

**In which we actually make
the ball bounce.**

LAB 07: Bouncing Ball, Part 2

Objective

Now that we have our displayLink frame timer in place, and our ball is moving, we'd like to complete the bouncing ball demo.

A major critique of part 1 of this demo is, "that's great, but the ball doesn't bounce." Which, of course, is correct. We'll make the ball bounce off the walls now.

To do this, we have to set up some imaginary walls. We'll make the edges of the screen those walls.

We need to track the *velocity* of the ball. This makes it easier for us to reverse the balls direction when it collides with the wall.

Reuse your project from LAB 6

This is continuation of the last lab. You should be able to just pick up where you left off.

Velocity 101

Wait, what's a velocity? Velocity combines the ideas of speed and direction. That's convenient for us, because we care about two things:

- How fast is the ball moving in the X-direction?
- How fast is the ball moving in the Y-direction?

That's it.

For our purposes, we'll use a **velocity property** to store the X and Y speeds. On the iPad, distance is measured in points.

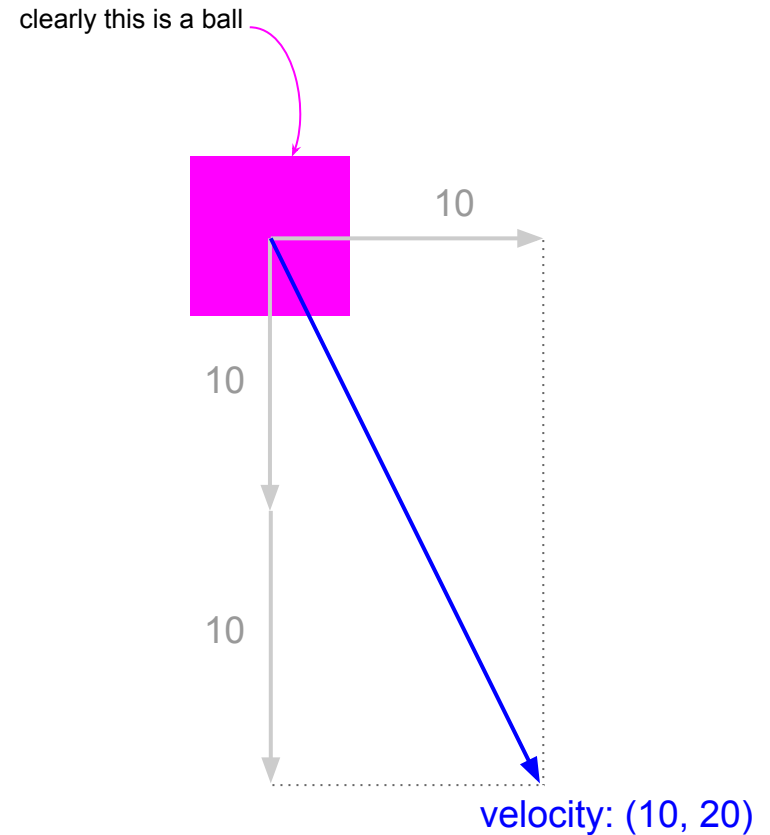
*Remember: The iPad screen has 768x1024 points. Also, Y **INCREASES** as you go **DOWN**.*

So, if I specify a velocity of (10, 20), this means that *for every frame*, the ball will move:

- 10 points in the X-direction, and
- 20 points in the Y-direction.

That “*for every frame*” business is somewhat arbitrary. But, so is bouncing a ball on the screen.

Let's just get started and see what all this does.



Add the ball's velocity property.

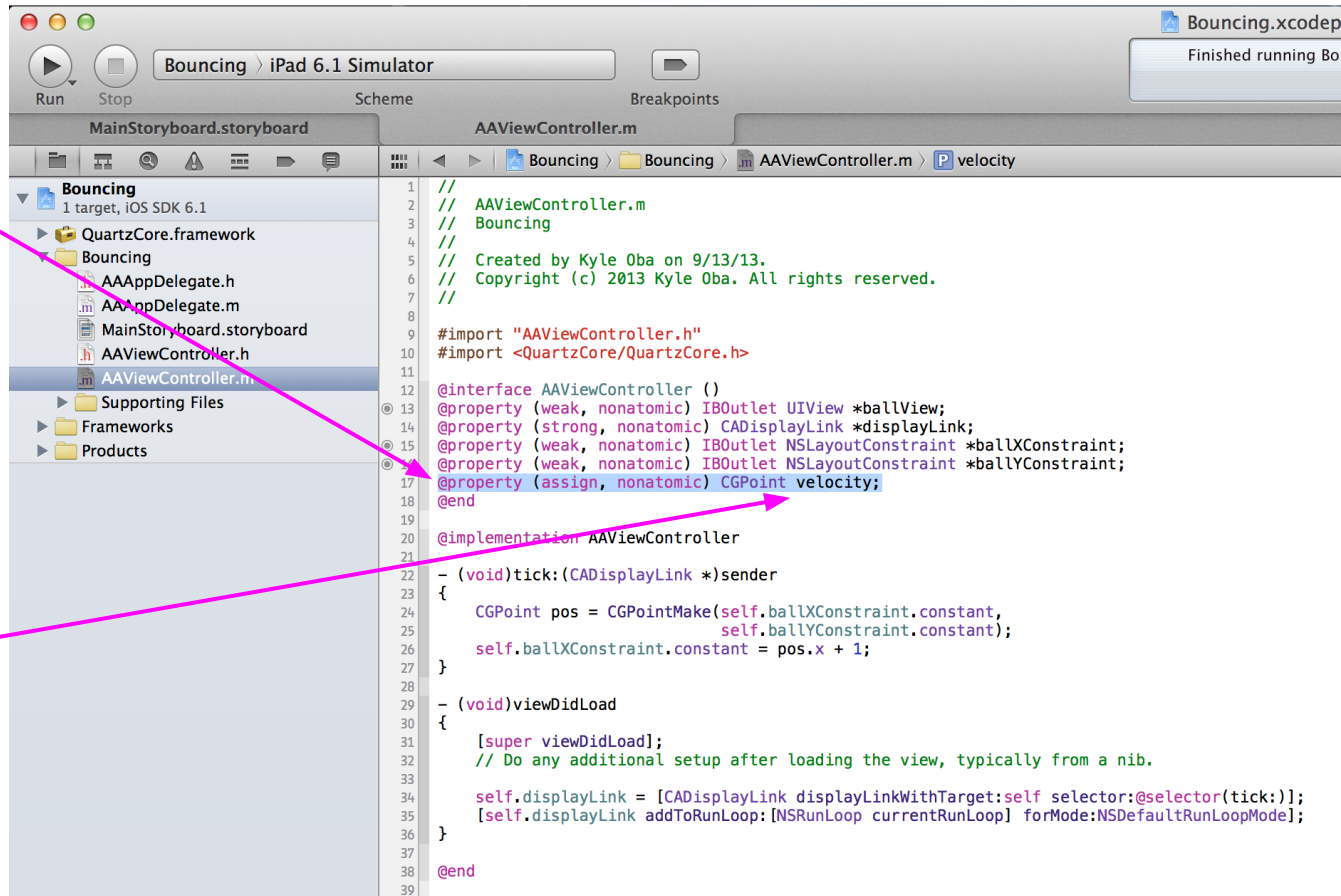
Okay. So, why all that talk about velocity? Well, we want an easy way to store how fast the ball is going, and in what direction. A velocity property is a great (easy) way to do this.

We'll add a velocity property now.

Add this property declaration to the **@interface** section of your implementation file.

*Aside: You may be wondering why I'm storing it in a **CGPoint**. I know it's not a point (it's a velocity). But, there isn't any **CGVelocity** type. And, **CGPoint**s are good for storing any x-y value pair, which is what our velocity is.*

See how I've aptly named this new property? It's called, "**velocity**."



Set the ball's velocity.

If we don't explicitly set the new velocity property to some values, it will automatically start with a velocity of (0,0).

That will result in a ball that just sits there, until your iPad runs out of power.

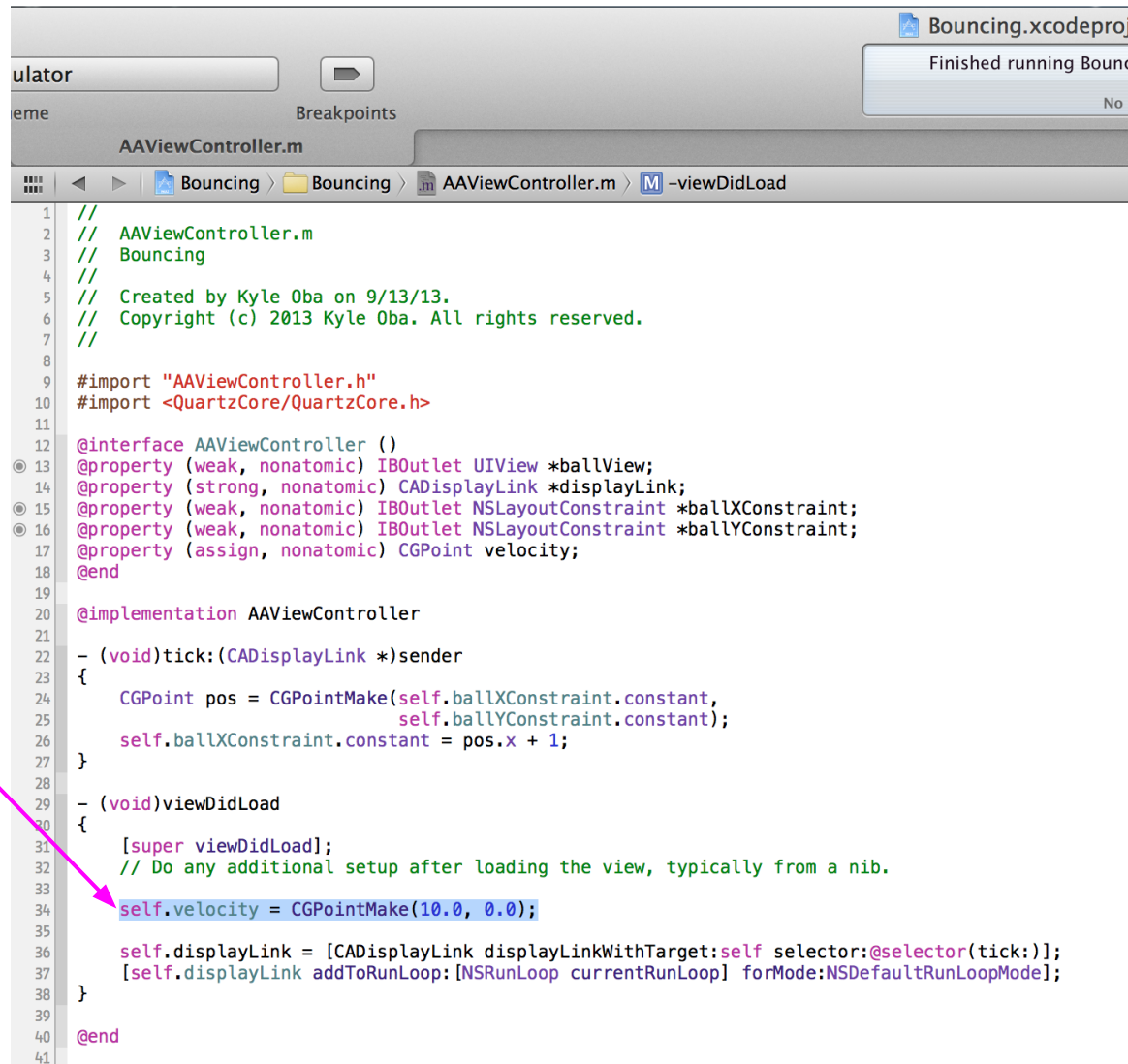
That's not what we want.

Let's initialize the **velocity** property to move to the right. Copy this code to create a new CGPoint value.

Aside: You'll notice that I'm saying 10.0, instead of 10. And, I'm saying 0.0, instead of 0. This is because points are stored with decimals (a.k.a floats, or floating point numbers).

Saying 10.0 is a good way to remind myself that I'm dealing with floats.

I'm also not moving the ball in the Y-direction yet. See how the Y-velocity value is 0.0? Let's keep it simple and only move to the right.



```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
25                               self.ballYConstraint.constant);
26     self.ballXConstraint.constant = pos.x + 1;
27 }
28
29 - (void)viewDidLoad
30 {
31     [super viewDidLoad];
32     // Do any additional setup after loading the view, typically from a nib.
33
34     self.velocity = CGPointMake(10.0, 0.0);
35
36     self.displayLink = [CADisplayLink displayLinkWithTarget:self selector:@selector(tick:)];
37     [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
38 }
39
40 @end
41
```

Now, let's use this velocity property.

Remember how we used to just add 1 to our X-position to move the ball to the right?

Well, now we're going to get all physicsy and just add the X-velocity to our position.

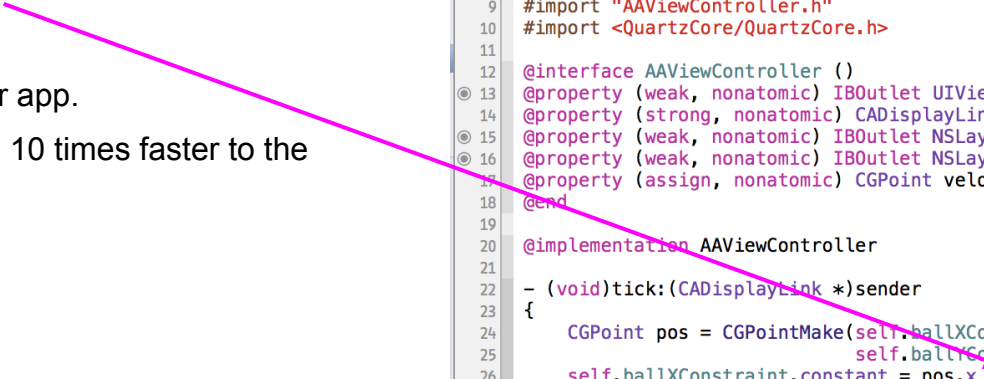
Real world physics FTW!

Copy this part here, replacing your hard-coded 1.

Now, **Run** your app.

It should move 10 times faster to the right.

Why?



```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
25                               self.ballYConstraint.constant);
26     self.ballXConstraint.constant = pos.x + self.velocity.x;
27 }
28
29 - (void)viewDidLoad
30 {
31     [super viewDidLoad];
32     // Do any additional setup after loading the view, typically from a nib.
33
34     self.velocity = CGPointMake(10.0, 0.0);
35
36     self.displayLink = [CADisplayLink displayLinkWithTarget:self selector:@selector(tick)];
37     [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
38 }
39
40 @end
41
```

Bounce it already. SRSLY.

How do we make the ball bounce off the right edge of the screen? Any takers?

Well, we need to know when the right edge of the ball collides with the right edge of the screen.

When this *collision* happens, we need to *reverse* the X-velocity.

So, step 1, how do we detect this right edge to right edge collision?

First, store the velocity, in a new variable **vel**. We need to do this, so we can make changes to it.

Next, look at the maximum X-value of the ball view's frame. Compare it to the maximum X-value of the screen's bounds.

If the ball is going too far to the right, reverse the X-velocity. We do this by taking the absolute value of the X-velocity and putting a minus sign in front of it.

Finally, replace the old velocity property with this new one (**vel**).

Run your app!

```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint vel = self.velocity;
25     if (CGRectGetMaxX(self.ballView.frame) >= CGRectGetMaxX(self.view.bounds)) {
26         vel.x = -ABS(vel.x);
27     }
28     self.velocity = vel;
29
30     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
31                               self.ballYConstraint.constant);
32     self.ballXConstraint.constant = pos.x + self.velocity.x;
33 }
34
```


It bounces! And, then disappears...

Again.

Well, at least we got one bounce out of that ball.

So, how do we bounce off the other side?

Now, we need to detect collisions between the left edge of the ball and the left edge of the screen.

We do this by adding this code here, which checks to see if

the minimum X-value of the ball's frame is *less than or equal to*

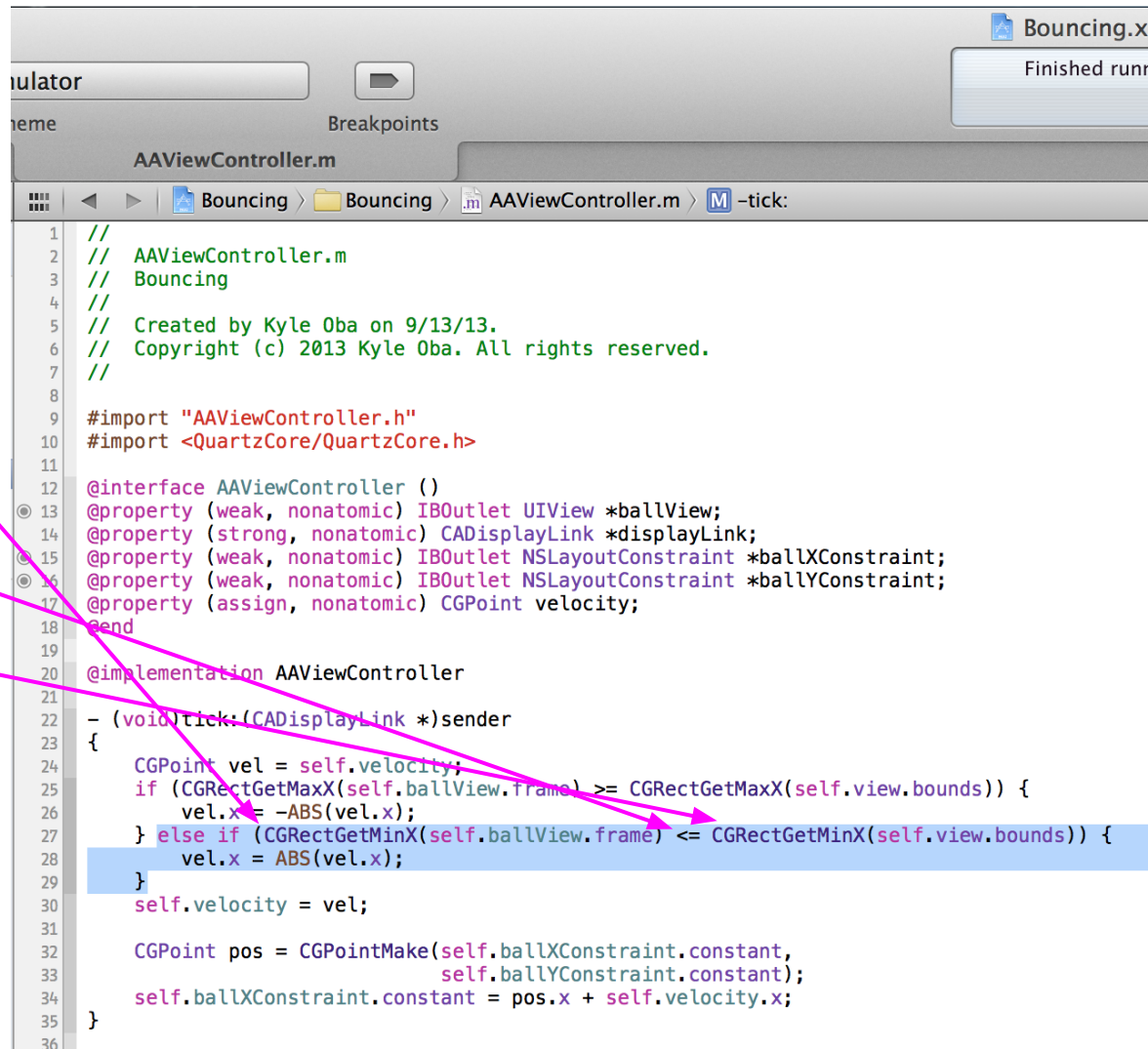
the minimum X-value of the screen's bounds.

If it *is*, then we need to reverse the X-velocity again.

We use the absolute value (**ABS**) to ensure that the ball moves to the right (has a positive X-velocity value).

Add this code, and **Run** you app.

It should bounce back and forth.
Hypnotic!



```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint vel = self.velocity;
25     if (CGRectGetMaxX(self.ballView.frame) >= CGRectGetMaxX(self.view.bounds)) {
26         vel.x = -ABS(vel.x);
27     } else if (CGRectGetMinX(self.ballView.frame) <= CGRectGetMinX(self.view.bounds)) {
28         vel.x = ABS(vel.x);
29     }
30     self.velocity = vel;
31
32     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
33                               self.ballYConstraint.constant);
34     self.ballXConstraint.constant = pos.x + self.velocity.x;
35 }
36
```

Painting the Fence

Raise your hand if you've seen the movie, *The Karate Kid*.

Remember when Daniel-san has to wax the car and paint the fence?

It was all part of Mr. Miyagi's plan to train him on fundamentals. Daniel-san had no idea that he was building muscles related to proper Karate form (or, Kata).

I know it's just a movie. But, that's what we're doing here.

You don't have to know what a CGPoint is, or a CGRectGetMinX function does *exactly*. You just have to have a general understanding that:

- we're bouncing a ball
- we're reversing velocity when it hits an edge
- programming looks a little like this
- I'm getting better for reasons that will become clearer later

If you haven't seen *The Karate Kid*, then I am old, and you are young with so much to live for.

Search YouTube for "wax on wax off" if you don't want to watch the entire movie. Warning: This movie was made in 1984.

Add a Y-velocity.

Now that we have that right-left bouncing thing down, let's add top-bottom bouncing.

Step one in this process is to add a Y-velocity.

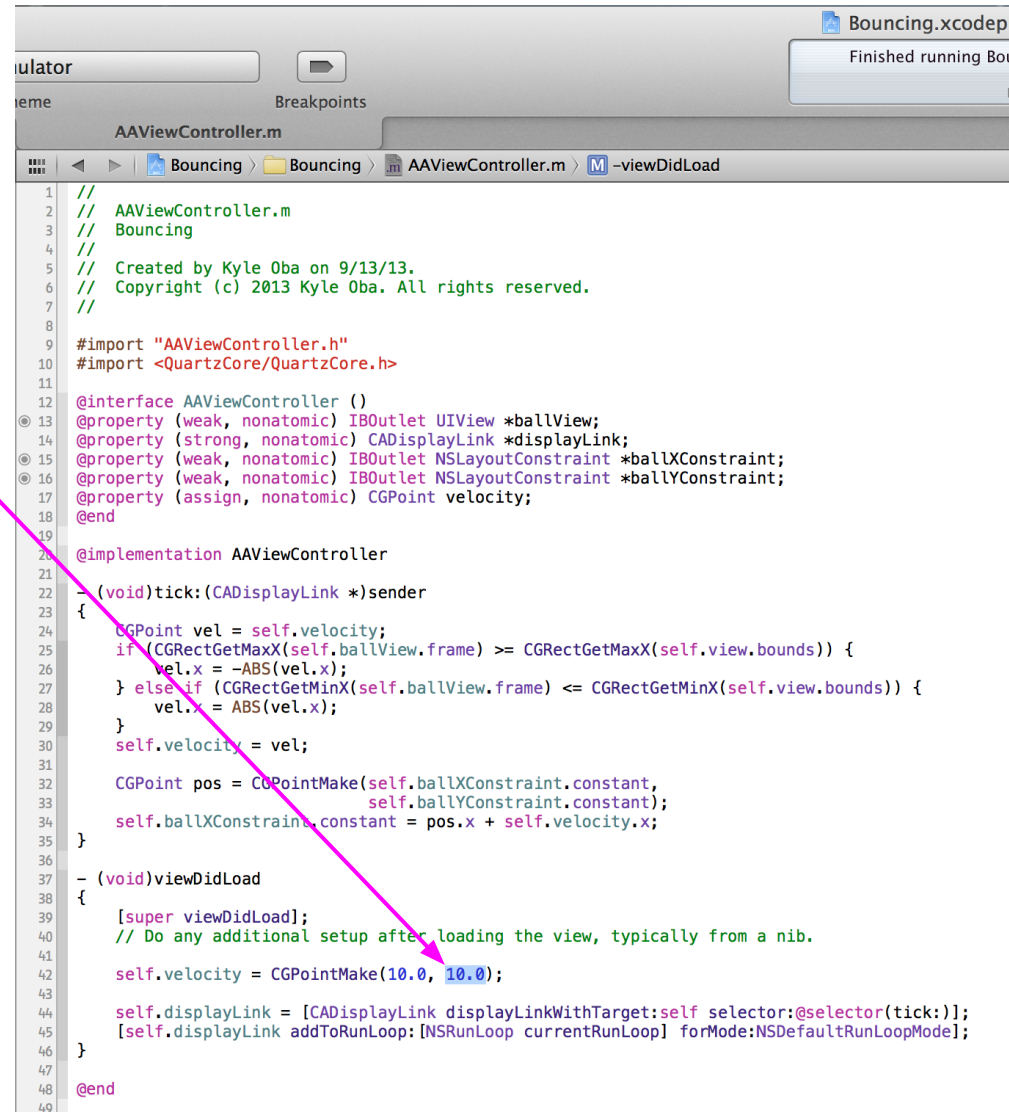
You can do this by changing it here.

Let's make it the same as the X-velocity, so the ball has a nice even motion.

You can use your considerable intuition about this little ball simulation here to guess what's going to happen now.

Run your app.

What happened? It just went left-to-right. Why?



```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint vel = self.velocity;
25     if (CGRectGetMaxX(self.ballView.frame) >= CGRectGetMaxX(self.view.bounds)) {
26         vel.x = -ABS(vel.x);
27     } else if (CGRectGetMinX(self.ballView.frame) <= CGRectGetMinX(self.view.bounds)) {
28         vel.x = ABS(vel.x);
29     }
30     self.velocity = vel;
31
32     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
33                               self.ballYConstraint.constant);
34     self.ballXConstraint.constant = pos.x + self.velocity.x;
35 }
36
37 - (void)viewDidLoad
38 {
39     [super viewDidLoad];
40     // Do any additional setup after loading the view, typically from a nib.
41
42     self.velocity = CGPointMake(10.0, 10.0);
43
44     self.displayLink = [CADisplayLink displayLinkWithTarget:self selector:@selector(tick:)];
45     [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
46 }
47
48 @end
49
```

We have to update the Y-position.

It just went left-to-right, because we haven't updated our Y-position with the new Y-velocity value yet.

Let's do that now. Copy this.

See how it's almost exactly the same as the line above it (for the X-position)?

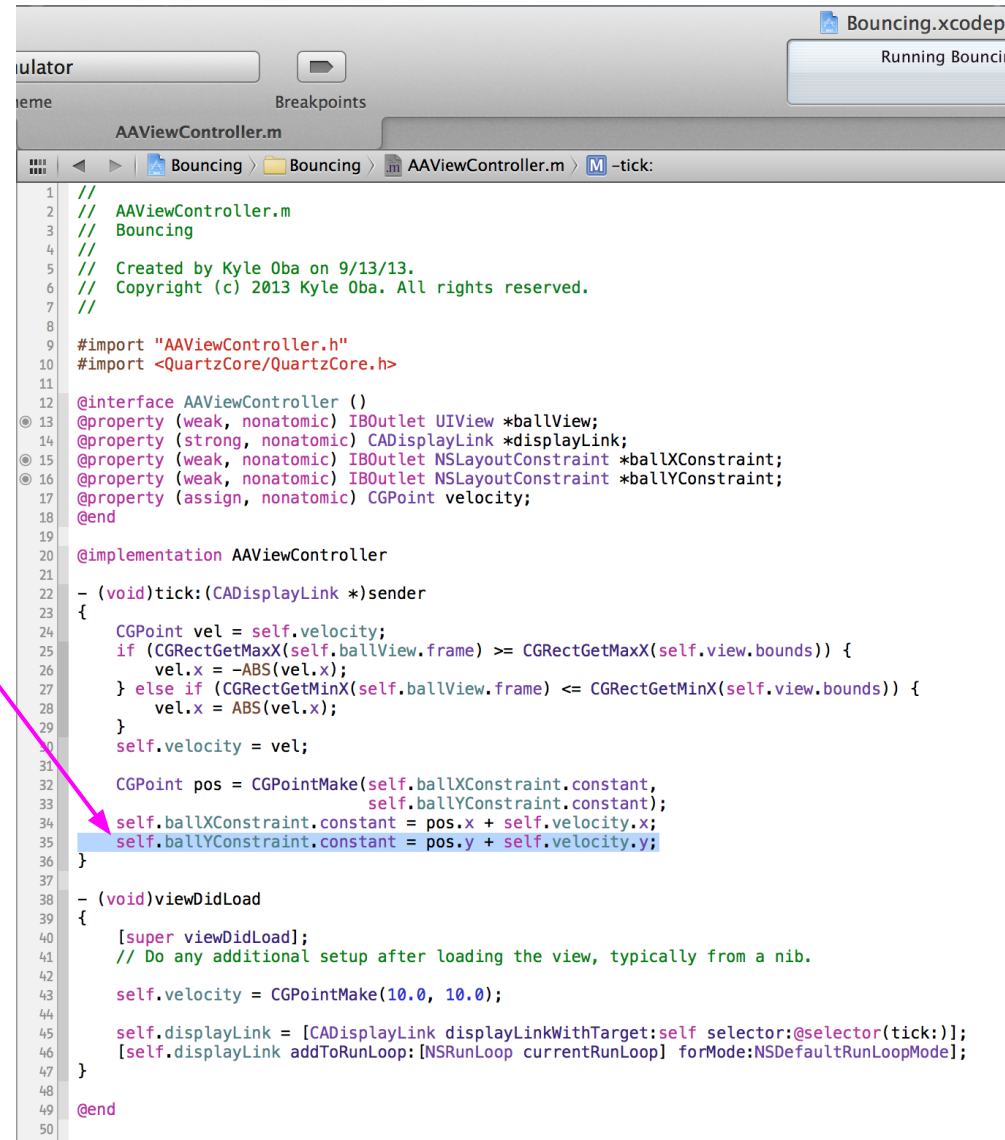
We're just changing all the Xs to Ys.

If you read this line in human English, it says something like:

"the ball's y-constraint value is equal to the former y-position plus the y-velocity"

If you run your app now, what do you think will happen?

Run it and see.



```
//
// AAViewController.m
// Bouncing
//
// Created by Kyle Oba on 9/13/13.
// Copyright (c) 2013 Kyle Oba. All rights reserved.

#import "AAViewController.h"
#import <QuartzCore/QuartzCore.h>

@interface AAViewController ()
@property (weak, nonatomic) IBOutlet UIView *ballView;
@property (strong, nonatomic) CADisplayLink *displayLink;
@property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
@property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballyConstraint;
@property (assign, nonatomic) CGPoint velocity;
@end

@implementation AAViewController

- (void)tick:(CADisplayLink *)sender
{
    CGPoint vel = self.velocity;
    if (CGRectGetMaxX(self.ballView.frame) >= CGRectGetMaxX(self.view.bounds)) {
        vel.x = -ABS(vel.x);
    } else if (CGRectGetMinX(self.ballView.frame) <= CGRectGetMinX(self.view.bounds)) {
        vel.x = ABS(vel.x);
    }
    self.velocity = vel;

    CGPoint pos = CGPointMake(self.ballXConstraint.constant,
                             self.ballyConstraint.constant);
    self.ballXConstraint.constant = pos.x + self.velocity.x;
    self.ballyConstraint.constant = pos.y + self.velocity.y;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    self.velocity = CGPointMake(10.0, 10.0);

    self.displayLink = [CADisplayLink displayLinkWithTarget:self selector:@selector(tick:)];
    [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
}

@end
```

Enough already.

I'm just going to belch out all the code to bounce your ball from top-to-bottom.

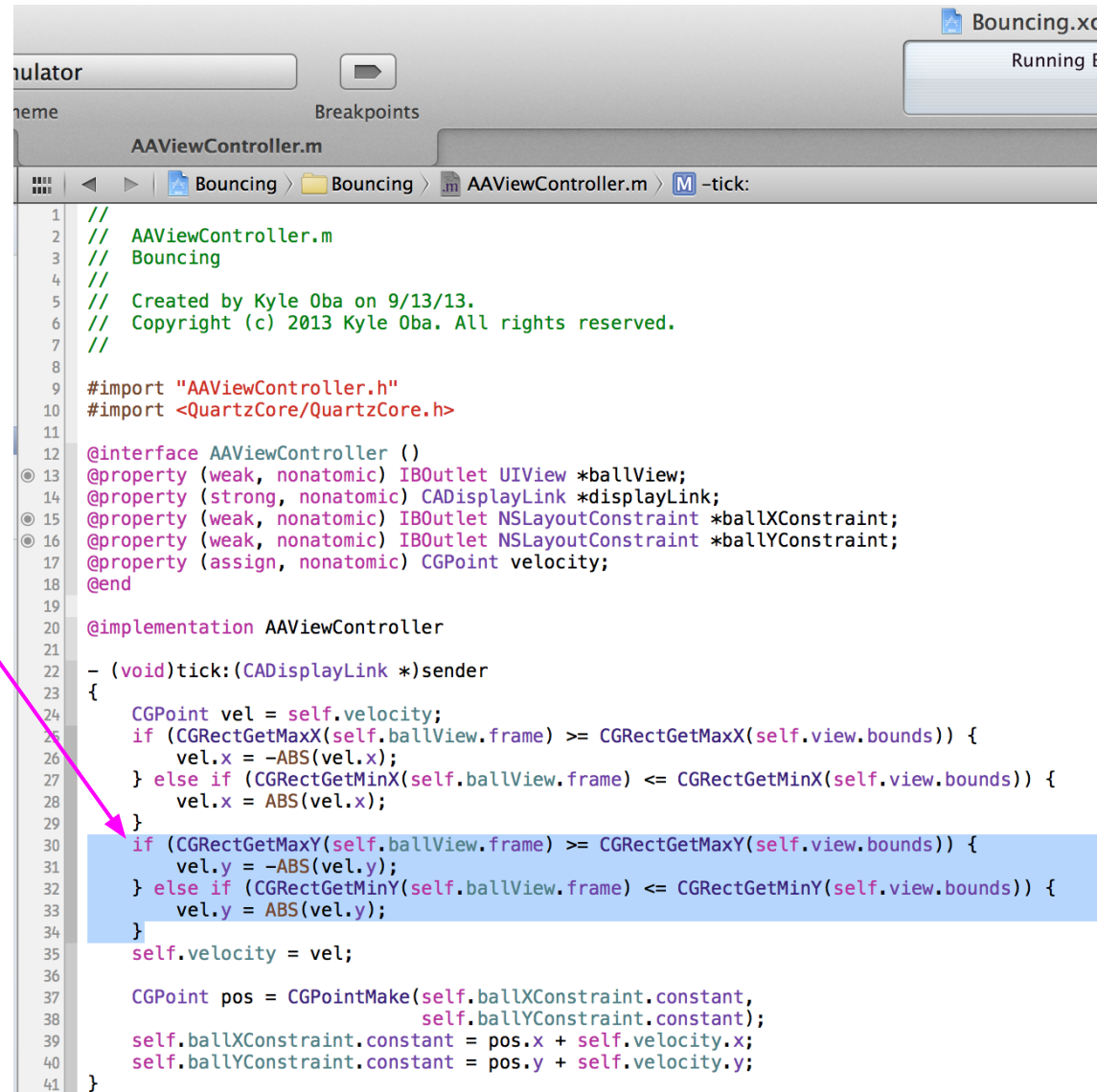
You already saw how to do it step by step for the right-to-left case.

Just add this code here.

You'll see that it's very similar to the code we used to reverse the X-velocity.

But, now, we're detecting collisions with the bottom and top of the screen, and reversing the Y-velocity.

Run your app.



```
1 //
2 // AAViewController.m
3 // Bouncing
4 //
5 // Created by Kyle Oba on 9/13/13.
6 // Copyright (c) 2013 Kyle Oba. All rights reserved.
7 //
8
9 #import "AAViewController.h"
10 #import <QuartzCore/QuartzCore.h>
11
12 @interface AAViewController ()
13 @property (weak, nonatomic) IBOutlet UIView *ballView;
14 @property (strong, nonatomic) CADisplayLink *displayLink;
15 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballXConstraint;
16 @property (weak, nonatomic) IBOutlet NSLayoutConstraint *ballYConstraint;
17 @property (assign, nonatomic) CGPoint velocity;
18 @end
19
20 @implementation AAViewController
21
22 - (void)tick:(CADisplayLink *)sender
23 {
24     CGPoint vel = self.velocity;
25     if (CGRectGetMaxX(self.ballView.frame) >= CGRectGetMaxX(self.view.bounds)) {
26         vel.x = -ABS(vel.x);
27     } else if (CGRectGetMinX(self.ballView.frame) <= CGRectGetMinX(self.view.bounds)) {
28         vel.x = ABS(vel.x);
29     }
30     if (CGRectGetMaxY(self.ballView.frame) >= CGRectGetMaxY(self.view.bounds)) {
31         vel.y = -ABS(vel.y);
32     } else if (CGRectGetMinY(self.ballView.frame) <= CGRectGetMinY(self.view.bounds)) {
33         vel.y = ABS(vel.y);
34     }
35     self.velocity = vel;
36
37     CGPoint pos = CGPointMake(self.ballXConstraint.constant,
38                               self.ballYConstraint.constant);
39     self.ballXConstraint.constant = pos.x + self.velocity.x;
40     self.ballYConstraint.constant = pos.y + self.velocity.y;
41 }
```

It bounces!



Flickr image by Ganesha Balunsat: http://www.flickr.com/photos/ganesha_isis/4458591011/
Creative Commons, Attribution 2.0 licensed: <http://creativecommons.org/licenses/by/2.0/deed.en>