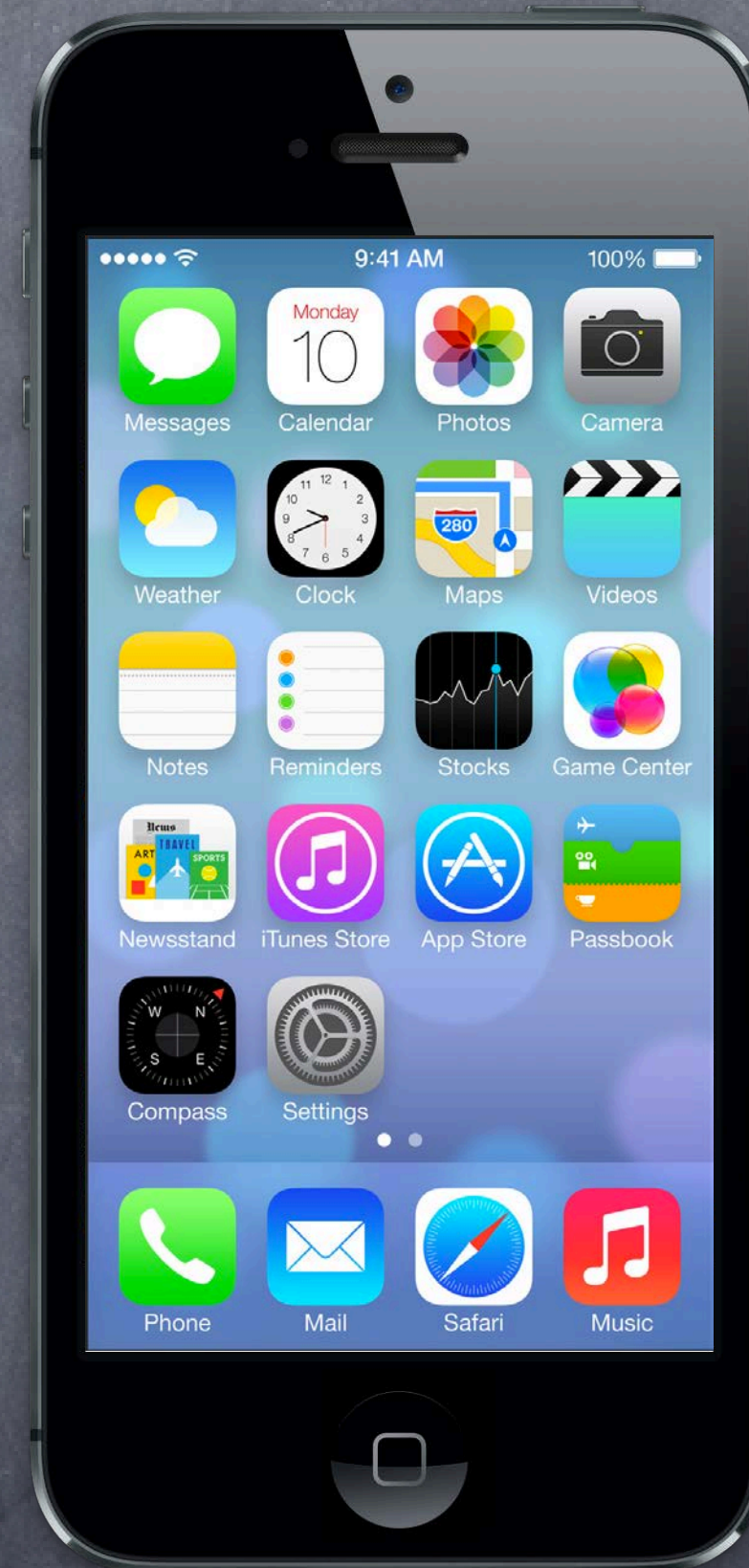


# Stanford CS193p

Developing Applications for iOS  
Fall 2013-14



# Today

- Introduction to Objective-C (con't)

Continue showing Card Game Model with Deck, PlayingCard, PlayingCardDeck

- Xcode 5 Demonstration

Start building the simple Card Game

# Objective-C

Card.h

Card.m

```
#import <Foundation/Foundation.h>

@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(NSArray *)otherCards;
```

@end

```
#import "Card.h"

@interface Card()

@end

@implementation Card

- (int)match:(NSArray *)otherCards
{
    int score = 0;

    for (Card *card in otherCards) {
        if ([card.contents isEqualToString:self.contents]) {
            score = 1;
        }
    }

    return score;
}
```

@end

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
```

```
@interface Deck : NSObject
```

```
@end
```

Let's look at another class.  
This one represents a deck of cards.

```
#import "Deck.h"
```

```
@interface Deck()
```

```
@end
```

```
@implementation Deck
```

```
@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;

- (Card *)drawRandomCard;

@end
```

Note that this method has 2 arguments (and returns nothing). It's called "addCard:atTop:".

And this one takes no arguments and returns a Card (i.e. a pointer to an instance of a Card in the heap).

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck
```

@end

# Objective-C

Deck.h

```
#import <Foundation/Foundation.h>
#import "Card.h"
```

We must `#import` the header file for any class we use in this file (e.g. Card).

```
@interface Deck : NSObject
```

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop;
```

```
- (Card *)drawRandomCard;
```

```
@end
```

Deck.m

```
#import "Deck.h"
```

```
@interface Deck()
```

```
@end
```

```
@implementation Deck
```

```
@end
```

# Objective-C

Deck.h

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;

- (Card *)drawRandomCard;

@end
```

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{

}

- (Card *)drawRandomCard { }

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;

- (Card *)drawRandomCard;

@end
```

Arguments to methods  
(like the atTop: argument)  
are never "optional."

```
#import "Deck.h"

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{

}

- (Card *)drawRandomCard { }

@end
```



# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"
```

```
@interface Deck : NSObject
```

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop;
```

```
- (void)addCard:(Card *)card;
```

```
- (Card *)drawRandomCard;
```

```
@end
```

Arguments to methods  
(like the atTop: argument)  
are never "optional."

However, if we want an addCard:  
method without atTop:, we can  
define it separately.

```
#import "Deck.h"
```

```
@implementation Deck
```

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop  
{
```

```
}
```

```
- (Card *)drawRandomCard { }
```

```
@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"
```

```
@interface Deck : NSObject
```

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop;
```

```
- (void)addCard:(Card *)card;
```

```
- (Card *)drawRandomCard;
```

```
@end
```

Arguments to methods  
(like the atTop: argument)  
are never "optional."

However, if we want an addCard:  
method without atTop:, we can  
define it separately.

And then simply implement it in  
terms of the the other method.

```
#import "Deck.h"
```

```
@implementation Deck
```

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop  
{
```

```
}
```

```
- (void)addCard:(Card *)card  
{  
    [self addCard:card atTop:NO];  
}
```

```
- (Card *)drawRandomCard { }
```

```
@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

A deck of cards obviously needs some storage to keep the cards in. We need an **@property** for that. But we don't want it to be public (since it's part of our private, internal implementation).

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{

}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

A deck of cards obviously needs some storage to keep the cards in. We need an **@property** for that. But we don't want it to be public (since it's part of our private, internal implementation).

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{

}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

So we put the **@property** declaration we need here in our **@implementation**.

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

self.cards is an **NSMutableArray** ...

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck
```

Now that we have a property to store our cards in, let's take a look at a sample implementation of the addCard:atTop: method.

```
- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

...and these are **NSMutableArray** methods. (insertObject:atIndex: and addObject:).

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

But there's a problem here.  
When does the object pointed to by the pointer returned by `self.cards` ever get created?

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Declaring a `@property` makes space in the instance for the pointer itself, but not does not allocate space in the heap for the object the pointer points to.

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

The place to put this needed heap allocation is in the getter for the cards **@property**.

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

The place to put this needed heap allocation is in the getter for the cards **@property**.

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

All properties start out with a value of 0 (called **nil** for pointers to objects). So all we need to do is allocate and initialize the object if the pointer to it is **nil**. This is called “lazy instantiation”. Now you can start to see the usefulness of a **@property**.



# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

We'll talk about allocating and initializing objects more later, but here's a simple way to do it.

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Now the cards property will always at least be an empty mutable array, so this code will always do what we want.

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Let's collapse the code we've written so far to make some space.

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{

}

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    drawRandomCard simply grabs a card from a
    random spot in our self.cards array.

    return randomCard;
}

@end
```

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    unsigned index = arc4random() % [self.cards count];
    randomCard = self.cards[index];
    [self.cards removeObjectAtIndex:index];

    return randomCard;
}

@end
```

arc4random() returns a random integer.

This is the C modulo operator.

These square brackets actually are the equivalent of sending the message `objectAtIndexedSubscript:` to the array.

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    if ([self.cards count]) {
        unsigned index = arc4random() % [self.cards count];
        randomCard = self.cards[index];
        [self.cards removeObjectAtIndex:index];
    }

    return randomCard;
}

@end
```

Calling `objectAtIndexedSubscript:` with an argument of zero on an empty array will **crash** (array index out of bounds)!

So let's protect against that case.

# Objective-C

Deck.h

Deck.m

```
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    if ([self.cards count]) {
        unsigned index = arc4random() % [self.cards count];
        randomCard = self.cards[index];
        [self.cards removeObjectAtIndex:index];
    }

    return randomCard;
}

@end
```

# Objective-C

PlayingCard.h

PlayingCard.m

Let's see what it's like to make a subclass of one of our own classes. In this example, a subclass of Card specific to a playing card (e.g. A♠).



# Objective-C

PlayingCard.h

```
#import "Card.h"  
  
@interface PlayingCard : Card
```

Of course we must `#import` our superclass.

And `#import` our own header file in our implementation file.

```
@end
```

PlayingCard.m

```
#import "PlayingCard.h"  
  
@implementation PlayingCard
```

```
@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard
```

```
@end
```

A PlayingCard has some properties that a vanilla Card doesn't have. Namely, the PlayingCard's suit and rank.

# Objective-C

## PlayingCard.h

## PlayingCard.m

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

```
#import "PlayingCard.h"

@implementation PlayingCard
```

We'll represent the suit as an `NSString` that simply contains a single character corresponding to the suit (i.e. one of these characters: ♠♣♥♦). If this property is `nil`, it'll mean "suit not set".

We'll represent the rank as an integer from 0 (rank not set) to 13 (a King).

`NSUInteger` is a typedef for an unsigned integer.

We could just use the C type `unsigned int` here. It's mostly a style choice. Many people like to use `NSUInteger` and `NSInteger` in public API and `unsigned int` and `int` inside implementation. But be careful, `int` is 32 bits, `NSInteger` might be 64 bits. If you have an `NSInteger` that is really big (i.e. > 32 bits worth) it could get truncated if you assign it to an `int`. Probably safer to use one or the other everywhere.

```
@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Users of our PlayingCard class might well simply access `suit` and `rank` properties directly. But we can also support our superclass's `contents` property by overriding the getter to return a suitable (no pun intended) `NSString`.

Even though we are overriding the implementation of the `contents` method, we are not re-declaring the `contents` property in our header file. We'll just inherit that declaration from our superclass.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    return [NSString stringWithFormat:@"%d%@", self.rank, self.suit];
}
```

@end

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Users of our PlayingCard class might well simply access suit and rank properties directly. But we can also support our superclass's contents property by overriding the getter to return a suitable (no pun intended) `NSString`.

Even though we are overriding the implementation of the contents method, we are not re-declaring the contents property in our header file. We'll just inherit that declaration from our superclass.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    return [NSString stringWithFormat:@"%d%", self.rank, self.suit];
}
```

The method `stringWithFormat:` is an `NSString` method that's sort of like using the C function `printf` to create the string.

Note we are creating an `NSString` here in a different way than `alloc/init`. We'll see more about "class methods" like `stringWithFormat:` a little later.

@end

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    return [NSString stringWithFormat:@"%d%", self.rank, self.suit];
}
```

Calling the getters of our two properties (rank and suit) on ourself.

But this is a pretty bad representation of the card (e.g., it would say 11♣ instead of J♣ and 1♥ instead of A♥).

@end

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}
```

We'll create an `NSArray` of `NSStrings`, each of which corresponds to a given rank.

Again, 0 will be "rank not set" (so we'll use ?).  
11, 12 and 13 will be J Q K and 1 will be A.

Then we'll create our "J♠" string by appending (with the `stringByAppendingString:` method) the suit onto the end of the string we get by looking in the array.

@end

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}
```

Notice the `@[]` notation to create an array.

Here's the array-accessing `[]` notation again (like we used with `self.cards[index]` earlier).

Also note the `@""` notation to create a (constant) `NSString`.

All of these notations are converted into normal message-sends by the compiler.  
For example, `@[...]` is `[[NSArray alloc] initWithObjects:...]`.  
`rankStrings[self.rank]` is `[rankStrings objectAtIndexedSubscript:self.rank]`.

```
@end
```



# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}
```

This is nice because a “not yet set” rank shows up as ?.

But what about a “not yet set” suit?  
Let’s override the getter for suit to make a suit of `nil` return ?.

Yet another nice use for properties versus direct instance variables.

```
- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Let's take this a little further and override the setter for suit to have it check to be sure no one tries to set a suit to something invalid.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥", @"♦", @"♠", @"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Notice that we can embed the array creation as the target of this message send. We're simply sending `containsObject:` to the array created by the `@[]`.

`containsObject:` is an `NSArray` method.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

But there's a problem here now.  
A compiler warning will be generated  
if we do this.  
Why?  
Because if you implement BOTH the  
setter and the getter for a property,  
then you have to create the instance  
variable for the property yourself.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

But there's a problem here now.  
A compiler warning will be generated  
if we do this.  
Why?

Because if you implement BOTH the  
setter and the getter for a property,  
then you have to create the instance  
variable for the property yourself.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Luckily, the compiler can help with this  
using the `@synthesize` directive.

If you implement only the setter OR  
the getter (or neither), the compiler  
adds this `@synthesize` for you.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Name of the property we're creating an instance variable for.

Name of the instance variable to associate with the property.

We almost always pick an instance variable name that is underbar followed by the name of the property.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

You should only ever access the instance variable directly ...

... in the property's setter ...

... in its getter ...

... or in an initializer (more on this later).

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

All of the methods we've seen so far are "instance methods".

They are methods sent to instances of a class. But it is also possible to create methods that are sent to the class itself. Usually these are either creation methods (like `alloc` or `stringWithFormat:`) or utility methods.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

- (void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣"] containsObject:suit) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```



# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Class methods start with +  
Instance methods start with -

Here's an example of a class utility method which returns an `NSArray` of the `NSString`s which are valid suits (e.g. ♠, ♣, ♥, and ♦).

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return
}

- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥", @"♦", @"♠", @"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Since a class method is not sent to an instance, we cannot reference our properties in here (since properties represent per-instance storage).

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Here's an example of a class utility method which returns an `NSArray` of the `NSString`s which are valid suits (e.g. ♠, ♣, ♥, and ♦).

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥", @"♦", @"♠", @"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if (![
        _suit = suit;
    containsObject:suit]) {

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

We actually already have the array of valid suits, so let's just move that up into our new class method.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Now let's invoke our new class method here.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥", @"♦", @"♠", @"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

See how the name of the class appears in the place you'd normally see a pointer to an instance of an object?

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Now let's invoke our new class method here.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥", @"♦", @"♠", @"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

See how the name of the class appears in the place you'd normally see a pointer to an instance of an object?

It'd probably be instructive to go back and look at the invocation of the `NSString` class method `stringWithFormat:` a few slides ago.

Also, make sure you understand that `stringByAppendingString:` above is not a class method, it is an instance method.

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;

@end
```

The `validSuits` class method might be useful to users of our `PlayingCard` class, so let's make it public.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @"?", "A", "2", "3", ..., "10", "J", "Q", "K";
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @"♥", @"♦", @"♠", @"♣";
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;

@end
```

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }
```

@end

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;

@end
```

Let's move our other array  
(the strings of the ranks)  
into a class method too.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings =
        return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;

@end
```

We'll leave this one private because the public API for the rank is purely numeric.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}

@end
```

And now let's call that class method.

Note that we are not required to declare this earlier in the file than we use it.



# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

But here's another class method that might be good to make public.

So we'll add it to the header file.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

And, finally, let's use maxRank inside the setter for the rank @property to make sure the rank is never set to an improper value.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

- (void)setRank:(NSUInteger)rank
{
    if (rank <= [PlayingCard maxRank]) {
        _rank = rank;
    }
}

@end
```

# Objective-C

## PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

That's it for our PlayingCard. It's a good example of array notation, `@synthesize`, class methods, and using getters and setters for validation.

## PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

- (void)setRank:(NSUInteger)rank
{
    if (rank <= [PlayingCard maxRank]) {
        _rank = rank;
    }
}

@end
```

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

Let's look at one last class.  
This one is a subclass of Deck and  
represents a full 52-card deck of  
PlayingCards.

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck
```

```
@end
```

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"  
  
@interface PlayingCardDeck : Deck  
  
@end
```

It appears to have no public API, but it is going to override a method that Deck inherits from NSObject called `init`.

`init` will contain everything necessary to initialize a PlayingCardDeck.

```
#import "PlayingCardDeck.h"  
  
@implementation PlayingCardDeck
```

```
@end
```

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{

}

@end
```

Initialization in Objective-C happens immediately after allocation. We always nest a call to `init` around a call to `alloc`.  
e.g. `Deck *myDeck = [[PlayingCardDeck alloc] init]`  
or `NSMutableArray *cards = [[NSMutableArray alloc] init]`

Classes can have more complicated initializers than just plain “init” (e.g. `initWithCapacity:` or some such). We’ll talk more about that next week as well.

Only call an `init` method immediately after calling `alloc` to make space in the heap for that new object. And never call `alloc` without immediately calling some `init` method on the newly allocated object.

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{

}

@end
```

Notice this weird “return type” of `instancetype`. It basically tells the compiler that this method returns an object which will be the same type as the object that this message was sent to.

We will pretty much only use it for `init` methods. Don't worry about it too much for now. But always use this return type for your `init` methods.

# Objective-C

## PlayingCardDeck.h

## PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {

    }

    return self;
}

@end
```

This sequence of code might also seem weird. Especially an assignment to **self**! This is the **ONLY** time you would ever assign something to **self**. The idea here is to return **nil** if you cannot initialize this object. But we have to check to see if our **super**class can initialize itself. The assignment to **self** is a bit of protection against our trying to continue to initialize ourselves if our **super**class couldn't initialize. Just always do this and don't worry about it too much.



# Objective-C

## PlayingCardDeck.h

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

## PlayingCardDeck.m

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {

    }

    return self;
}

@end
```

Sending a message to **super** is how we send a message to ourselves, but use our superclass's implementation instead of our own. Standard object-oriented stuff.

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

The implementation of `init` is quite simple. We'll just iterate through all the suits and then through all the ranks in that suit ...

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {
        for (NSString *suit in [PlayingCard validSuits]) {
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {

            }
        }
    }

    return self;
}

@end
```

# Objective-C

PlayingCardDeck.h

PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

Then we will allocate and initialize a PlayingCard and then set its suit and rank.

```
#import "PlayingCardDeck.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {
        for (NSString *suit in [PlayingCard validSuits]) {
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {
                PlayingCard *card = [[PlayingCard alloc] init];
                card.rank = rank;
                card.suit = suit;
            }
        }

        return self;
    }
}

@end
```

We never implemented an `init` method in `PlayingCard`, so it just inherits the one from `NSObject`. Even so, we must always call an `init` method after `alloc`.

# Objective-C

## PlayingCardDeck.h

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

Then we will allocate and initialize a `PlayingCard` and then set its `suit` and `rank`.

## PlayingCardDeck.m

```
#import "PlayingCardDeck.h"
#import "PlayingCard.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {
        for (NSString *suit in [PlayingCard validSuits]) {
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {
                PlayingCard *card = [[PlayingCard alloc] init];
                card.rank = rank;
                card.suit = suit;
            }
        }

        return self;
    }
}

@end
```

We will need to `#import` `PlayingCard`'s header file since we are referencing it now in our implementation.

We never implemented an `init` method in `PlayingCard`, so it just inherits the one from `NSObject`. Even so, we must always call an `init` method after `alloc`.

# Objective-C

## PlayingCardDeck.h

## PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

```
#import "PlayingCardDeck.h"
#import "PlayingCard.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {
        for (NSString *suit in [PlayingCard validSuits]) {
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {
                PlayingCard *card = [[PlayingCard alloc] init];
                card.rank = rank;
                card.suit = suit;
                [self addCard:card];
            }
        }

        return self;
    }
}

@end
```

Finally we just add each `PlayingCard` we create to our `self` (we are a Deck, remember).

# Objective-C

## PlayingCardDeck.h

## PlayingCardDeck.m

```
#import "Deck.h"

@interface PlayingCardDeck : Deck

@end
```

```
#import "PlayingCardDeck.h"
#import "PlayingCard.h"

@implementation PlayingCardDeck

- (instancetype)init
{
    self = [super init];

    if (self) {
        for (NSString *suit in [PlayingCard validSuits]) {
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {
                PlayingCard *card = [[PlayingCard alloc] init];
                card.rank = rank;
                card.suit = suit;
                [self addCard:card];
            }
        }
    }

    return self;
}

@end
```

And that's it!  
We inherit everything else we need to  
be a Deck of cards  
(like the ability to drawRandomCard)  
from our superclass.

# Demo

- Let's start building a Card Game out of these classes  
Today we'll just have a single card that we can flip over to reveal the Ace of clubs.

The following slides are a walkthrough of the demonstration done in class.  
You will need this walkthrough to do your first homework assignment.

Yellow Bubbles  
mean “do something.”

Green Bubbles  
are just for  
“information.”

Red Bubbles  
mean “important!”

Green Bubbles  
with small text is for  
“minor notes.”





## Xcode 5 Splash Screen

# Welcome to Xcode

Version 5.0



### Create a new Xcode project

Start building a new iPhone, iPad or Mac application.



### Check out an existing project

Start working on something from an SCM Repository.

No Recent Projects

 Open Other...

Launch Xcode 5 and click here to create a new project.

As you create projects, they will appear here.

### Choose a template for your new project

**iOS**

- Application
- Framework & Library
- Other

**OS X**

- Application
- Framework & Library
- Application Plug-in
- System Plug-in
- Other

Master-Detail Application

OpenGL Game

Page-Based Application

**1 Single View Application**

Tabbed Application

Utility Application

Empty Application

SpriteKit Game

**1 Single View Application**

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Cancel Previous **Next**

Xcode 5 can be used to develop both iOS and Mac OSX applications.

These buttons are used to select a template which Xcode 5 uses to generate some code to get you started.

Click on the "Single View Application" template. It creates a simple MVC application.

Then click Next.

### Choose options for your new project:

Our first application is going to be a Card Matching Game

These fields describe your project. We'll be filling them in during the next few slides.

Product Name

Organization Name

Company Identifier

Bundle Identifier

Class Prefix

Devices

The name of our project is going to be "Matchismo" so type that in here.

No Selection

Cancel

Previous

Next

## Choose options for your new project:

This will appear in the copyright at the top of all code files you create.

|                    |  |
|--------------------|--|
| Product Name       | <input type="text" value="Matchismo"/>                 |
| Organization Name  | <input type="text" value="CS193p"/>                    |
| Company Identifier | <input type="text"/>                                   |
| Bundle Identifier  | <input type="text" value="com.yourcompany.Matchismo"/> |
| Class Prefix       | <input type="text"/>                                   |
| Devices            | <input type="text" value="Universal"/>                 |

Here you can enter CS193p or Stanford or Bob's Awesome App House.

No Selection

### Choose options for your new project:

Product Name

Organization Name

Company Identifier

Bundle Identifier

Class Prefix

Devices

Cancel

Previous

Next

This field is used to uniquely identify your application.

Enter edu.stanford.cs193p.yoursunetid

Using an entity's reverse DNS lookup string is a pretty good way to get a unique identifier.

No Selection

## Choose options for your new project:

|                    |  |
|--------------------|--|
| Product Name       | Matchismo                                |
| Organization Name  | CS193p                                   |
| Company Identifier | edu.stanford.cs193p.instructor           |
| Bundle Identifier  | edu.stanford.cs193p.instructor.Matchismo |
| Class Prefix       | CardGame                                 |
| Devices            | Universal                                |

Enter “CardGame” as the prefix for the name of the Controller this template is going to generate for us. Thus our Controller class will be called CardGameViewController.

Sometimes we would use the name of the application for this prefix. In fact, older versions of Xcode would automatically do this whether we wanted it or not.

We don't want the names of the classes generated by the template to be too generic (e.g. “ViewController”). That's why we specify this prefix.

Cancel

No Selection

### Choose options for your new project:

Product Name

Organization Name

Company Identifier

Bundle Identifier

Class Prefix

Device:  iPad  iPhone  Universal

No Selection

Cancel

Previous

Next

Set the Device we're developing for to iPhone.

Our first application is going to be for the iPhone (not iPad). At least for starters.

A Universal application runs on both iPhone and iPad. In a Universal application, the iPad and the iPhone each has its own UI design (since they have different UI idioms). Xcode provides tools for designing two different UIs in the same application.

Loading

No Issues

### Choose options for your new project:

Product Name

Organization Name

Company Identifier

Bundle Identifier

Class Prefix

Devices

Cancel

Previous

Next

No Selection

Then click Next.



Xcode wants to know where to store this project's directory.

Home directory.

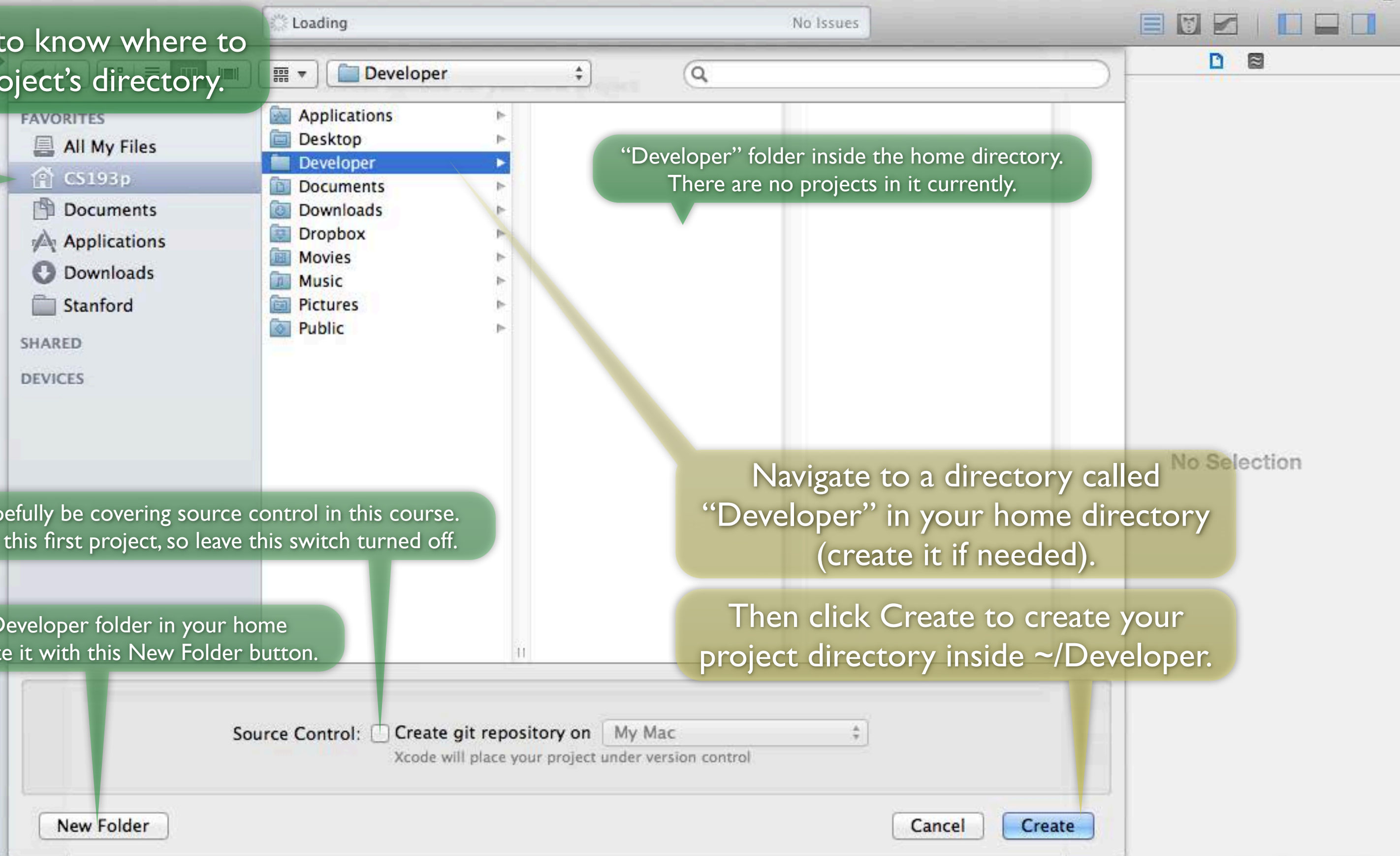
"Developer" folder inside the home directory. There are no projects in it currently.

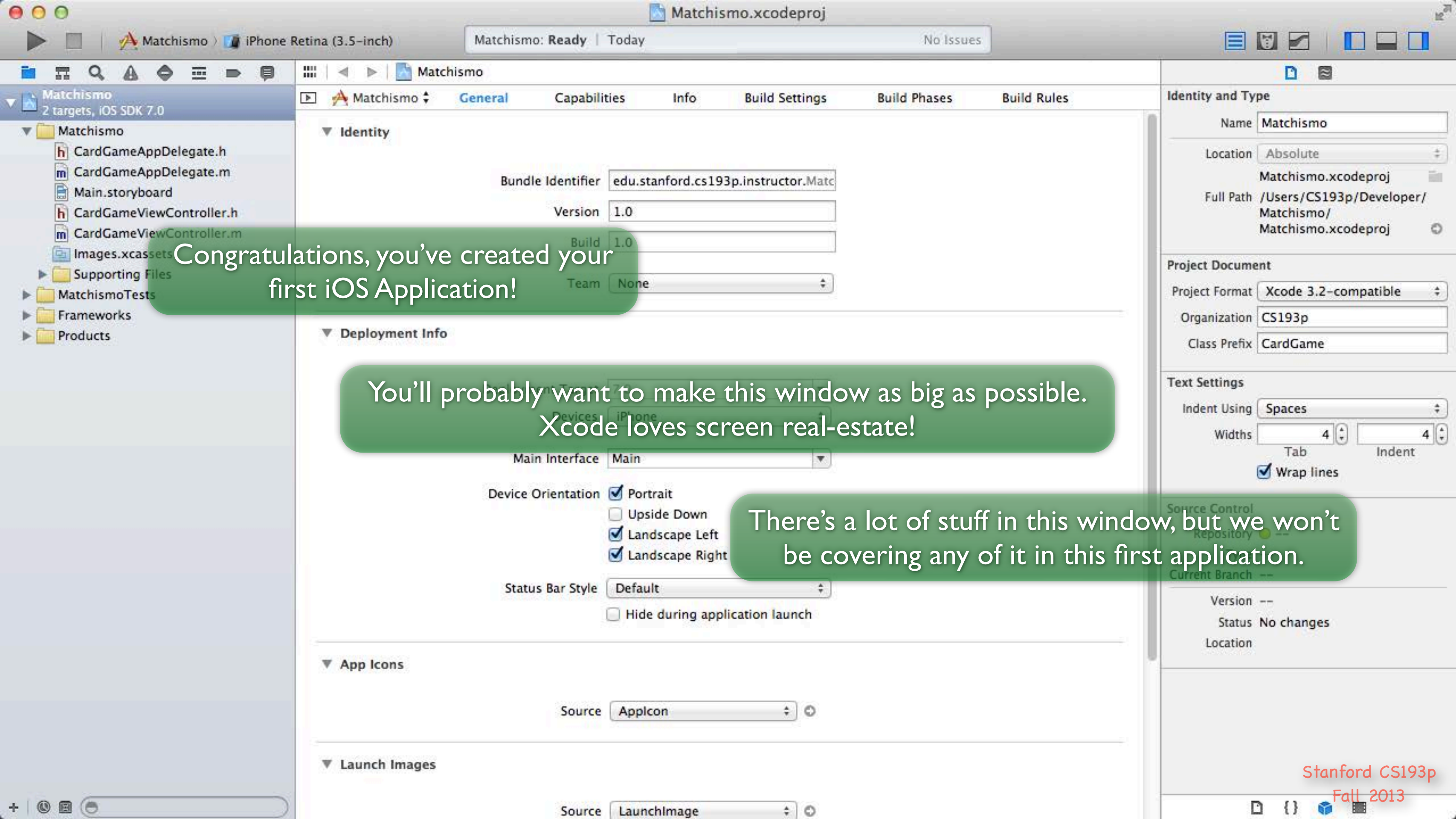
We will hopefully be covering source control in this course. But not for this first project, so leave this switch turned off.

If you don't have a Developer folder in your home directory, you can create it with this New Folder button.

Navigate to a directory called "Developer" in your home directory (create it if needed).

Then click Create to create your project directory inside ~/Developer.





Congratulations, you've created your first iOS Application!

You'll probably want to make this window as big as possible. Xcode loves screen real-estate!

There's a lot of stuff in this window, but we won't be covering any of it in this first application.

The Single View Application template we chose at the beginning has created a simple MVC for us.

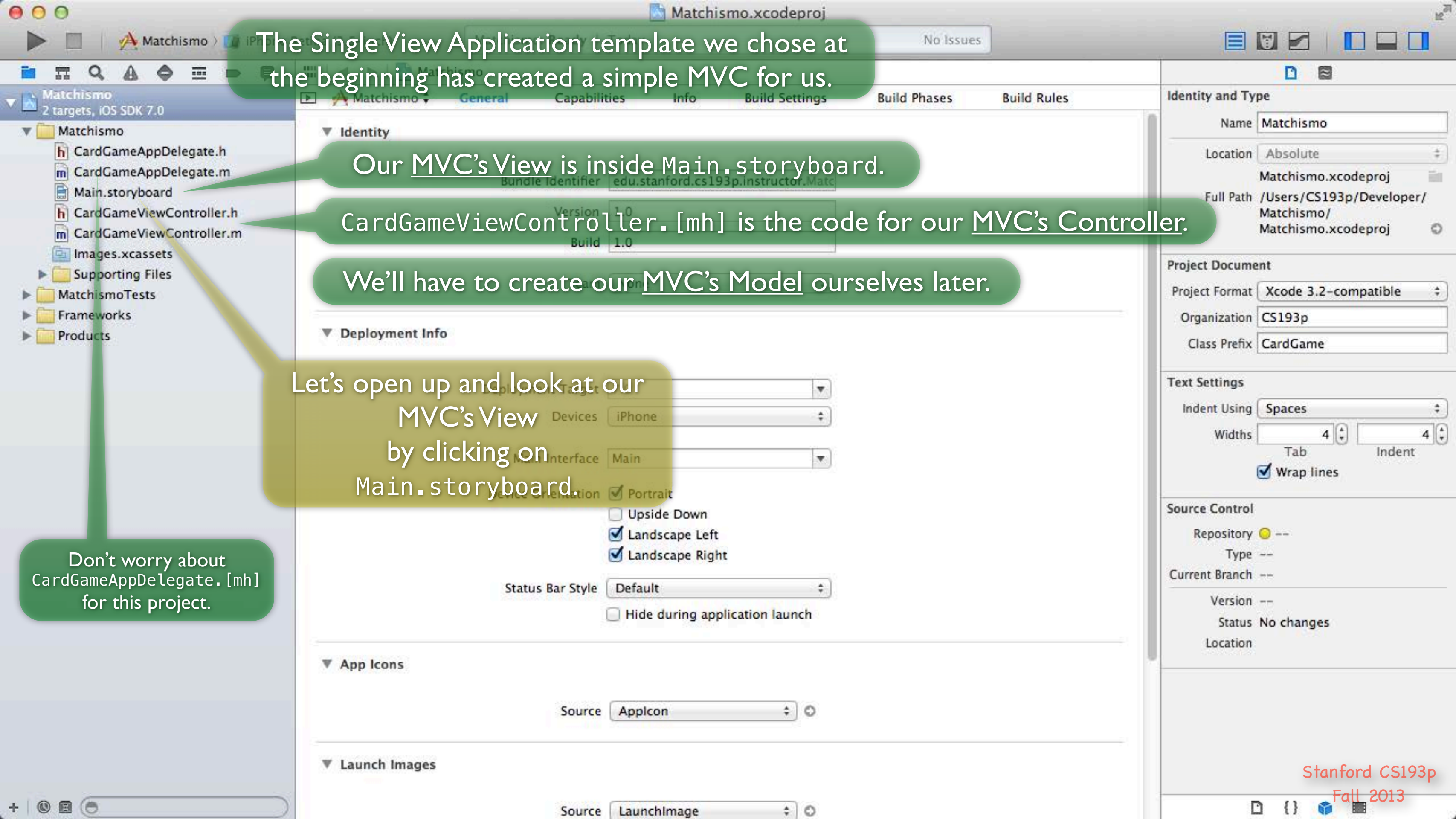
Our MVC's View is inside Main.storyboard.

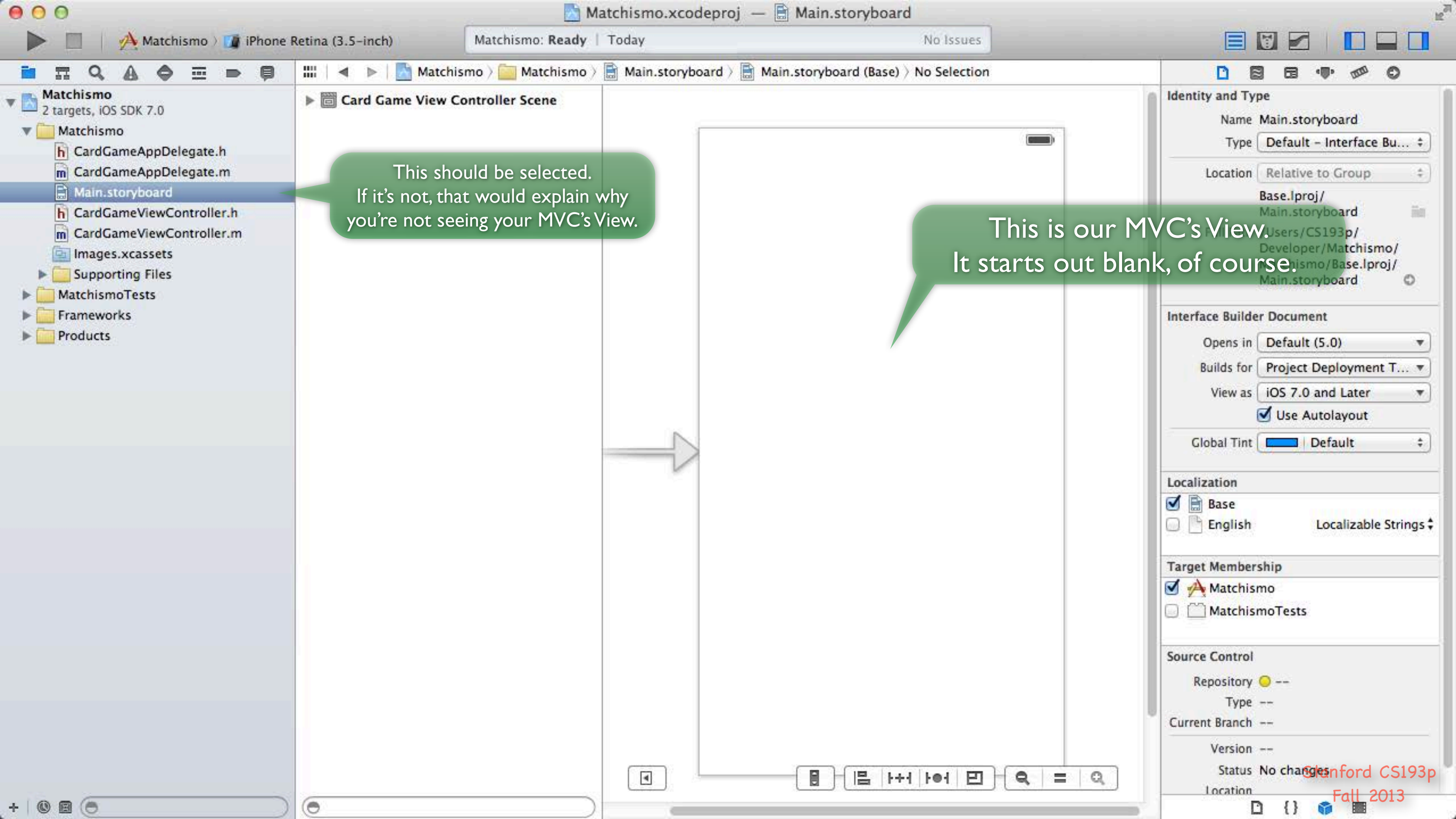
CardGameViewController.[mh] is the code for our MVC's Controller.

We'll have to create our MVC's Model ourselves later.

Let's open up and look at our MVC's View by clicking on Main.storyboard.

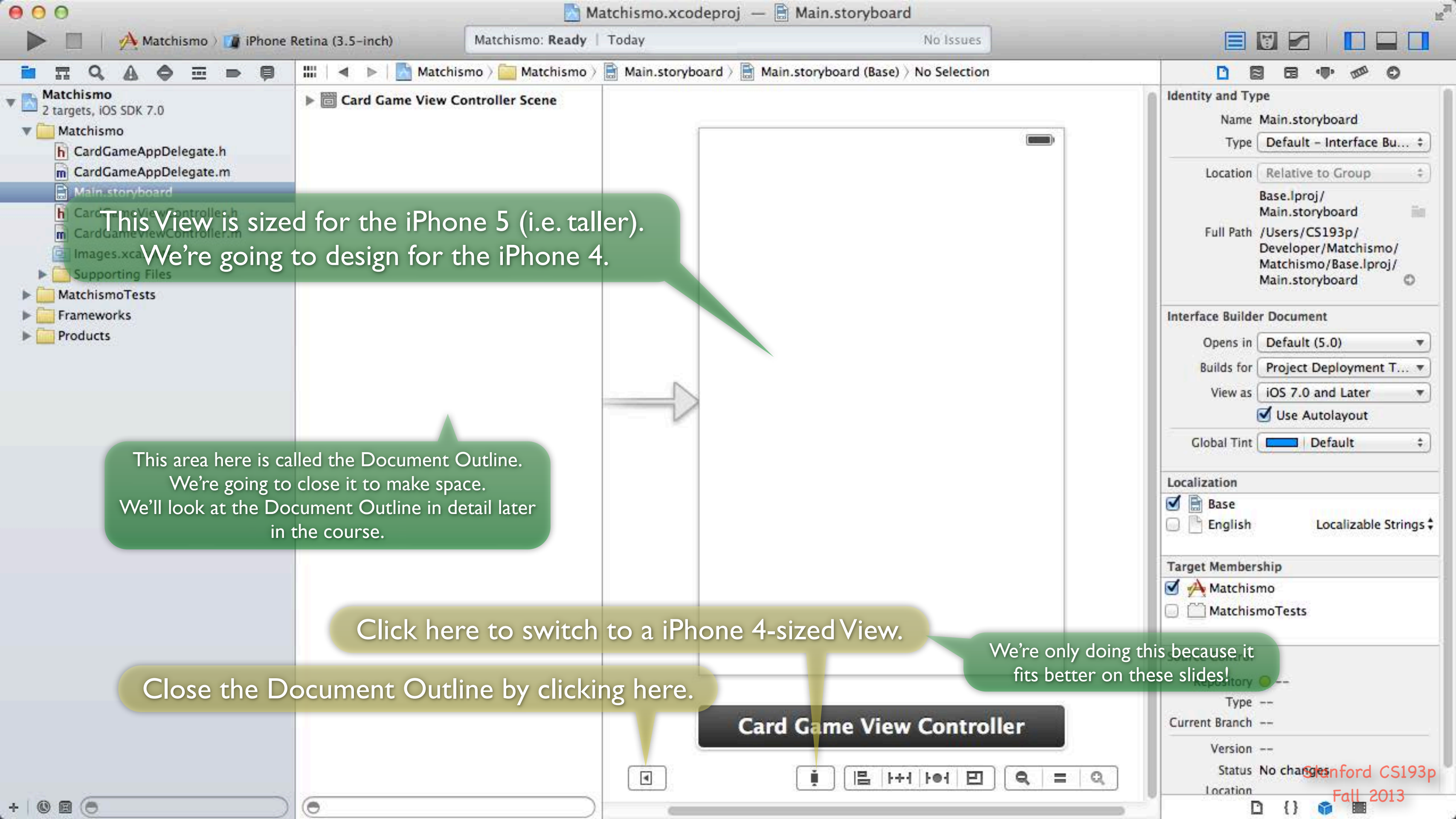
Don't worry about CardGameAppDelegate.[mh] for this project.





This should be selected.  
If it's not, that would explain why  
you're not seeing your MVC's View.

This is our MVC's View.  
It starts out blank, of course.



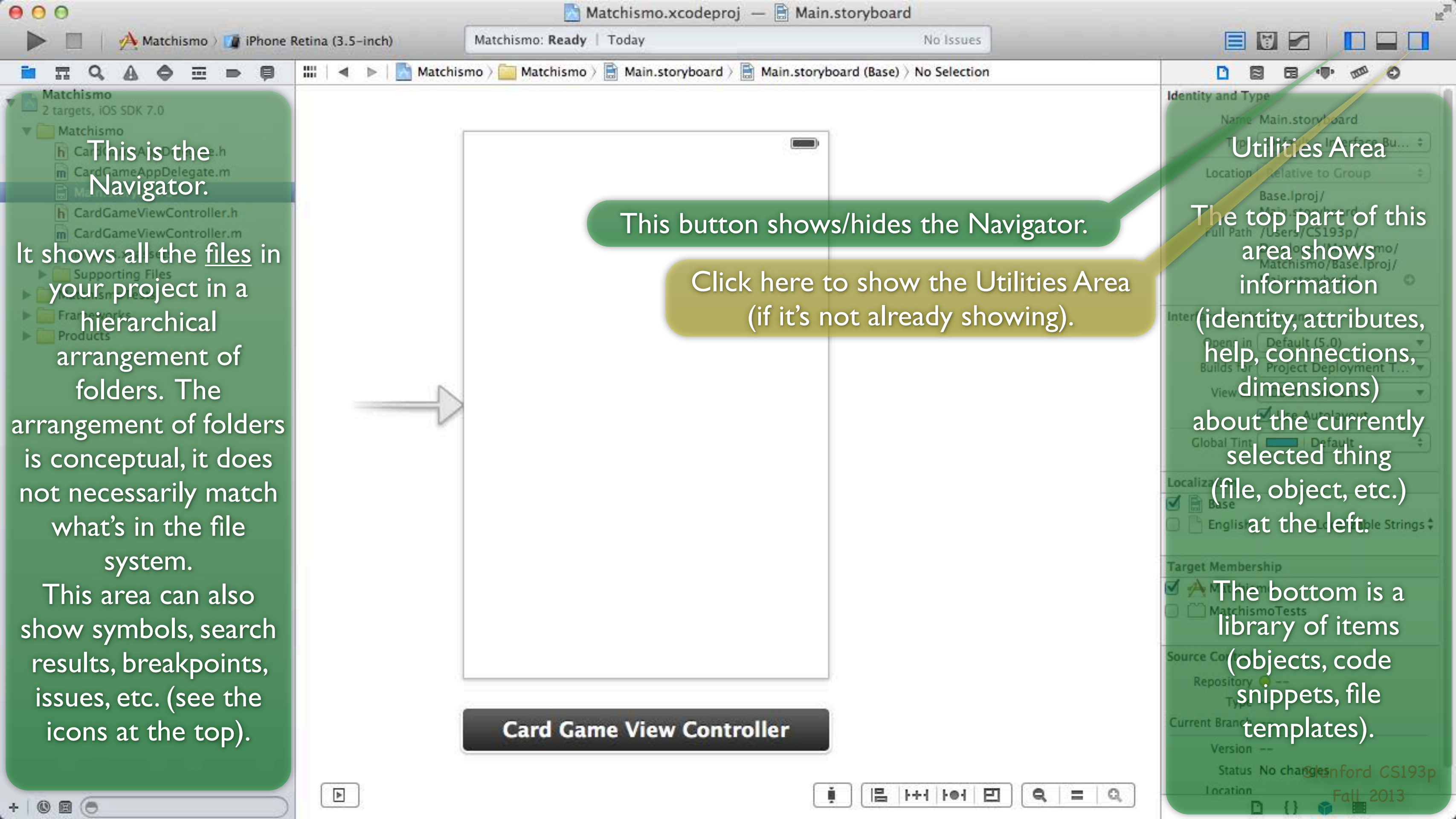
This View is sized for the iPhone 5 (i.e. taller). We're going to design for the iPhone 4.

This area here is called the Document Outline. We're going to close it to make space. We'll look at the Document Outline in detail later in the course.

Click here to switch to a iPhone 4-sized View.

Close the Document Outline by clicking here.

We're only doing this because it fits better on these slides!



This is the Navigator.

It shows all the files in your project in a hierarchical arrangement of folders. The arrangement of folders is conceptual, it does not necessarily match what's in the file system.

This area can also show symbols, search results, breakpoints, issues, etc. (see the icons at the top).

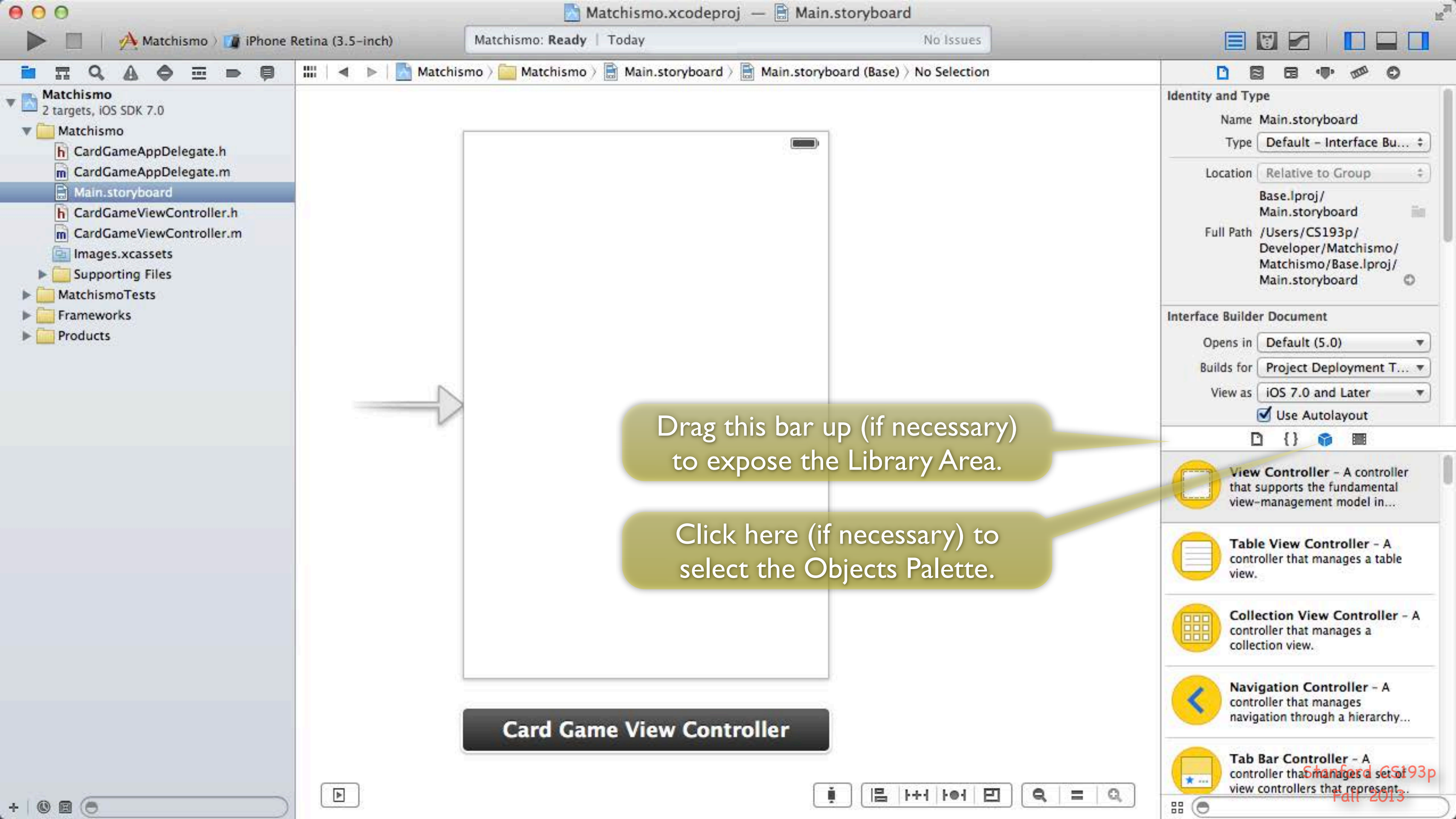
This button shows/hides the Navigator.

Click here to show the Utilities Area (if it's not already showing).

Utilities Area

The top part of this area shows information (identity, attributes, help, connections, dimensions) about the currently selected thing (file, object, etc.) at the left.

The bottom is a library of items (objects, code snippets, file templates).



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



Drag this bar up (if necessary) to expose the Library Area.

Click here (if necessary) to select the Objects Palette.

Card Game View Controller

**Identity and Type**

Name Main.storyboard

Type Default - Interface Bu... ▾

Location Relative to Group ▾

Base.lproj/  
Main.storyboard

Full Path /Users/CS193p/  
Developer/Matchismo/  
Matchismo/Base.lproj/  
Main.storyboard

**Interface Builder Document**

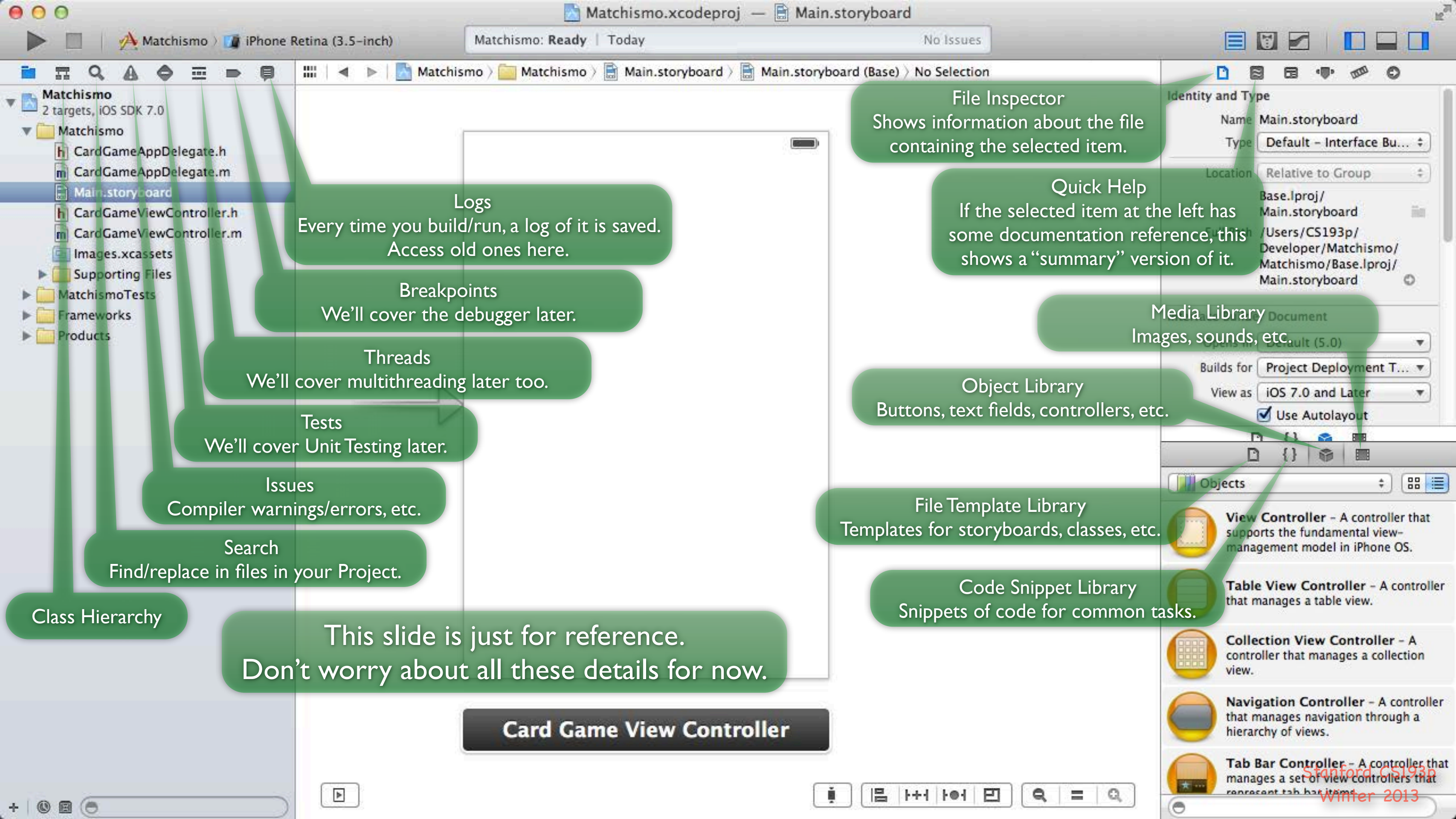
Opens in Default (5.0) ▾

Builds for Project Deployment T... ▾

View as iOS 7.0 and Later ▾

Use Autolayout

- View Controller** - A controller that supports the fundamental view-management model in...
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy...
- Tab Bar Controller** - A controller that manages a set of view controllers that represent...



Class Hierarchy

Search  
Find/replace in files in your Project.

Issues  
Compiler warnings/errors, etc.

Tests  
We'll cover Unit Testing later.

Threads  
We'll cover multithreading later too.

Breakpoints  
We'll cover the debugger later.

Logs  
Every time you build/run, a log of it is saved.  
Access old ones here.

Threads  
We'll cover multithreading later too.

Breakpoints  
We'll cover the debugger later.

Logs  
Every time you build/run, a log of it is saved.  
Access old ones here.

Tests  
We'll cover Unit Testing later.

File Inspector  
Shows information about the file containing the selected item.

Quick Help  
If the selected item at the left has some documentation reference, this shows a "summary" version of it.

Media Library  
Images, sounds, etc.

Object Library  
Buttons, text fields, controllers, etc.

File Template Library  
Templates for storyboards, classes, etc.

Code Snippet Library  
Snippets of code for common tasks.

This slide is just for reference.  
Don't worry about all these details for now.

Card Game View Controller



It's time to start building the user-interface in our MVC View. We're building a card game, so we'll start with our first "card." We'll use a button to represent it.

The Objects Palette contains a bunch of objects you can use to build your View.

Scroll down to find Button.

If you are not seeing Button in the list, try clicking on your View.

Card Game View Controller

**Identity and Type**  
Name: Main.storyboard  
Type: Default - Interface Bu...  
Location: Relative to Group  
Base.lproj/Main.storyboard  
Full Path: /Users/CS193p/Developer/Matchismo/Matchismo/Base.lproj/Main.storyboard

**Interface Builder Document**  
Opens in: Default (5.0)  
Builds for: Project Deployment T...  
View as: iOS 7.0 and Later  
 Use Autolayout

directly available in Interface...

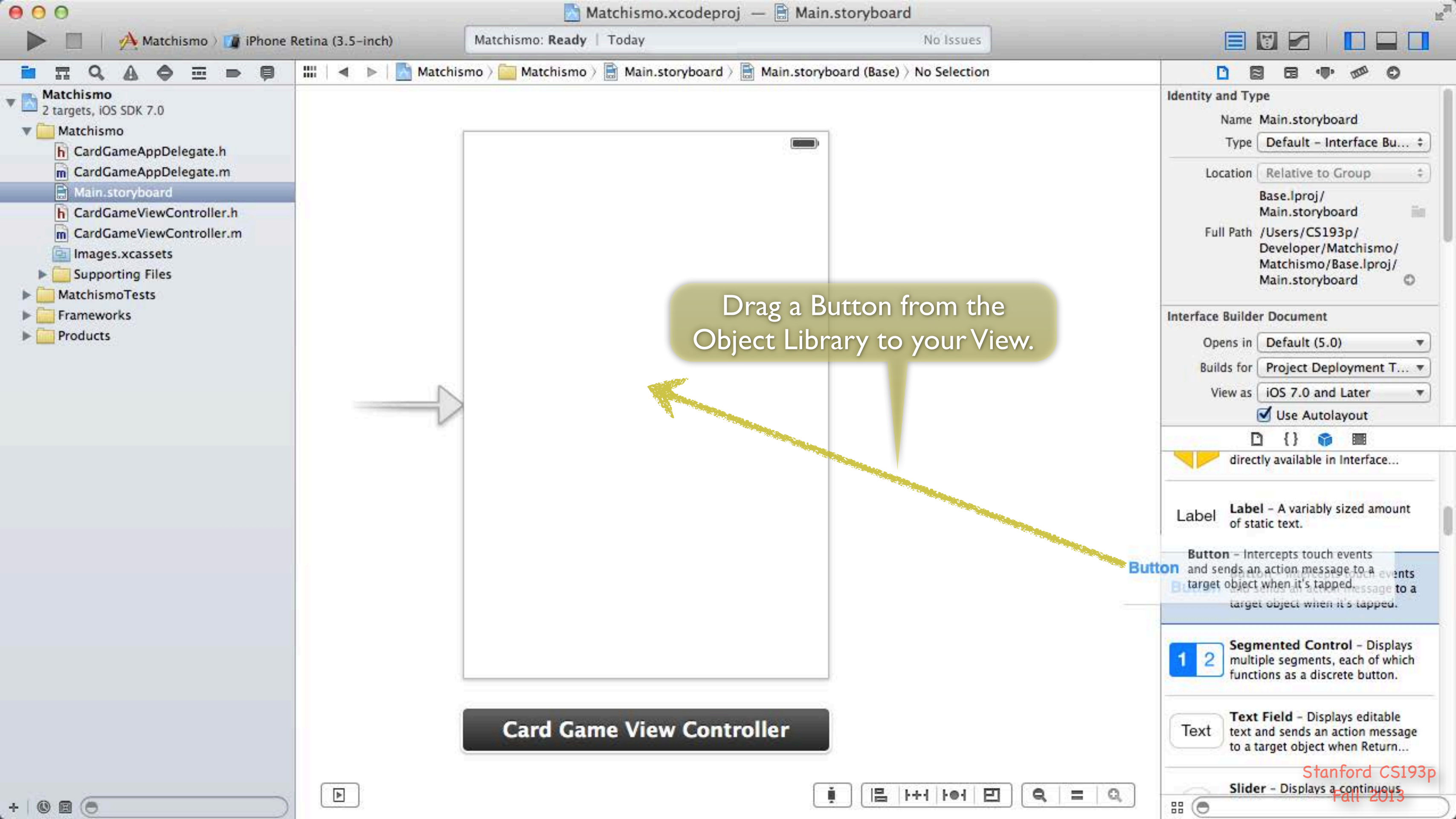
**Label** Label - A variably sized amount of static text.

**Button** Button - Intercepts touch events and sends an action message to a target object when it's tapped.

**1 2 Segmented Control** - Displays multiple segments, each of which functions as a discrete button.

**Text** Text Field - Displays editable text and sends an action message to a target object when Return...

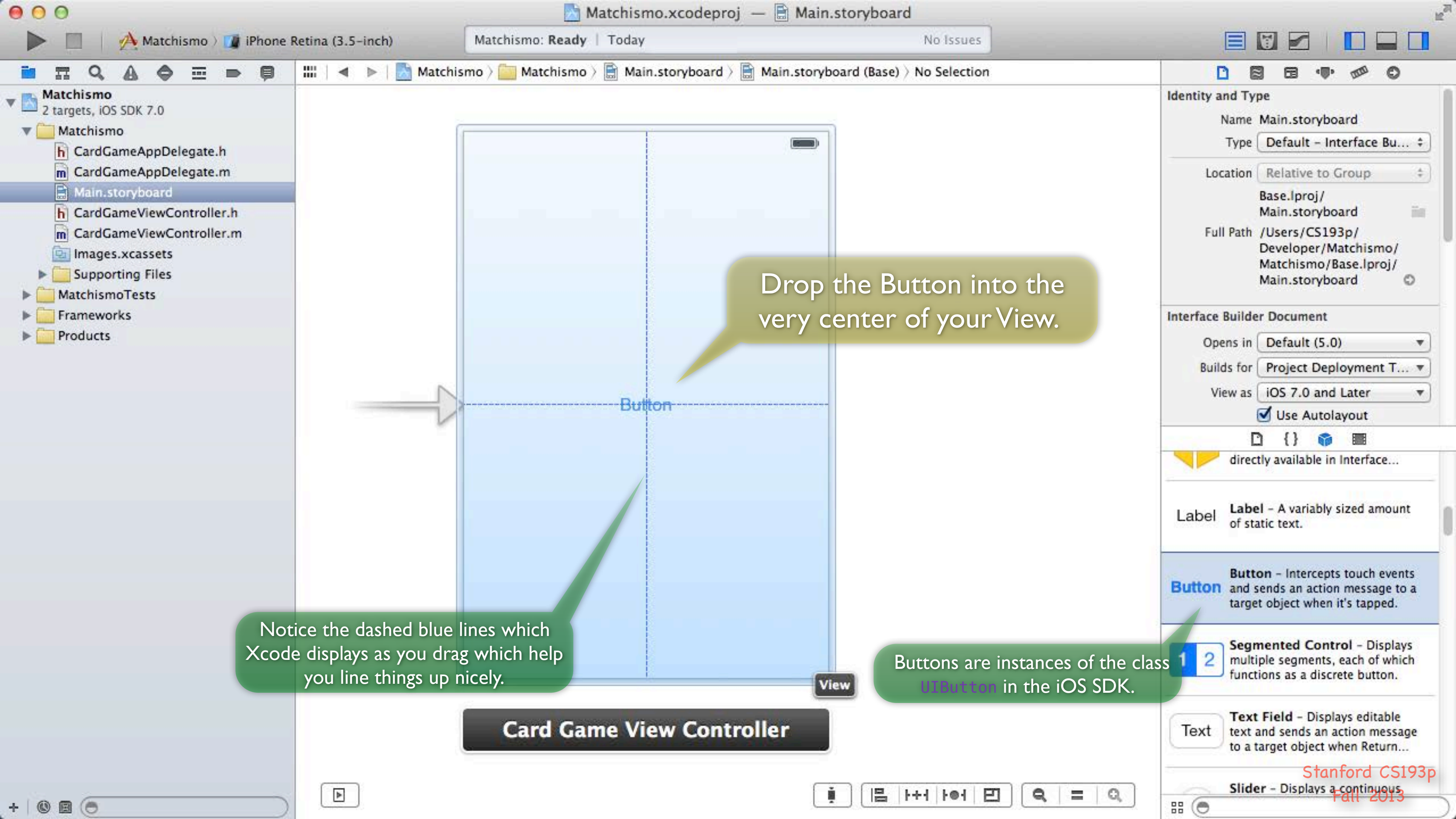
**Slider** - Displays a continuous



Drag a Button from the Object Library to your View.

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Card Game View Controller



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Main.storyboard > Main.storyboard (Base) > No Selection

Drop the Button into the very center of your View.

Notice the dashed blue lines which Xcode displays as you drag which help you line things up nicely.

Buttons are instances of the class `UIButton` in the iOS SDK.

View

Card Game View Controller

Identity and Type

Name Main.storyboard

Type Default - Interface Bu...

Location Relative to Group

Base.lproj/  
Main.storyboard

Full Path /Users/CS193p/  
Developer/Matchismo/  
Matchismo/Base.lproj/  
Main.storyboard

Interface Builder Document

Opens in Default (5.0)

Builds for Project Deployment T...

View as iOS 7.0 and Later

Use Autolayout

directly available in Interface...

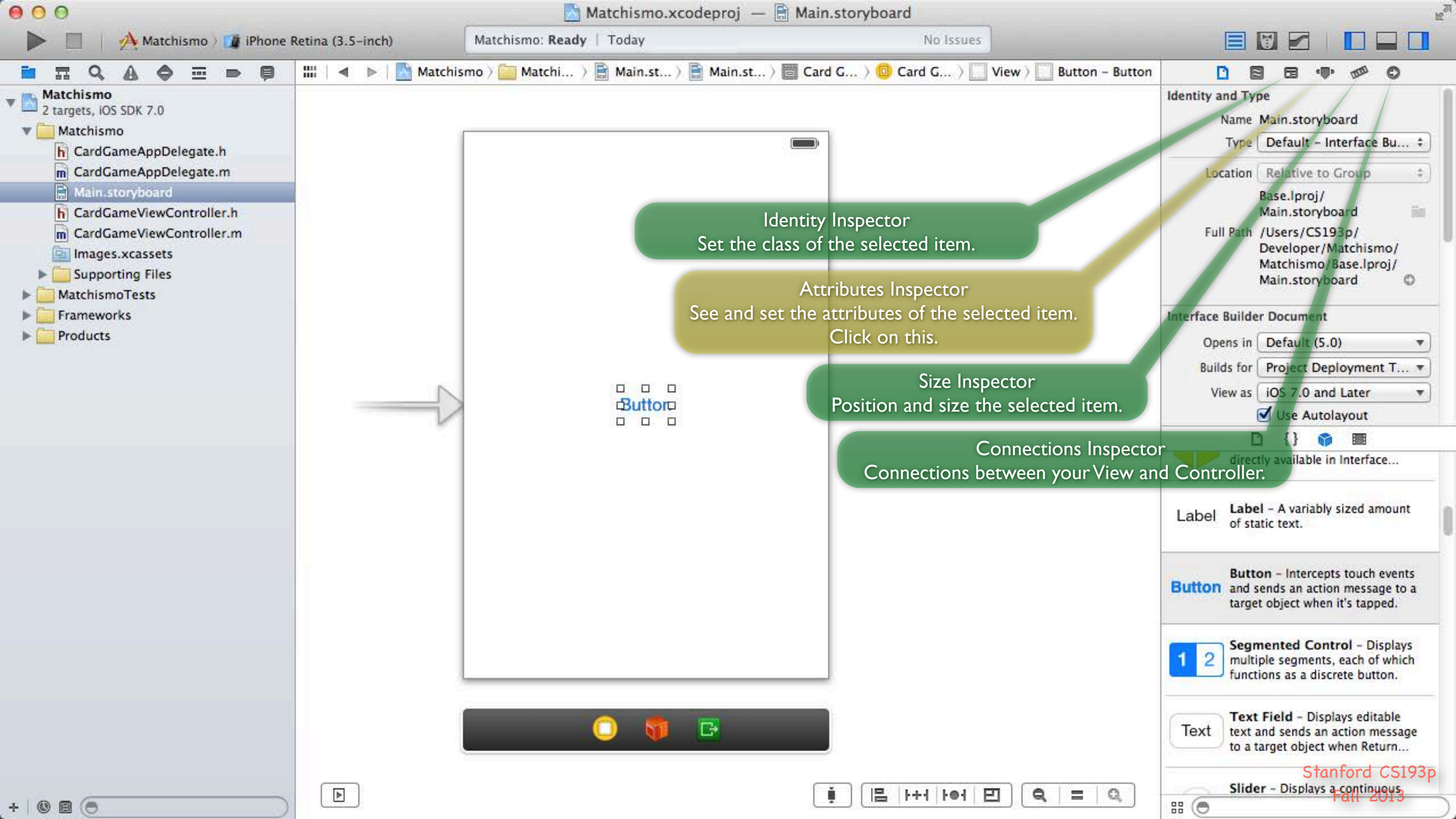
Label Label - A variably sized amount of static text.

Button Button - Intercepts touch events and sends an action message to a target object when it's tapped.

1 2 Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Text Text Field - Displays editable text and sends an action message to a target object when Return...

Slider - Displays a continuous...

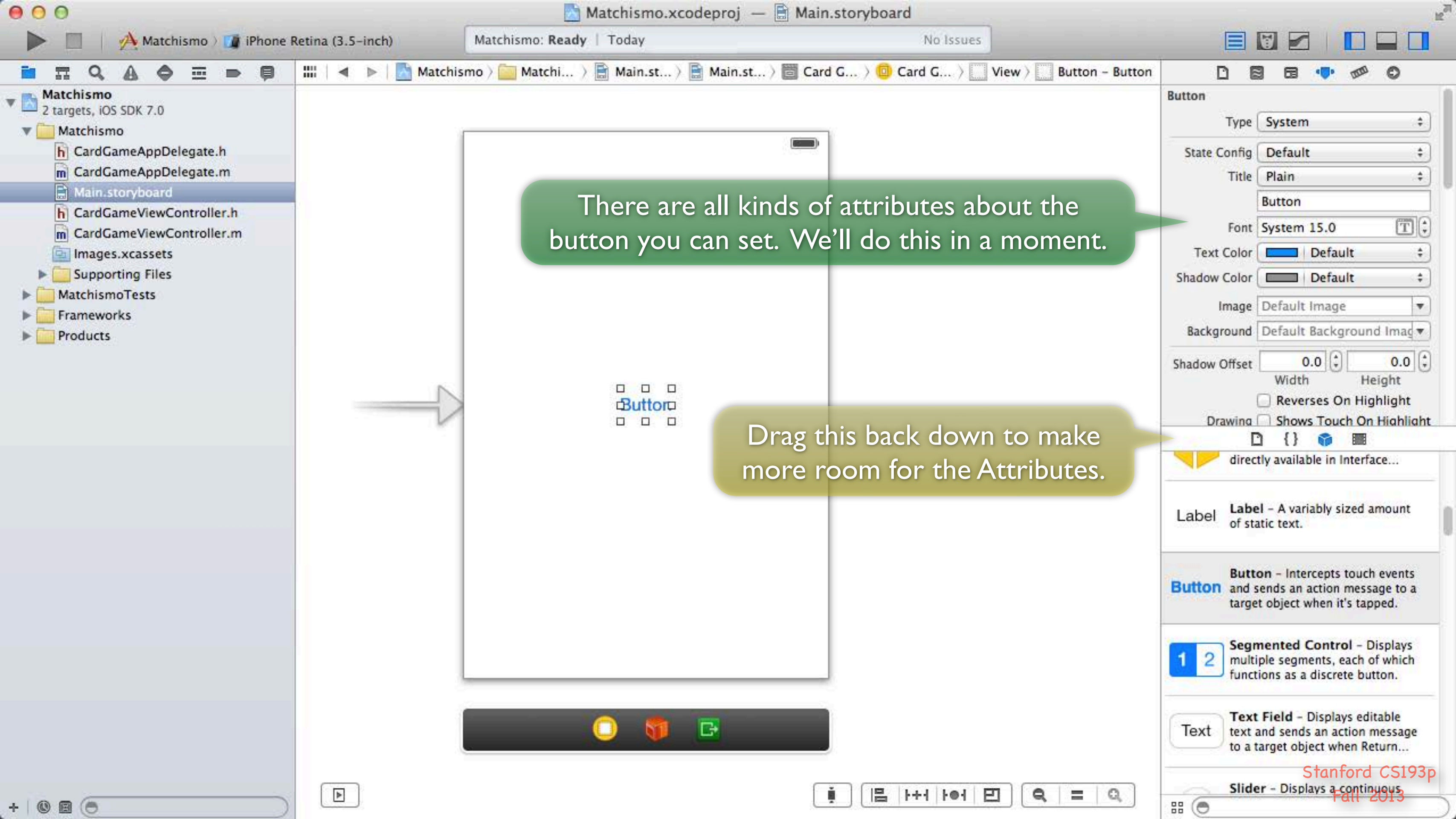


Identity Inspector  
Set the class of the selected item.

Attributes Inspector  
See and set the attributes of the selected item.  
Click on this.

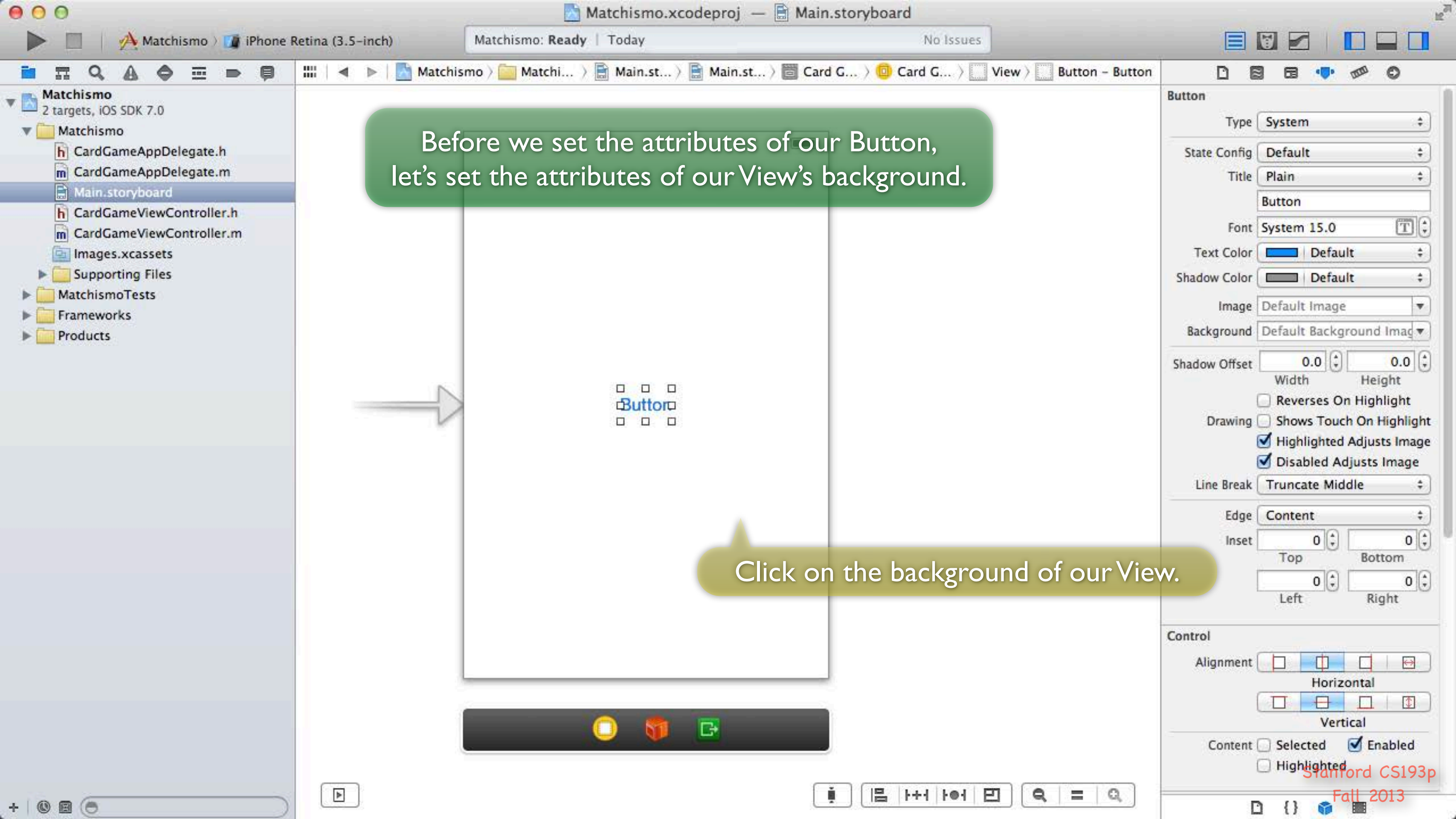
Size Inspector  
Position and size the selected item.

Connections Inspector  
Connections between your View and Controller.



There are all kinds of attributes about the button you can set. We'll do this in a moment.

Drag this back down to make more room for the Attributes.



Before we set the attributes of our Button, let's set the attributes of our View's background.

Click on the background of our View.

**Button**

Type

State Config

Title

Font

Text Color

Shadow Color

Image

Background

Shadow Offset

Width Height

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break

Edge

Inset

Top Bottom

Left Right

**Control**

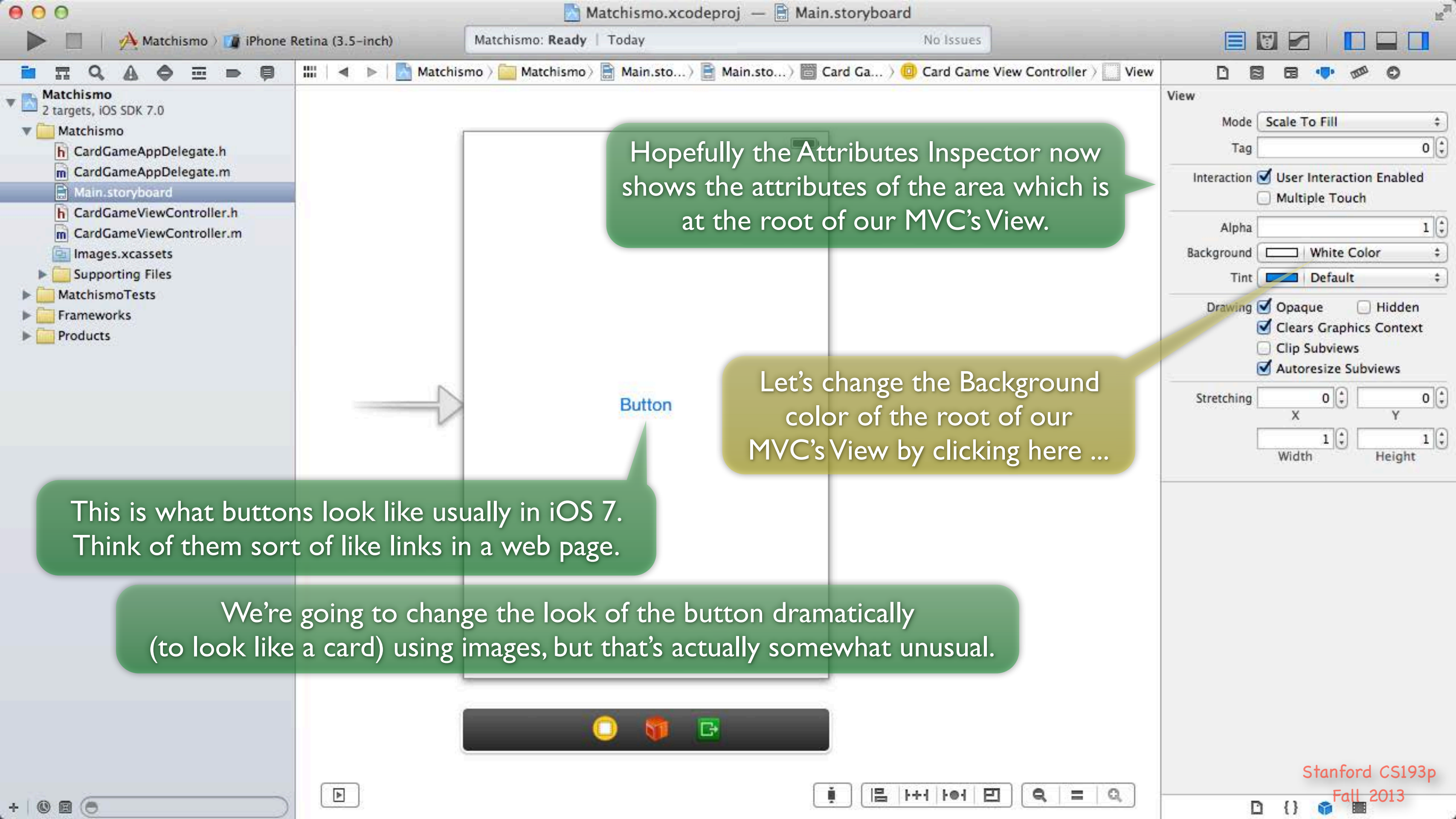
Alignment

Horizontal

Vertical

Content  Selected  Enabled

Highlighted

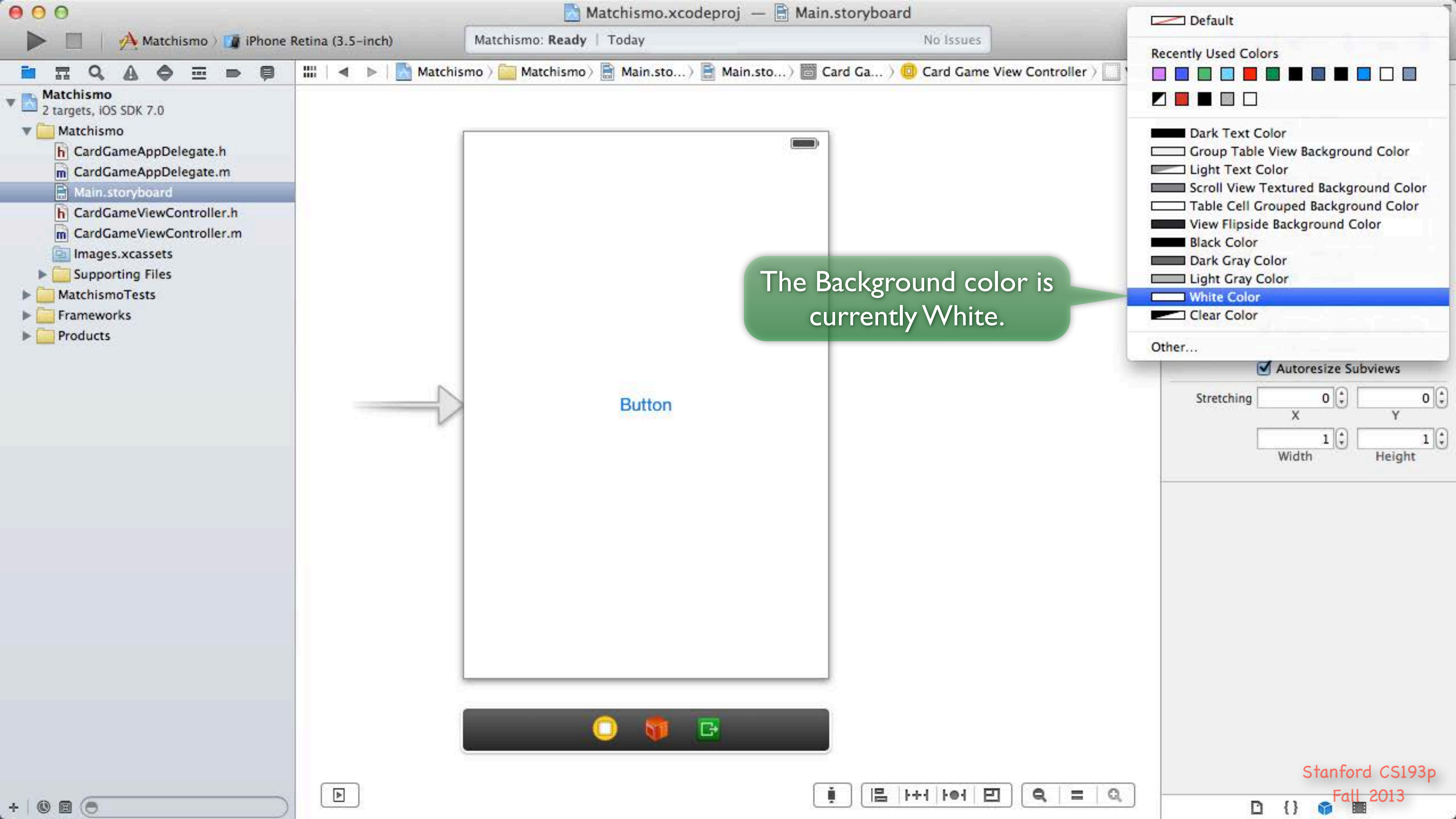


Hopefully the Attributes Inspector now shows the attributes of the area which is at the root of our MVC's View.

Let's change the Background color of the root of our MVC's View by clicking here ...

This is what buttons look like usually in iOS 7. Think of them sort of like links in a web page.

We're going to change the look of the button dramatically (to look like a card) using images, but that's actually somewhat unusual.



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

The Background color is currently White.

Default

Recently Used Colors

- Dark Text Color
- Group Table View Background Color
- Light Text Color
- Scroll View Textured Background Color
- Table Cell Grouped Background Color
- View Flipside Background Color
- Black Color
- Dark Gray Color
- Light Gray Color
- White Color**
- Clear Color

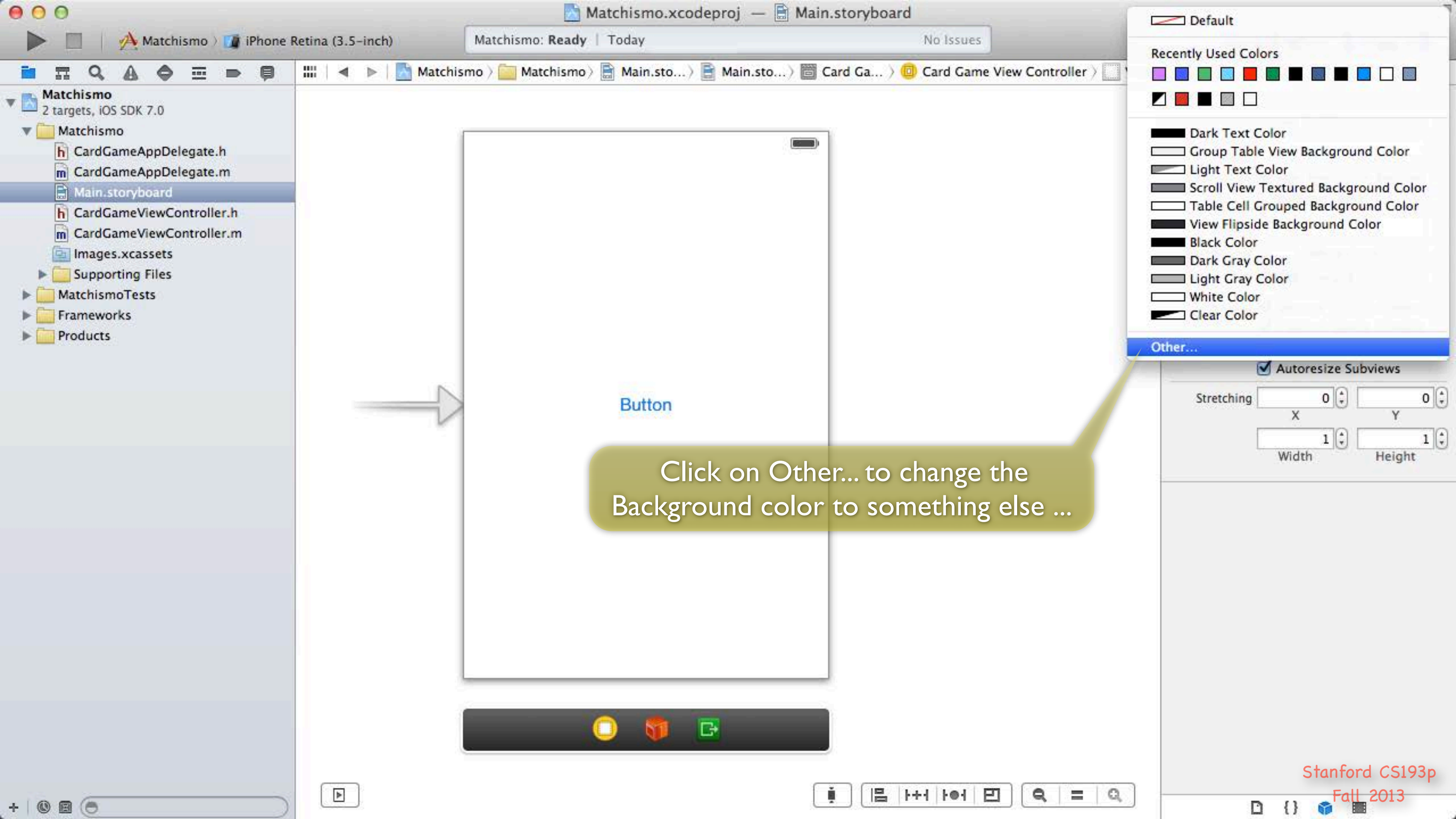
Other...

Autosize Subviews

Stretching X: 0 Y: 0

Width: 1 Height: 1





- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



Click on Other... to change the Background color to something else ...

Default

Recently Used Colors

- Color swatches

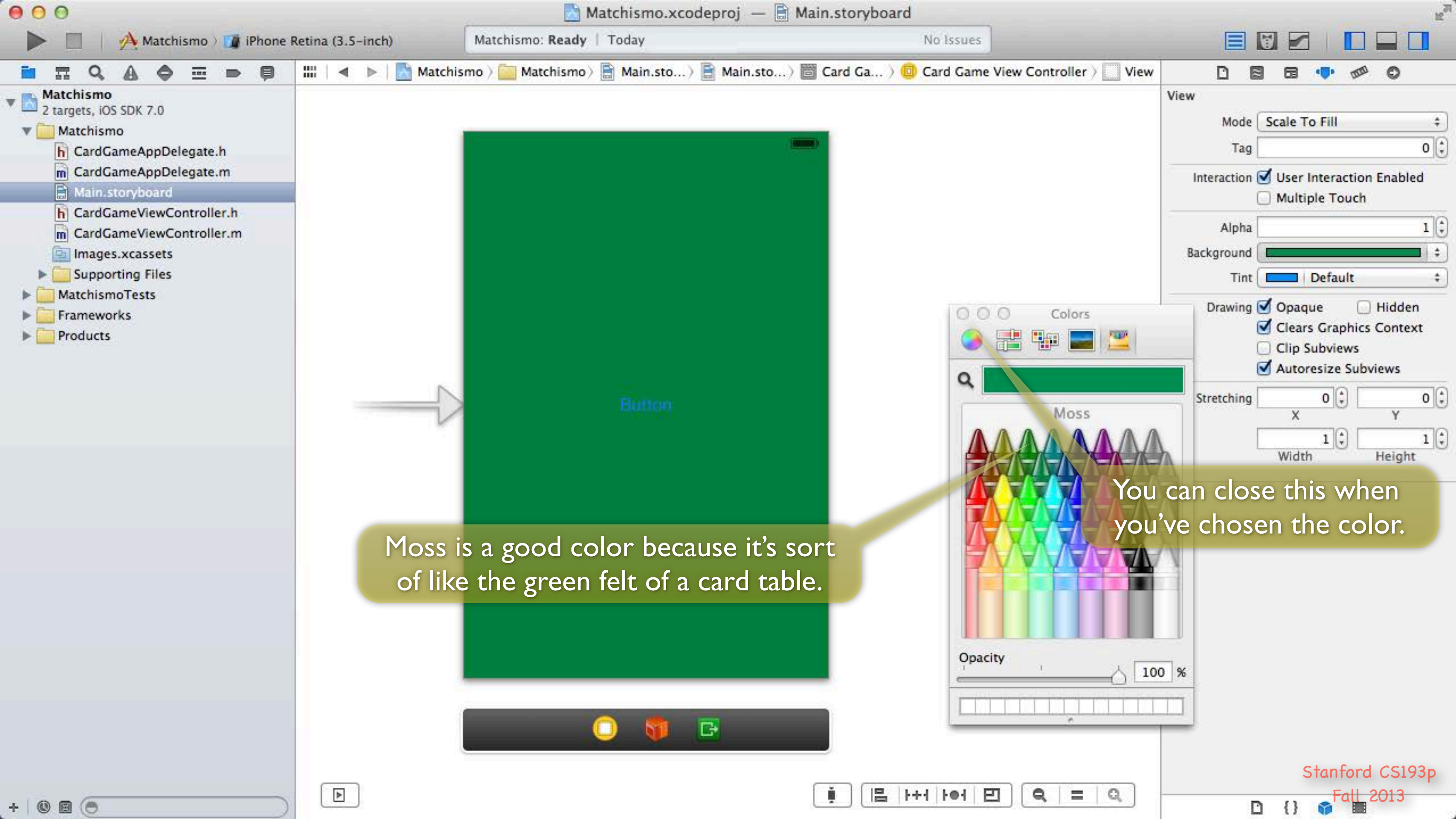
- Dark Text Color
- Group Table View Background Color
- Light Text Color
- Scroll View Textured Background Color
- Table Cell Grouped Background Color
- View Flipside Background Color
- Black Color
- Dark Gray Color
- Light Gray Color
- White Color
- Clear Color

Other...

Autosize Subviews

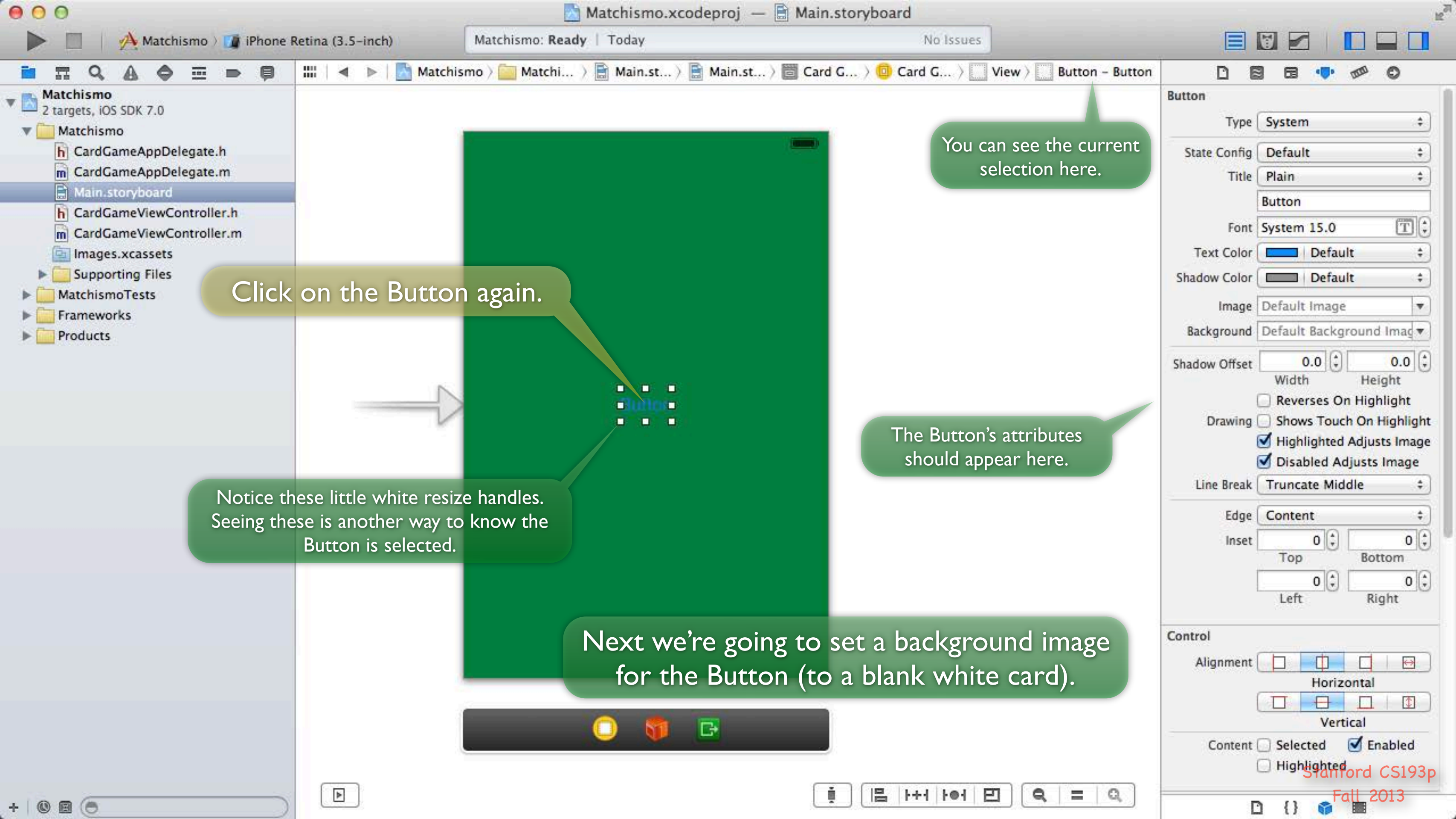
Stretching X: 0 Y: 0

Width: 1 Height: 1



Moss is a good color because it's sort of like the green felt of a card table.

You can close this when you've chosen the color.



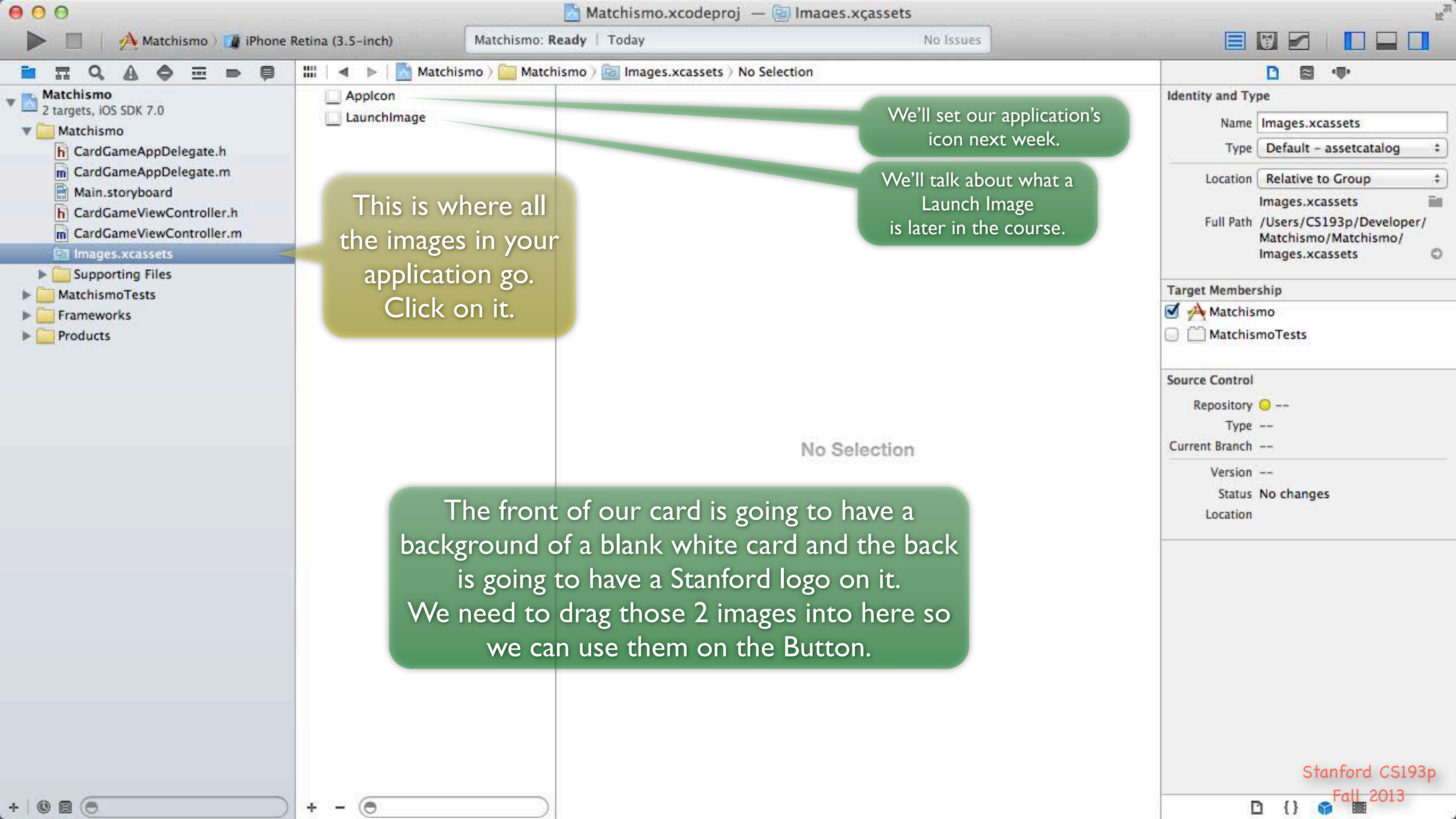
Click on the Button again.

Notice these little white resize handles. Seeing these is another way to know the Button is selected.

Next we're going to set a background image for the Button (to a blank white card).

You can see the current selection here.

The Button's attributes should appear here.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets**
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products

Matchismo > Matchismo > Images.xcassets > No Selection

- Applcon
- LaunchImage

This is where all the images in your application go. Click on it.

We'll set our application's icon next week.

We'll talk about what a Launch Image is later in the course.

No Selection

The front of our card is going to have a background of a blank white card and the back is going to have a Stanford logo on it. We need to drag those 2 images into here so we can use them on the Button.

**Identity and Type**

Name: Images.xcassets

Type: Default - assetcatalog

Location: Relative to Group

Images.xcassets

Full Path: /Users/CS193p/Developer/Matchismo/Matchismo/Images.xcassets

**Target Membership**

Matchismo

MatchismoTests

**Source Control**

Repository: --

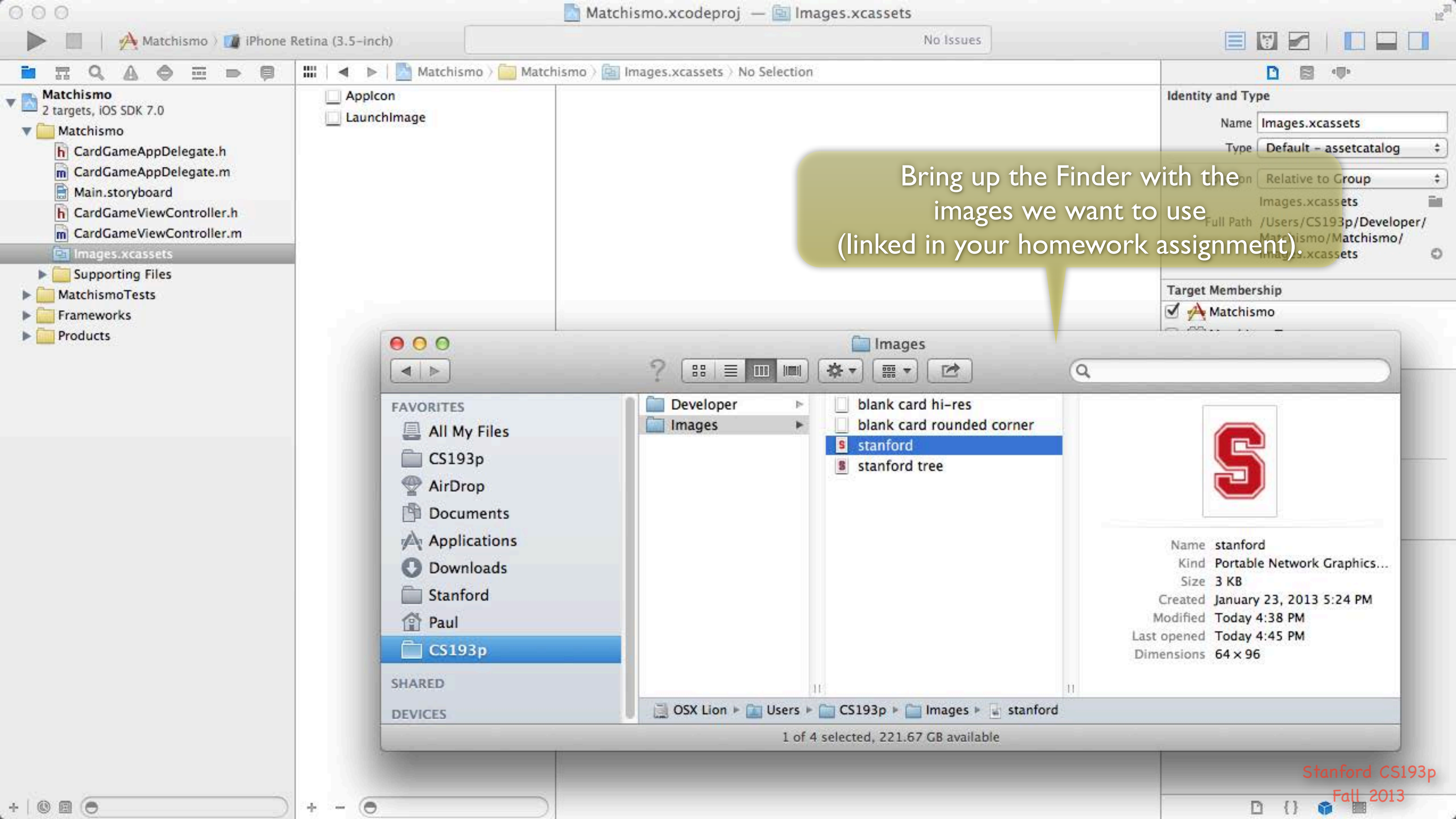
Type: --

Current Branch: --

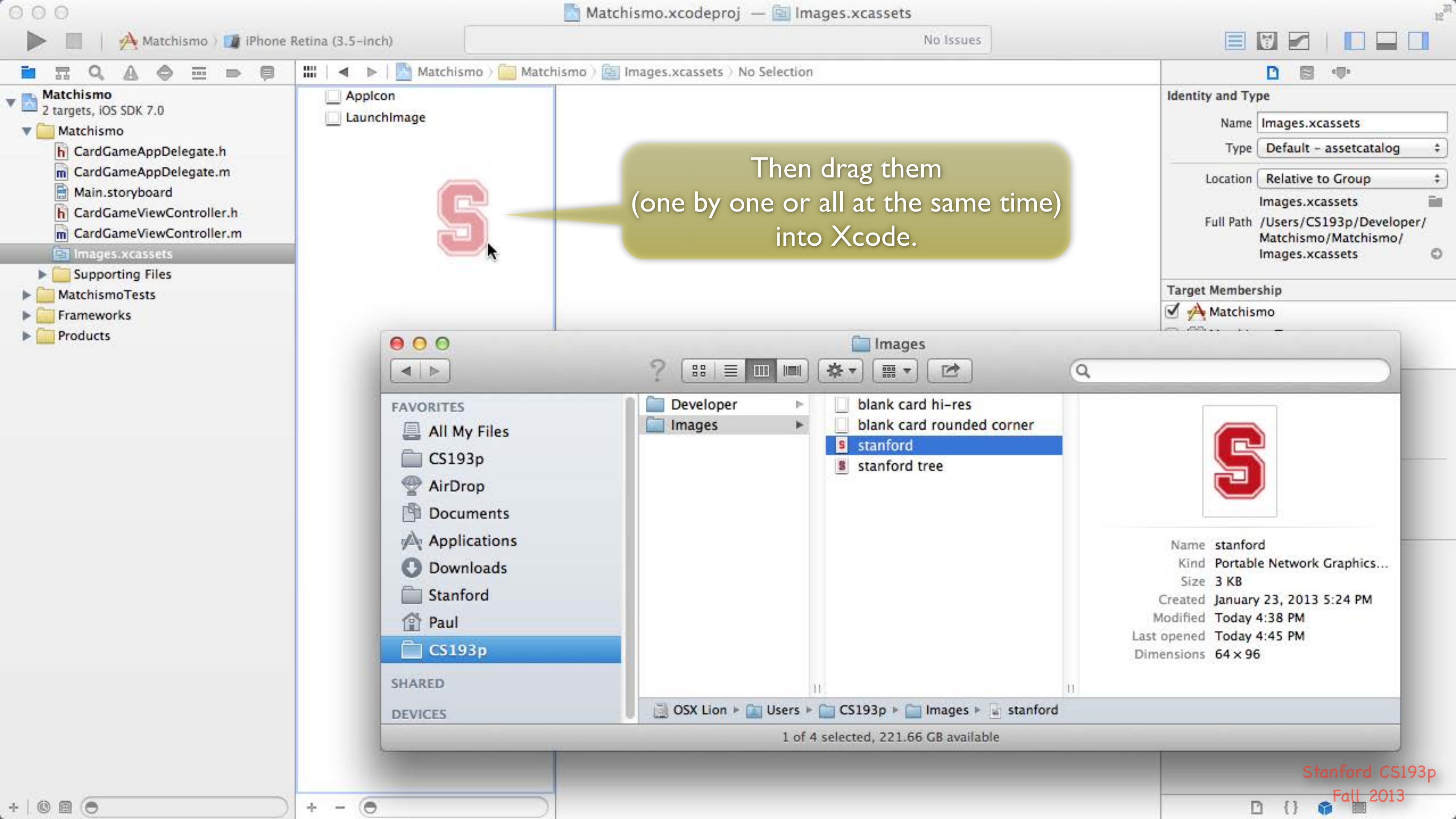
Version: --

Status: No changes

Location: --



Bring up the Finder with the images we want to use (linked in your homework assignment).



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
      - Supporting Files
      - MatchismoTests
      - Frameworks
      - Products

- Applcon
- LaunchImage



Then drag them  
(one by one or all at the same time)  
into Xcode.

**Identity and Type**

Name: Images.xcassets  
Type: Default - assetcatalog  
Location: Relative to Group  
Images.xcassets  
Full Path: /Users/CS193p/Developer/Matchismo/Matchismo/Images.xcassets

**Target Membership**

- Matchismo

**Images**

FAVORITES: All My Files, CS193p, AirDrop, Documents, Applications, Downloads, Stanford, Paul, CS193p

DEVELOPER: Images

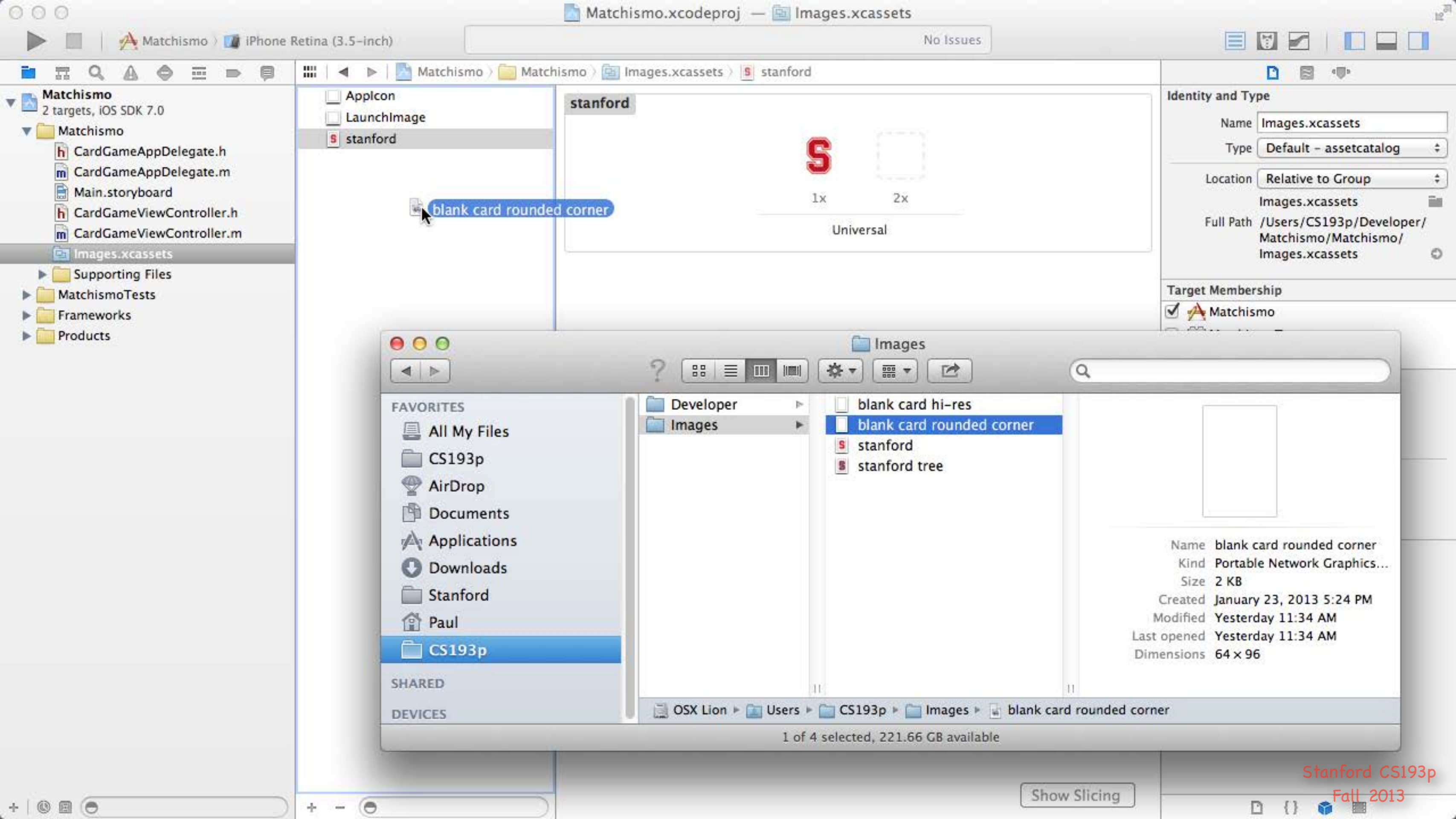
- blank card hi-res
- blank card rounded corner
- stanford**
- stanford tree

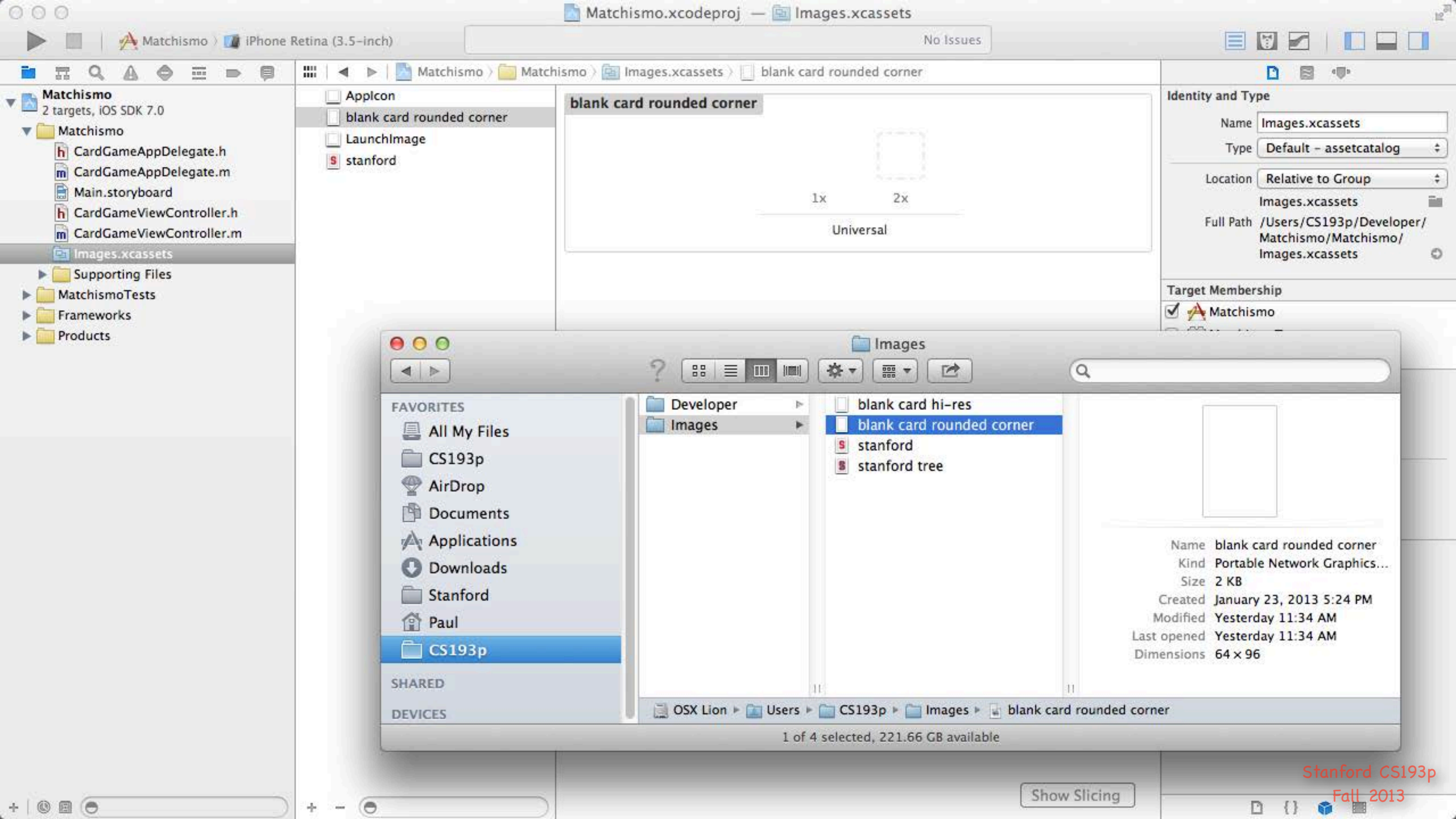
stanford

Name: stanford  
Kind: Portable Network Graphics...  
Size: 3 KB  
Created: January 23, 2013 5:24 PM  
Modified: Today 4:38 PM  
Last opened: Today 4:45 PM  
Dimensions: 64 x 96

OSX Lion > Users > CS193p > Images > stanford

1 of 4 selected, 221.66 GB available





blank card rounded corner

1x 2x

Universal

Identity and Type

Name Images.xcassets  
Type Default - assetcatalog  
Location Relative to Group  
Images.xcassets  
Full Path /Users/CS193p/Developer/Matchismo/Matchismo/Images.xcassets

Target Membership

Matchismo

FAVORITES

- All My Files
- CS193p
- AirDrop
- Documents
- Applications
- Downloads
- Stanford
- Paul
- CS193p

- Developer
- Images

- blank card hi-res
- blank card rounded corner
- stanford
- stanford tree

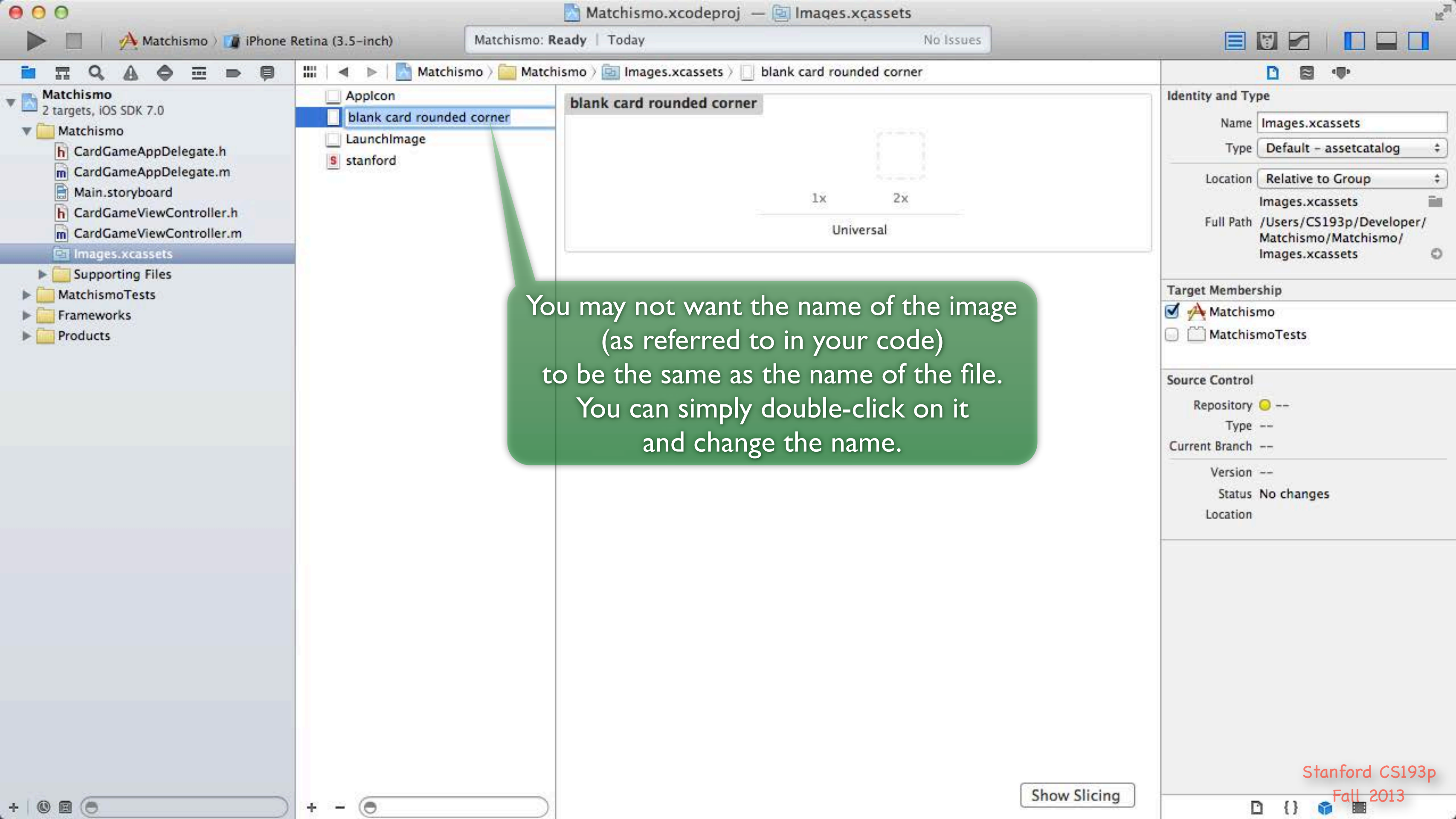
Name blank card rounded corner  
Kind Portable Network Graphics...  
Size 2 KB  
Created January 23, 2013 5:24 PM  
Modified Yesterday 11:34 AM  
Last opened Yesterday 11:34 AM  
Dimensions 64 x 96

OSX Lion > Users > CS193p > Images > blank card rounded corner

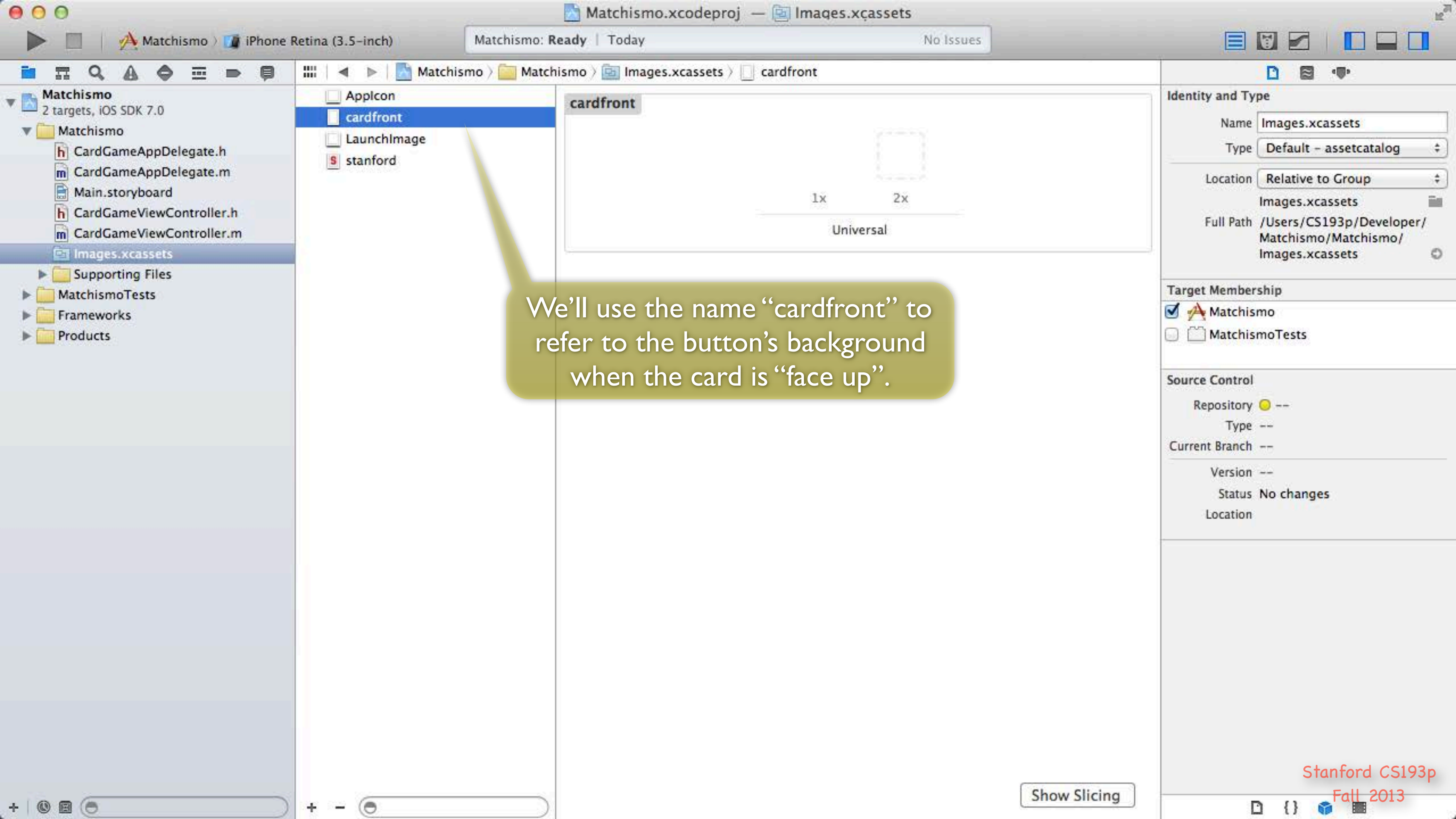
1 of 4 selected, 221.66 GB available

Show Slicing

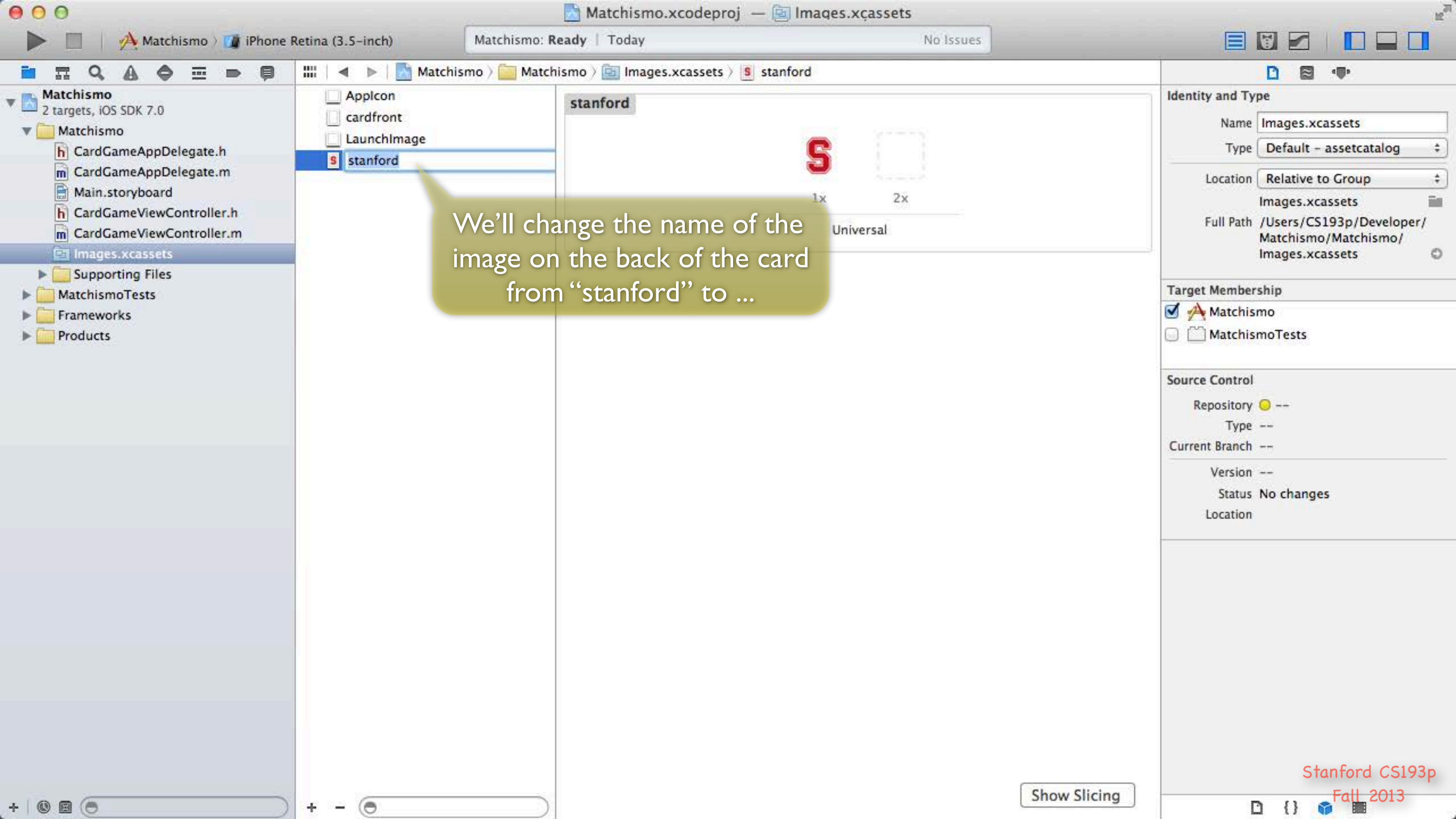




You may not want the name of the image (as referred to in your code) to be the same as the name of the file. You can simply double-click on it and change the name.

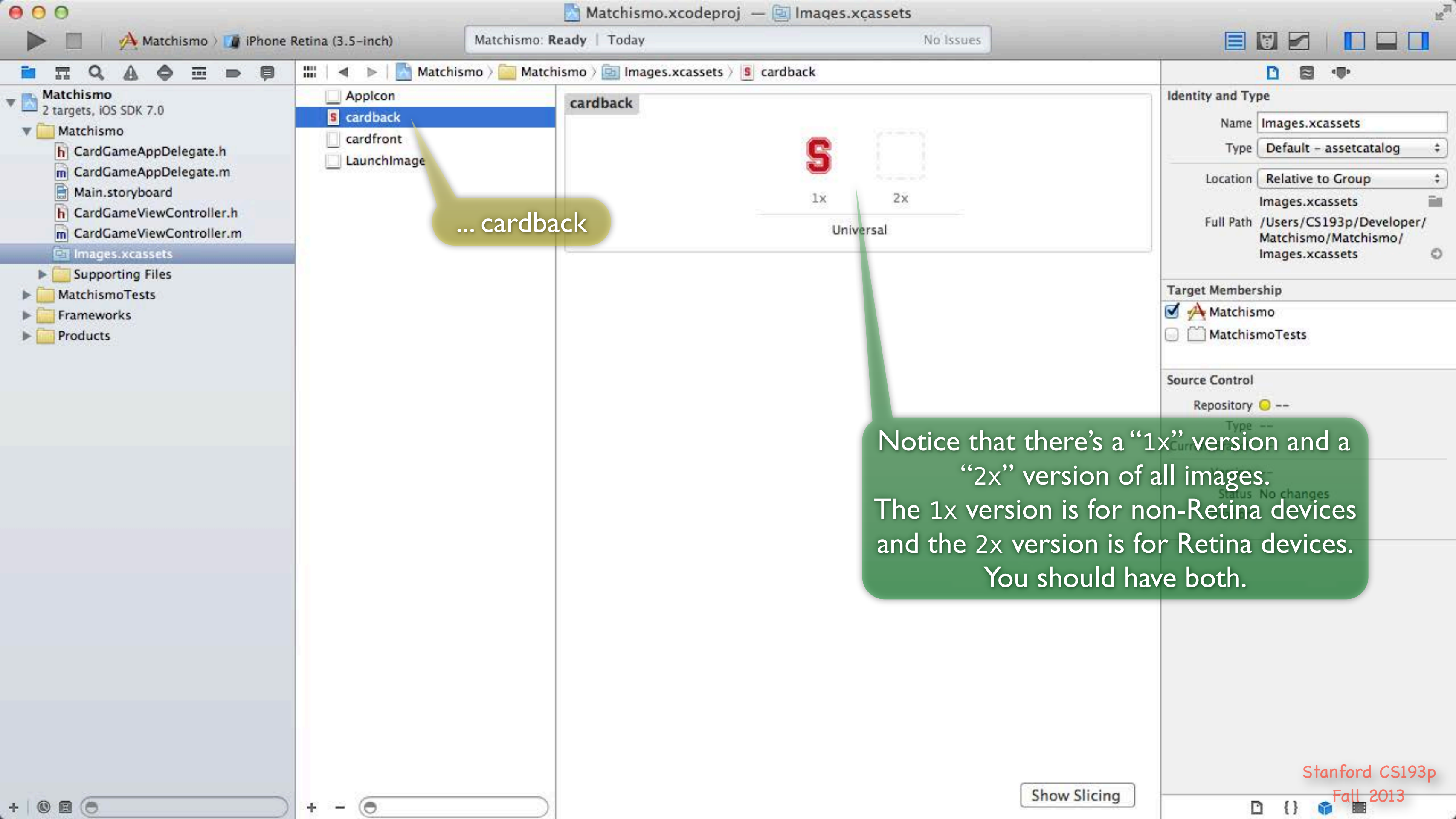


We'll use the name "cardfront" to refer to the button's background when the card is "face up".



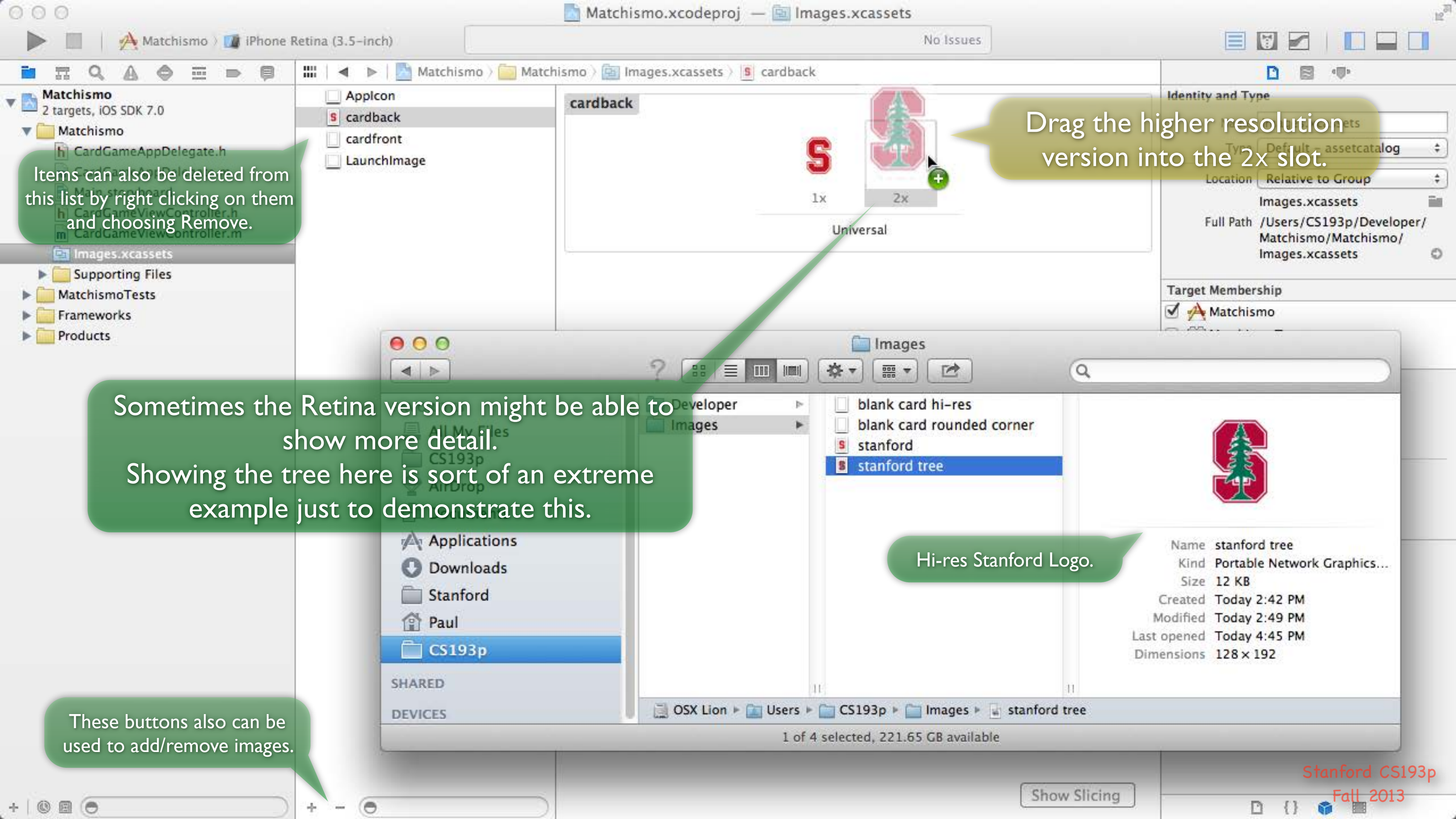
We'll change the name of the image on the back of the card from "stanford" to ...

Show Slicing



... cardback

Notice that there's a "1x" version and a "2x" version of all images. The 1x version is for non-Retina devices and the 2x version is for Retina devices. You should have both.



Items can also be deleted from this list by right clicking on them and choosing Remove.

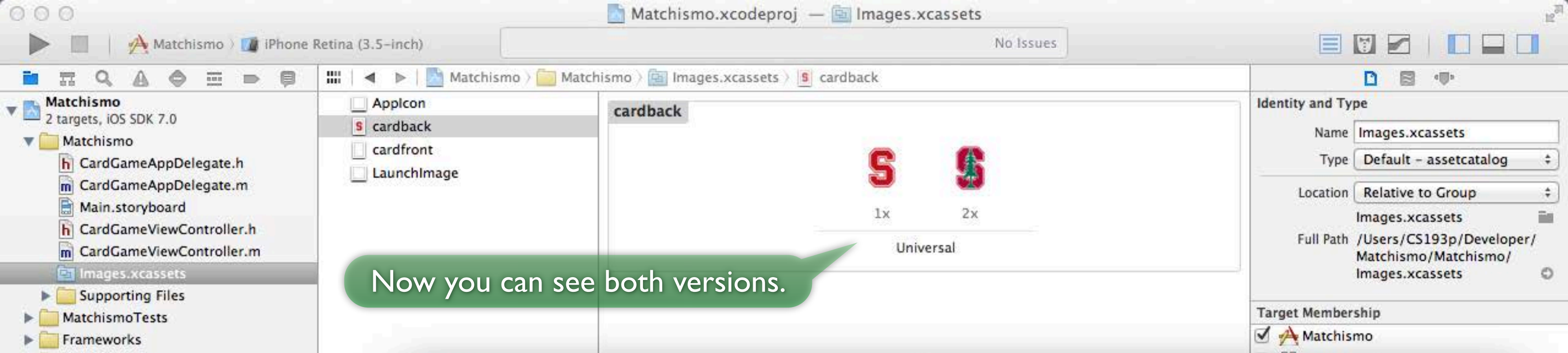
Drag the higher resolution version into the 2x slot.

Sometimes the Retina version might be able to show more detail. Showing the tree here is sort of an extreme example just to demonstrate this.

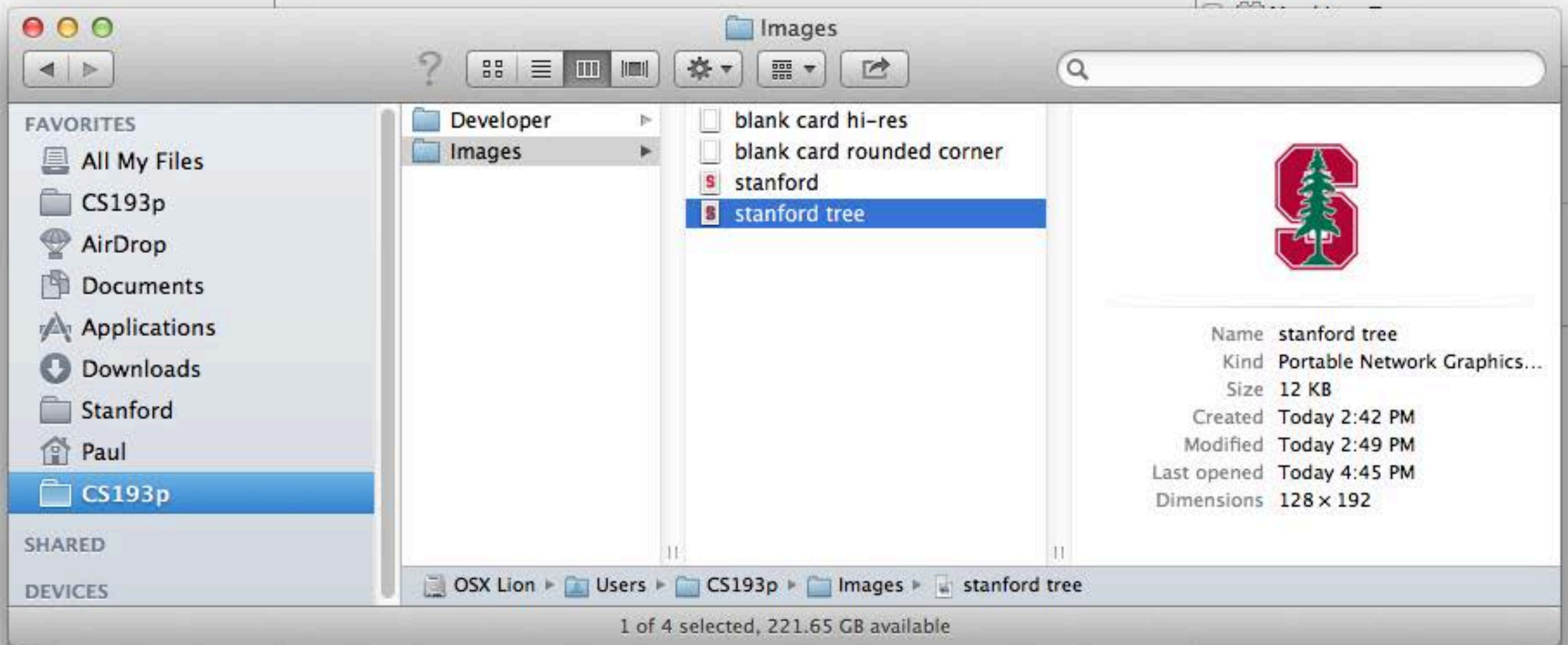
Hi-res Stanford Logo.

These buttons also can be used to add/remove images.

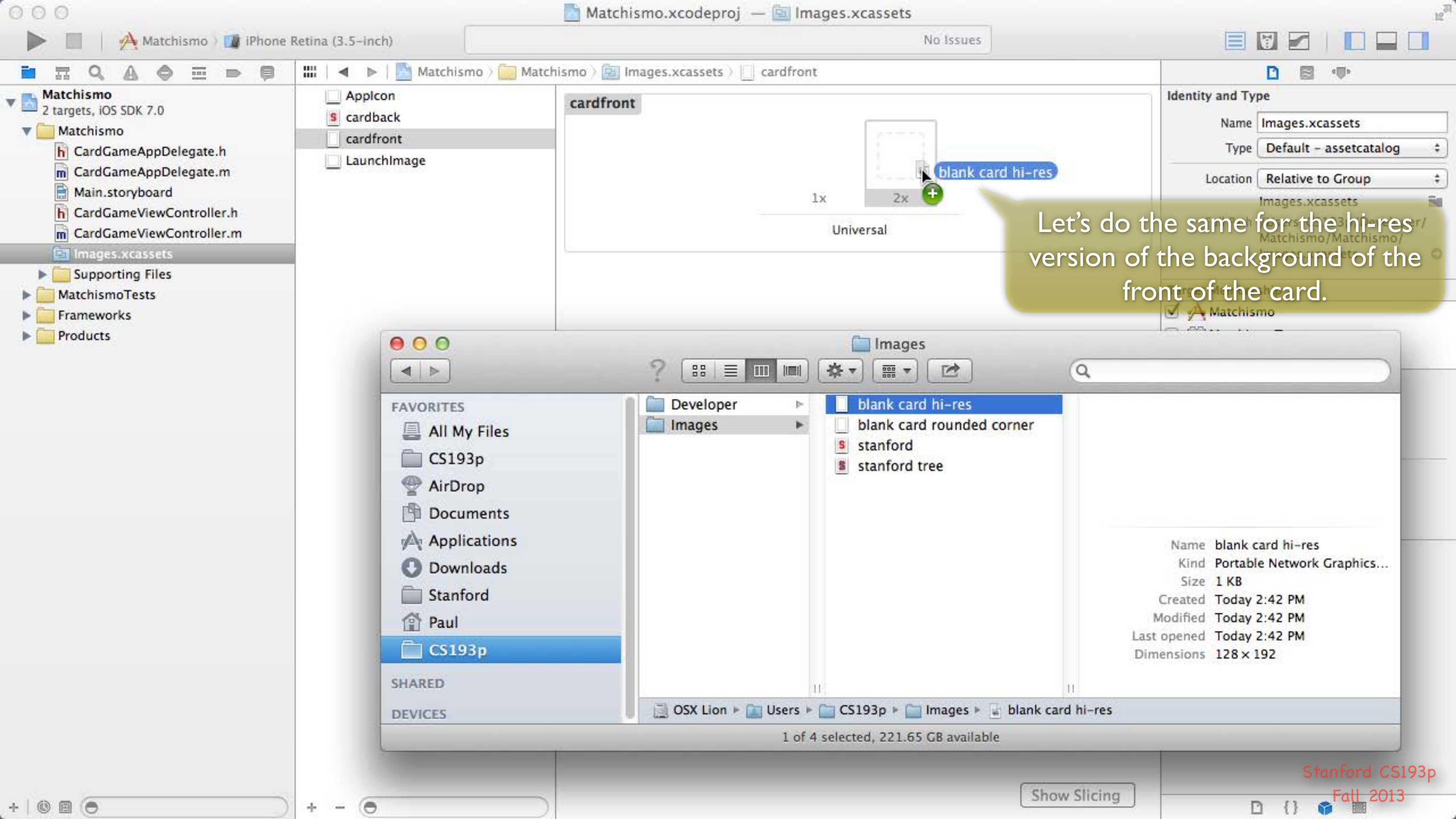
|             |                              |
|-------------|------------------------------|
| Name        | stanford tree                |
| Kind        | Portable Network Graphics... |
| Size        | 12 KB                        |
| Created     | Today 2:42 PM                |
| Modified    | Today 2:49 PM                |
| Last opened | Today 4:45 PM                |
| Dimensions  | 128 x 192                    |



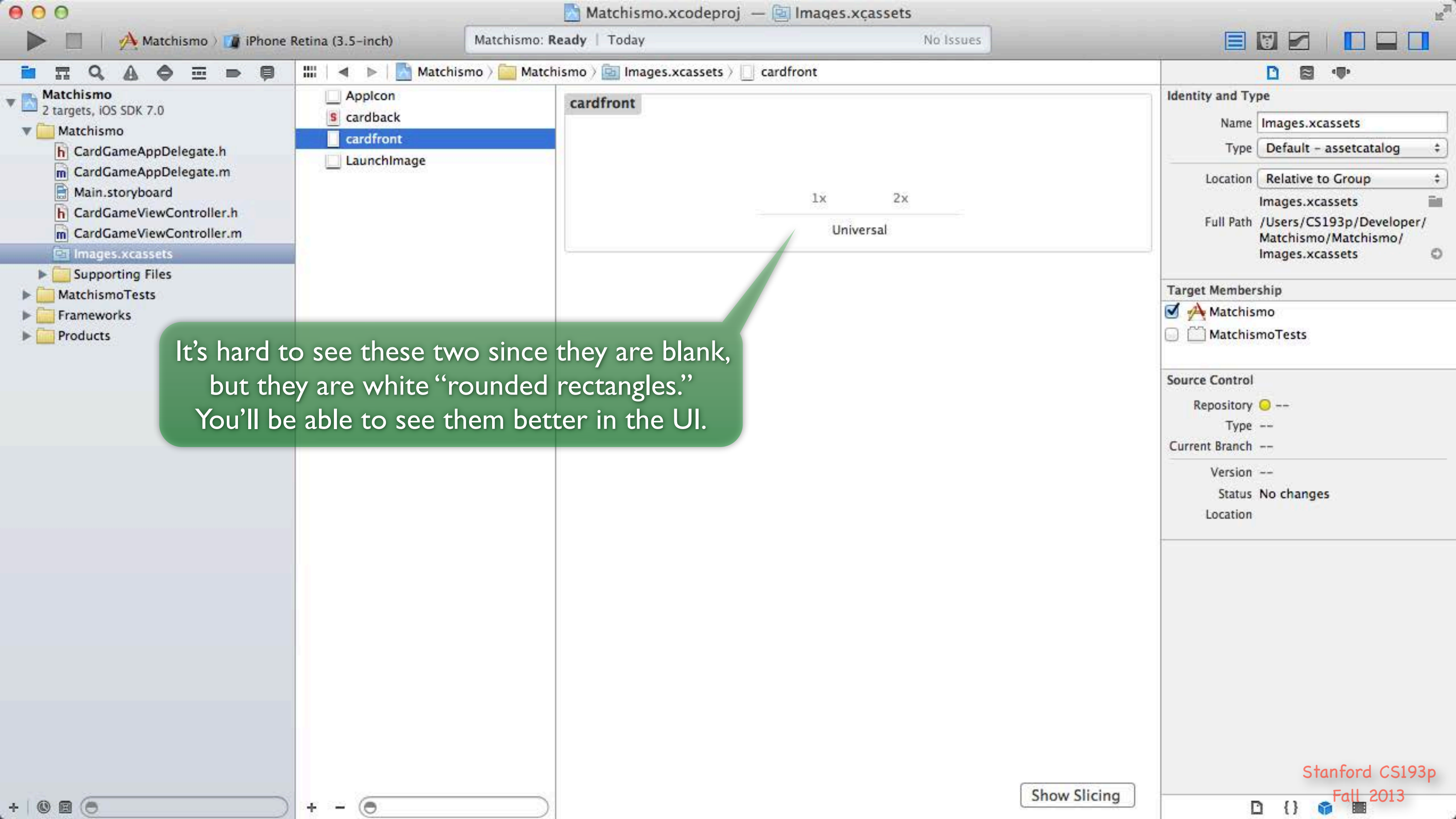
Now you can see both versions.



Show Slicing

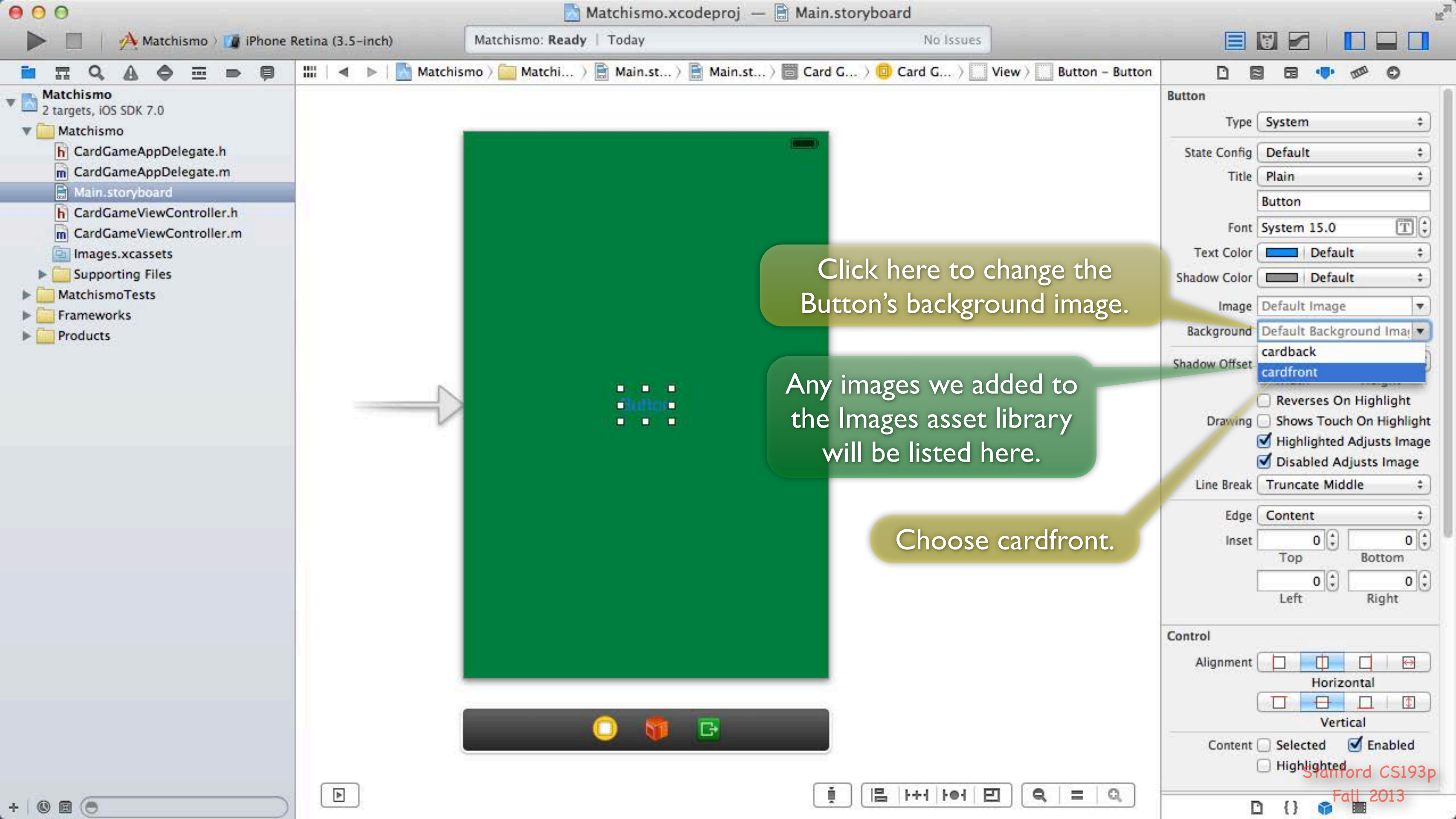


Let's do the same for the hi-res version of the background of the front of the card.



It's hard to see these two since they are blank, but they are white "rounded rectangles." You'll be able to see them better in the UI.

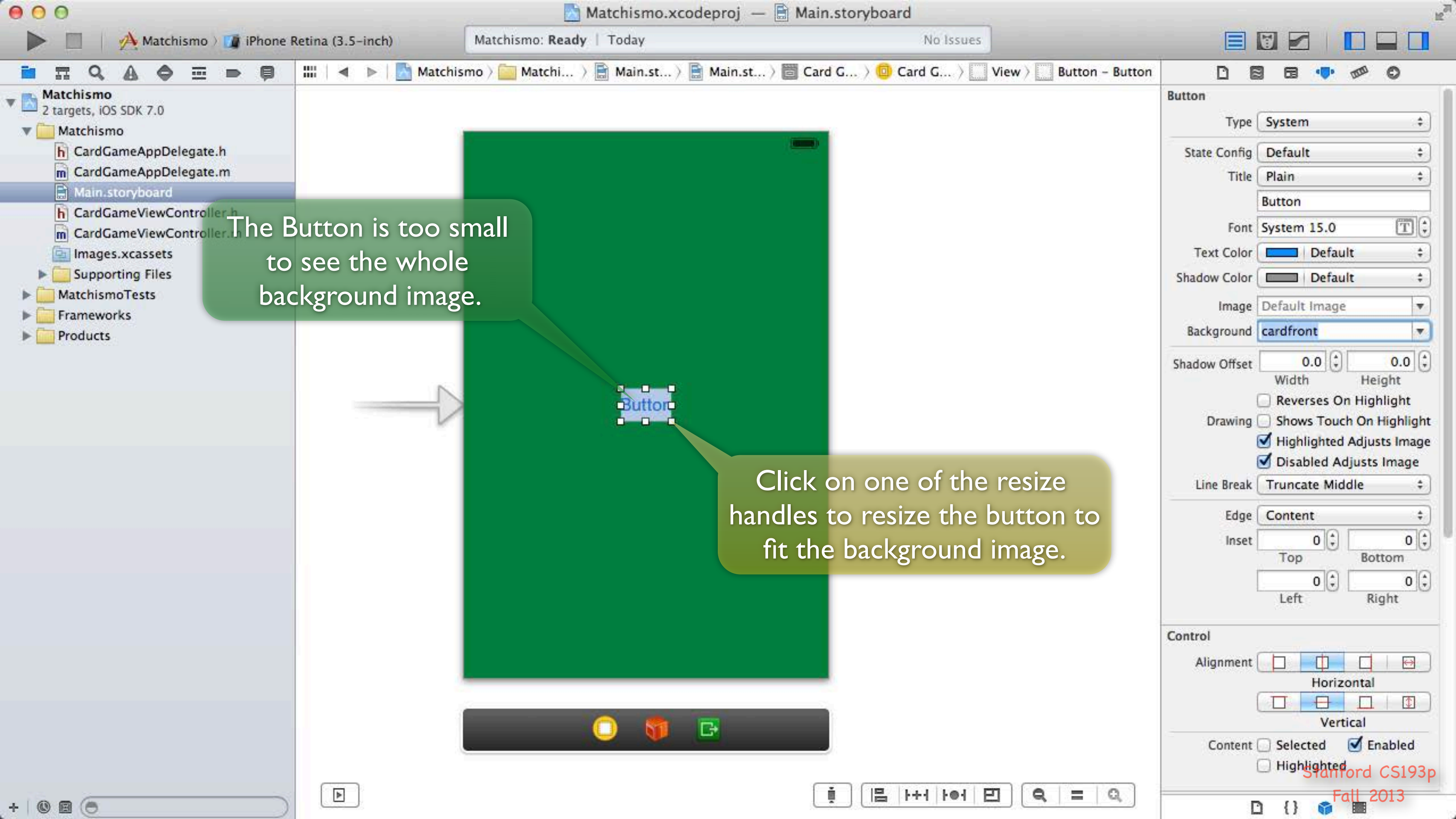




Click here to change the Button's background image.

Any images we added to the Images asset library will be listed here.

Choose cardfront.



The Button is too small to see the whole background image.

Click on one of the resize handles to resize the button to fit the background image.

**Button**

Type: System

State Config: Default

Title: Plain

Button

Font: System 15.0

Text Color: Default

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width), 0.0 (Height)

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break: Truncate Middle

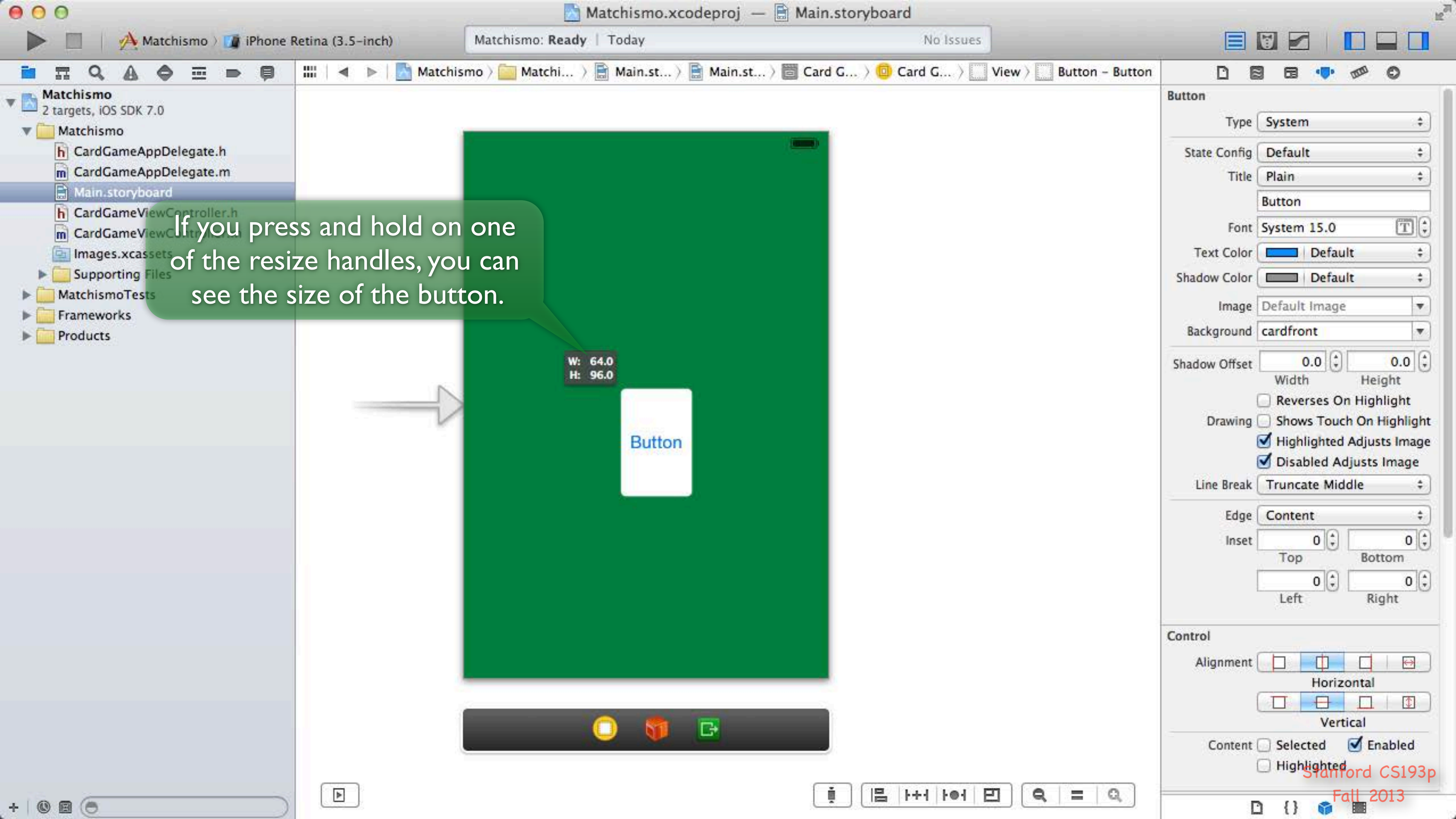
Edge: Content

Inset: 0 (Top), 0 (Bottom), 0 (Left), 0 (Right)

**Control**

Alignment: Horizontal, Vertical

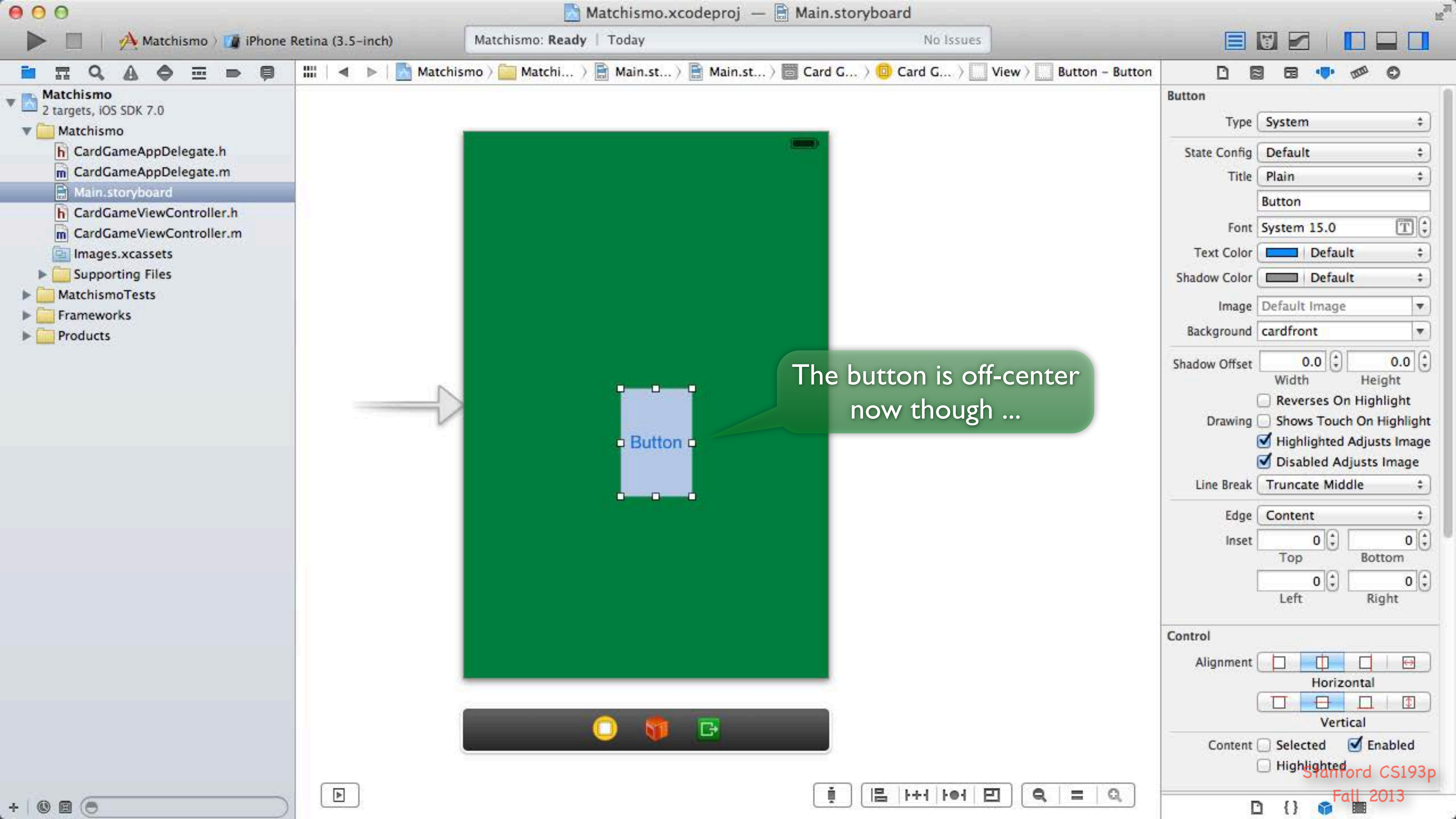
Content  Selected  Enabled  Highlighted



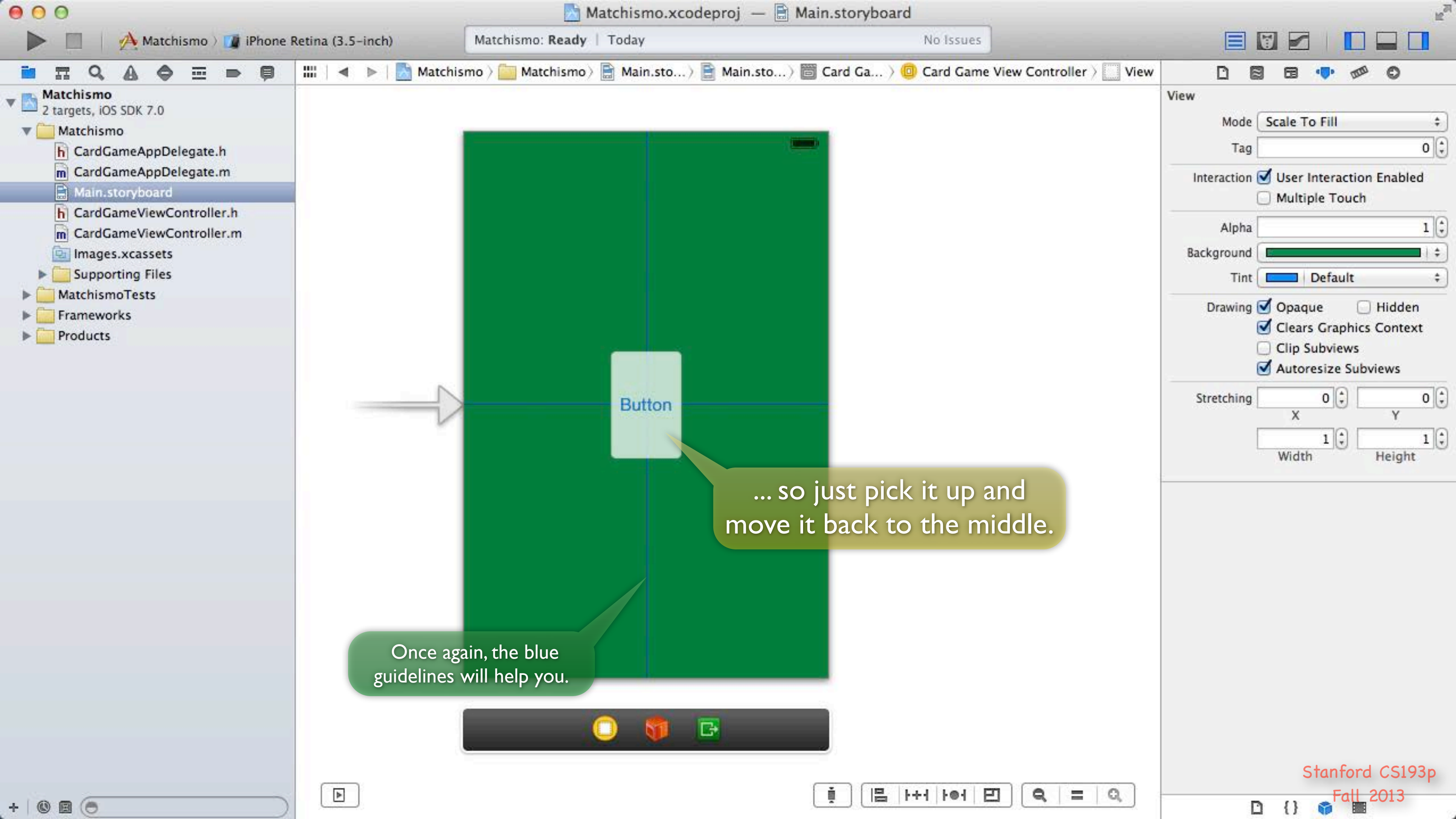
If you press and hold on one of the resize handles, you can see the size of the button.

W: 64.0  
H: 96.0

Button

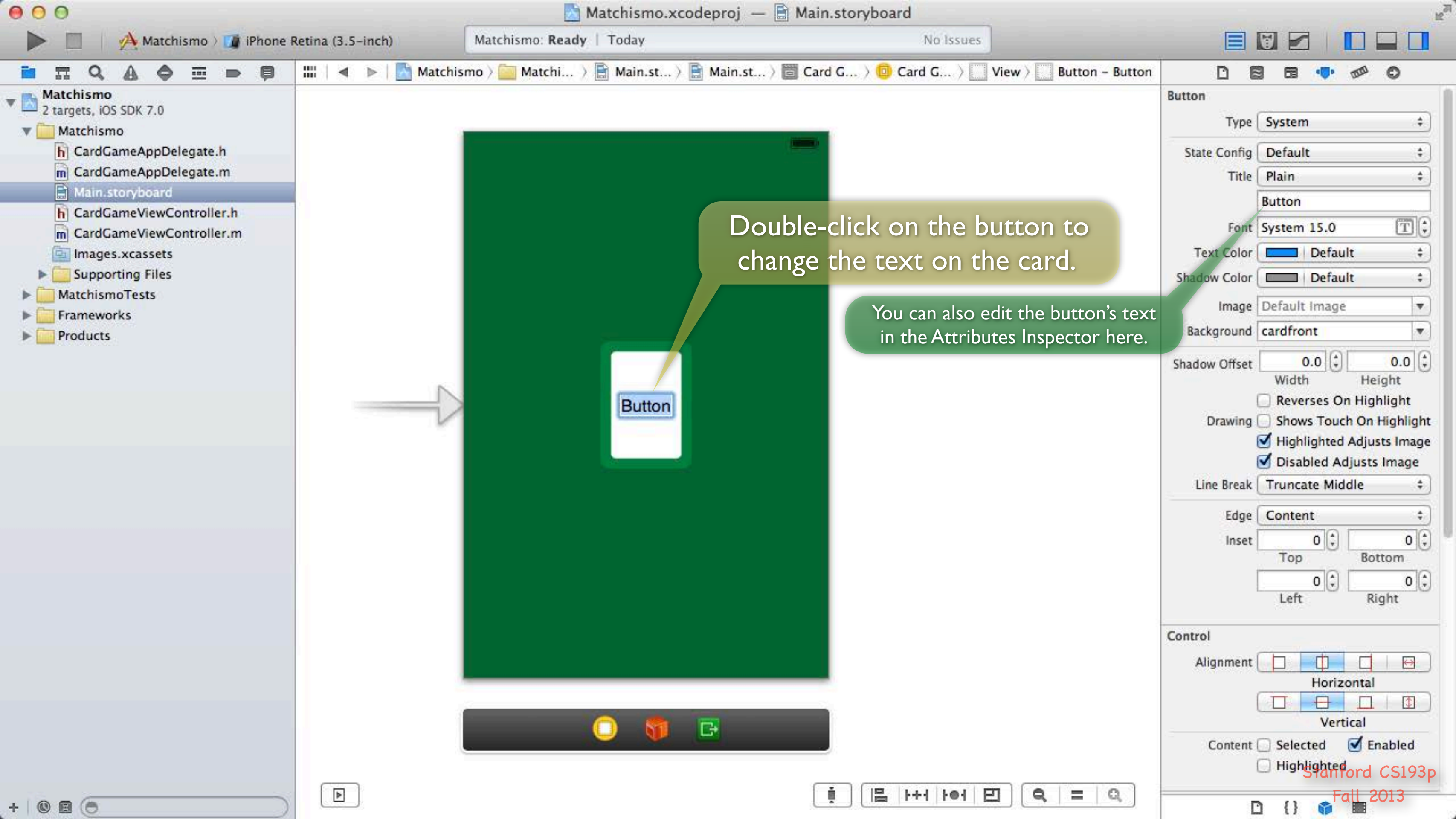


The button is off-center now though ...



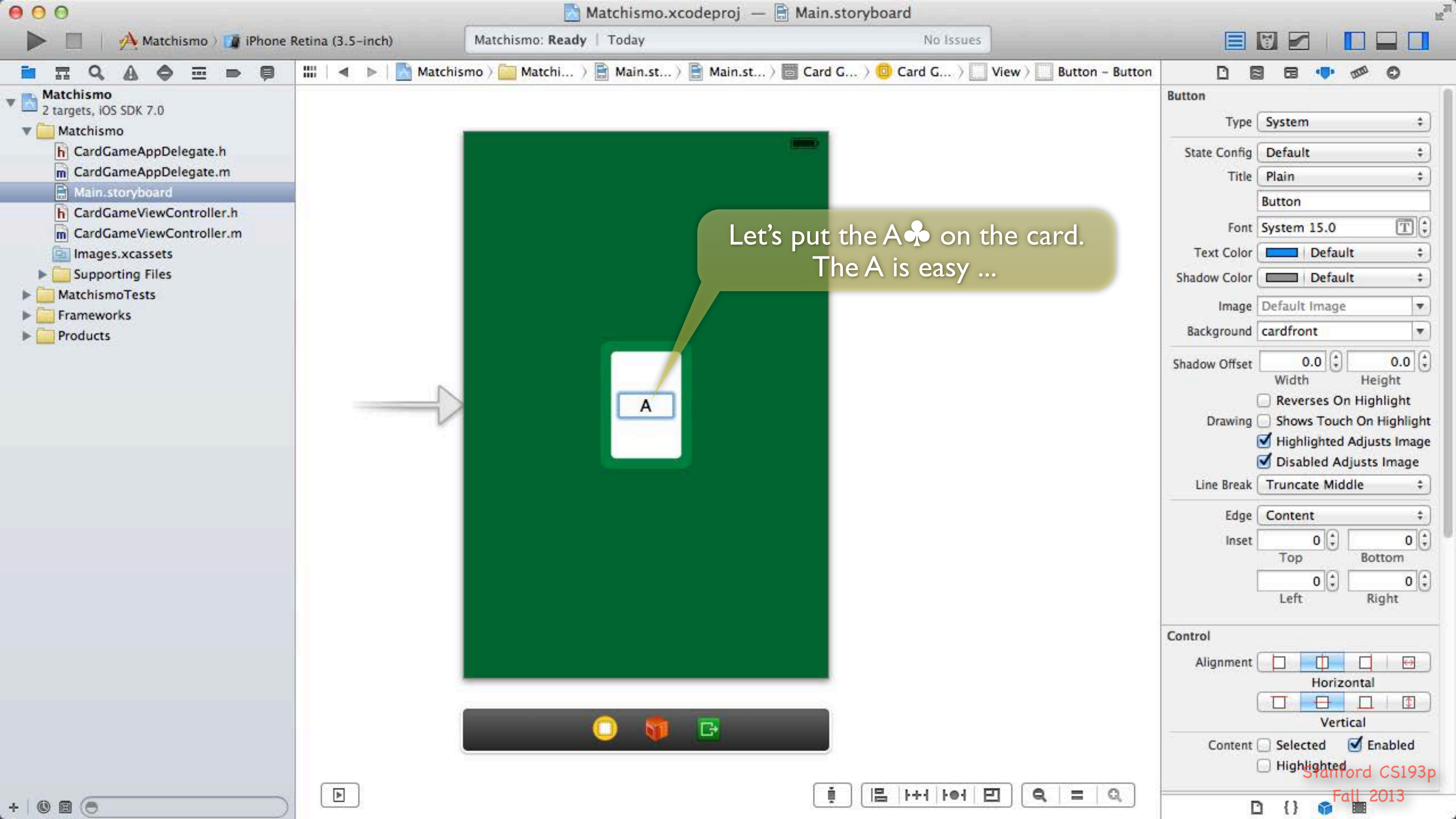
... so just pick it up and move it back to the middle.

Once again, the blue guidelines will help you.



Double-click on the button to change the text on the card.

You can also edit the button's text in the Attributes Inspector here.



Let's put the A♣ on the card.  
The A is easy ...

**Button**

Type: System

State Config: Default

Title: Plain

Button

Font: System 15.0

Text Color: Default

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width), 0.0 (Height)

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break: Truncate Middle

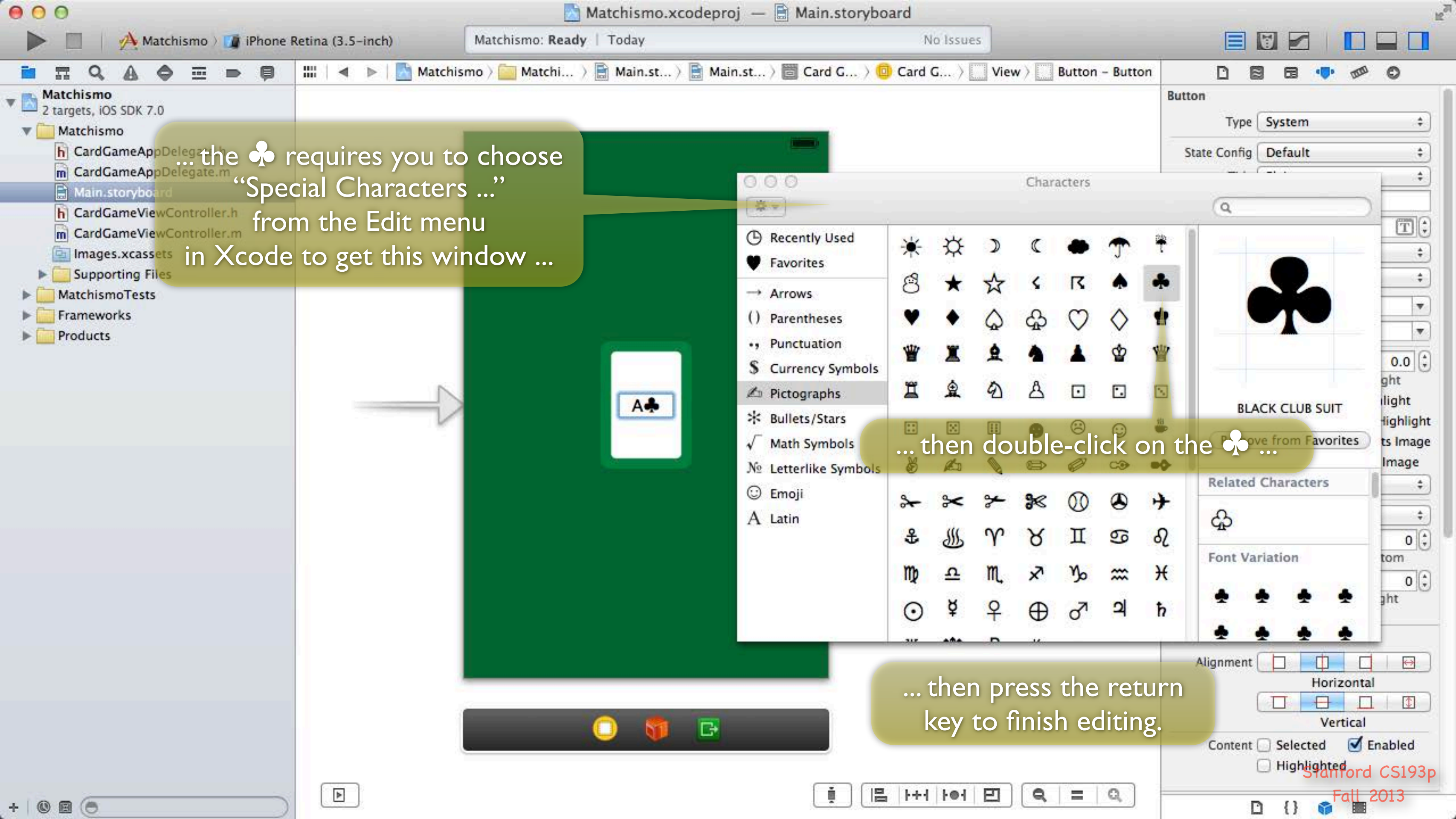
Edge: Content

Inset: 0 (Top), 0 (Bottom), 0 (Left), 0 (Right)

**Control**

Alignment: Horizontal, Vertical

Content  Selected  Enabled  Highlighted

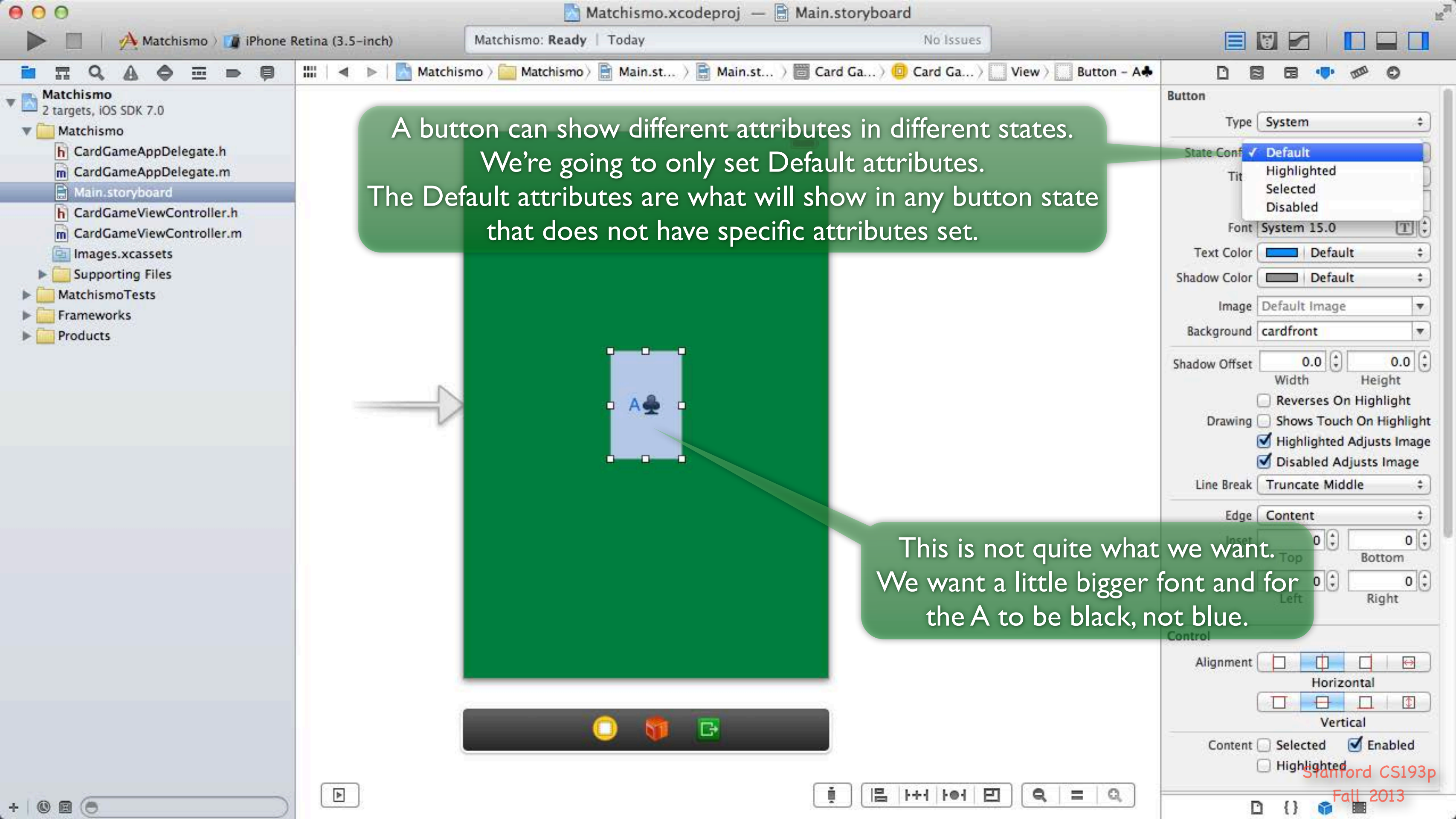


...the ♣ requires you to choose "Special Characters ..." from the Edit menu in Xcode to get this window ...

... then double-click on the ♣ ...

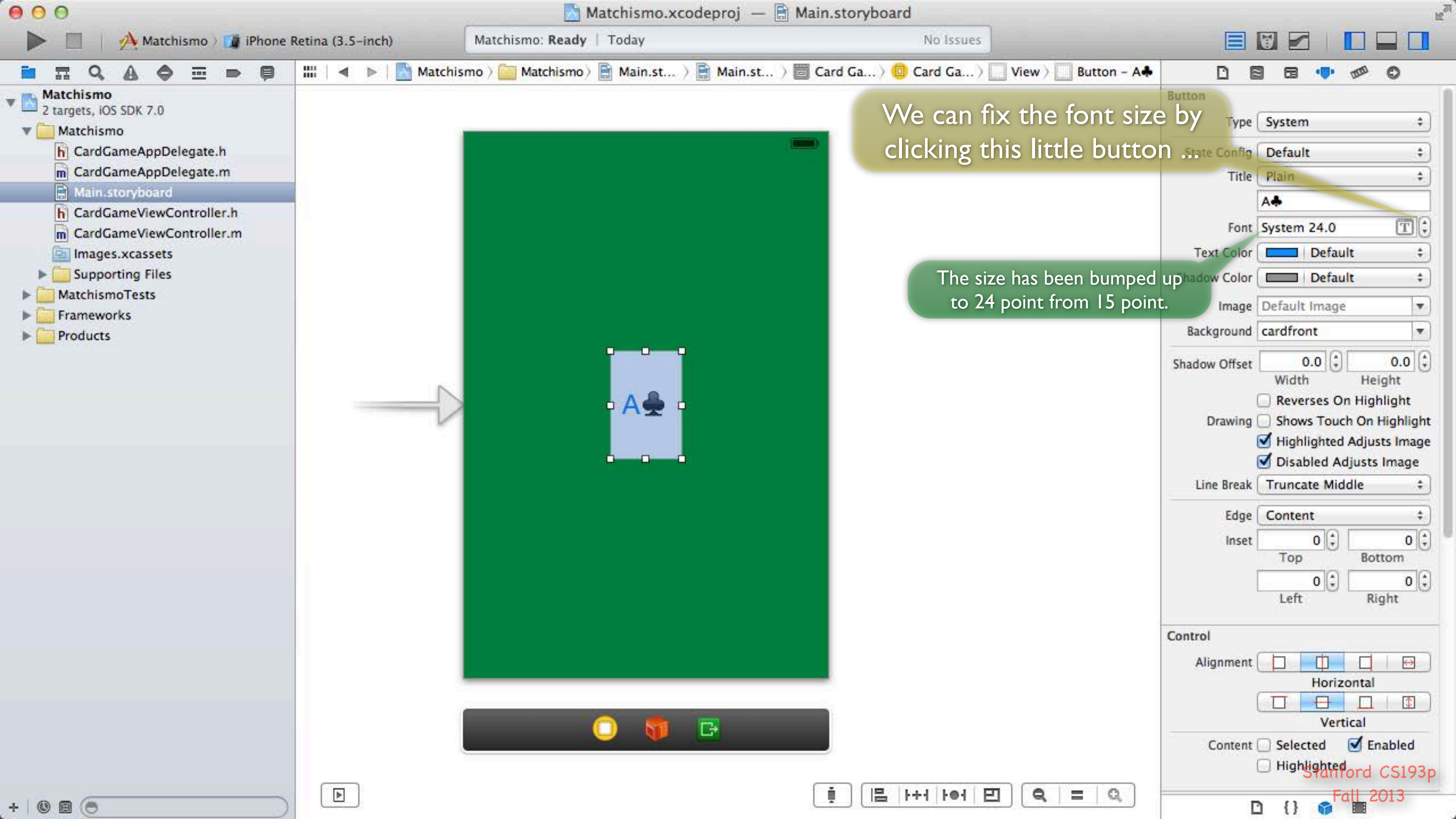
... then press the return key to finish editing.





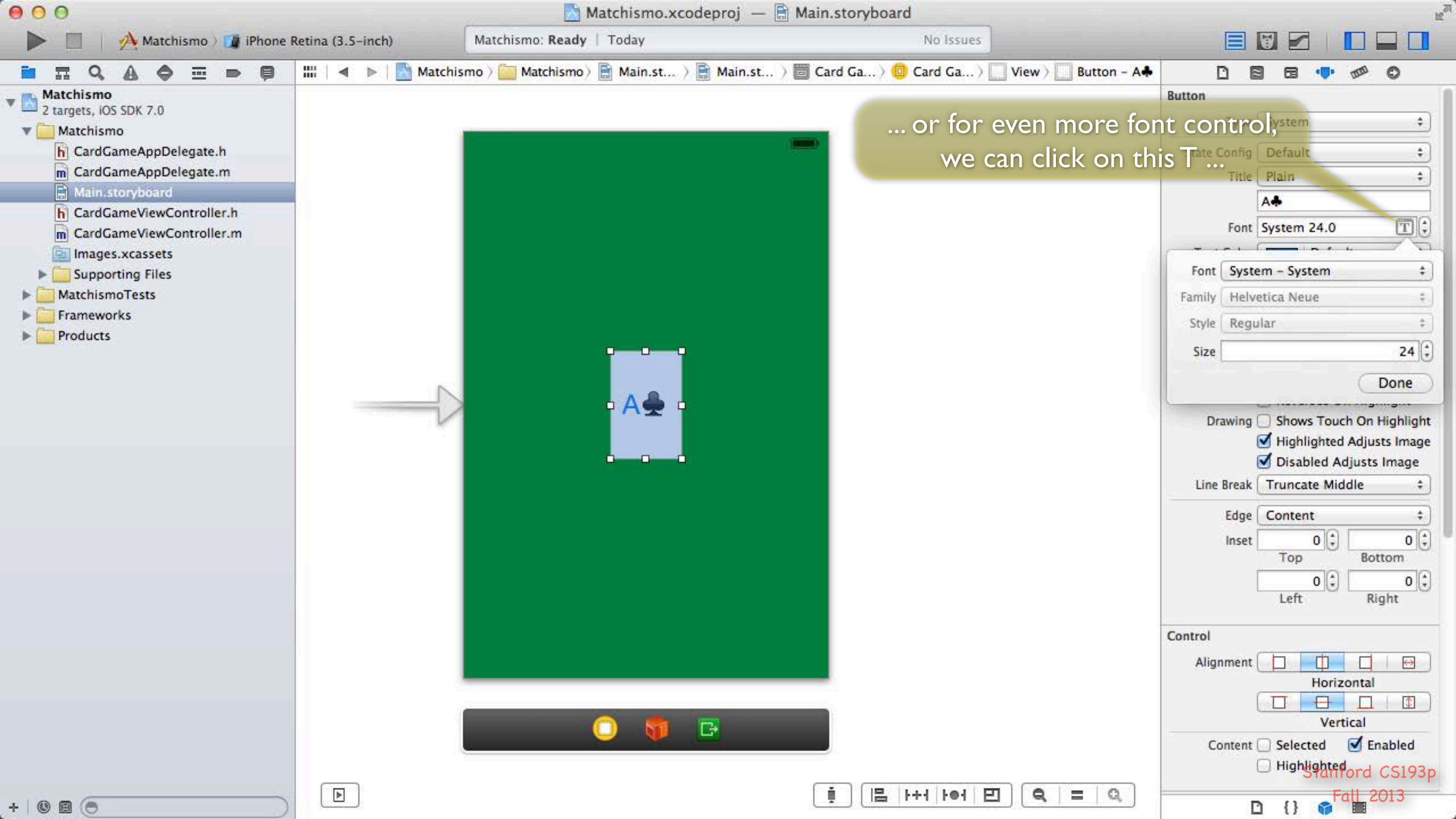
A button can show different attributes in different states. We're going to only set Default attributes. The Default attributes are what will show in any button state that does not have specific attributes set.

This is not quite what we want. We want a little bigger font and for the A to be black, not blue.



We can fix the font size by clicking this little button ...

The size has been bumped up to 24 point from 15 point.



... or for even more font control, we can click on this T ...

**Button**

Font: System - System

Family: Helvetica Neue

Style: Regular

Size: 24

Done

Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break: Truncate Middle

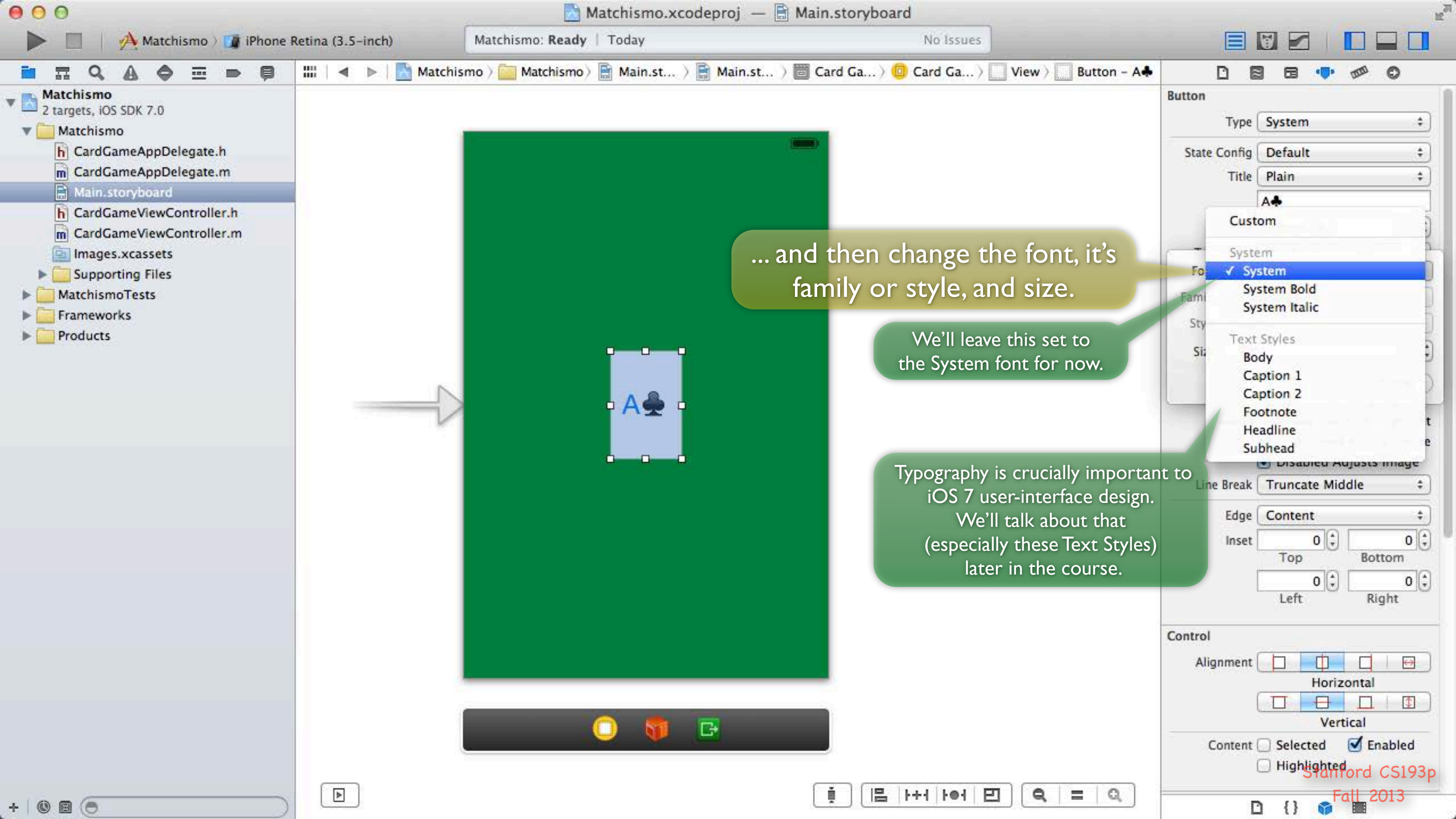
Edge: Content

Inset: Top: 0, Bottom: 0, Left: 0, Right: 0

**Control**

Alignment: Horizontal, Vertical

Content:  Selected,  Enabled,  Highlighted



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products

... and then change the font, it's family or style, and size.

We'll leave this set to the System font for now.

Typography is crucially important to iOS 7 user-interface design. We'll talk about that (especially these Text Styles) later in the course.

Button

Type System

State Config Default

Title Plain

A♣

Custom

System

System

System Bold

System Italic

Text Styles

Body

Caption 1

Caption 2

Footnote

Headline

Subhead

Line Break Truncate Middle

Edge Content

Inset 0 0 0 0

Top Bottom

Left Right

Control

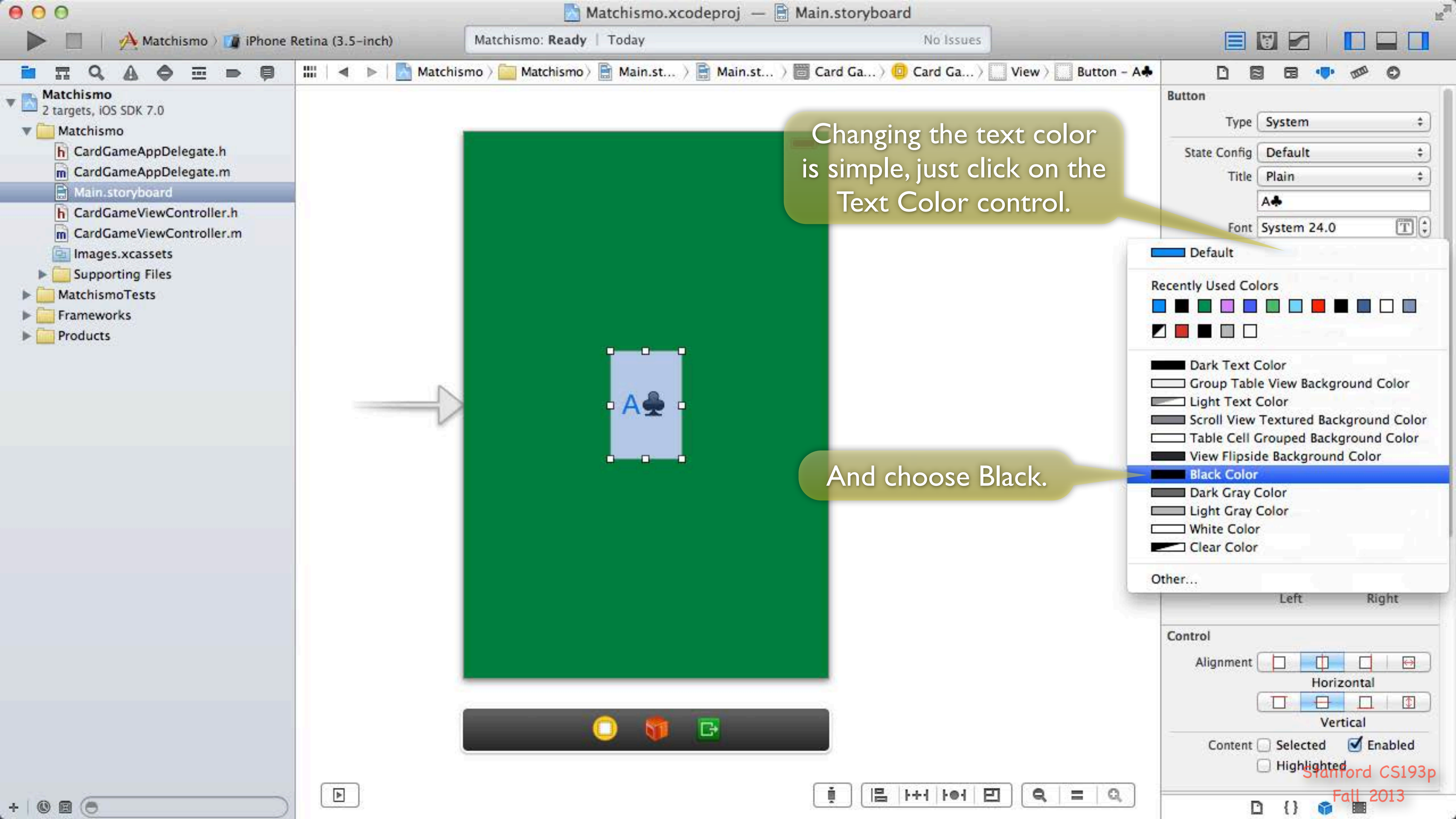
Alignment

Horizontal

Vertical

Content Selected Enabled

Highlighted



Changing the text color is simple, just click on the Text Color control.

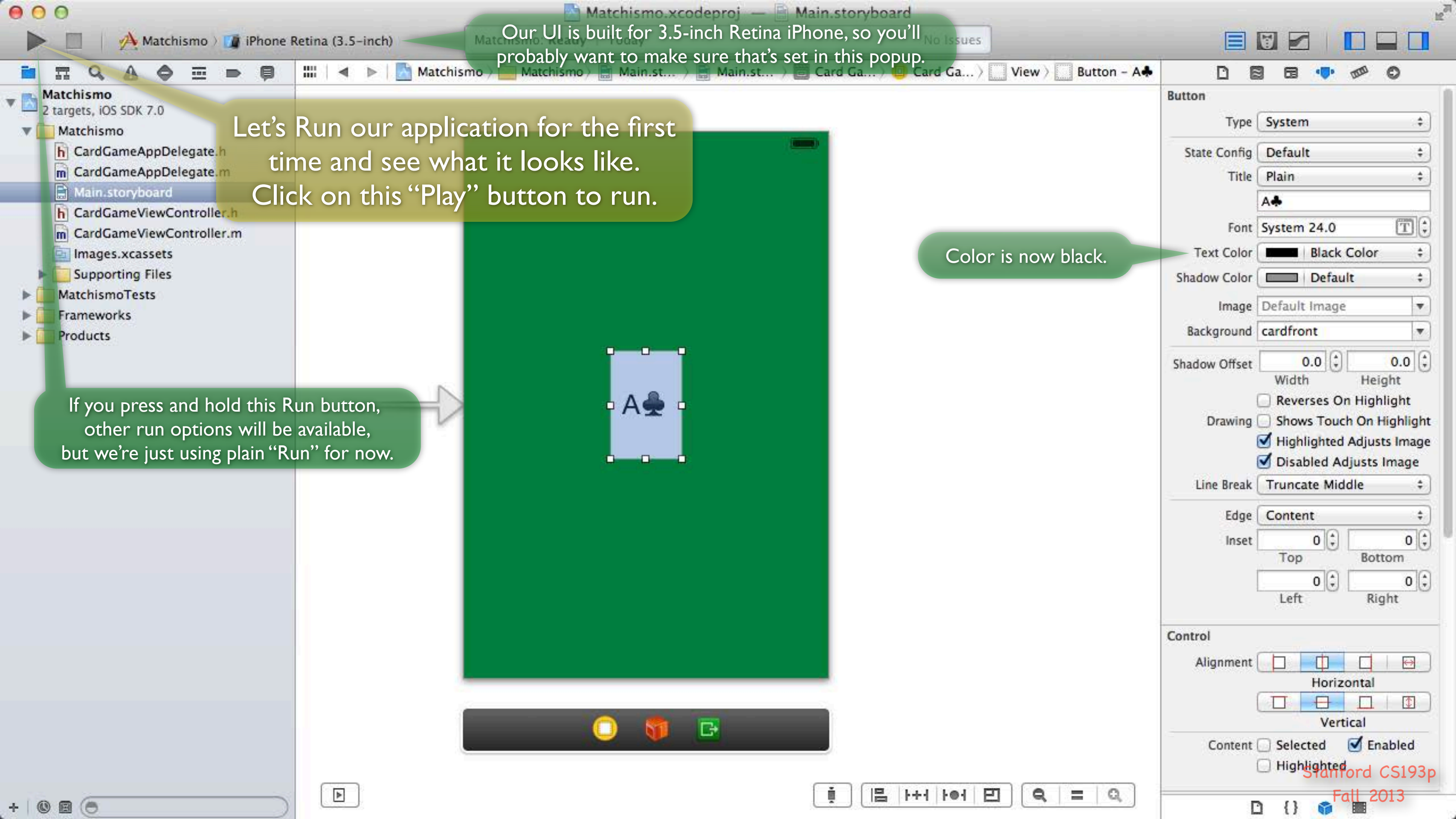
And choose Black.

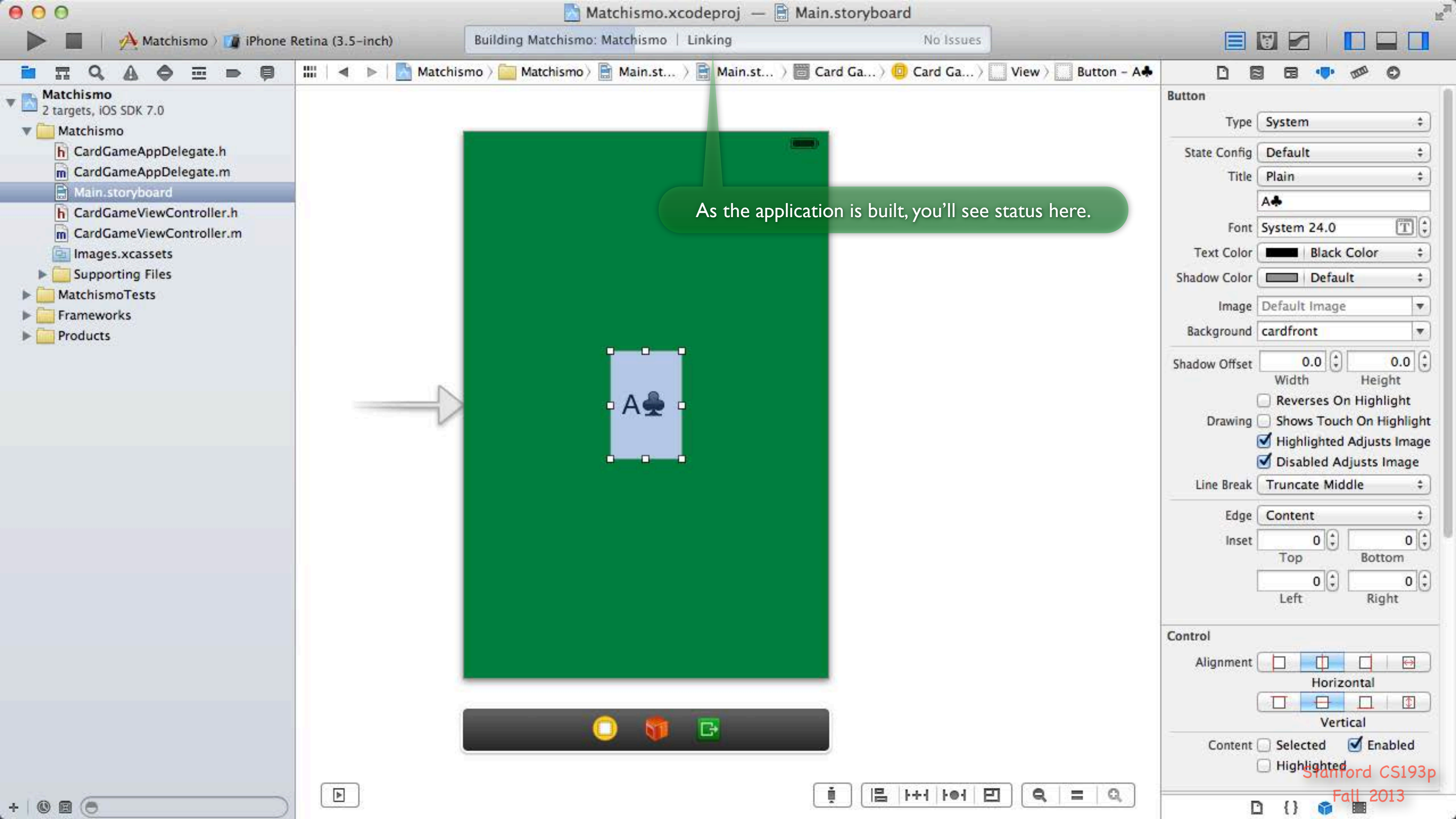
Our UI is built for 3.5-inch Retina iPhone, so you'll probably want to make sure that's set in this popup.

Let's Run our application for the first time and see what it looks like. Click on this "Play" button to run.

If you press and hold this Run button, other run options will be available, but we're just using plain "Run" for now.

Color is now black.





Building Matchismo: Matchismo | Linking No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

As the application is built, you'll see status here.

**Button**

Type System

State Config Default

Title Plain

A♣

Font System 24.0

Text Color Black Color

Shadow Color Default

Image Default Image

Background cardfront

Shadow Offset 0.0 0.0

Width Height

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break Truncate Middle

Edge Content

Inset 0 0

Top Bottom

0 0

Left Right

**Control**

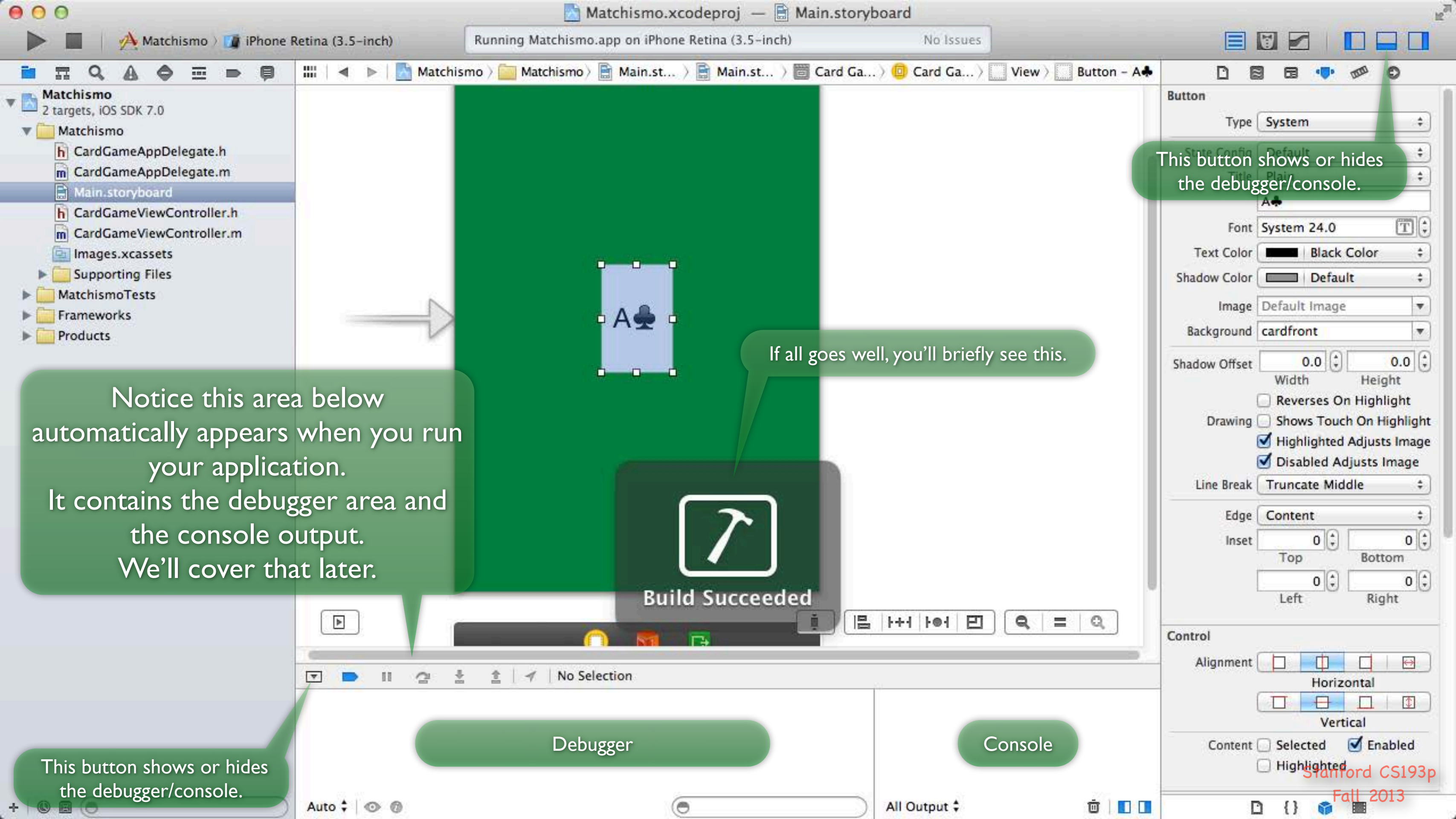
Alignment

Horizontal

Vertical

Content  Selected  Enabled

Highlighted



Notice this area below automatically appears when you run your application. It contains the debugger area and the console output. We'll cover that later.

This button shows or hides the debugger/console.

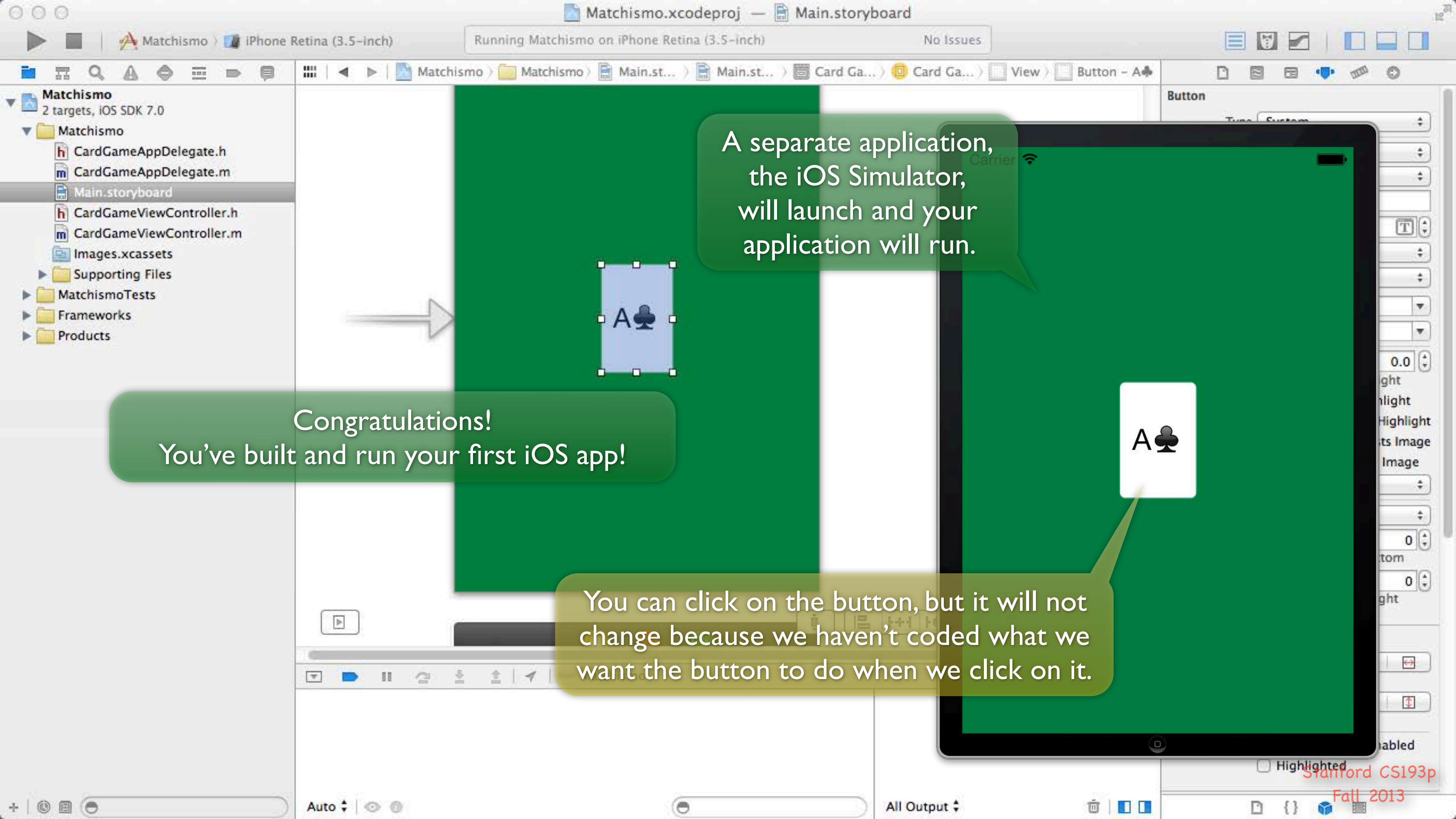
If all goes well, you'll briefly see this.

This button shows or hides the debugger/console.

Debugger

Console

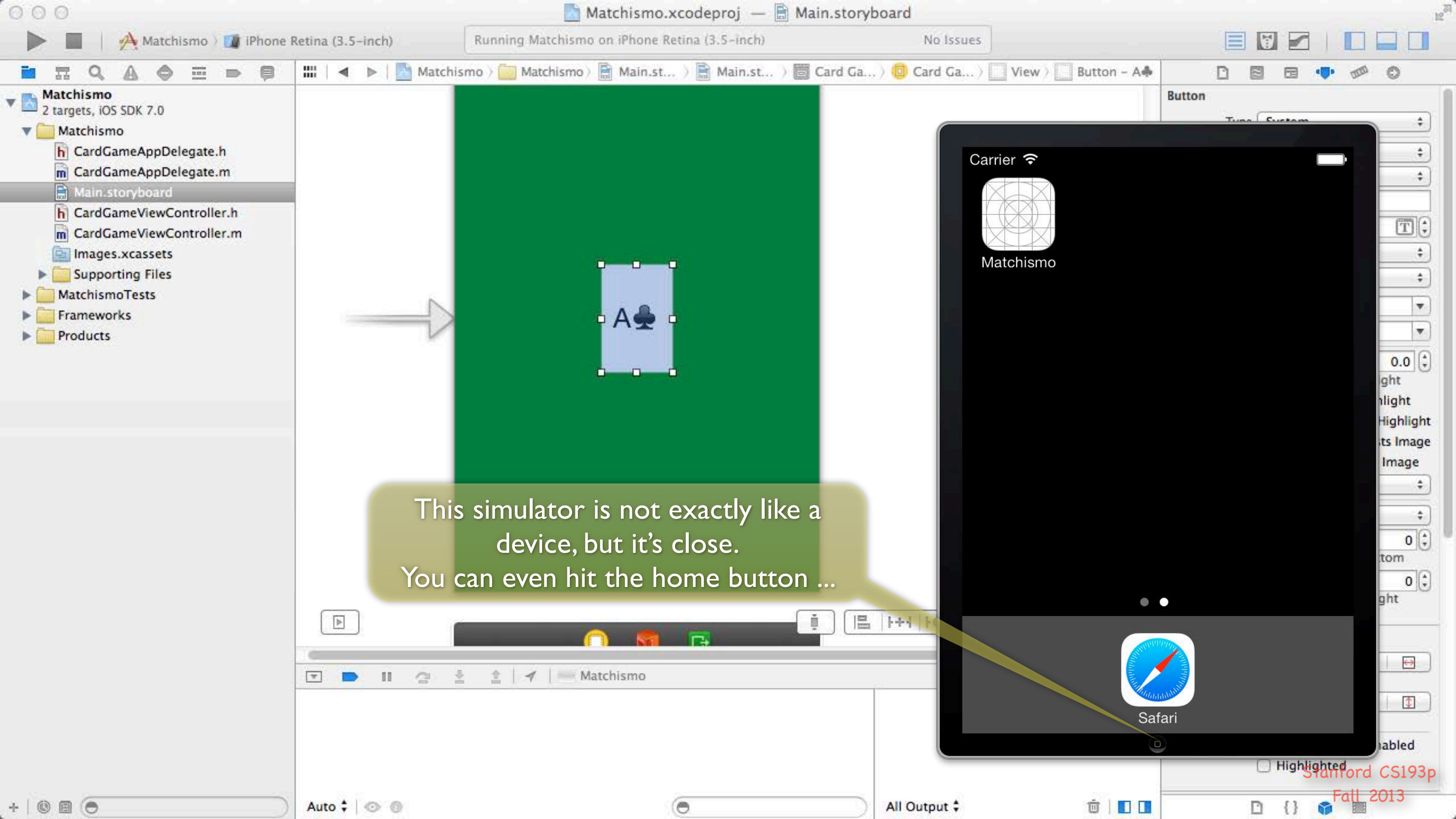




A separate application, the iOS Simulator, will launch and your application will run.

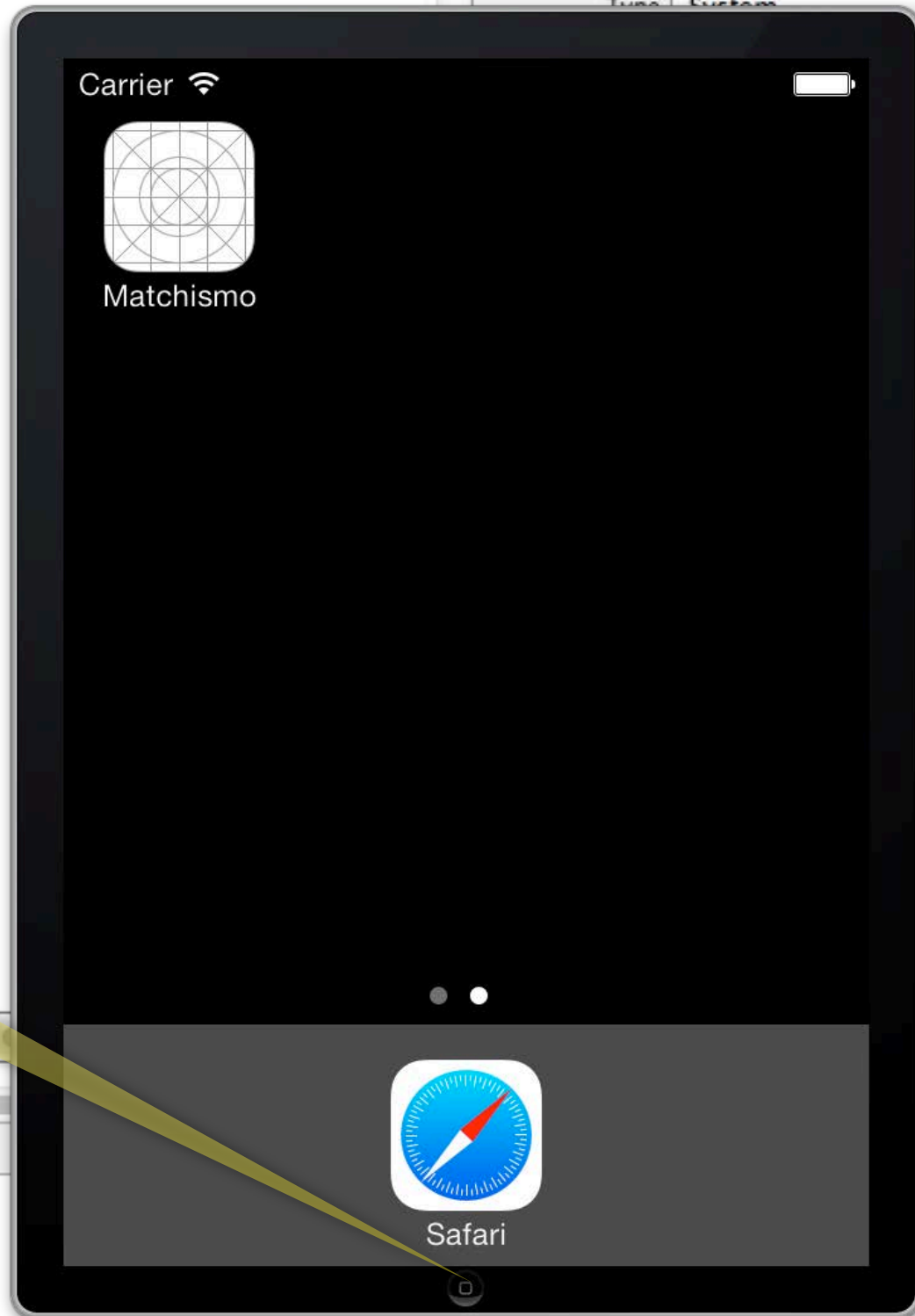
Congratulations!  
You've built and run your first iOS app!

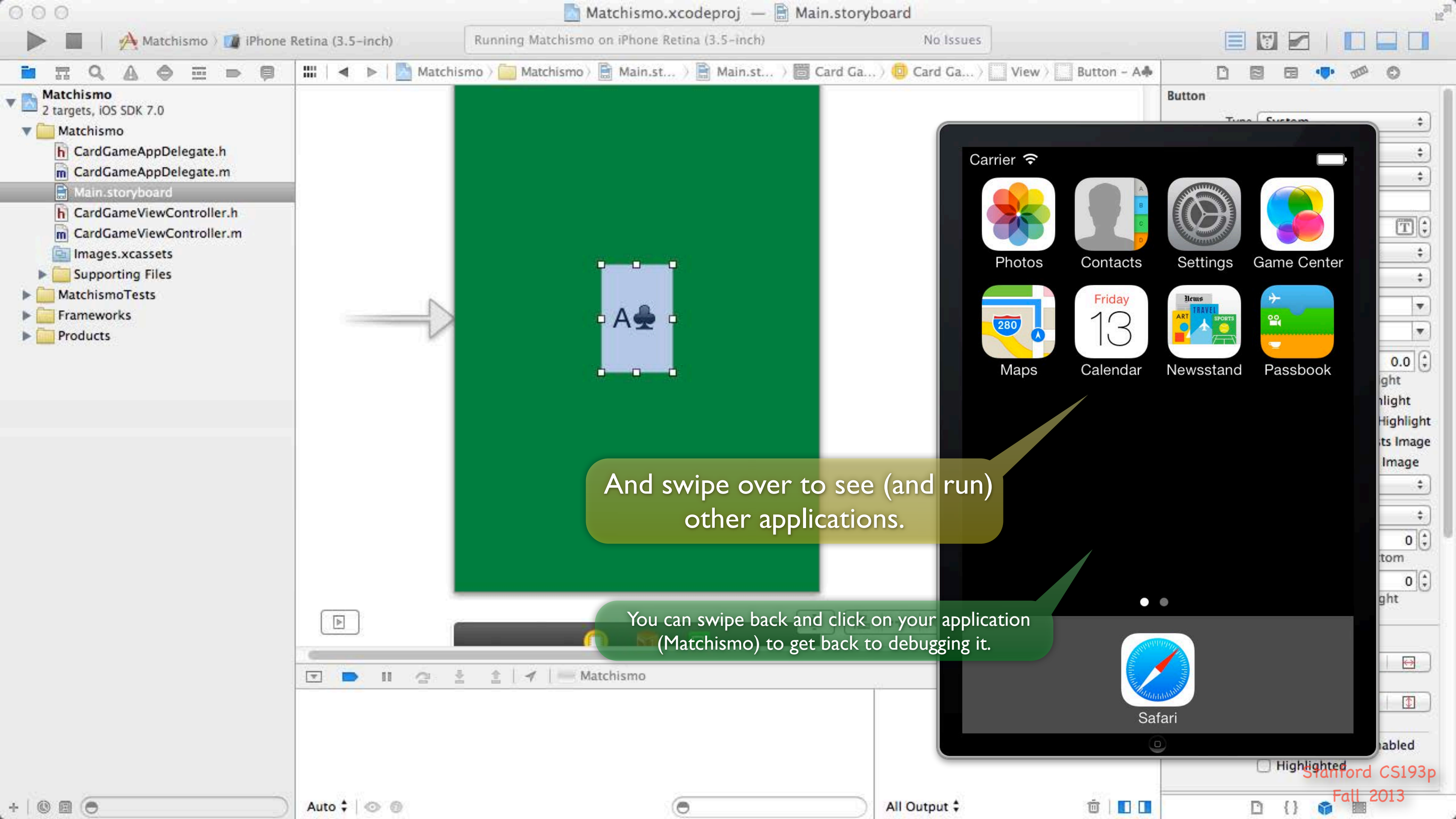
You can click on the button, but it will not change because we haven't coded what we want the button to do when we click on it.



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

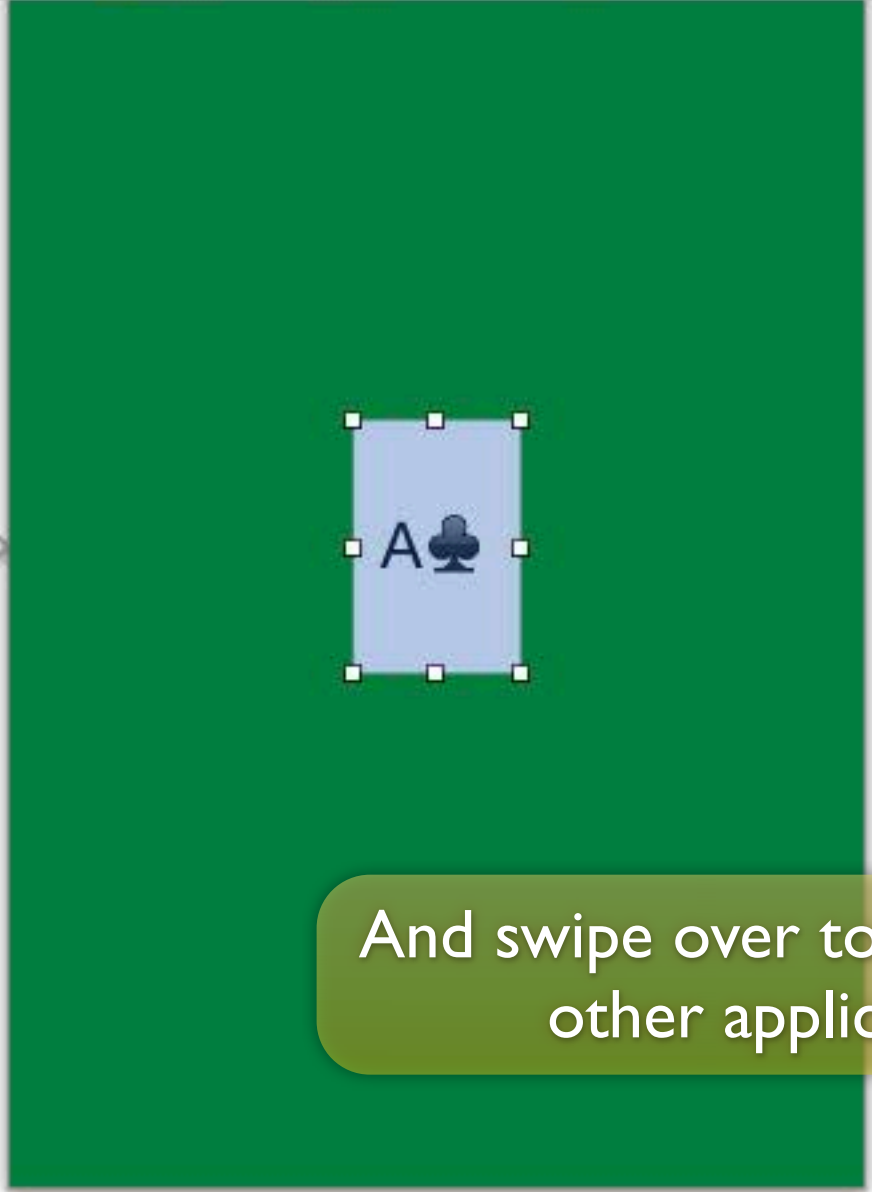
This simulator is not exactly like a device, but it's close. You can even hit the home button ...





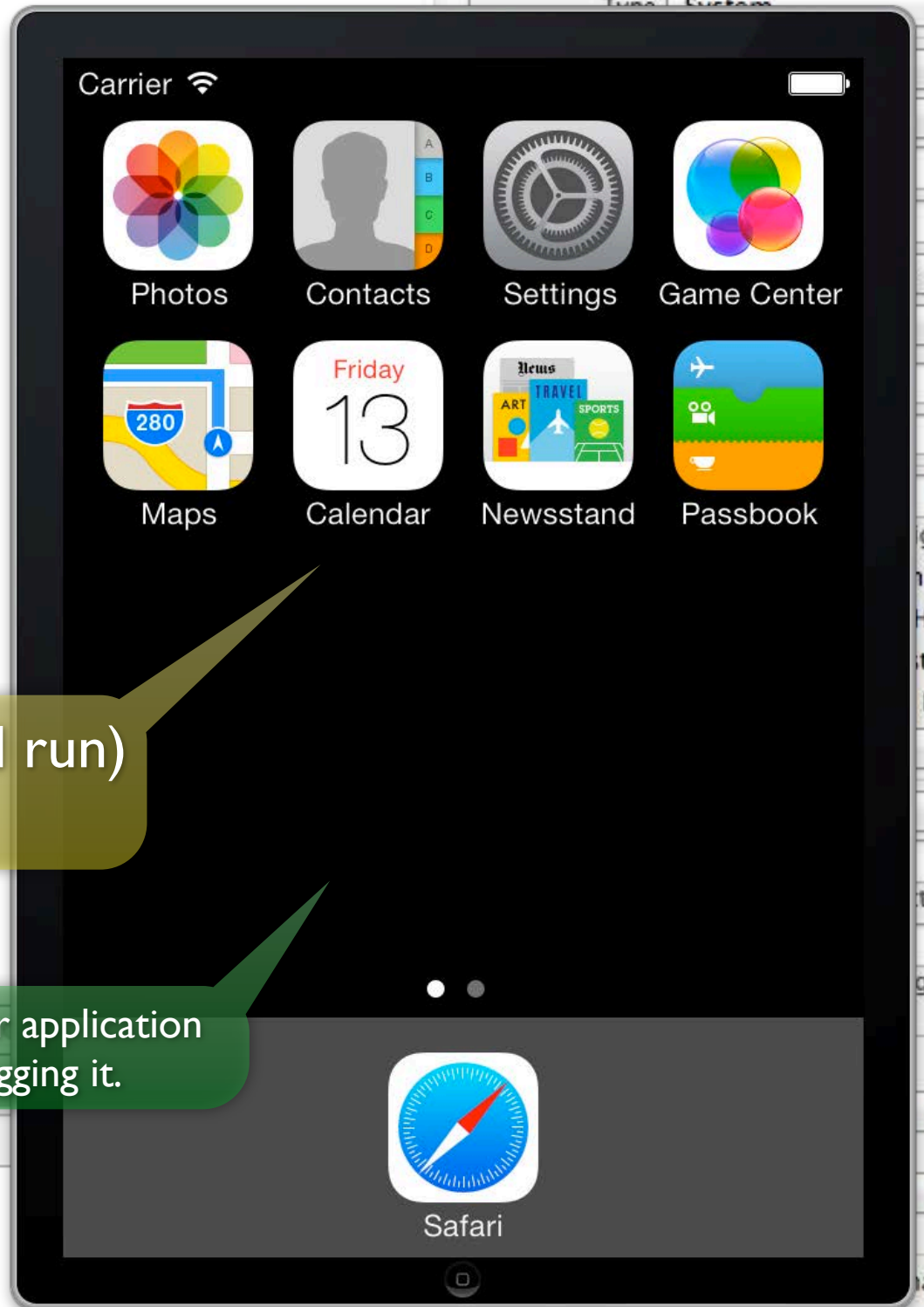
Running Matchismo on iPhone Retina (3.5-inch) No Issues

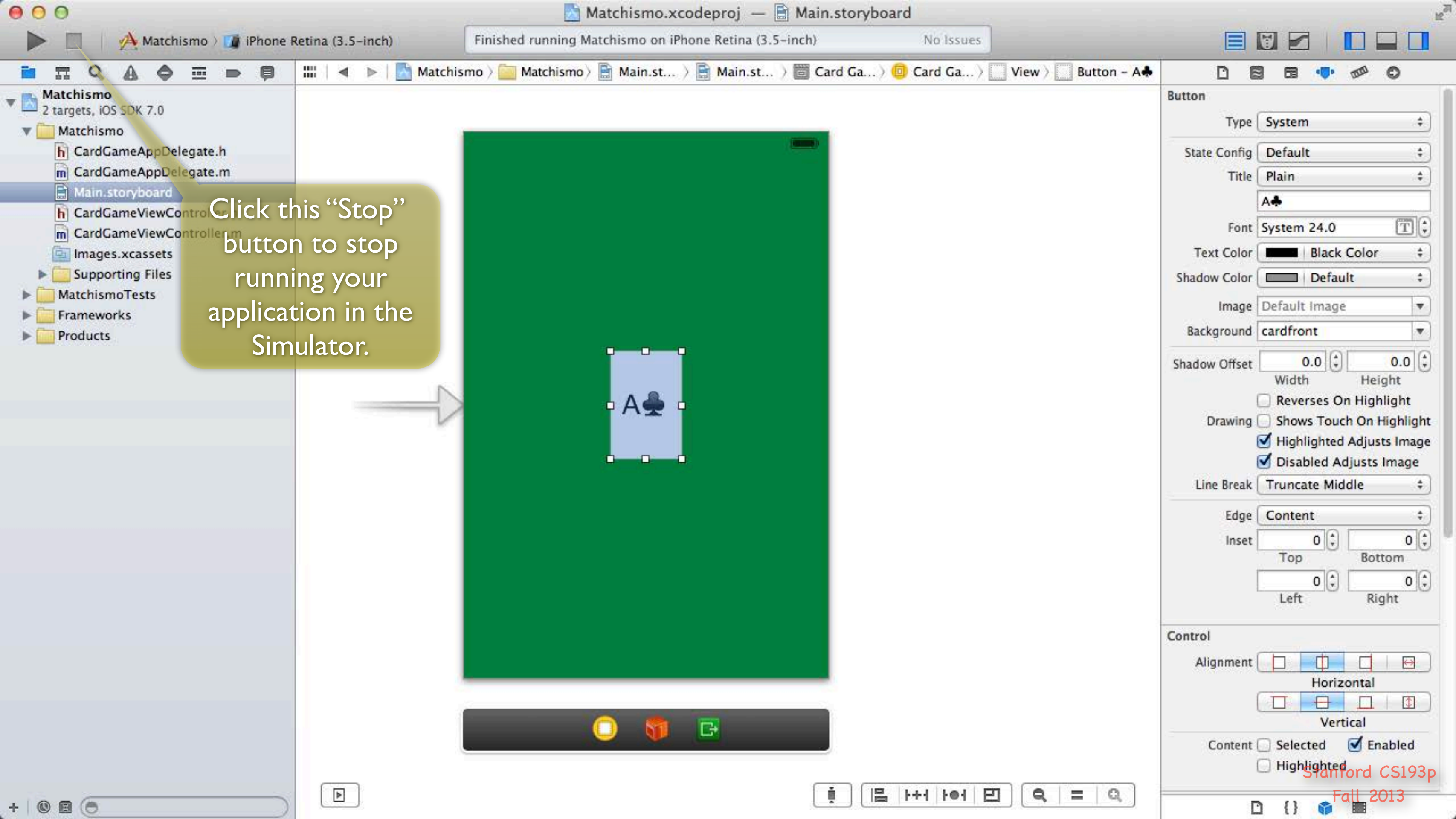
- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



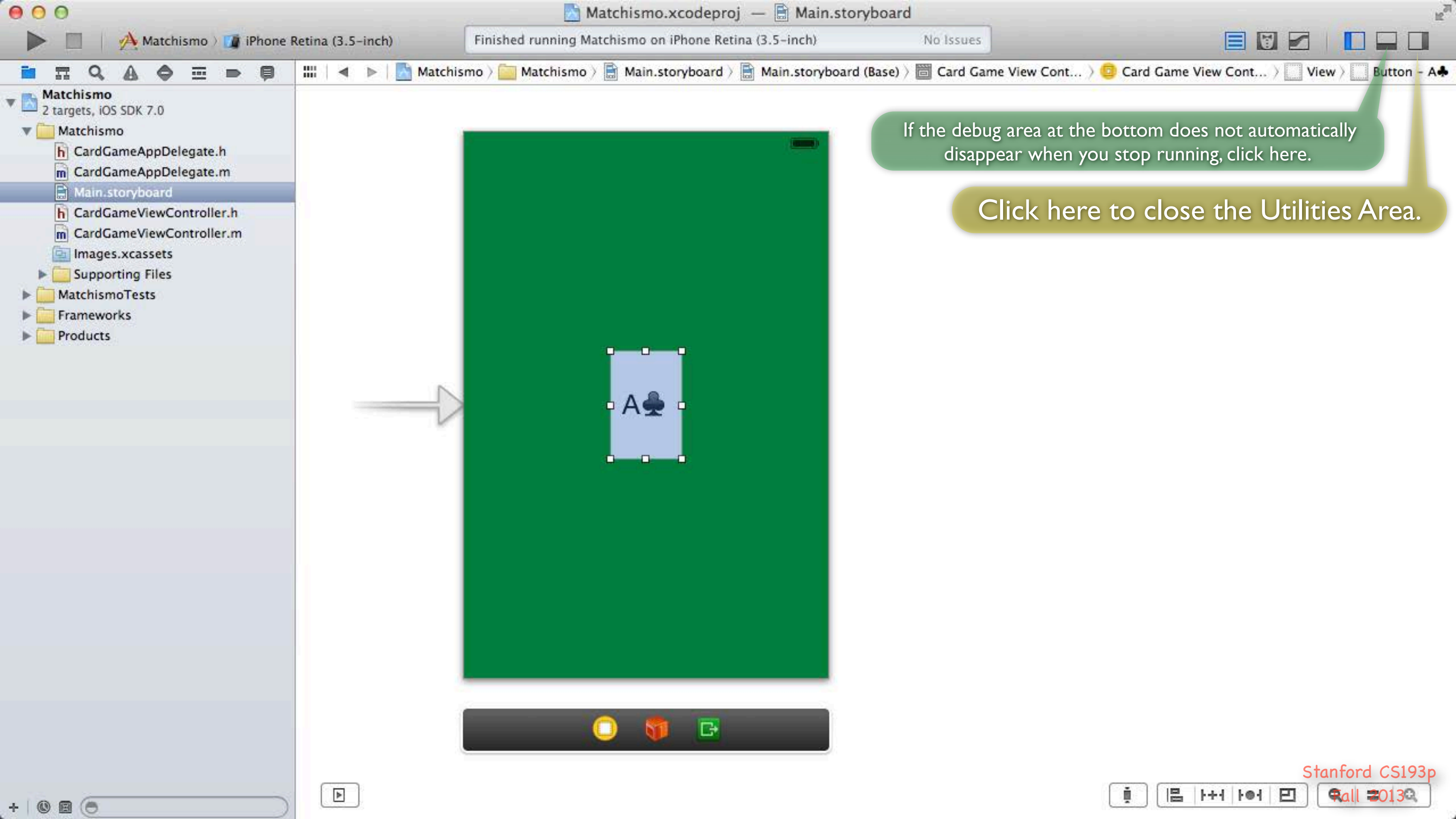
And swipe over to see (and run) other applications.

You can swipe back and click on your application (Matchismo) to get back to debugging it.





Click this "Stop" button to stop running your application in the Simulator.



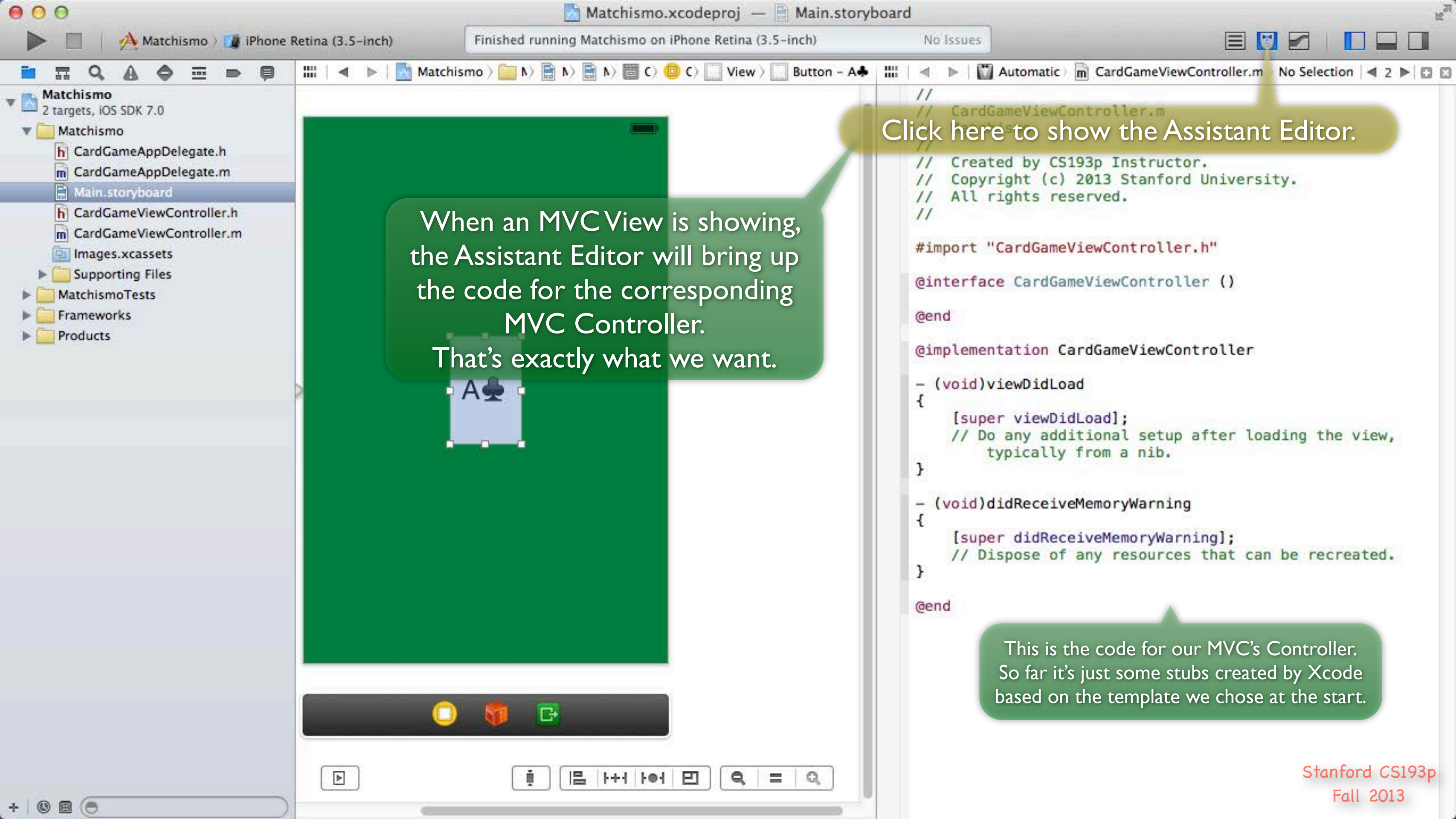
Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

Matchismo > Matchismo > Main.storyboard > Main.storyboard (Base) > Card Game View Cont... > Card Game View Cont... > View > Button - A♣

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

If the debug area at the bottom does not automatically disappear when you stop running, click here.

Click here to close the Utilities Area.



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

Matchismo iPhone Retina (3.5-inch)

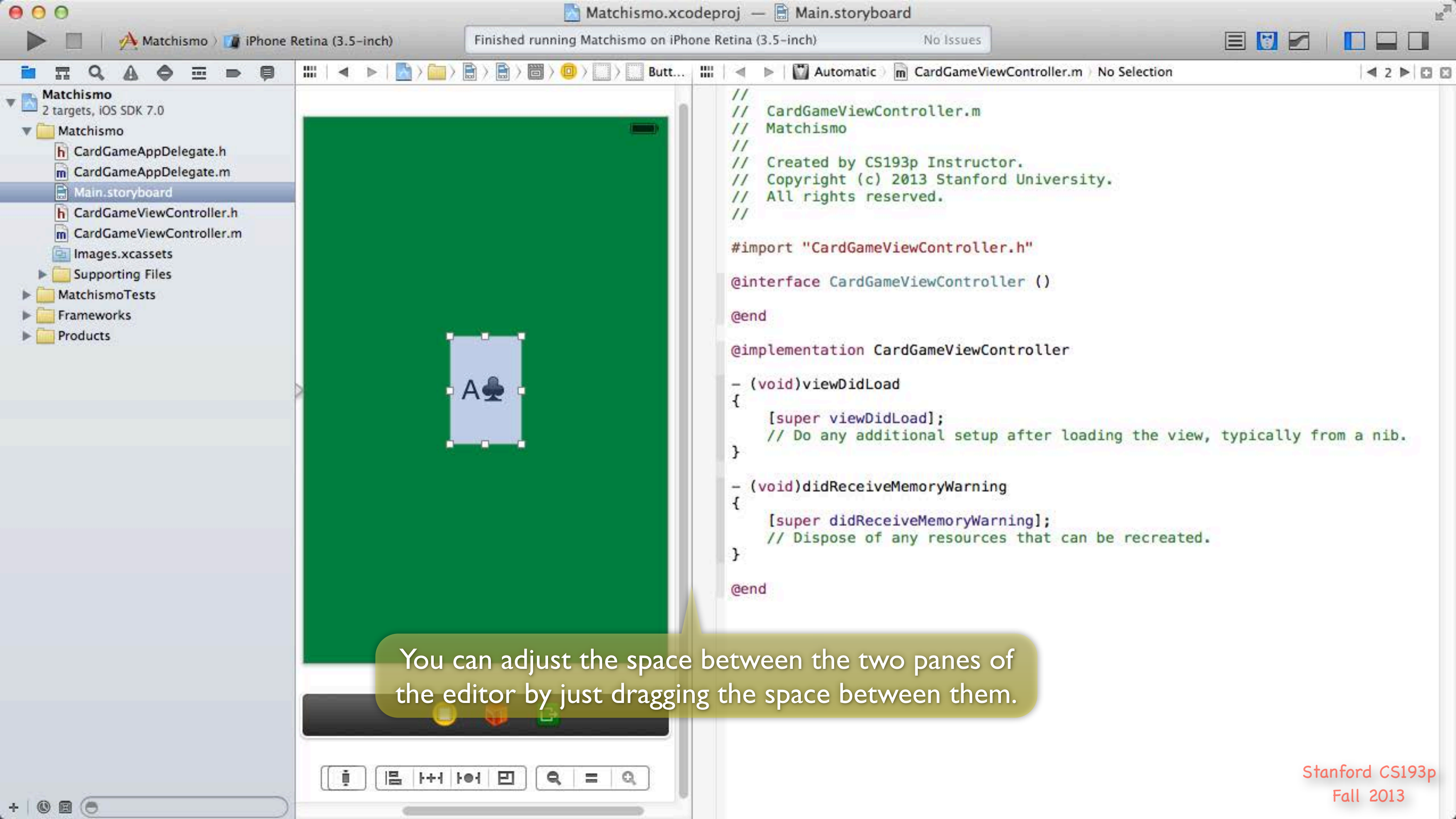
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

When an MVC View is showing, the Assistant Editor will bring up the code for the corresponding MVC Controller. That's exactly what we want.

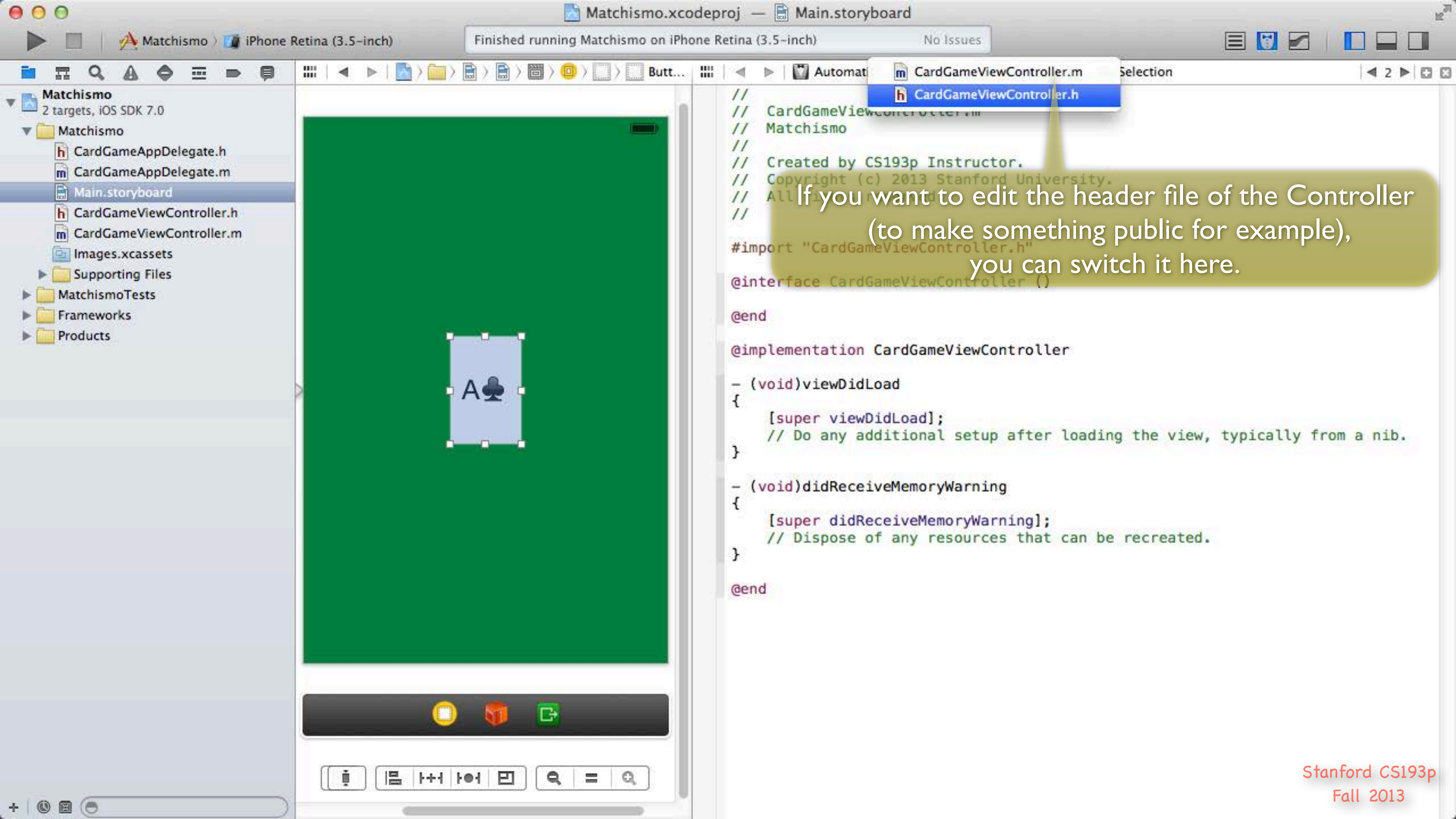
Click here to show the Assistant Editor.

```
//  
// CardGameViewController.m  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    // Do any additional setup after loading the view,  
    // typically from a nib.  
}  
  
- (void)didReceiveMemoryWarning  
{  
    [super didReceiveMemoryWarning];  
    // Dispose of any resources that can be recreated.  
}  
  
@end
```

This is the code for our MVC's Controller. So far it's just some stubs created by Xcode based on the template we chose at the start.



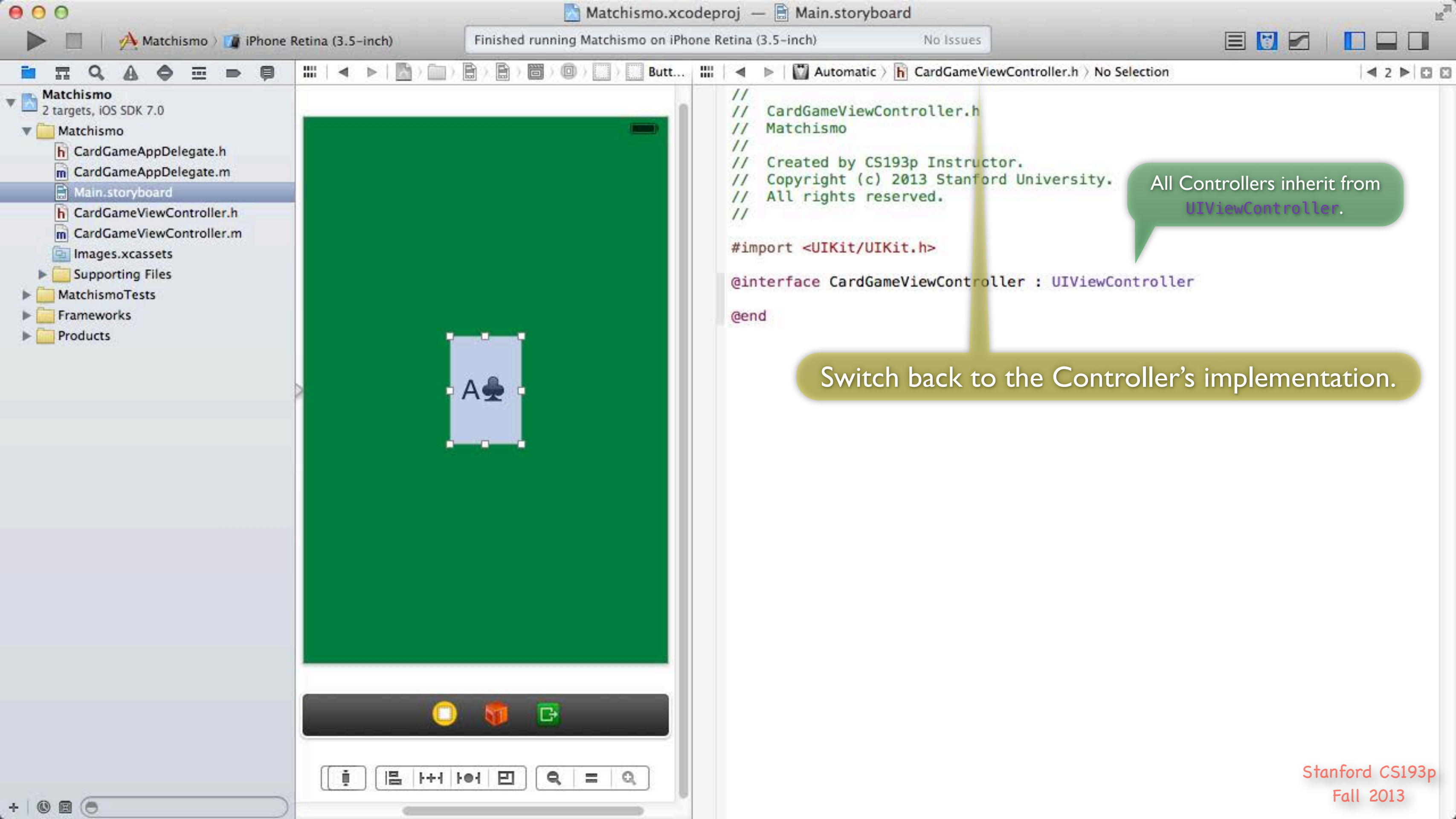
You can adjust the space between the two panes of the editor by just dragging the space between them.



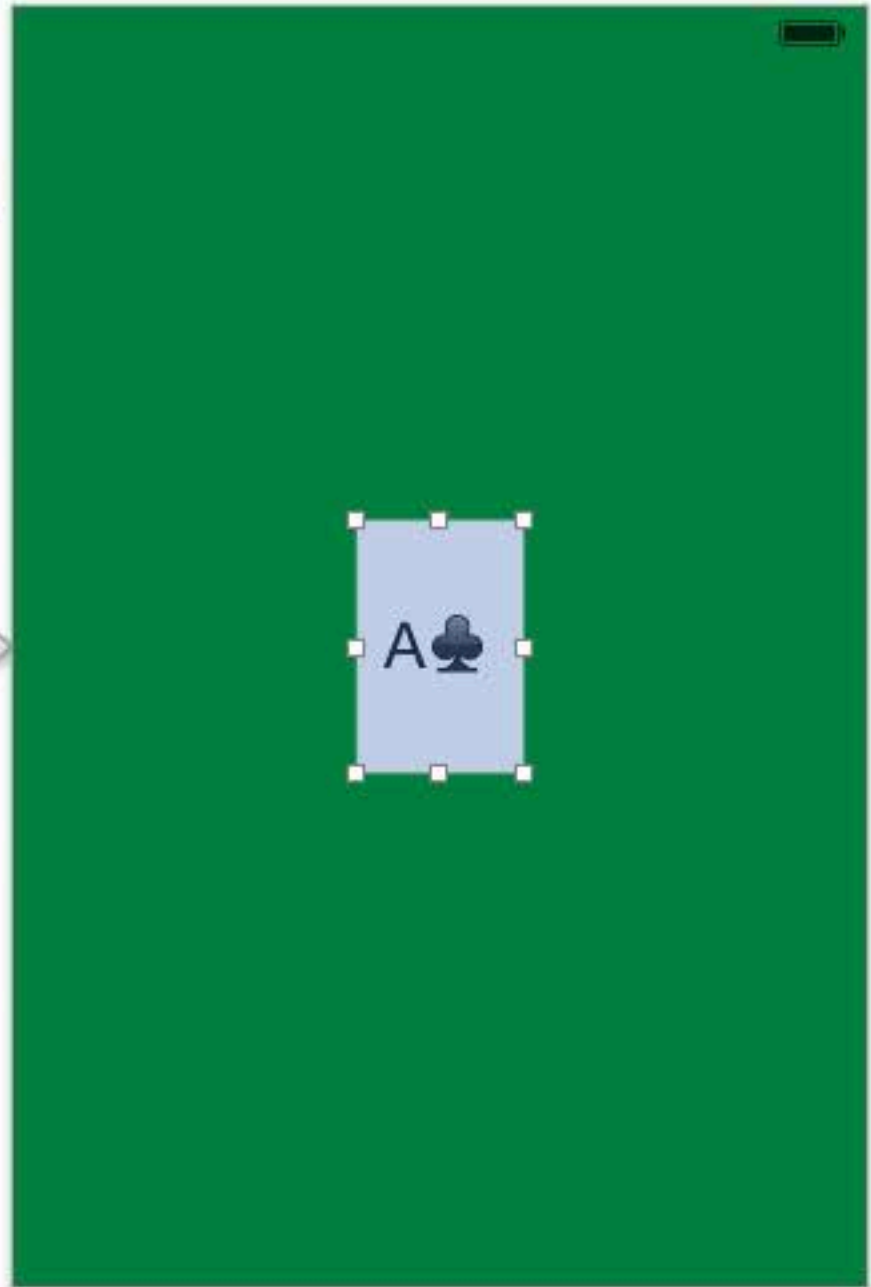
If you want to edit the header file of the Controller (to make something public for example), you can switch it here.

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    // Do any additional setup after loading the view, typically from a nib.  
}  
  
- (void)didReceiveMemoryWarning  
{  
    [super didReceiveMemoryWarning];  
    // Dispose of any resources that can be recreated.  
}  
  
@end
```





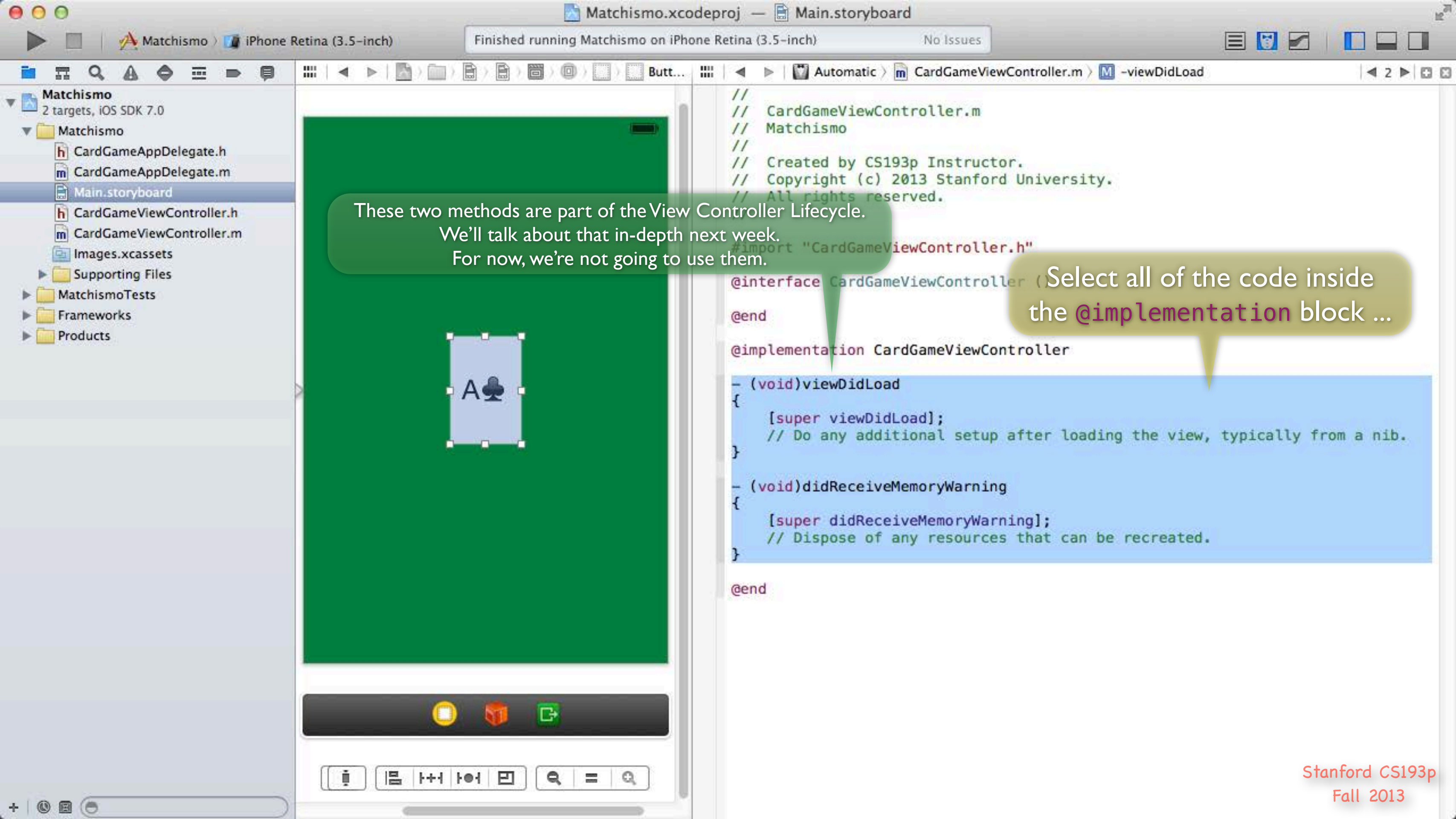
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



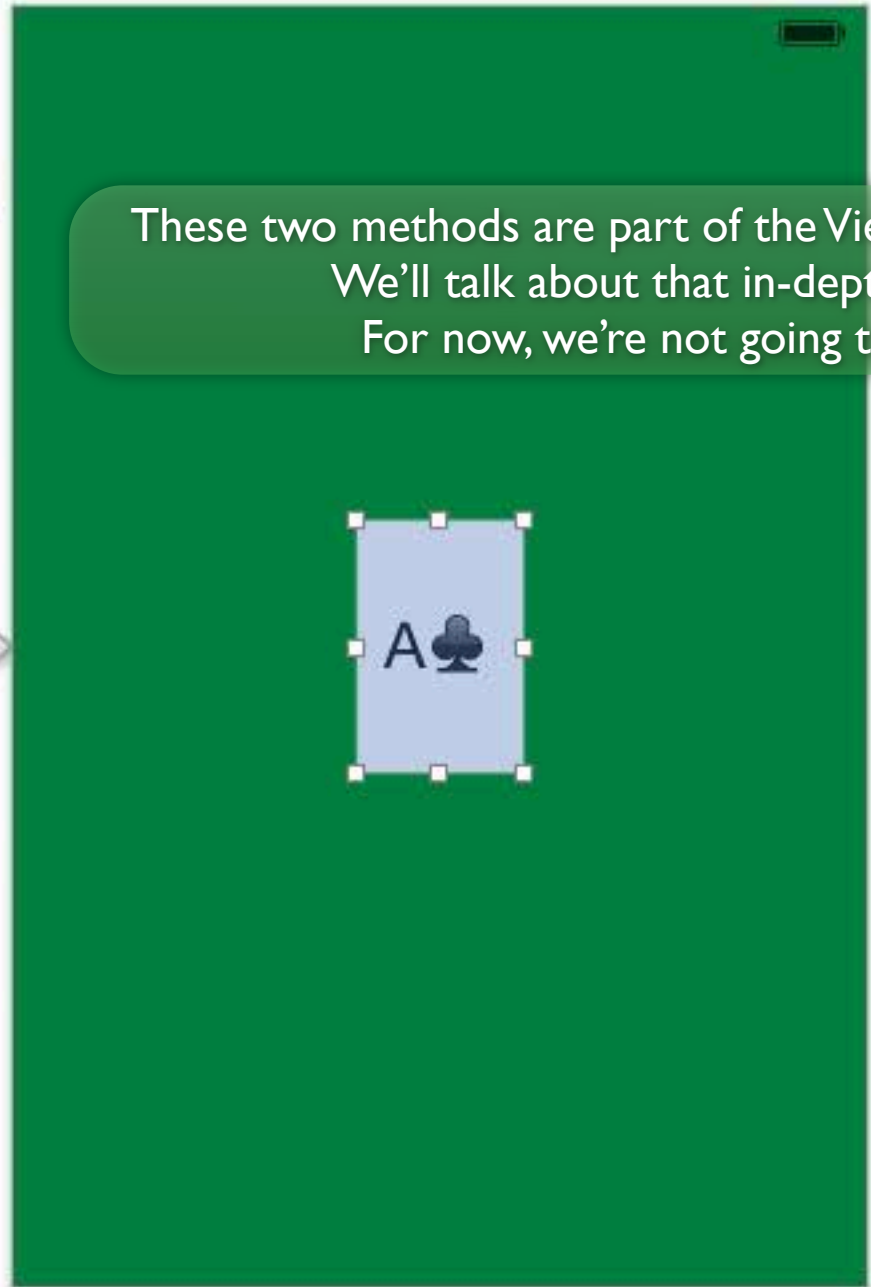
```
//  
// CardGameViewController.h  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import <UIKit/UIKit.h>  
  
@interface CardGameViewController : UIViewController  
  
@end
```

All Controllers inherit from UIViewController.

Switch back to the Controller's implementation.



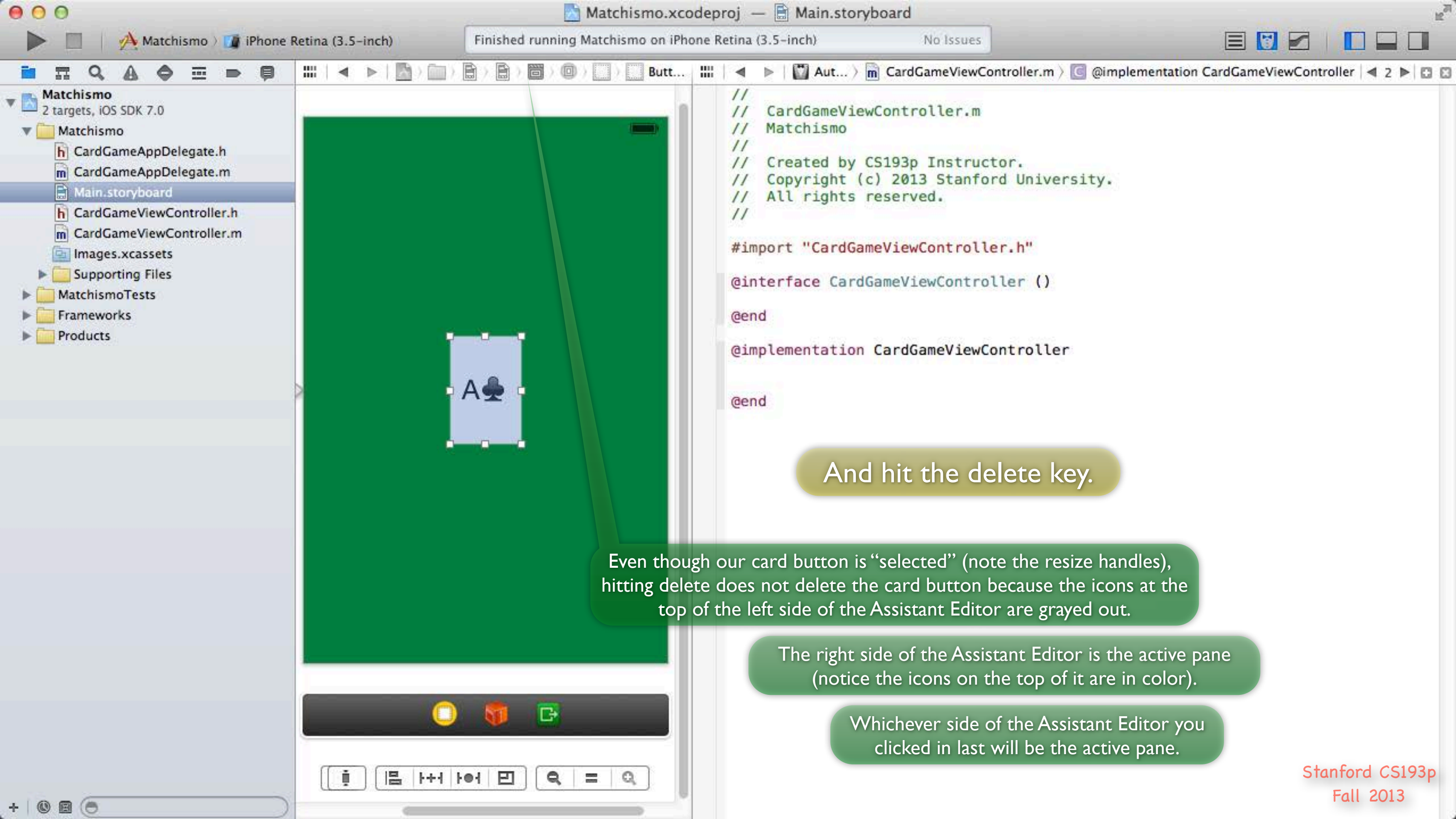
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



These two methods are part of the View Controller Lifecycle. We'll talk about that in-depth next week. For now, we're not going to use them.

Select all of the code inside the @implementation block ...

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    // Do any additional setup after loading the view, typically from a nib.  
}  
  
- (void)didReceiveMemoryWarning  
{  
    [super didReceiveMemoryWarning];  
    // Dispose of any resources that can be recreated.  
}  
  
@end
```



And hit the delete key.

Even though our card button is "selected" (note the resize handles), hitting delete does not delete the card button because the icons at the top of the left side of the Assistant Editor are grayed out.

The right side of the Assistant Editor is the active pane (notice the icons on the top of it are in color).

Whichever side of the Assistant Editor you clicked in last will be the active pane.

Now it's time to connect the button to our Controller so that touching the button "flips the card over."

Believe it or not, you connect your View to your Controller by directly dragging from objects in your View into your source code. Sounds crazy, I know ...

Hold down the ctrl key while dragging from the card button into the **@implementation** block somewhere.

If you do not hold down the ctrl key, this will simply drag the button around.

Be sure to drag somewhere between the **@implementation** and the **@end**.

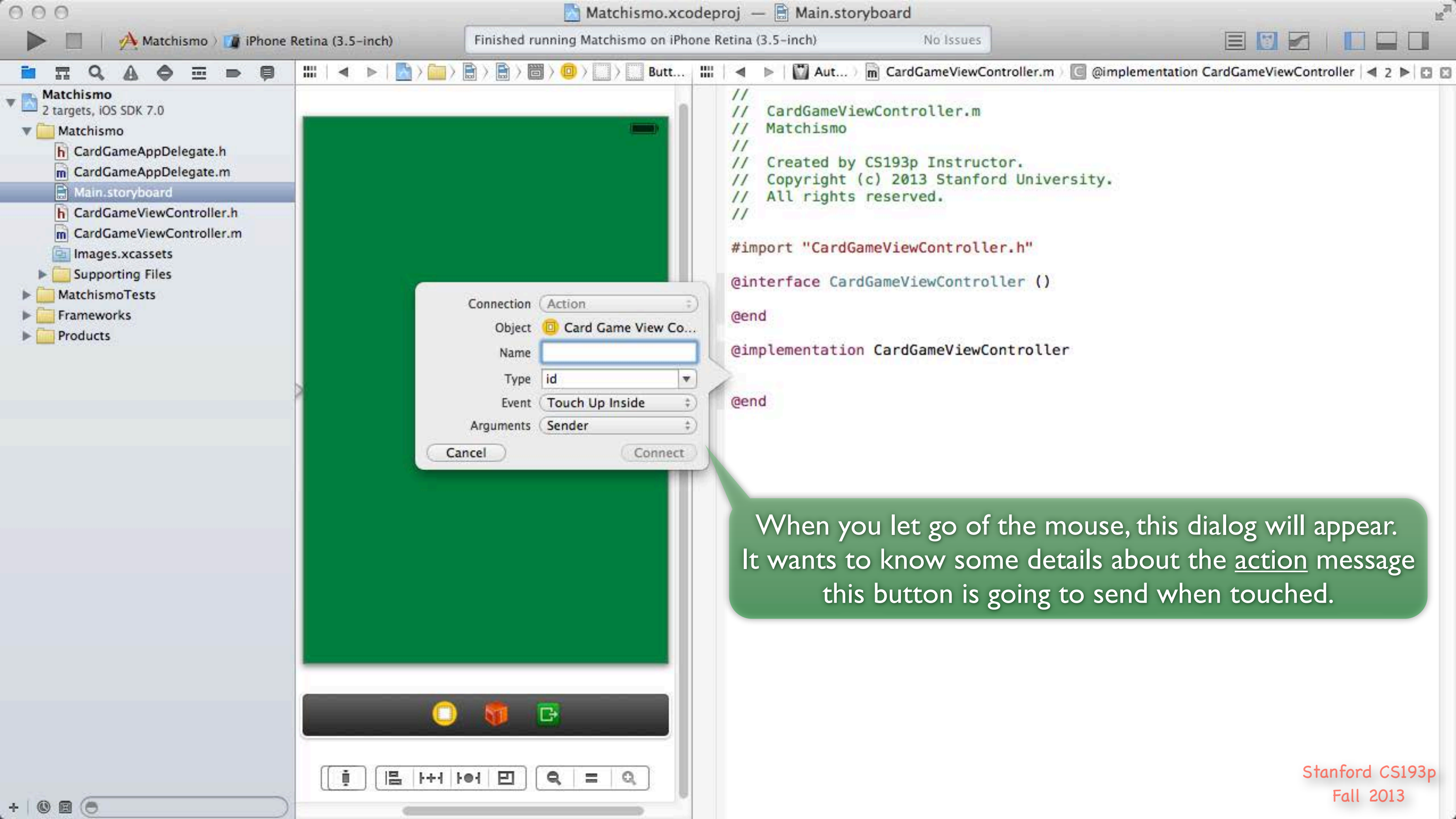
You can drag from your View to the header (.h) file of your Controller if you want to make a public connection (rare).

```
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.

#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController
    [Insert Action]
@end
```



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Connection: Action

Object: Card Game View Co...

Name:

Type: id

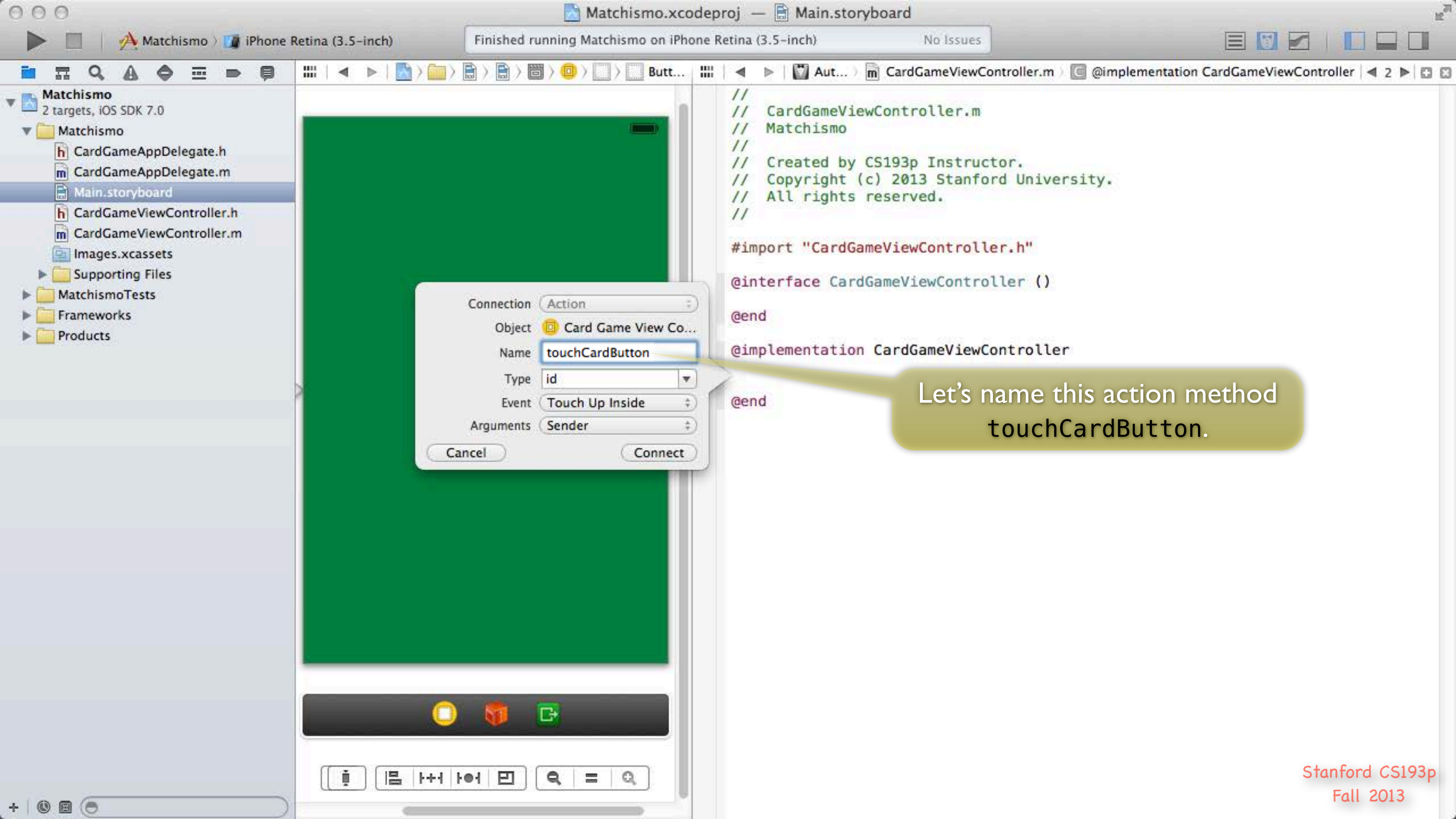
Event: Touch Up Inside

Arguments: Sender

Cancel Connect

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
@end
```

When you let go of the mouse, this dialog will appear. It wants to know some details about the action message this button is going to send when touched.



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Connection: Action

Object: Card Game View Co...

Name: touchCardButton

Type: id

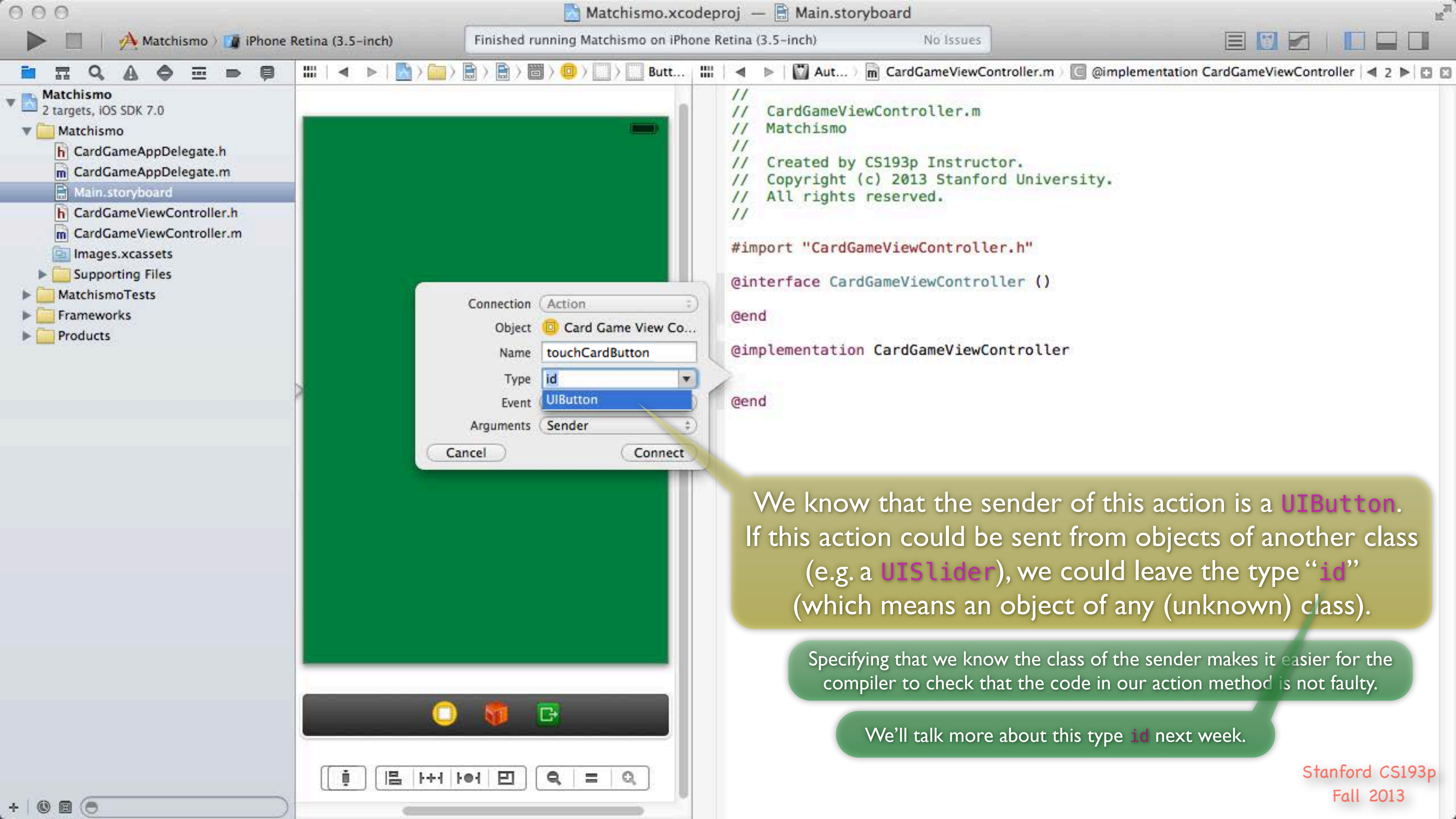
Event: Touch Up Inside

Arguments: Sender

Cancel Connect

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
@end
```

Let's name this action method touchCardButton.



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Connection: Action

Object: Card Game View Co...

Name: touchCardButton

Type: id

Event: UIButton

Arguments: Sender

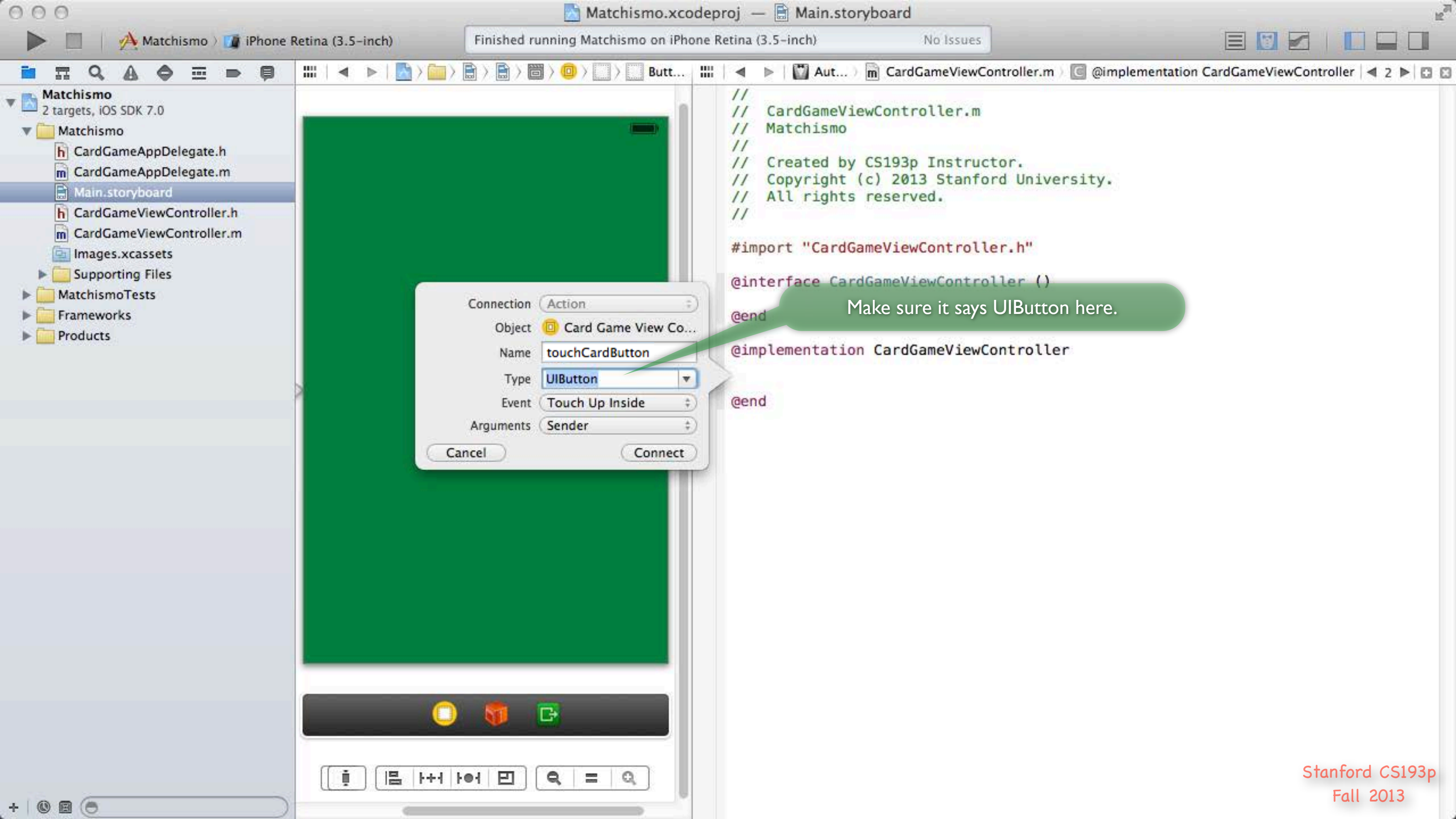
Cancel Connect

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
@end
```

We know that the sender of this action is a **UIButton**. If this action could be sent from objects of another class (e.g. a **UISlider**), we could leave the type "id" (which means an object of any (unknown) class).

Specifying that we know the class of the sender makes it easier for the compiler to check that the code in our action method is not faulty.

We'll talk more about this type **id** next week.



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo  
2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products

Connection: Action

Object: Card Game View Co...

Name: touchCardButton

Type: UIButton

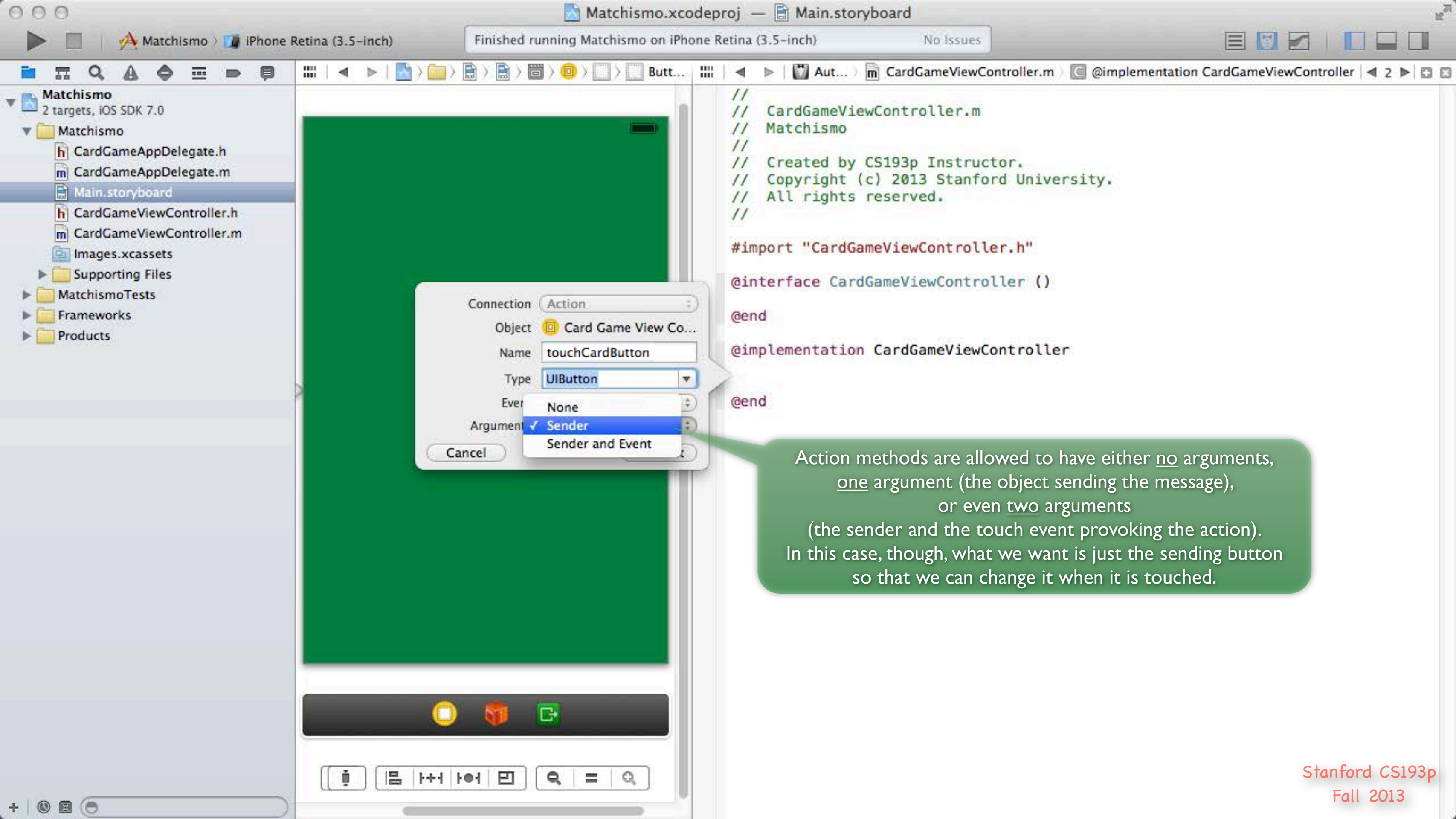
Event: Touch Up Inside

Arguments: Sender

Cancel Connect

Make sure it says UIButton here.





- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Connection: Action

Object: Card Game View Co...

Name: touchCardButton

Type: UIButton

Event: None

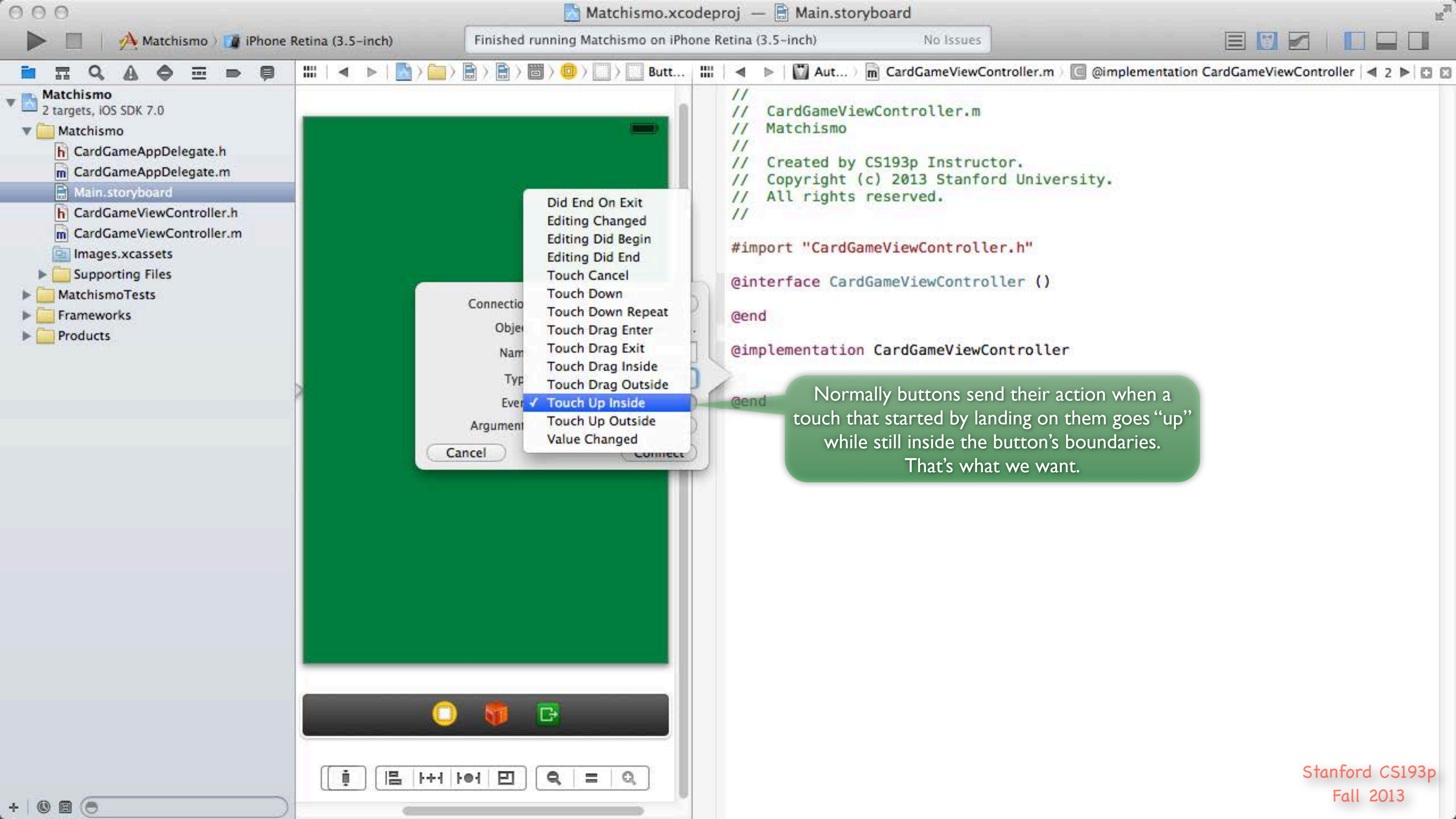
Argument:  Sender

Sender and Event

Cancel

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
@end
```

Action methods are allowed to have either no arguments, one argument (the object sending the message), or even two arguments (the sender and the touch event provoking the action). In this case, though, what we want is just the sending button so that we can change it when it is touched.

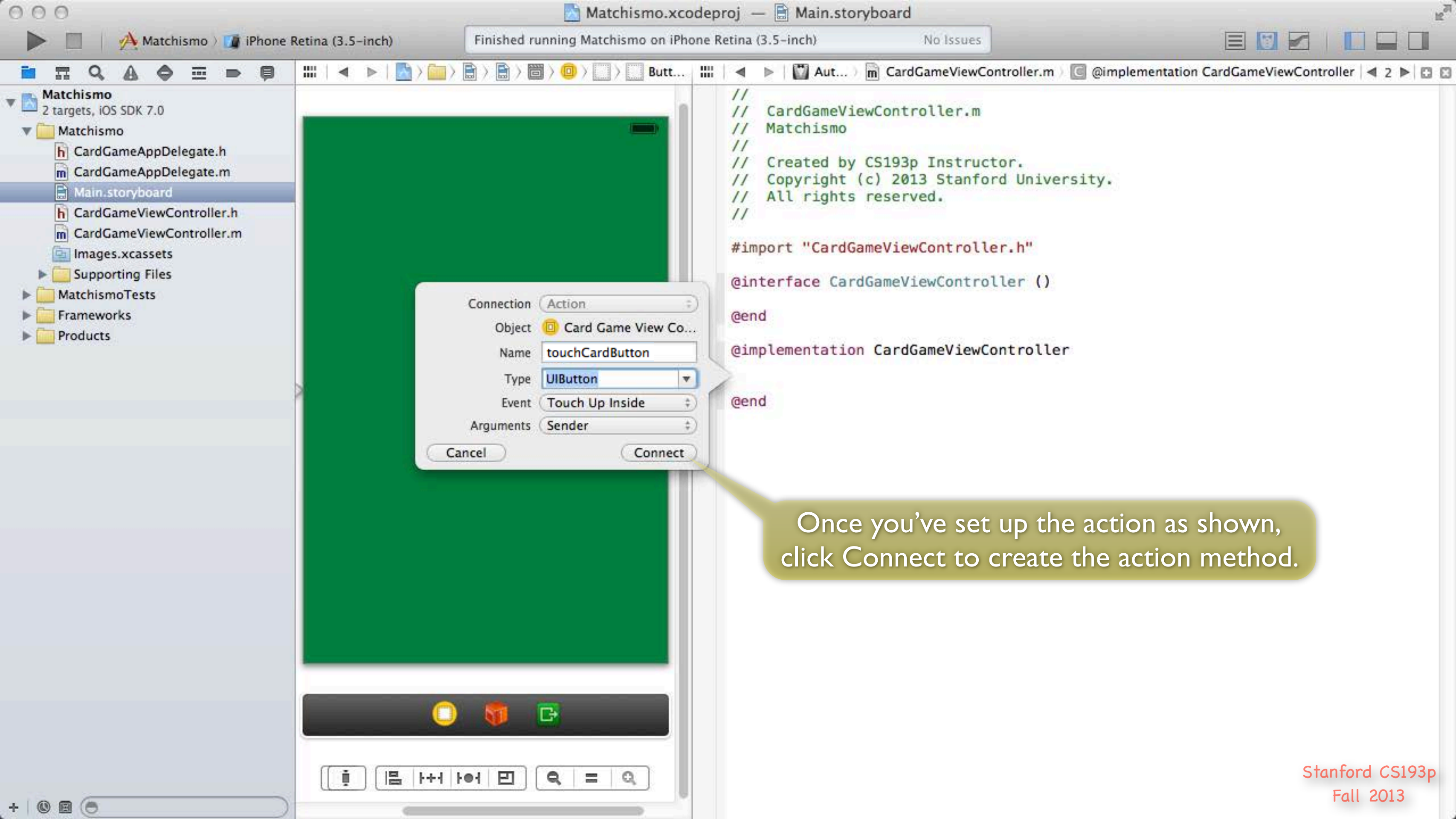


- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

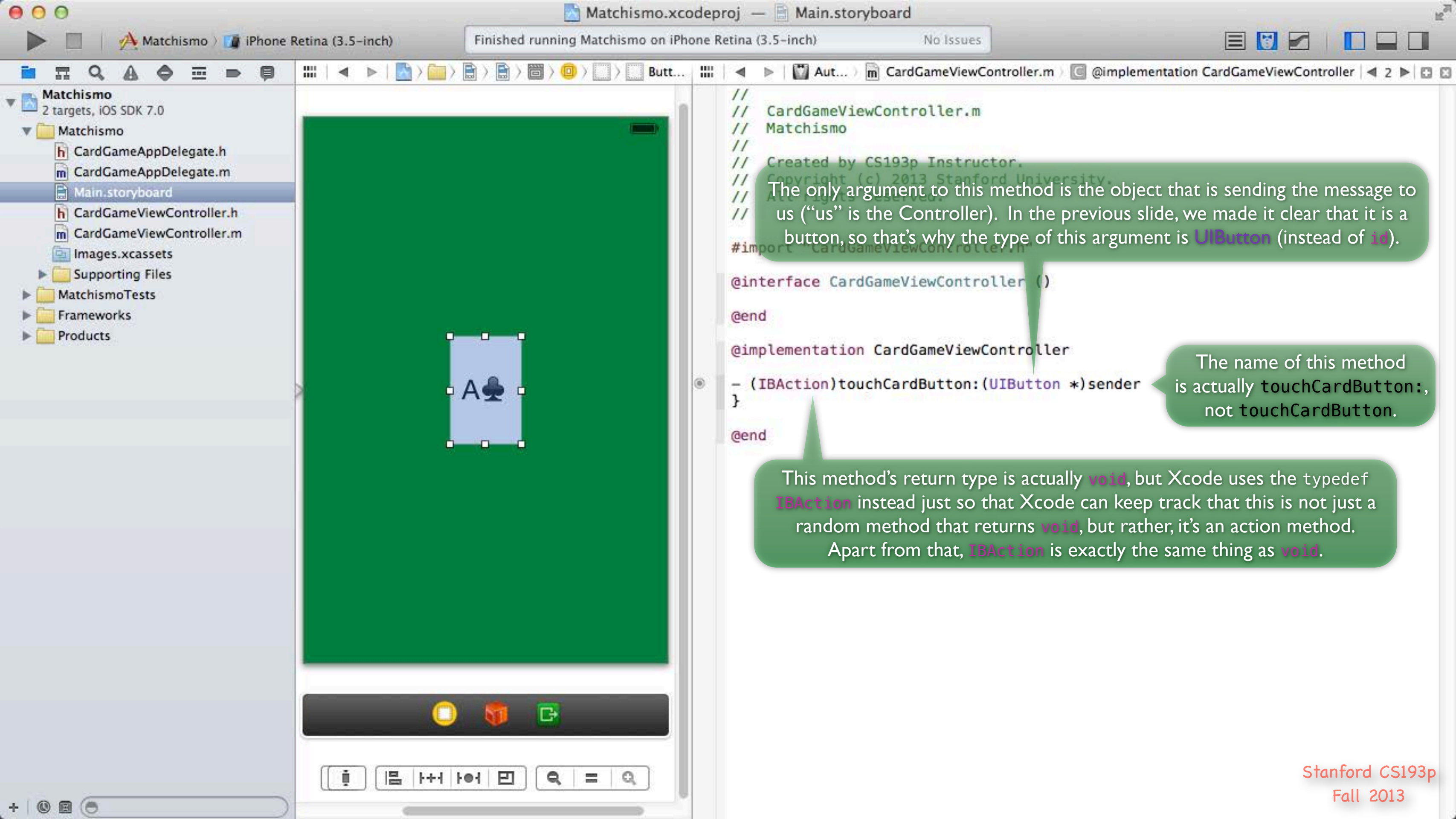
- Did End On Exit
- Editing Changed
- Editing Did Begin
- Editing Did End
- Touch Cancel
- Touch Down
- Touch Down Repeat
- Touch Drag Enter
- Touch Drag Exit
- Touch Drag Inside
- Touch Drag Outside
- Touch Up Inside**
- Touch Up Outside
- Value Changed

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
@end
```

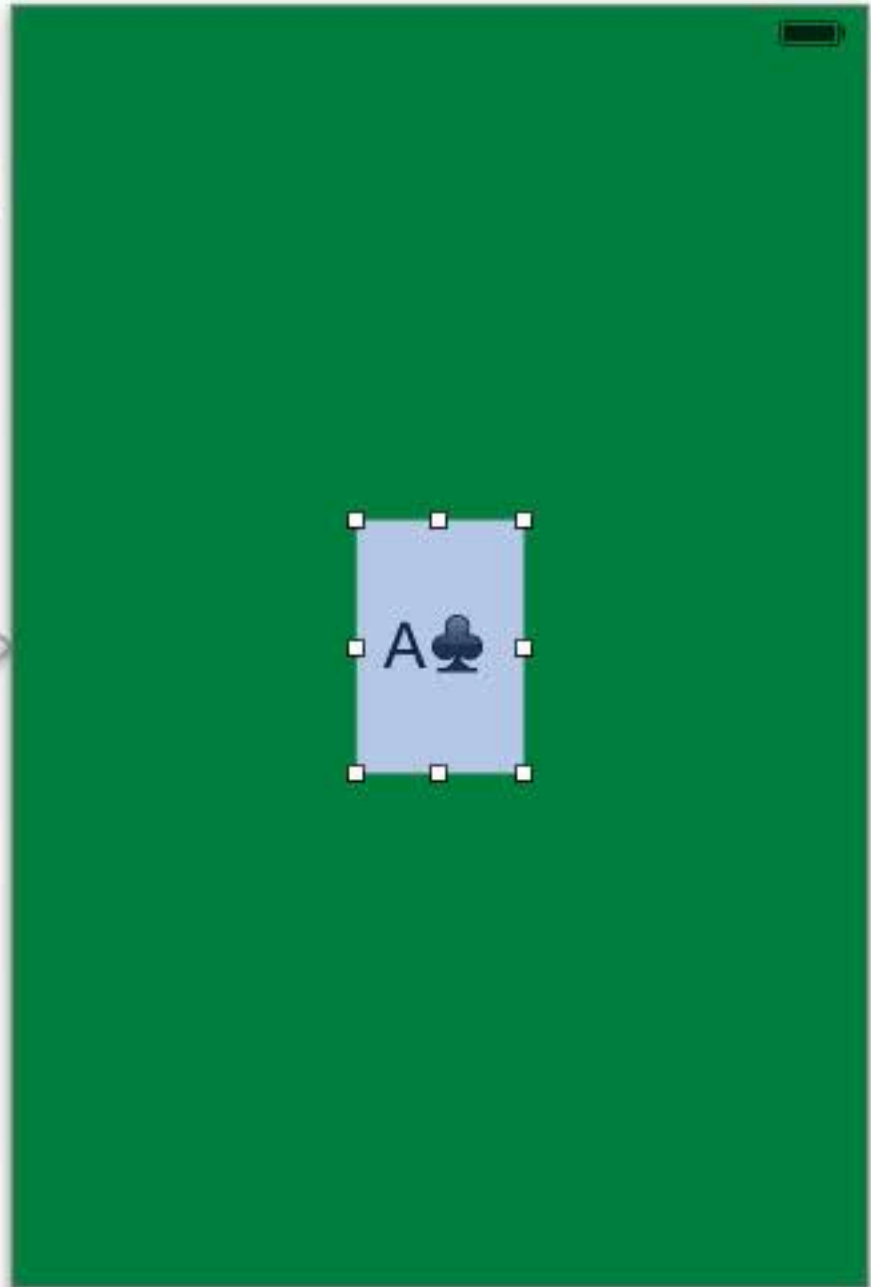
Normally buttons send their action when a touch that started by landing on them goes "up" while still inside the button's boundaries. That's what we want.



Once you've set up the action as shown, click Connect to create the action method.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

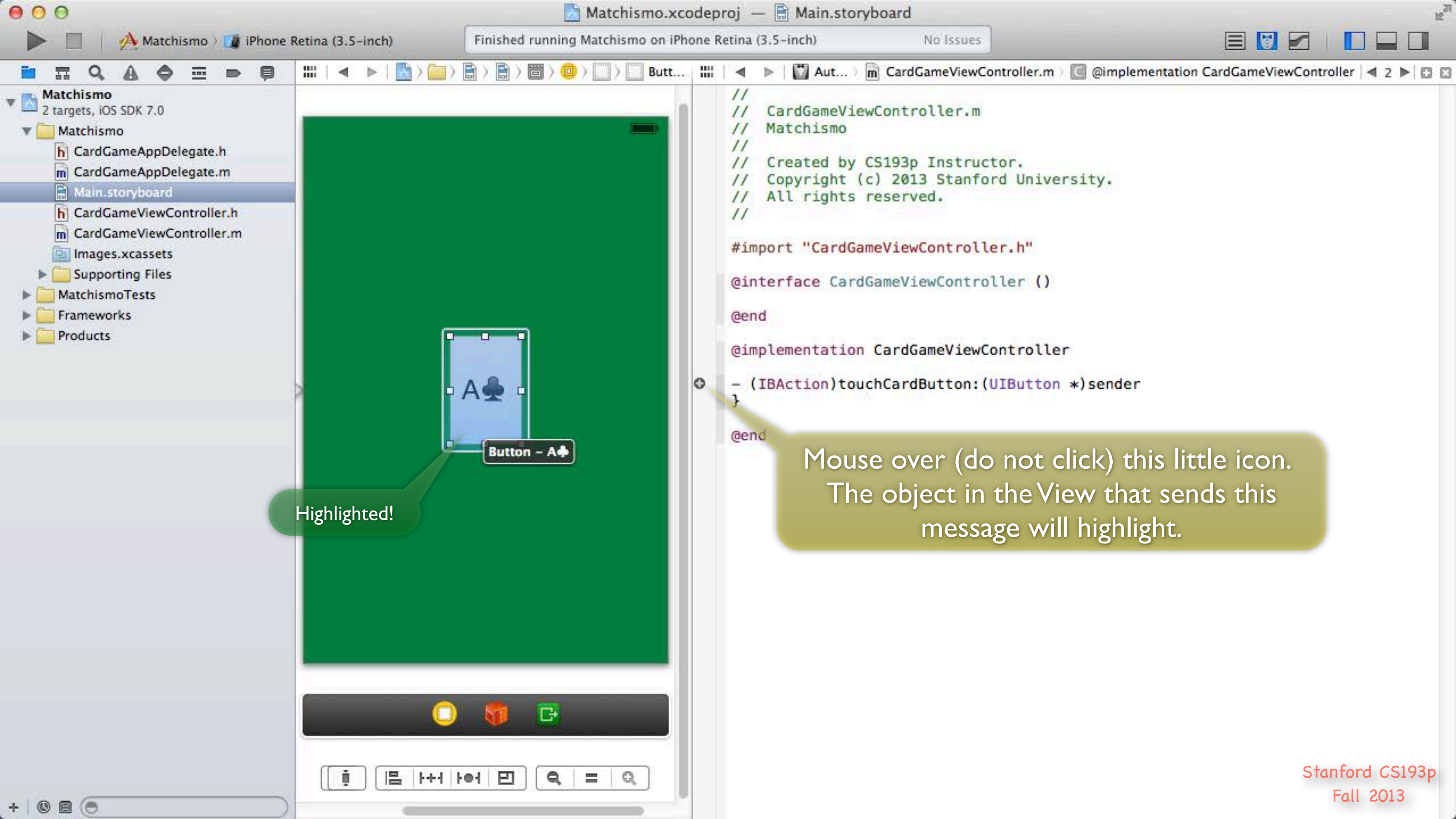
- (IBAction)touchCardButton:(UIButton *)sender
}

@end
```

The only argument to this method is the object that is sending the message to us ("us" is the Controller). In the previous slide, we made it clear that it is a button, so that's why the type of this argument is **UIButton** (instead of **id**).

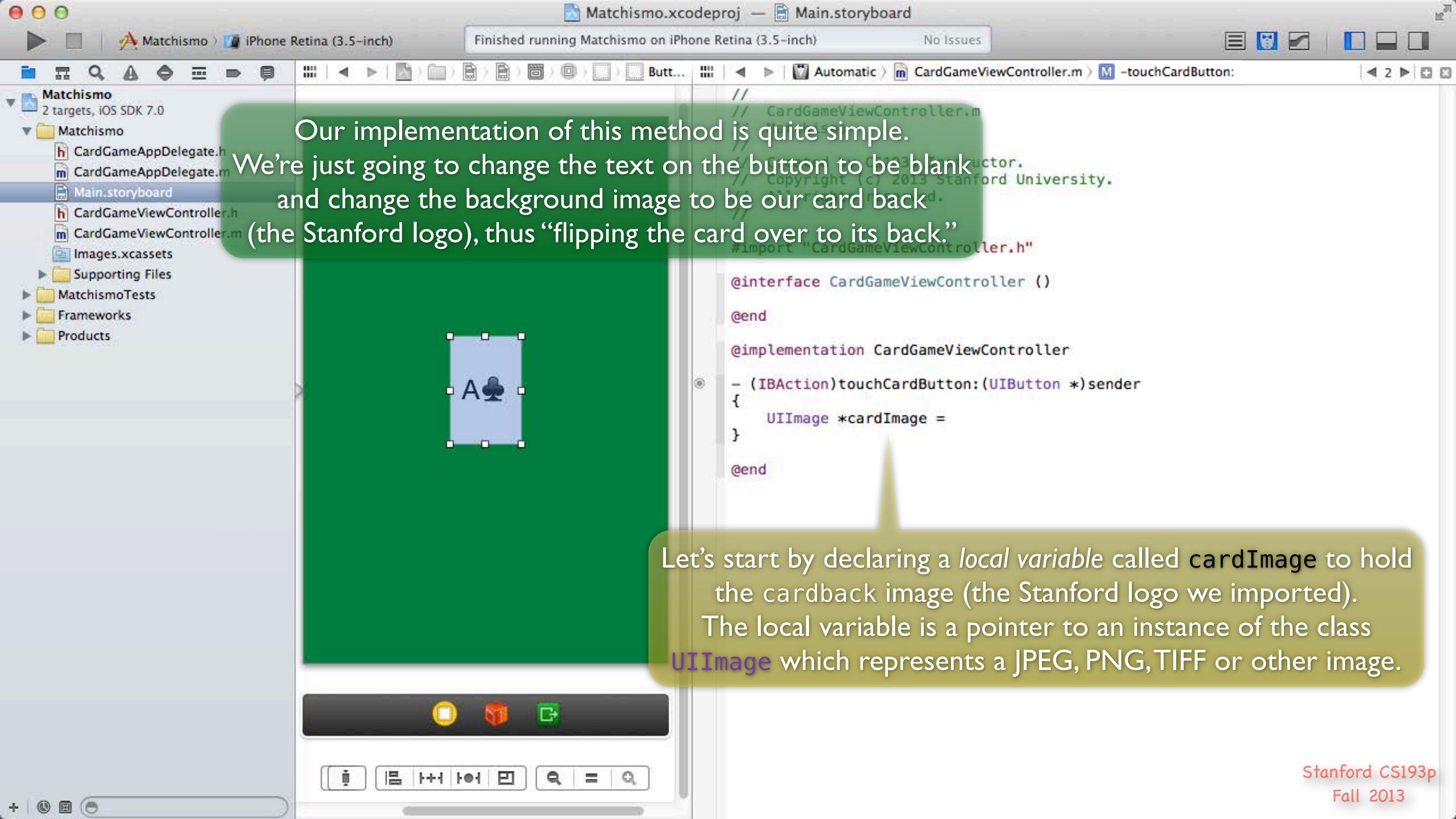
The name of this method is actually **touchCardButton:**, not **touchCardButton**.

This method's return type is actually **void**, but Xcode uses the typedef **IBAction** instead just so that Xcode can keep track that this is not just a random method that returns **void**, but rather, it's an action method. Apart from that, **IBAction** is exactly the same thing as **void**.



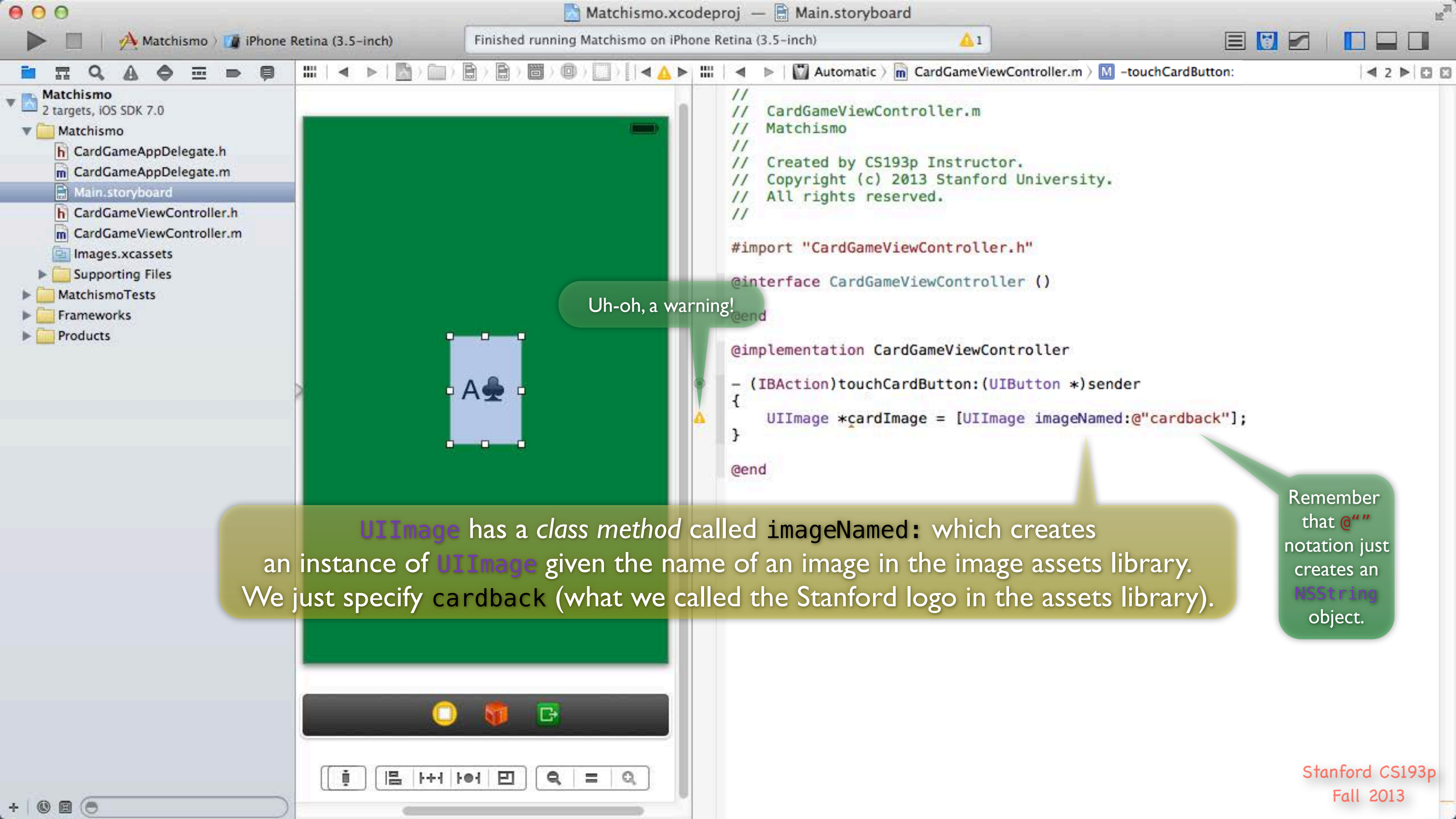
Highlighted!

Mouse over (do not click) this little icon. The object in the View that sends this message will highlight.

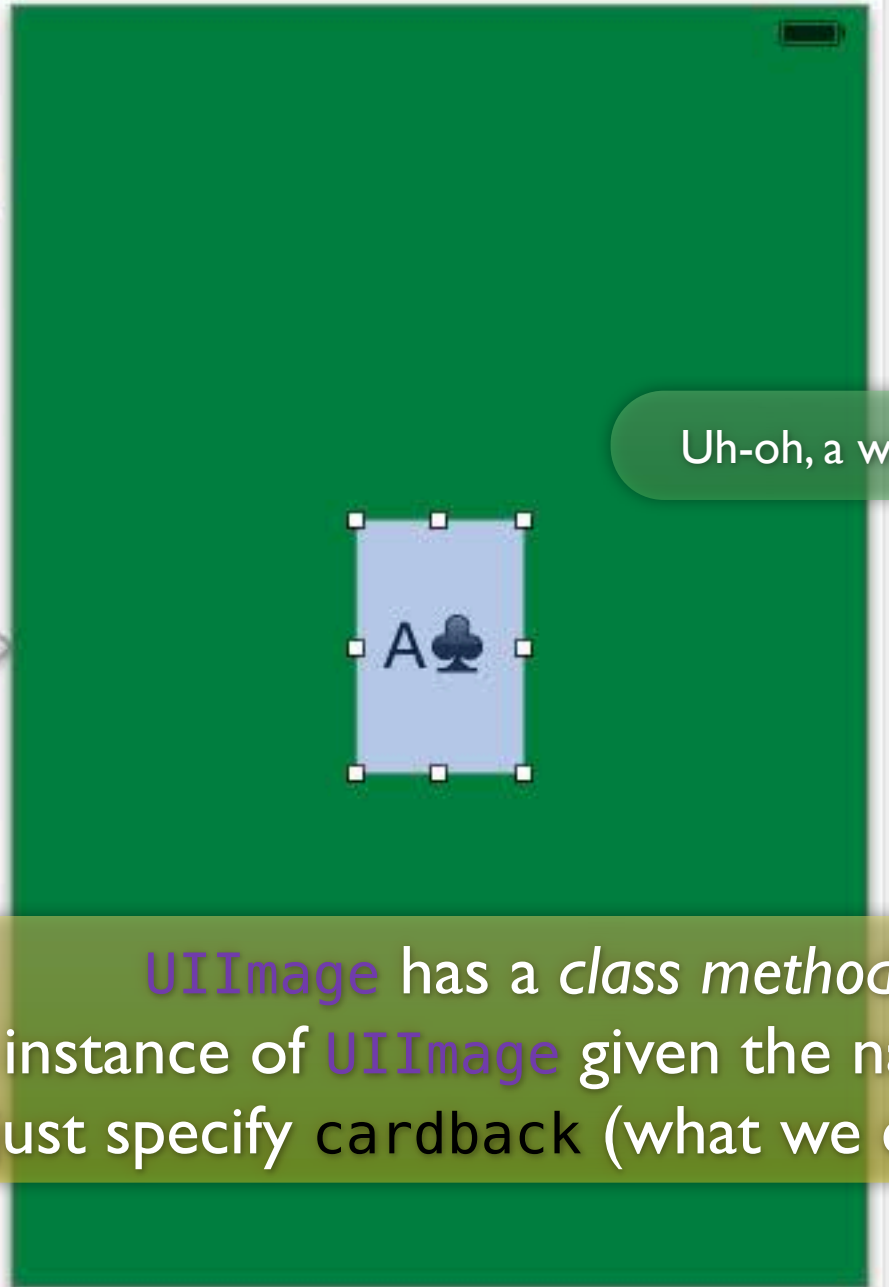


Our implementation of this method is quite simple. We're just going to change the text on the button to be blank and change the background image to be our card back (the Stanford logo), thus "flipping the card over to its back."

Let's start by declaring a *local variable* called `cardImage` to hold the cardback image (the Stanford logo we imported). The local variable is a pointer to an instance of the class `UIImage` which represents a JPEG, PNG, TIFF or other image.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

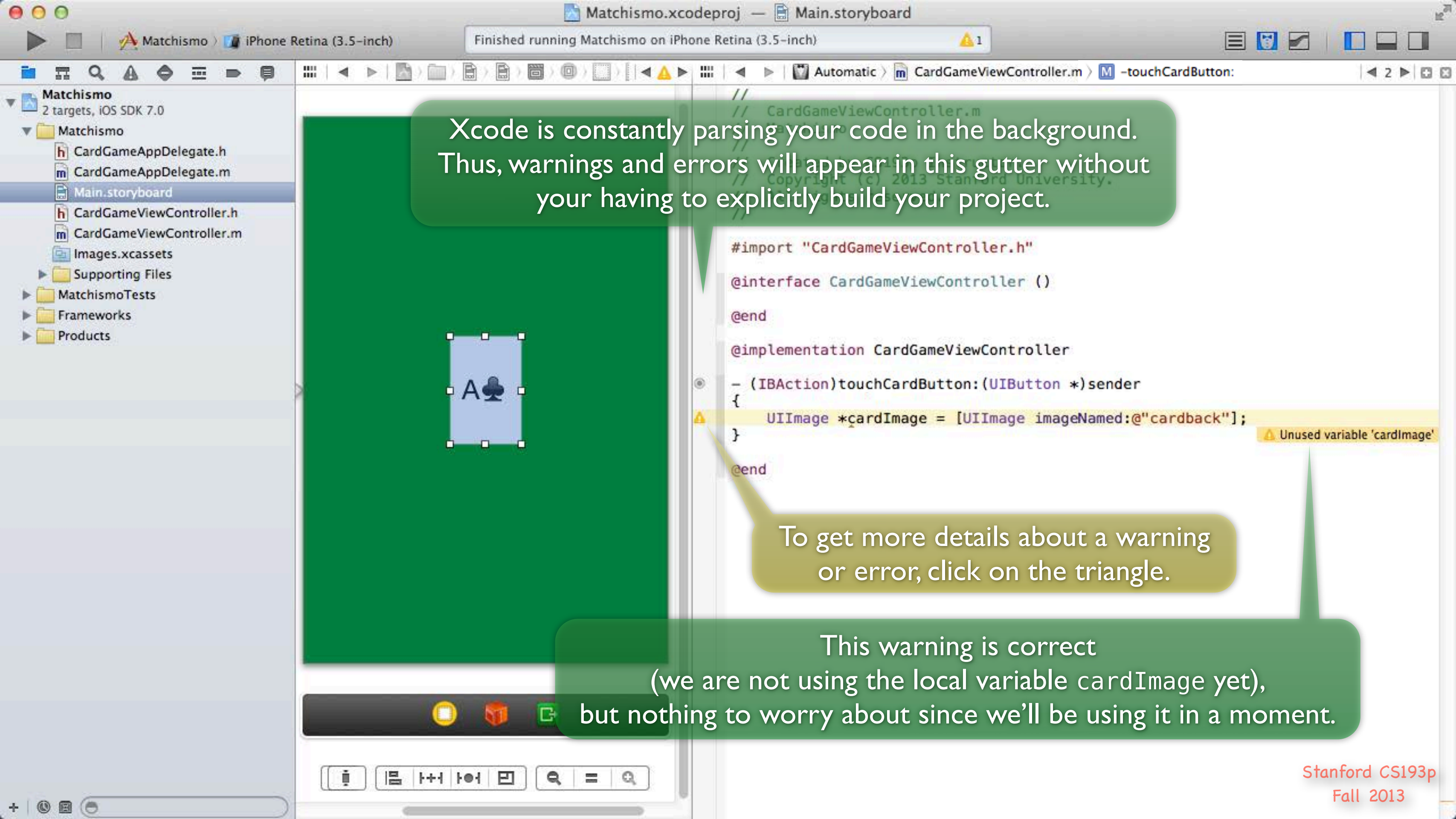
- (IBAction)touchCardButton:(UIButton *)sender
{
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];
}

@end
```

Uh-oh, a warning!

UIImage has a class method called imageNamed: which creates an instance of UIImage given the name of an image in the image assets library. We just specify cardback (what we called the Stanford logo in the assets library).

Remember that @" notation just creates an NSString object.



Xcode is constantly parsing your code in the background. Thus, warnings and errors will appear in this gutter without your having to explicitly build your project.

```
// CardGameViewController.m
// Copyright (c) 2013 Stanford University.

#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];
}

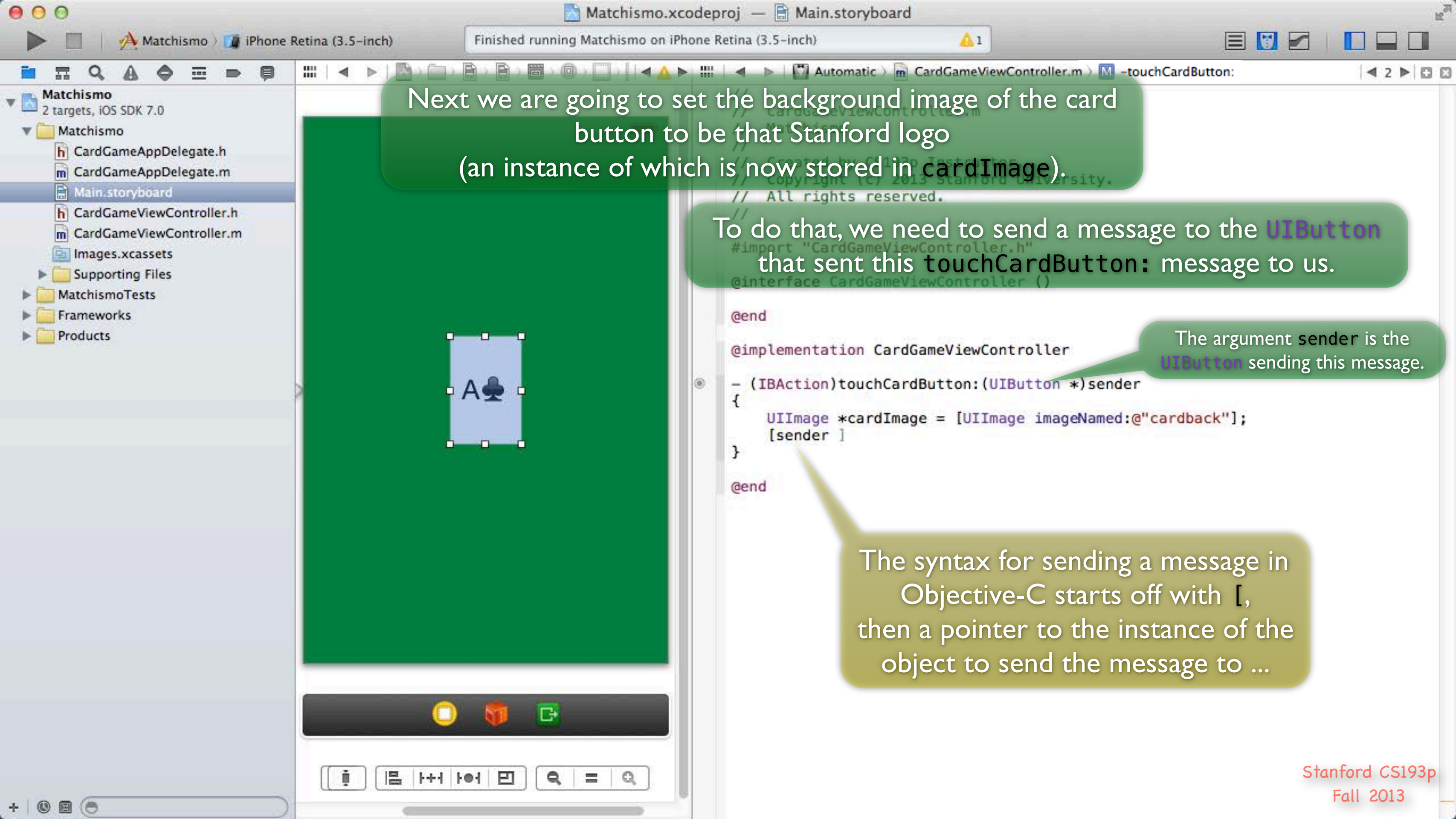
@end
```

Unused variable 'cardImage'

To get more details about a warning or error, click on the triangle.

This warning is correct (we are not using the local variable cardImage yet), but nothing to worry about since we'll be using it in a moment.



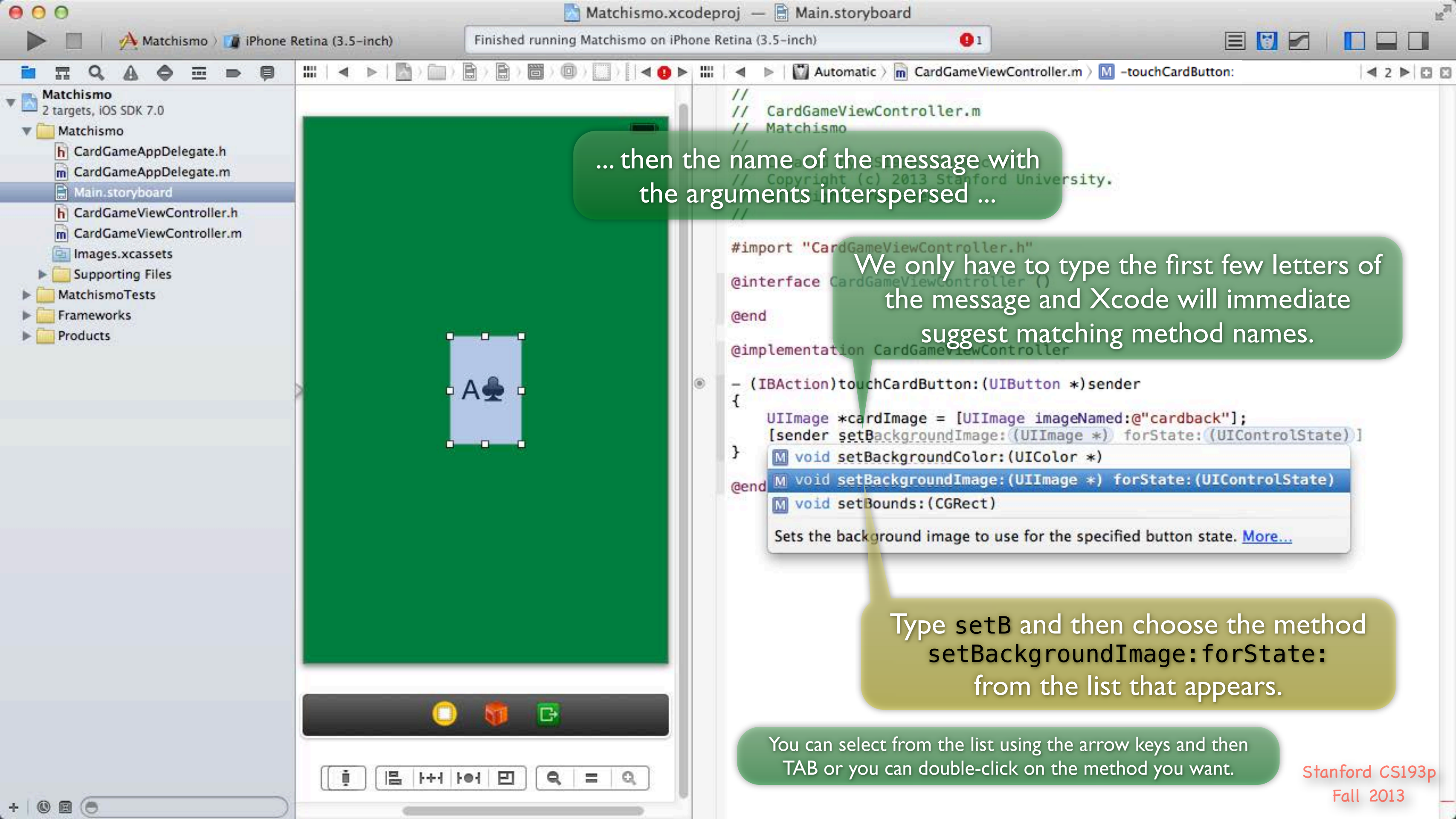


Next we are going to set the background image of the card button to be that Stanford logo (an instance of which is now stored in cardImage).

To do that, we need to send a message to the UIButton that sent this touchCardButton: message to us.

The argument sender is the UIButton sending this message.

The syntax for sending a message in Objective-C starts off with [, then a pointer to the instance of the object to send the message to ...



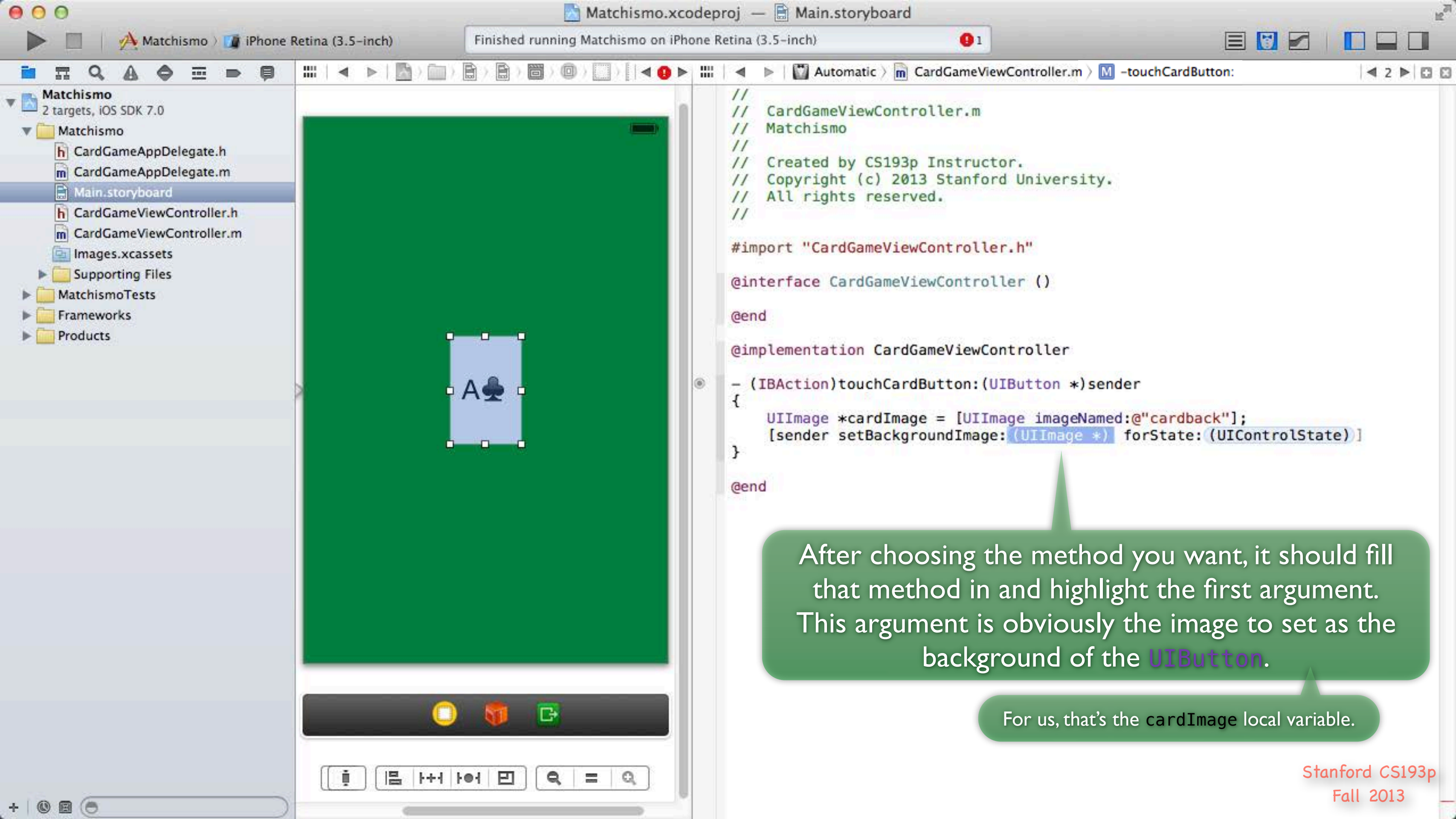
... then the name of the message with the arguments interspersed ...

We only have to type the first few letters of the message and Xcode will immediately suggest matching method names.

M void setBackgroundColor: (UIColor \*)  
M void setBackgroundImage: (UIImage \*) forState: (UIControlState)  
M void setBounds: (CGRect)  
Sets the background image to use for the specified button state. [More...](#)

Type setB and then choose the method setBackgroundImage: forState: from the list that appears.

You can select from the list using the arrow keys and then TAB or you can double-click on the method you want.

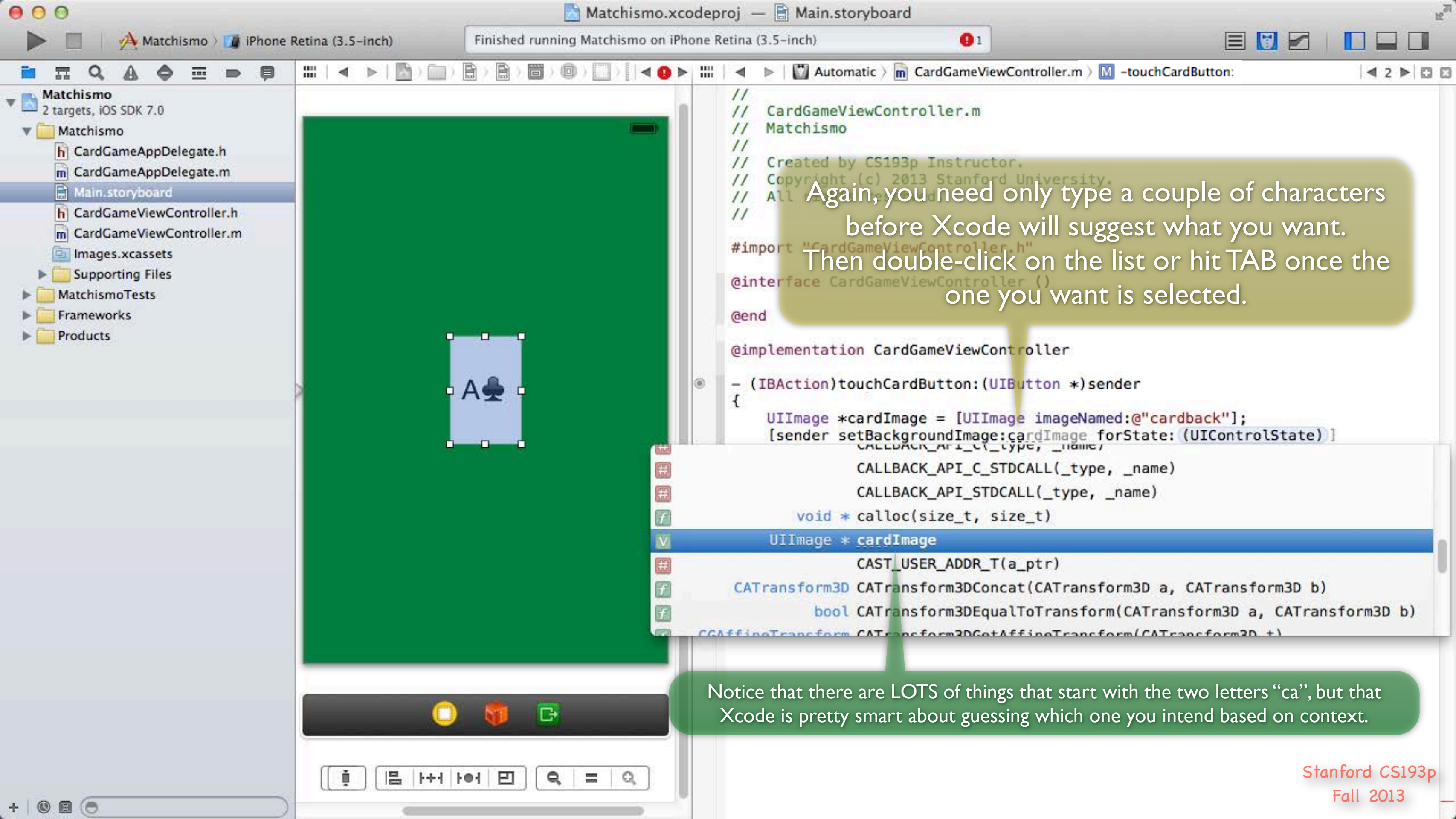


- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];  
    [sender setBackgroundImage:(UIImage *) cardImage forState:(UIControlState)]  
}  
  
@end
```

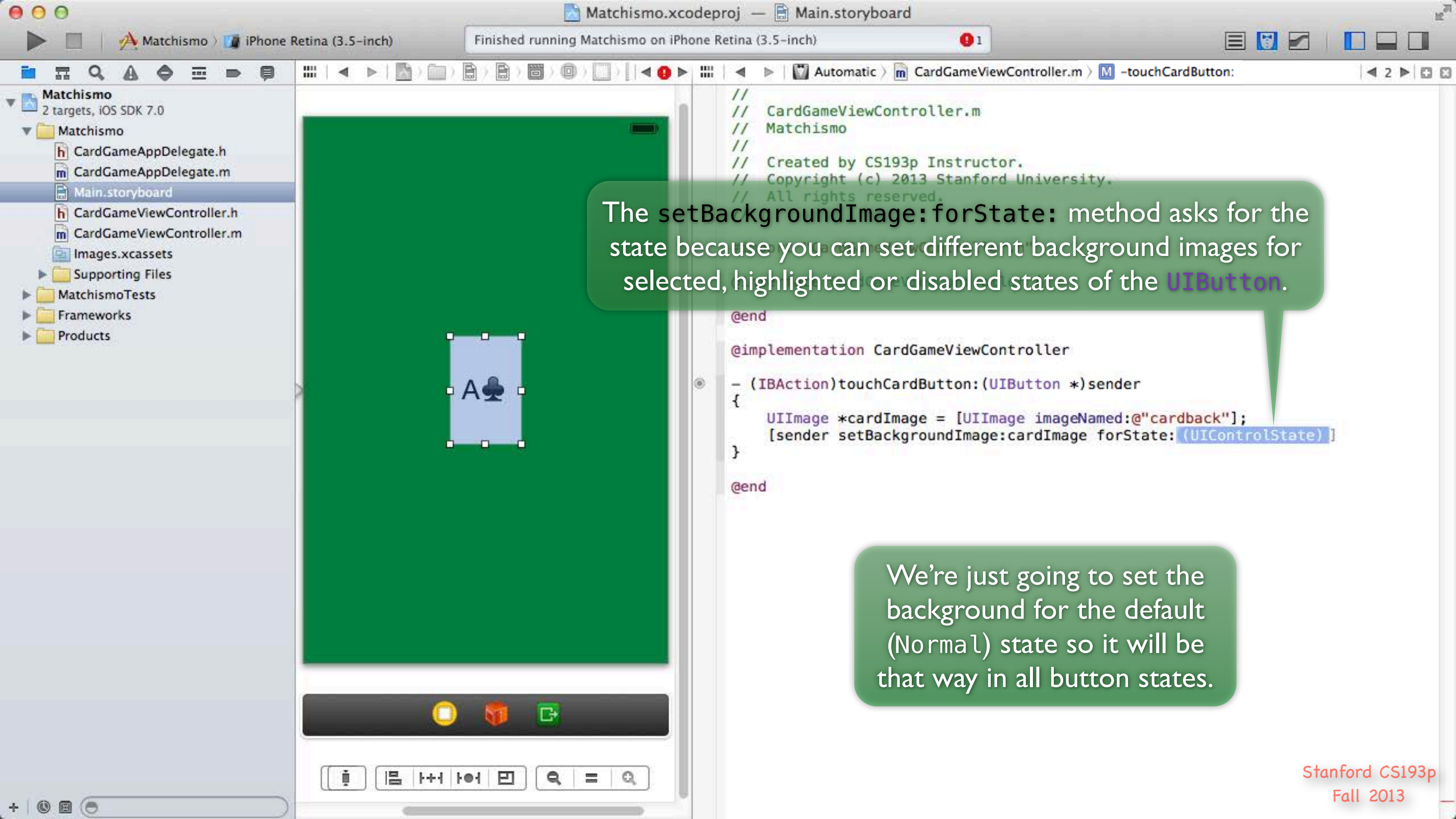
After choosing the method you want, it should fill that method in and highlight the first argument. This argument is obviously the image to set as the background of the UIButton.

For us, that's the cardImage local variable.



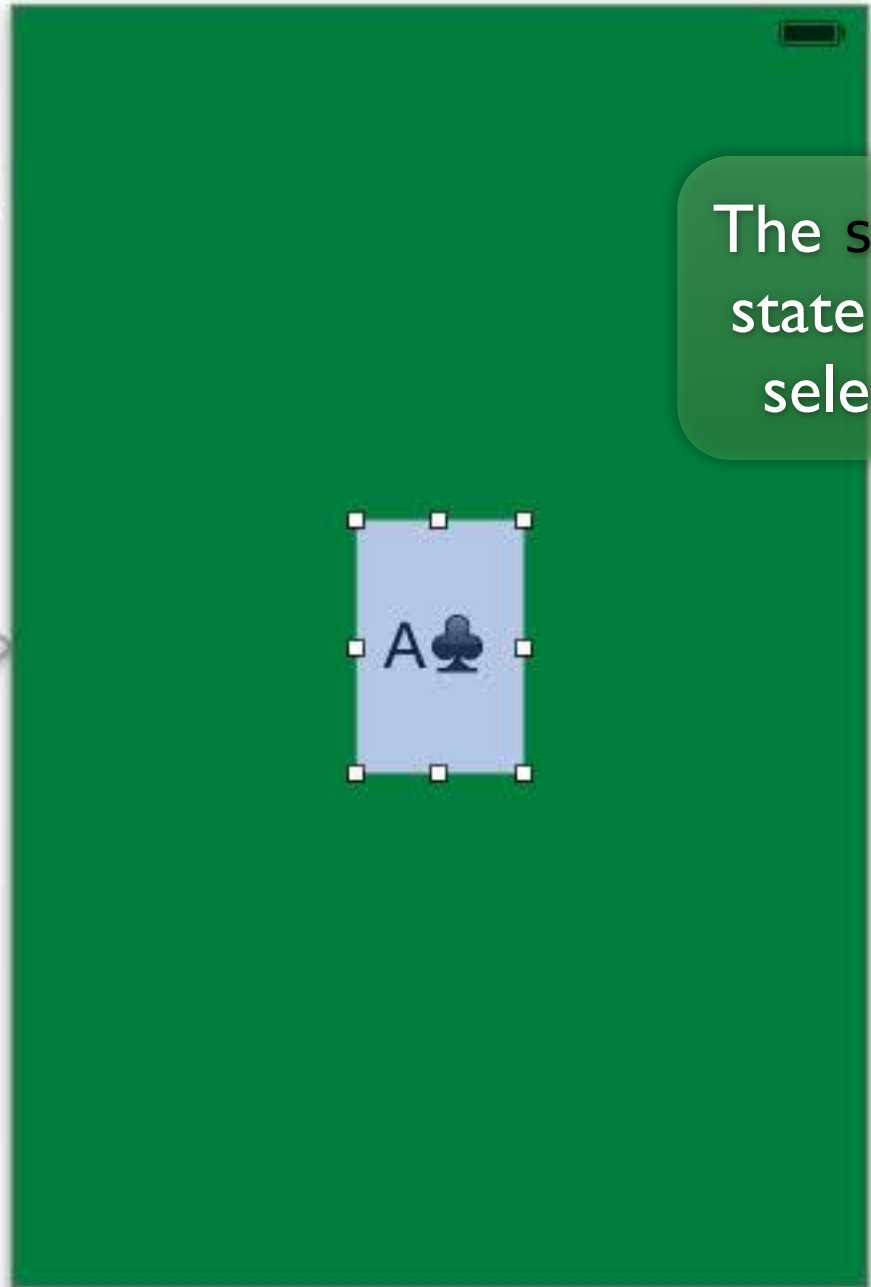
Again, you need only type a couple of characters before Xcode will suggest what you want. Then double-click on the list or hit TAB once the one you want is selected.

Notice that there are LOTS of things that start with the two letters "ca", but that Xcode is pretty smart about guessing which one you intend based on context.



Matchismo  
2 targets, iOS SDK 7.0

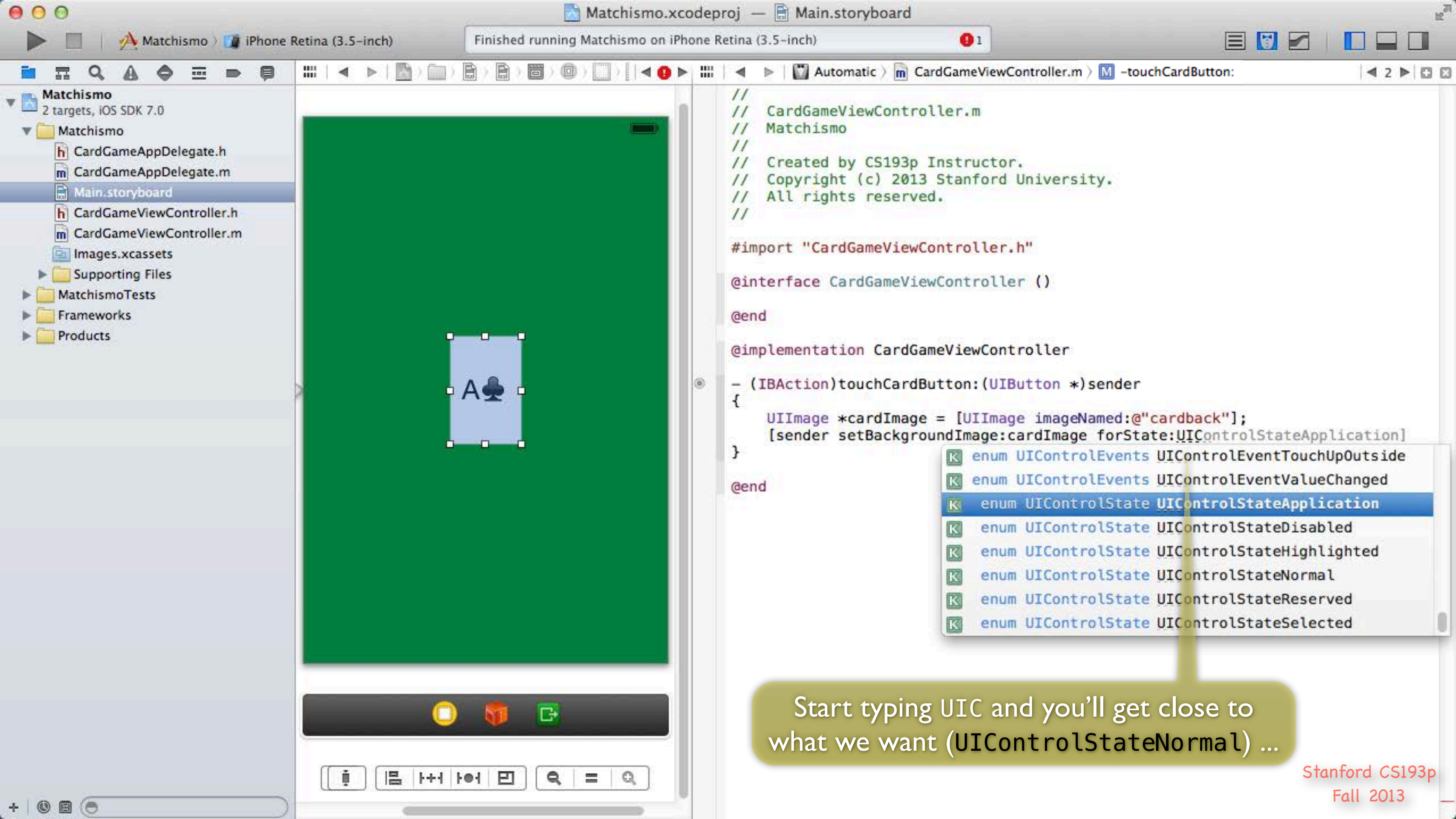
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



The setBackgroundImage: forState: method asks for the state because you can set different background images for selected, highlighted or disabled states of the UIButton.

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];  
    [sender setBackgroundImage:cardImage forState:(UIControlState)]  
}  
  
@end
```

We're just going to set the background for the default (Normal) state so it will be that way in all button states.



Finished running Matchismo on iPhone Retina (3.5-inch) 1

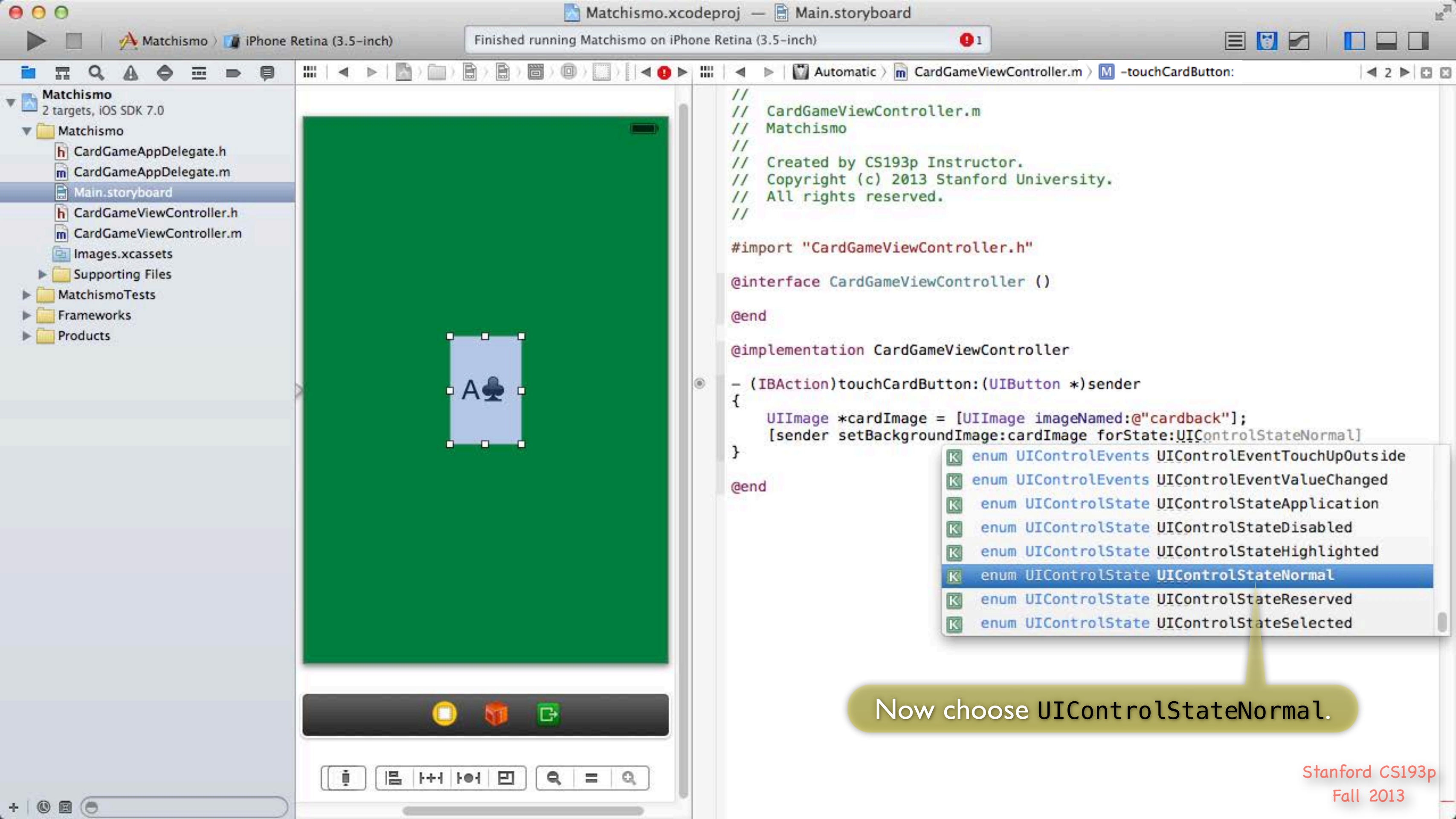
Automatic > CardGameViewController.m > -touchCardButton:

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];  
    [sender setBackgroundImage:cardImage forState:UIControlStateApplication]  
}  
  
@end
```

- enum UIControlEvents UIControlEventTouchUpOutside
- enum UIControlEvents UIControlEventValueChanged
- enum UIControlState **UIControlStateApplication**
- enum UIControlState UIControlStateDisabled
- enum UIControlState UIControlStateHighlighted
- enum UIControlState UIControlStateNormal
- enum UIControlState UIControlStateReserved
- enum UIControlState UIControlStateSelected

Start typing UIC and you'll get close to what we want (UIControlStateNormal) ...



Finished running Matchismo on iPhone Retina (3.5-inch) 1

Automatic > CardGameViewController.m > -touchCardButton:

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"
```

```
@interface CardGameViewController ()
```

```
@end
```

```
@implementation CardGameViewController
```

```
- (IBAction)touchCardButton:(UIButton *)sender
```

```
{
```

```
    UIImage *cardImage = [UIImage imageNamed:@"cardback"];
```

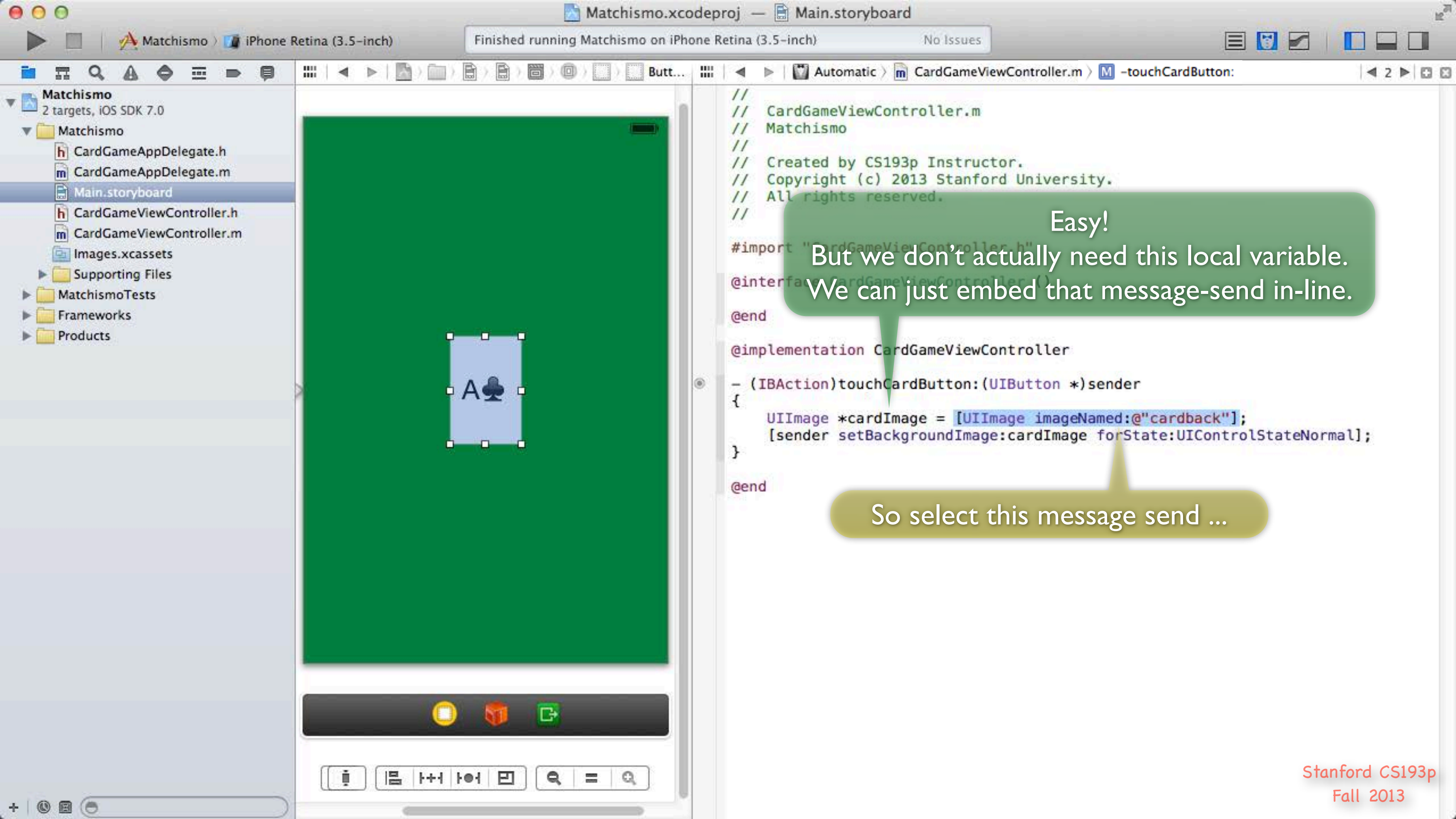
```
    [sender setBackgroundImage:cardImage forState:UIControlStateNormal]
```

```
}
```

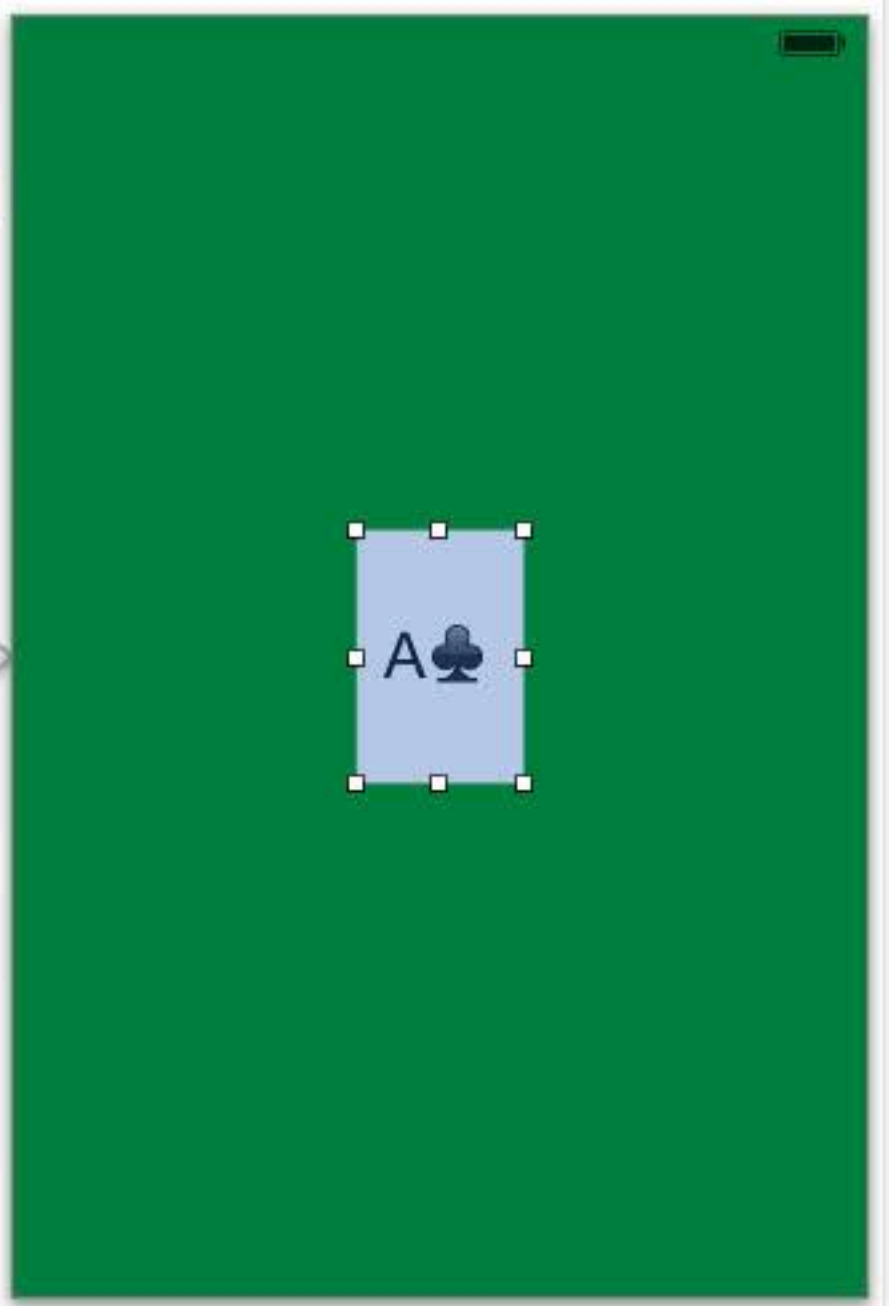
```
@end
```

- enum UIControlEvents UIControlEventTouchUpInside
- enum UIControlEvents UIControlEventValueChanged
- enum UIControlState UIControlStateApplication
- enum UIControlState UIControlStateDisabled
- enum UIControlState UIControlStateHighlighted
- enum UIControlState **UIControlStateNormal**
- enum UIControlState UIControlStateReserved
- enum UIControlState UIControlStateSelected

Now choose UIControlStateNormal.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

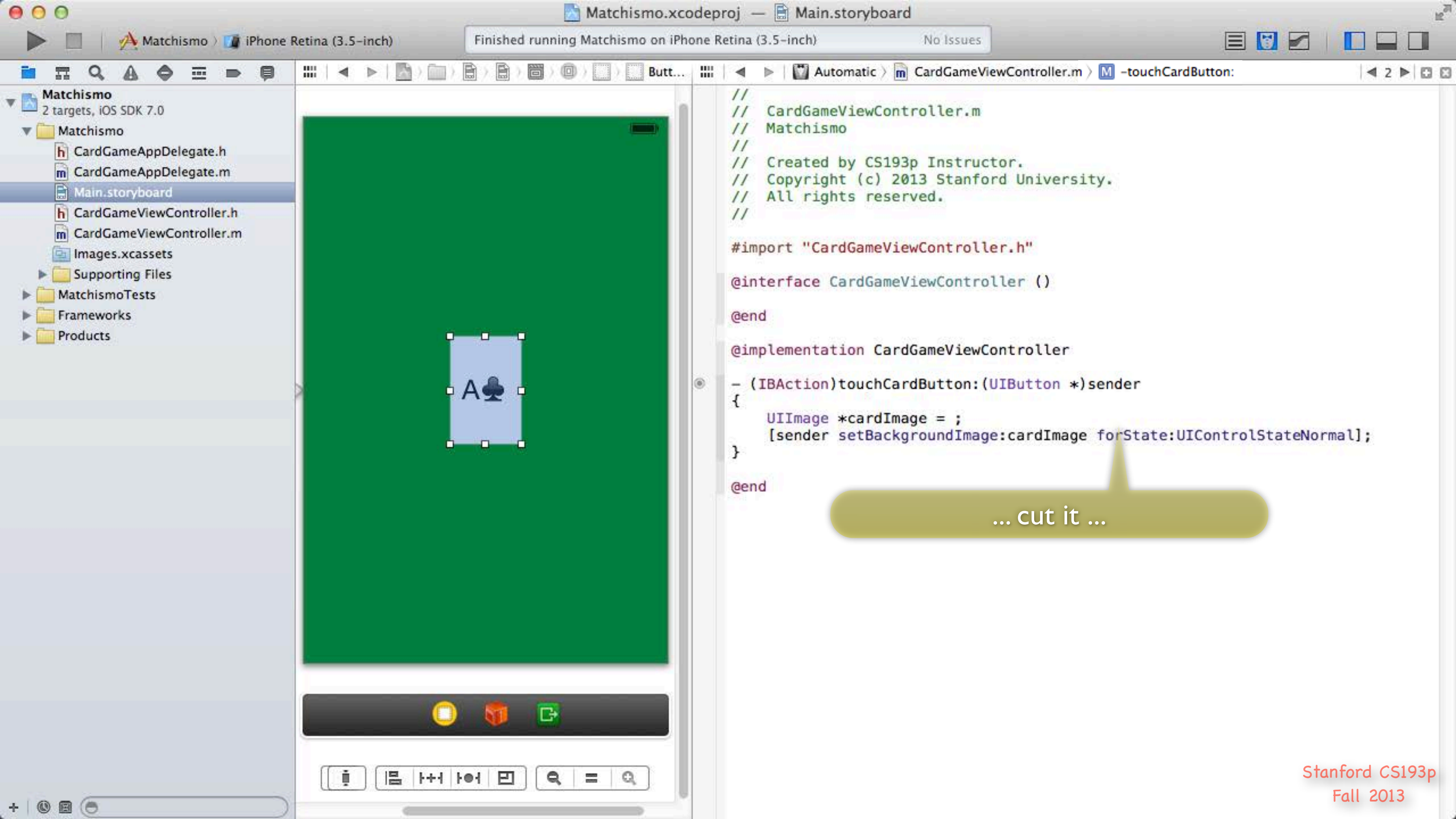
- (IBAction)touchCardButton:(UIButton *)sender
{
  UIImage *cardImage = [UIImage imageNamed:@"cardback"];
  [sender setBackgroundImage:cardImage forState:UIControlStateNormal];
}

@end
```

Easy!  
But we don't actually need this local variable.  
We can just embed that message-send in-line.

So select this message send ...

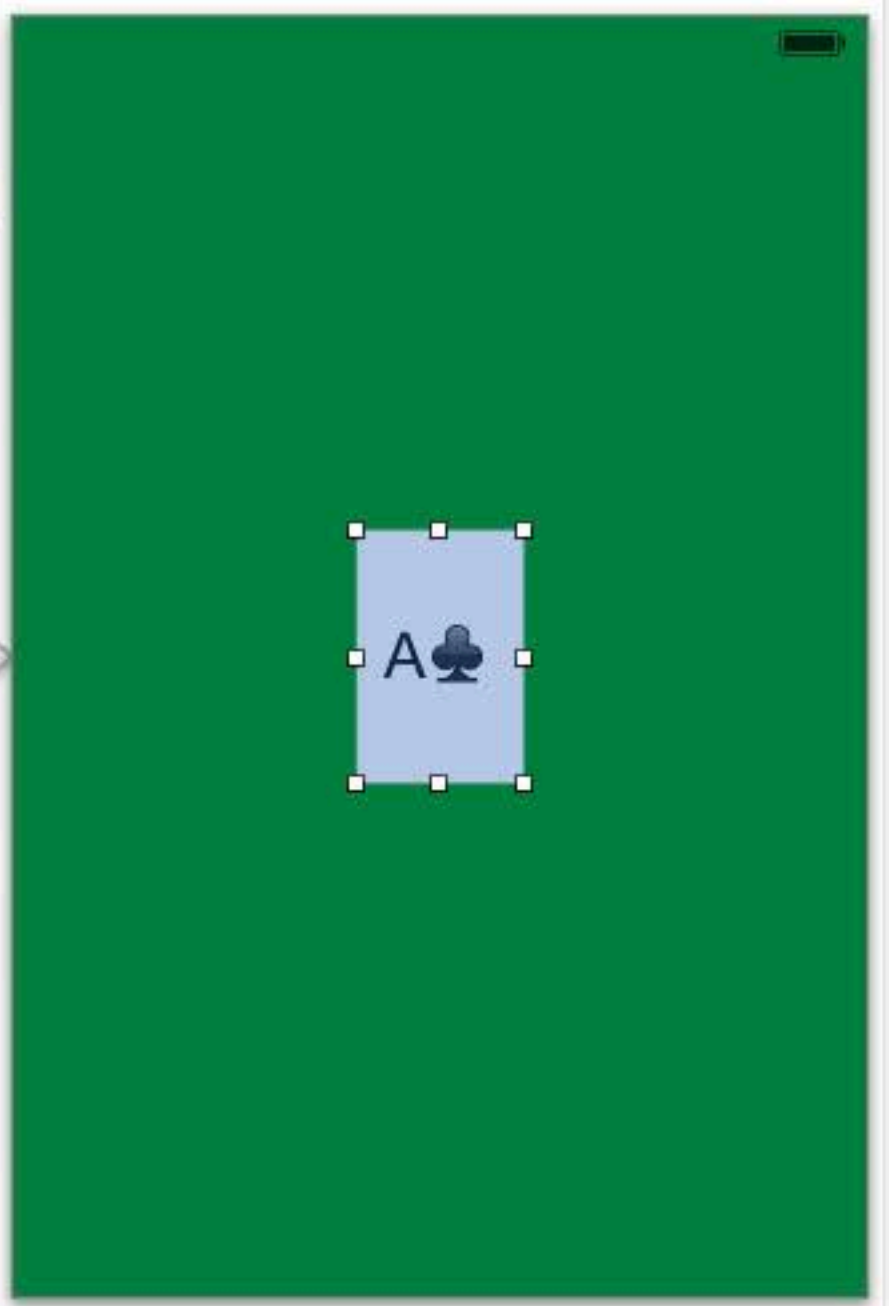




Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

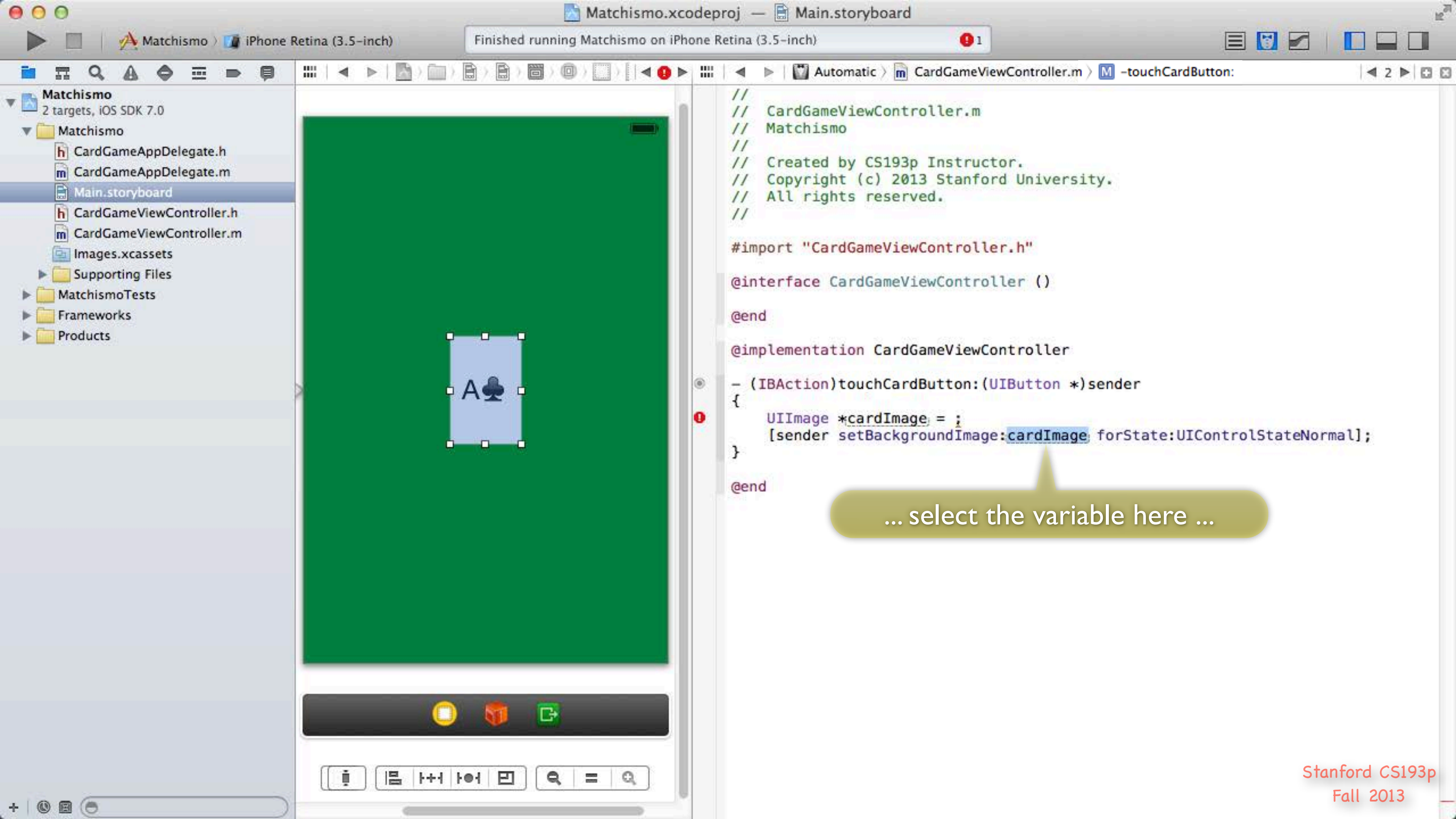
Matchismo iPhone Retina (3.5-inch)

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = ;  
    [sender setBackgroundImage:cardImage forState:UIControlStateNormal];  
}  
  
@end
```

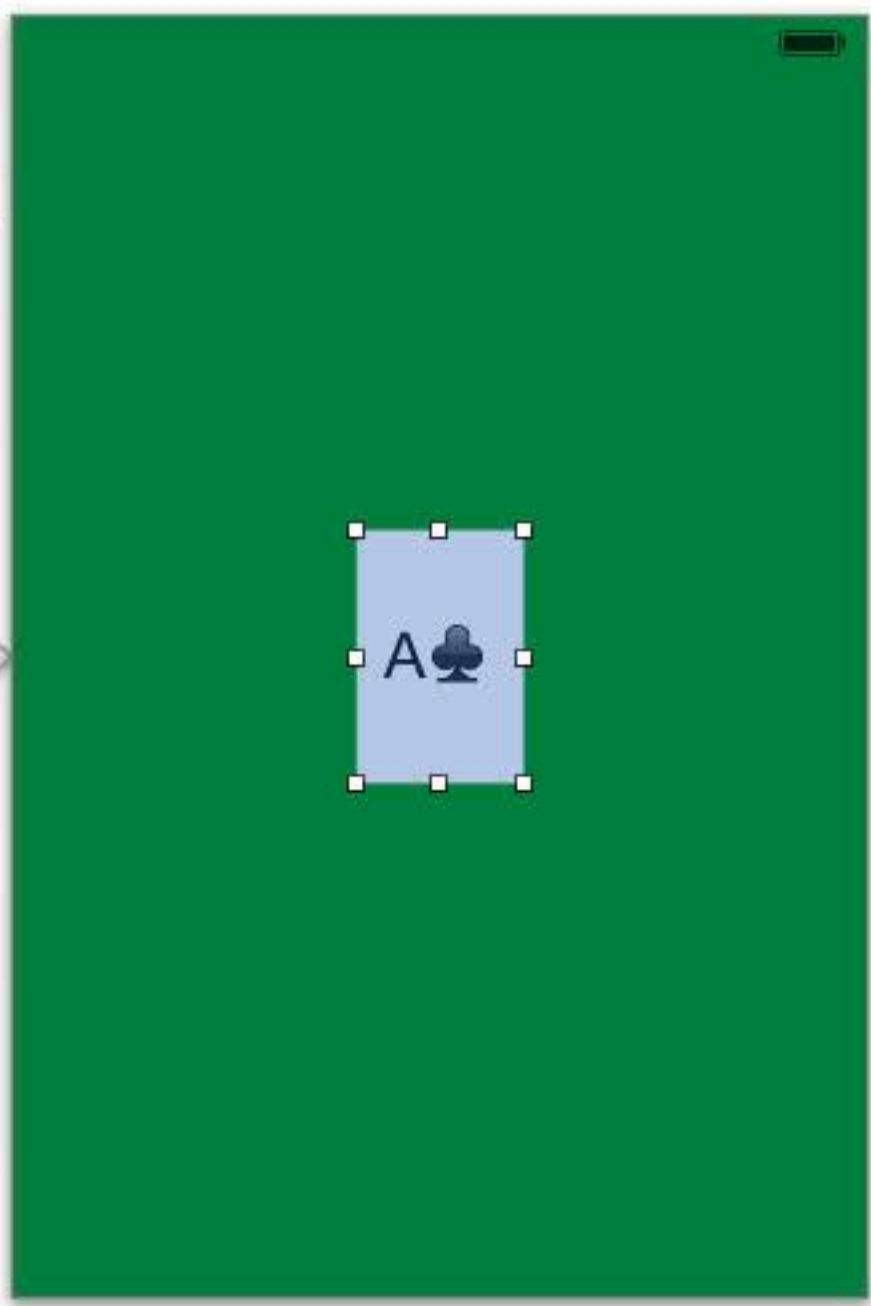
... cut it ...



Finished running Matchismo on iPhone Retina (3.5-inch) 1

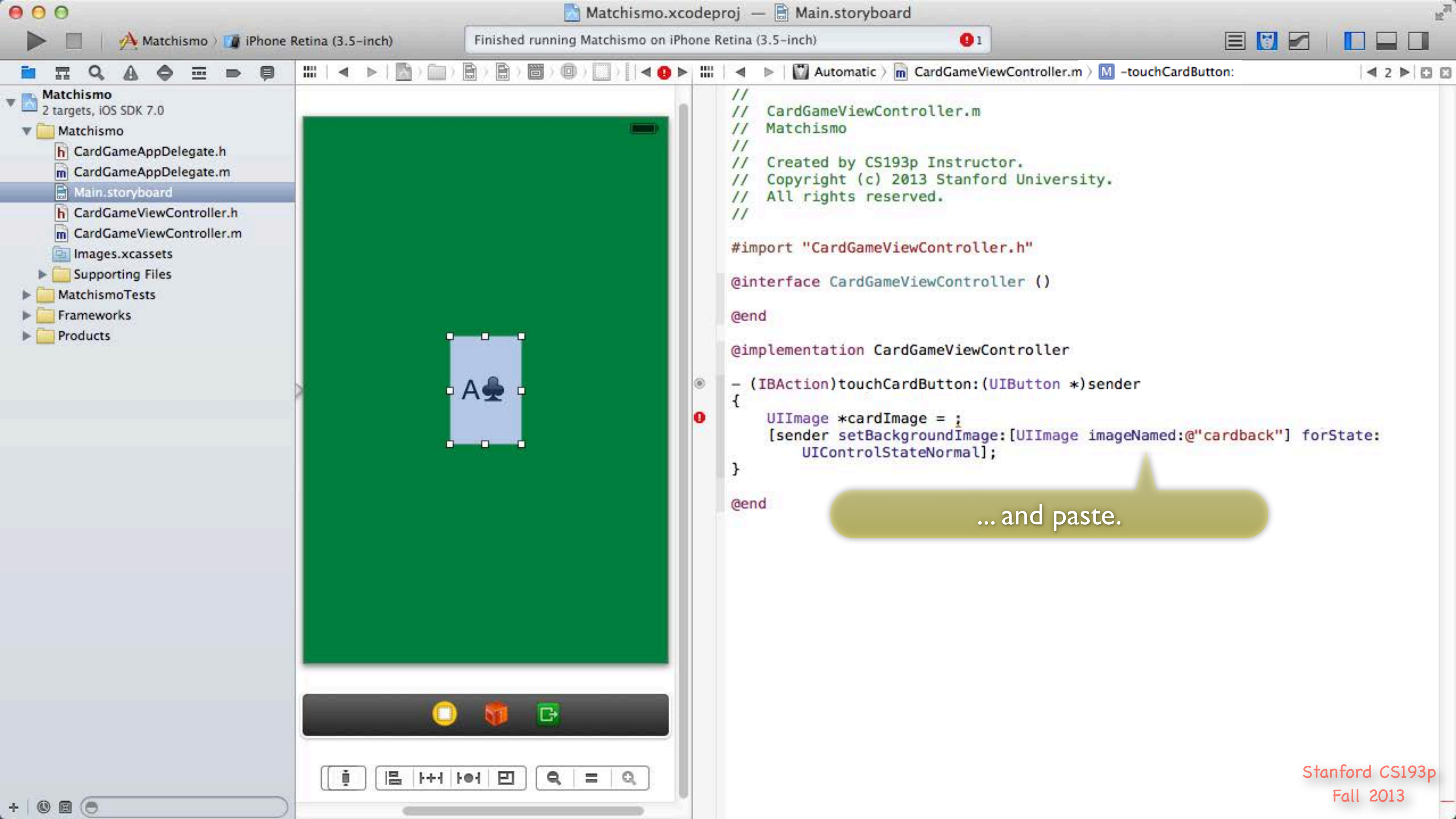
Automatic > CardGameViewController.m > -touchCardButton:

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



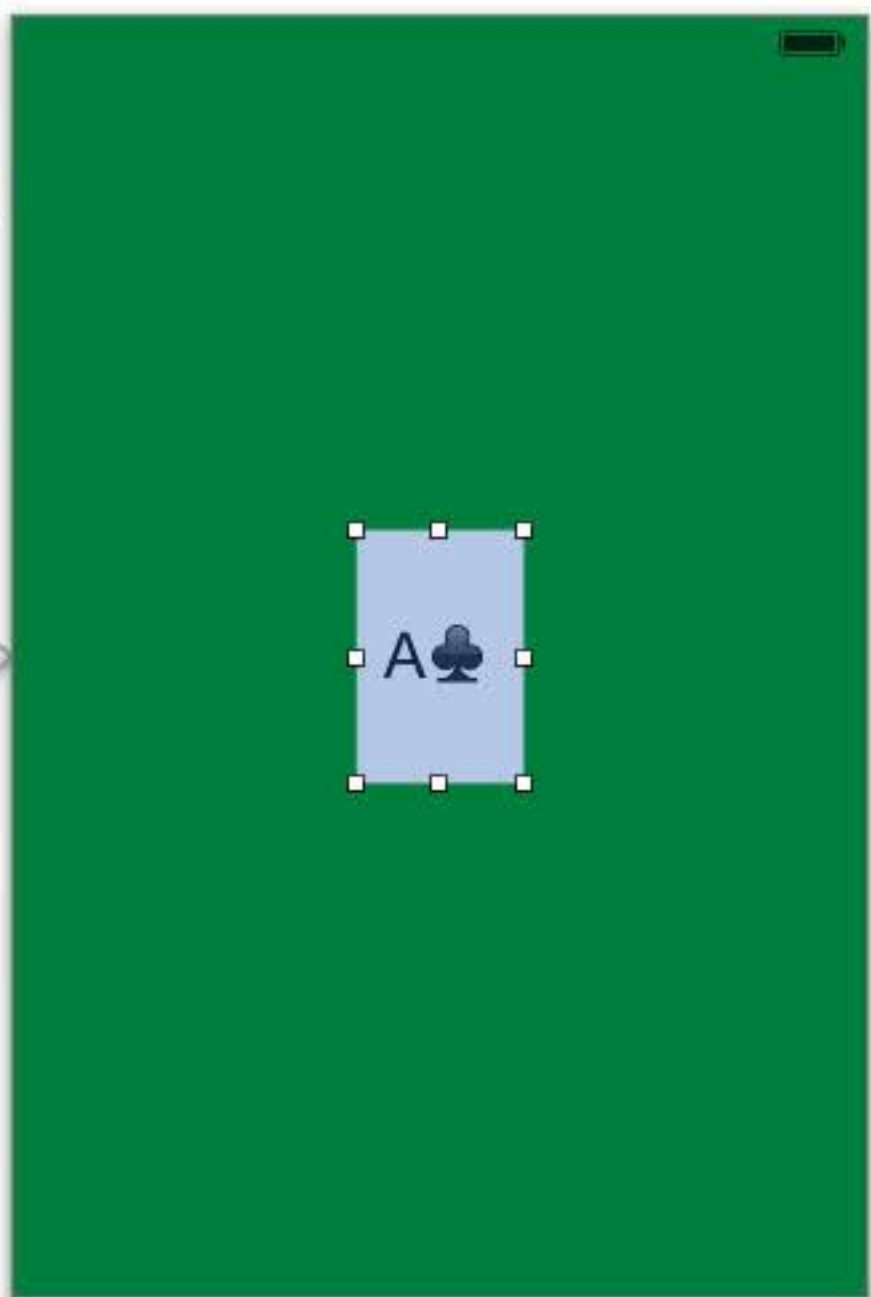
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = ;  
    [sender setBackgroundImage:cardImage forState:UIControlStateNormal];  
}  
  
@end
```

... select the variable here ...



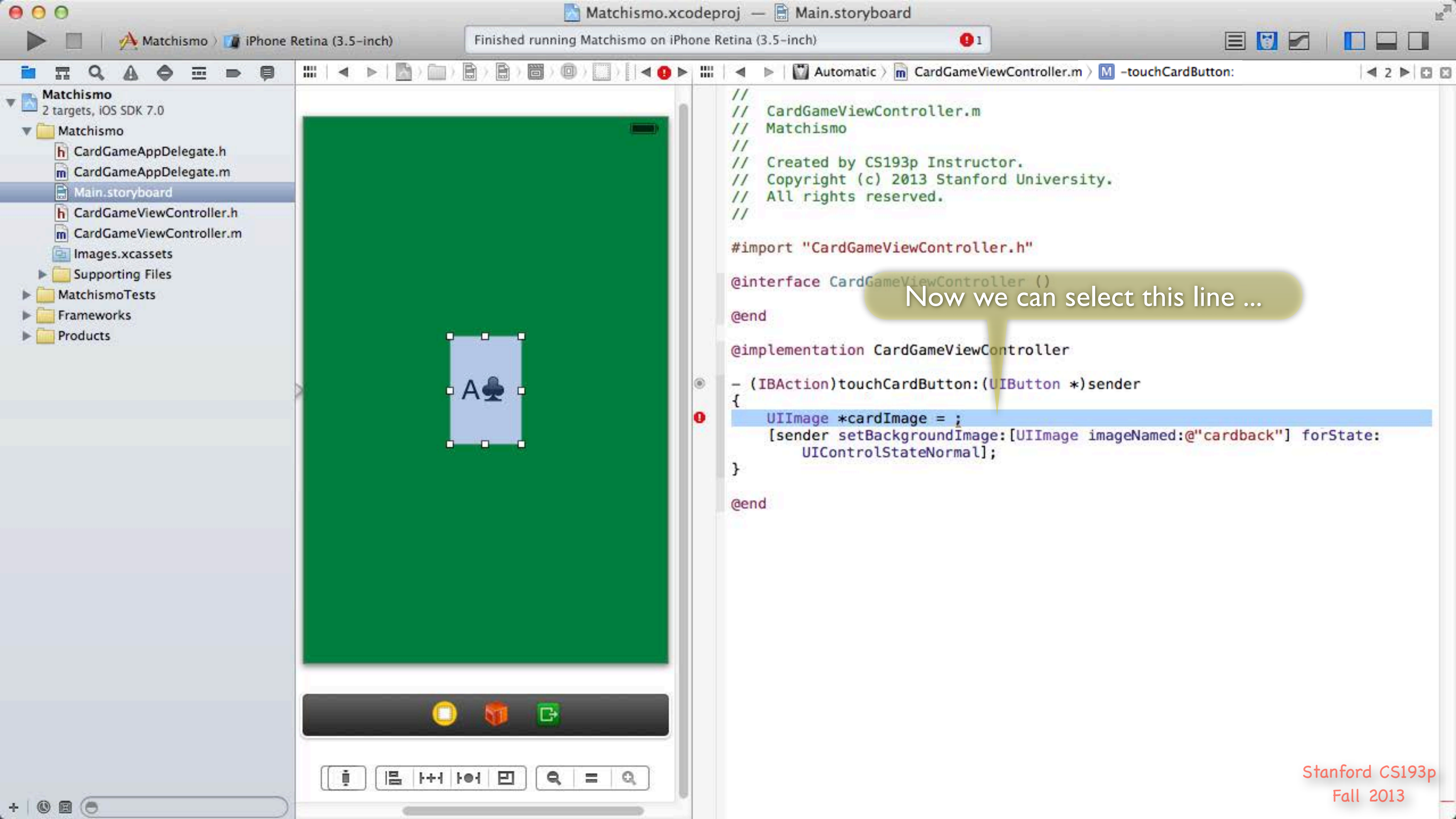
Finished running Matchismo on iPhone Retina (3.5-inch) 1

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = ;  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"] forState:  
        UIControlStateNormal];  
}  
  
@end
```

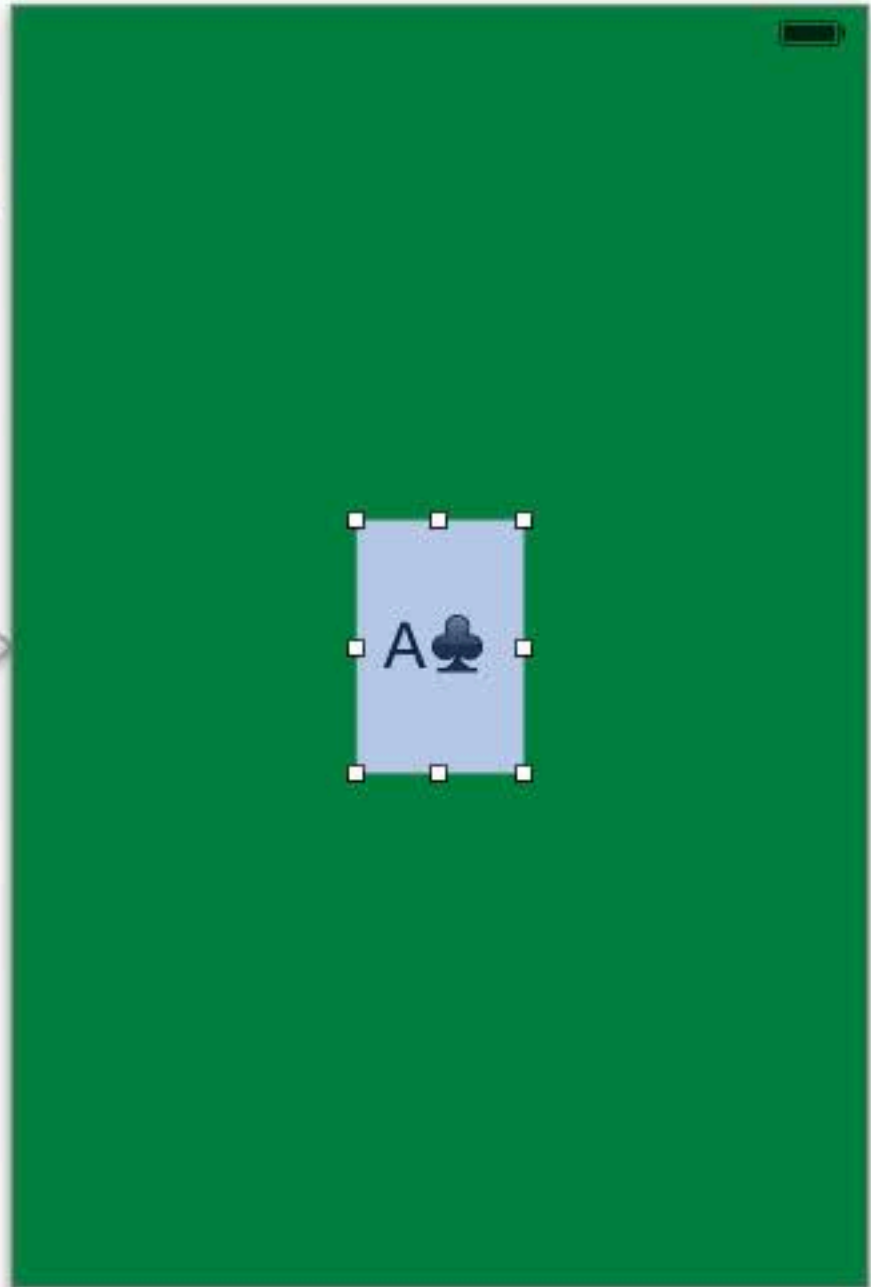
... and paste.



Finished running Matchismo on iPhone Retina (3.5-inch) 1

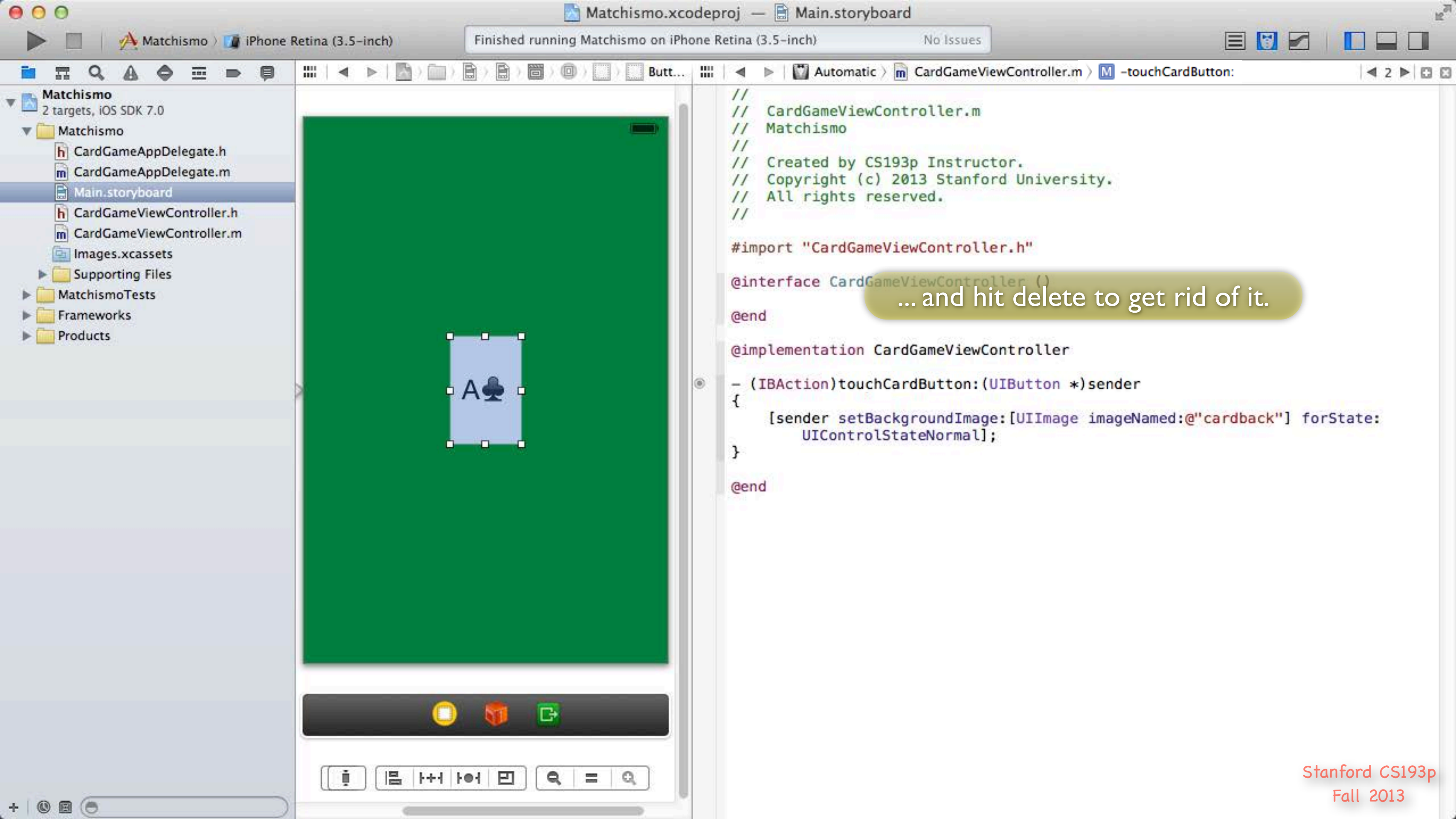
Automatic > CardGameViewController.m > -touchCardButton:

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



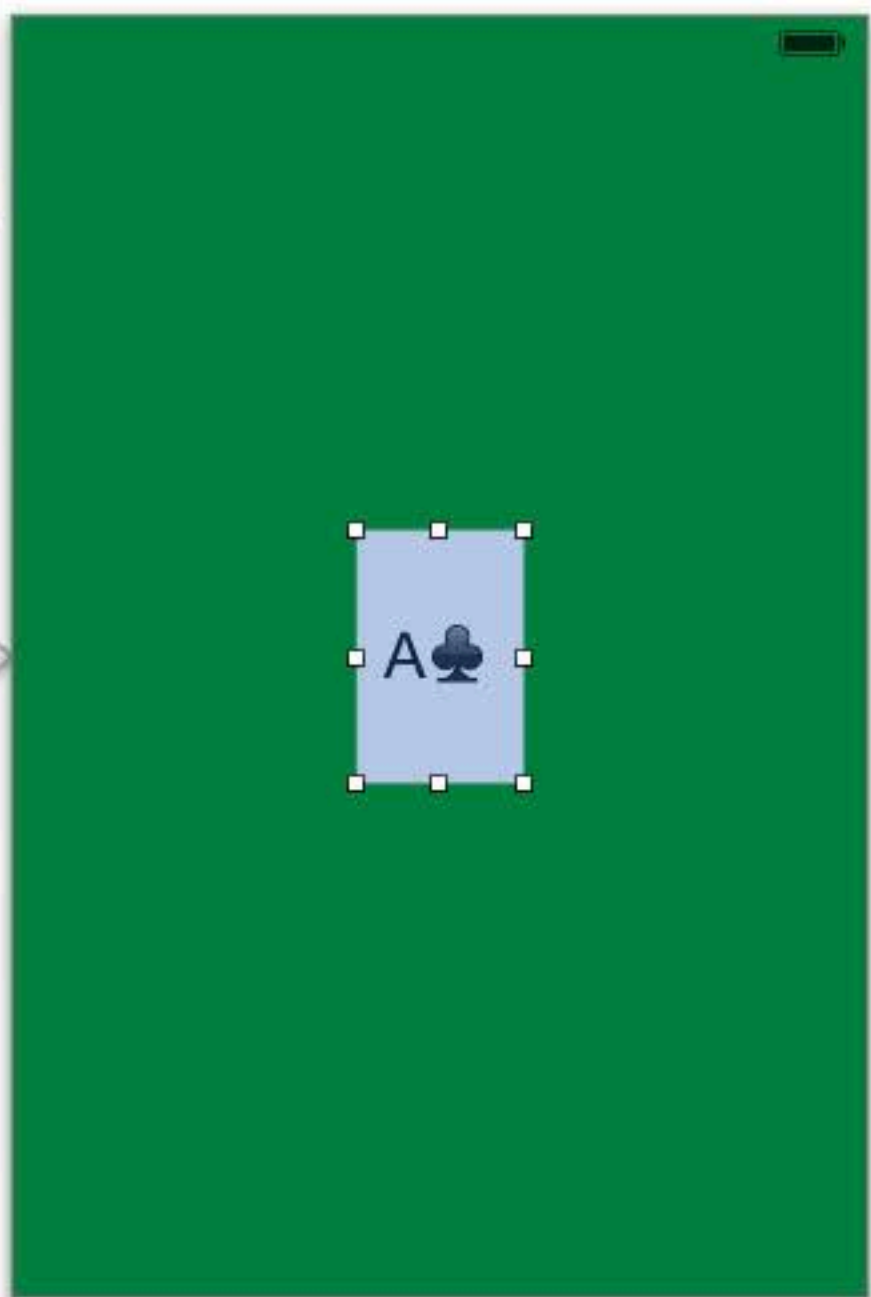
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    UIImage *cardImage = ;  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"] forState:  
        UIControlStateNormal];  
}  
  
@end
```

Now we can select this line ...

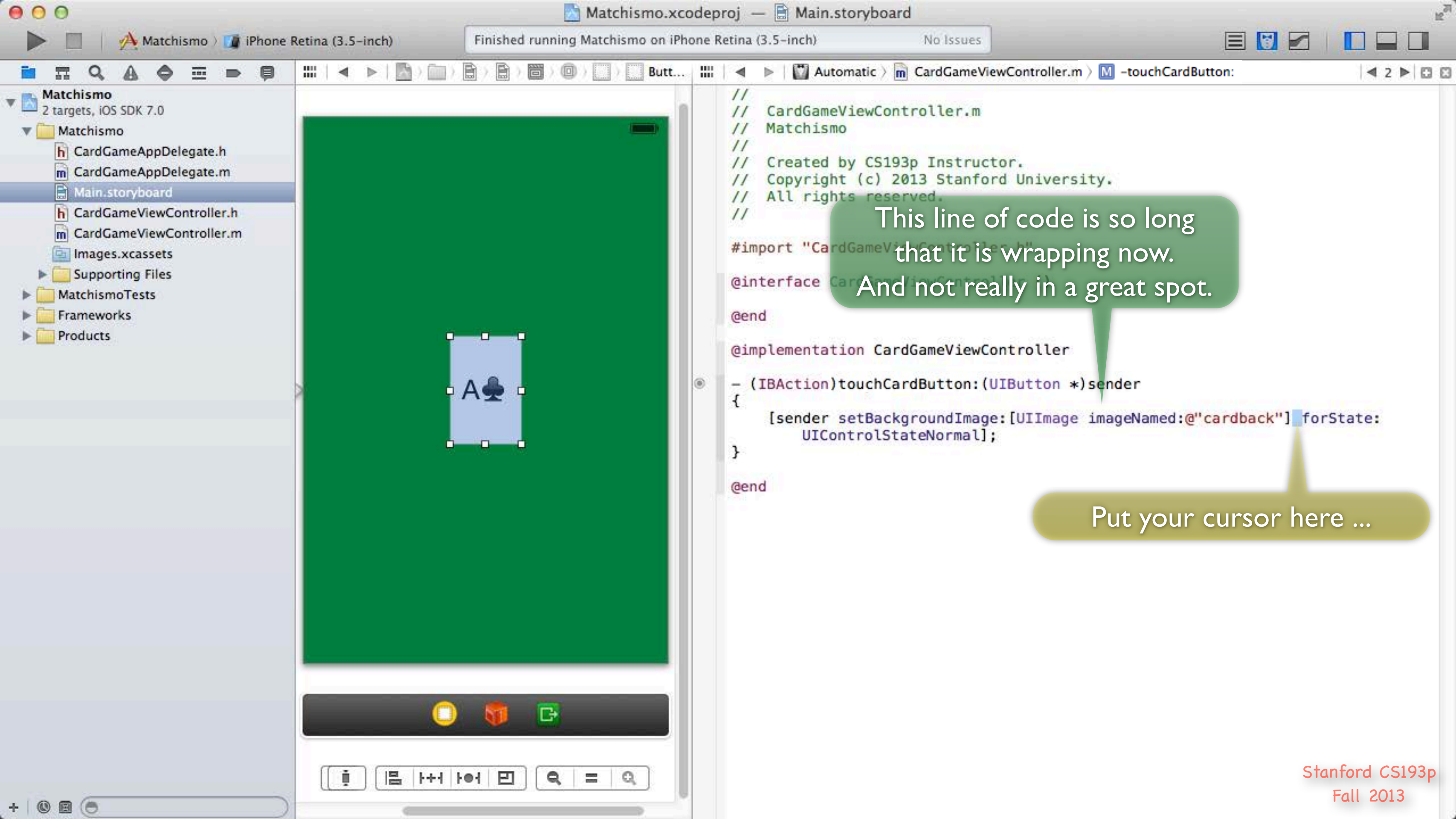


Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

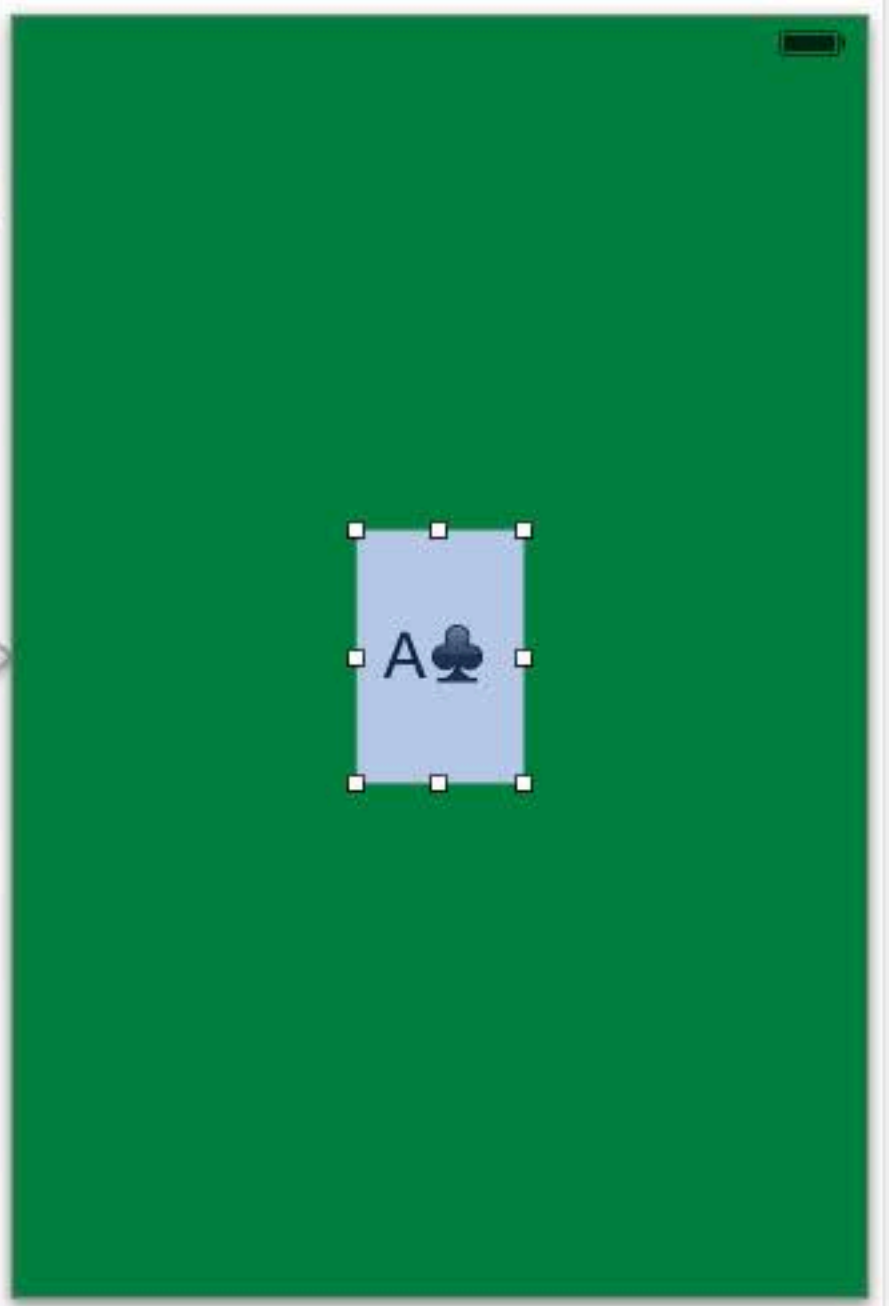
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
... and hit delete to get rid of it.  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"] forState:  
        UIControlStateNormal];  
}  
  
@end
```



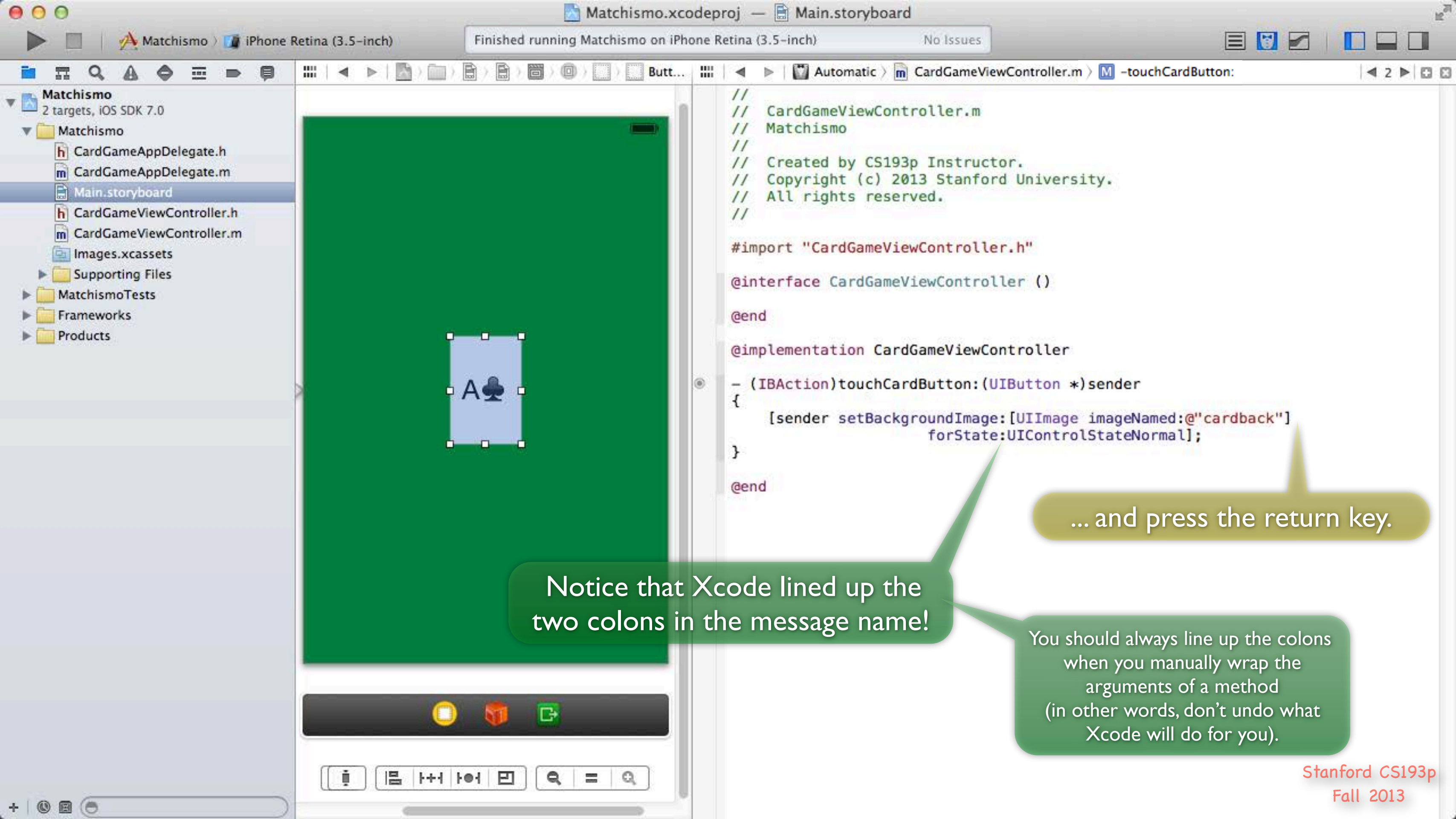
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]] forState:  
        UIControlStateNormal];  
}  
  
@end
```

This line of code is so long  
that it is wrapping now.  
And not really in a great spot.

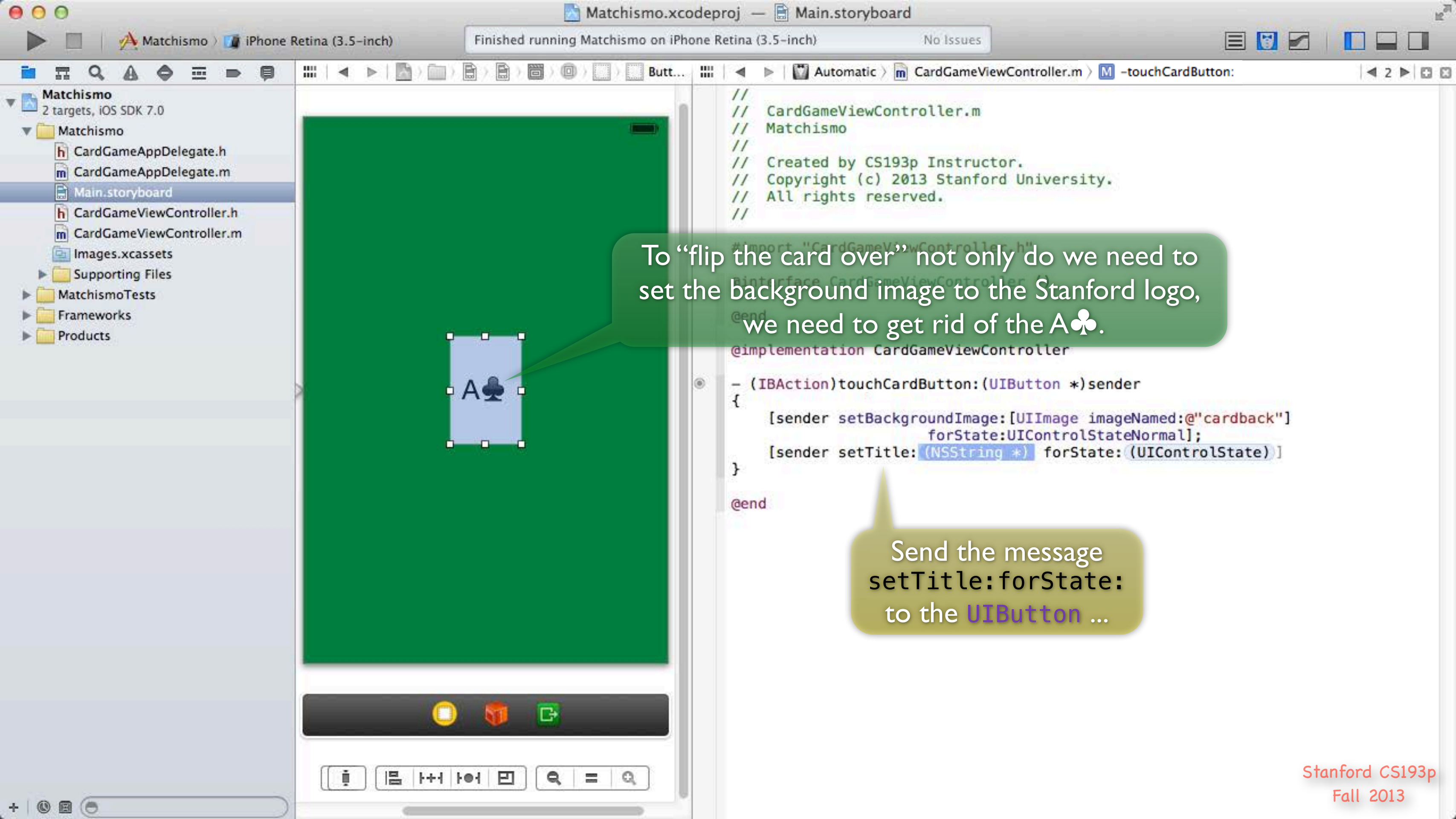
Put your cursor here ...



Notice that Xcode lined up the two colons in the message name!

... and press the return key.

You should always line up the colons when you manually wrap the arguments of a method (in other words, don't undo what Xcode will do for you).



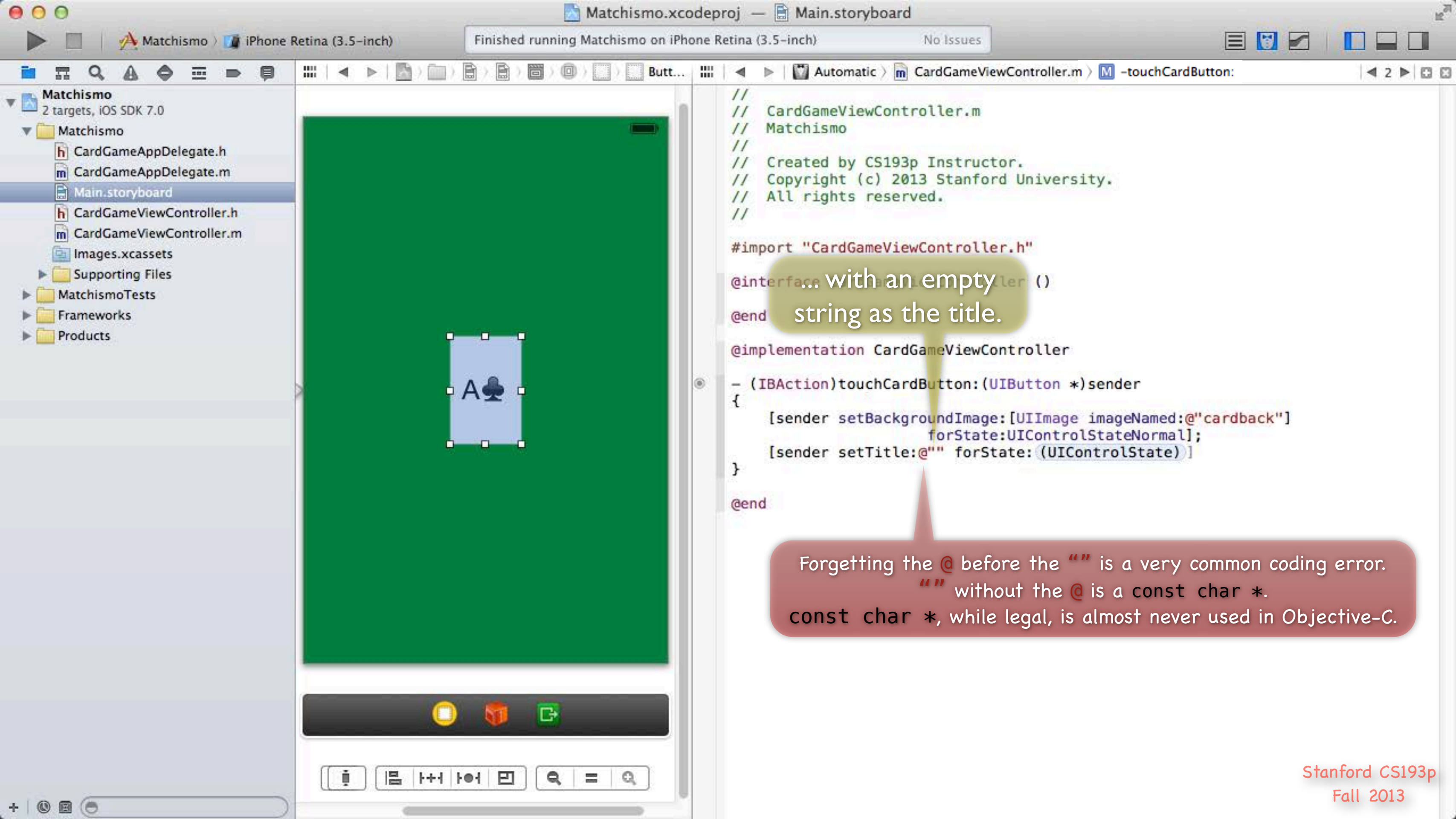
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

To “flip the card over” not only do we need to set the background image to the Stanford logo, we need to get rid of the A♣.

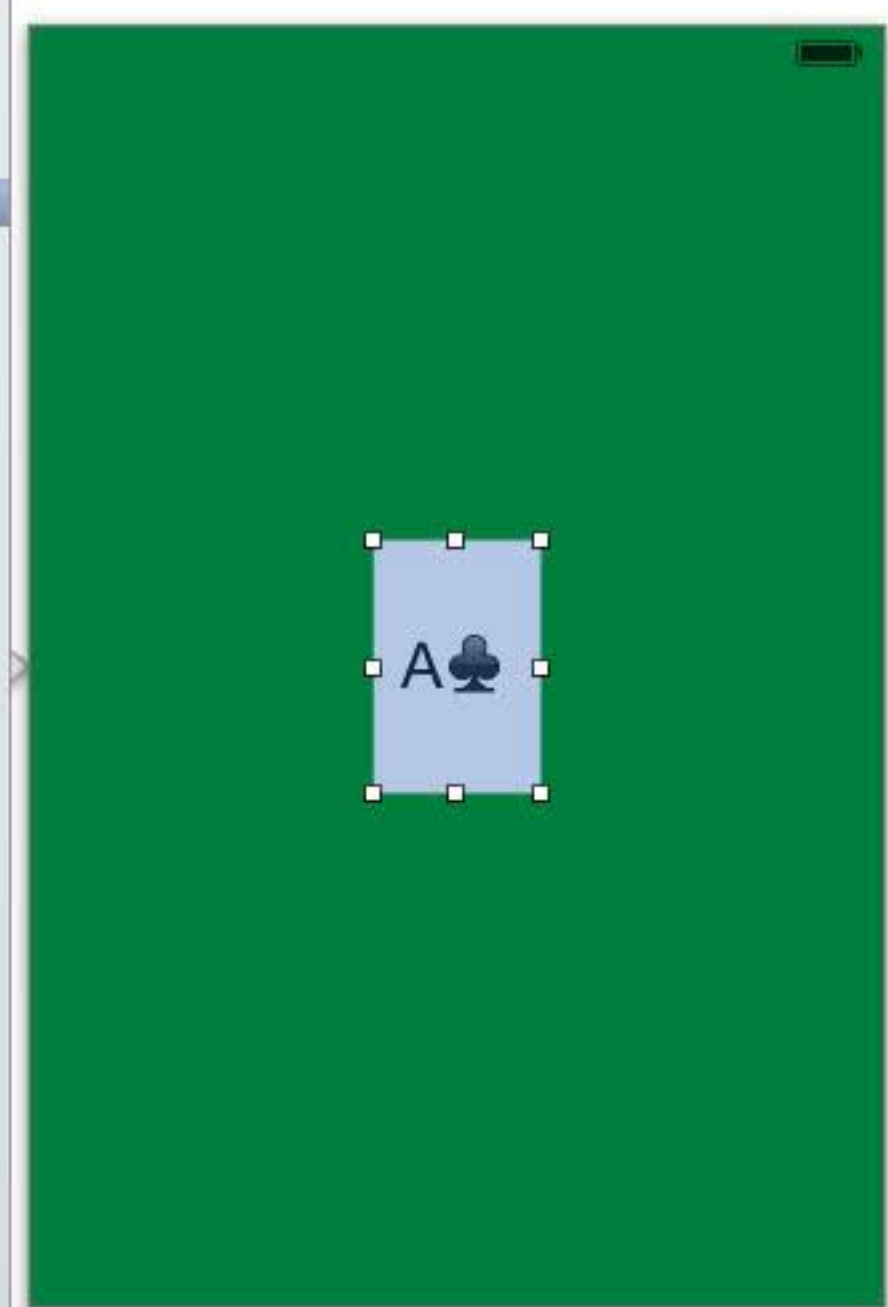
Send the message setTitle:forState: to the UIButton ...

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
#interface CardGameViewController  
@end  
@implementation CardGameViewController  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:(NSString *) forState:(UIControlState)]  
}  
@end
```





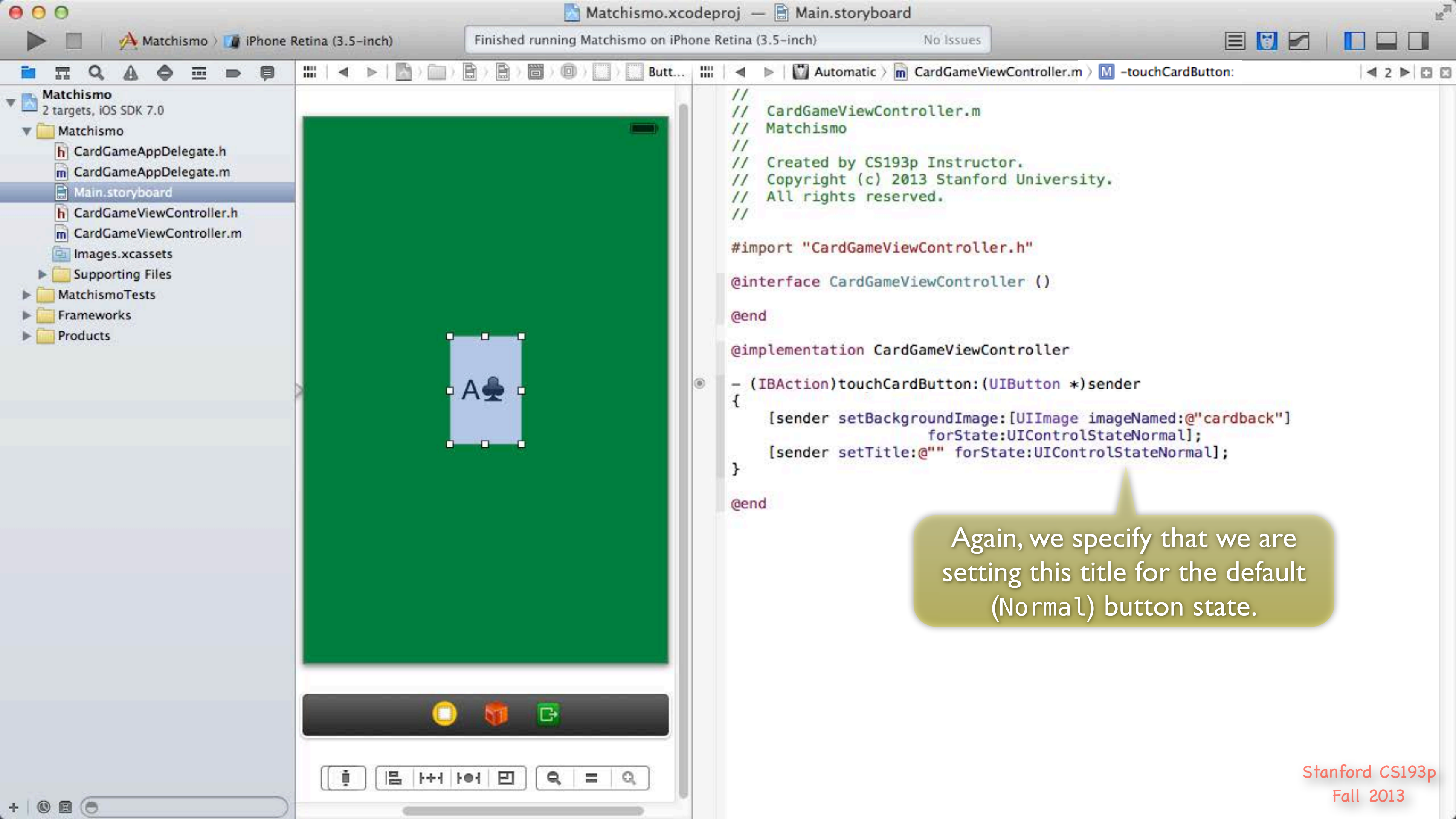
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



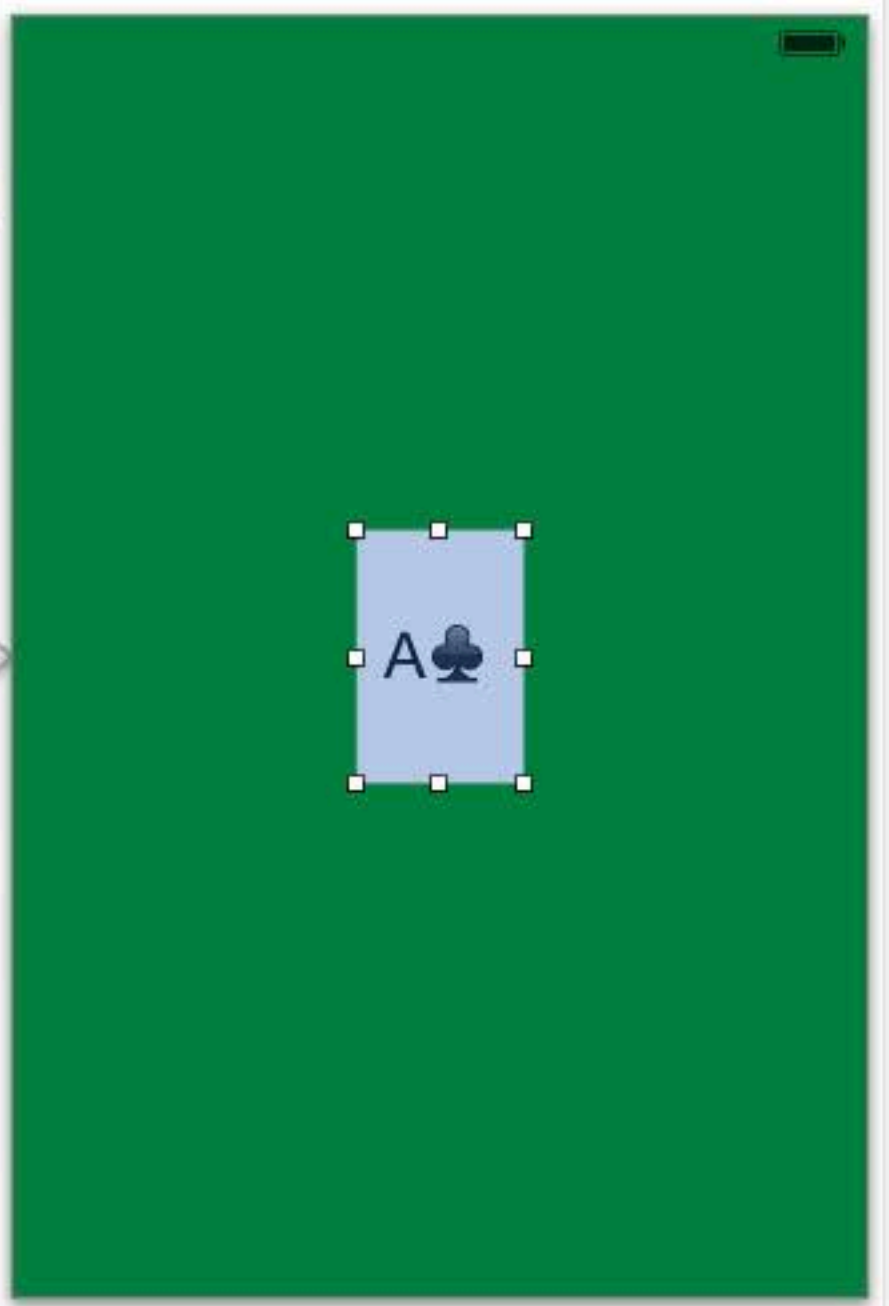
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:(UIControlState)]  
}  
  
@end
```

... with an empty string as the title.

Forgetting the @ before the "" is a very common coding error. "" without the @ is a const char \*. const char \*, while legal, is almost never used in Objective-C.

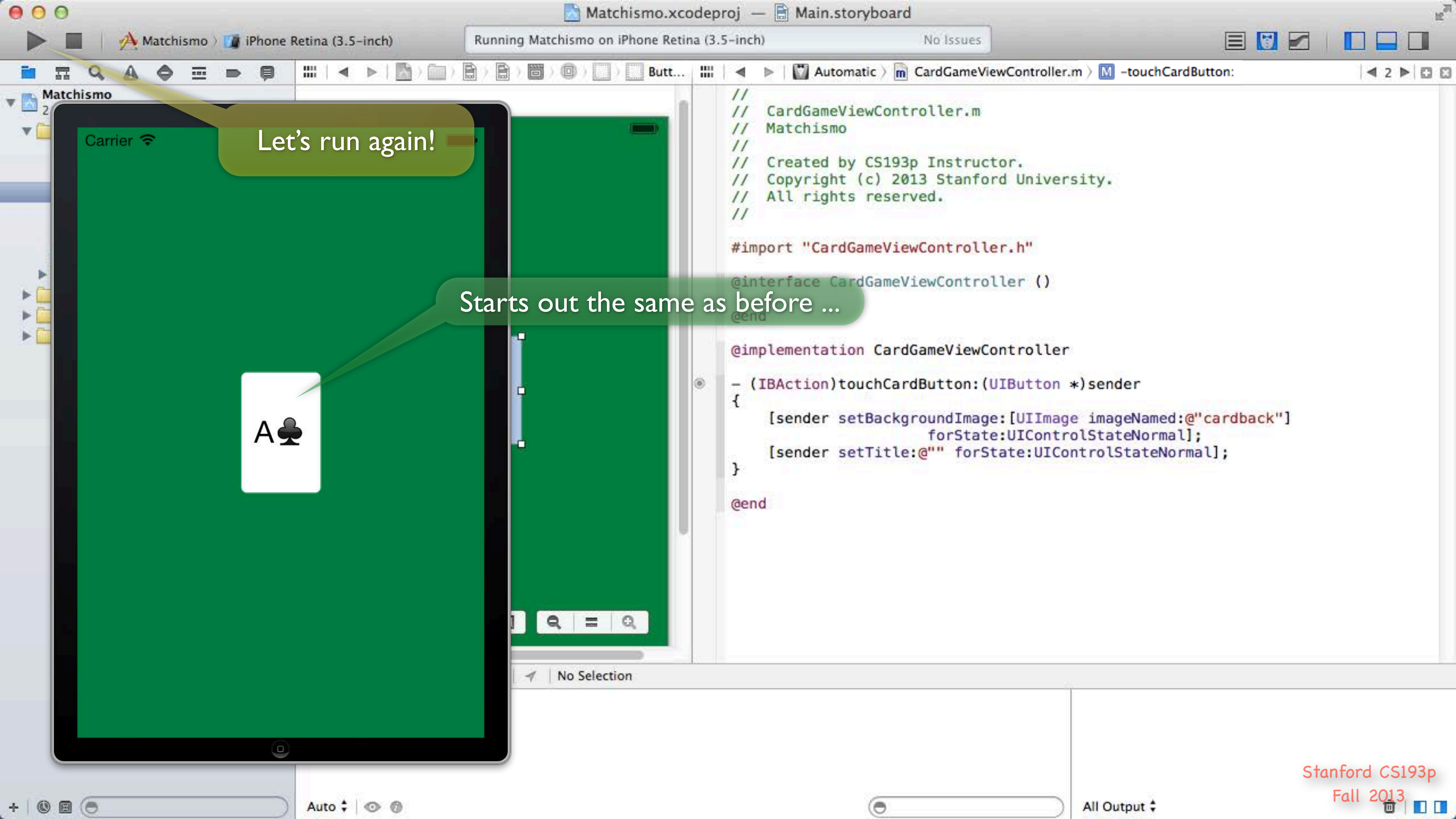


- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:UIControlStateNormal];  
}  
  
@end
```

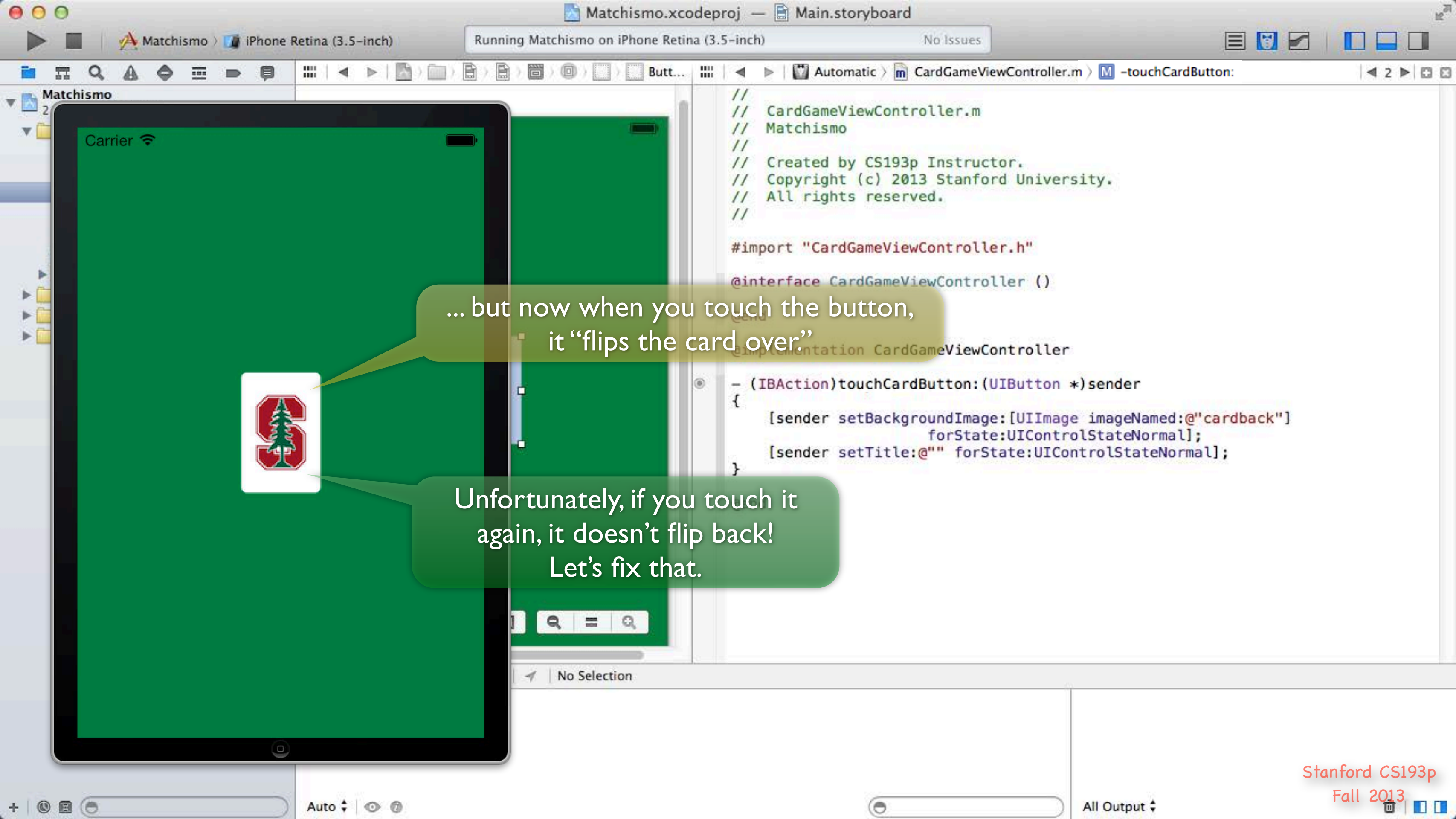
Again, we specify that we are setting this title for the default (Normal) button state.



Let's run again!

Starts out the same as before ...

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:UIControlStateNormal];  
}  
  
@end
```

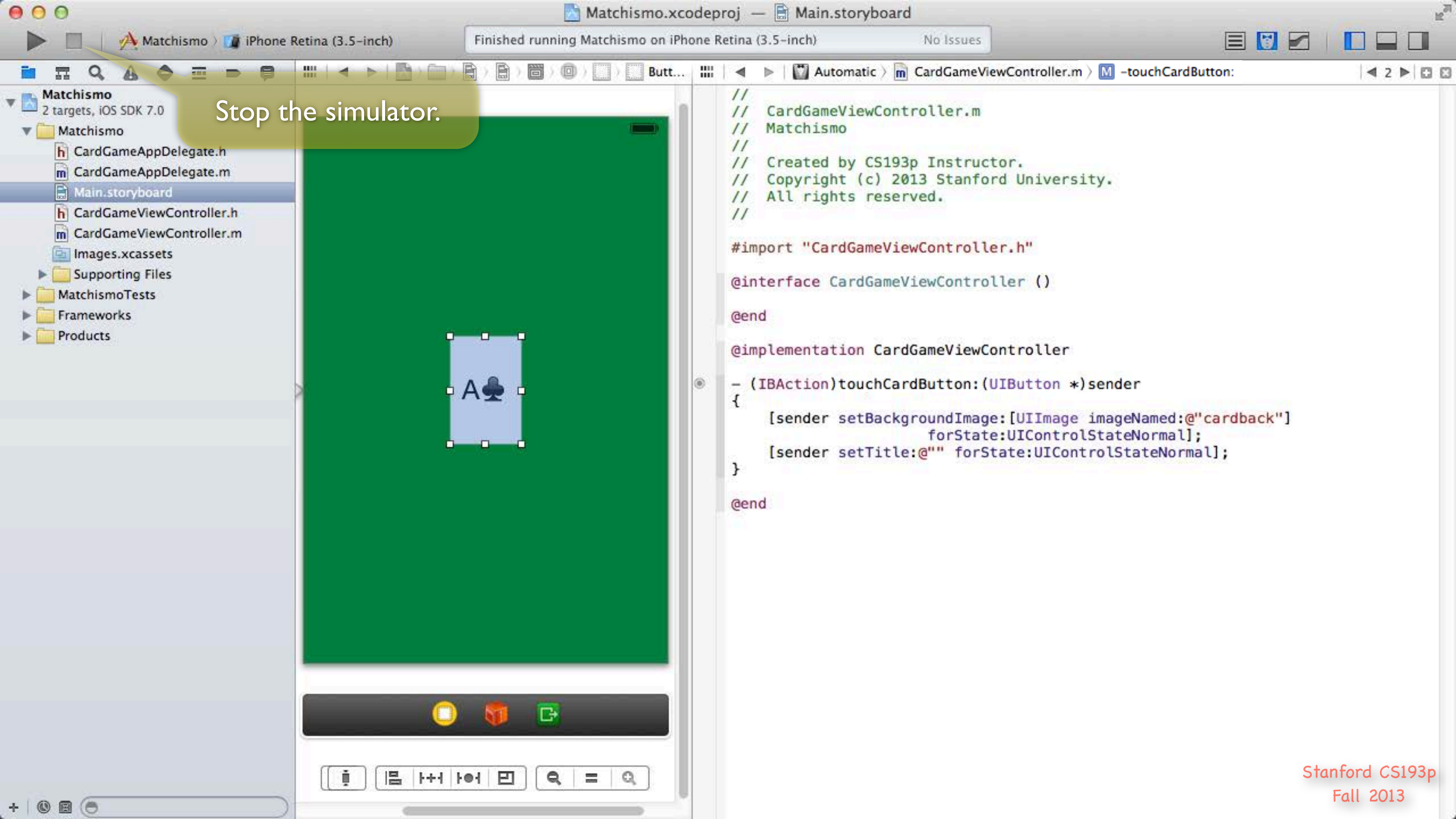


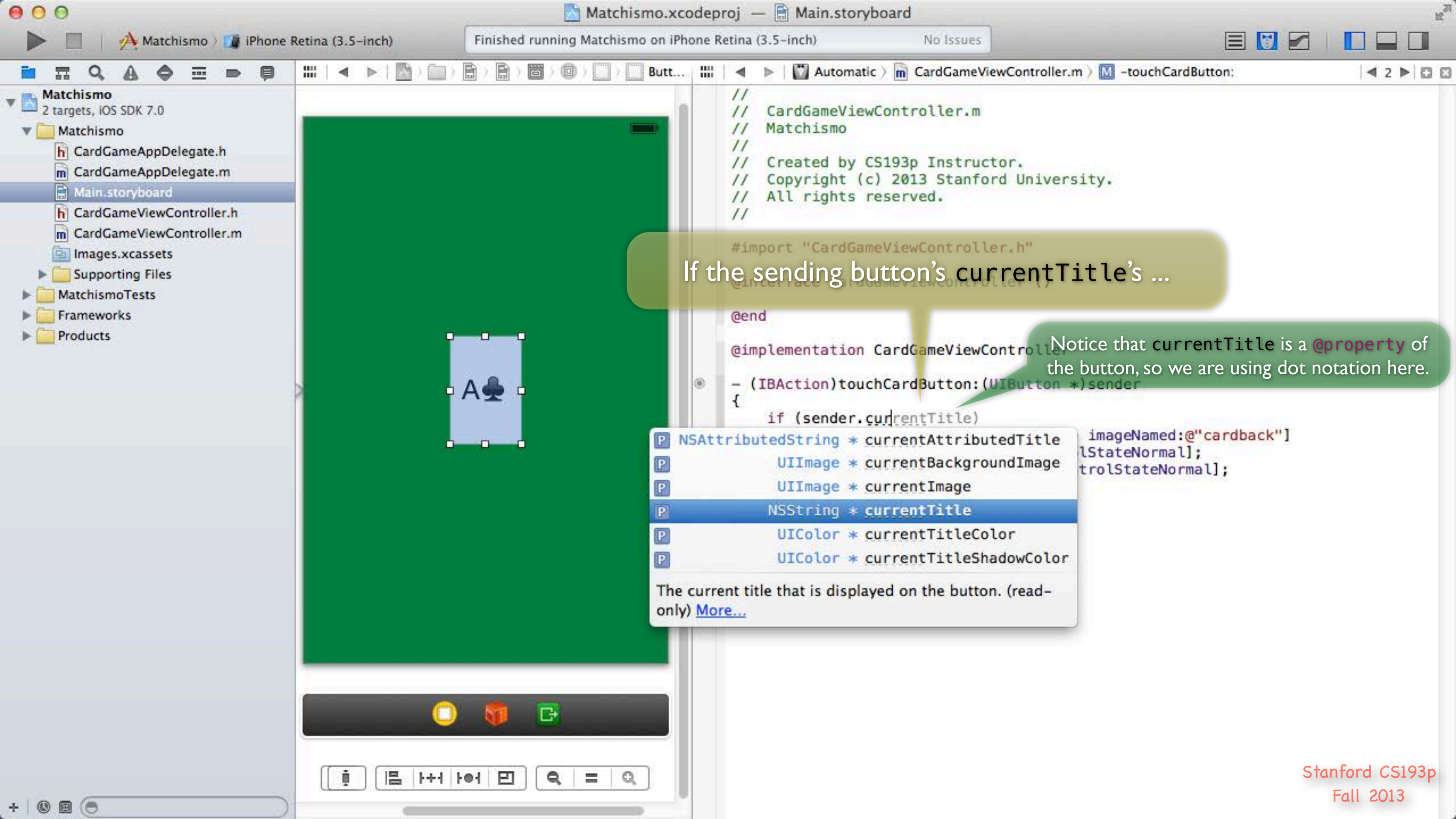
... but now when you touch the button,  
it "flips the card over."

Unfortunately, if you touch it  
again, it doesn't flip back!  
Let's fix that.

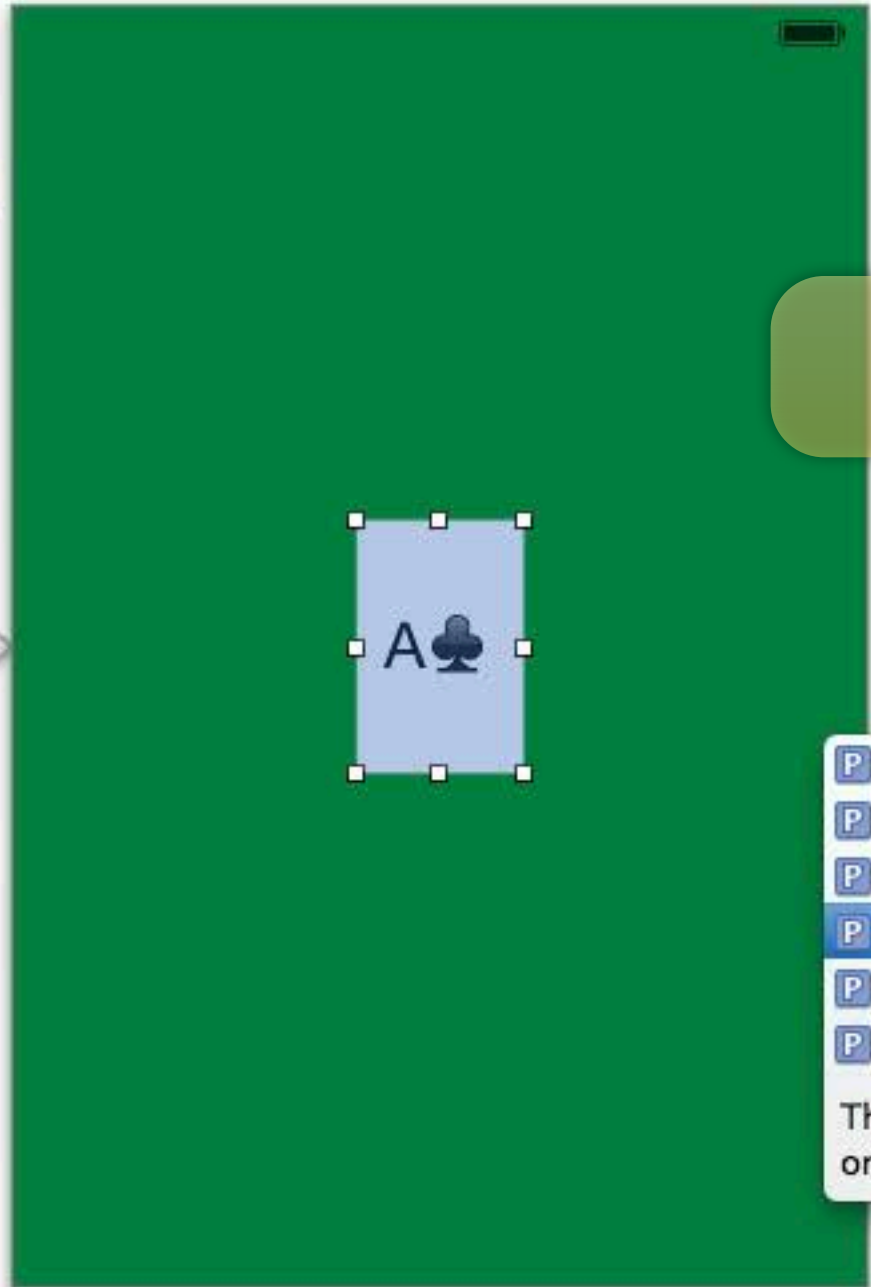
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:UIControlStateNormal];  
}  
}
```

Stop the simulator.





- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



If the sending button's currentTitle's ...

Notice that currentTitle is a @property of the button, so we are using dot notation here.

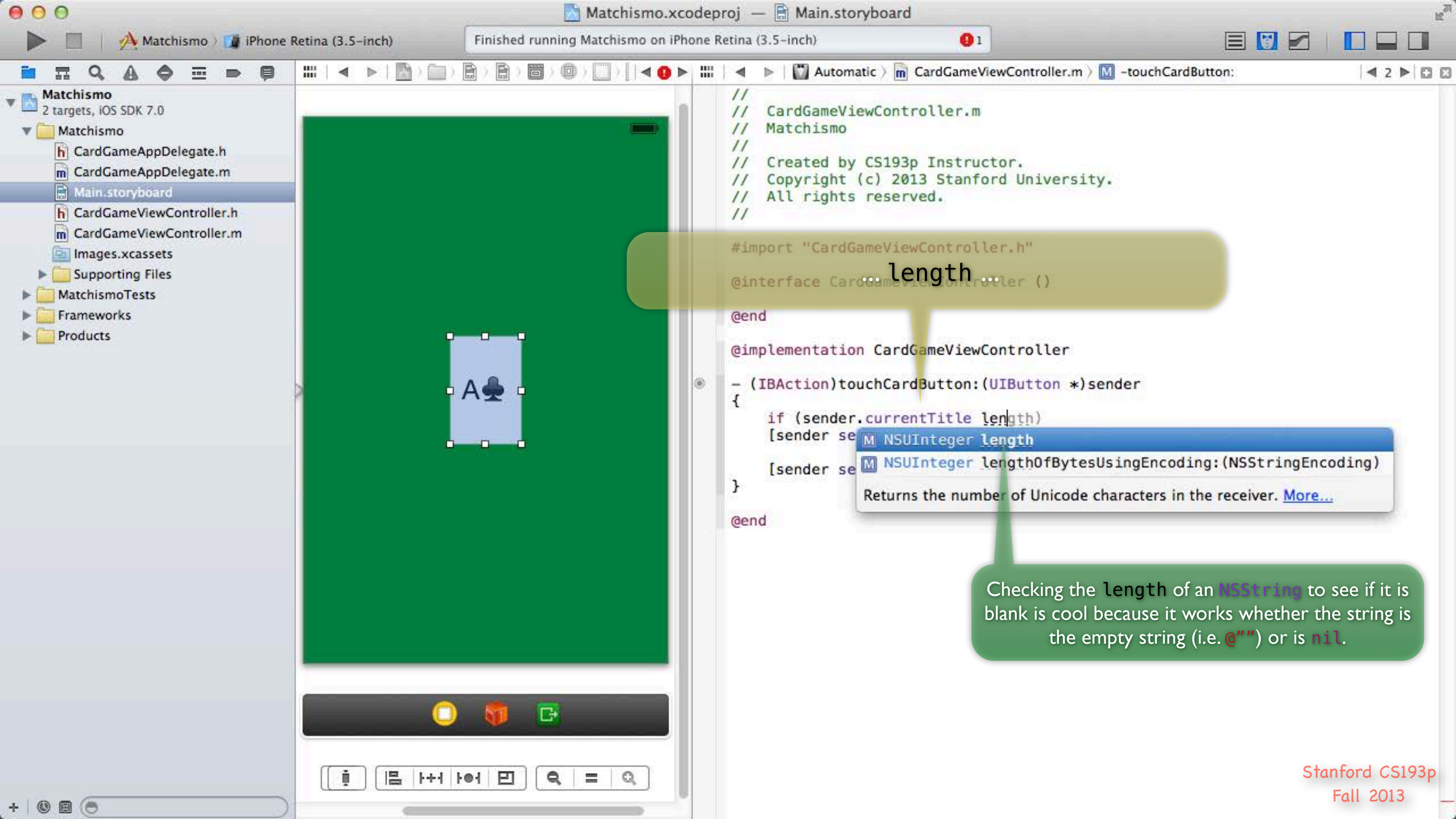
- NSAttributedString \* currentAttributedTitle
- UIImage \* currentBackgroundImage
- UIImage \* currentImage
- NSString \* currentTitle**
- UIColor \* currentTitleColor
- UIColor \* currentTitleShadowColor

The current title that is displayed on the button. (read-only) [More...](#)

```

imageName:@"cardback"]
[StateNormal];
trolStateNormal];

```



Finished running Matchismo on iPhone Retina (3.5-inch)

Automatic > CardGameViewController.m > -touchCardButton:

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

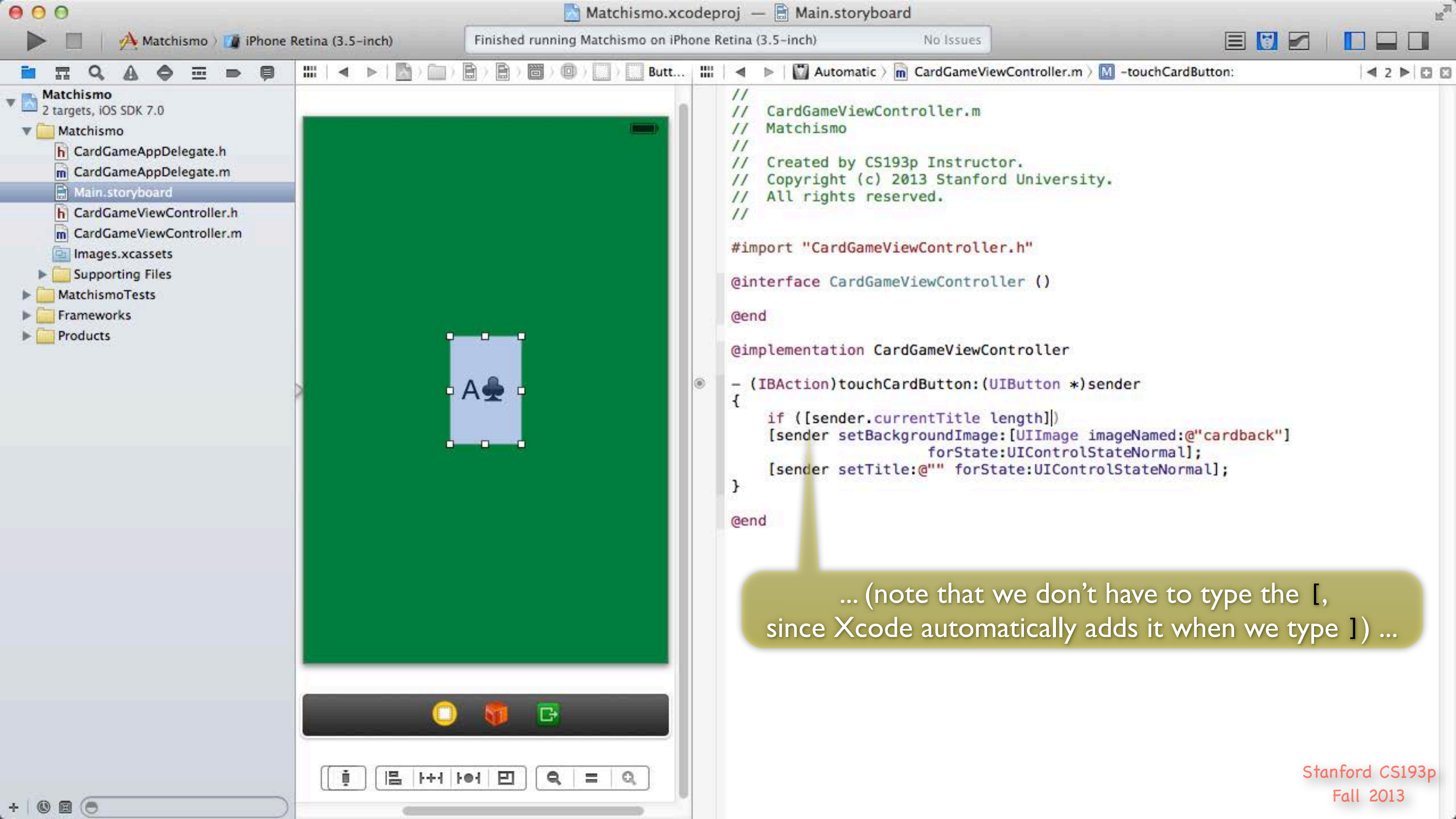
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

length

```
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if (sender.currentTitle.length)  
        [sender setTitle:@"Deal" forState:UIControlStateNormal];  
    [sender setTitle:@"Deal" forState:UIControlStateNormal];  
}  
  
@end
```

NSUInteger length  
NSUInteger lengthOfBytesUsingEncoding:(NSStringEncoding)  
Returns the number of Unicode characters in the receiver. More...

Checking the length of an NSString to see if it is blank is cool because it works whether the string is the empty string (i.e. @"") or is nil.

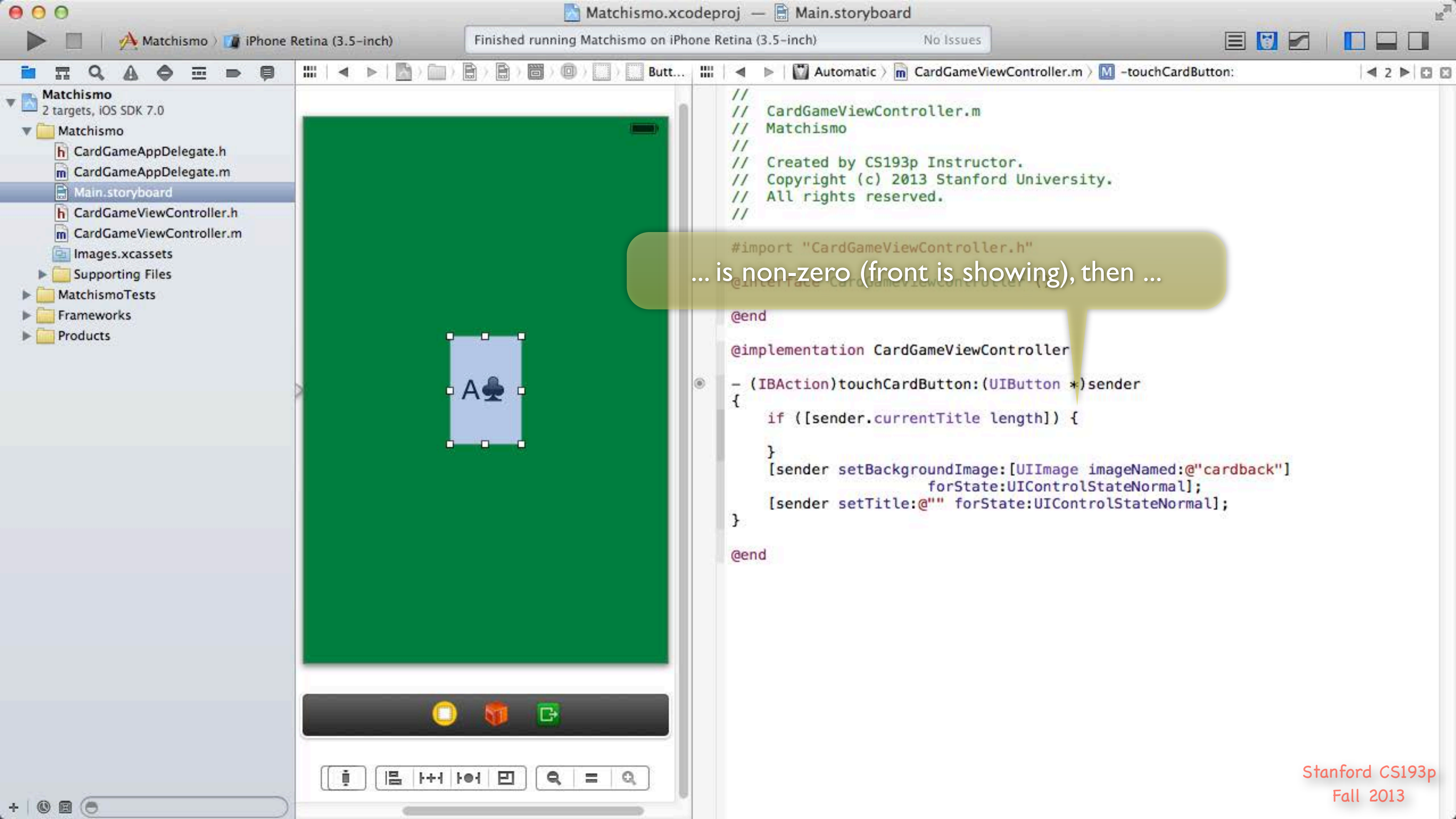


- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length])  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:UIControlStateNormal];  
}  
  
@end
```

... (note that we don't have to type the [, since Xcode automatically adds it when we type ]) ...



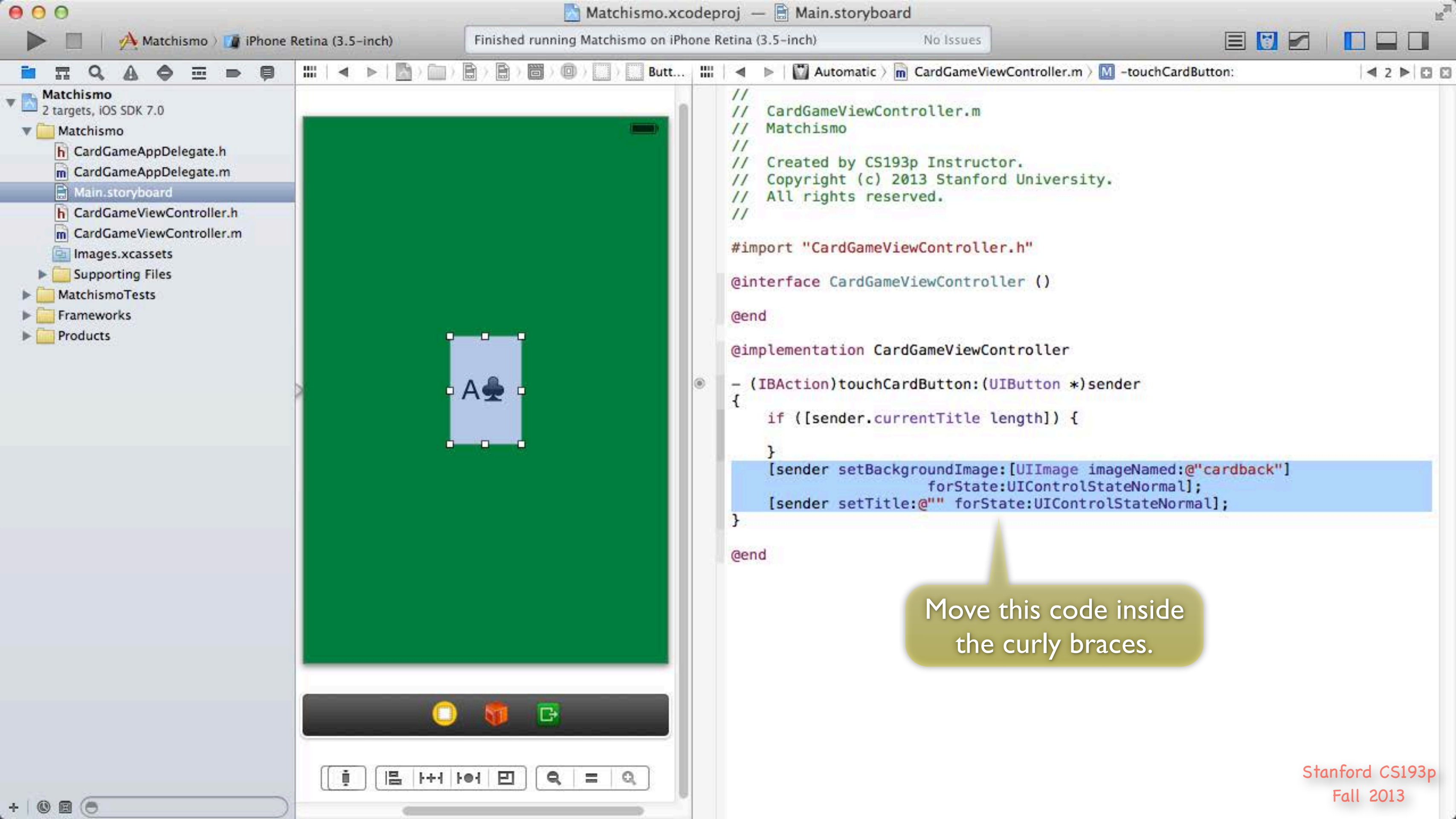


Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

... is non-zero (front is showing), then ...

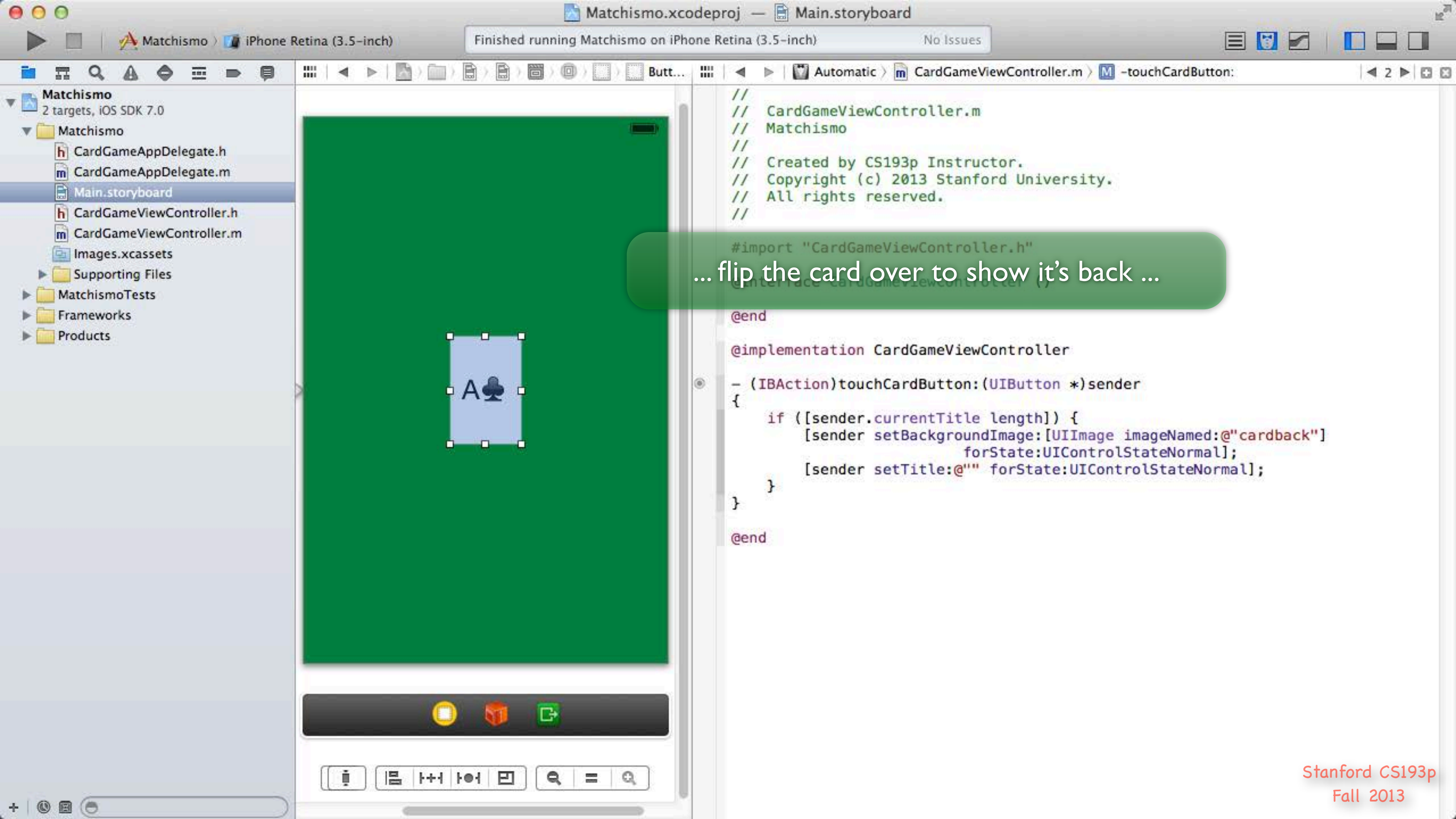
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
    }  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                             forState:UIControlStateNormal];  
    [sender setTitle:@"" forState:UIControlStateNormal];  
}  
  
@end
```



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    }  
  
@end
```

Move this code inside the curly braces.

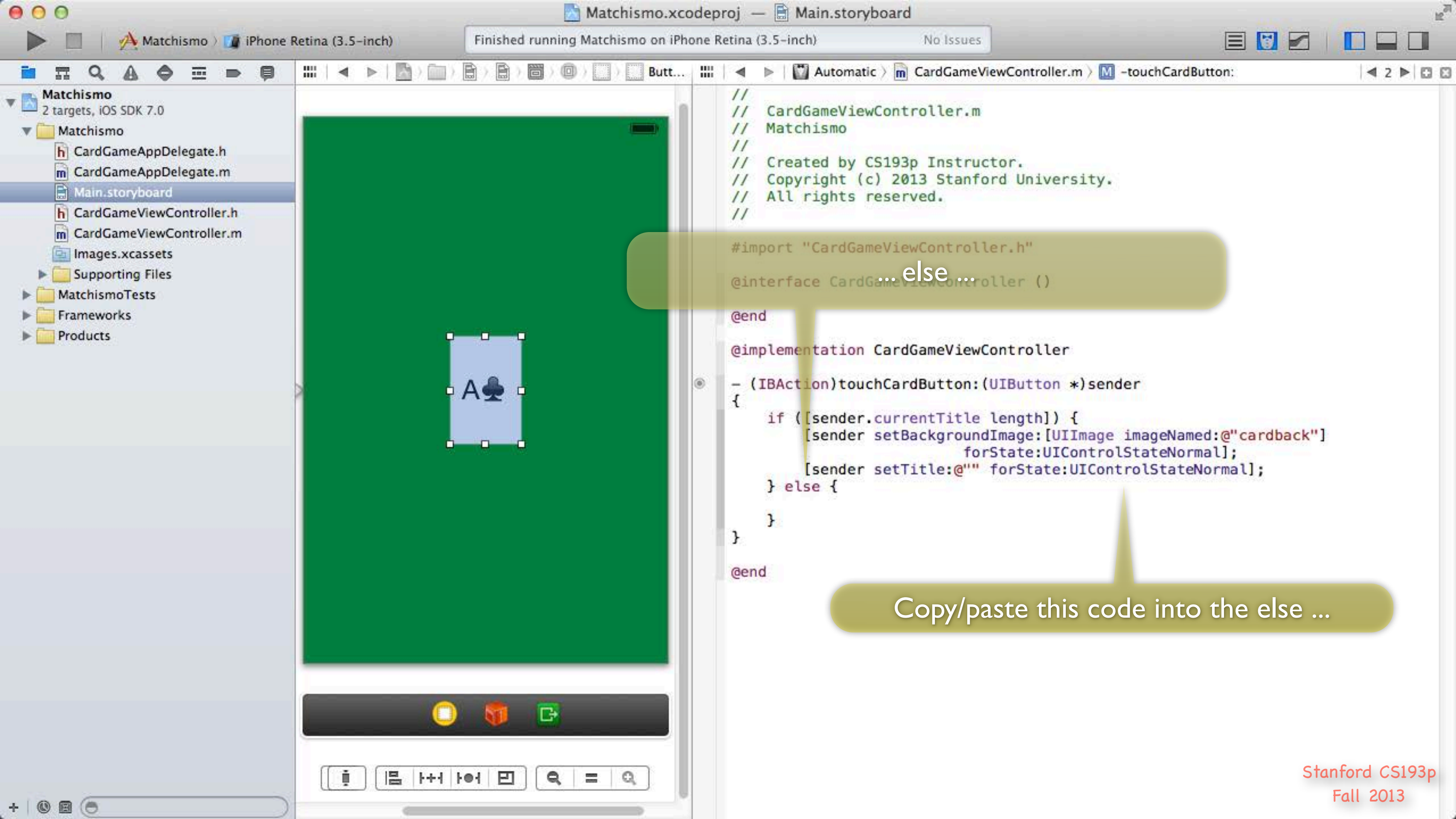


Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

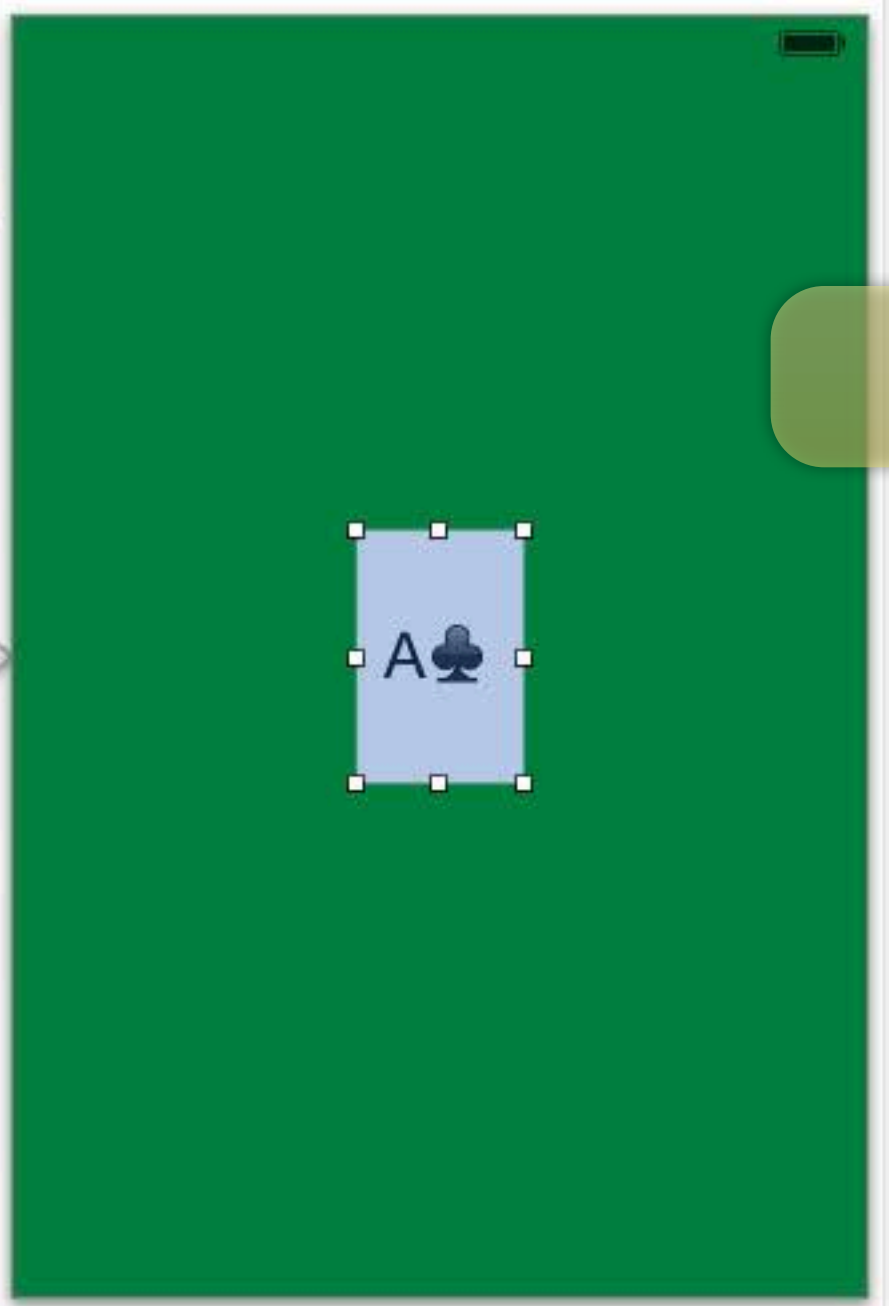
... flip the card over to show it's back ...

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    }  
}  
  
@end
```



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

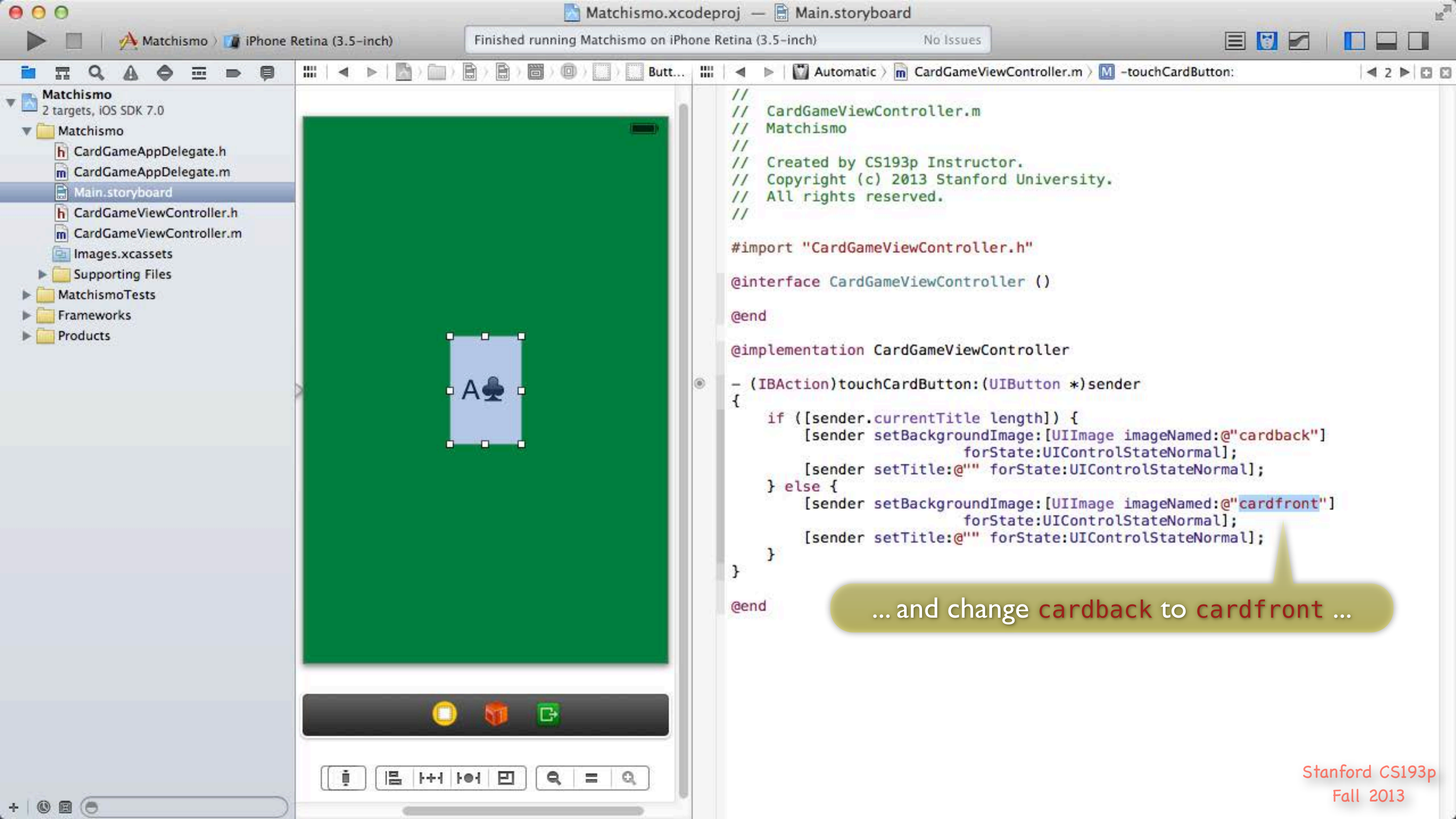
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
    }  
}  
  
@end
```

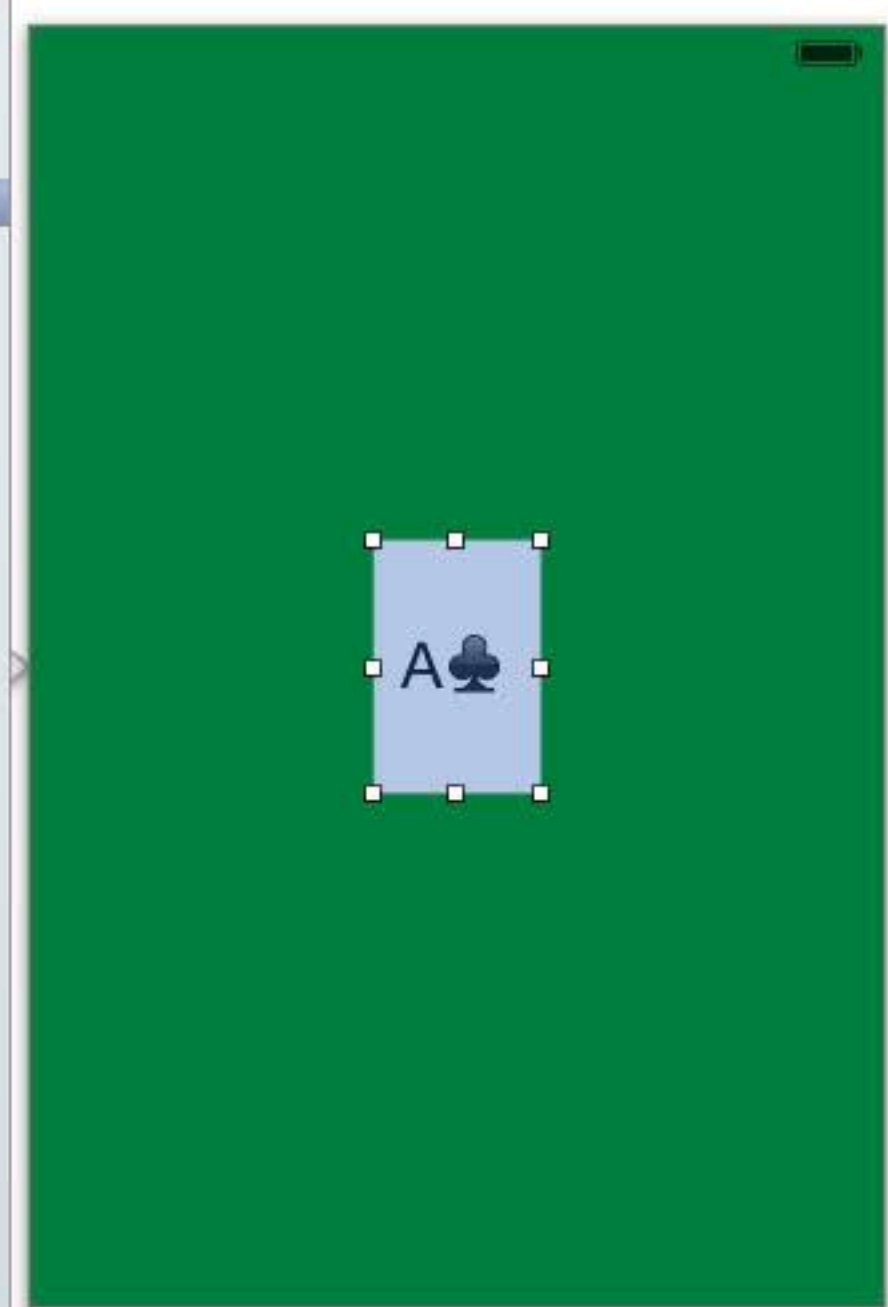
...else ...

Copy/paste this code into the else ...



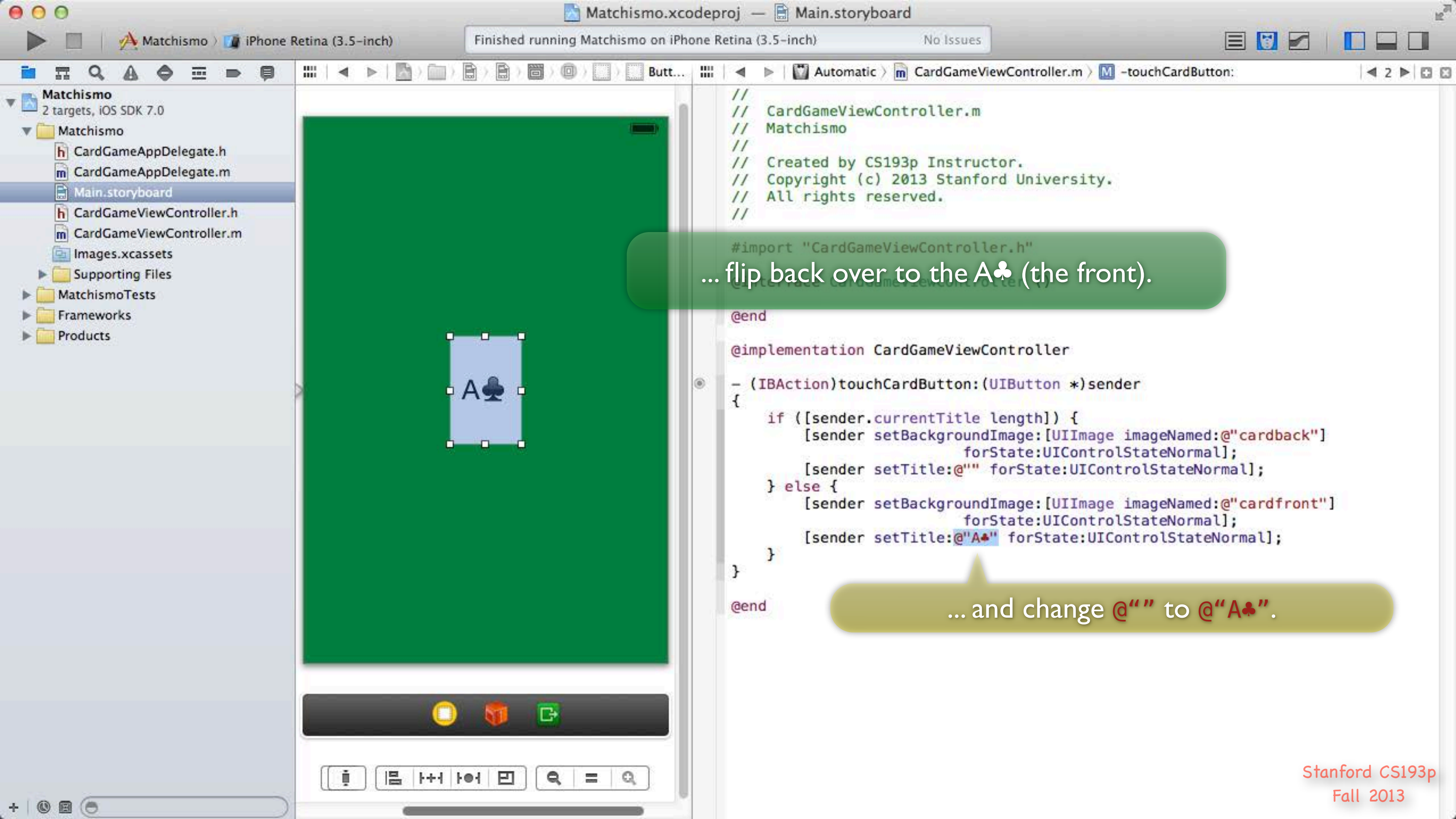
Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

... and change cardback to cardfront ...



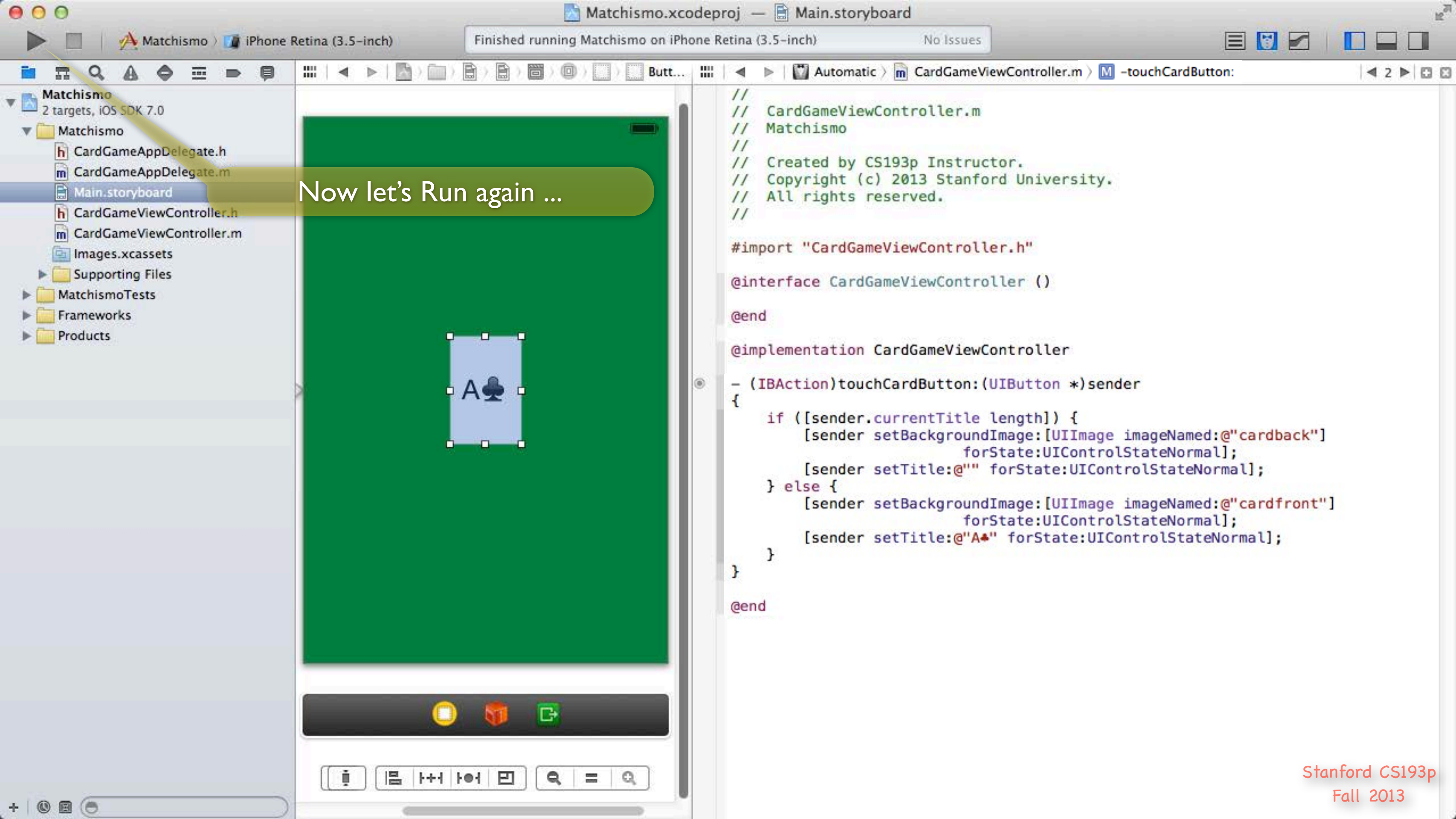
- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - CardGameAppDelegate.h
  - CardGameAppDelegate.m
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

... flip back over to the A♣ (the front).

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@end
```

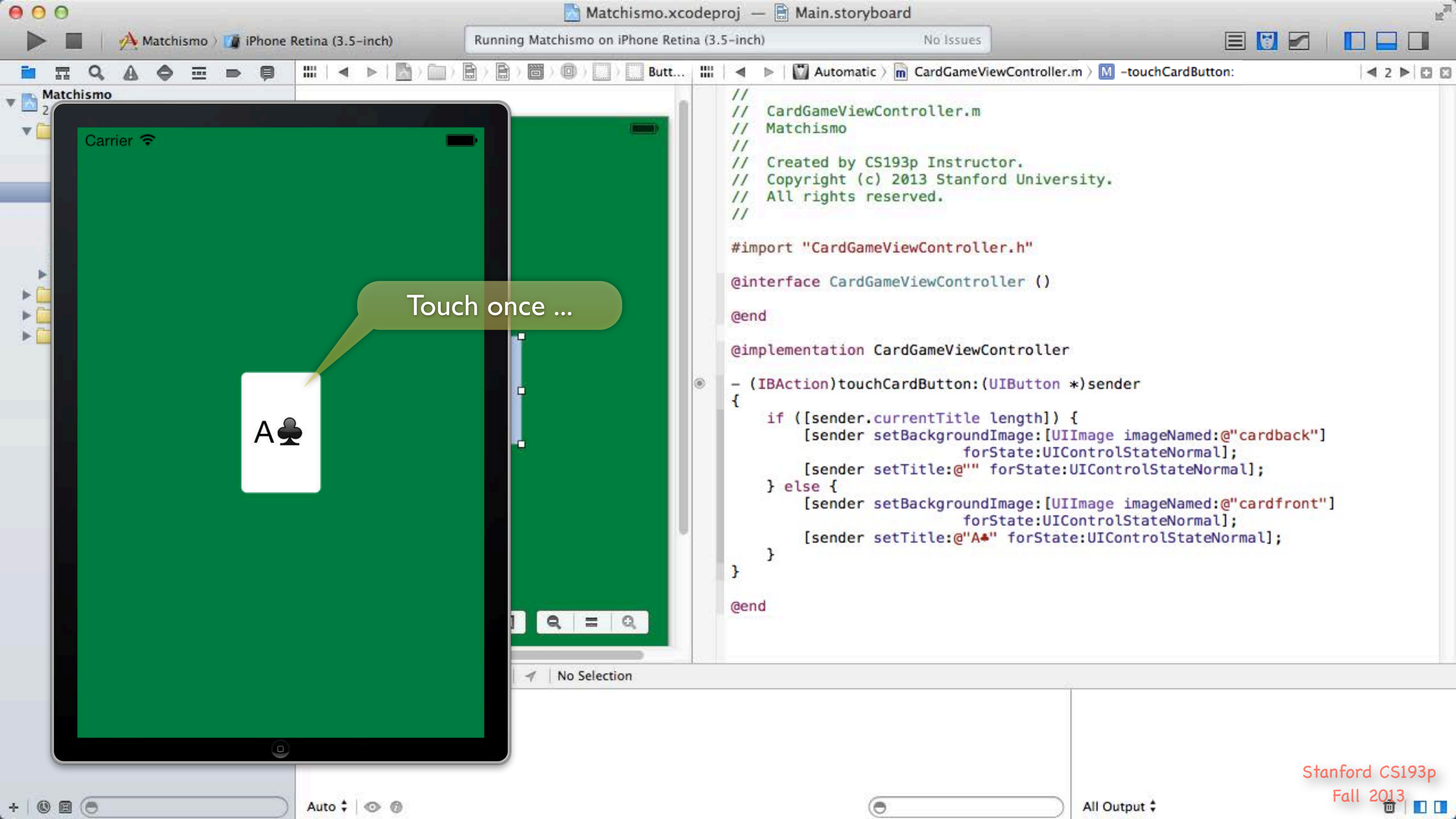
```
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

... and change @"" to @"A♣".



Now let's Run again ...

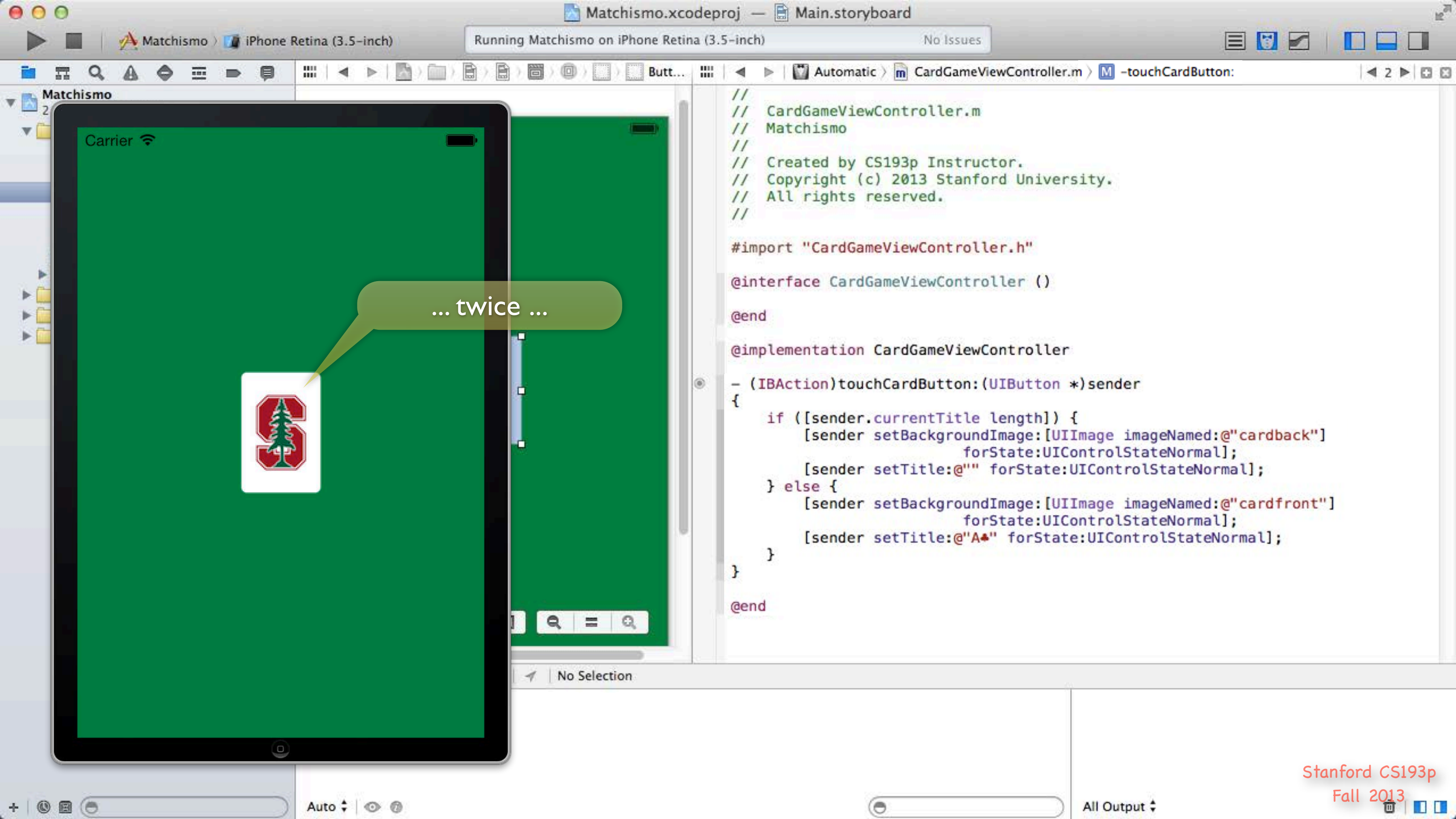
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```



Touch once ...

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

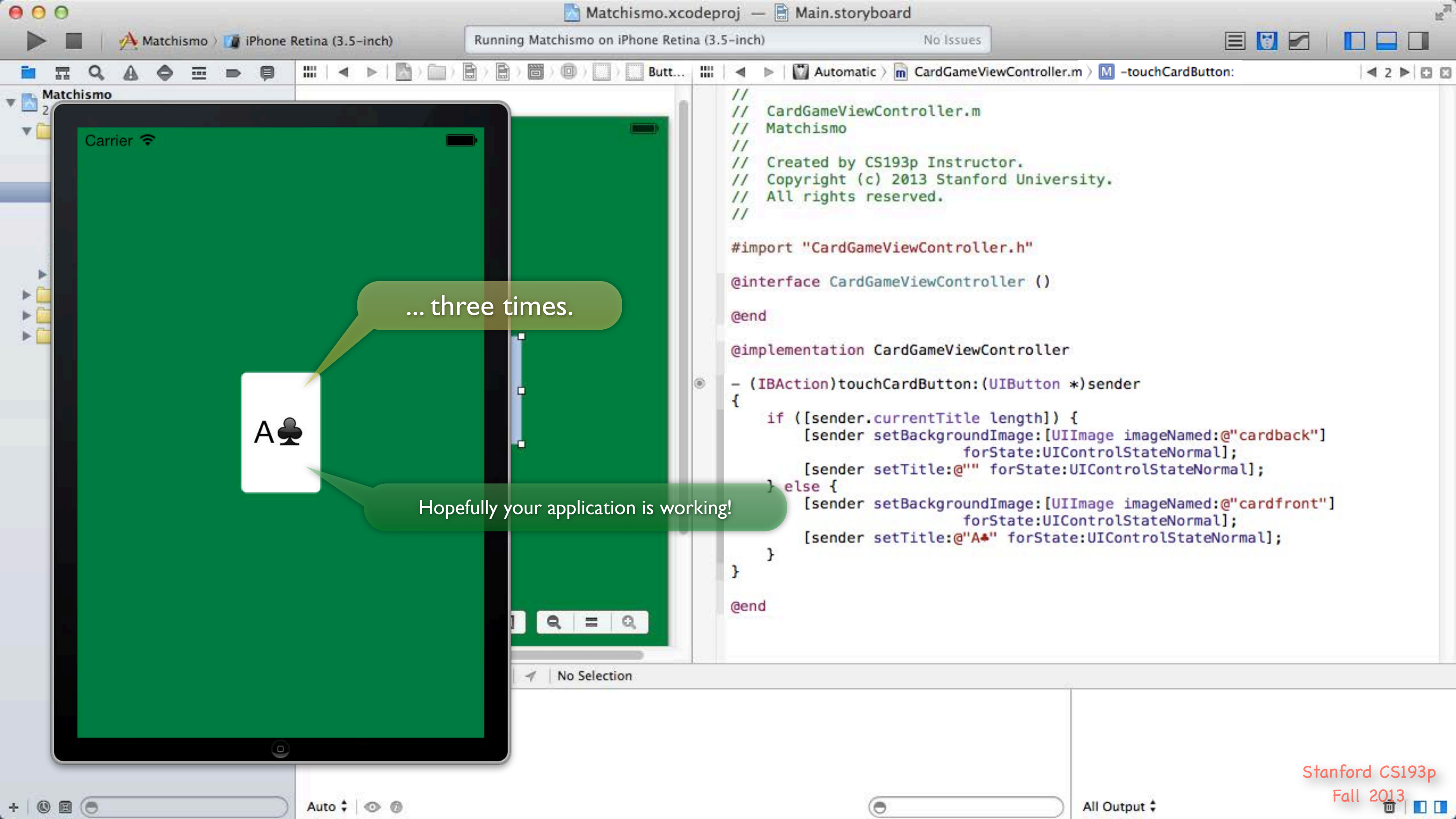




Running Matchismo on iPhone Retina (3.5-inch) No Issues

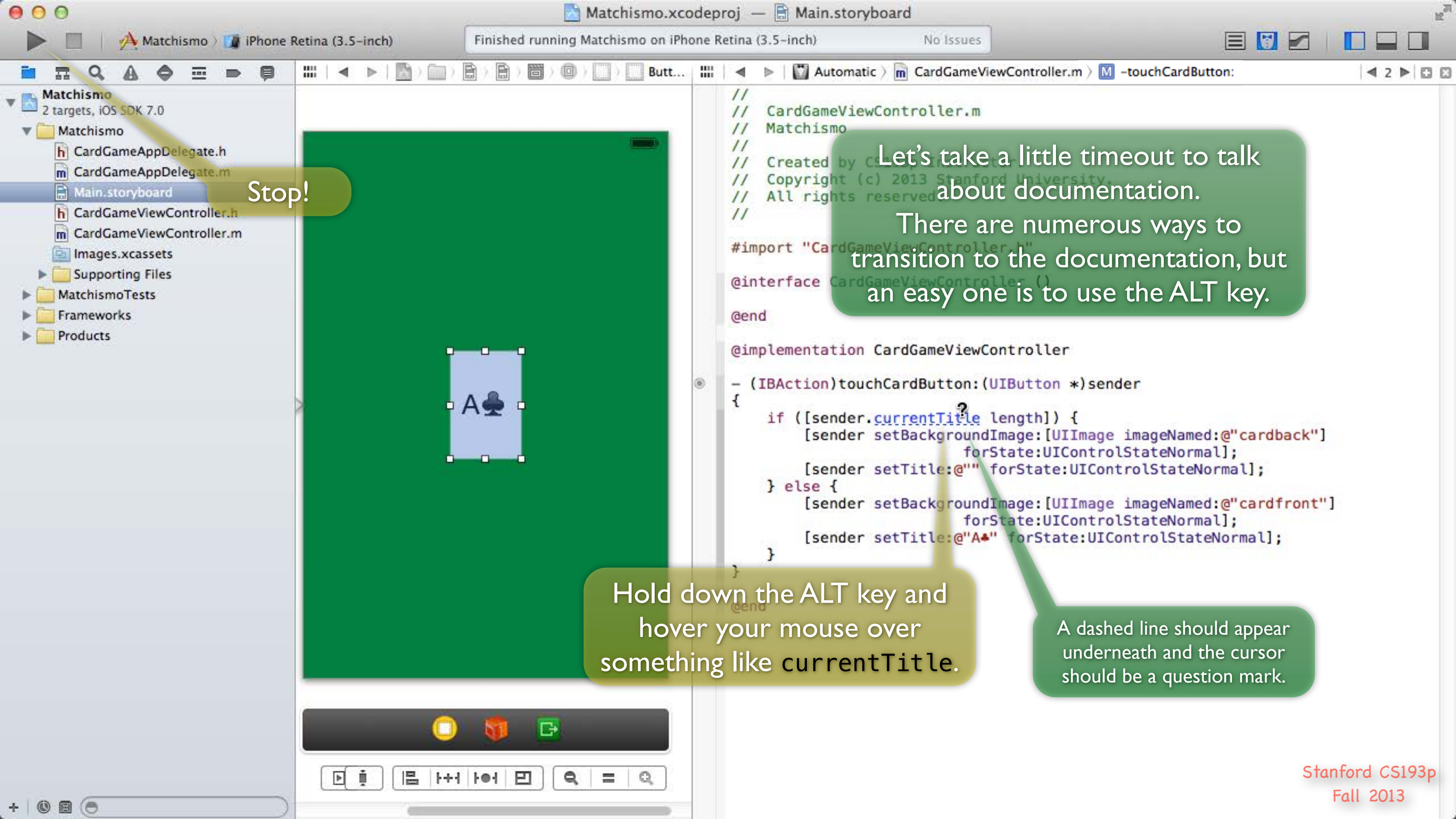
```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

... twice ...



... three times.

Hopefully your application is working!

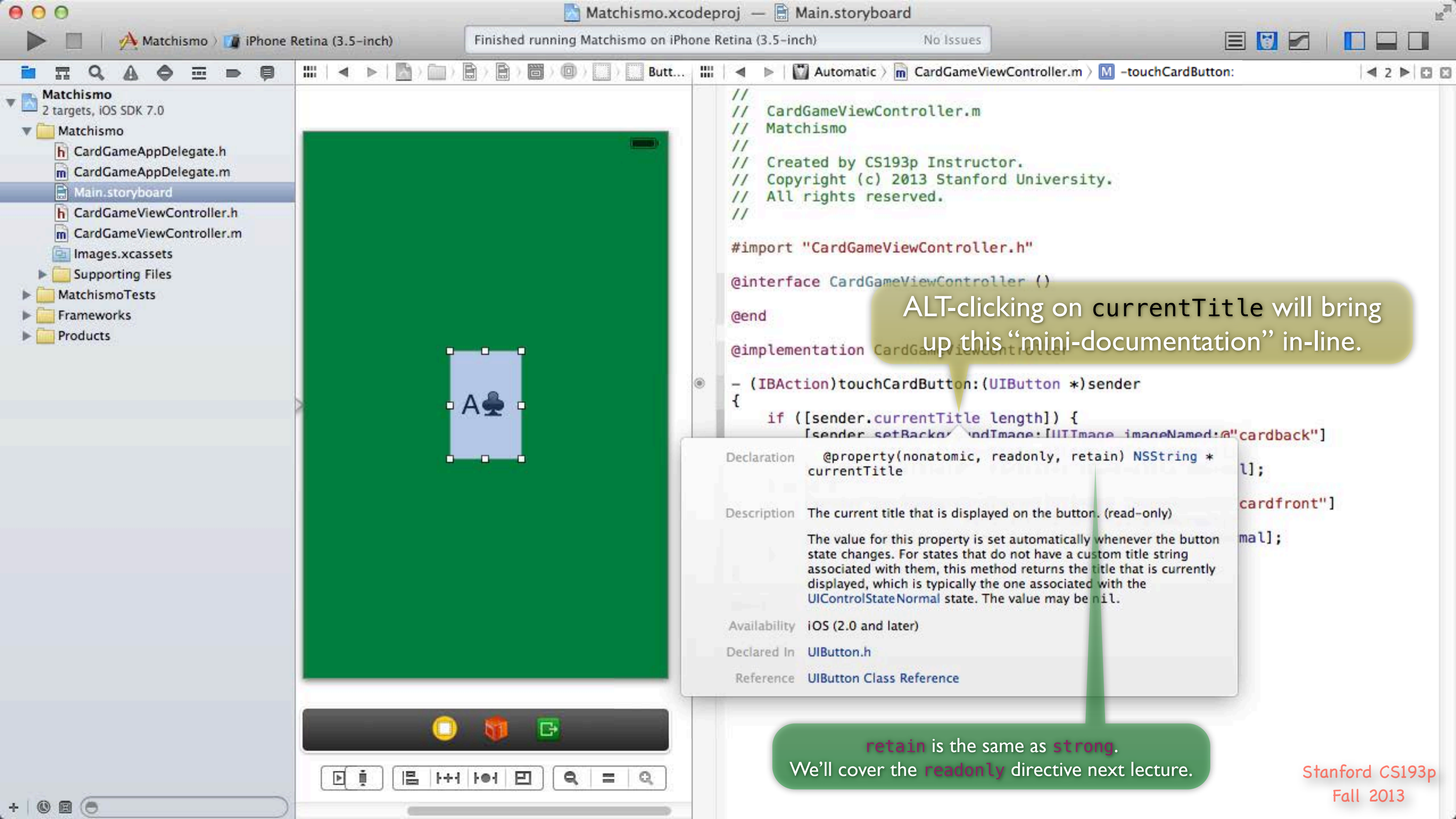


Stop!

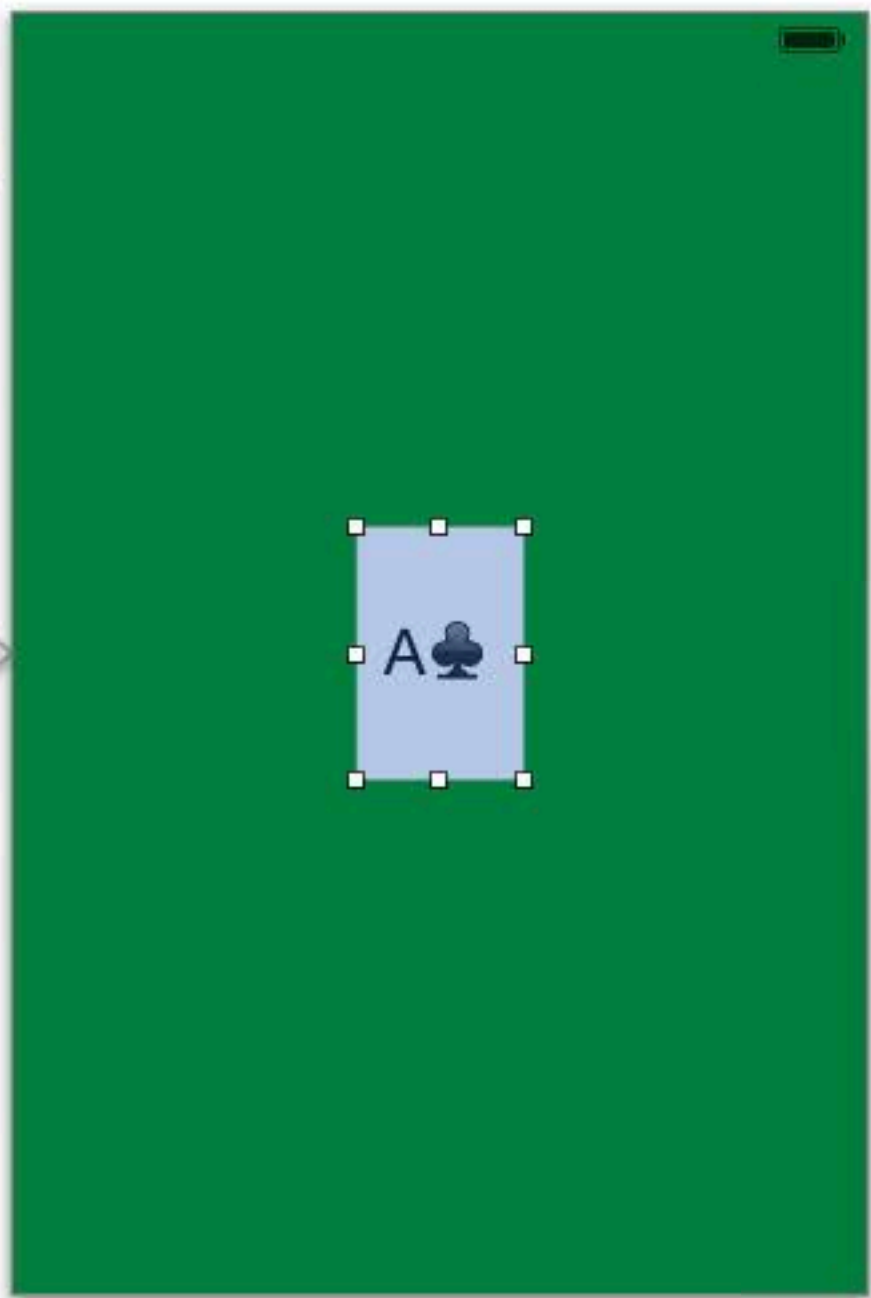
Let's take a little timeout to talk about documentation. There are numerous ways to transition to the documentation, but an easy one is to use the ALT key.

Hold down the ALT key and hover your mouse over something like `currentTitle`.

A dashed line should appear underneath and the cursor should be a question mark.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//
```

ALT-clicking on currentTitle will bring up this "mini-documentation" in-line.

```
#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
```

Declaration @property(n nonatomic, readonly, retain) NSString \* currentTitle

Description The current title that is displayed on the button. (read-only)

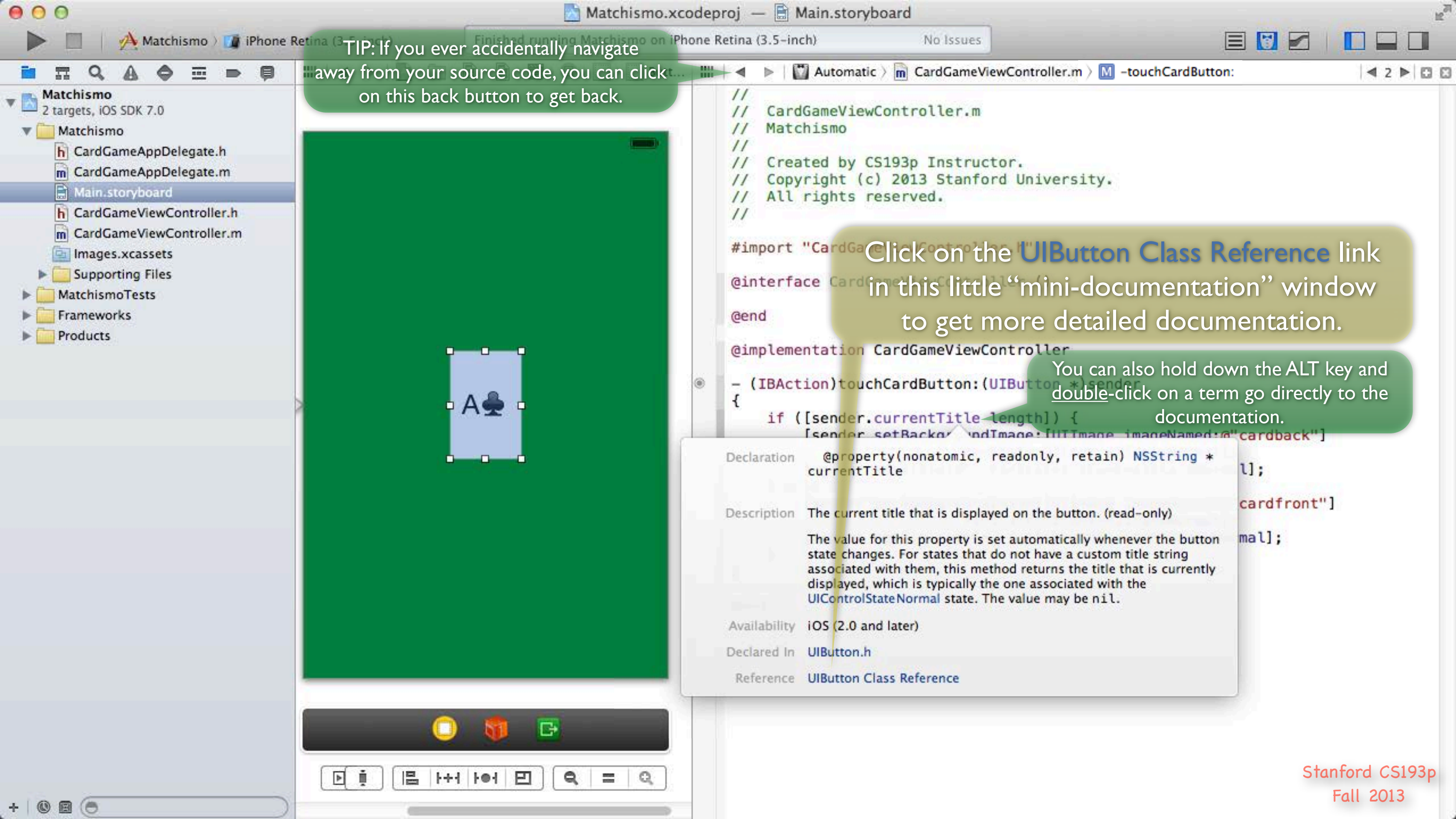
The value for this property is set automatically whenever the button state changes. For states that do not have a custom title string associated with them, this method returns the title that is currently displayed, which is typically the one associated with the UIControlStateNormal state. The value may be nil.

Availability iOS (2.0 and later)

Declared In UIButton.h

Reference UIButton Class Reference

retain is the same as strong. We'll cover the readonly directive next lecture.



TIP: If you ever accidentally navigate away from your source code, you can click on this back button to get back.

Click on the UIButton Class Reference link in this little "mini-documentation" window to get more detailed documentation.

You can also hold down the ALT key and double-click on a term go directly to the documentation.

Declaration @property(nonatomic, readonly, retain) NSString \* currentTitle];

Description The current title that is displayed on the button. (read-only)

The value for this property is set automatically whenever the button state changes. For states that do not have a custom title string associated with them, this method returns the title that is currently displayed, which is typically the one associated with the UIControlStateNormal state. The value may be nil.

Availability iOS (2.0 and later)

Declared In UIButton.h

Reference UIButton Class Reference

## Overview

**Important:** This is a preliminary document for an API or technology *in development*. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.

An instance of the `UIButton` class implements a button on the touch screen. A button intercepts touch events and sends an action message to a target object when tapped. Methods for setting the target and action are inherited from `UIControl`. This class provides methods for setting the title, image, and other appearance properties of a button. By using these accessors, you can specify a different appearance for each button state.

For information about basic view behaviors, see [View Programming Guide for iOS](#).

For more information about appearance and behavior configuration, see ["Buttons"](#).

## Tasks

### Creating Buttons

+ `buttonWithType:`

### Configuring the Button Title

`titleLabel` *property*

`reversesTitleShadowWhenHighlighted` *property*

– `setTitle:forState:`

– `setAttributedTitle:forState:`

– `setTitleColor:forState:`

– `setTitleShadowColor:forState:`

– `titleColorForState:`

– `titleForState:`

– `attributedTitleForState:`

– `titleShadowColorForState:`

### Configuring Button Presentation

This is the Documentation window.

You should explore what is here.  
It is substantial.  
Being able to maneuver through the  
documentation is critical to success in  
iOS Development.

## UIButton Class Reference

### Overview

#### Tasks

- Creating Buttons
- Configuring the Button Title
- Configuring Button Presentation
- Configuring Edge Insets
- Getting the Current State
- Getting Dimensions
- Deprecated Properties

#### Properties

- `adjustsImageWhenDisabled`
- `adjustsImageWhenHighlighted`
- `buttonType`
- `contentEdgeInsets`
- `currentAttributedTitle`
- `currentBackgroundImage`
- `currentImage`
- `currentTitle`
- `currentTitleColor`
- `currentTitleShadowColor`
- `imageEdgeInsets`
- `imageView`
- `reversesTitleShadowWhenHighlighted`
- `showsTouchWhenHighlighted`
- `tintColor`
- `titleEdgeInsets`
- `titleLabel`

#### Class Methods

- `buttonWithType:`

#### Instance Methods

- `attributedTitleForState:`
- `backgroundImageForState:`
- `backgroundRectForBounds:`
- `contentRectForBounds:`
- `imageForState:`

Available in iOS 6.0 and later.

Declared In  
UIButton.h

## setBackgroundImage:forState:

Sets the background image to use for the specified button state.

```
- (void)setBackgroundImage:(UIImage *)image forState:(UIControlState)state
```

### Parameters

*image*

The background image to use for the specified state.

*state*

The state that uses the specified image. The values are described in [UIControlState](#).

### Discussion

In general, if a property is not specified for a state, the default is to use the [UIControlStateNormal](#) value. If the [UIControlStateNormal](#) value is not set, then the property defaults to a system value. Therefore, at a minimum, you should set the value for the normal state.

### Availability

Available in iOS 2.0 and later.

### See Also

– [backgroundImageForState:](#)

### Related Sample Code

[Accessory](#)

[AddMusic](#)

[UICatalog](#)

### Declared In

UIButton.h

## setImage:forState:

Sets the image to use for the specified state.

```
- (void)setImage:(UIImage *)image forState:(UIControlState)state
```

### Parameters

For example, scroll down to  
`setBackgroundImage:forState:`.

You can click on the many  
links here, like [UIImage](#) ...

- UIButton Class Reference
  - Overview
  - Tasks
    - Creating Buttons
    - Configuring the Button Title
    - Configuring Button Presentation
    - Configuring Edge Insets
    - Getting the Current State
    - Getting Dimensions
    - Deprecated Properties
  - Properties
    - [adjustsImageWhenDisabled](#)
    - [adjustsImageWhenHighlighted](#)
    - [buttonType](#)
    - [contentEdgeInsets](#)
    - [currentAttributedTitle](#)
    - [currentBackgroundImage](#)
    - [currentImage](#)
    - [currentTitle](#)
    - [currentTitleColor](#)
    - [currentTitleShadowColor](#)
    - [imageEdgeInsets](#)
    - [imageView](#)
    - [reversesTitleShadowWhenHighlighted](#)
    - [showsTouchWhenHighlighted](#)
    - [tintColor](#)
    - [titleEdgeInsets](#)
    - [titleLabel](#)
  - Class Methods
    - [buttonWithType:](#)
  - Instance Methods
    - [attributedStringForState:](#)
    - [backgroundImageForState:](#)
    - [backgroundRectForBounds:](#)
    - [contentRectForBounds:](#)
    - [imageForState:](#)

# UIImage Class Reference

Next

And get detailed class overviews.

## Overview

**Important:** This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology.

A `UIImage` object is a high-level way to display image data. You can create images from files, from Quartz image objects, or from raw image data you receive. The `UIImage` class also offers several options for drawing images to the current graphics context using different blend modes and opacity values.

Image objects are immutable, so you cannot change their properties after creation. This means that you generally specify an image's properties at initialization time or rely on the image's metadata to provide the property value. It also means that image objects are themselves safe to use from any thread. The way you change the properties of an existing image object is to use one of the available convenience methods to create a copy of the image but with the custom value you want.

Because image objects are immutable, they also do not provide direct access to their underlying image data. However, you can get an `NSData` object containing either a PNG or JPEG representation of the image data using the `UIImagePNGRepresentation` and `UIImageJPEGRepresentation` functions.

The system uses image objects to represent still pictures taken with the camera on supported devices. To take a picture, use the `UIImagePickerController` class. To save a picture to the Saved Photos album, use the `UIImageWriteToSavedPhotosAlbum` function.

## Images and Memory Management

In low-memory situations, image data may be purged from a `UIImage` object to free up memory on the system. This purging behavior affects only the image data stored internally by the `UIImage` object and not the object itself. When you attempt to draw an image whose data has been purged, the image object automatically reloads the data from its original file. This extra load step, however, may incur a small performance penalty.

You should avoid creating `UIImage` objects that are greater than 1024 x 1024 in size. Besides the large amount of memory such an image would consume, you may run into problems when using the image as a texture in OpenGL ES or when drawing the image to a view or layer. This size restriction does not apply if you are performing code-based manipulations, such as resizing an image larger than 1024 x 1024 pixels by drawing to a bitmap-backed graphics context. In fact, you may need to resize an image in this manner (or break it into several smaller images) in order to






- UIImage Class Reference
  - Overview
    - Images and Memory Management
    - Supported Image Formats
  - Tasks
    - Cached Image Loading Routines
    - Creating New Images
    - Initializing Images
    - Image Attributes
    - Drawing Images
  - Properties
    - alignmentRectInsets
    - capInsets
    - CGImage
    - CImage
    - duration
    - imageOrientation
    - images
    - renderingMode
    - resizingMode
    - scale
    - size
  - Class Methods
    - animatedImageNamed:duration:
    - animatedImageWithImages:duration:
    - animatedResizableImageNamed:ca...
    - animatedResizableImageNamed:ca...
    - imageNamed:
    - imageWithCGImage:
    - imageWithCGImage:scale:orientation:
    - imageWithCImage:
    - imageWithCImage:scale:orientation:
    - imageWithContentsOfFile:
    - imageWithData:Stanford CS193p
    - imageWithData:scale:Fall 2013
  - Instance Methods







Top Hit

 NSString

API Reference

-  Appendix A: Deprecated NSString Methods NSString Class Reference
-  Appendix A: Deprecated NSString UIKit Additions Methods NSString UIKit Additions Reference
-  CFStringConvertEncodingToNSStringEncoding
-  CFStringConvertNSStringEncodingToEncoding
-  NSString

SDK Guides

-  NSString Appendix A: Old-Style ASCII Property...
-  Strings Are Represented b...ances of the NSString Class Objects Can Represent Primitive Values
-  NSString from C Strings and Data Creating Strings
-  Technical Q&A QA1235 Converting to Precomposed Unicode

Sample Code

-  Birthdays IDEs
-  LLDBCustomDataFormatter Languages & Utilities
-  Show All Results

You can also search for classes, methods, or just general topics of interest.

## Supported

Table 1 lists the

Table 1 Support

## Format

Tagged Image

Joint Photogra

Graphic Inter

Portable Netw

Windows Bitm

Windows Icon

Windows Cursor

.cur

XWindow bitmap

.xbm

**Note:** Windows Bitmap Format (BMP) files that are formatted as RGB-565 are converted to ARGB-1555 when they are loaded.

## Tasks

## Cached Image Loading Routines

+ [imageNamed:](#)

## Creating New Images

+ [initWithContentsOfFile:](#)

Ullimage Class Reference

## Overview

- Images and Memory Management
- Supported Image Formats

## Tasks

- Cached Image Loading Routines
- Creating New Images
- Initializing Images
- Image Attributes
- Drawing Images

## Properties

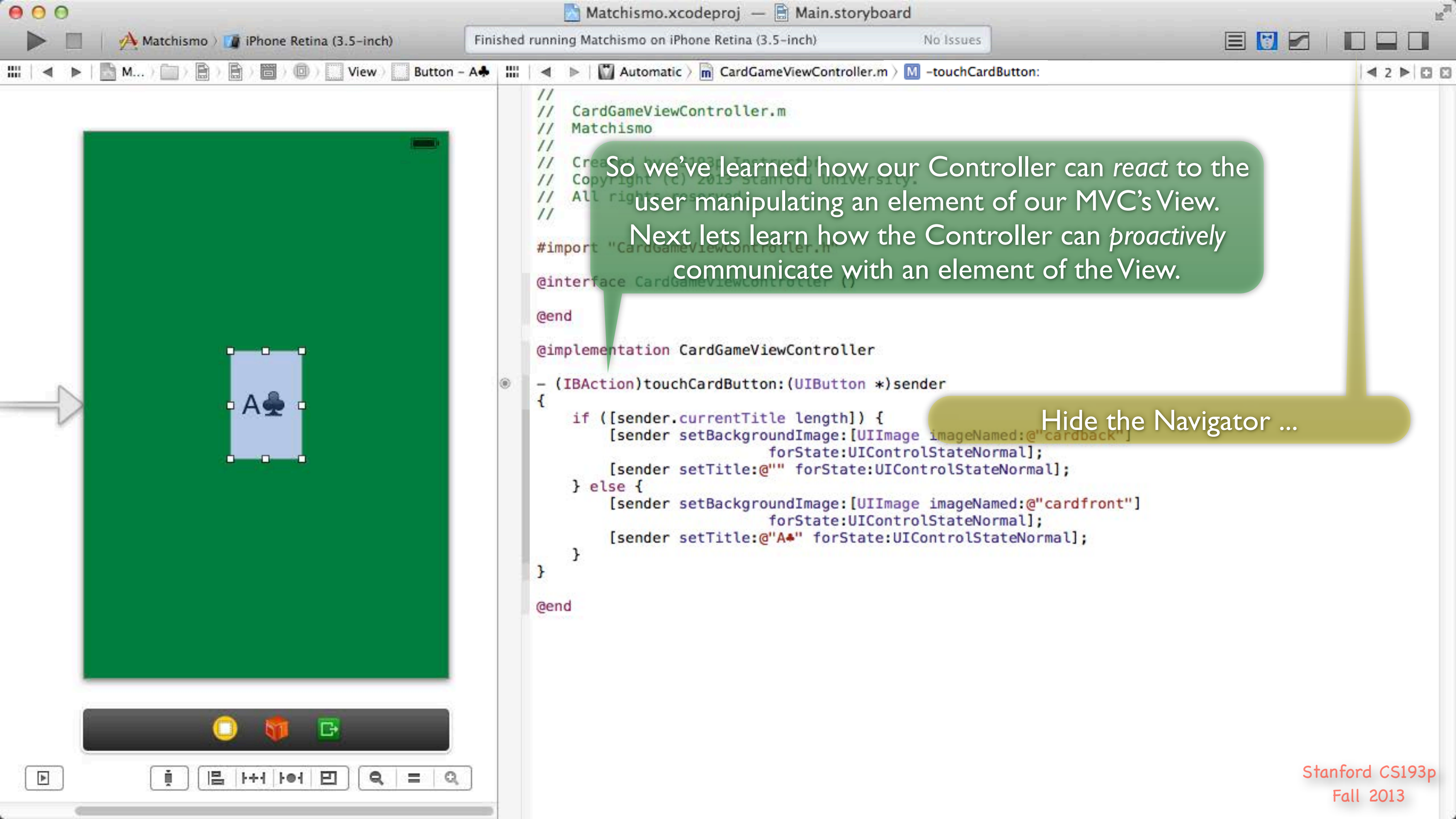
- alignmentRectInsets
- capInsets
- CGImage
- CImage
- duration
- imageOrientation
- images
- renderingMode
- resizingMode
- scale
- size

## Class Methods

- animatedImageNamed:duration:
- animatedImageWithImages:duration:
- animatedResizableImageNamed:ca...
- animatedResizableImageNamed:ca...
- imageNamed:
- imageWithCGImage:
- imageWithCGImage:scale:orientation:
- imageWithCImage:
- imageWithCImage:scale:orientation:
- imageWithContentsOfFile:
- imageWithData:Stanford CS193p
- imageWithData:scale:Fall 2013

## Instance Methods

Provide Feedback



```
//
// CardGameViewController.m
// Matchismo
// Created by CS193p Instructor
// Copyright (c) 2015 Stanford University.
// All rights reserved.
//
```

So we've learned how our Controller can *react* to the user manipulating an element of our MVC's View. Next lets learn how the Controller can *proactively* communicate with an element of the View.

```
#import "CardGameViewController.h"
@interface CardGameViewController ()
```

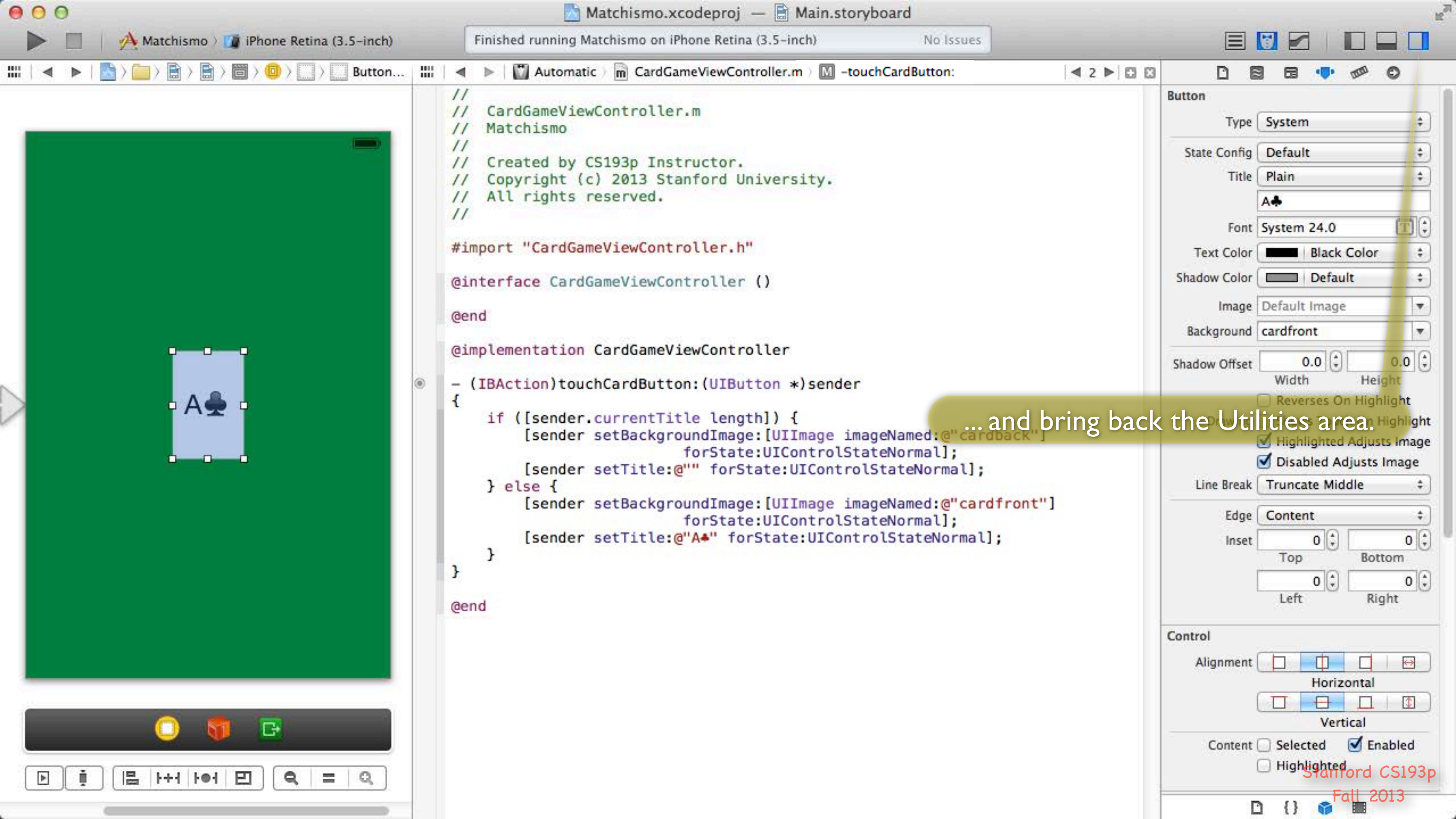
```
@end
@implementation CardGameViewController
```

```
- (IBAction)touchCardButton:(UIButton *)sender
```

```
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
}
```

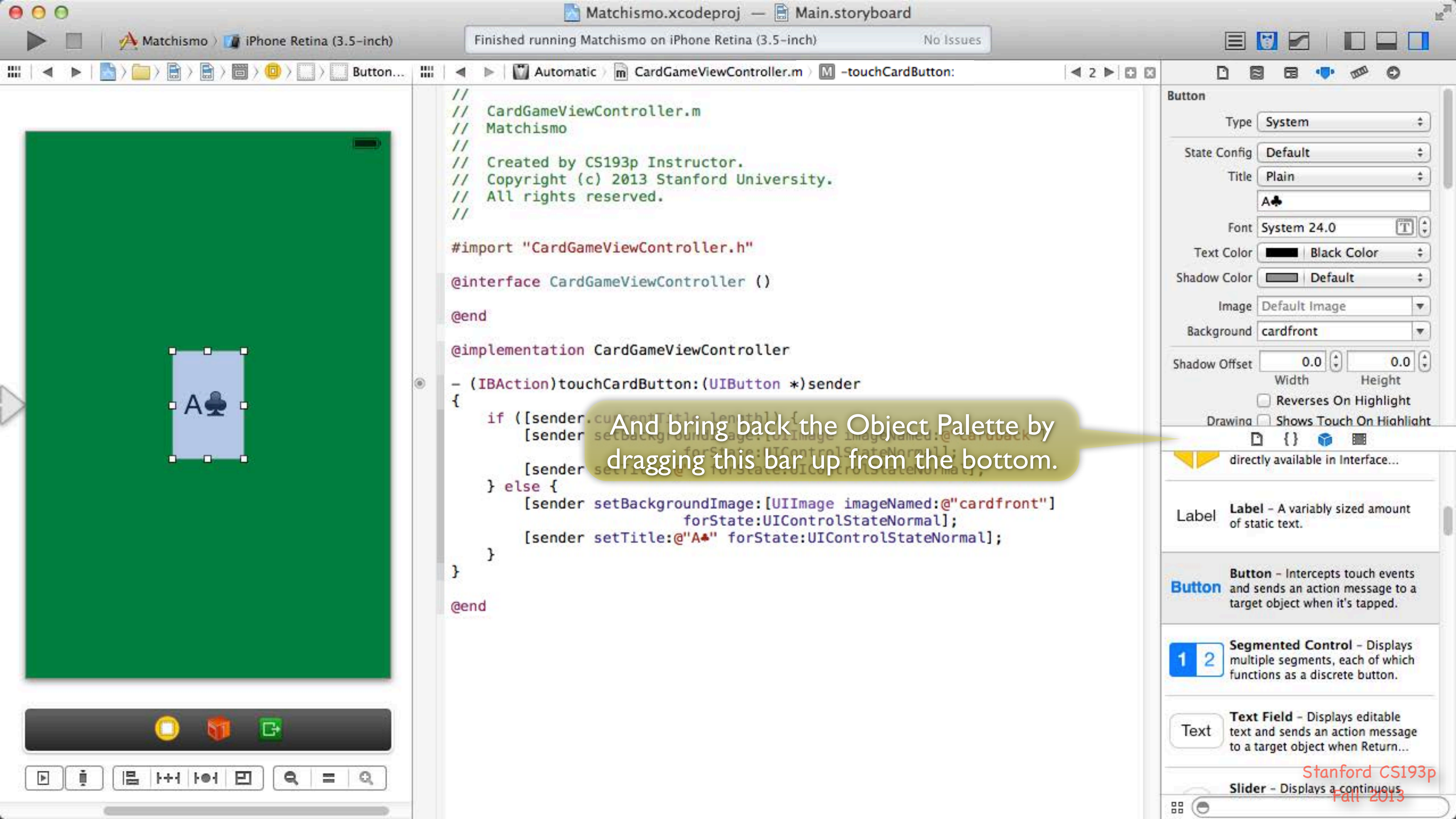
Hide the Navigator ...

```
@end
```



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

... and bring back the Utilities area.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"
```

```
@interface CardGameViewController ()
```

```
@end
```

```
@implementation CardGameViewController
```

```
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.titleLabel length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
@end
```

And bring back the Object Palette by dragging this bar up from the bottom.

**Button**

Type: System

State Config: Default

Title: Plain

A♣

Font: System 24.0

Text Color: Black Color

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width) 0.0 (Height)

Reverses On Highlight

Shows Touch On Highlight

directly available in Interface...

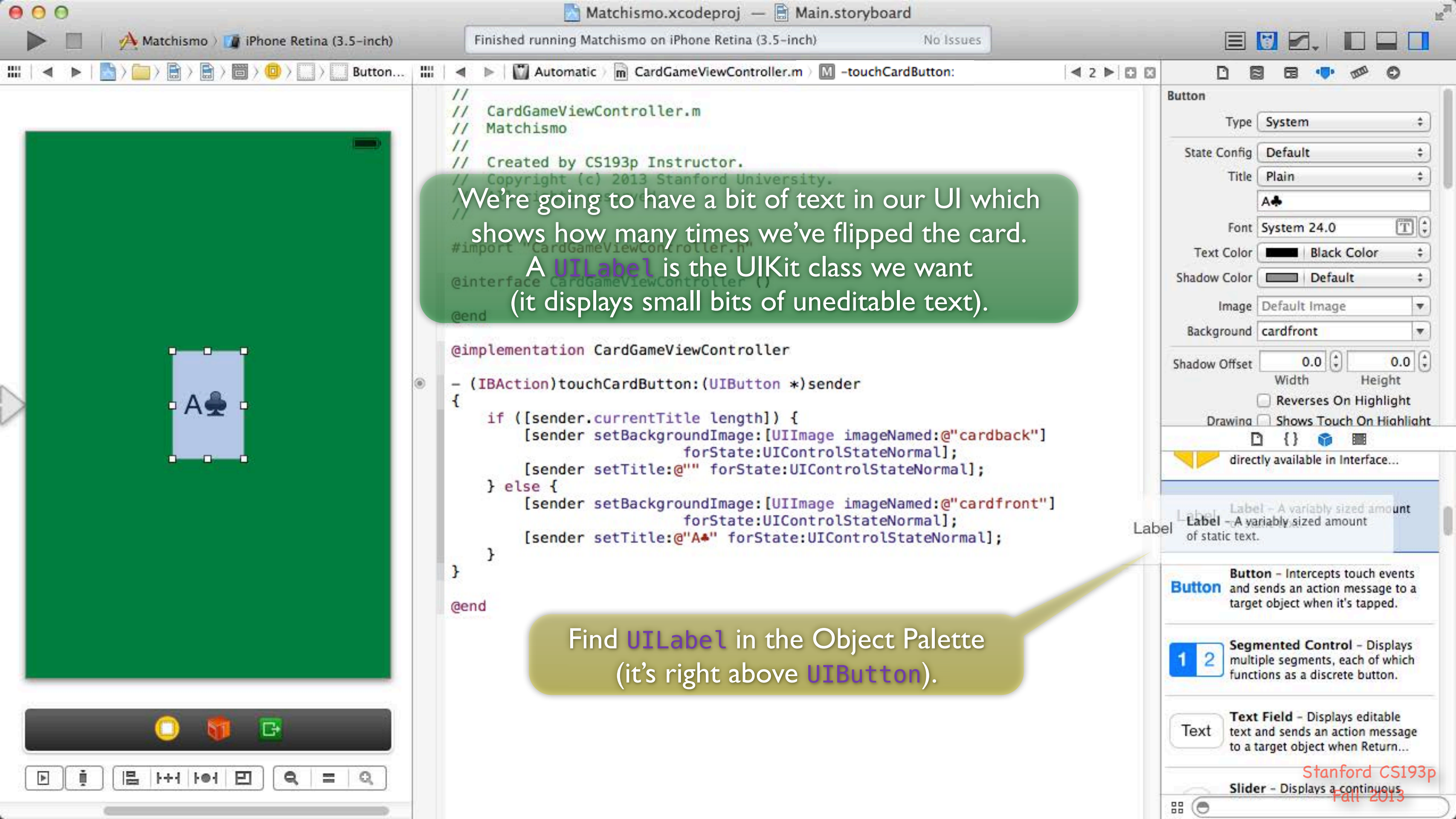
**Label** Label - A variably sized amount of static text.

**Button** Button - Intercepts touch events and sends an action message to a target object when it's tapped.

**1 2** Segmented Control - Displays multiple segments, each of which functions as a discrete button.

**Text** Text Field - Displays editable text and sends an action message to a target object when Return...

Slider - Displays a continuous



We're going to have a bit of text in our UI which shows how many times we've flipped the card. A UILabel is the UIKit class we want (it displays small bits of uneditable text).

Find UILabel in the Object Palette (it's right above UIButton).

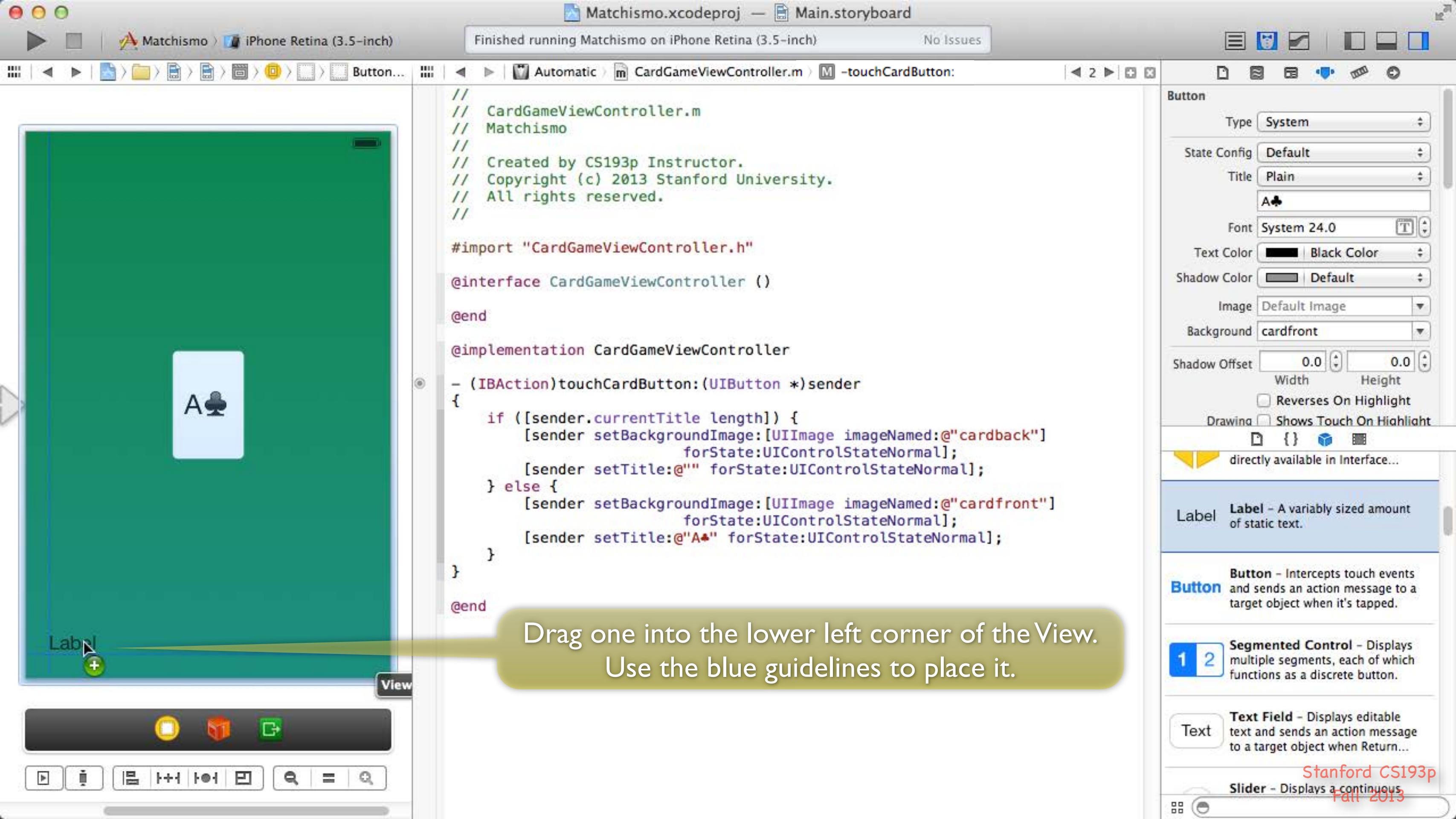
Label - A variably sized amount of static text.

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

1 2 Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Text Text Field - Displays editable text and sends an action message to a target object when Return...

Slider - Displays a continuous



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

Drag one into the lower left corner of the View.  
Use the blue guidelines to place it.

**Button**

Type: System

State Config: Default

Title: Plain

A♣

Font: System 24.0

Text Color: Black Color

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width) 0.0 (Height)

Reverses On Highlight

Shows Touch On Highlight

directly available in Interface...

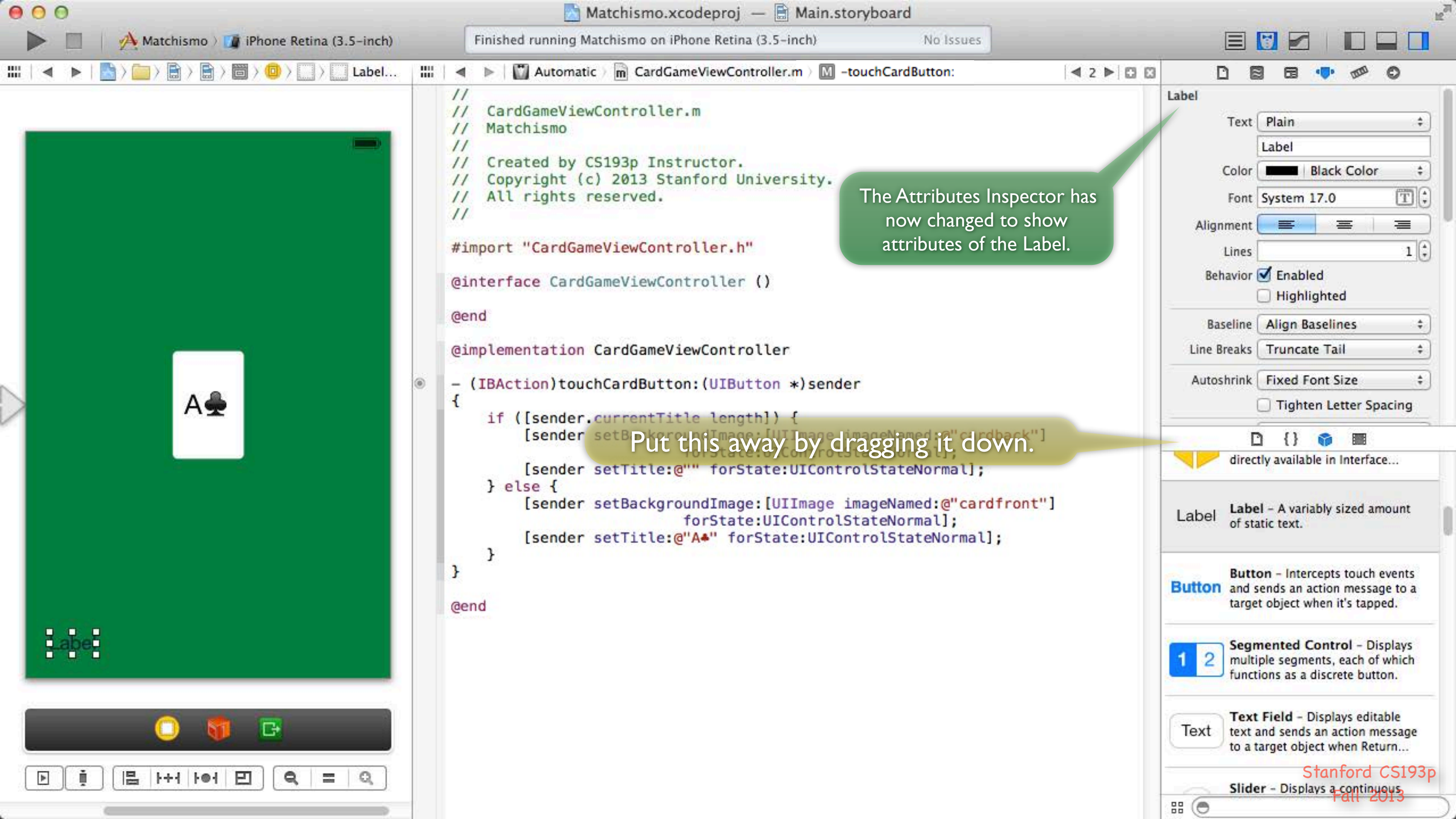
**Label** Label - A variably sized amount of static text.

**Button** Button - Intercepts touch events and sends an action message to a target object when it's tapped.

**1 2 Segmented Control** - Displays multiple segments, each of which functions as a discrete button.

**Text** Text Field - Displays editable text and sends an action message to a target object when Return...

**Slider** - Displays a continuous



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

The Attributes Inspector has now changed to show attributes of the Label.

Put this away by dragging it down.

**Label**

Text: Plain  
Label  
Color: Black Color  
Font: System 17.0  
Alignment: [Left, Center, Right] [Left selected]  
Lines: 1  
Behavior:  Enabled  
 Highlighted  
Baseline: Align Baselines  
Line Breaks: Truncate Tail  
Autoshrink: Fixed Font Size  
 Tighten Letter Spacing

directly available in Interface...

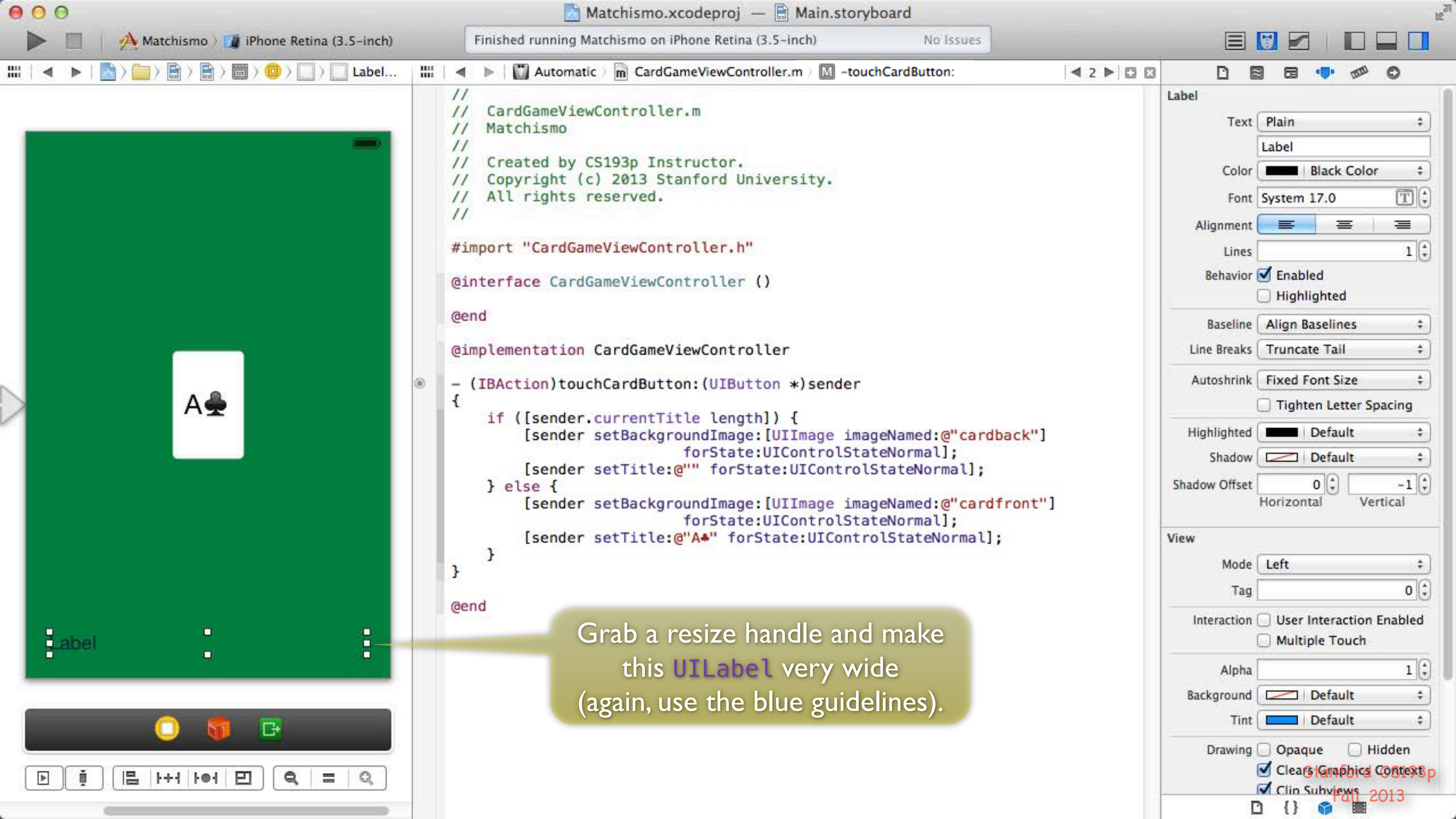
**Label** Label - A variably sized amount of static text.

**Button** Button - Intercepts touch events and sends an action message to a target object when it's tapped.

**1 2** Segmented Control - Displays multiple segments, each of which functions as a discrete button.

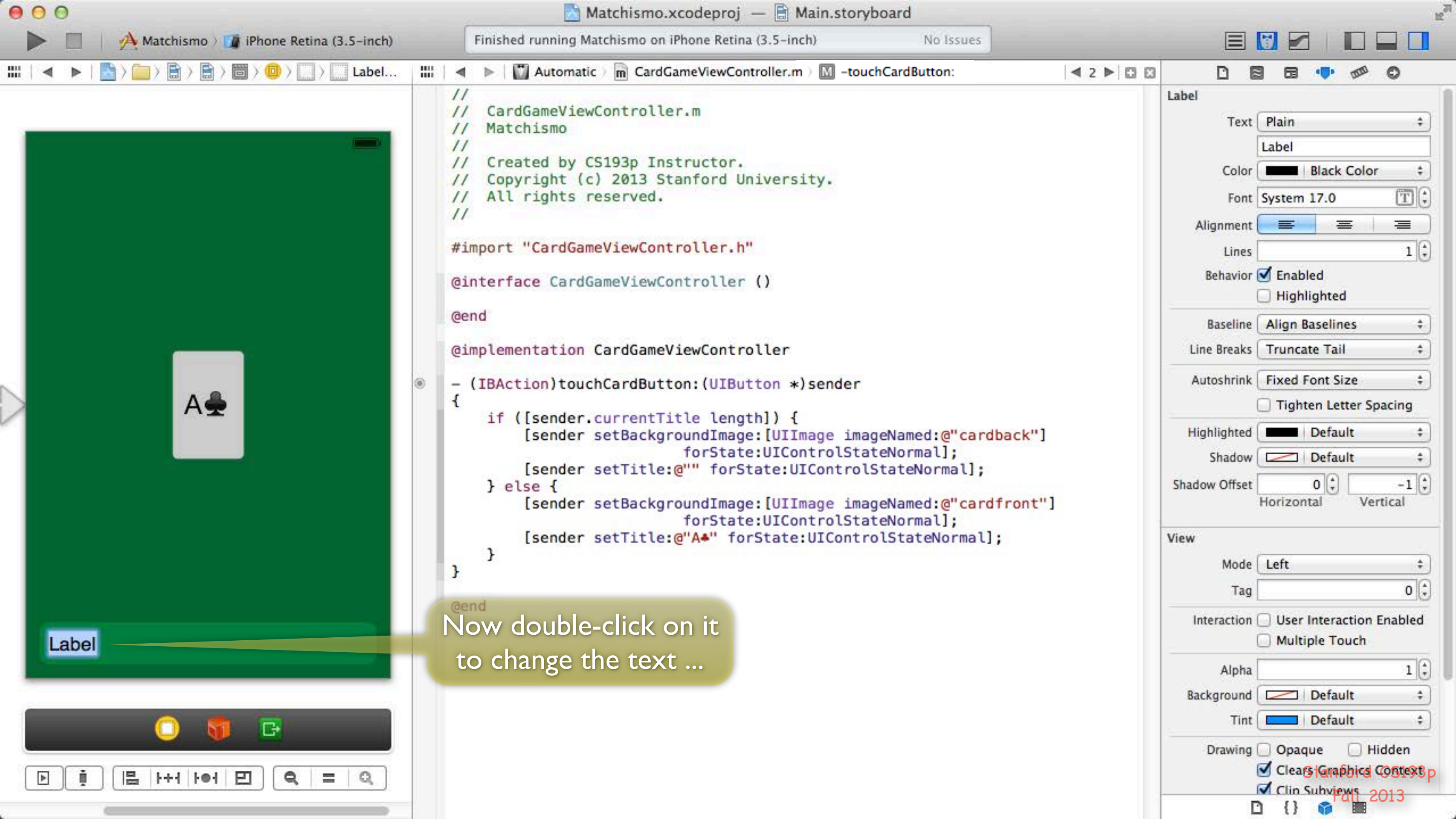
**Text** Text Field - Displays editable text and sends an action message to a target object when Return...

**Slider** Slider - Displays a continuous



Grab a resize handle and make this UILabel very wide (again, use the blue guidelines).





Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

Now double-click on it to change the text ...

**Label**

Text: Plain  
Label

Color: Black Color

Font: System 17.0

Alignment: [Left] [Center] [Right]

Lines: 1

Behavior:  Enabled  
 Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: 0 Horizontal, -1 Vertical

**View**

Mode: Left

Tag: 0

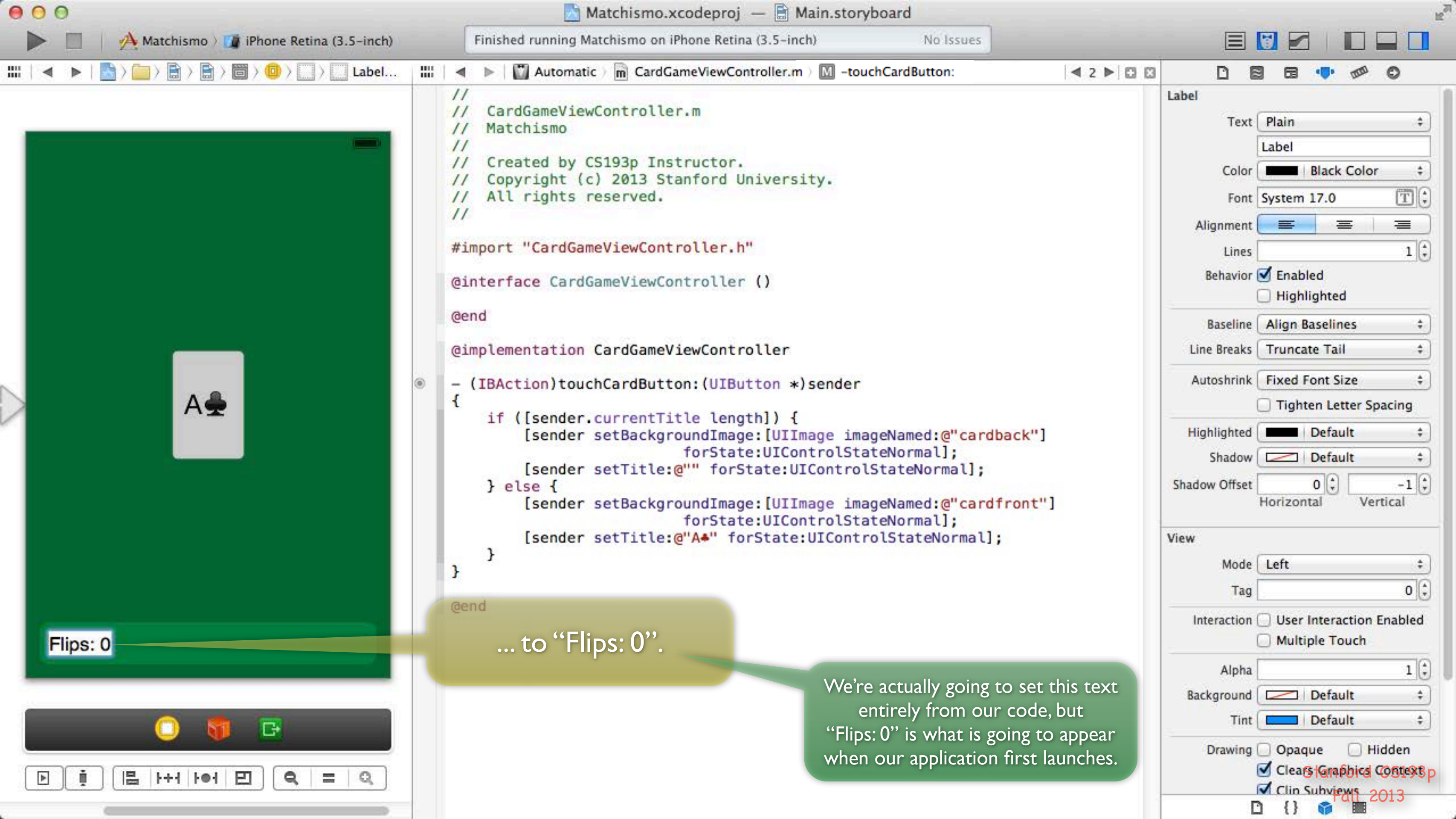
Interaction:  User Interaction Enabled  
 Multiple Touch

Alpha: 1

Background: Default

Tint: Default

Drawing:  Opaque  Hidden  
 Clear Graphic Context  
 Clip Subviews



```

//
//  CardGameViewController.m
//  Matchismo
//
//  Created by CS193p Instructor.
//  Copyright (c) 2013 Stanford University.
//  All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()

@end

@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
}

@end

```

... to "Flips: 0".

We're actually going to set this text entirely from our code, but "Flips: 0" is what is going to appear when our application first launches.

**Label**

Text: Plain

Label

Color: Black Color

Font: System 17.0

Alignment: [Left] [Center] [Right]

Lines: 1

Behavior:  Enabled  Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: Horizontal 0 Vertical -1

**View**

Mode: Left

Tag: 0

Interaction:  User Interaction Enabled  Multiple Touch

Alpha: 1

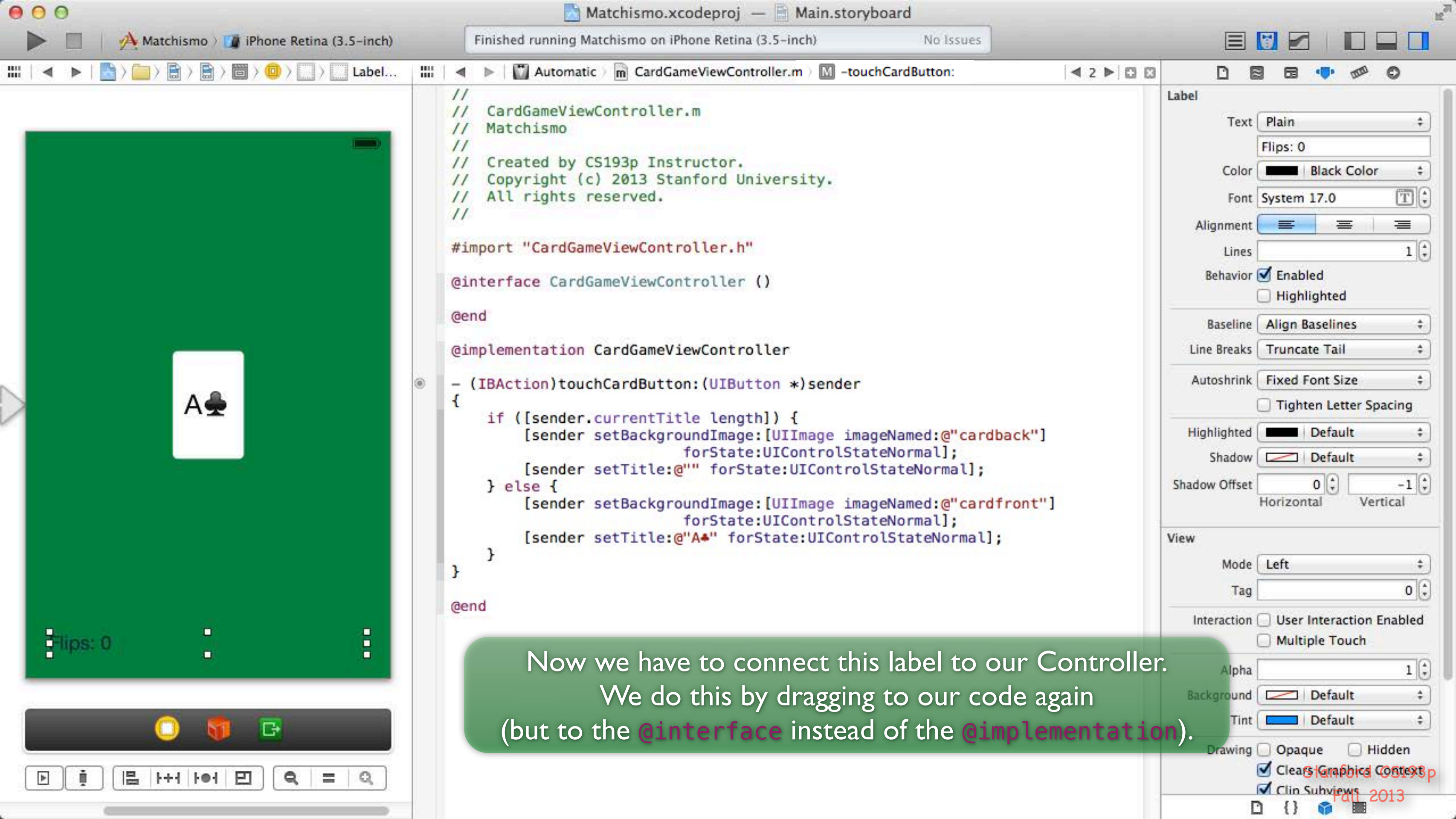
Background: Default

Tint: Default

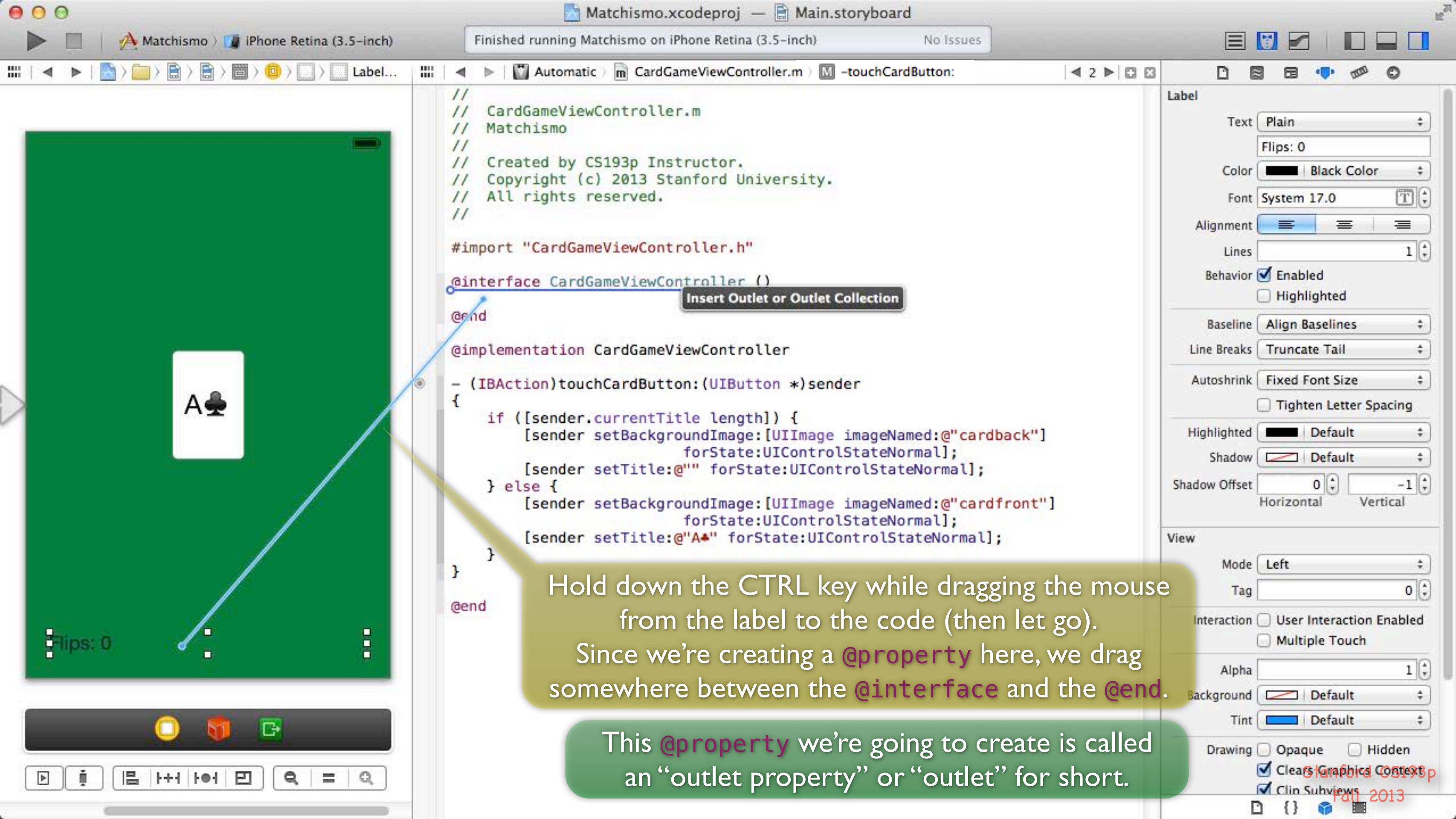
Drawing:  Opaque  Hidden

Clear Graphic Context

Clip Subviews



Now we have to connect this label to our Controller. We do this by dragging to our code again (but to the @interface instead of the @implementation).



```

//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@end

@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
}
@end

```

Insert Outlet or Outlet Collection

Hold down the CTRL key while dragging the mouse from the label to the code (then let go). Since we're creating a **@property** here, we drag somewhere between the **@interface** and the **@end**.

This **@property** we're going to create is called an "outlet property" or "outlet" for short.

**Label**

Text: Plain

Flips: 0

Color: Black Color

Font: System 17.0

Alignment: [Left, Center, Right]

Lines: 1

Behavior:  Enabled,  Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: 0 Horizontal, -1 Vertical

**View**

Mode: Left

Tag: 0

Interaction:  User Interaction Enabled,  Multiple Touch

Alpha: 1

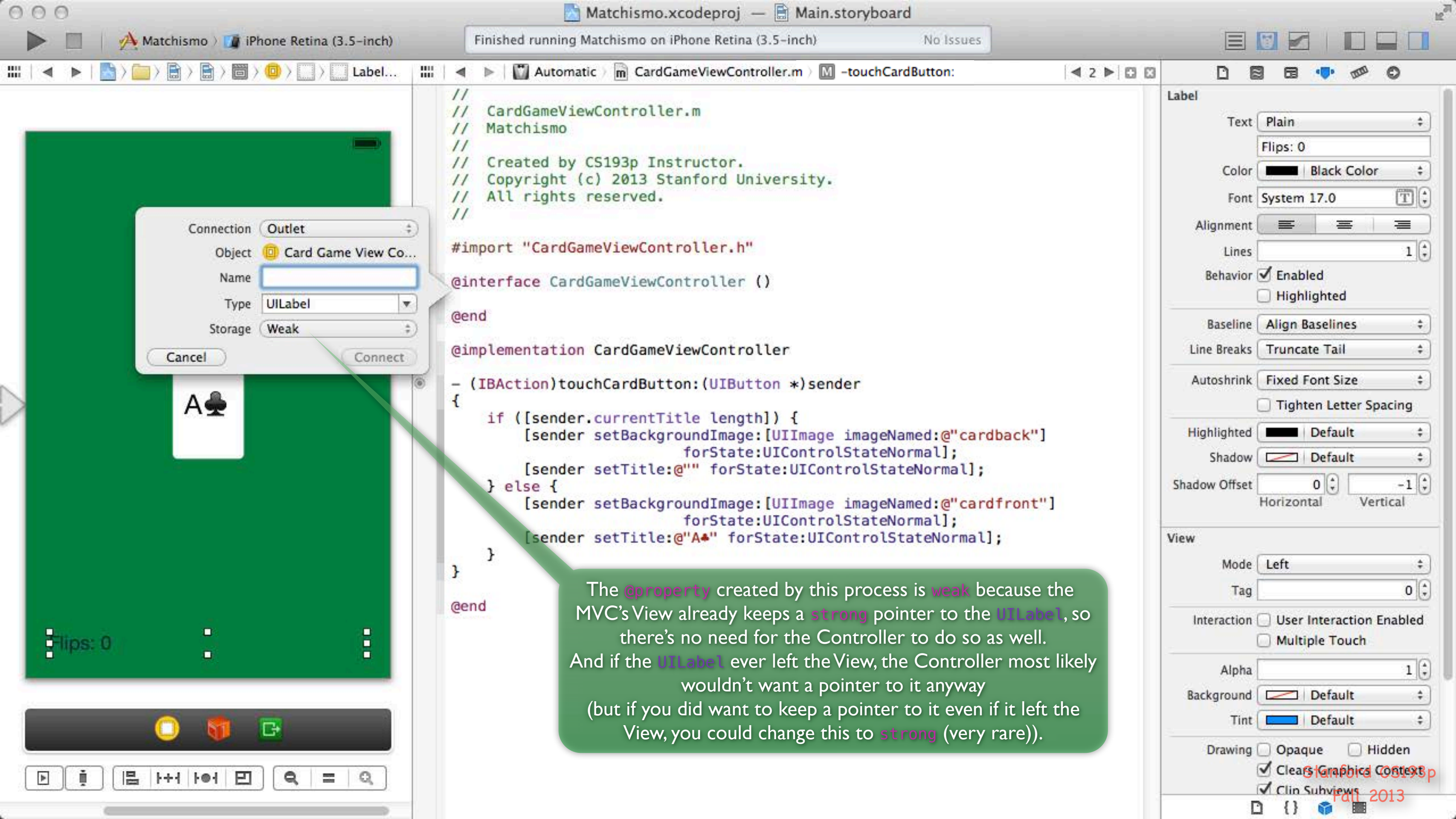
Background: Default

Tint: Default

Drawing:  Opaque,  Hidden

Clear Graphic Context

Clip Subviews



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

The `@property` created by this process is `weak` because the MVC's View already keeps a `strong` pointer to the `UILabel`, so there's no need for the Controller to do so as well. And if the `UILabel` ever left the View, the Controller most likely wouldn't want a pointer to it anyway (but if you did want to keep a pointer to it even if it left the View, you could change this to `strong` (very rare)).

**Label**

Text: Plain  
Flips: 0  
Color: Black Color  
Font: System 17.0  
Alignment: [Left, Center, Right]  
Lines: 1  
Behavior:  Enabled  
 Highlighted

Baseline: Align Baselines  
Line Breaks: Truncate Tail  
Autoshrink: Fixed Font Size  
 Tighten Letter Spacing

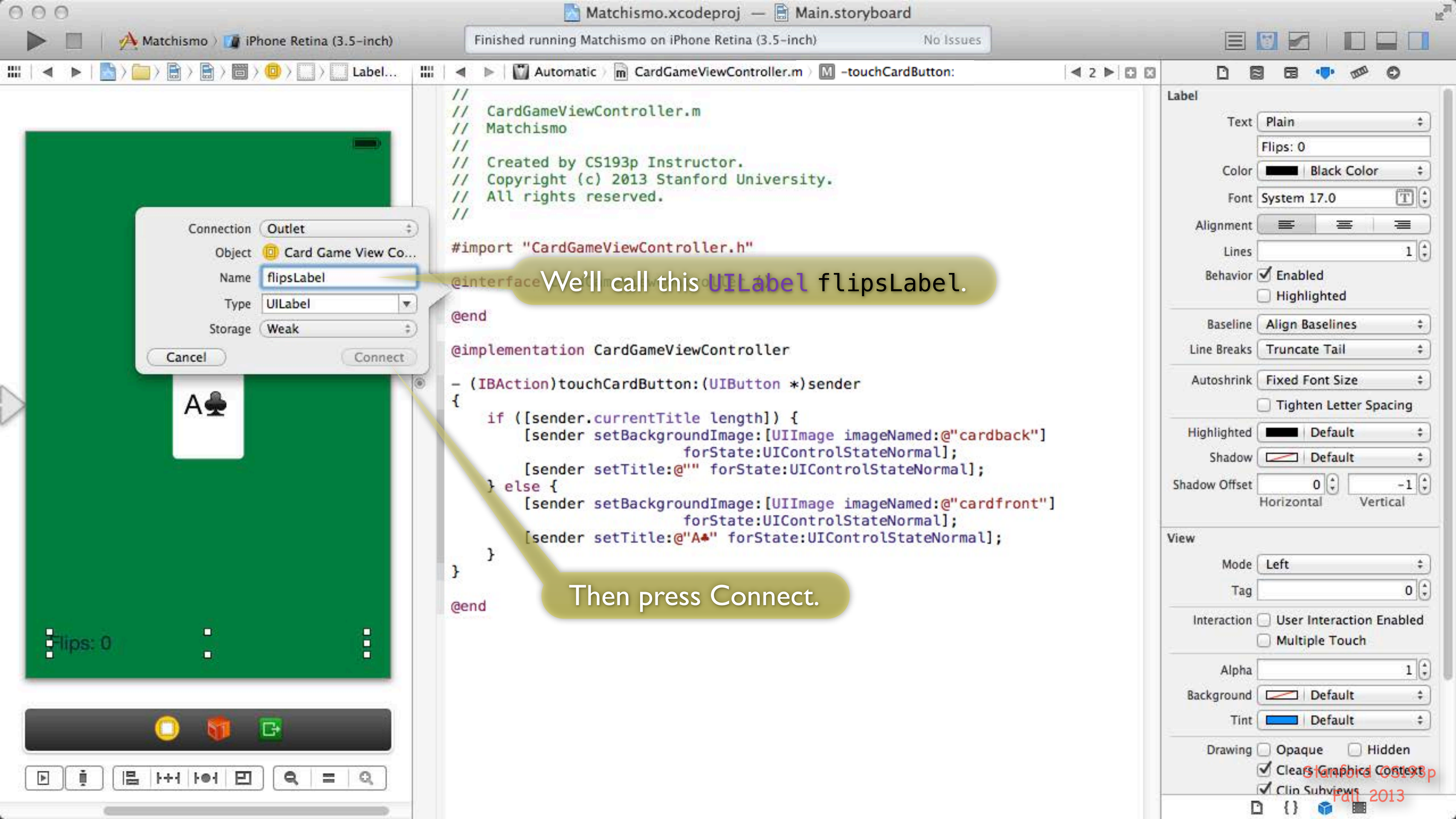
Highlighted: Default  
Shadow: Default  
Shadow Offset: 0 Horizontal, -1 Vertical

**View**

Mode: Left  
Tag: 0  
Interaction:  User Interaction Enabled  
 Multiple Touch

Alpha: 1  
Background: Default  
Tint: Default

Drawing:  Opaque  Hidden  
 Clear Graphic Context  
 Clip Subviews



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"  
  
@interface CardGameViewController  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

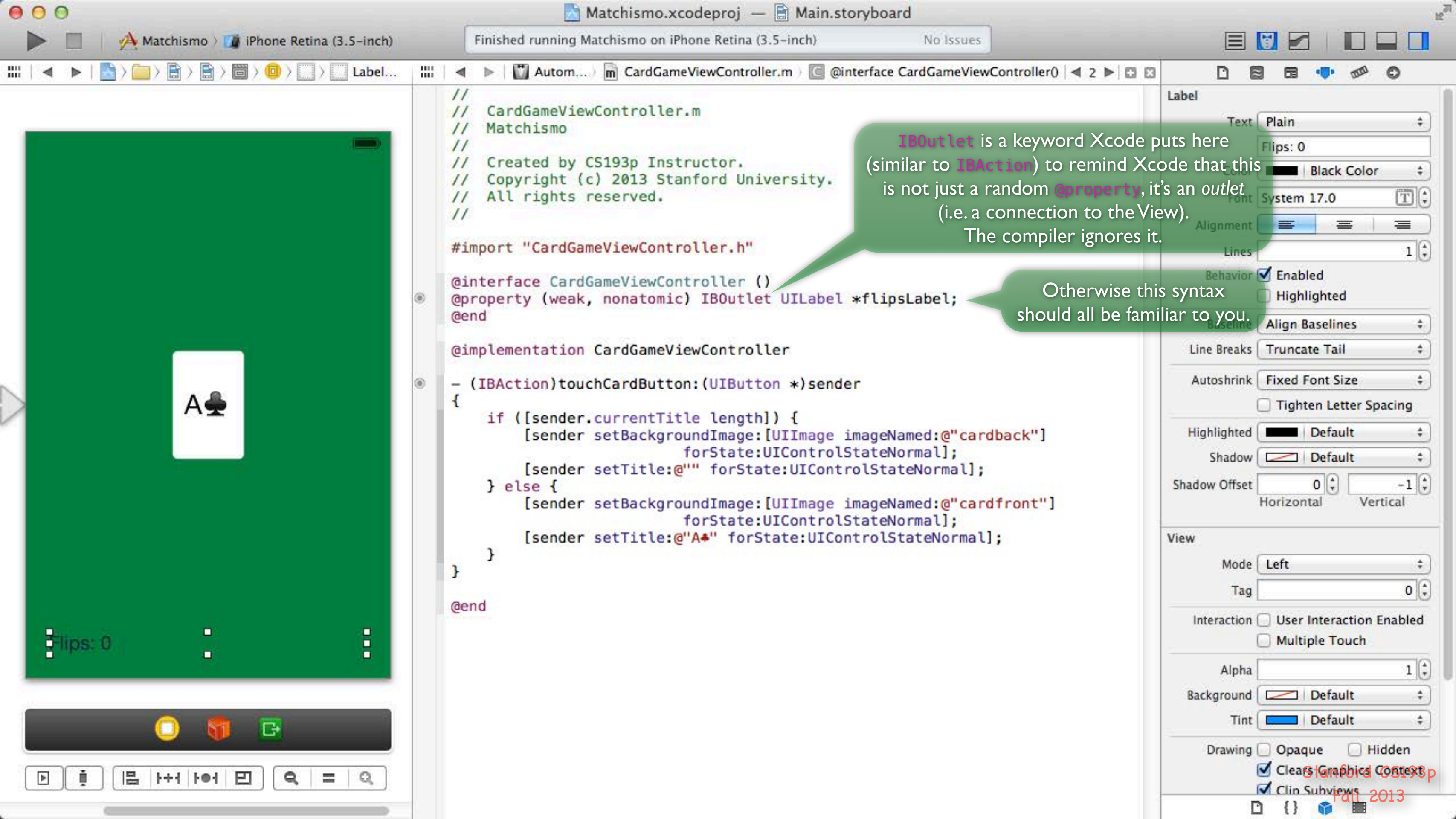
Connection: Outlet  
Object: Card Game View Co...  
Name: flipsLabel  
Type: UILabel  
Storage: Weak  
Buttons: Cancel, Connect

We'll call this UILabel flipsLabel.

Then press Connect.

Label

Text: Plain  
Flips: 0  
Color: Black Color  
Font: System 17.0  
Alignment: [Left, Center, Right]  
Lines: 1  
Behavior:  Enabled  
 Highlighted  
Baseline: Align Baselines  
Line Breaks: Truncate Tail  
Autoshrink: Fixed Font Size  
 Tighten Letter Spacing  
Highlighted: Default  
Shadow: Default  
Shadow Offset: 0 Horizontal, -1 Vertical  
View  
Mode: Left  
Tag: 0  
Interaction:  User Interaction Enabled  
 Multiple Touch  
Alpha: 1  
Background: Default  
Tint: Default  
Drawing:  Opaque  Hidden  
 Clear Graphic Context  
 Clip Subviews



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

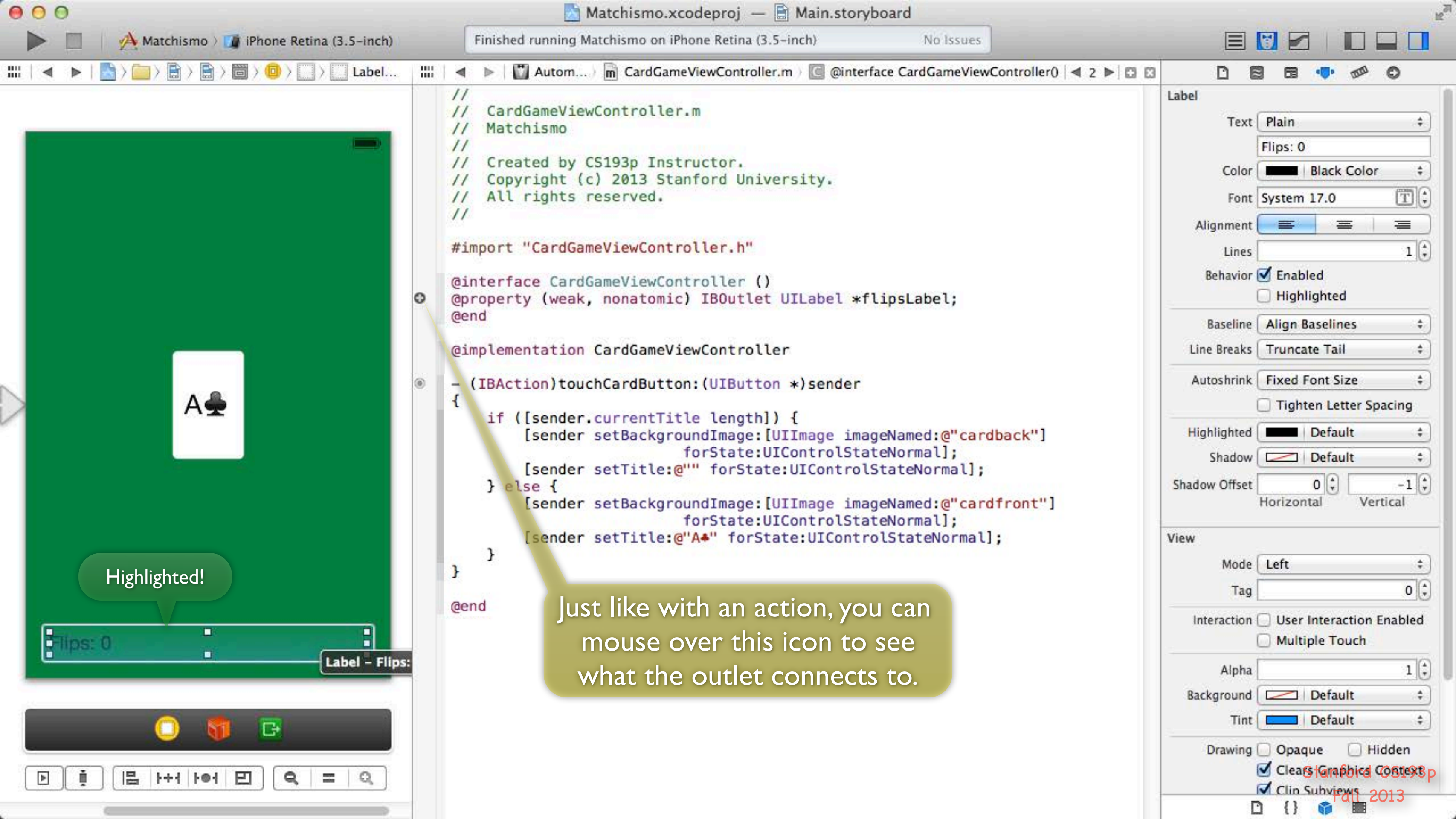
```
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

IBOutlet is a keyword Xcode puts here (similar to IBAction) to remind Xcode that this is not just a random @property, it's an outlet (i.e. a connection to the View). The compiler ignores it.

Otherwise this syntax should all be familiar to you.

Inspector for UILabel:

- Text: Plain
- Flips: 0
- Color: Black Color
- Font: System 17.0
- Alignment: Center
- Lines: 1
- Behavior:  Enabled,  Highlighted
- Baseline: Align Baselines
- Line Breaks: Truncate Tail
- Autoshrink: Fixed Font Size
- Tighten Letter Spacing
- Highlighted: Default
- Shadow: Default
- Shadow Offset: 0 Horizontal, -1 Vertical
- View: Mode Left, Tag 0
- Interaction:  User Interaction Enabled,  Multiple Touch
- Alpha: 1
- Background: Default
- Tint: Default
- Drawing:  Opaque,  Hidden
- Clear Graphic Context
- Clip Subviews



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

Just like with an action, you can mouse over this icon to see what the outlet connects to.

**Label**

Text: Plain

Flips: 0

Color: Black Color

Font: System 17.0

Alignment: [Left] [Center] [Right]

Lines: 1

Behavior:  Enabled  
 Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: 0 Horizontal, -1 Vertical

**View**

Mode: Left

Tag: 0

Interaction:  User Interaction Enabled  
 Multiple Touch

Alpha: 1

Background: Default

Tint: Default

Drawing:  Opaque  Hidden  
 Clear Graphic Context  
 Clip Subviews



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//
#import "CardGameViewController.h"
@interface CardGameViewController ()
@property (nonatomic) IBOutlet UILabel *flipsLabel;
@end

@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
}

@end
```

**Label**

Text: Plain

Flips: 0

Color: Black Color

Font: System 17.0

Alignment: [Left, Center, Right]

Lines: 1

Behavior:  Enabled,  Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size

Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: 0 Horizontal, -1 Vertical

---

**View**

Mode: Left

Tag: 0

Interaction:  User Interaction Enabled,  Multiple Touch

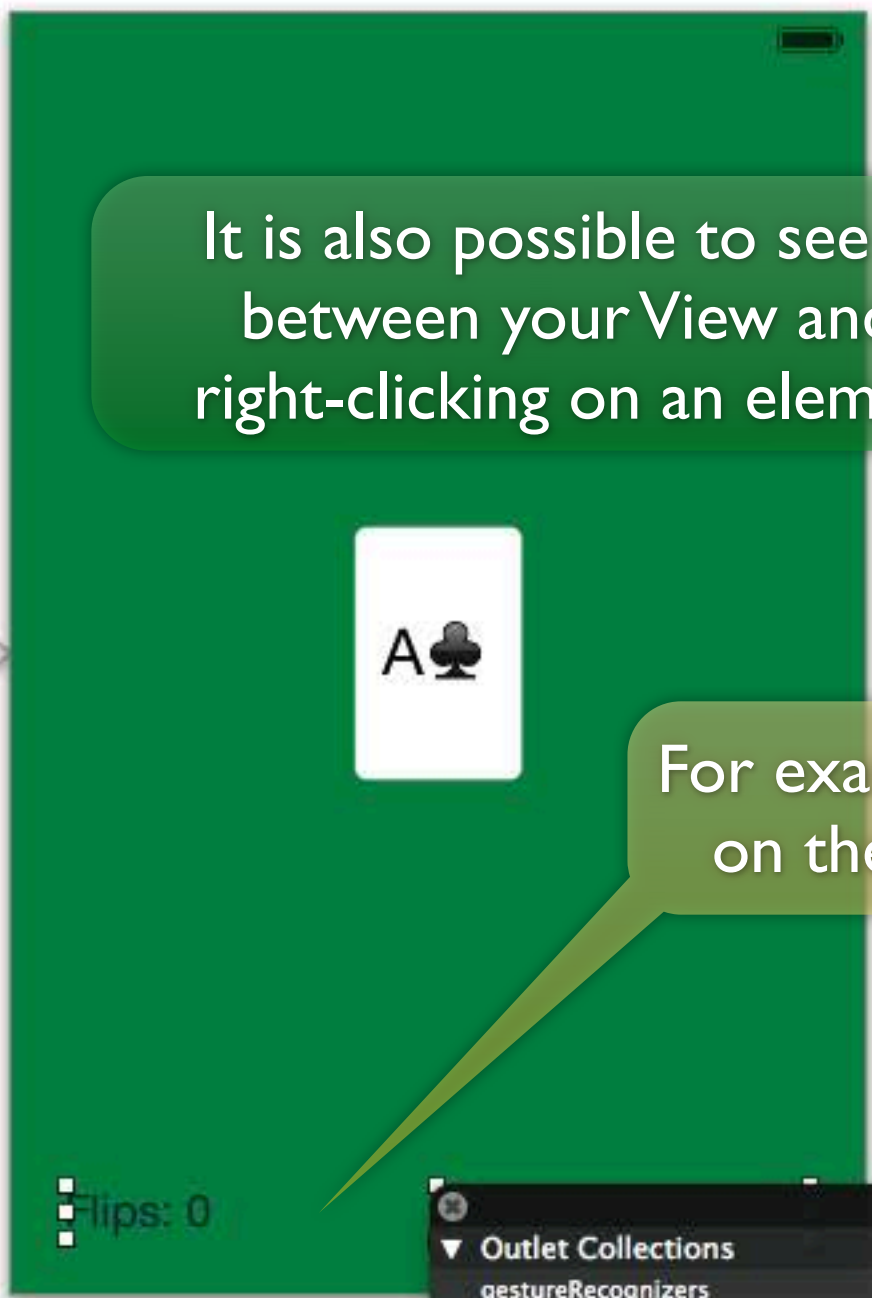
Alpha: 1

Background: Default

Tint: Default

Drawing:  Opaque,  Hidden,  Clear Graphic Context,  Clip Subviews

It is also possible to see the connections between your View and Controller by right-clicking on an element in your View.



For example, right-click on the Flips:0 label.

Here are all the connections to/from this UILabel. Notice the outlet flipsLabel.

Label - Flips: 0

- Outlet Collections
  - gestureRecognizers
- Referencing Outlets
  - flipsLabel — Card Game View Controller
  - New Referencing Outlet
- Referencing Outlet Collections
  - New Referencing Outlet Collection

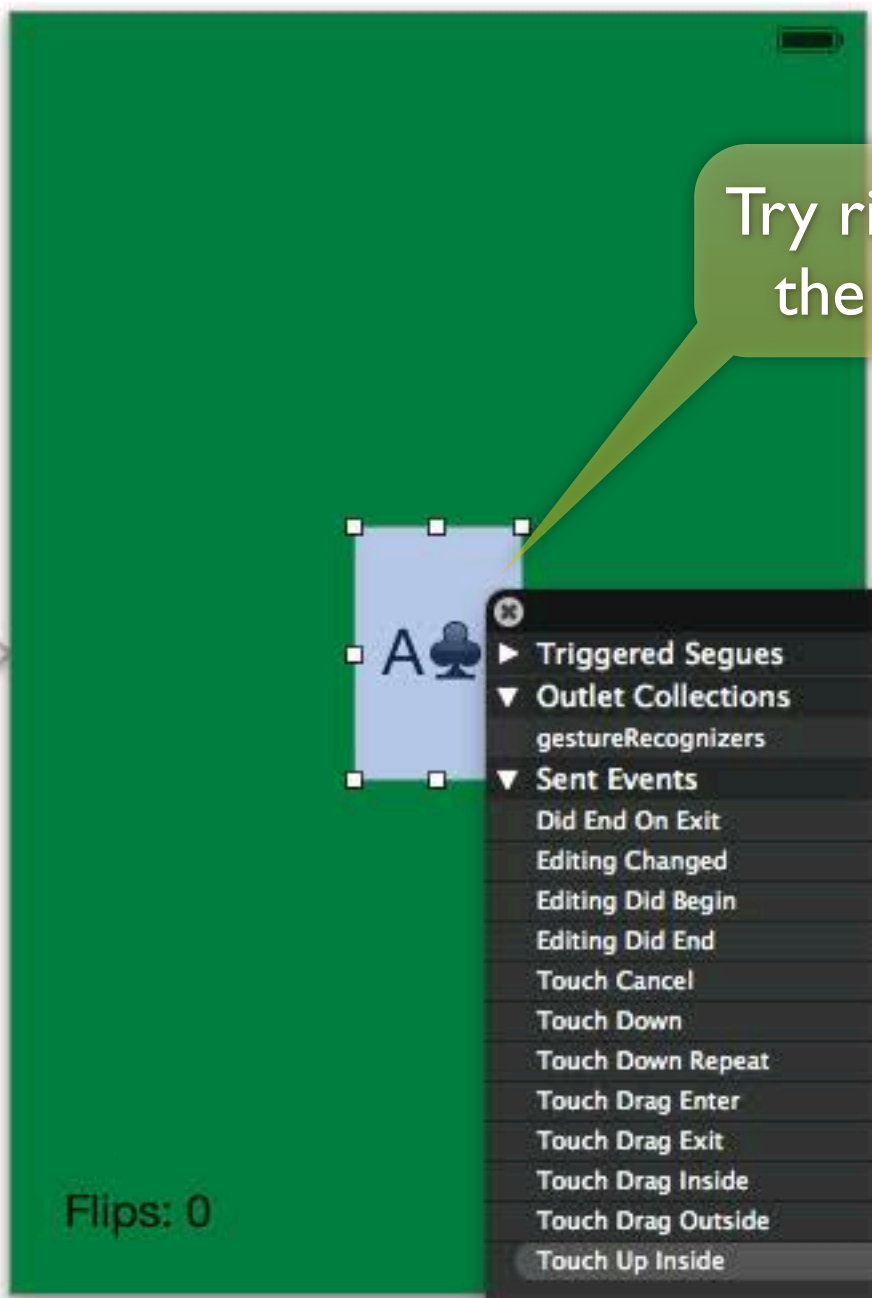
Try right-clicking on the card button.

You can “disconnect” connections by clicking on these little x’s. But don’t do it now! If you do so accidentally, just drag again from the View element to the appropriate code (method or **@property**).

It is a source of annoying bugs to forget to disconnect a no-longer-being-used connection. At runtime, you’ll get a crash with a complaint like “no such method, touchCardButton:”.

Here’s the touchCardButton: action connection.

You can actually ctrl-drag from these little circles to make connections too but it’s pretty rare to do it that way.



```
// CardGameViewController.m
// Matchismo
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *
@end
@implementation CardGameViewController
*) sender
{
    UIImage imageNamed:@"cardback"]
    UIControlStateNormal];
    UIControlStateNormal];
    UIImage imageNamed:@"cardfront"]
    UIControlStateNormal];
    UIControlStateNormal];
}
```

Button - A♣

- Triggered Segues
- Outlet Collections
  - gestureRecognizers
- Sent Events
  - Did End On Exit
  - Editing Changed
  - Editing Did Begin
  - Editing Did End
  - Touch Cancel
  - Touch Down
  - Touch Down Repeat
  - Touch Drag Enter
  - Touch Drag Exit
  - Touch Drag Inside
  - Touch Drag Outside
  - Touch Up Inside **\* Card Game View Controller touchCardButton:**
  - Touch Up Outside
  - Value Changed
- Referencing Outlets
  - New Referencing Outlet
- Referencing Outlet Collections
  - New Referencing Outlet Collection

Button

Type System

State Config Default

Title Plain

A♣

Font System 24.0

Text Color Black Color

Shadow Color Default

Image Default Image

Background cardfront

Shadow Offset 0.0 0.0

Width Height

Reverses On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break Truncate Middle

Edge Content

Inset 0 0

Top Bottom

0 0

Left Right

Control

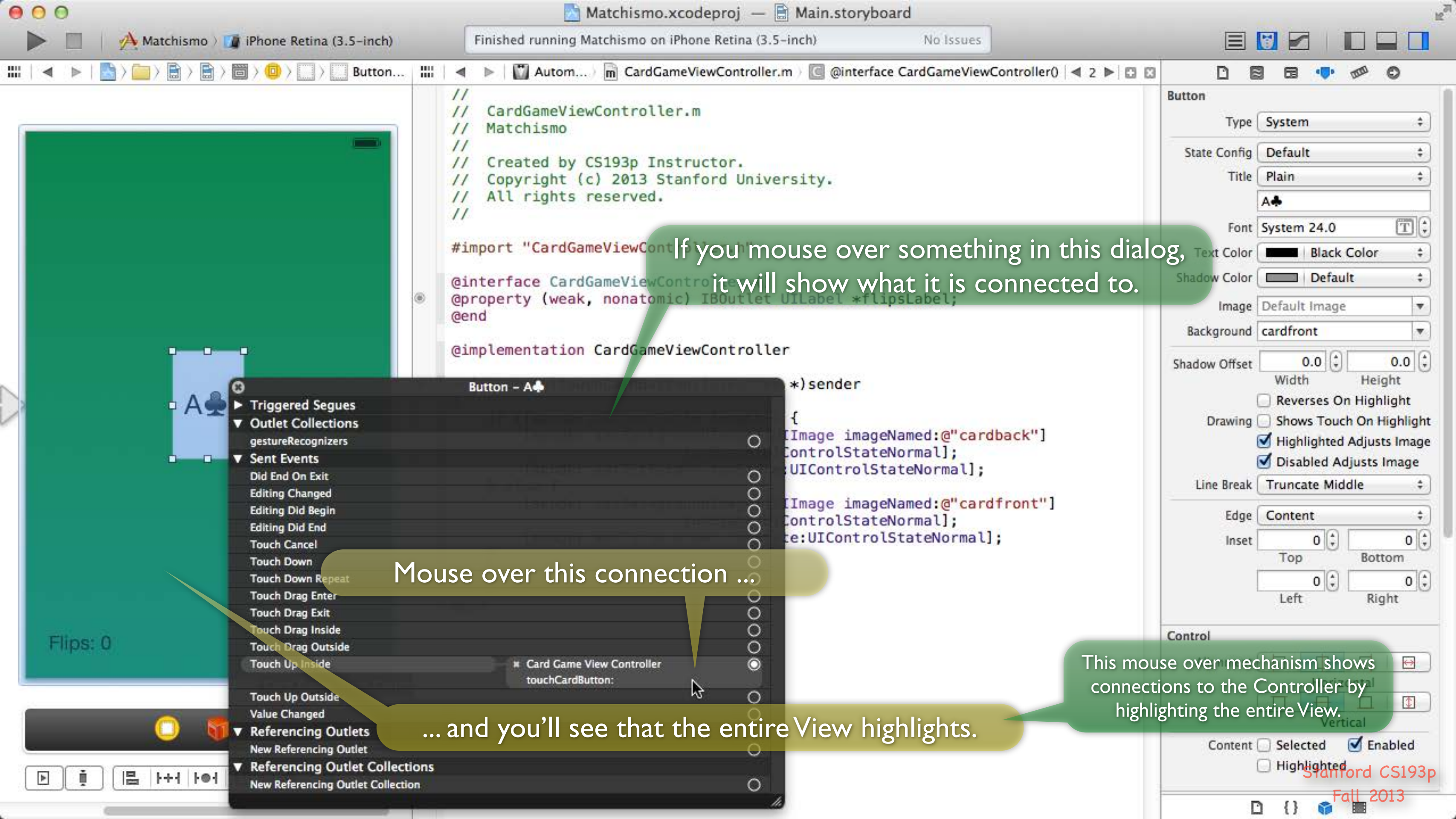
Alignment

Horizontal

Vertical

Selected Enabled

Highlighted



Finished running Matchismo on iPhone Retina (3.5-inch) No Issues

```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//
```

```
#import "CardGameViewController.h"

@interface CardGameViewController
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@end

@implementation CardGameViewController

- (IBAction) touchCardButton:(UIButton *)sender
{
    [UIImage imageNamed:@"cardback"]
    forState:UIControlStateNormal];
    [UIImage imageNamed:@"cardfront"]
    forState:UIControlStateNormal];
}
```

If you mouse over something in this dialog, it will show what it is connected to.

Button - A♣

- Triggered Segues
- Outlet Collections
- gestureRecognizers
- Sent Events
  - Did End On Exit
  - Editing Changed
  - Editing Did Begin
  - Editing Did End
  - Touch Cancel
  - Touch Down
  - Touch Down Repeat
  - Touch Drag Enter
  - Touch Drag Exit
  - Touch Drag Inside
  - Touch Drag Outside
  - Touch Up Inside
    - \* Card Game View Controller touchCardButton:
  - Touch Up Outside
  - Value Changed
- Referencing Outlets
- Referencing Outlet Collections

Mouse over this connection ...

... and you'll see that the entire View highlights.

Button

Type: System

State Config: Default

Title: Plain

A♣

Font: System 24.0

Text Color: Black Color

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 Width, 0.0 Height

Reverses On Highlight:

Drawing Shows Touch On Highlight:

Highlighted Adjusts Image:

Disabled Adjusts Image:

Line Break: Truncate Middle

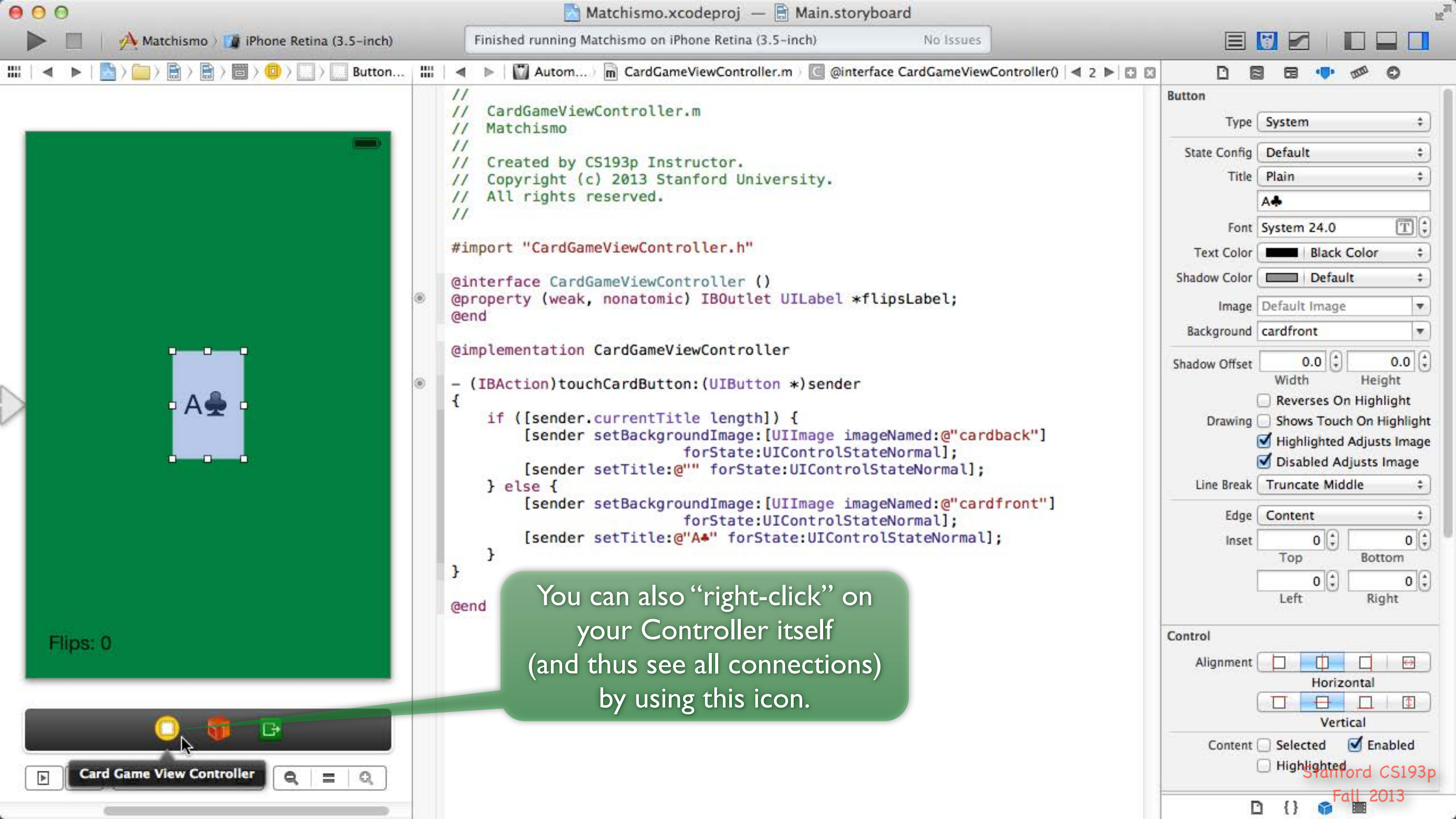
Edge: Content

Inset: 0 Top, 0 Bottom, 0 Left, 0 Right

Control

Content:  Selected:  Enabled:  Highlighted:

This mouse over mechanism shows connections to the Controller by highlighting the entire View.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

You can also “right-click” on your Controller itself (and thus see all connections) by using this icon.

**Button**

Type: System

State Config: Default

Title: Plain

A♣

Font: System 24.0

Text Color: Black Color

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width) 0.0 (Height)

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break: Truncate Middle

Edge: Content

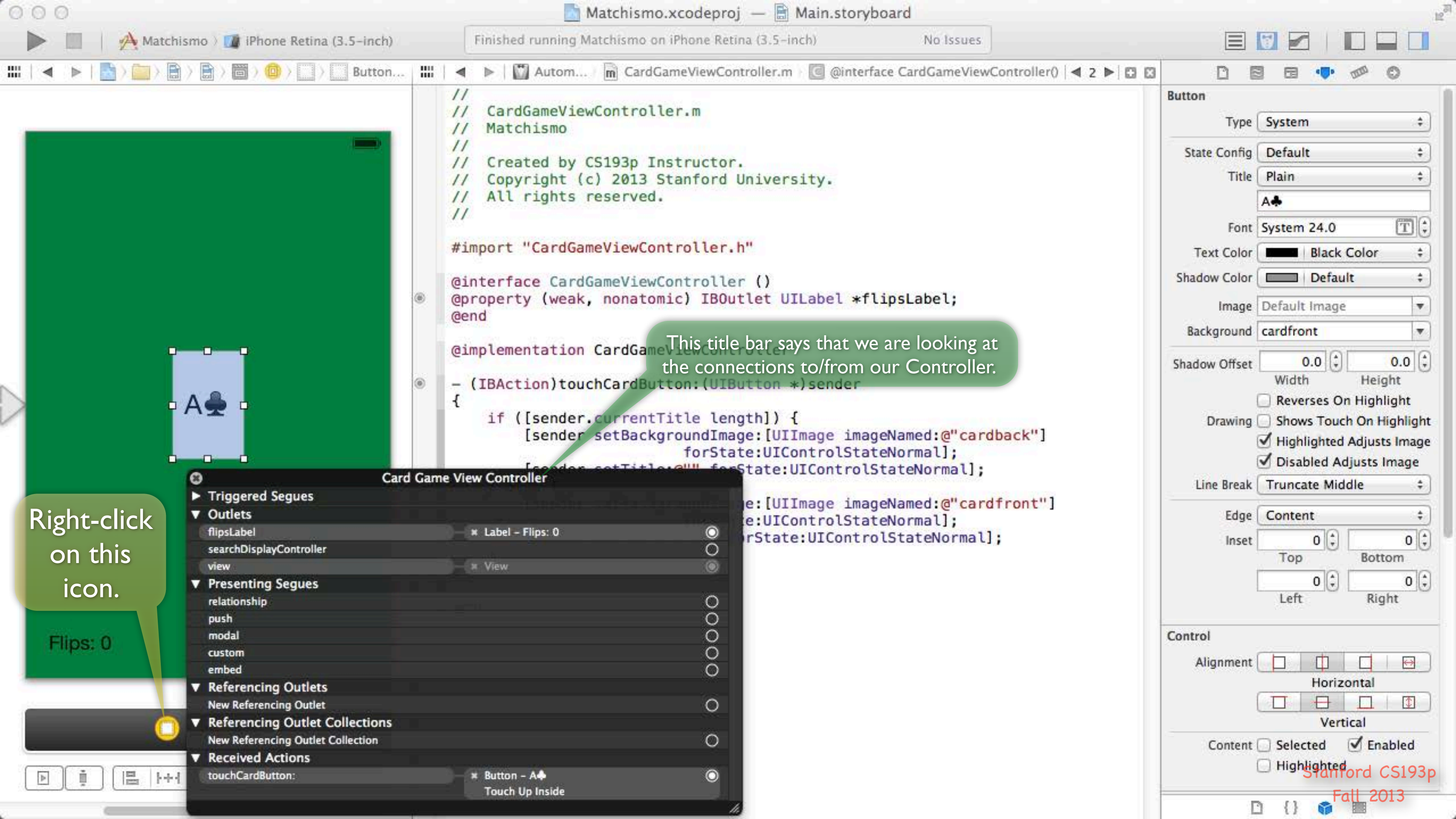
Inset: 0 (Top) 0 (Bottom) 0 (Left) 0 (Right)

**Control**

Alignment: Horizontal

Vertical

Content  Selected  Enabled  Highlighted



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    }  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                            forState:UIControlStateNormal];  
}
```

This title bar says that we are looking at the connections to/from our Controller.

Right-click on this icon.

Card Game View Controller

- Triggered Segues
- Outlets
  - flipsLabel \* Label - Flips: 0
  - searchDisplayController
  - view \* View
- Presenting Segues
  - relationship
  - push
  - modal
  - custom
  - embed
- Referencing Outlets
  - New Referencing Outlet
- Referencing Outlet Collections
  - New Referencing Outlet Collection
- Received Actions
  - touchCardButton: \* Button - A♣ Touch Up Inside

Flips: 0

Button

Type System

State Config Default

Title Plain

A♣

Font System 24.0

Text Color Black Color

Shadow Color Default

Image Default Image

Background cardfront

Shadow Offset 0.0 0.0

Width Height

Reverses On Highlight

Drawing Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break Truncate Middle

Edge Content

Inset 0 0

Top Bottom

Left Right

Control

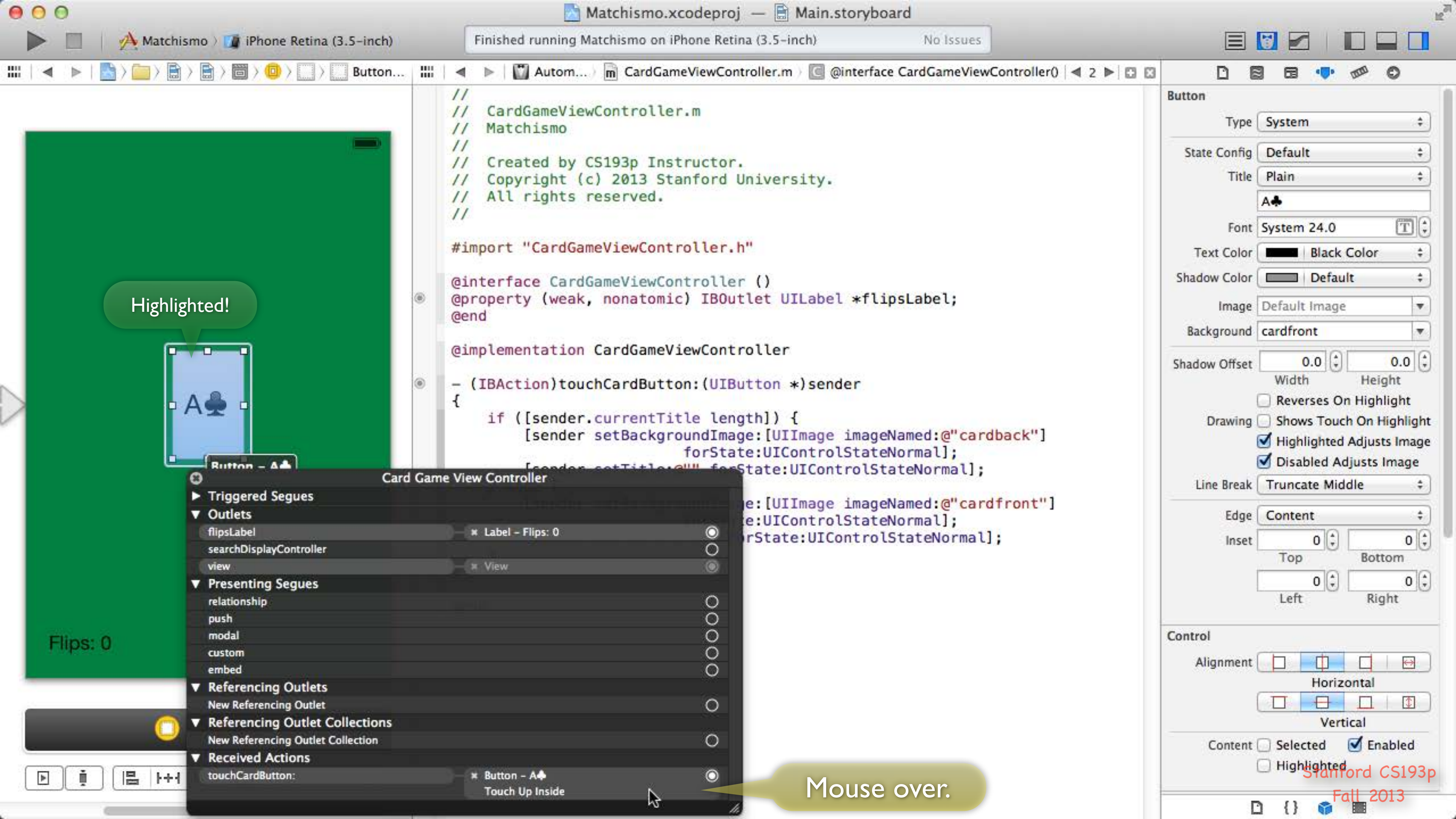
Alignment

Horizontal

Vertical

Content Selected Enabled

Highlighted



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    }  
    [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                            forState:UIControlStateNormal];  
}
```

**Button**

Type: System

State Config: Default

Title: Plain

A♣

Font: System 24.0

Text Color: Black Color

Shadow Color: Default

Image: Default Image

Background: cardfront

Shadow Offset: 0.0 (Width) 0.0 (Height)

Reverses On Highlight

Drawing  Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break: Truncate Middle

Edge: Content

Inset: 0 (Top) 0 (Bottom) 0 (Left) 0 (Right)

**Control**

Alignment: Horizontal

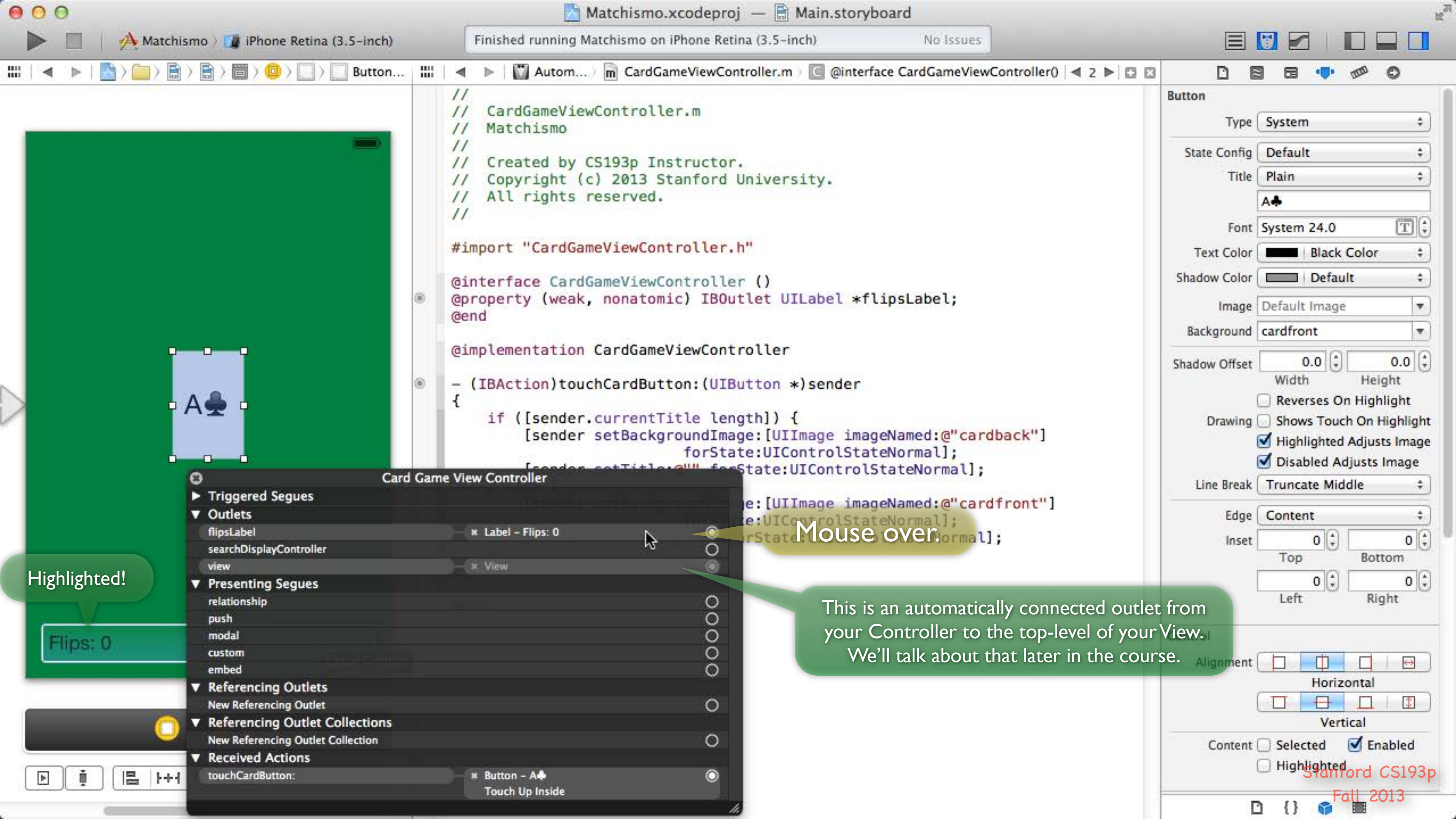
Vertical

Content  Selected  Enabled

Highlighted

Highlighted!

Mouse over.



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@end

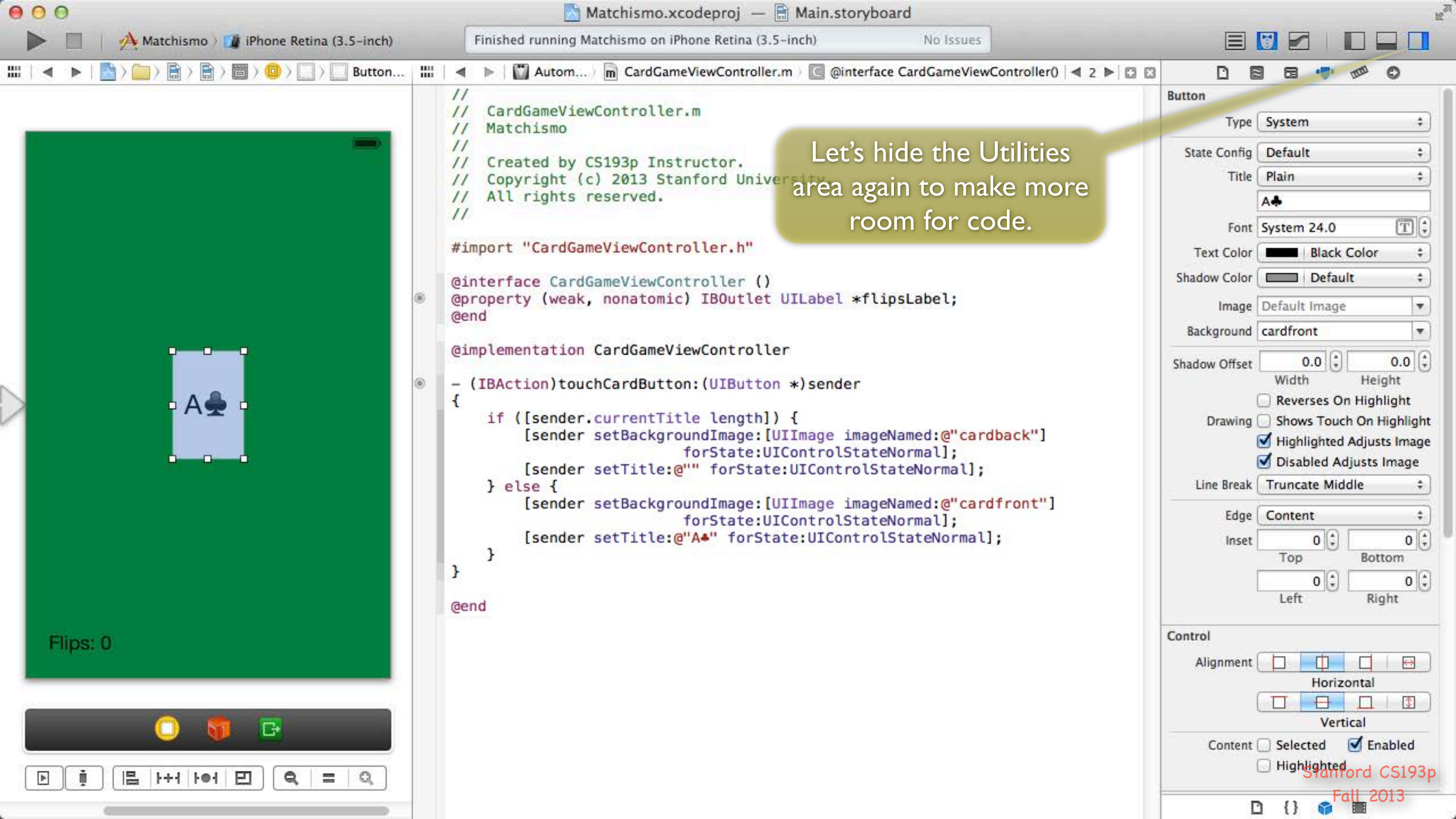
@implementation CardGameViewController

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
}
```

Highlighted!

Mouse over

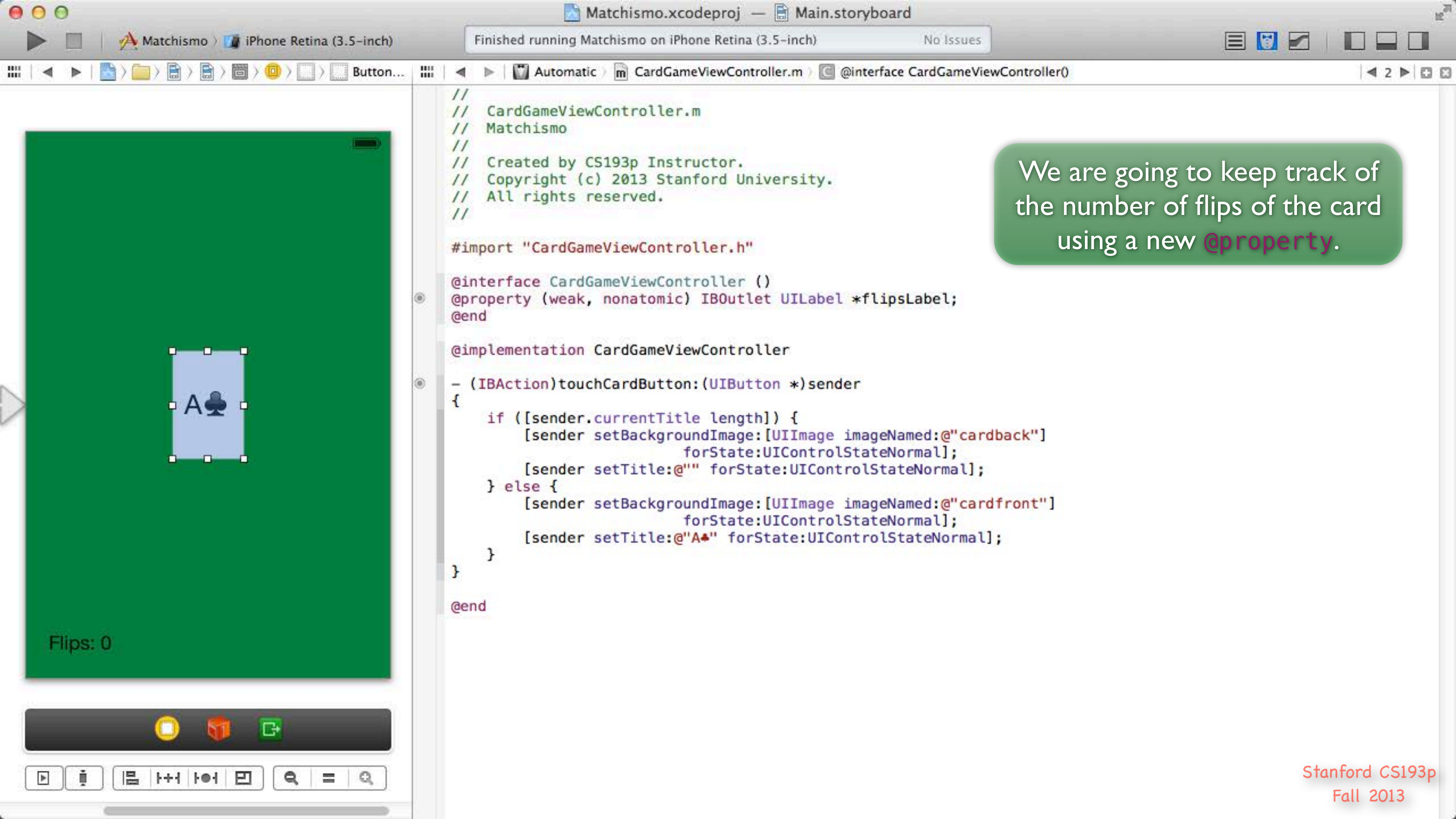
This is an automatically connected outlet from your Controller to the top-level of your View. We'll talk about that later in the course.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

Let's hide the Utilities area again to make more room for code.





```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

```
#import "CardGameViewController.h"
```

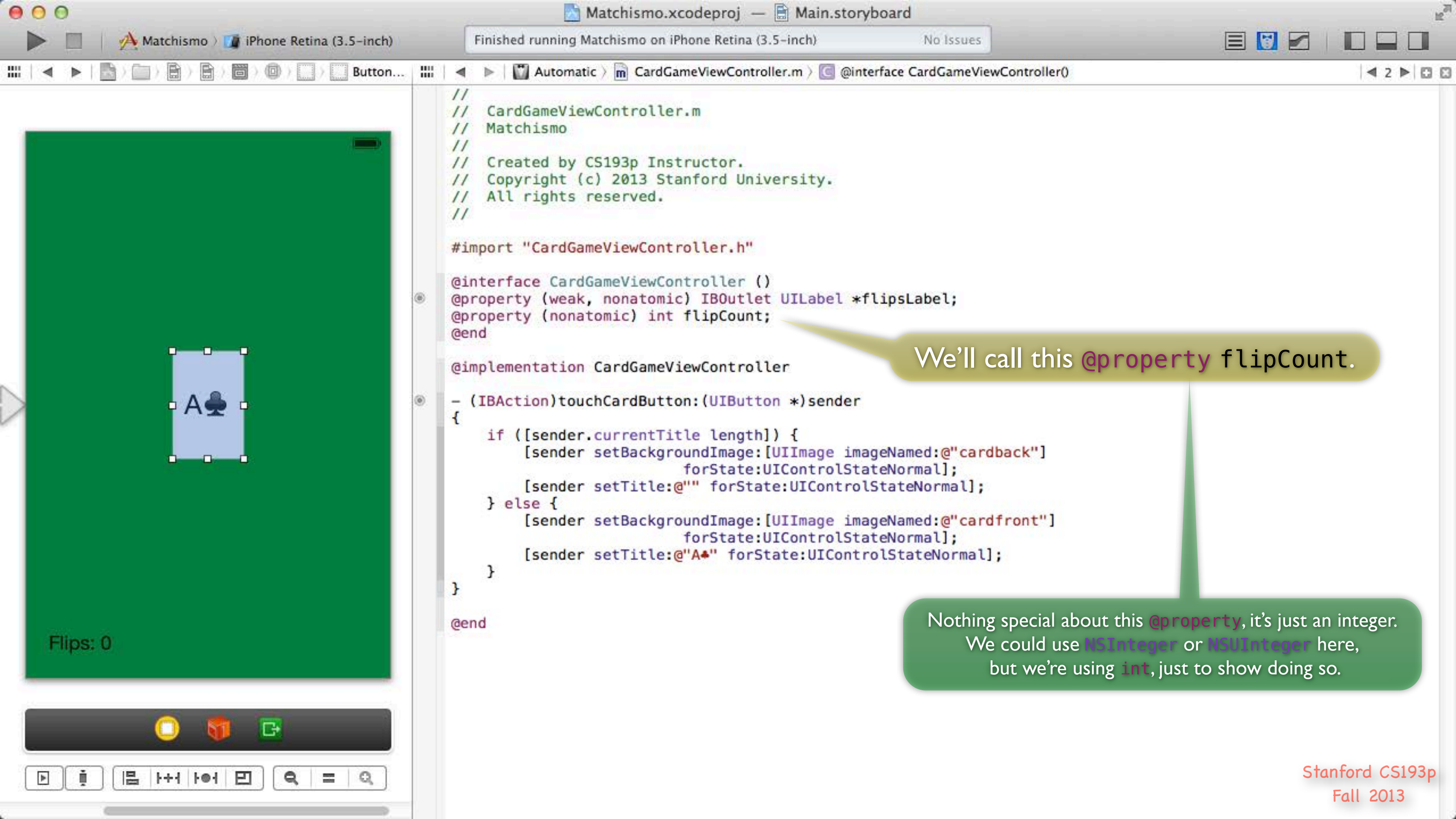
```
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@end
```

```
@implementation CardGameViewController
```

```
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}
```

```
@end
```

We are going to keep track of the number of flips of the card using a new @property.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

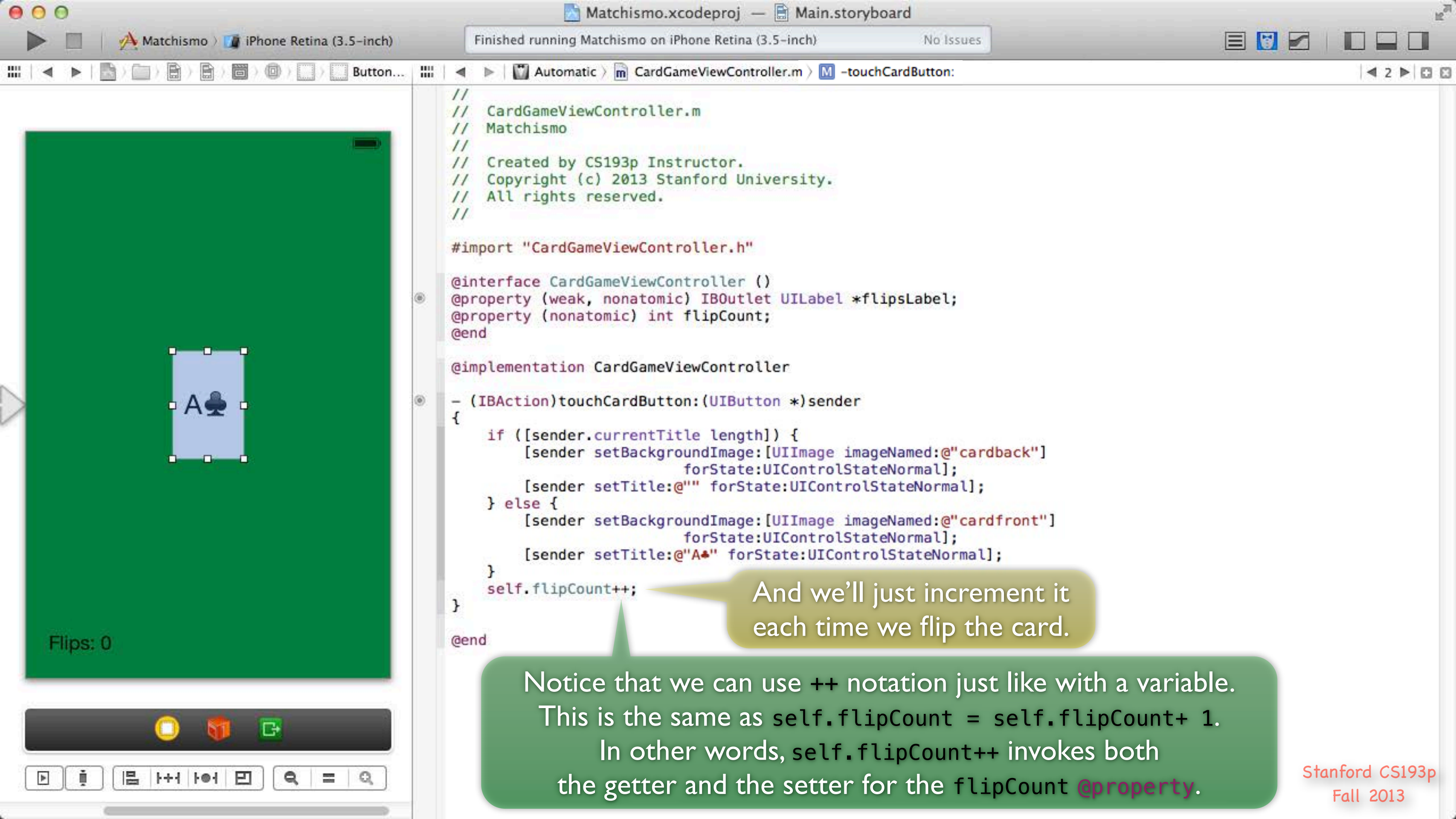
```
#import "CardGameViewController.h"
```

```
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@property (nonatomic) int flipCount;  
@end
```

We'll call this @property flipCount.

```
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
}  
  
@end
```

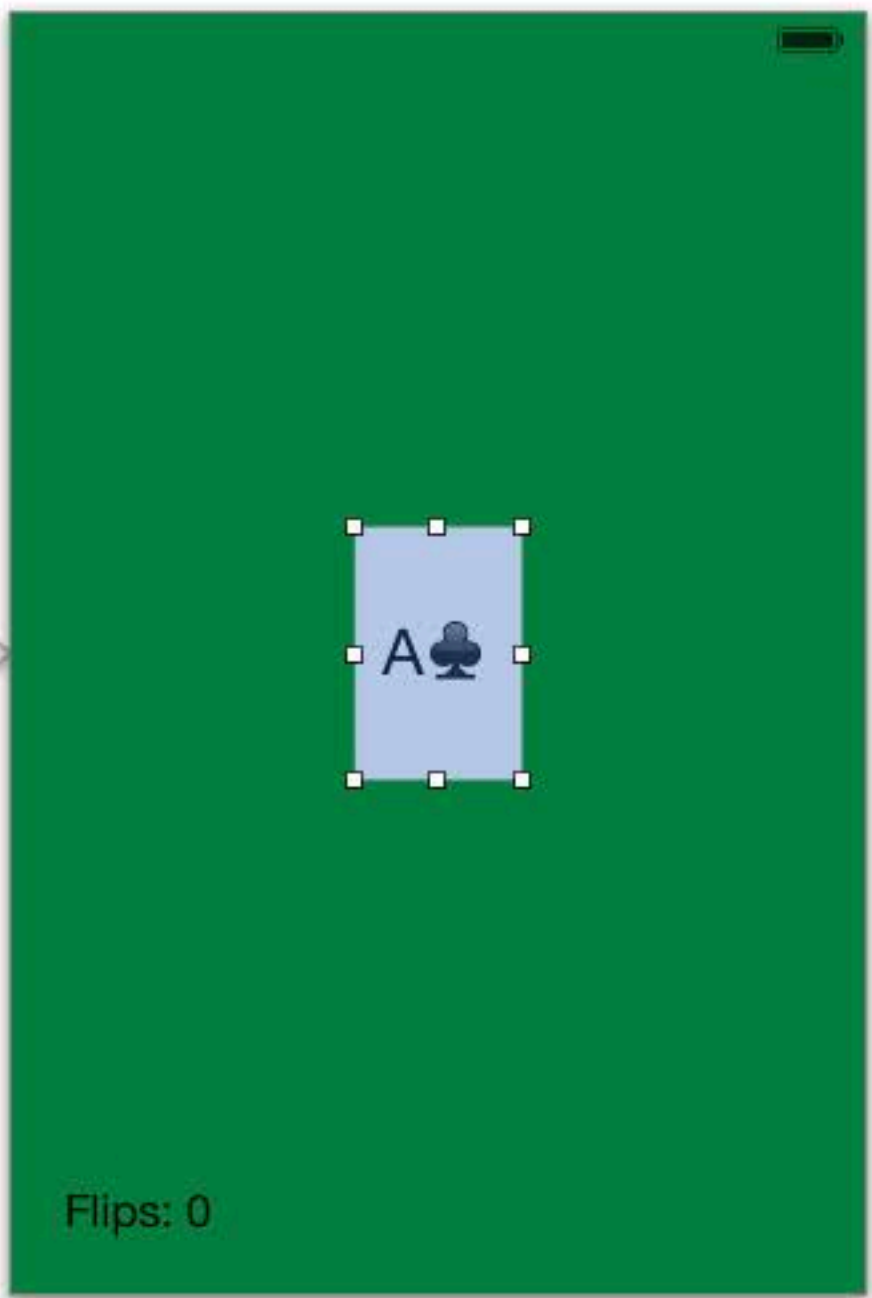
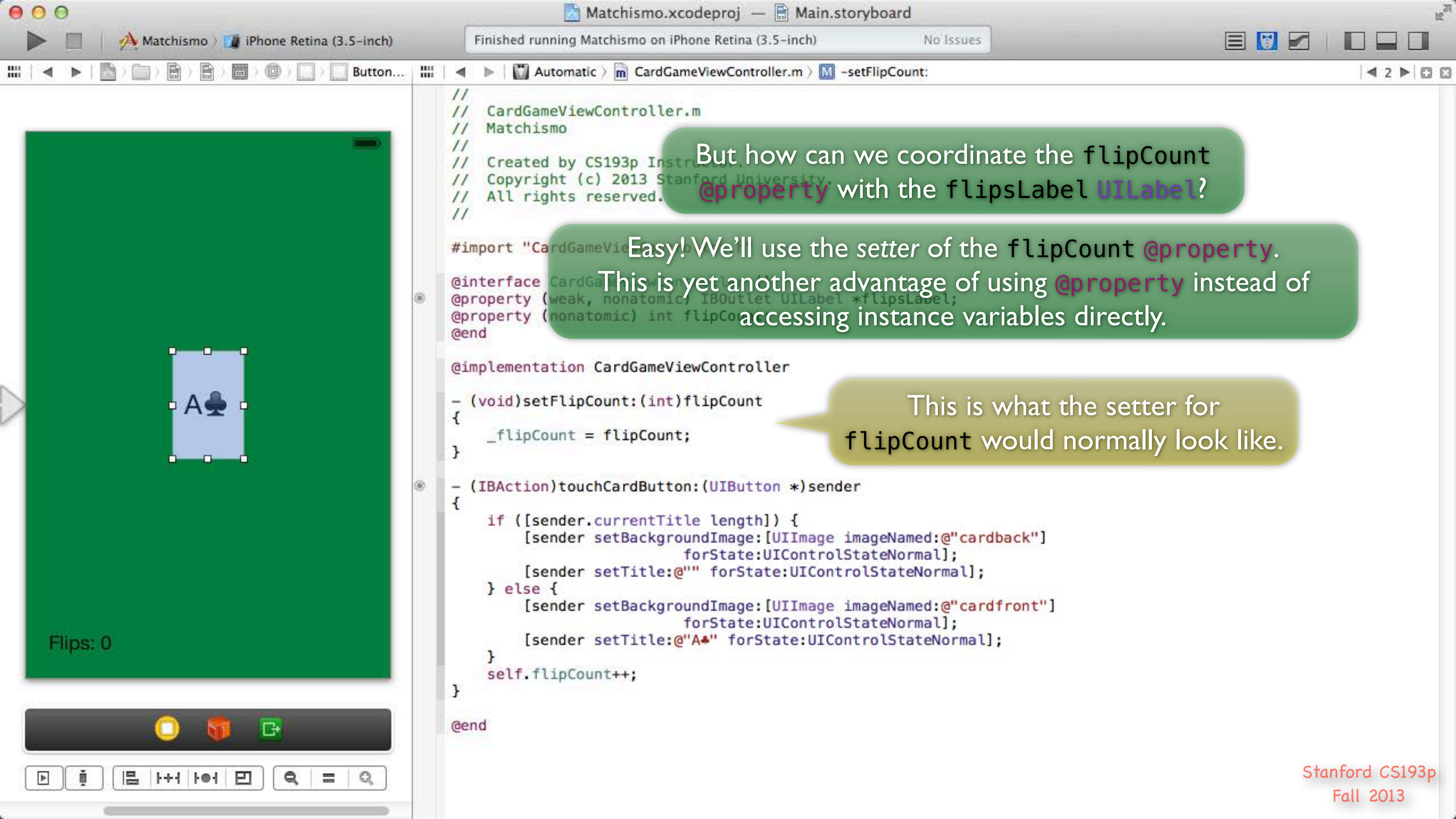
Nothing special about this @property, it's just an integer. We could use NSInteger or NSUInteger here, but we're using int, just to show doing so.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@property (nonatomic) int flipCount;  
@end  
  
@implementation CardGameViewController  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
    self.flipCount++;  
}  
  
@end
```

And we'll just increment it each time we flip the card.

Notice that we can use ++ notation just like with a variable. This is the same as self.flipCount = self.flipCount+ 1. In other words, self.flipCount++ invokes both the getter and the setter for the flipCount @property.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

But how can we coordinate the flipCount @property with the flipsLabel UILabel?

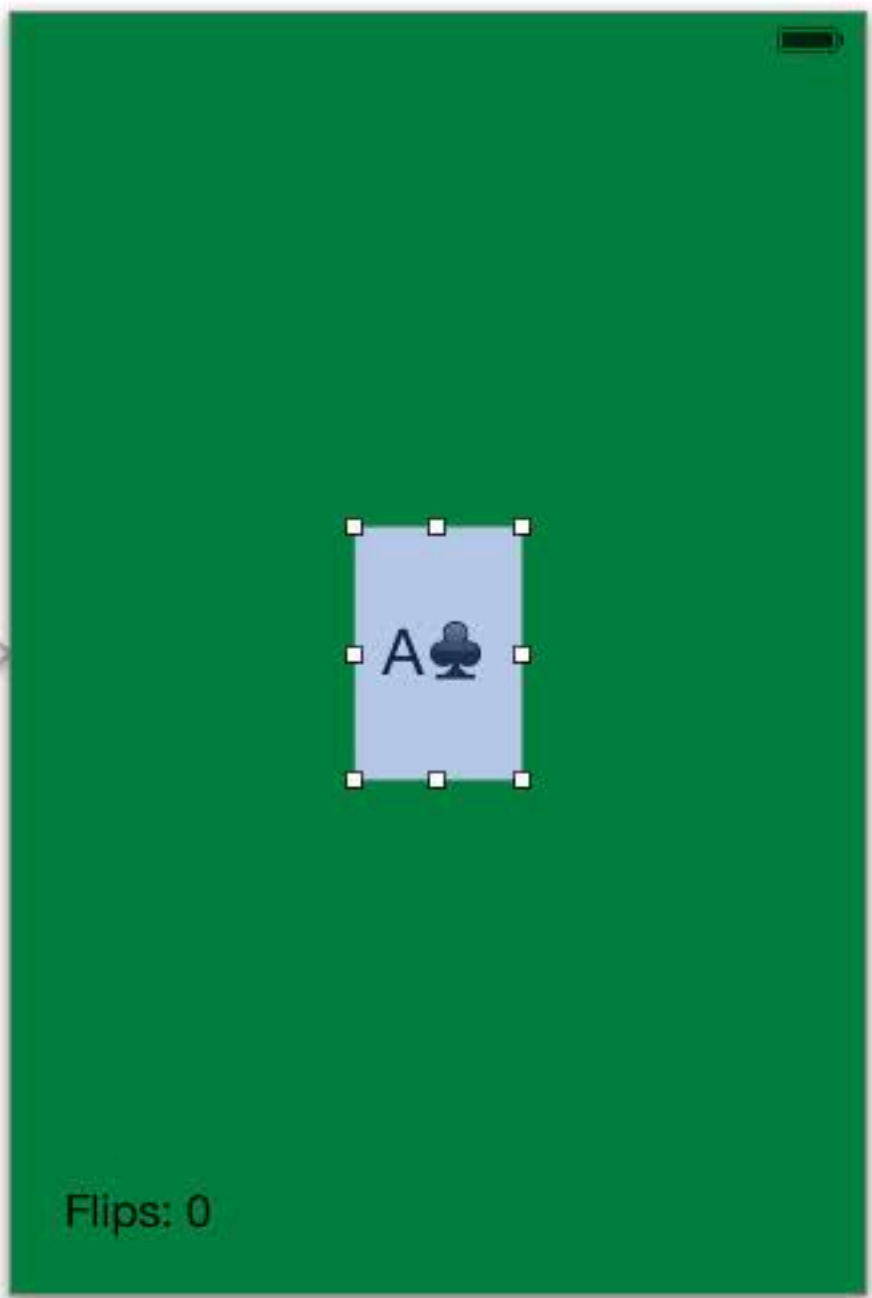
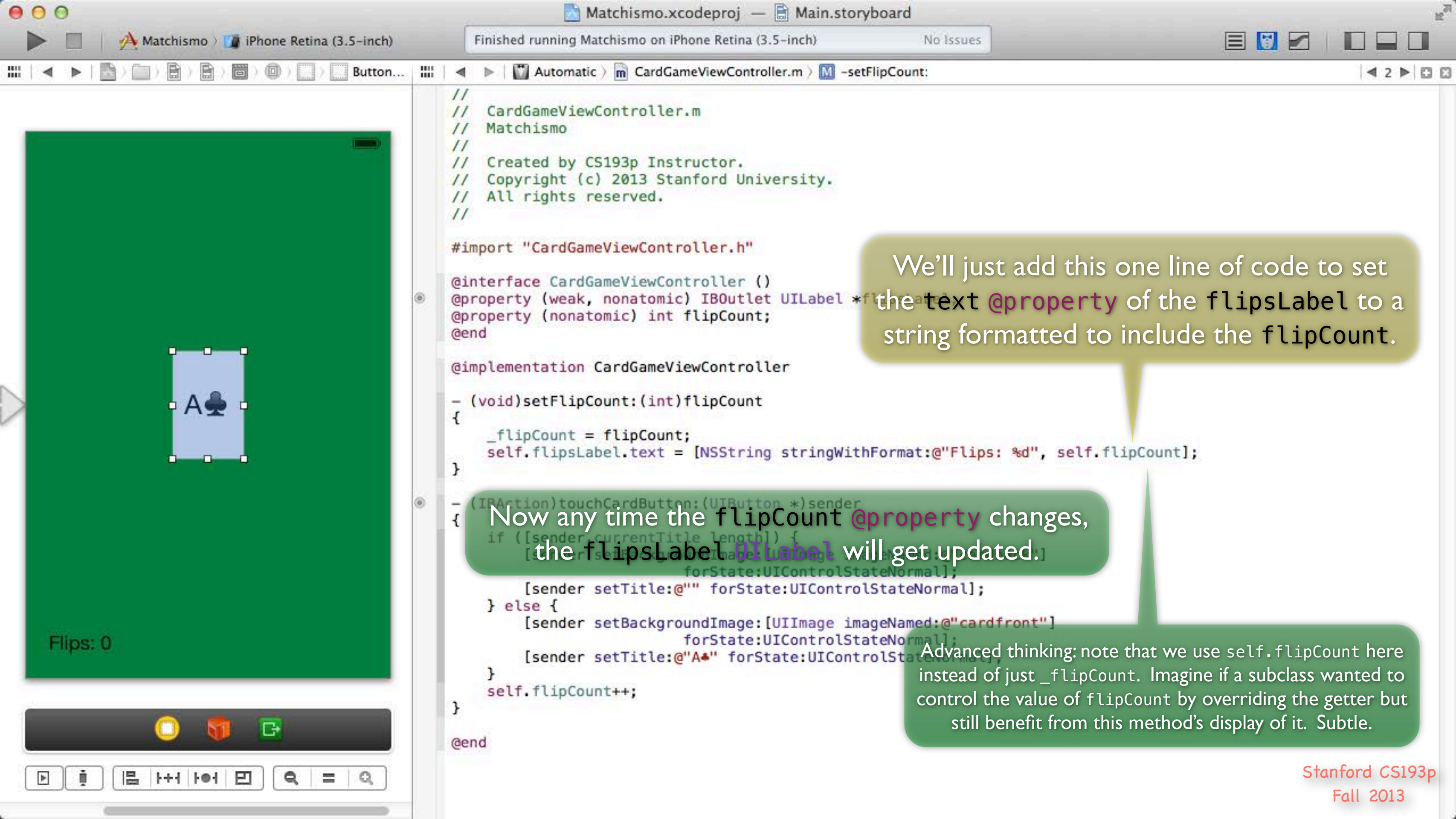
Easy! We'll use the setter of the flipCount @property. This is yet another advantage of using @property instead of accessing instance variables directly.

```
#import "CardGameView...  
  
@interface CardGameViewController  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@property (nonatomic) int flipCount;  
@end
```

This is what the setter for flipCount would normally look like.

```
@implementation CardGameViewController  
  
- (void)setFlipCount:(int)flipCount  
{  
    _flipCount = flipCount;  
}  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
    self.flipCount++;  
}  
  
@end
```





```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//
```

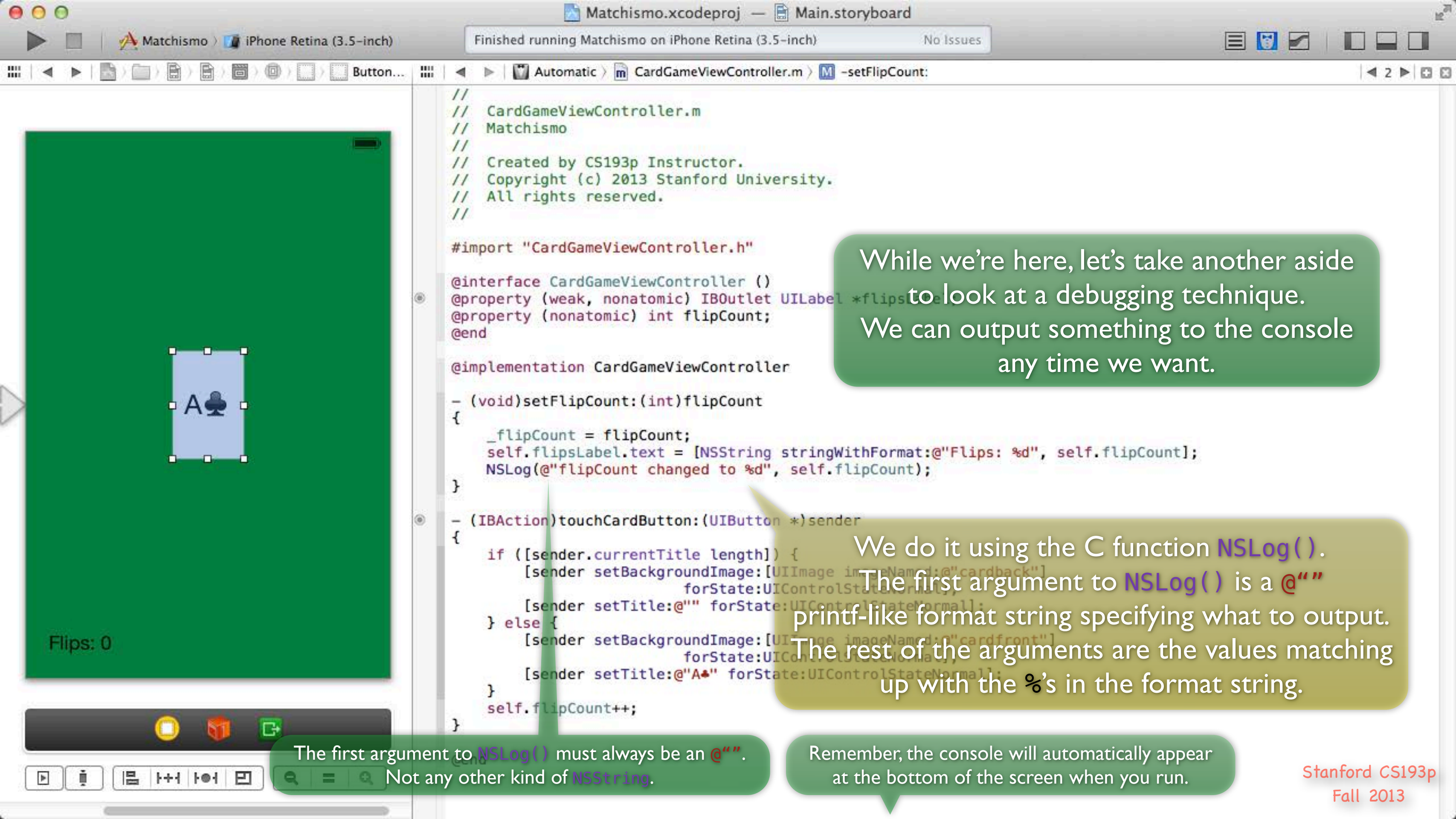
We'll just add this one line of code to set the text **@property** of the flipsLabel to a string formatted to include the flipCount.

```
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@property (nonatomic) int flipCount;  
@end  
  
@implementation CardGameViewController  
  
- (void)setFlipCount:(int)flipCount  
{  
    _flipCount = flipCount;  
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.flipCount];  
}
```

Now any time the flipCount **@property** changes, the flipsLabel **UILabel** will get updated.

```
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    if ([sender.currentTitle length]) {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
    } else {  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
                                forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
    }  
    self.flipCount++;  
}  
  
@end
```

Advanced thinking: note that we use self.flipCount here instead of just \_flipCount. Imagine if a subclass wanted to control the value of flipCount by overriding the getter but still benefit from this method's display of it. Subtle.



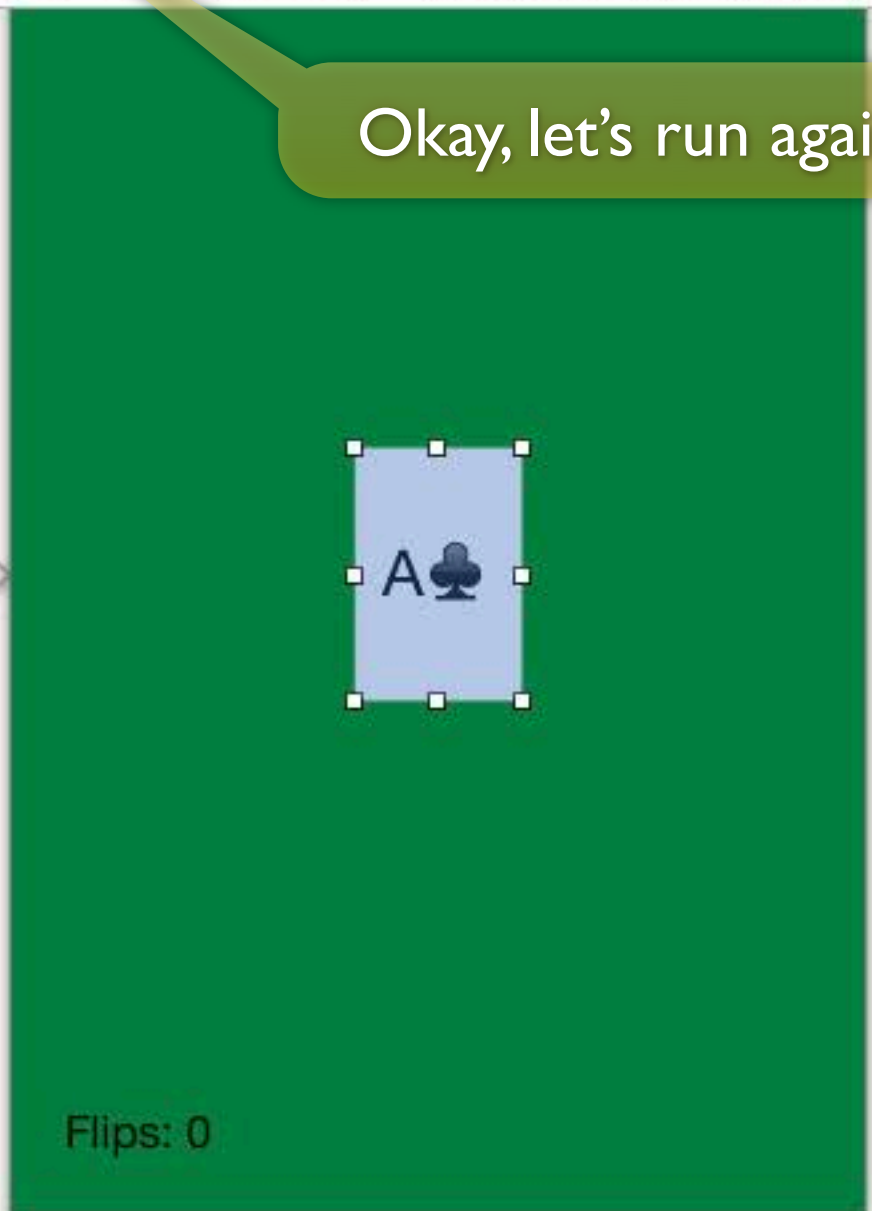
While we're here, let's take another aside to look at a debugging technique. We can output something to the console any time we want.

We do it using the C function `NSLog()`. The first argument to `NSLog()` is a `@""` printf-like format string specifying what to output. The rest of the arguments are the values matching up with the `%`'s in the format string.

The first argument to `NSLog()` must always be an `@""`. Not any other kind of `NSString`.

Remember, the console will automatically appear at the bottom of the screen when you run.

Okay, let's run again!



```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

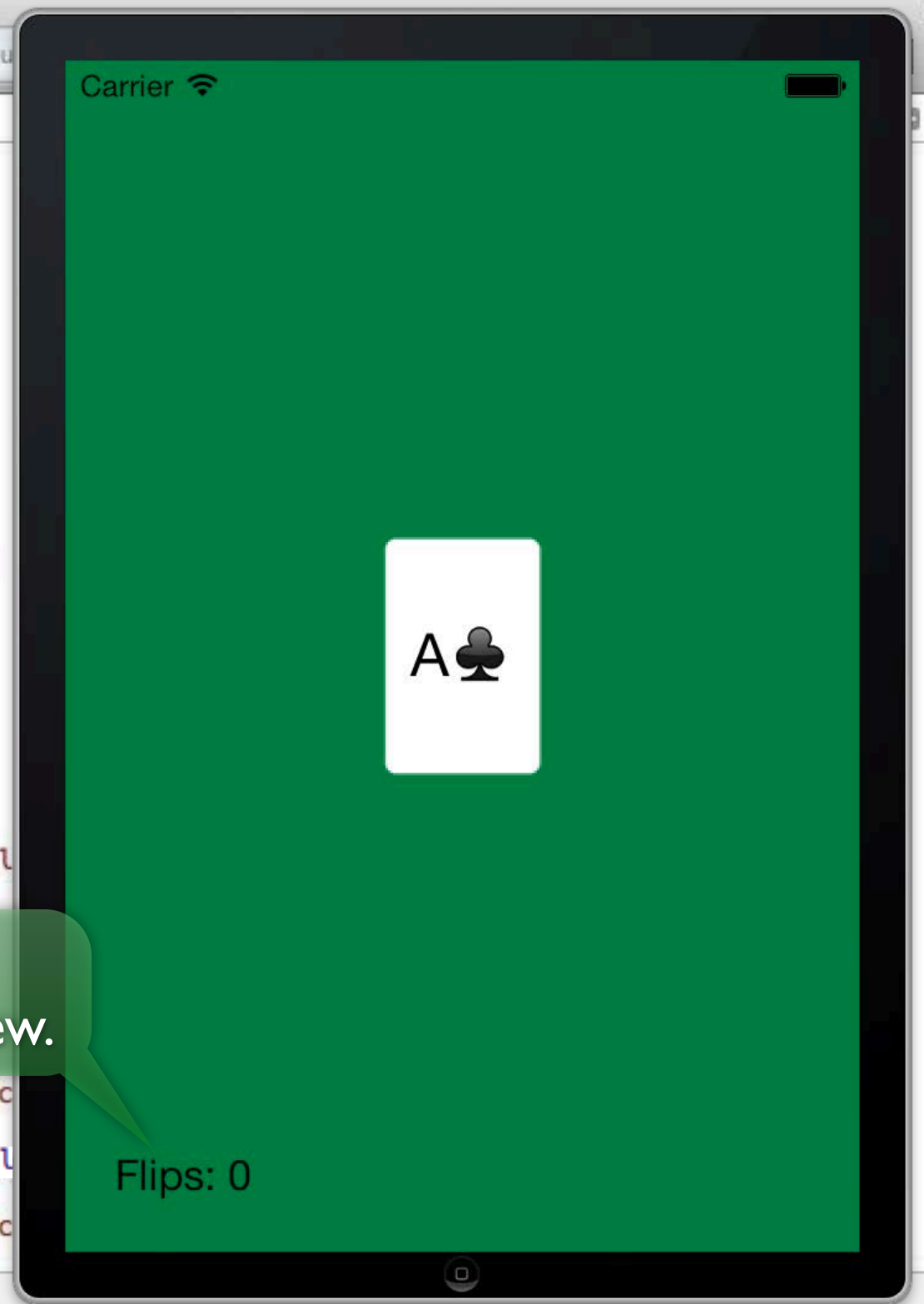
@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@property (nonatomic) int flipCount;
@end

@implementation CardGameViewController

- (void)setFlipCount:(int)flipCount
{
    _flipCount = flipCount;
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.flipCount];
    NSLog(@"flipCount changed to %d", self.flipCount);
}

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardBack"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardFront"]
                               forState:UIControlStateNormal];
    }
}
```

Note this starts out with whatever we typed in our View.



```

//
//  CardGameViewController.m
//  Matchismo
//
//  Created by CS193p Instructor.
//  Copyright (c) 2013 Stanford University.
//  All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@property (nonatomic) int flipCount;
@end

@implementation CardGameViewController

- (void)setFlipCount:(int)flipCount
{
    _flipCount = flipCount;
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.flipCount];
    NSLog(@"flipCount changed to %d", self.flipCount);
}

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardBack"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardFront"]
                               forState:UIControlStateNormal];
    }
}

```

2013-00-00 14:29:14.411 Matchismo[72114:a0b] flipCount changed to 1

And here's the output of the NSLog().



Matchismo.xcodeproj — Main.storyboard

Running Matchismo on iPhone Retina (3.5-inch) No Issues

Matchismo > iPhone Retina (3.5-inch)

Automatic > CardGameViewController.m > -setFlipCount:

```
//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@property (nonatomic) int flipCount;
@end

@implementation CardGameViewController

- (void)setFlipCount:(int)flipCount
{
    _flipCount = flipCount;
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.flipCount];
    NSLog(@"flipCount changed to %d", self.flipCount);
}

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardBack"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardFront"]
                               forState:UIControlStateNormal];
    }
}

@end
```

Carrier

Flips: 0

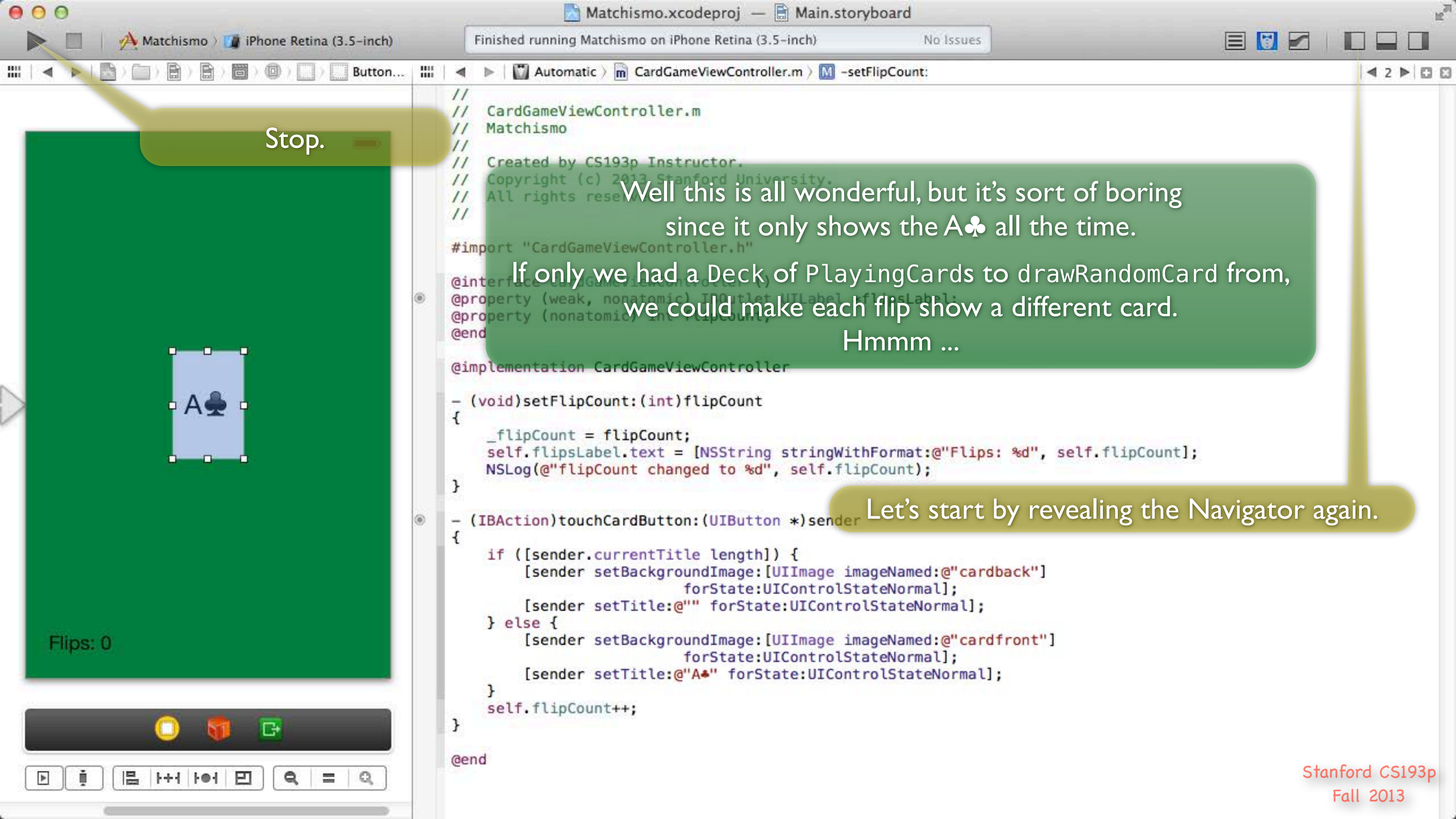
Flips: 2

Touch again.

Changes again.

```
2013-00-00 14:29:14.411 Matchismo[72114:a0b] flipCount changed to 1
2013-00-00 14:29:14.834 Matchismo[72114:a0b] flipCount changed to 2
```

And outputs here again.



Stop.

Well this is all wonderful, but it's sort of boring since it only shows the A♣ all the time.

If only we had a Deck of PlayingCards to drawRandomCard from, we could make each flip show a different card.

Hmmm ...

Let's start by revealing the Navigator again.

```

//
// CardGameViewController.m
// Matchismo
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University.
// All rights reserved.
//

#import "CardGameViewController.h"

@interface CardGameViewController ()
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;
@property (nonatomic) int flipCount;
@end

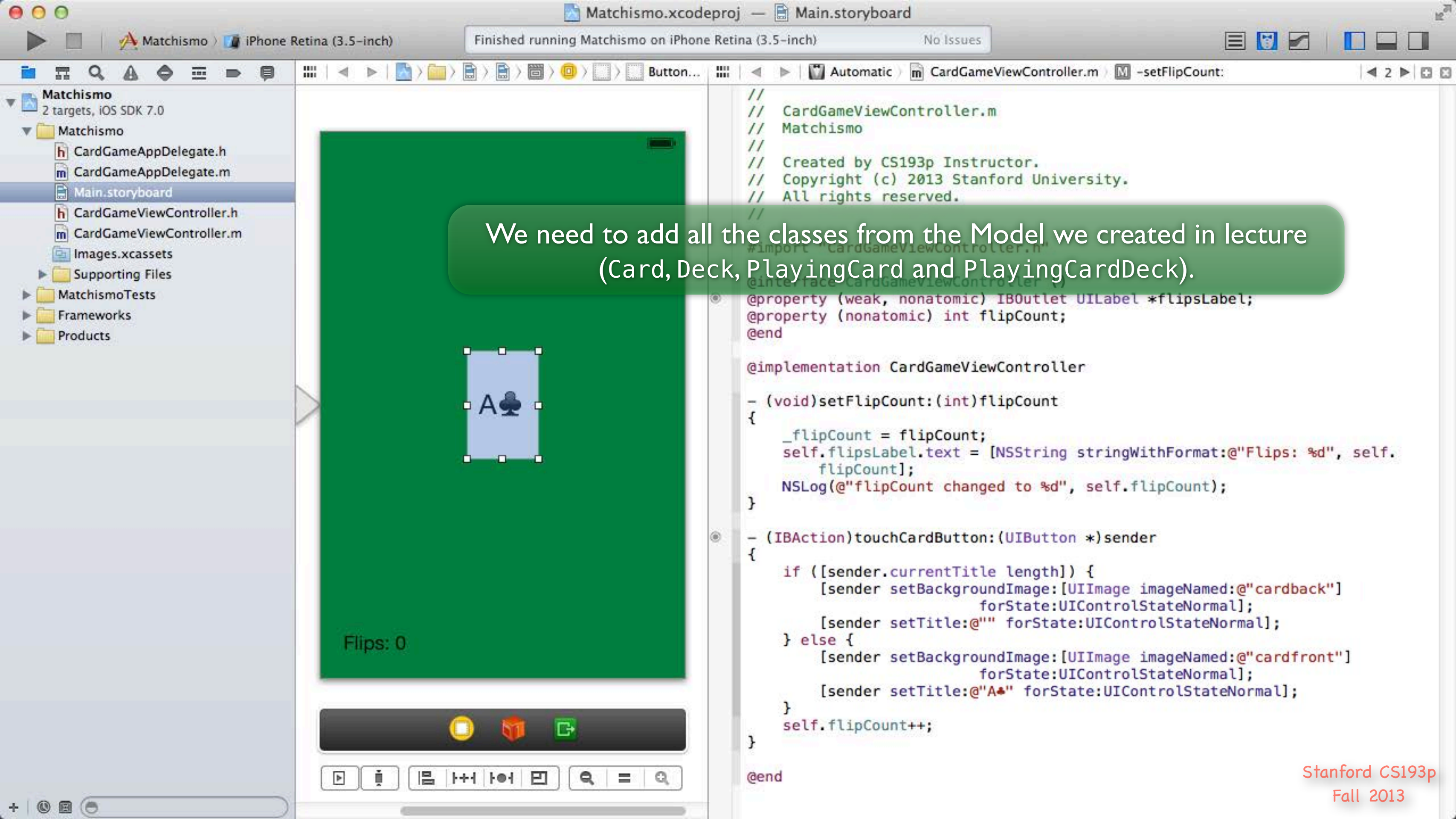
@implementation CardGameViewController

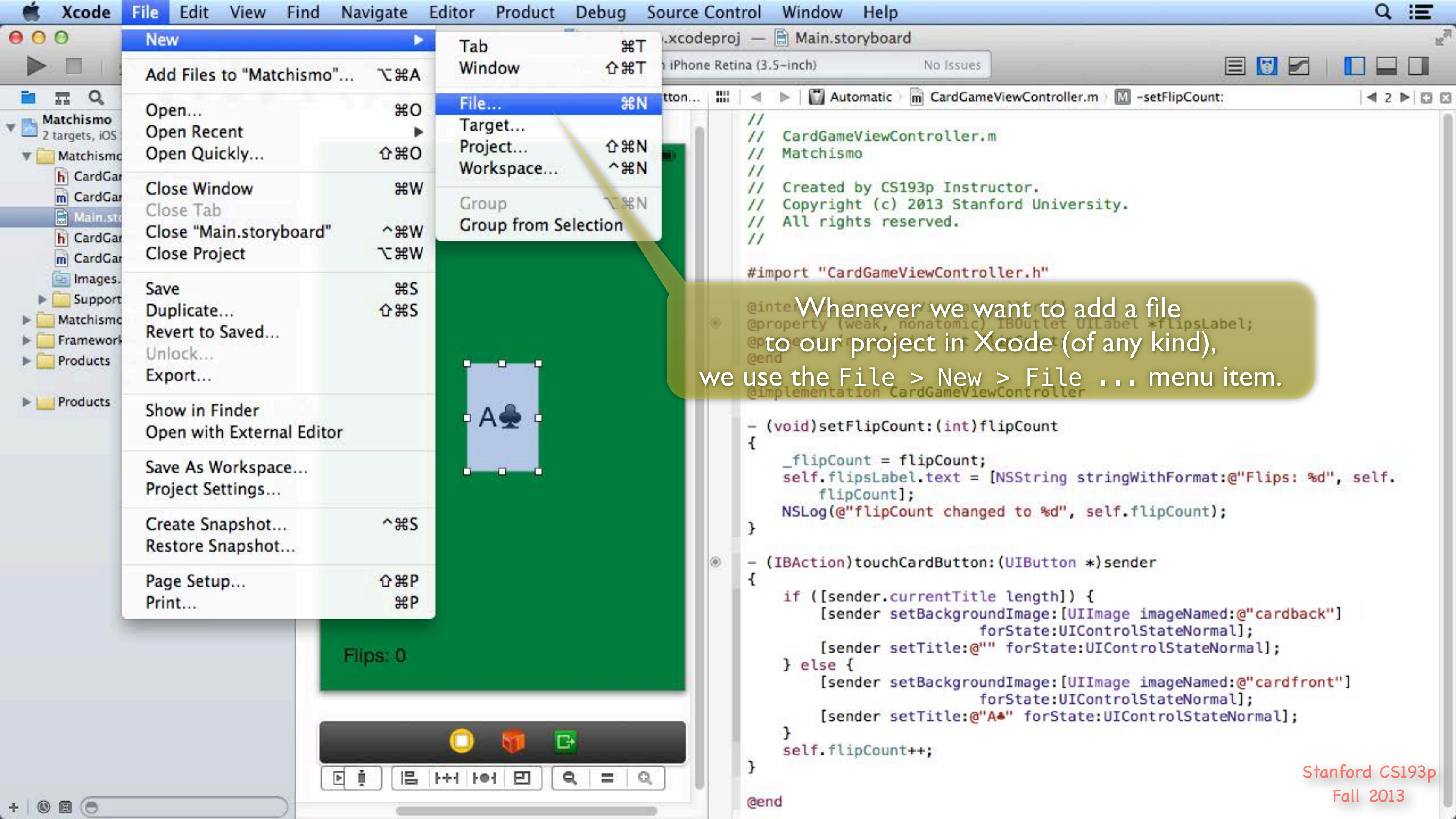
- (void)setFlipCount:(int)flipCount
{
    _flipCount = flipCount;
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.flipCount];
    NSLog(@"flipCount changed to %d", self.flipCount);
}

- (IBAction)touchCardButton:(UIButton *)sender
{
    if ([sender.currentTitle length]) {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardback"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"" forState:UIControlStateNormal];
    } else {
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
                               forState:UIControlStateNormal];
        [sender setTitle:@"A♣" forState:UIControlStateNormal];
    }
    self.flipCount++;
}

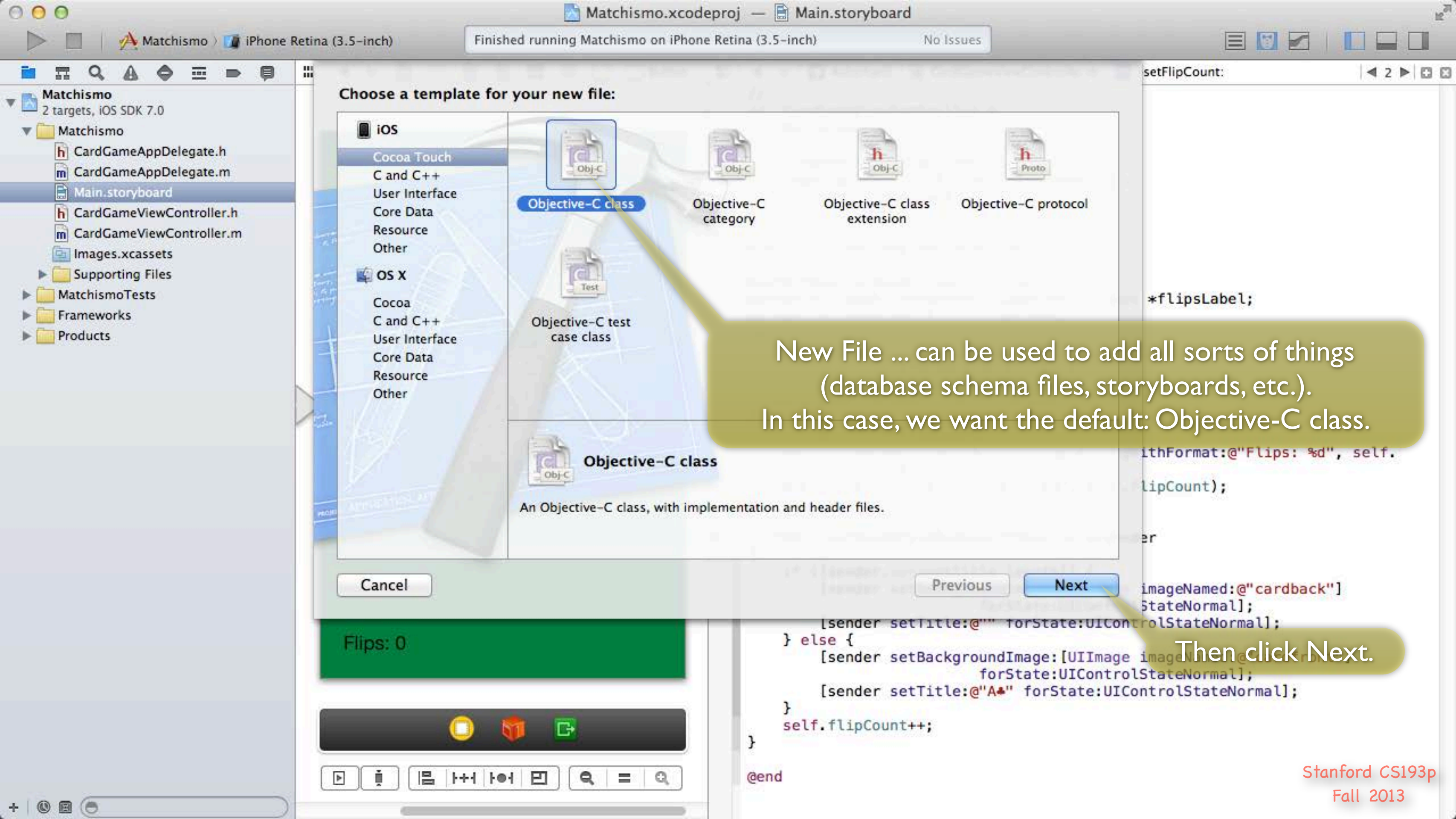
@end

```





Whenever we want to add a file to our project in Xcode (of any kind), we use the File > New > File ... menu item.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - CardGameAppDelegate.h
    - CardGameAppDelegate.m
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products

### Choose a template for your new file:

**iOS**

- Cocoa Touch
  - C and C++
  - User Interface
  - Core Data
  - Resource
  - Other
- OS X
  - Cocoa
  - C and C++
  - User Interface
  - Core Data
  - Resource
  - Other

**Objective-C class** (Selected)

Objective-C category

Objective-C class extension

Objective-C protocol

Objective-C test case class

**Objective-C class**

An Objective-C class, with implementation and header files.

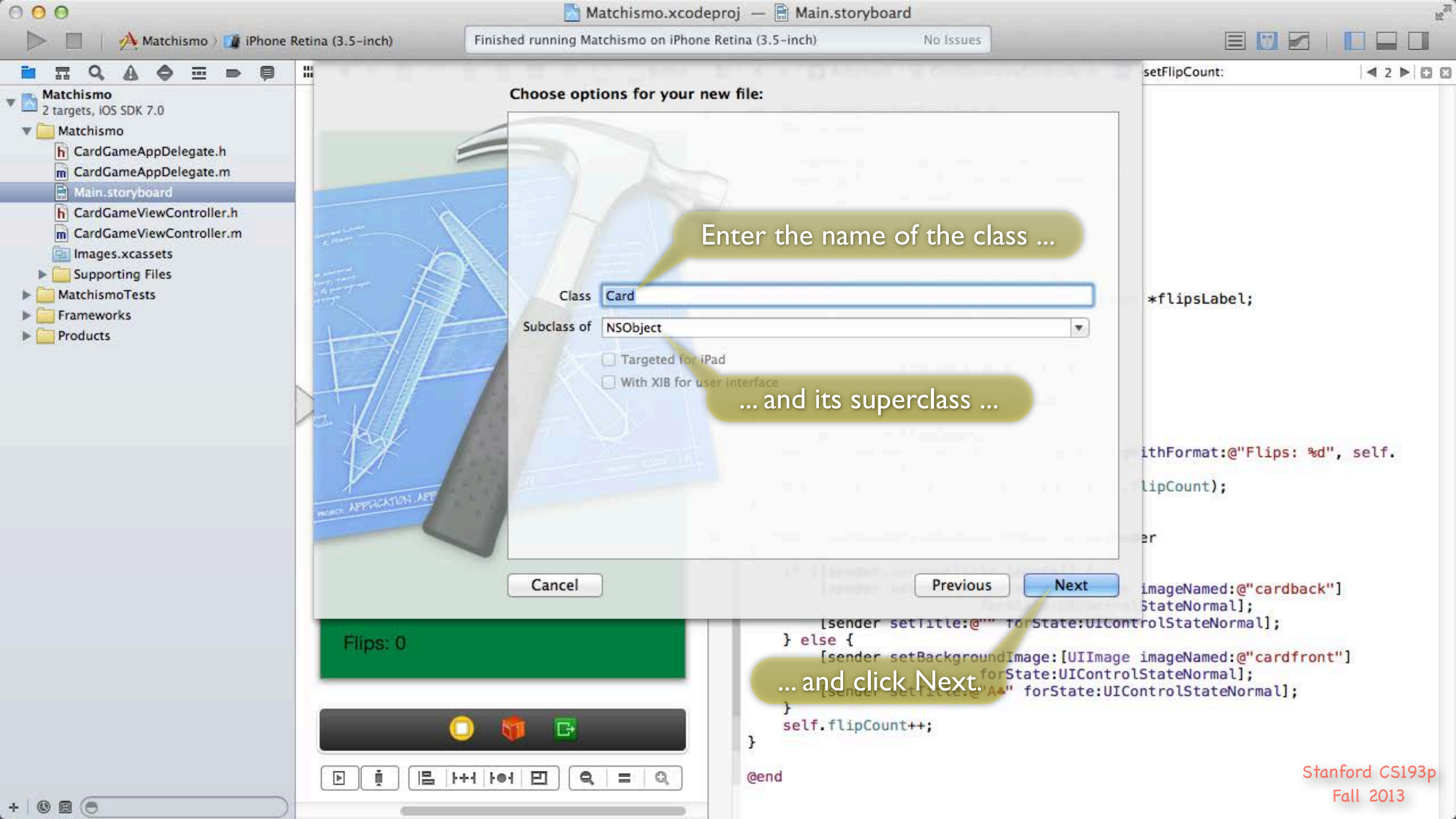
New File ... can be used to add all sorts of things (database schema files, storyboards, etc.). In this case, we want the default: Objective-C class.

```
setFlipCount:
- (void)flipCount {
    *flipsLabel;
    NSString *formatString = @"Flips: %d", self.flipCount);
    [self.flipCountLabel setTitle:formatString forState:UIControlStateNormal];
    [self.flipCountLabel setBackgroundImage:[UIImage imageNamed:@"cardback"] forState:UIControlStateNormal];
    [self.flipCountLabel setTitle:@"A+" forState:UIControlStateNormal];
    self.flipCount++;
}
@end
```

Then click Next.

Flips: 0





Choose options for your new file:

Class

Subclass of

- Targeted for iPad
- With XIB for user interface

Enter the name of the class ...

... and its superclass ...

... and click Next.

setFlipCount: |< 2 > + x

```
*flipsLabel;

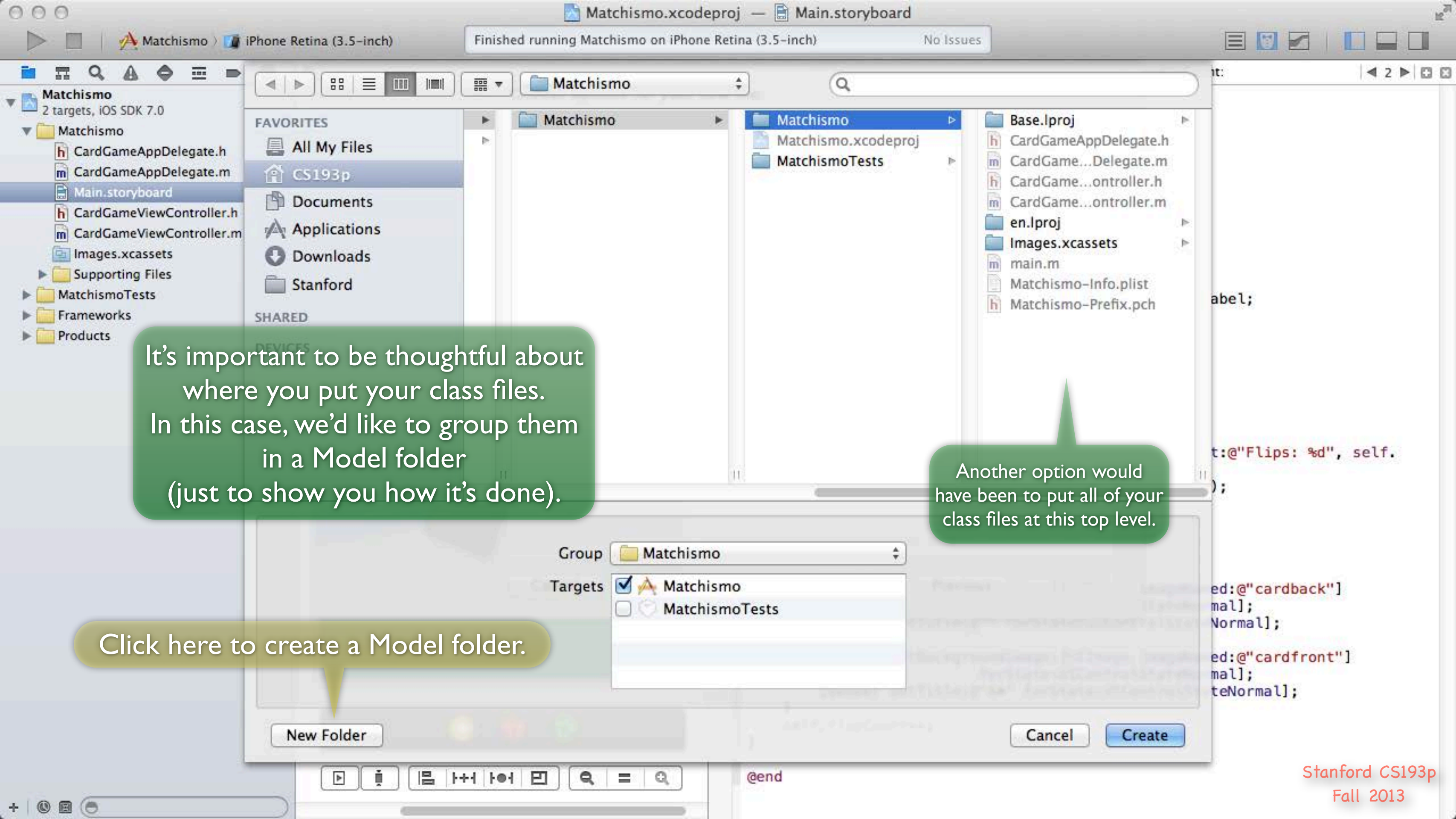
withFormat:@"Flips: %d", self.
lipCount);

er

imageName:@"cardback"]
StateNormal];

[sender setTitle:@"" forState:UIControlStateNormal];
} else {
[sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]
forState:UIControlStateNormal];
[sender setTitle:@"A" forState:UIControlStateNormal];
}
self.flipCount++;
}

@end
```



It's important to be thoughtful about where you put your class files. In this case, we'd like to group them in a Model folder (just to show you how it's done).

Another option would have been to put all of your class files at this top level.

Click here to create a Model folder.

```
label;

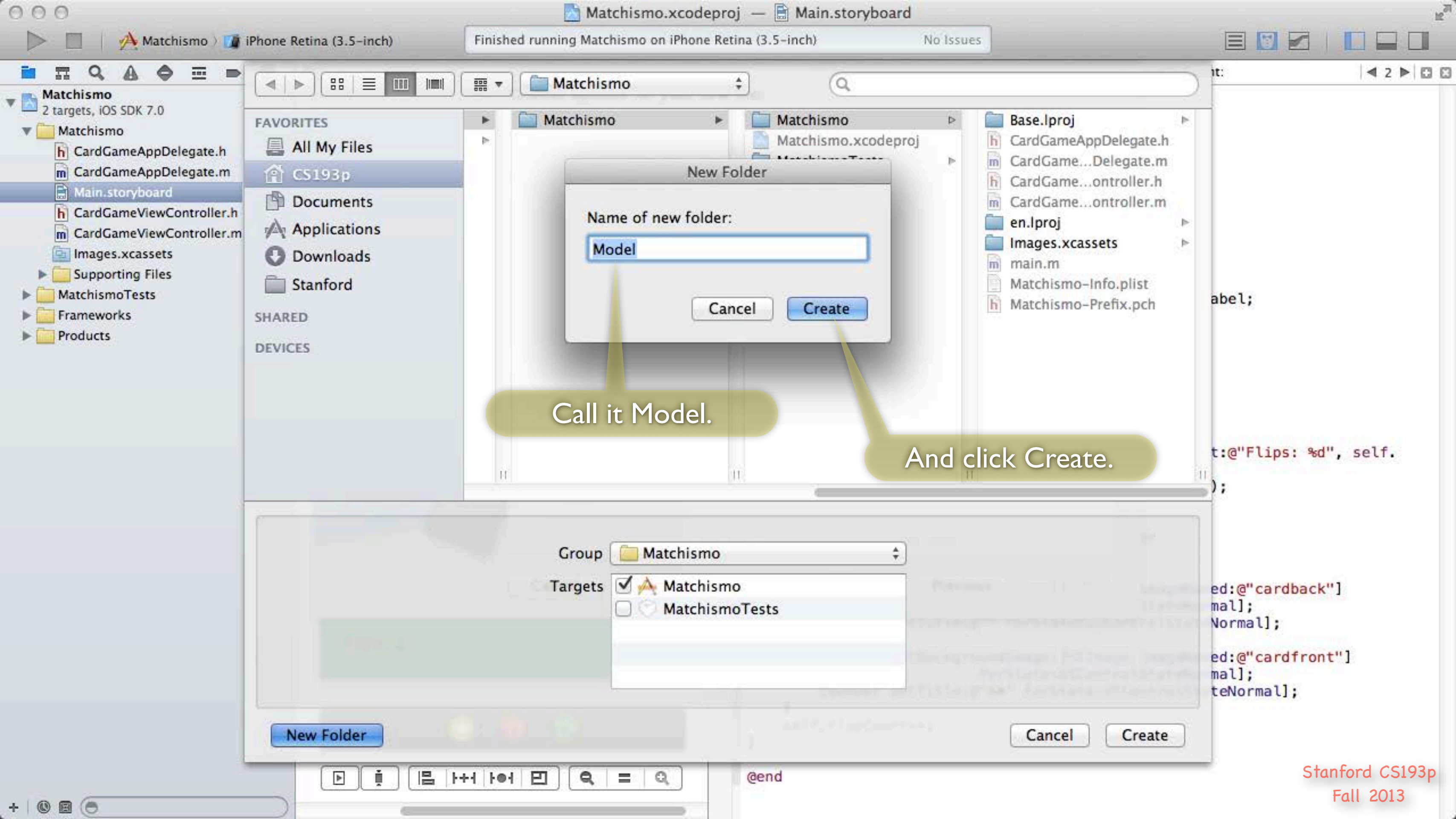
t:@"Flips: %d", self.

);

ed:@"cardback"]
mal];
Normal];

ed:@"cardfront"]
mal];
teNormal];

@end
```



Call it Model.

And click Create.

Group Matchismo

Targets  Matchismo  
 MatchismoTests

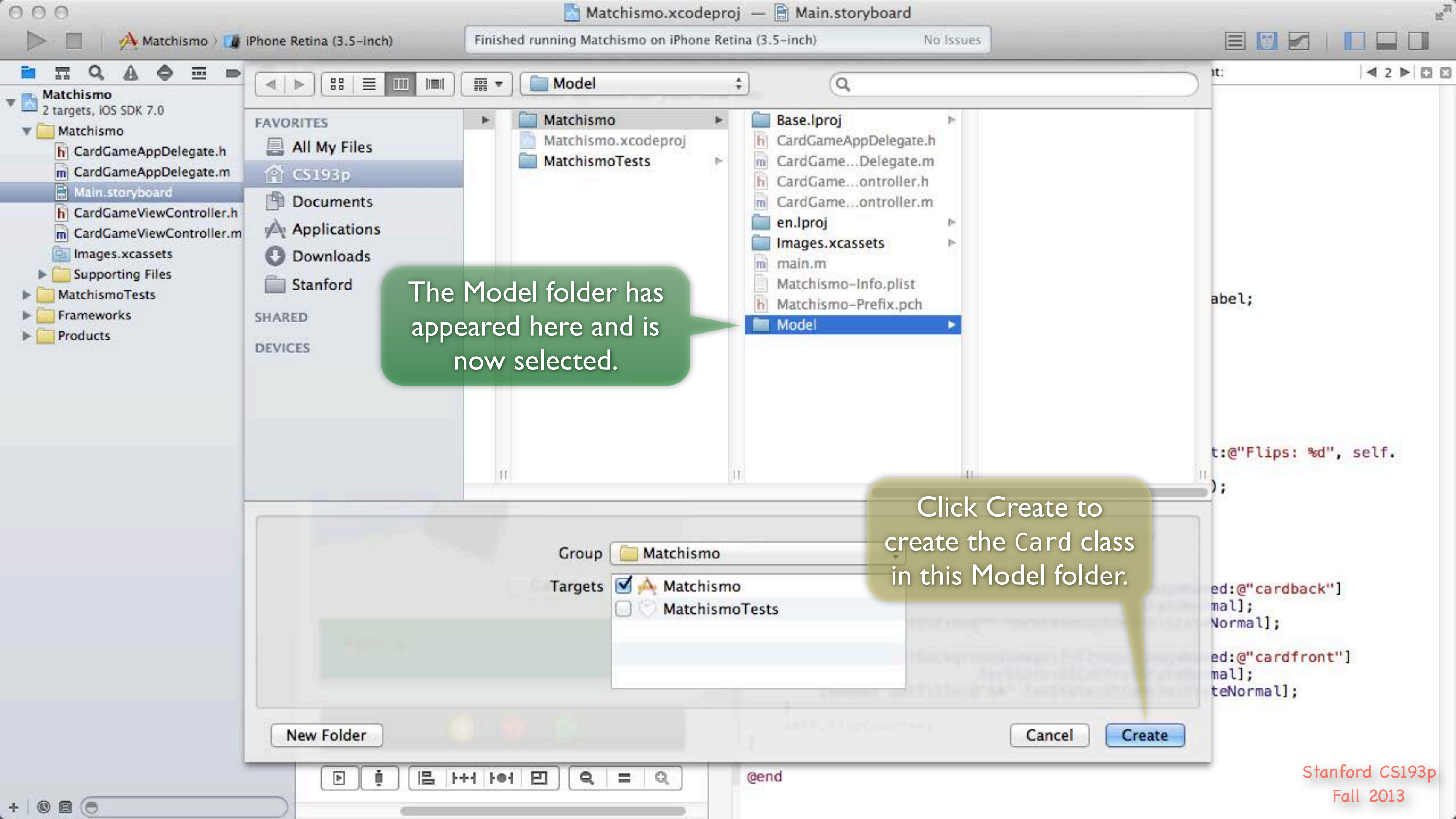
New Folder

Cancel

Create

```
label;  
  
t:@"Flips: %d", self.  
);  
  
ed:@"cardback"]  
mal];  
Normal];  
  
ed:@"cardfront"]  
mal];  
teNormal];
```

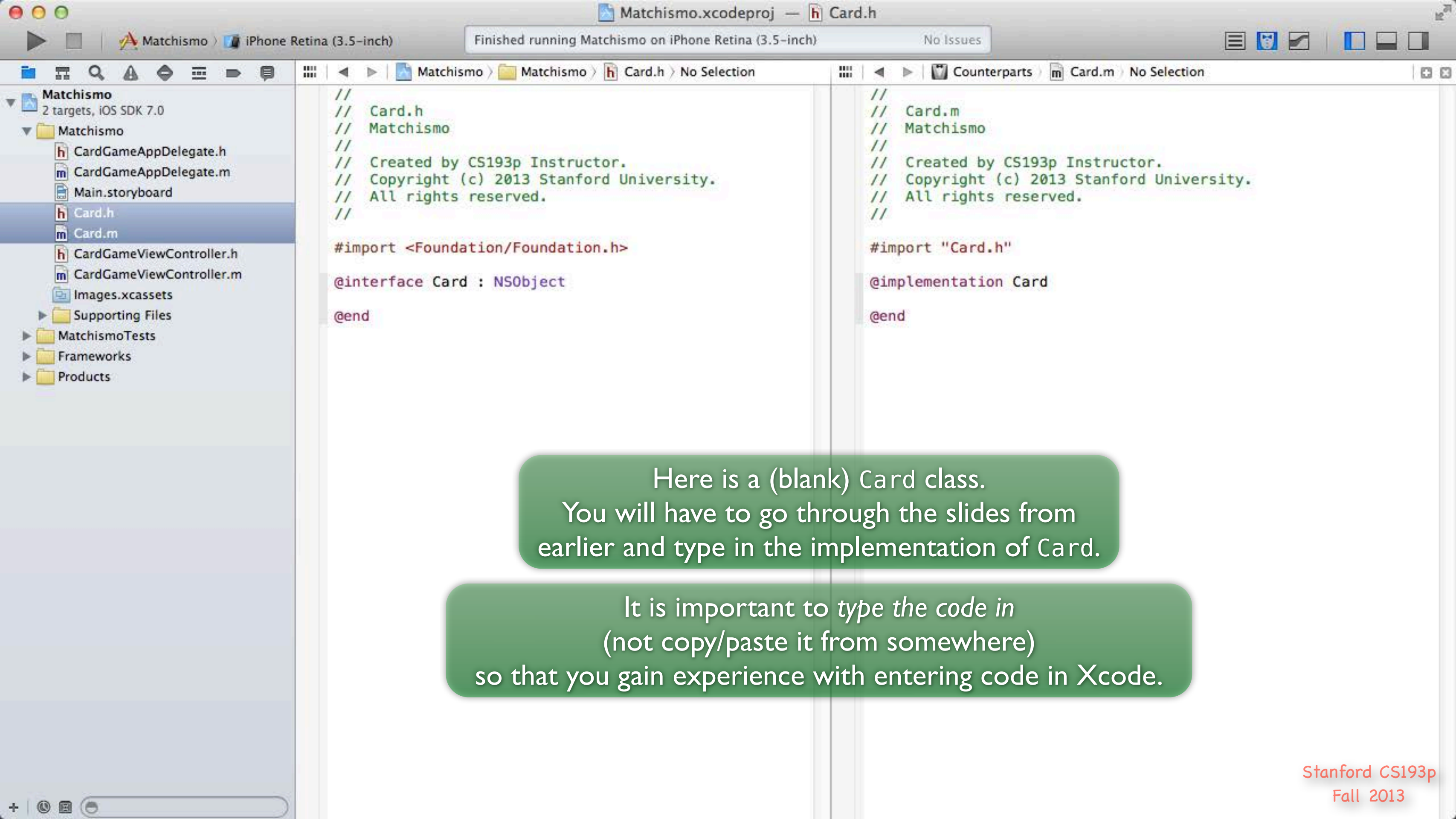




The Model folder has appeared here and is now selected.

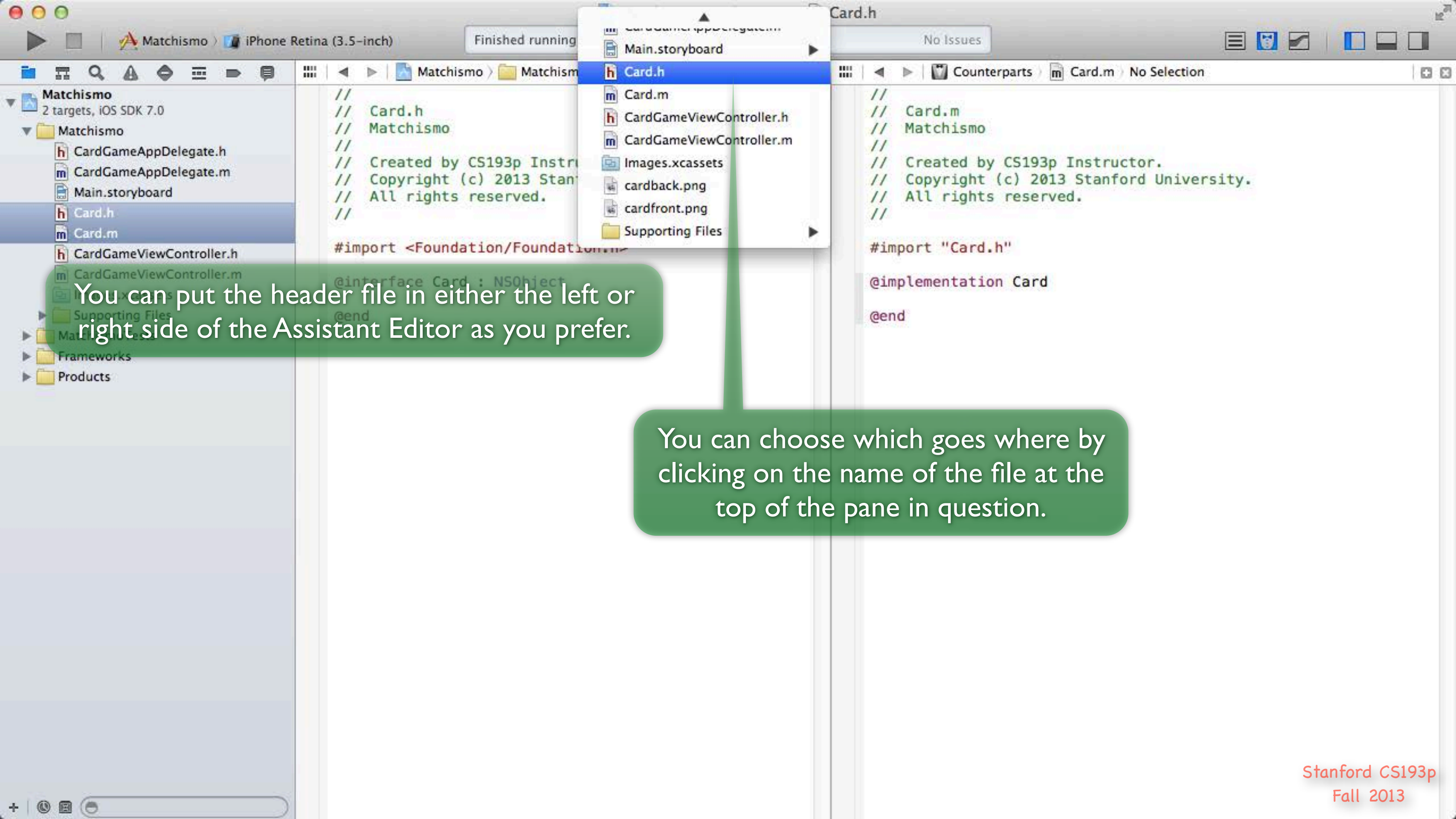
Click Create to create the Card class in this Model folder.

```
label;  
  
t:@"Flips: %d", self.  
);  
  
ed:@"cardback"]  
mal];  
Normal];  
  
ed:@"cardfront"]  
mal];  
teNormal];
```



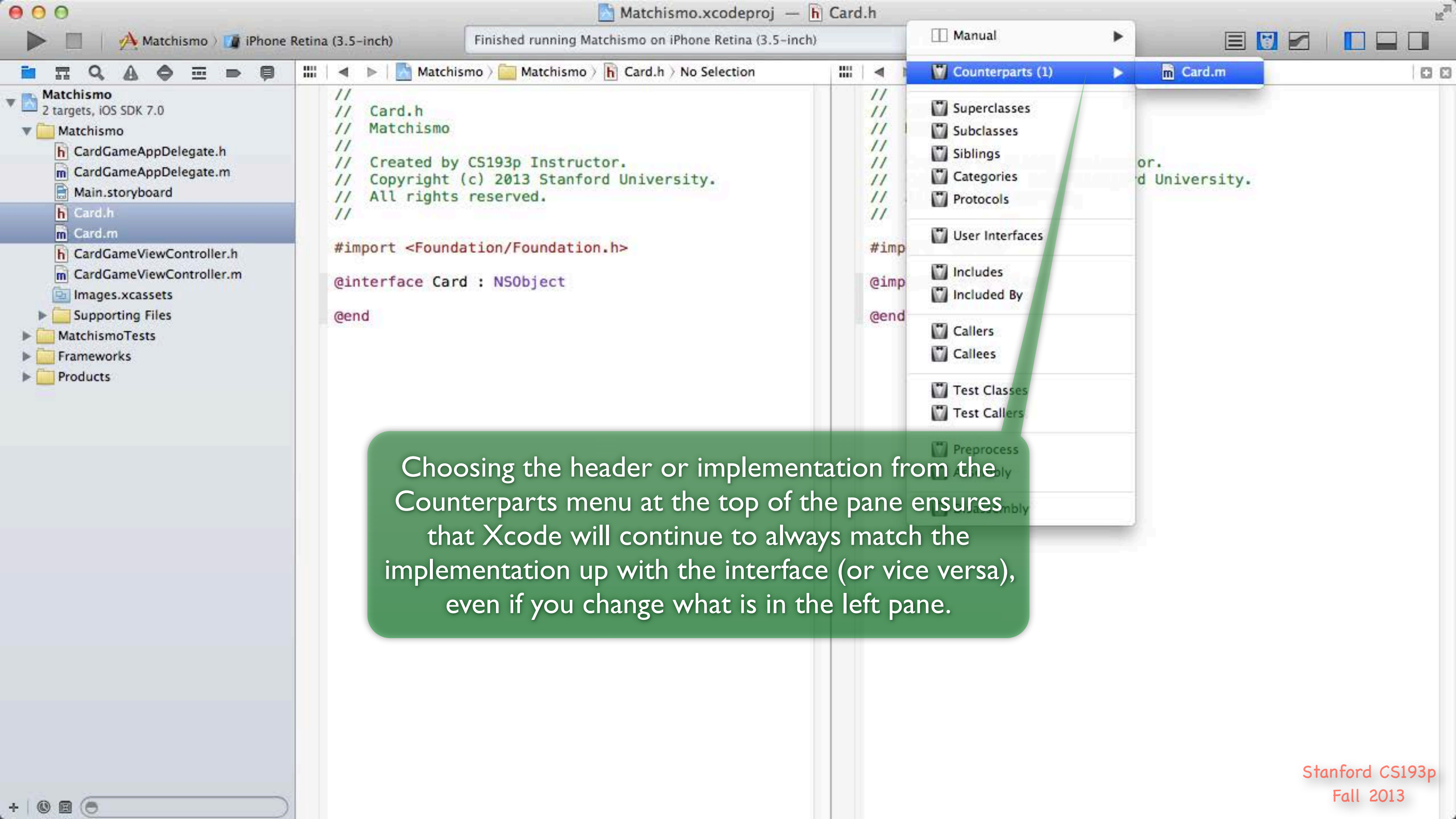
Here is a (blank) Card class.  
You will have to go through the slides from earlier and type in the implementation of Card.

It is important to *type the code in* (not copy/paste it from somewhere) so that you gain experience with entering code in Xcode.



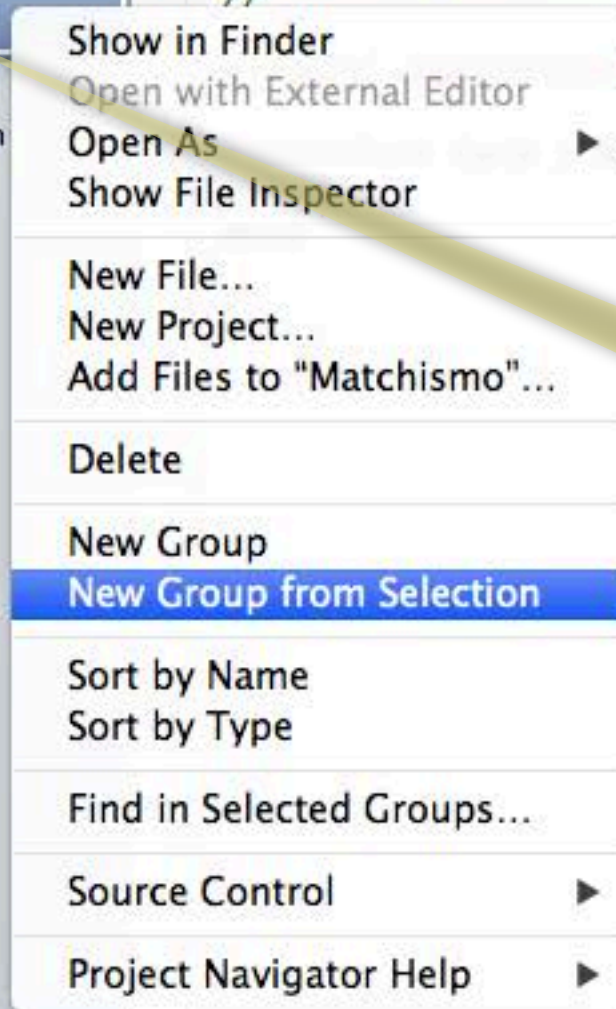
You can put the header file in either the left or right side of the Assistant Editor as you prefer.

You can choose which goes where by clicking on the name of the file at the top of the pane in question.

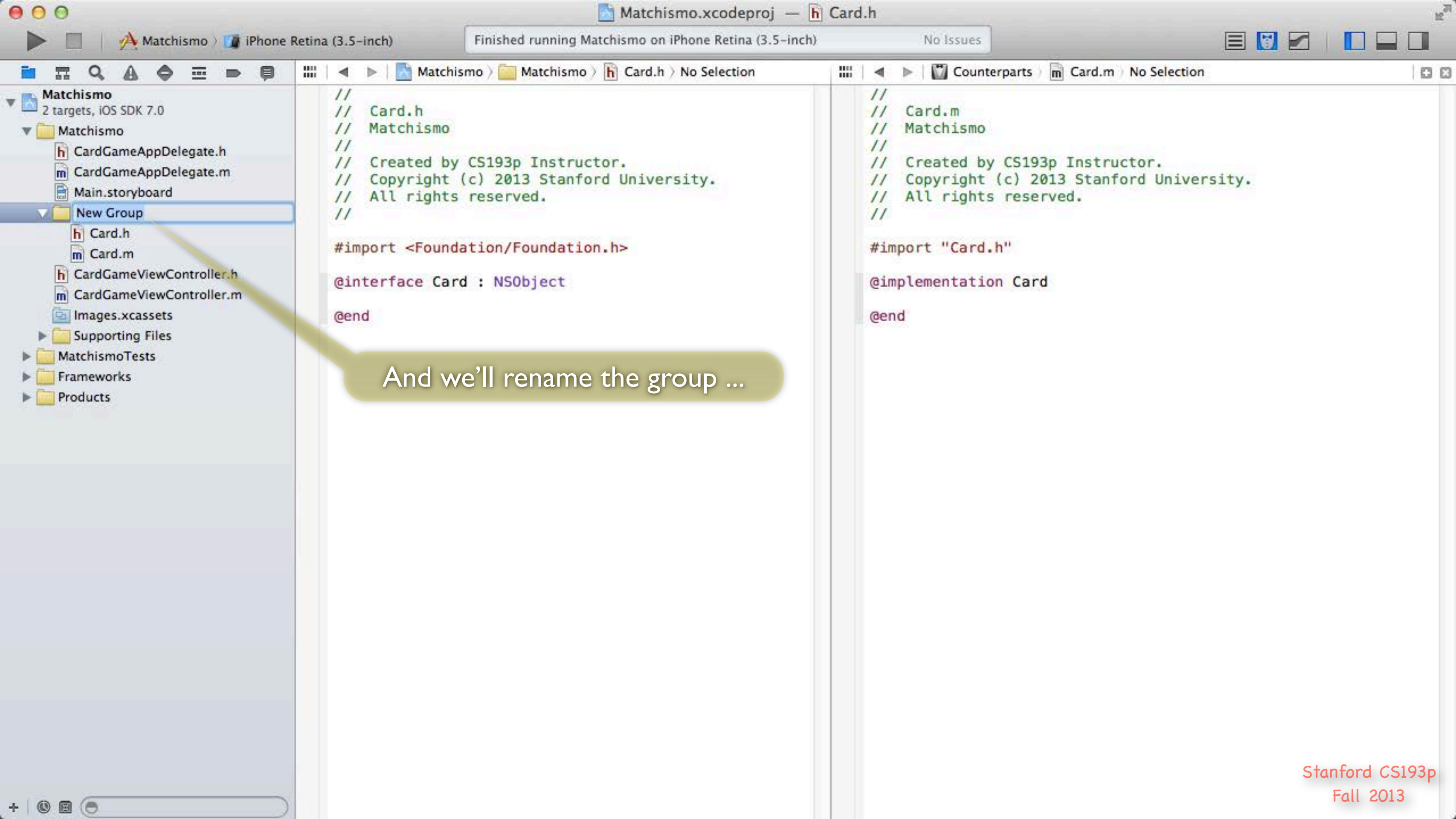


Choosing the header or implementation from the Counterparts menu at the top of the pane ensures that Xcode will continue to always match the implementation up with the interface (or vice versa), even if you change what is in the left pane.

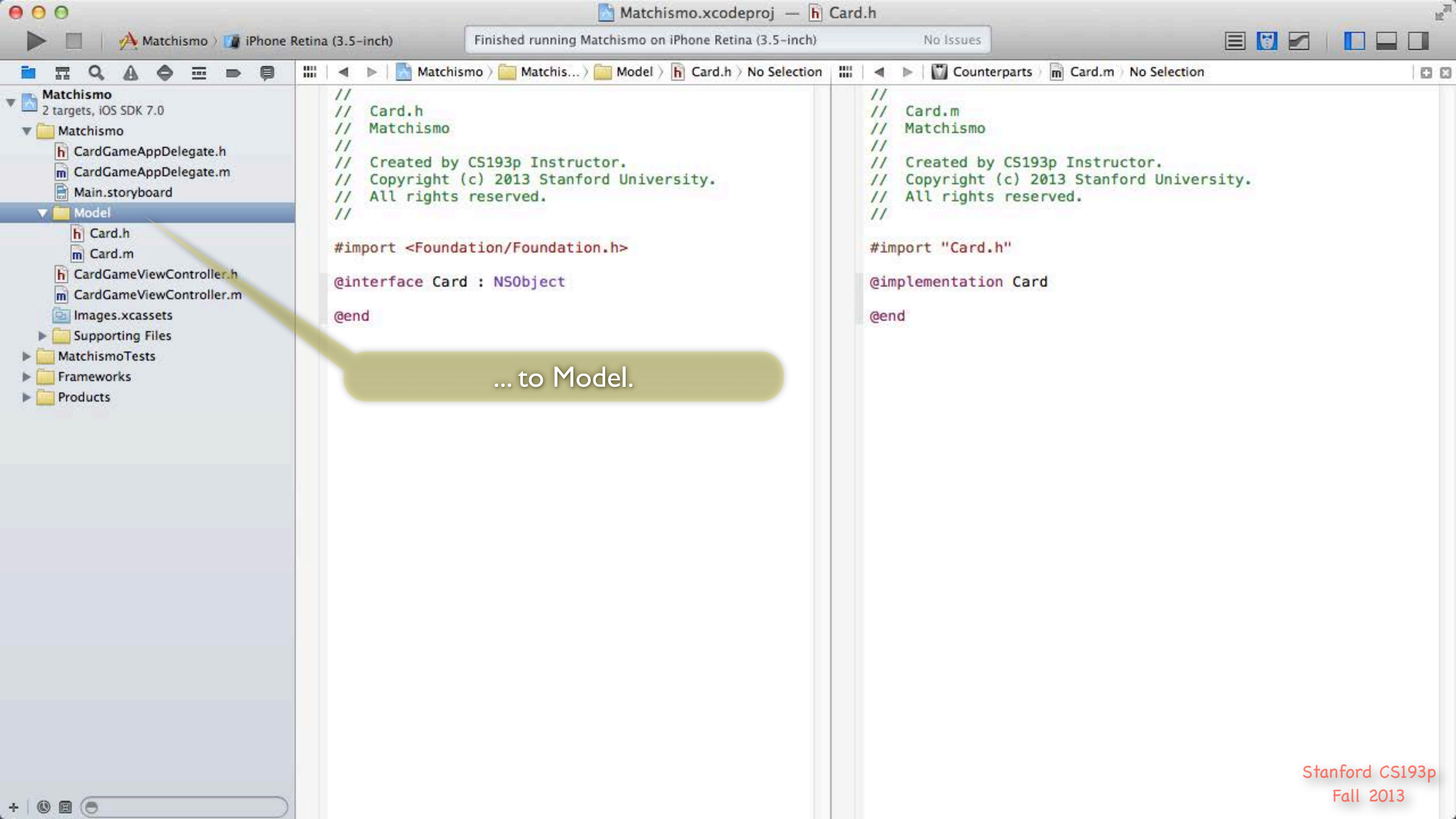
Let's create a Navigator group for all of our Model classes.



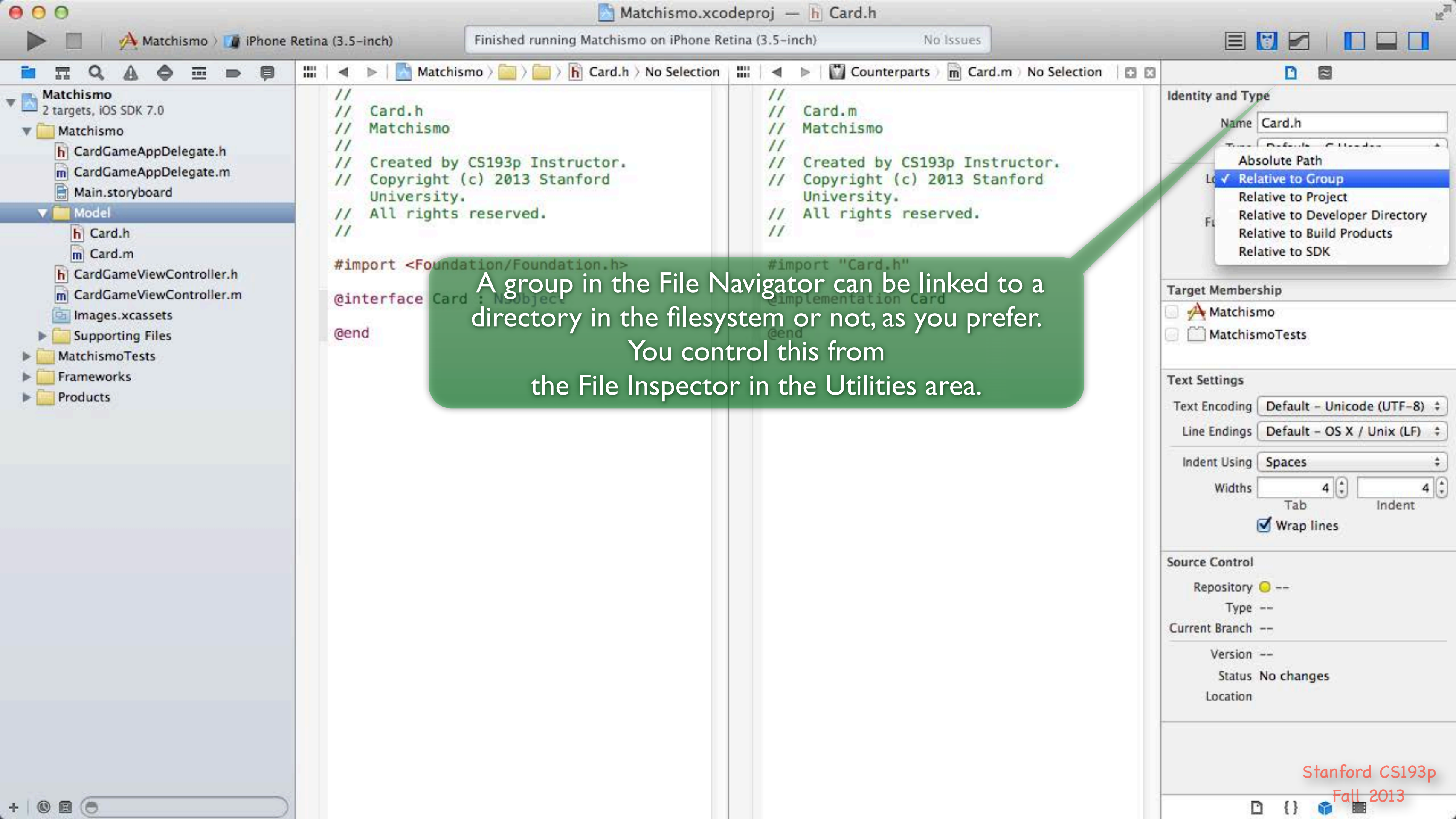
Select both Card.h and Card.m and then right-click on them to get this menu, then choose New Group from Selection.



And we'll rename the group ...

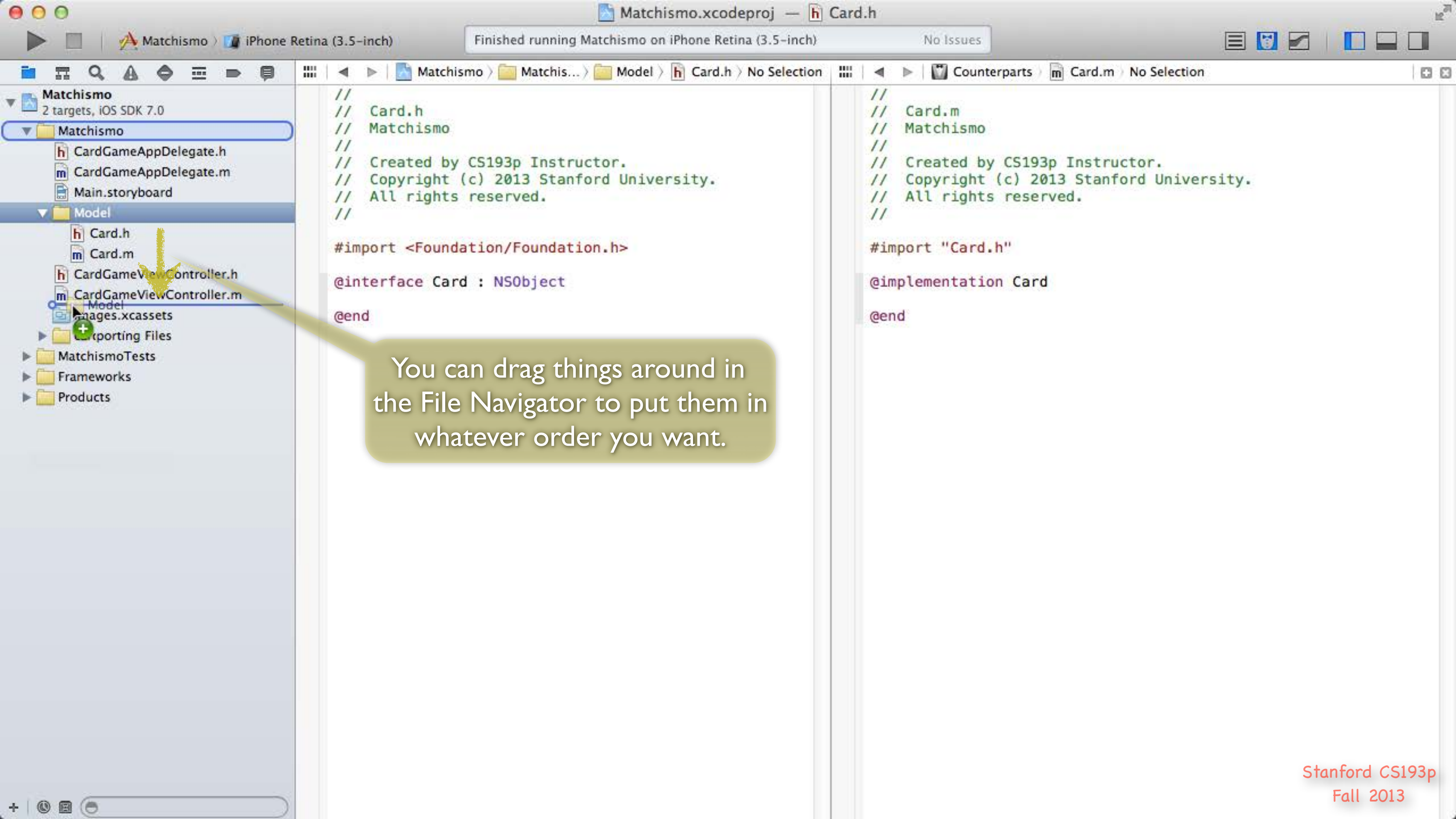


... to Model.

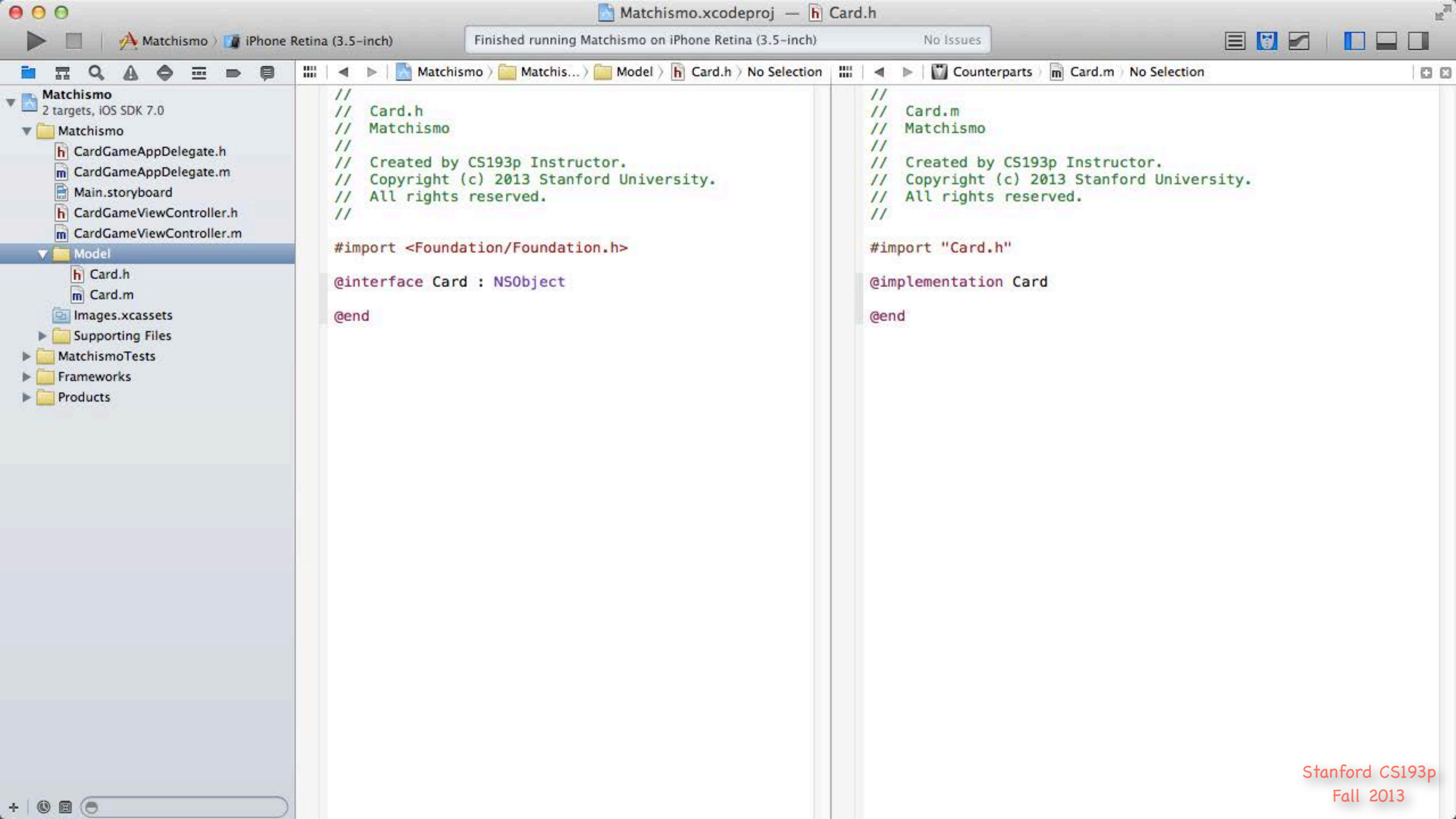


A group in the File Navigator can be linked to a directory in the filesystem or not, as you prefer. You control this from the File Inspector in the Utilities area.



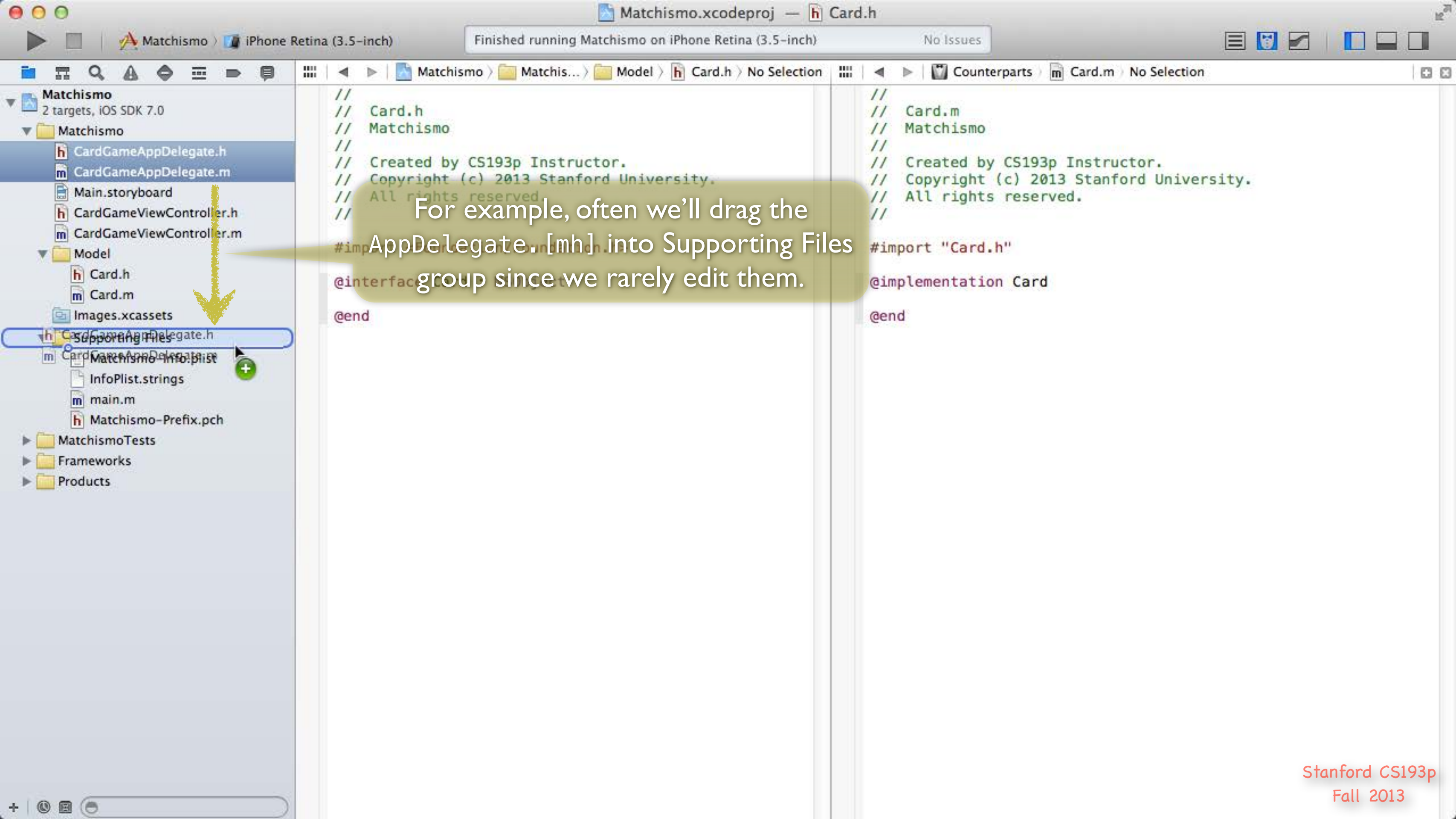


You can drag things around in the File Navigator to put them in whatever order you want.

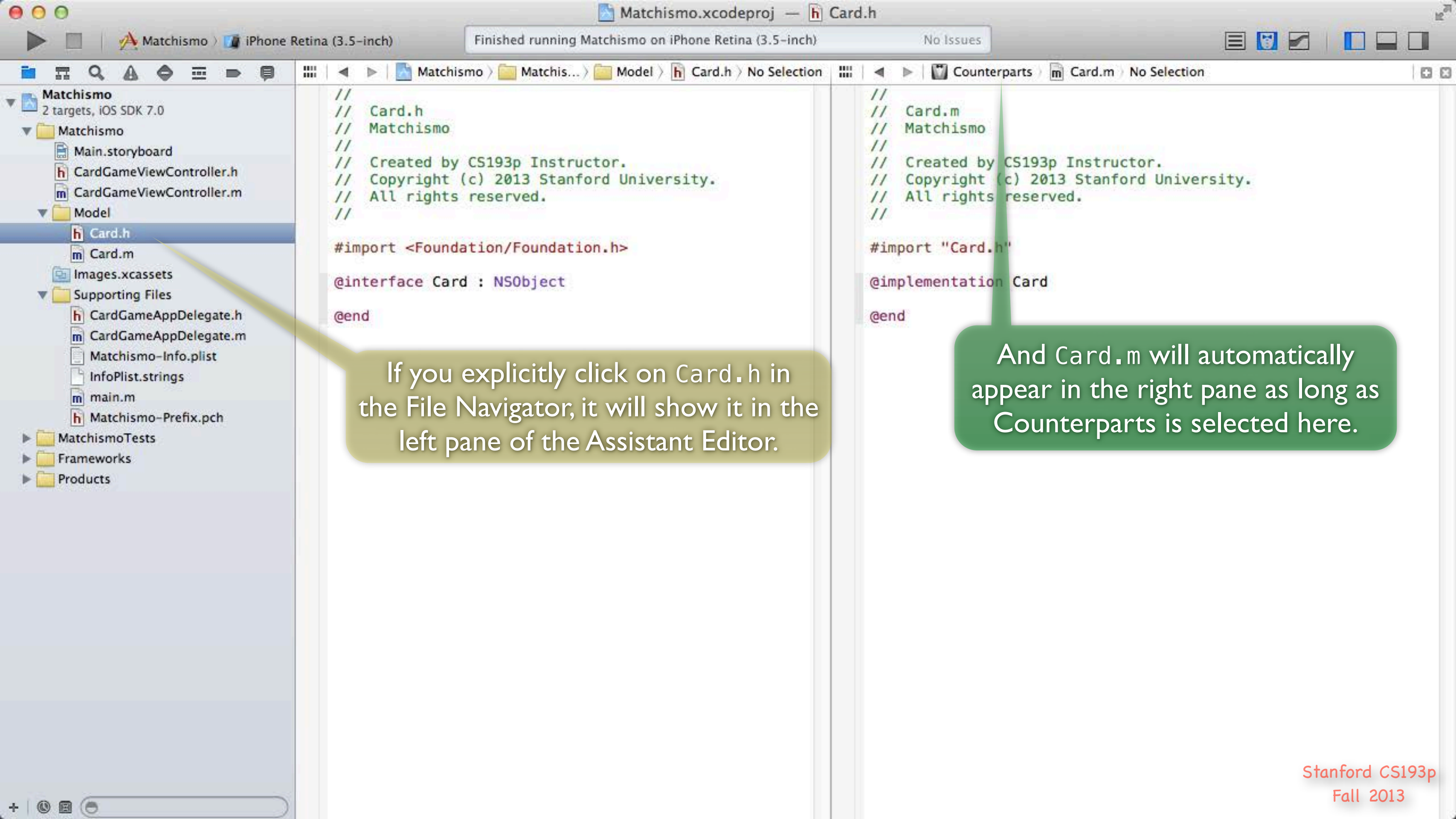


```
//  
// Card.h  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import <Foundation/Foundation.h>  
  
@interface Card : NSObject  
  
@end
```

```
//  
// Card.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "Card.h"  
  
@implementation Card  
  
@end
```

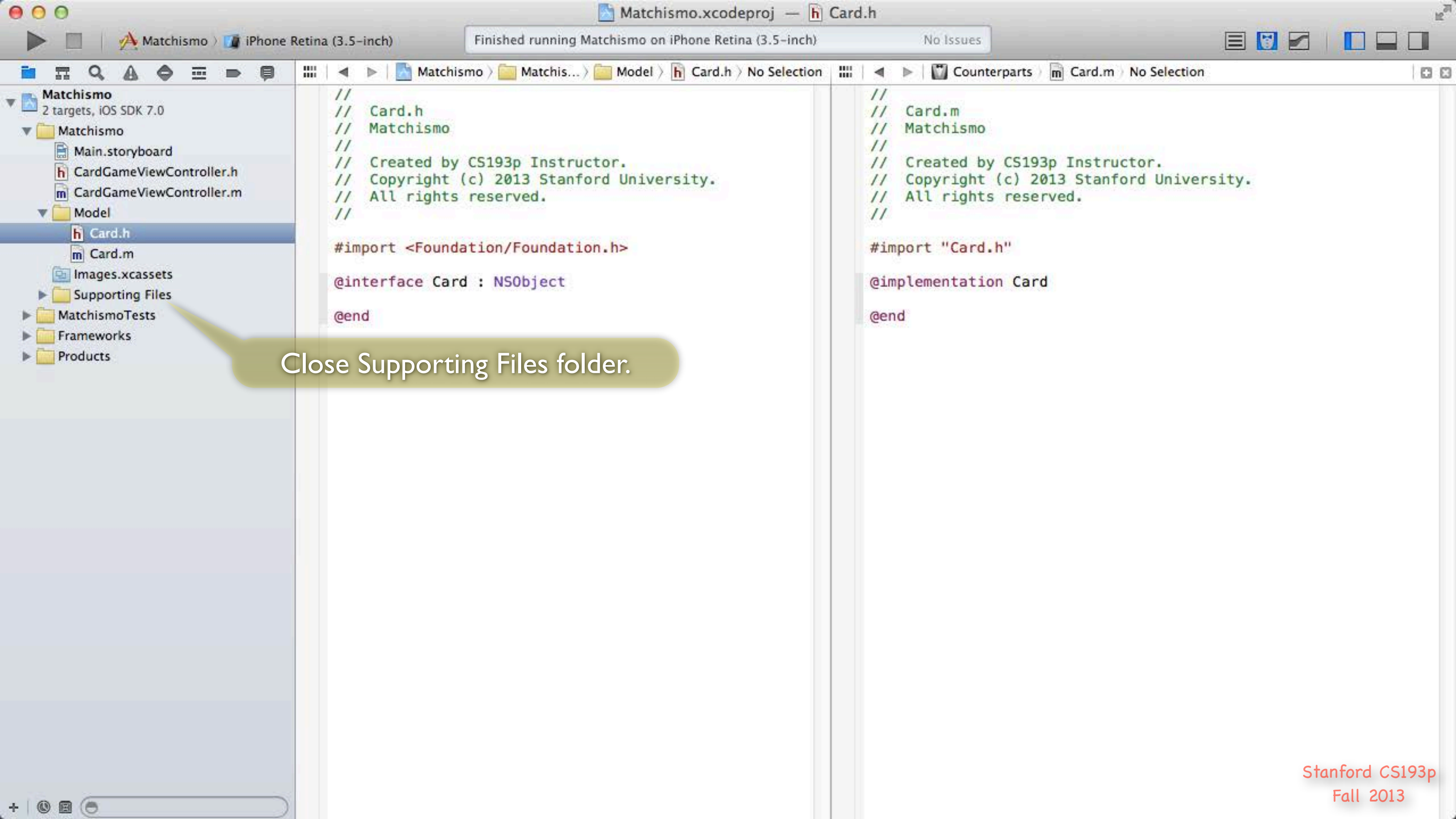


For example, often we'll drag the AppDelegate.[mh] into Supporting Files group since we rarely edit them.



If you explicitly click on Card.h in the File Navigator, it will show it in the left pane of the Assistant Editor.

And Card.m will automatically appear in the right pane as long as Counterparts is selected here.



Close Supporting Files folder.

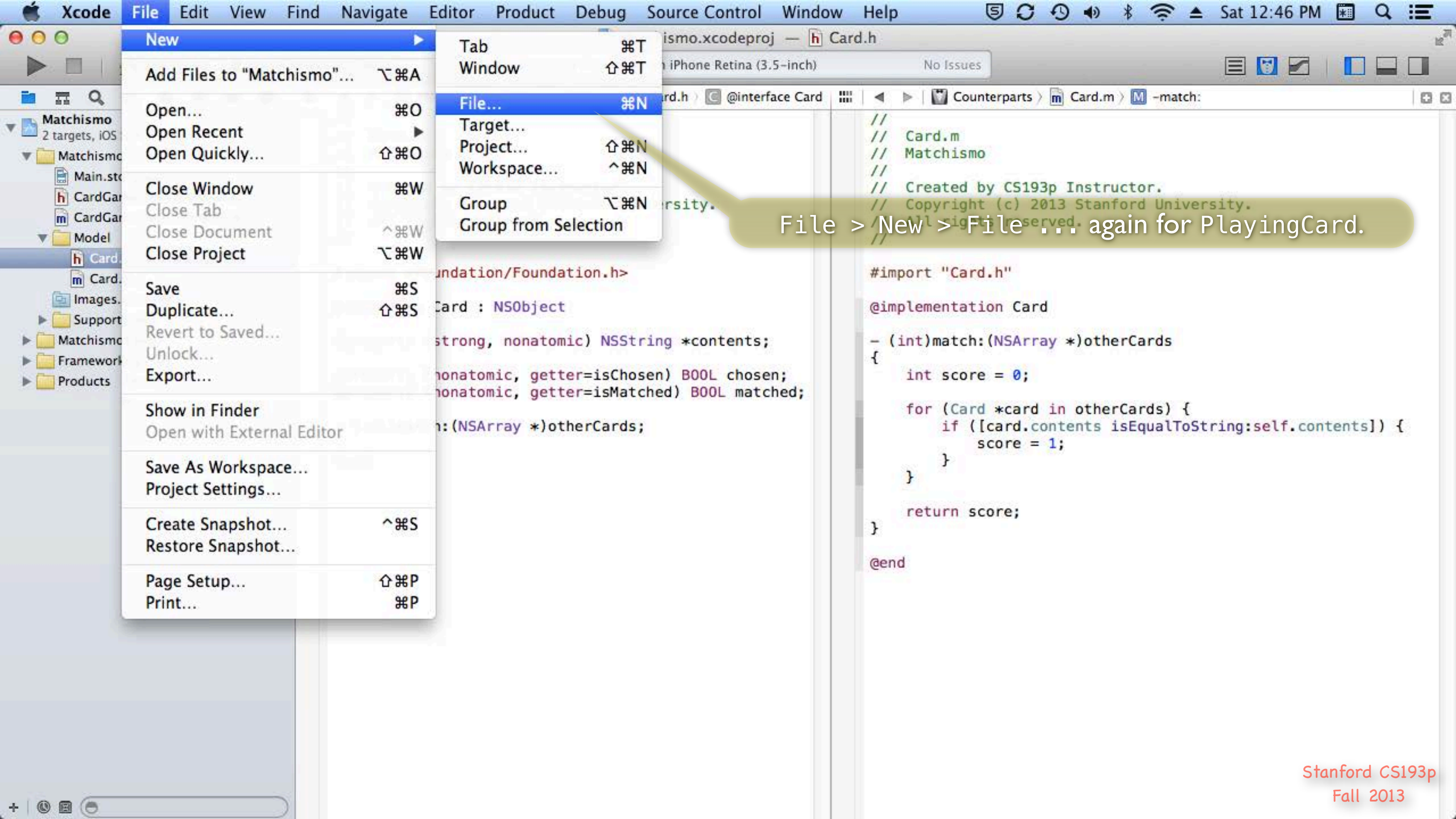


```
//  
// Card.h  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import <Foundation/Foundation.h>  
  
@interface Card : NSObject  
  
@property (strong, nonatomic) NSString *contents;  
  
@property (nonatomic, getter=isChosen) BOOL chosen;  
@property (nonatomic, getter=isMatched) BOOL matched;  
  
- (int)match:(NSArray *)otherCards;  
  
@end
```

```
//  
// Card.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "Card.h"  
  
@implementation Card  
  
- (int)match:(NSArray *)otherCards  
{  
    int score = 0;  
  
    for (Card *card in otherCards) {  
        if ([card.contents isEqualToString:self.contents]) {  
            score = 1;  
        }  
    }  
  
    return score;  
}  
  
@end
```

Type in the code for Card. [mh].

Now let's move on to creating templates for the Deck, PlayingCard and PlayingCardDeck classes.









File > New > File ... again for PlayingCard.

Matchismo  
2 targets, iOS SDK 7.0

- Matchismo
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Model
    - Card.h
    - Card.m
  - Images.xcassets
    - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

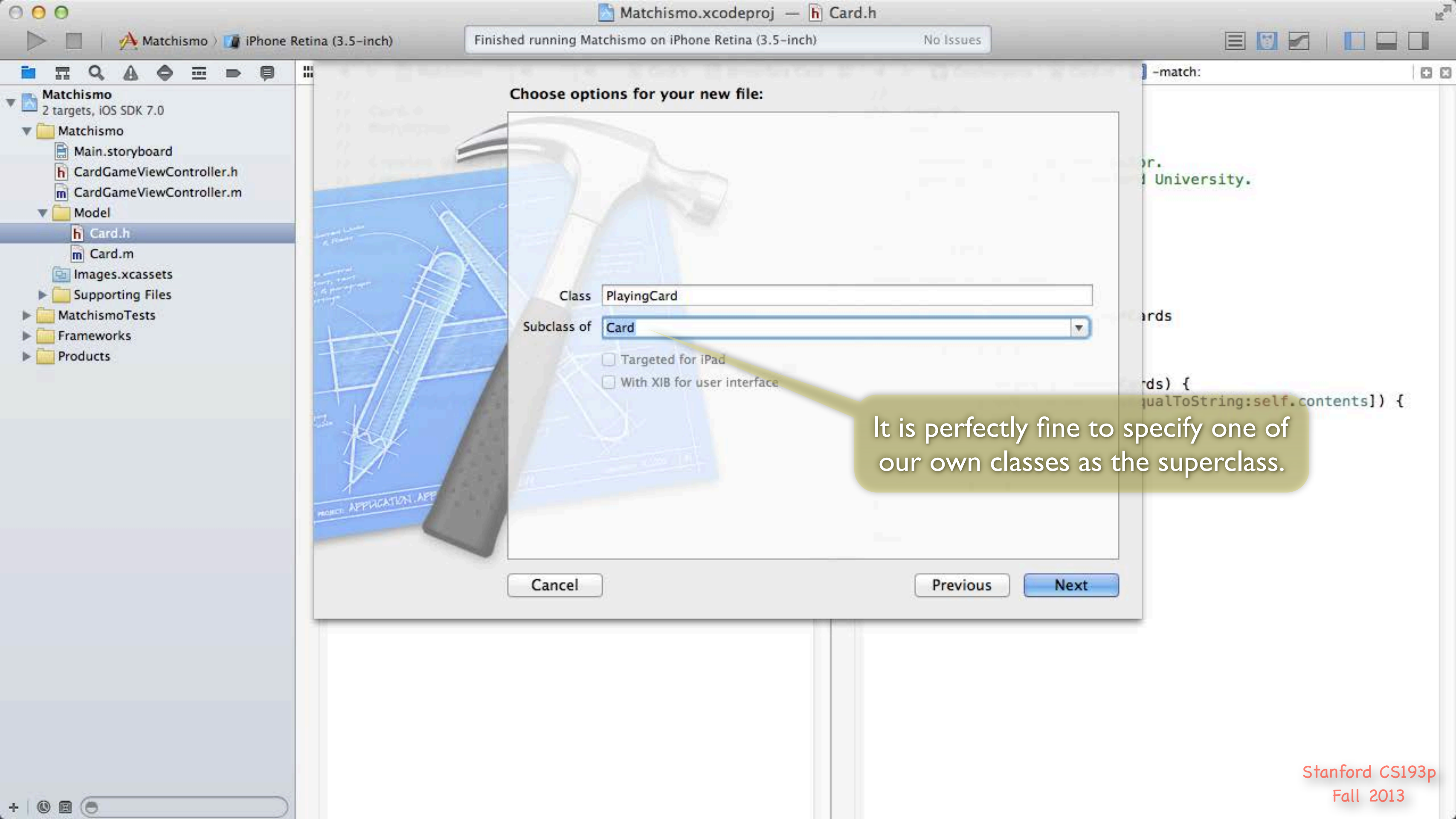
### Choose a template for your new file:

|   |  |   |  |   |
|---|--|---|--|---|
| <b>iOS</b> <ul style="list-style-type: none"><li>Cocoa Touch<ul style="list-style-type: none"><li>C and C++</li><li>User Interface</li><li>Core Data</li><li>Resource</li><li>Other</li></ul></li><li>OS X<ul style="list-style-type: none"><li>Cocoa</li><li>C and C++</li><li>User Interface</li><li>Core Data</li><li>Resource</li><li>Other</li></ul></li></ul> | <br><b>Objective-C class</b>  | <br>Objective-C category | <br>Objective-C class extension | <br>Objective-C protocol extension |
|   | <br>Objective-C test case class   |   |  |   |
|   | <br><b>Objective-C class</b><br>An Objective-C class, with implementation and header files. |   |  |   |

Cancel Previous Next

```
-match:  
  
University.  
  
Cards  
  
Cards) {  
    equalToString:self.contents]) {
```





Choose options for your new file:

Class

Subclass of

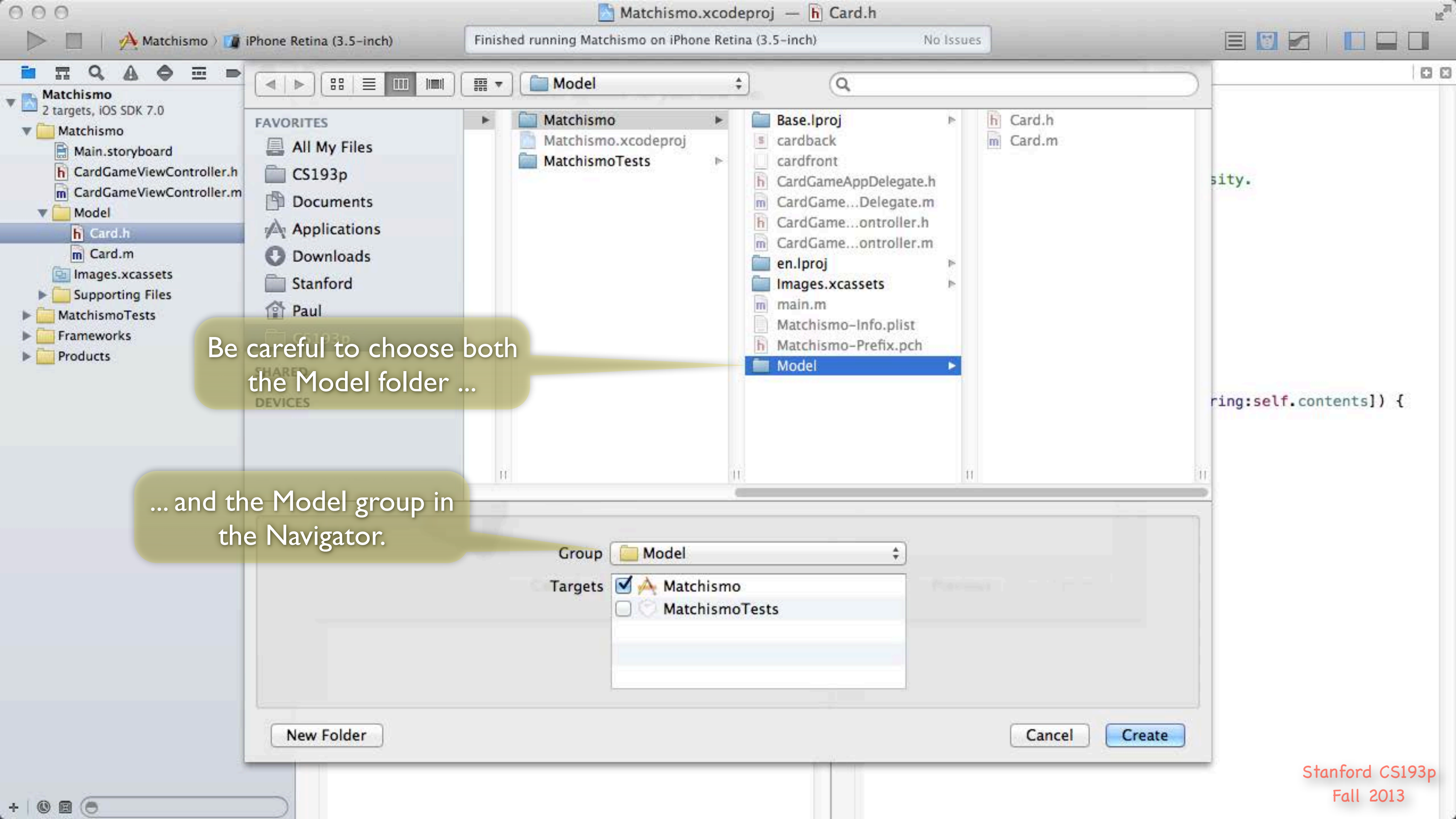
- Targeted for iPad
- With XIB for user interface

It is perfectly fine to specify one of our own classes as the superclass.

Cancel

Previous

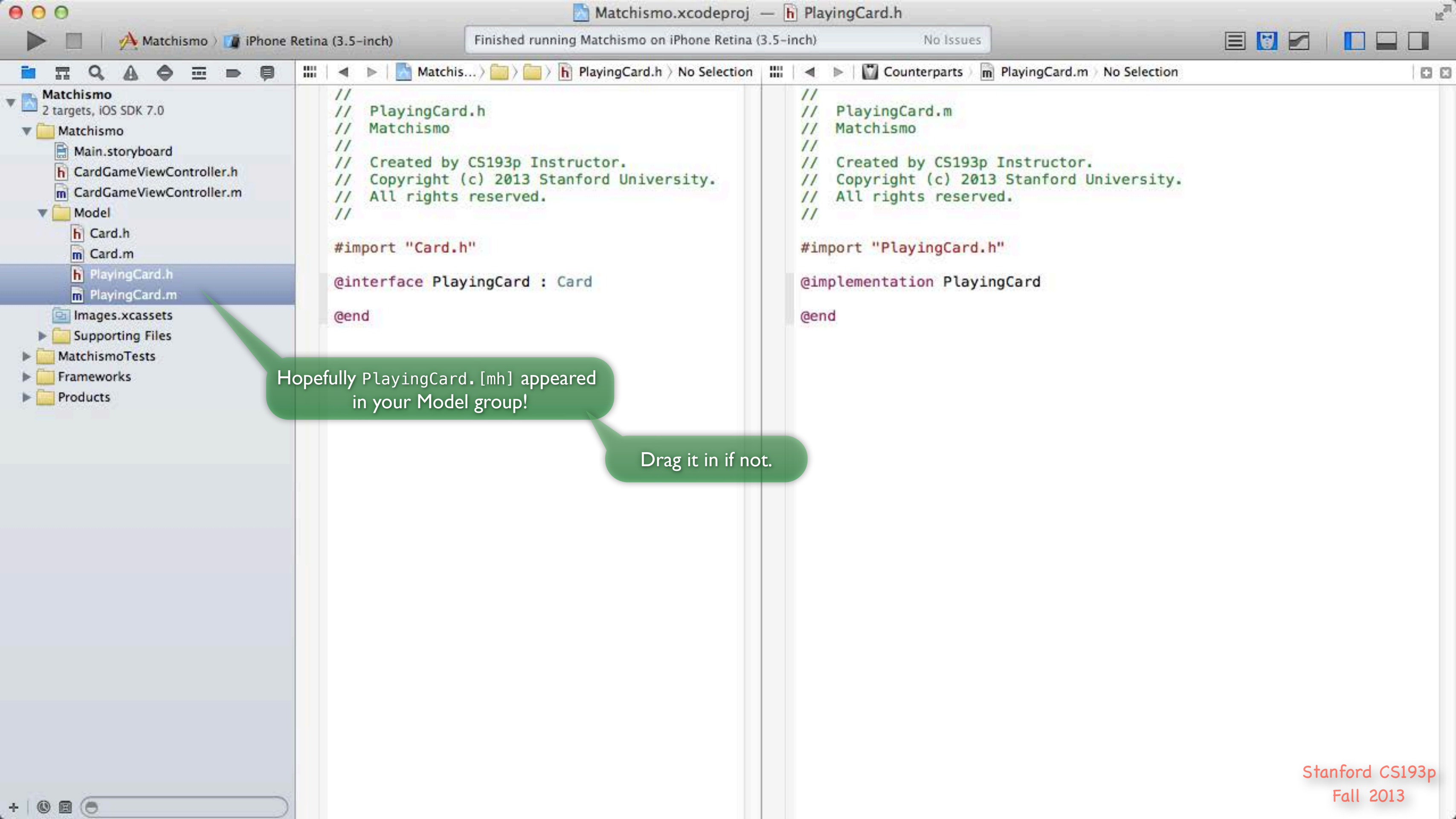
Next



Be careful to choose both the Model folder ...

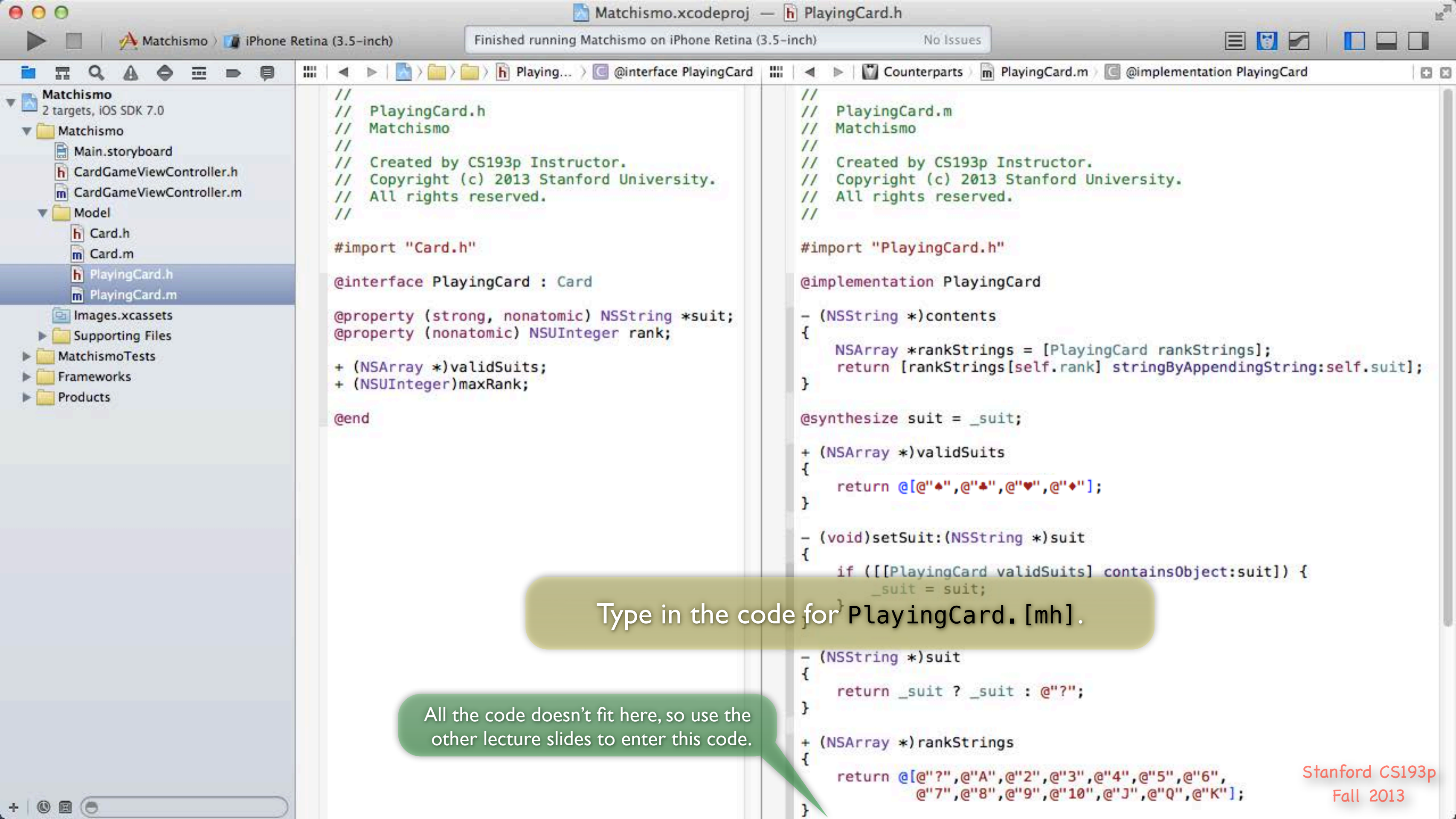
... and the Model group in the Navigator.

```
sity.  
  
ring:self.contents]) {
```



Hopefully PlayingCard.[mh] appeared in your Model group!

Drag it in if not.



- Matchismo
  - 2 targets, iOS SDK 7.0
  - Matchismo
    - Main.storyboard
    - CardGameViewController.h
    - CardGameViewController.m
    - Model
      - Card.h
      - Card.m
      - PlayingCard.h
      - PlayingCard.m
    - Images.xcassets
    - Supporting Files
    - MatchismoTests
    - Frameworks
    - Products

```
//
//  PlayingCard.h
//  Matchismo
//
//  Created by CS193p Instructor.
//  Copyright (c) 2013 Stanford University.
//  All rights reserved.
//

#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

```
//
//  PlayingCard.m
//  Matchismo
//
//  Created by CS193p Instructor.
//  Copyright (c) 2013 Stanford University.
//  All rights reserved.
//

#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit;

+ (NSArray *)validSuits
{
    return @[@"♠", @"♣", @"♥", @"♦"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

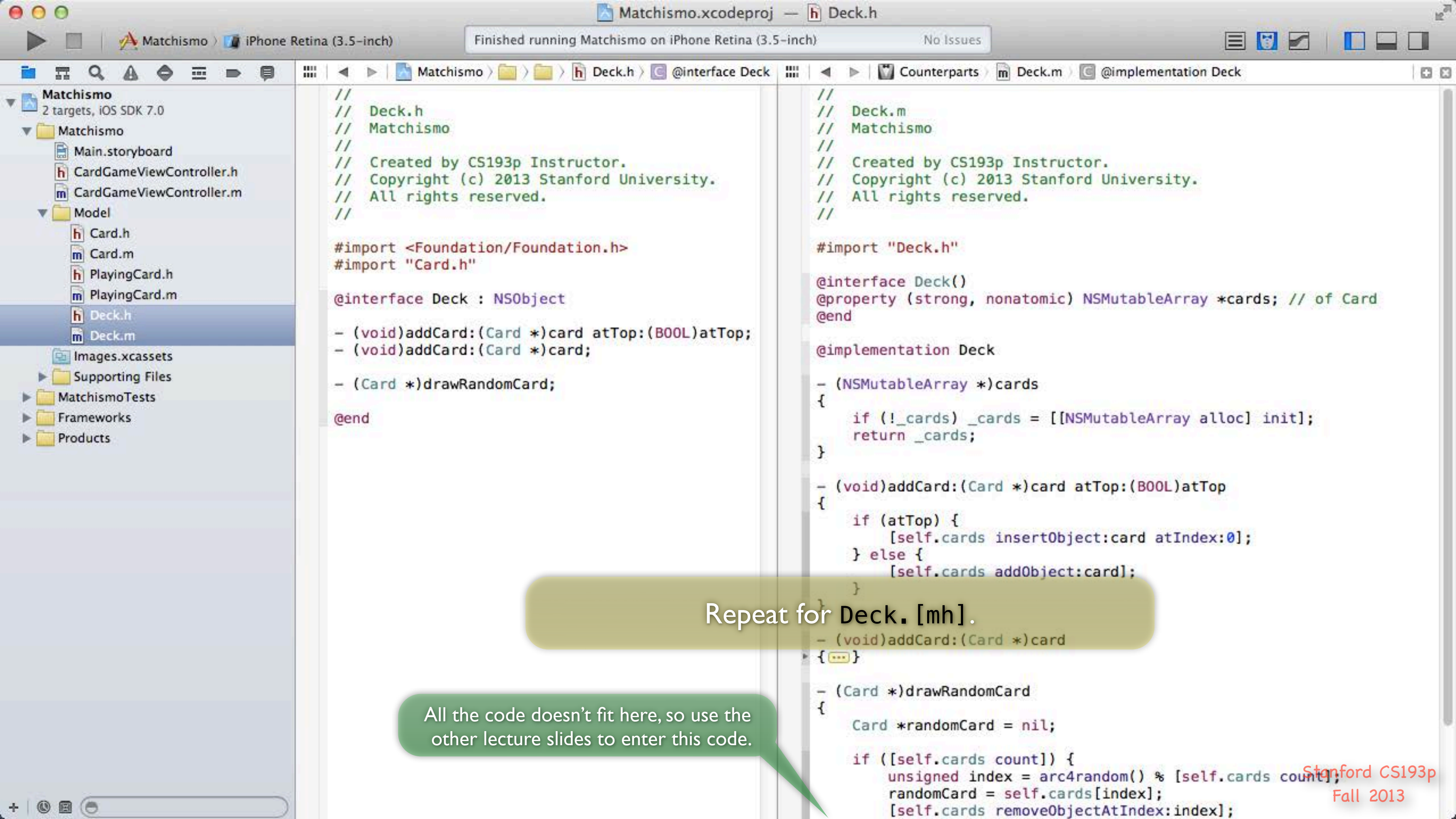
- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", @"4", @"5", @"6",
            @"7", @"8", @"9", @"10", @"J", @"Q", @"K"];
}

@end
```

Type in the code for PlayingCard. [mh].

All the code doesn't fit here, so use the other lecture slides to enter this code.

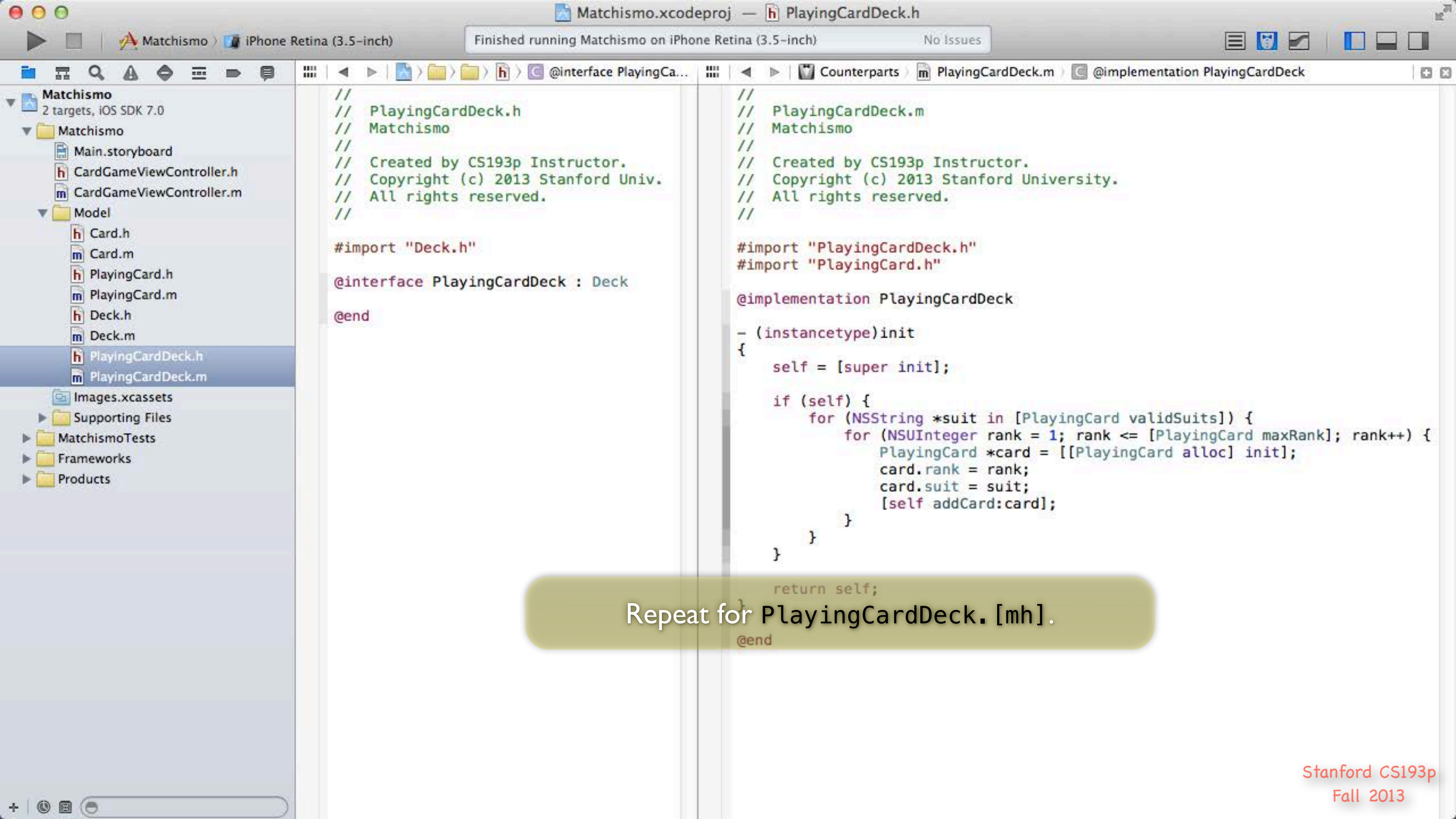


```
//  
// Deck.h  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import <Foundation/Foundation.h>  
#import "Card.h"  
  
@interface Deck : NSObject  
  
- (void)addCard:(Card *)card atTop:(BOOL)atTop;  
- (void)addCard:(Card *)card;  
  
- (Card *)drawRandomCard;  
  
@end
```

```
//  
// Deck.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "Deck.h"  
  
@interface Deck()  
@property (strong, nonatomic) NSMutableArray *cards; // of Card  
@end  
  
@implementation Deck  
  
- (NSMutableArray *)cards  
{  
    if (!_cards) _cards = [[NSMutableArray alloc] init];  
    return _cards;  
}  
  
- (void)addCard:(Card *)card atTop:(BOOL)atTop  
{  
    if (atTop) {  
        [self.cards insertObject:card atIndex:0];  
    } else {  
        [self.cards addObject:card];  
    }  
}  
  
- (void)addCard:(Card *)card  
{ ... }  
  
- (Card *)drawRandomCard  
{  
    Card *randomCard = nil;  
  
    if ([self.cards count]) {  
        unsigned index = arc4random() % [self.cards count];  
        randomCard = self.cards[index];  
        [self.cards removeObjectAtIndex:index];  
    }  
}
```

Repeat for Deck. [mh].

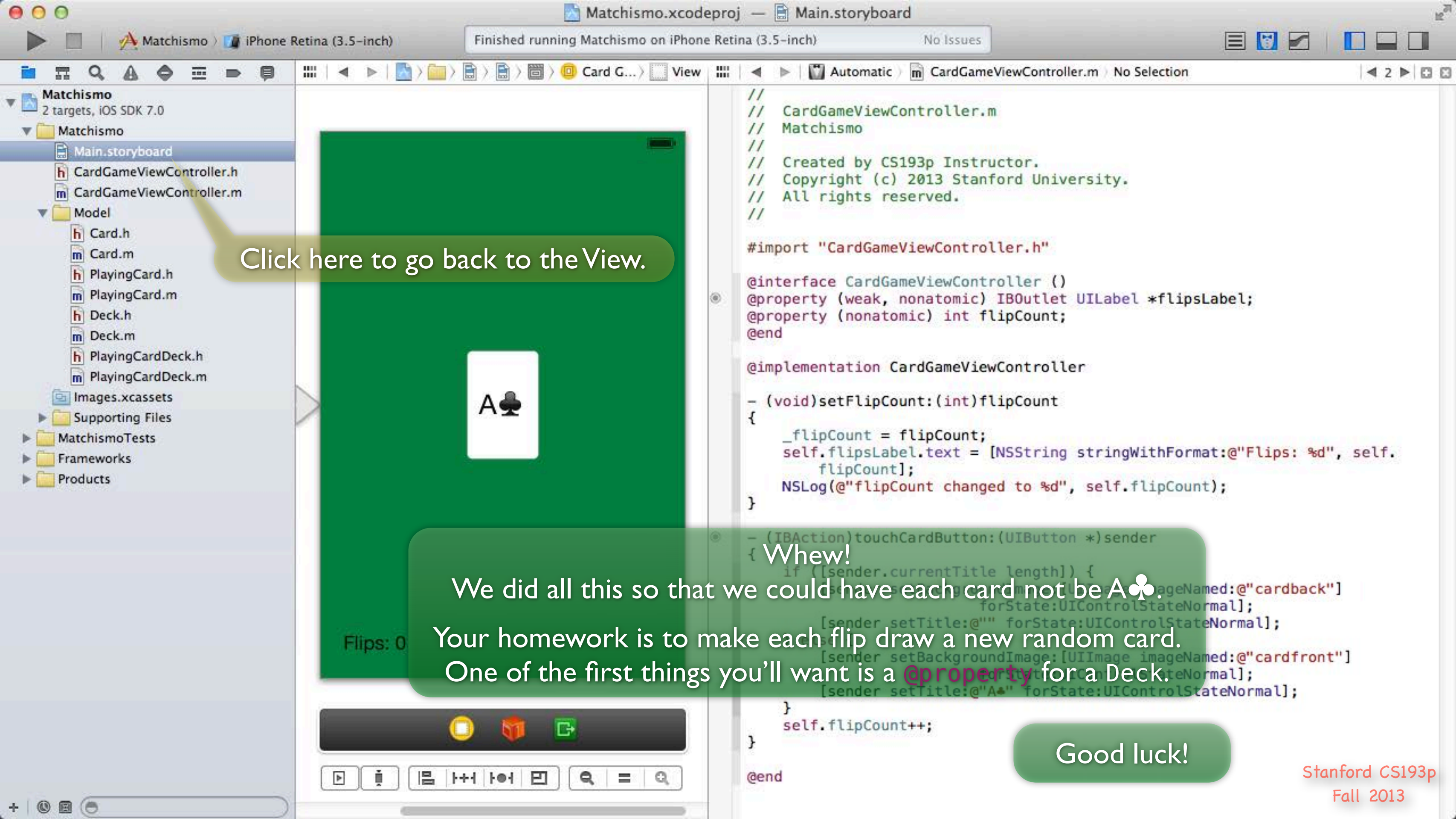
All the code doesn't fit here, so use the other lecture slides to enter this code.



```
//  
// PlayingCardDeck.h  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford Univ.  
// All rights reserved.  
//  
  
#import "Deck.h"  
  
@interface PlayingCardDeck : Deck  
  
@end
```

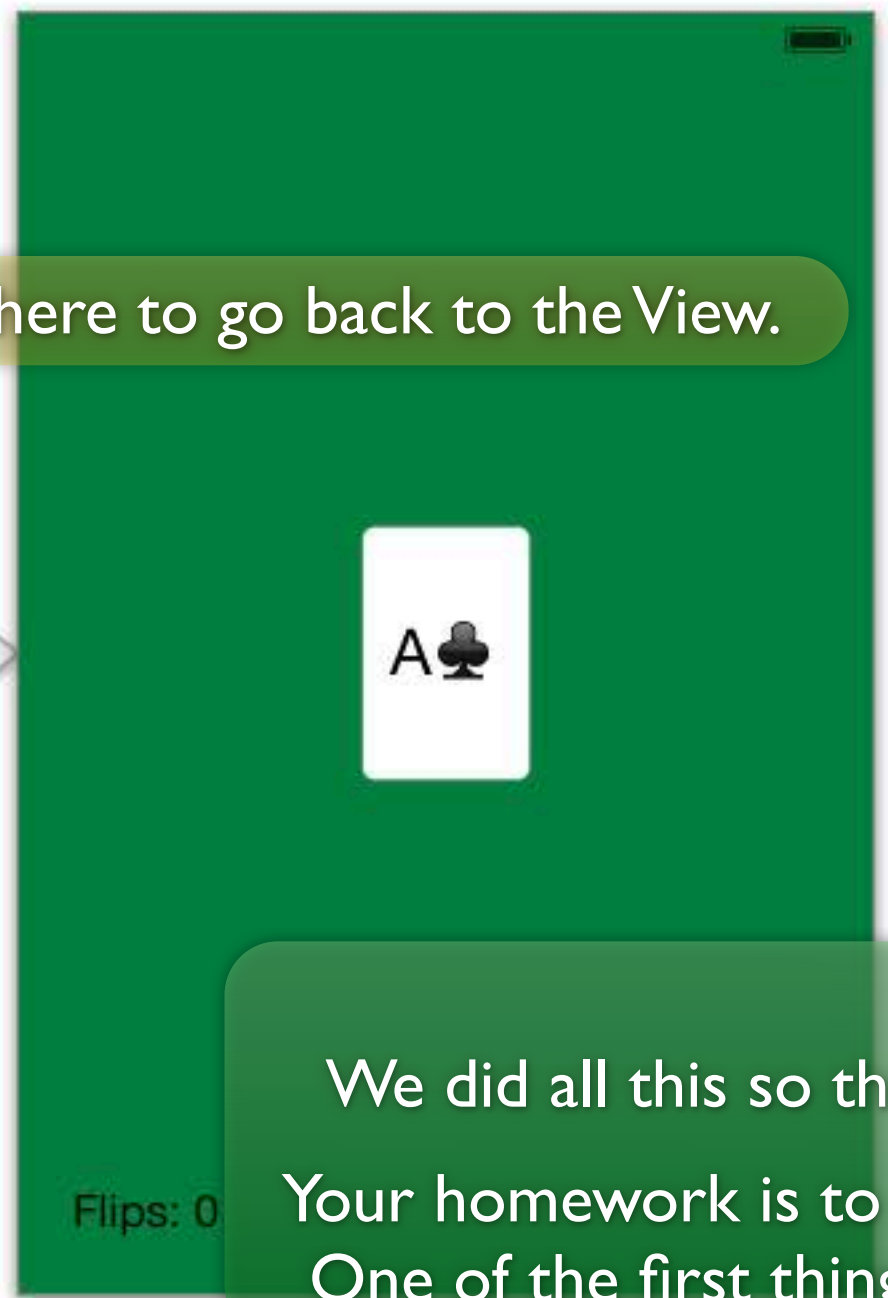
```
//  
// PlayingCardDeck.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "PlayingCardDeck.h"  
#import "PlayingCard.h"  
  
@implementation PlayingCardDeck  
  
- (instancetype)init  
{  
    self = [super init];  
  
    if (self) {  
        for (NSString *suit in [PlayingCard validSuits]) {  
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {  
                PlayingCard *card = [[PlayingCard alloc] init];  
                card.rank = rank;  
                card.suit = suit;  
                [self addCard:card];  
            }  
        }  
    }  
  
    return self;  
}  
  
@end
```

Repeat for PlayingCardDeck. [mh].



- Matchismo
- 2 targets, iOS SDK 7.0
- Matchismo
  - Main.storyboard
  - CardGameViewController.h
  - CardGameViewController.m
  - Model
    - Card.h
    - Card.m
    - PlayingCard.h
    - PlayingCard.m
    - Deck.h
    - Deck.m
    - PlayingCardDeck.h
    - PlayingCardDeck.m
  - Images.xcassets
  - Supporting Files
  - MatchismoTests
  - Frameworks
  - Products

Click here to go back to the View.



```
//  
// CardGameViewController.m  
// Matchismo  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "CardGameViewController.h"  
  
@interface CardGameViewController ()  
@property (weak, nonatomic) IBOutlet UILabel *flipsLabel;  
@property (nonatomic) int flipCount;  
@end  
  
@implementation CardGameViewController  
  
- (void)setFlipCount:(int)flipCount  
{  
    _flipCount = flipCount;  
    self.flipsLabel.text = [NSString stringWithFormat:@"Flips: %d", self.  
        flipCount];  
    NSLog(@"flipCount changed to %d", self.flipCount);  
}  
  
- (IBAction)touchCardButton:(UIButton *)sender  
{  
    Whew!  
    if ([sender.currentTitle length]) {  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
        [sender setTitle:@"" forState:UIControlStateNormal];  
        [sender setBackgroundImage:[UIImage imageNamed:@"cardfront"]  
            forState:UIControlStateNormal];  
        [sender setTitle:@"A♣" forState:UIControlStateNormal];  
        self.flipCount++;  
    }  
}  
  
@end
```

Whew!  
We did all this so that we could have each card not be A♣.  
Your homework is to make each flip draw a new random card.  
One of the first things you'll want is a @property for a Deck.

Good luck!

# Coming Up

- Needs more Card Game!

Your homework will be to have that single card flip through an entire Deck of PlayingCards. Next week we'll make multiple cards and put in logic to match them against each other.

- Also next week ...

Objective-C language in depth

Foundation classes: arrays, dictionaries, strings, etc.

Dynamic vs. static typing

Protocols, categories and much, much more!