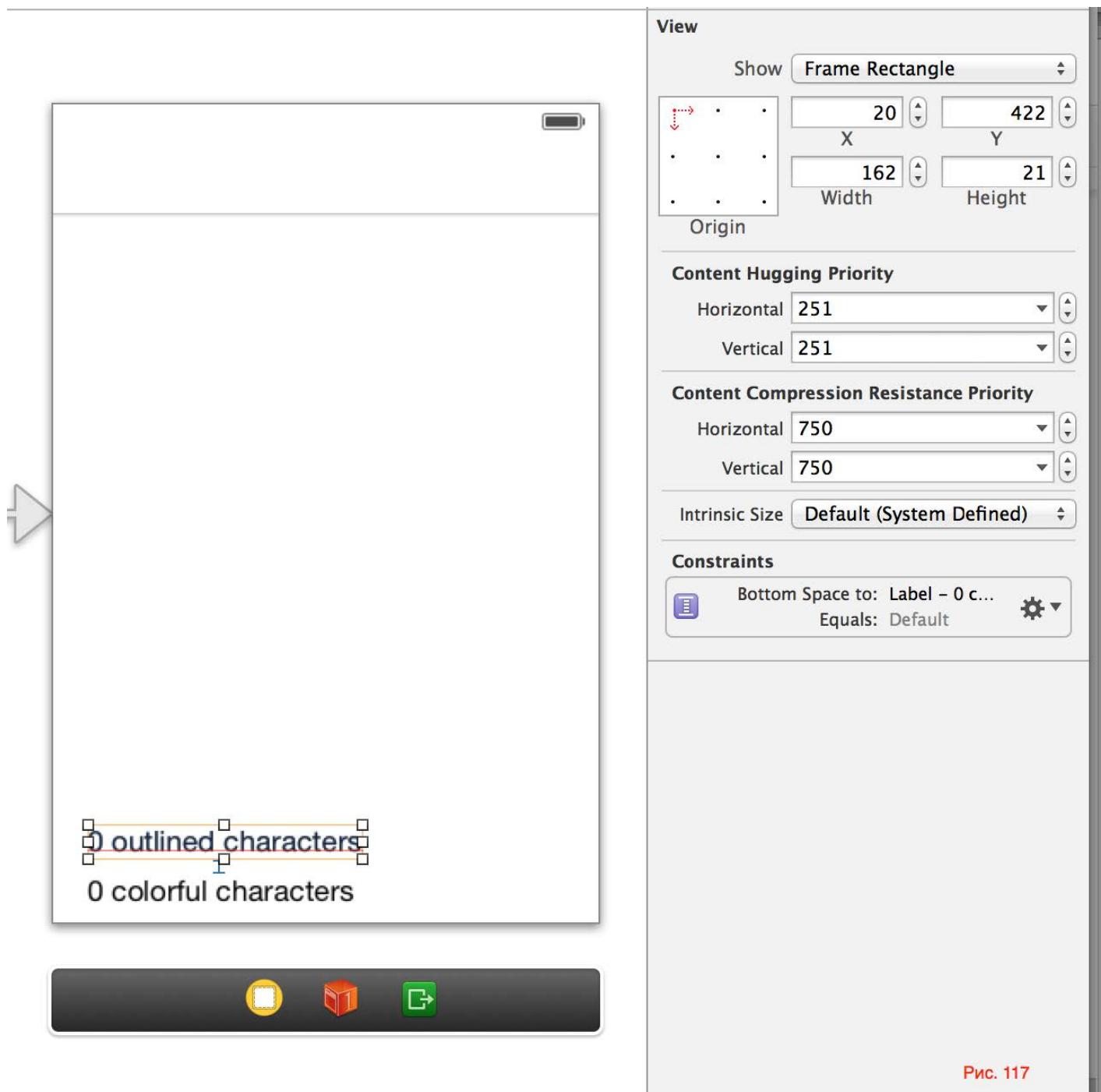


Рис. 117

Теперь эта метка больше не выравнена относительно другой метки. Заметьте, что как только мы удаляем это ограничение, метка стала обрамляться желтым цветом, потому что с ограничениями этого объекта возникли проблемы.

В любой момент могут возникнуть проблемы с ограничениями и объект будет обрамлять желтыми или даже красным цветом в случае конфликтных ограничений. Вы видите в Document Outline мы

получили красный кружочек, что говорит о проблемах с ограничениями и мы знаем что это за проблемы: мы не можем задать положение объекта по горизонтали, только по вертикали. Давайте нажмем на маленький красный кружок и получим

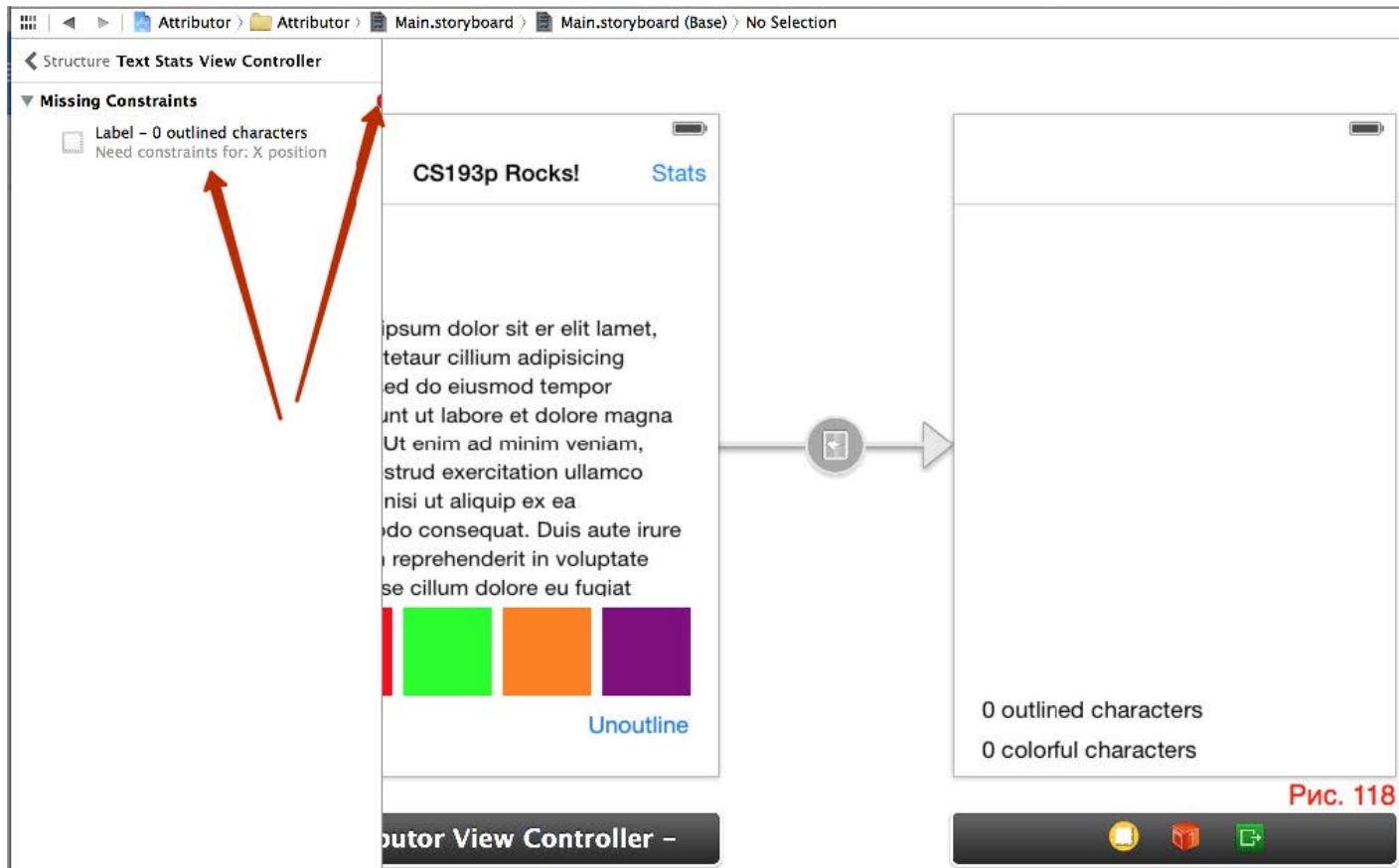


Рис. 118

Здесь говорится, что нам необходимо ограничение для X позиции, но если вы кликните на красный полукруг и нажмете кнопку “Add missing Constraints”, то система поступит очень разумно и добавит ограничение выравнивания, так как она знает, что рядом расположены две метки, следовательно их нужно выравнять слева.

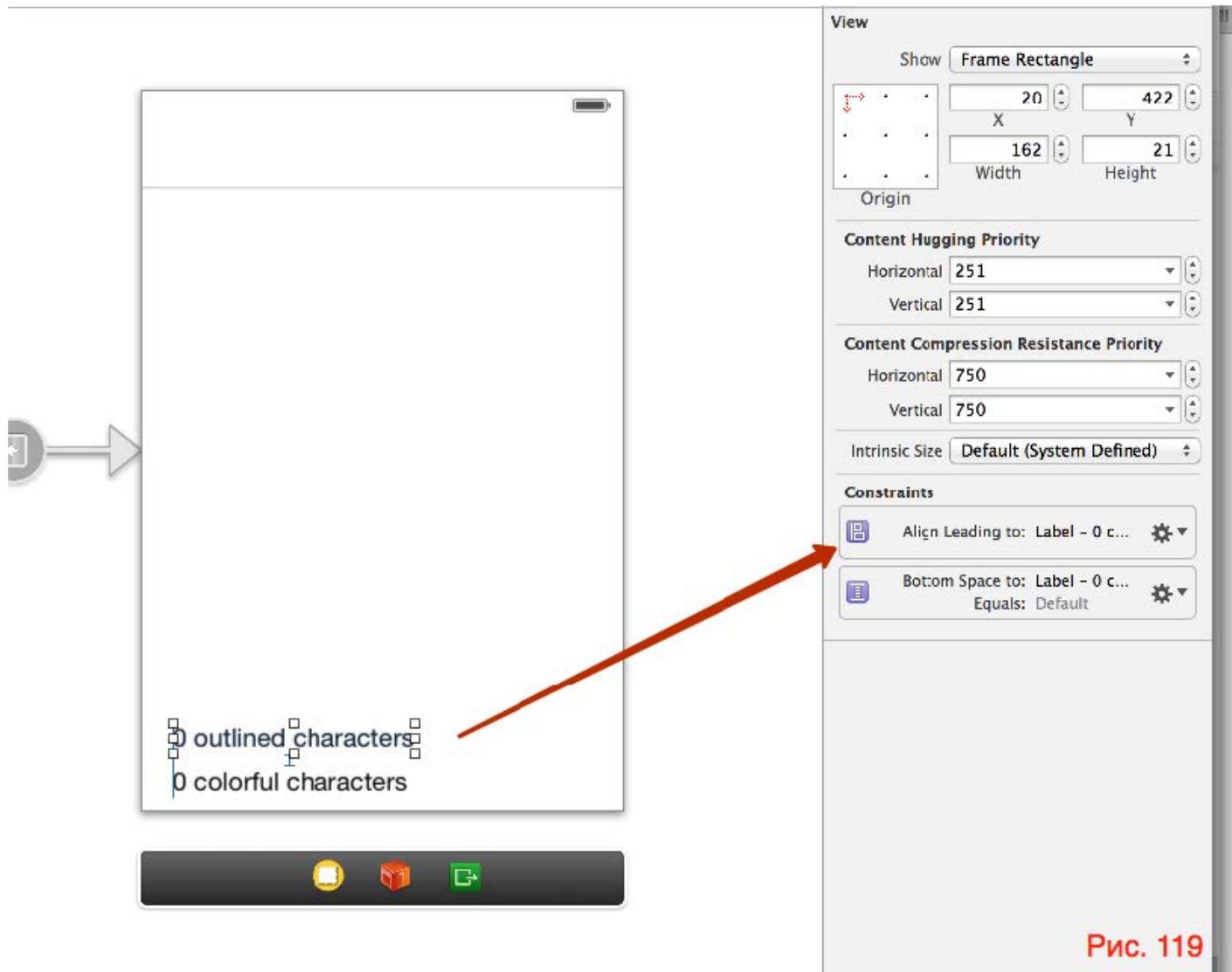
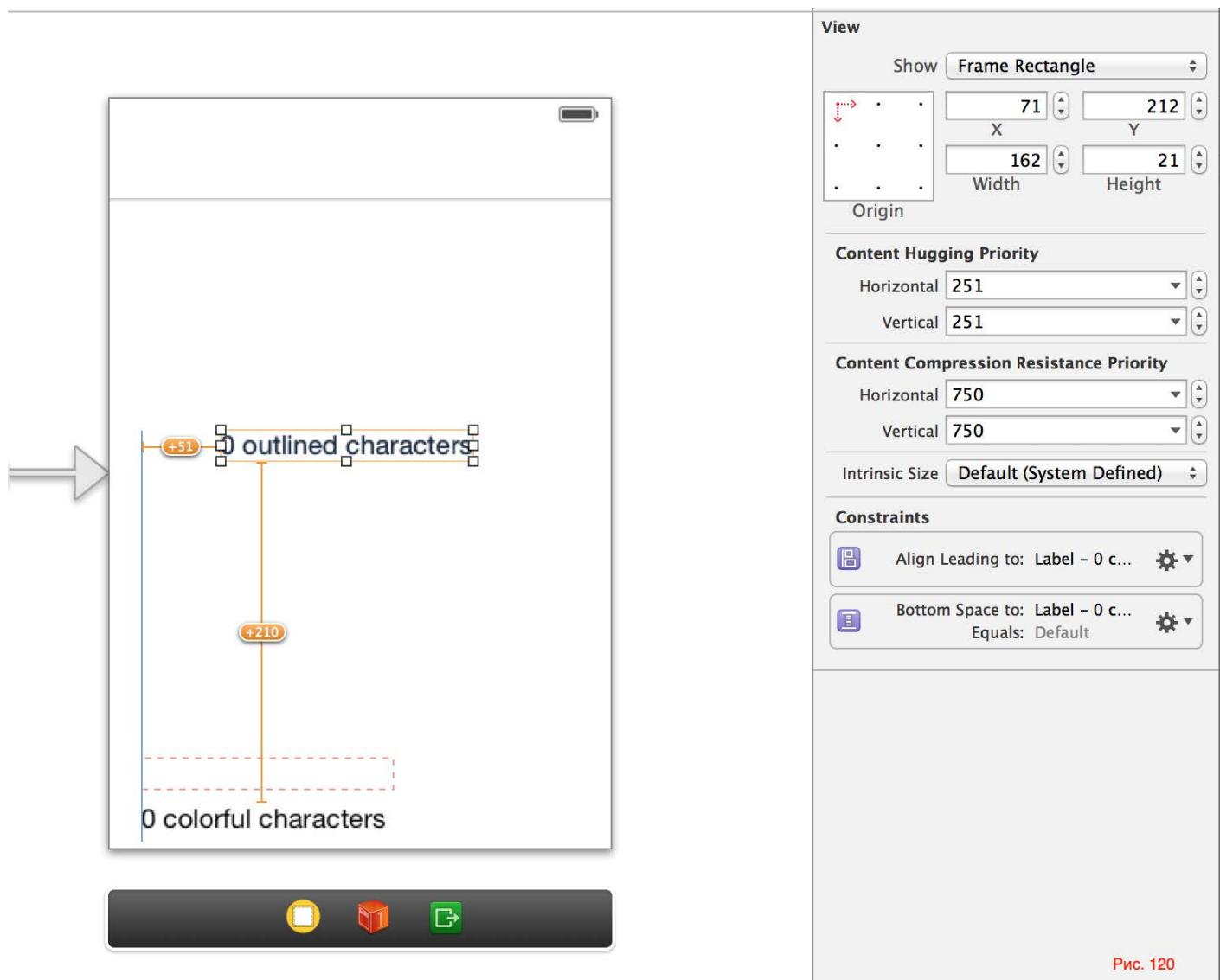


Рис. 119

И еще одна вещь. Если передвинуть метку то все станем желтым, потому что ограничения не подходят новому положению метки, обозначая пунктирной линией положение метки, соответствующее оставшимся ограничениям



Если кликнуть на желтый полукруг в Document Outline и выбрать  
“FixMisplacement” (Исправить несоответствие), то frame нашей метки будет скорректирован  
и метка вернется назад.

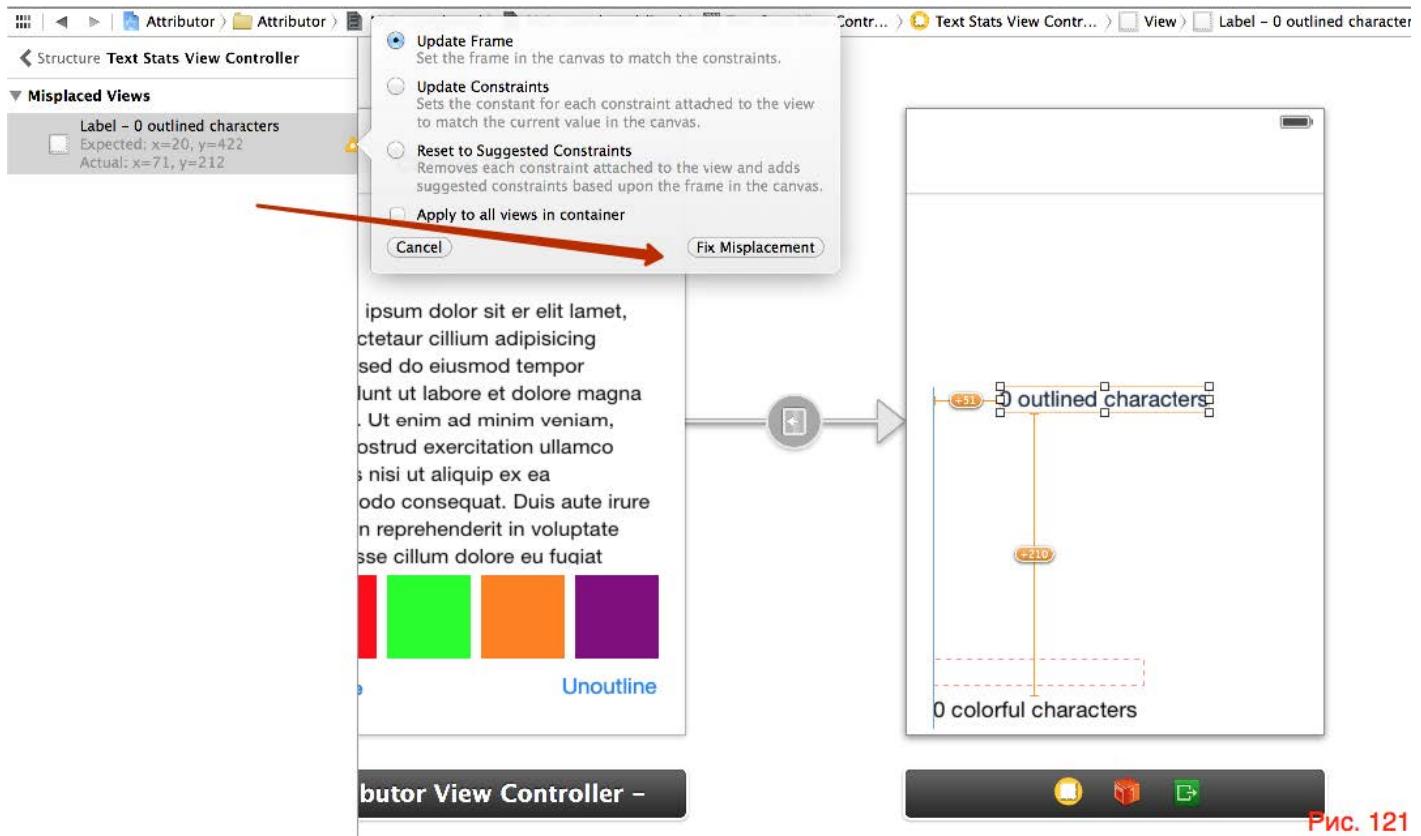


Рис. 121

На этом все.