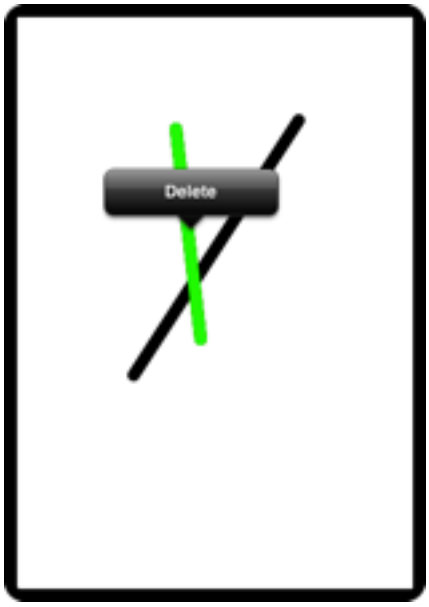

Creamos en Xcode, cree un nuevo proyecto “Single View Application” para iPhone vacío y con nombre **TouchTracker**. No especificamos un prefijo de clase y marcamos sólo la casilla de ARC (referencia automático de conteo).

Creamos una nueva clase tipo NSObject con el nombre “Line”.

En **Line.h**, declaramos dos propiedades CGPoint:

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

@interface Line : NSObject

@property (nonatomic) CGPoint begin;
@property (nonatomic) CGPoint end;

@end
```

En **Line.m**, sintetizamos las dos propiedades:

```
#import "Line.h"

@implementation Line

@synthesize begin, end;

@end
```

Creamos una nueva clase tipo NSObject llamada **TouchDrawView**.

En **TouchDrawView.h**, cambiamos la superclase a UIView.

Declarar dos objetos: un array: que contiene las líneas completas y un diccionario para mantener las líneas que aún se están dibujando.

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

@interface TouchDrawView : NSObject
@interface TouchDrawView : UIView
{
    NSMutableDictionary *linesInProcess;
    NSMutableArray *completeLines;
}
- (void)clearAll;
@end
```

Ahora se necesita un controlador de vista para administrar una instancia de **TouchDrawView** en TouchTracker.
Por ello creamos una nueva subclase NSObject llamado TouchViewController.
En **TouchViewController.h**, cambiamos la superclase de UIViewController.

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

@interface TouchDrawView : NSObject
@interface TouchDrawView : UIViewController

@end
```

En **TouchViewController.m**, escribimos en **loadView** para configurar una instancia de TouchDrawView tal como TouchViewController. Tenemos que importar el archivo de cabecera TouchDrawView.h en la parte superior de este archivo.

```
#import "TouchViewController.h"
#import "TouchDrawView.h"

@implementation TouchViewController

- (void)loadView
{
    [self setView:[[TouchDrawView alloc] initWithFrame:CGRectZero]];
}

@end
```

En **AppDelegate.m**, creamos una instancia de TouchViewController y la establecemos como el rootViewController de la ventana (window) . No nos olvidamos de importar el archivo de cabecera TouchViewController en este archivo.

```
#import "AppDelegate.h"
#import "TouchViewController.h"

@implementation AppDelegate

@synthesize window = _window;

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
    // Override point for customization after application launch.

    TouchViewController *tvc = [[TouchViewController alloc] init];
    [[self window] setRootViewController:tvc];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

DIBUJANDO CON TouchDrawView

TouchDrawView mantendrá un registro de todas las líneas que han sido dibujadas y las que están en proceso de elaboración.
En **TouchDrawView.m**, creamos las dos colecciones e importamos el encabezado para la clase **Line**.

```
#import "TouchDrawView.h"
#import "Line.h"

@implementation TouchDrawView

- (id)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];

    if (self) {
        linesInProcess = [[NSMutableDictionary alloc] init];

        // Don't let the autocomplete fool you on the next line,
        // make sure you are instantiating an NSMutableArray
        // and not an NSMutableDictionary!
        completeLines = [[NSMutableArray alloc] init];

        [self setBackgroundColor:[UIColor whiteColor]];

        [self setMultipleTouchEnabled:YES];
    }

    return self;
}
```

Hemos habilitado explícitamente eventos multitáctiles enviando el mensaje **setMultipleTouchEnabled:**. Sin esto, sólo un toque a la vez puede estar activo en una vista. Si otro dedo toca la vista, será ignorado, y la vista no se enviará **touchesBegan:withEvent**: o cualquiera de los otros mensajes **UIResponder**.

Ahora escribimos en el método **drawRect:** para crear líneas usando funciones de “Core Graphics”:

```
- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 10.0);
    CGContextSetLineCap(context, kCGLineCapRound);

    // Draw complete lines in black
    [[UIColor blackColor] set];
    for (Line *line in completeLines) {
        CGContextMoveToPoint(context, [line begin].x, [line begin].y);
        CGContextAddLineToPoint(context, [line end].x, [line end].y);
        CGContextStrokePath(context);
    }

    // Draw lines in process in red (Don't copy and paste the previous loop;
    // this one is way different)
    [[UIColor redColor] set];
    for (NSValue *v in linesInProcess) {
        Line *line = [linesInProcess objectForKey:v];
        CGContextMoveToPoint(context, [line begin].x, [line begin].y);
        CGContextAddLineToPoint(context, [line end].x, [line end].y);
        CGContextStrokePath(context);
    }
}
```

Por último en **TouchDrawView.m** escribimos un método para limpiar las “colecciones” y redibujar la vista.

```
- (void)clearAll
{
    // Clear the collections
    [linesInProcess removeAllObjects];
    [completeLines removeAllObjects];

    // Redraw
    [self setNeedsDisplay];
}
```

TOUCH en líneas

Una línea se define por dos puntos. Nuestra **Line** almacena estos puntos como propiedades con nombre **begin** y **end** .
Cuando comienza un “touch” , crea una línea y configurar tanto “begin” y “end” en el punto donde se inició el contacto, donde empezó el “touch”.
Cuando el contacto/touch se mueve , se va actualizando el “end”.
Cuando termina el touch , tendremos la línea completa.

Hay dos objetos de colección que mantienen instancias a **Line**.
La líneas de que se han completado se almacenan en el array **completeLines** .
Las líneas que siguen siendo dibujadas , sin embargo , se almacenan en un **NSMutableDictionary**. ¿Por qué necesitamos un diccionario ? Hemos habilitado multi-touch, por lo que un usuario puede dibujar más de una línea a la vez. Esto significa que tenemos que realizar un seguimiento de los eventos táctiles van con qué línea . Por ejemplo , imagine que el usuario toca la pantalla con dos dedos la creación de dos instancias de **Line** . Entonces uno de esos dedos se mueve . El TouchDrawView se envía un mensaje para el evento, pero ¿cómo puede saber qué línea de actualizar ?
Es por esto que estamos usando un diccionario en lugar de una matriz para las líneas de proceso. Cuando comienza un touch , vamos a grabar la dirección del objeto **UITouch** que se pasa y se envuelve en una instancia **NSValue** . Una nueva línea se crea y se agrega al diccionario , y la NSValue será su clave. A medida que recibimos más eventos táctiles , podemos utilizar la dirección de la **UITouch** que se pasa para acceder y actualizar la línea correcta.

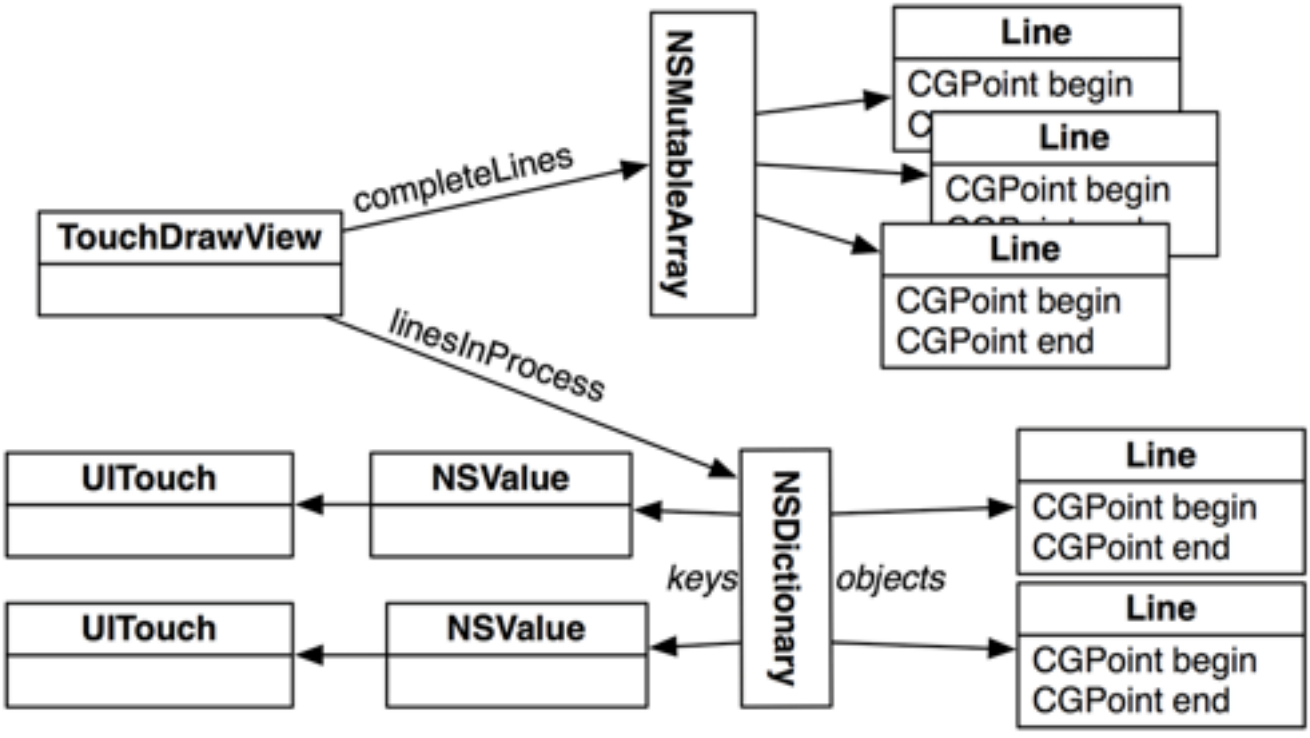

Figure - Object diagram for TouchTracker

Volvamos a los métodos para el manejo de eventos de toque.

En primer lugar, en **TouchDrawView.m**, sobrescribimos **touchesBegan:withEvent:** para crear una nueva instancia de **Line** y guárdela en un **NSMutableDictionary**.

```
- (void)touchesBegan:(NSSet *)touches
           withEvent:(UIEvent *)event
{
    for (UITouch *t in touches) {

        // Is this a double tap?
        if ([t tapCount] > 1) {
            [self clearAll];
            return;
        }

        // Use the touch object (packed in an NSValue) as the key
        NSValue *key = [NSValue valueWithNonretainedObject:t];

        // Create a line for the value
        CGPoint loc = [t locationInView:self];
        Line *newLine = [[Line alloc] init];
        [newLine setBegin:loc];
        [newLine setEnd:loc];

        // Put pair in dictionary
        [linesInProcess setObject:newLine forKey:key];
    }
}
```

En segundo lugar, en **TouchDrawView.m**, sobrescribimos **touchesMoved:withEvent:** para actualizar el final del punto de la línea asociada con el movimiento del “touch”.

```
- (void)touchesMoved:(NSSet *)touches
           withEvent:(UIEvent *)event
{
    // Update linesInProcess with moved touches
    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];

        // Find the line for this touch
        Line *line = [linesInProcess objectForKey:key];

        // Update the line
        CGPoint loc = [t locationInView:self];
        [line setEnd:loc];
    }
    // Redraw
    [self setNeedsDisplay];
}
```

Cuando termina un ”touch”, se necesita finalizar la línea. Sin embargo, un toque puede terminar por dos razones: el usuario levanta el dedo de la pantalla (touchesEnded: withEvent :) o el sistema operativo interrumpe su aplicación (touchesCancelled: withEvent :). Una llamada de teléfono, por ejemplo, puede interrumpir la aplicación.

En muchas aplicaciones, tendremos que manejar estos dos eventos de manera diferente. Sin embargo, para TouchTracker, escribiremos un método para manejar ambos casos.

Por ello declaramos un nuevo método en **TouchDrawView.h**.

```
- (void)endTouches:(NSSet *)touches;
```

En **TouchDrawView.m** implementamos el método.

```
- (void)endTouches:(NSSet *)touches
{
    // Remove ending touches from dictionary
    for (UITouch *t in touches) {
        NSValue *key = [NSValue valueWithNonretainedObject:t];
        Line *line = [linesInProcess objectForKey:key];

        // If this is a double tap, 'line' will be nil,
        // so make sure not to add it to the array
        if (line) {
            [completeLines addObject:line];
            [linesInProcess removeObjectForKey:key];
        }
    }
    // Redraw
    [self setNeedsDisplay];
}
```

Finalmente sobrescribimos los dos métodos desde UIResponder para llamar a **endTouches**.

```
- (void)touchesEnded:(NSSet *)touches
           withEvent:(UIEvent *)event
{
    [self endTouches:touches];
}

- (void)touchesCancelled:(NSSet *)touches
               withEvent:(UIEvent *)event
{
    [self endTouches:touches];
}
```

Compilamos y ejecutamos la aplicación. Podemos dibujar líneas .

La cadena de respuesta

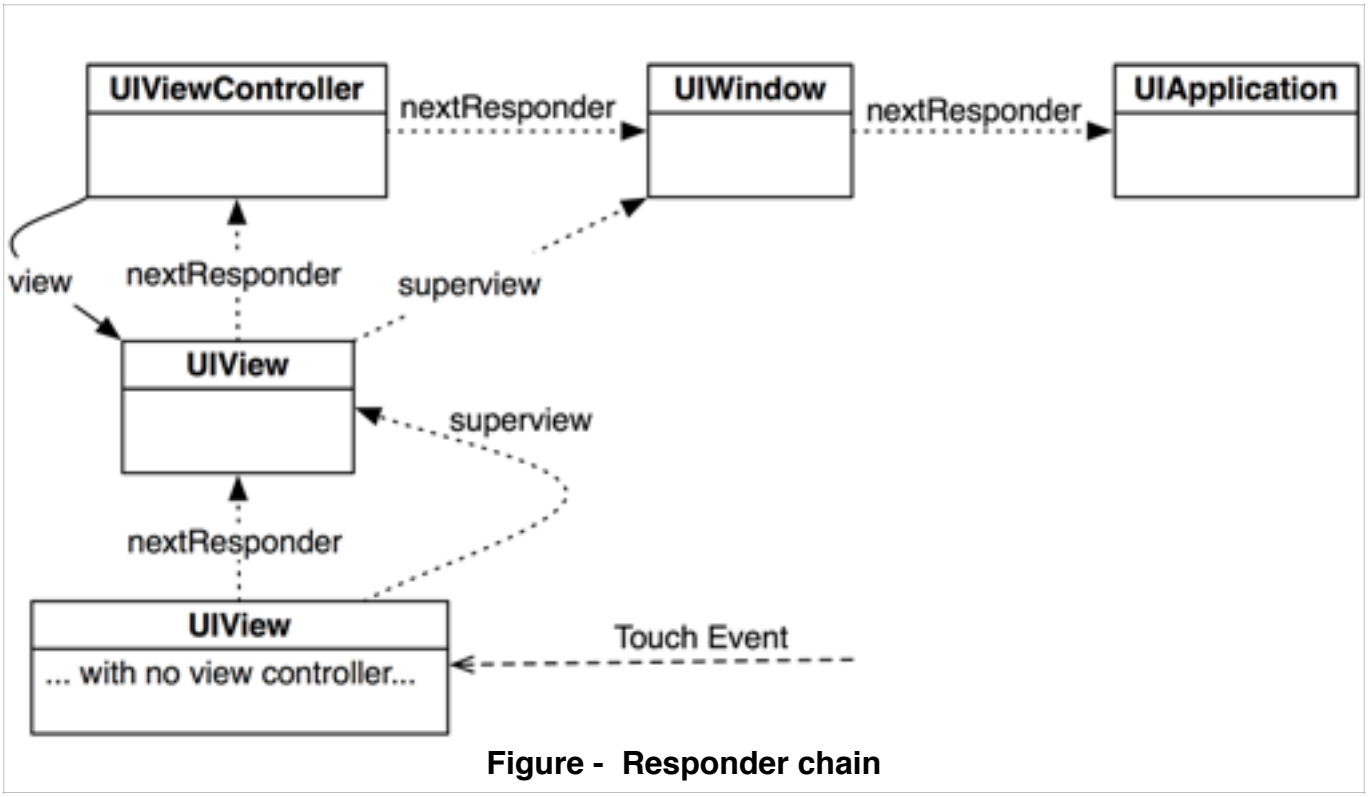

Figure - Responder chain

UIResponder puede recibir eventos de toque.
UIView es un ejemplo de una subclase **UIResponder** , pero existen muchos otros, incluyendo **UIViewController , UIApplication y UIWindow** .
Aunque no se puede tocar una **UIViewController**, puesto que no es un objeto de la pantalla y no podemos enviar un evento de contacto directamente a un **UIViewController** , sí que los controladores de vista puede recibir eventos a través de la cadena de respuesta .

Cada **UIResponder** tiene un puntero llamado “nextResponder” , y juntos estos objetos constituyen la cadena de resultados. Un evento táctil se inicia en la vista en que se toca. El nextResponder de una vista es normalmente su **UIViewController** (si lo tiene) o su supervista/superview (si no lo tiene) .
El nextResponder de un controlador de vista es normalmente el “superview” de su vista/view .
La superview de nivel superior es la ventana/window .
El nextResponder de la ventana es la instancia “singleton” de **UIApplication** . Si la aplicación no maneja el evento , entonces se descarta.

UIGestureRecognizer

No se crean instancias directamente en **UIGestureRecognizer**. En lugar de ello, hay un número de subclases de **UIGestureRecognizer**, y cada una es responsable de reconocer un gesto particular.

Para utilizar una instancia de una subclase **UIGestureRecognizer**, le das un par de objetivos de acción para atacar a una vista. Cada vez que el reconocedor de movimientos reconoce su gesto en la vista, se enviará el mensaje de acción a su objetivo. Todos los mensajes de acción **UIGestureRecognizer** tienen la misma forma:

- (Void) acción: (UIGestureRecognizer *) GestureRecognizer;

Al reconocer un gesto, el reconocedor de movimientos intercepta los toques con destino a la vista. Por lo tanto, una vista con reconocedores gesto puede no recibir los mensajes **UIResponder** típicos como **touchesBegan:withEvent:**.

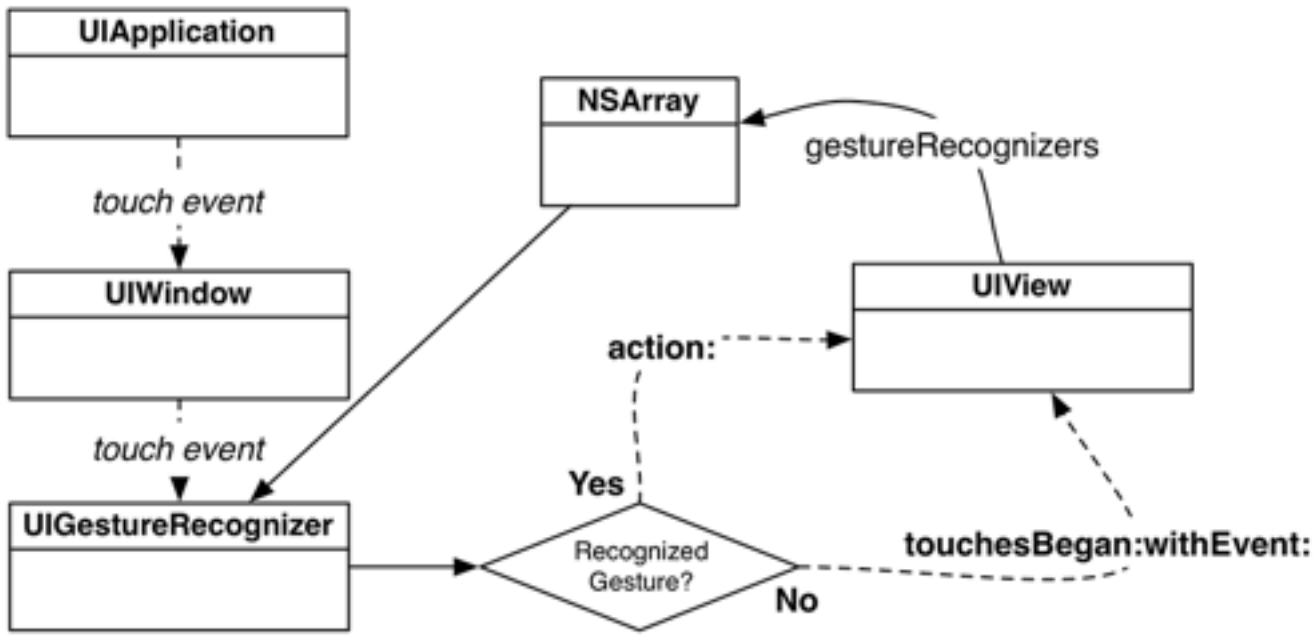

Figura . UIGestureRecognizers intercept UITouches

Detectando “Taps” con UITapGestureRecognizer

La primera subclase **UIGestureRecognizer** que vamos a utilizar es **UITapGestureRecognizer**.
Cuando el usuario toca una línea en el programa “TouchTracker”, se le presentará un menú que les permite borrarlo.

En la primera parte de esta sección, vamos a reconocer a un “tap”, determinar qué línea se encuentra cerca de donde se produjo el “tap”, almacenar una referencia a esa línea, y cambiar el color de esa línea a verde para que el usuario sepa que ha sido seleccionado .

En **TouchDrawView.m**, editamos el método **initWithFrame:** para crear una instancia de **UITapGestureRecognizer** y atacar a la **TouchDrawView** que se está inicializando.

```
- (id)initWithFrame:(CGRect)r
{
    self = [super initWithFrame:r];

    if (self) {
        linesInProcess = [[NSMutableDictionary alloc] init];
        completeLines = [[NSMutableArray alloc] init];

        [self setBackgroundColor:[UIColor whiteColor]];

        [self setMultipleTouchEnabled:YES];

        UITapGestureRecognizer *tapRecognizer =
        [[UITapGestureRecognizer alloc] initWithTarget:self
                                                action:@selector(tap:)];

        [self addGestureRecognizer:tapRecognizer];
    }
    return self;
}
```

Ahora cada vez que un “tap” se produce en **TouchDrawView**, el **UITapGestureRecognizer** enviará el mensaje “**tap:**”.
En TouchDrawView.m, implementar el método **tap:**. Mandamos un mensaje a la consola.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");
}
```

Al compilar y ejecutar la aplicación y luego tocar en la pantalla, debemos notar dos cosas:
la consola informa de que un “tap” fue reconocido, y un punto se dibuja en la vista. Ese punto es una línea muy corta y un “bug”. En esta aplicación, no queremos dibujar nada en respuesta al “tap”.
Agregamos el código siguiente al “tap”: para eliminar las líneas en proceso y redibujar la vista.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");

    // If we just tapped, remove all lines in process
    // so that a tap doesn't result in a new line
    [linesInProcess removeAllObjects];

    [self setNeedsDisplay];
}
```

Compilamos y ejecutamos de nuevo. Ahora un “tap” no debe tener ningún efecto, excepto para el registro de una declaración en la consola.
Ahora, con el fin de "seleccionar una línea," tenemos que encontrar una línea cerca de donde se produjo el “tap” y luego almacenar una referencia a esa línea para más adelante.
En **TouchDrawView.h**, declarar un nuevo método y una nueva propiedad. Además, en la parte superior de este archivo, declararemos la clase **Line**.

```
@class Line;

@interface TouchDrawView : UIView
{
    NSMutableDictionary *linesInProcess;
    NSMutableArray *completeLines;
}
@property (nonatomic, weak) Line *selectedLine;

- (Line *)lineAtPoint:(CGPoint)p;

- (void)clearAll;
- (void)endTouches:(NSSet *)touches;

@end
```

Sintetizamos **selectedLine** en **TouchDrawView.m** .

```
@implementation TouchDrawView
@synthesize selectedLine;
```

Implementamos **lineAtPoint:** en **TouchDrawView.m** para obtener una línea cerca del punto dado.

```
- (Line *)lineAtPoint:(CGPoint)p
{
    // Find a line close to p
    for (Line *l in completeLines) {
        CGPoint start = [l begin];
        CGPoint end = [l end];

        // Check a few points on the line
        for (float t = 0.0; t <= 1.0; t += 0.05) {
            float x = start.x + t * (end.x - start.x);
            float y = start.y + t * (end.y - start.y);

            // If the tapped point is within 20 points, let's return this
line
            if (hypot(x - p.x, y - p.y) < 20.0) {
                return l;
            }
        }
    }

    // If nothing is close enough to the tapped point, then we didn't select
a line
    return nil;
}
```

(Hay una mejor forma de implementar **lineAtPoint:**, y esta implementación simplista está bien para nuestros propósitos.)
El punto que nos interesa, por supuesto, es el lugar donde se produjo el "tap". Podemos obtener fácilmente esta información. Cada **UIGestureRecognize**r tiene un método **locationInView:**. El envío de este mensaje al reconocedor de movimientos nos dará las coordenadas donde se produjo el gesto en el sistema de coordenadas de la vista que se pasa como argumento.
En **TouchDrawView.m**, envíe el mensaje **locationInView:** al reconocedor de movimientos, pasa el resultado a **lineAtPoint**:, y hacer que la línea devuelta la **selectedLine**.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");

    CGPoint point = [gr locationInView:self];
    [self setSelectedLine:[self lineAtPoint:point]];

    // If we just tapped, remove all lines in process so a new one
    // isn't drawn on every tap
    [linesInProcess removeAllObjects];

    [self setNeedsDisplay];
}
```

Por último, en **TouchDrawView.m**, actualizamos **drawRect:** para dibujar "selectedLine" en verde.
Nos Aseguramos que el código viene después de dibujar el completo y en las líneas de progreso.

```
[[UIColor redColor] set];
    for (NSValue *v in linesInProcess) {
        Line *line = [linesInProcess objectForKey:v];
        CGContextMoveToPoint(context, [line begin].x, [line begin].y);
        CGContextAddLineToPoint(context, [line end].x, [line end].y);
        CGContextStrokePath(context);
    }

    // If there is a selected line, draw it
    if ([self selectedLine]) {
        [[UIColor greenColor] set];
        CGContextMoveToPoint(context, [[self selectedLine] begin].x,
                             [[self selectedLine] begin].y);
        CGContextAddLineToPoint(context, [[self selectedLine] end].x,
                                [[self selectedLine] end].y);
        CGContextStrokePath(context);
    }

}
```

Al compilar y ejecutar la aplicación. Si dibujamos unas pocas líneas y luego punteamos(tapped) en una. La línea "tapped" debería aparecer en verde.

UIMenuController

Cuando el usuario selecciona una línea, queremos que un menú aparezca justo donde el usuario realizó el “tap”, ofreciendo la opción de borrar esa línea.

Hay una clase integrada para la prestación de este tipo de menú llamada **UIMenuController**.

Un controlador de menú tiene una lista de elementos de menú y se presenta en una vista existente.

Cada elemento tiene un título (que aparece en el menú) y una acción (el mensaje que envía a la vista donde se presenta el menú).

Figura UIMenuController

Sólo hay una **UIMenuController** por aplicación.

Cuando deseamos presentar esta instancia, lo llenaremos de elementos de menú (menu items), le daremos un rectángulo para presentar ello, y lo configuramos para que sea visible.

Hacer esto con un método **tap:** en TouchDrawView.m si el usuario ha pulsado sobre una línea. Si el usuario tocó en alguna parte que no está cerca de una línea, la línea actualmente seleccionada será desactivada, y el controlador de menú se ocultará.

```
- (void)tap:(UIGestureRecognizer *)gr
{
    NSLog(@"Recognized tap");
    CGPoint point = [gr locationInView:self];
    [self setSelectedLine:[self lineAtPoint:point]];

    [linesInProcess removeAllObjects];

    if ([self selectedLine]) {
        // We'll talk about this shortly
        [self becomeFirstResponder];

        // Grab the menu controller
        UIMenuController *menu = [UIMenuController sharedMenuController];

        // Create a new "Delete" UIMenuItem
        UIMenuItem *deleteItem = [[UIMenuItem alloc] initWithTitle:@"Delete"

action:@selector(deleteLine:)];
        [menu setMenuItems:[NSArray arrayWithObject:deleteItem]];

        // Tell the menu where it should come from and show it
        [menu setTargetRect:CGRectMake(point.x, point.y, 2, 2) inView:self];
        [menu setMenuVisible:YES animated:YES];
    } else {
        // Hide the menu if no line is selected
        [[UIMenuController sharedMenuController] setMenuVisible:NO
animated:YES];
    }
    [self setNeedsDisplay];
}
```

Para que aparezca un controlador de menú, la vista que se presenta el controlador de menú tiene que ser la primera respuesta de la ventana. Así que enviamos el mensaje **becomeFirstResponder** al **TouchDrawView** antes de obtener el controlador de menú. Se debe reemplazar **canBecomeFirstResponder** en una vista si se necesita para convertirse en el primer nivel de respuesta.

En **TouchDrawView.m**, sobrescribimos este método para devolver YES.

```
- (BOOL)canBecomeFirstResponder
{
    return YES;
}
```

Podemos compilar y ejecutar la aplicación, pero al seleccionar una línea, no aparecerá el menú. Esto se debe a que los controladores de menú son inteligentes: Cuando un controlador de menú se va a mostrar, va a través de cada elemento de menú y le pregunta su opinión de si implementa el mensaje de acción para ese elemento. Si la vista no implementa este método, el controlador de menú no se mostrará la opción de menú correspondiente. Para obtener que el elemento de menú Eliminar (“Delete”) aparezca, implementamos **deleteLine:** en **TouchDrawView.m**.

```
- (void)deleteLine:(id)sender
{
    // Remove the selected line from the list of completeLines
    [completeLines removeObject:[self selectedLine]];

    // Redraw everything
    [self setNeedsDisplay];
}
```

Compilamos y ejecutamos la aplicación. Dibujamos una línea, pulse sobre ella (tap) y entonces seleccionamos Eliminar (Delete) en el menú. La propiedad **selectedLine** fue declarado como débil (weak) al declararla, por lo que cuando se elimina el array **completeLines**, **selectedLine** se ajusta automáticamente a cero (nil).

UILongPressGestureRecognizer

Vamos a probar otras dos subclases de **UIGestureRecognizer: UILongPressGestureRecognizer** y **UIPanGestureRecognizer**. Cuando el usuario presiona en una línea (una pulsación larga), se debe seleccionar esa línea. Mientras se selecciona una línea de esta manera, el usuario debe ser capaz de arrastrar la línea (“pan”) a una nueva posición.

En esta sección, nos centraremos en reconocer una larga presión.

En **TouchDrawView.m**, instanciamos **UILongPressGestureRecognizer** en **initWithFrame:**.

```
[self addGestureRecognizer:tapRecognizer];

UILongPressGestureRecognizer *pressRecognizer =
    [[UILongPressGestureRecognizer alloc] initWithTarget:self

action:@selector(longPress:)];
[self addGestureRecognizer:pressRecognizer];
```

Ahora, cuando el usuario mantiene presionado sobre el **TouchDrawView** , el mensaje **LongPress:** se enviará a él. Por defecto, un toque (tap) debe mantenerse 0,5 segundos para llegar a ser una pulsación larga, pero podemos cambiar el **minimumPressDuration** del reconocedor de movimientos , si queremos.

Un “tap” es un simple gesto. En el momento en que se reconoce , el gesto ha terminado, y se ha producido un “tap”. Una pulsación larga , por otro lado , es un gesto que se produce con el tiempo y se define por tres eventos separados .

Por ejemplo, cuando el usuario toca una vista, el reconocedor de larga pulsación (long press) notifica un posible (“possible”) “long press” , pero debe esperar y ver si este contacto se mantiene el tiempo suficiente para convertirse en un gesto ”long press”.

Una vez que el usuario mantiene el contacto el tiempo suficiente, la pulsación larga se reconoce y el gesto ha comenzado (“begun”). Cuando el usuario quita el dedo , el gesto ha terminado (“ended”) .

Cada uno de estos acontecimientos provoca un cambio en la propiedad estado del reconocedor de movimientos/gestos . Por ejemplo, el estado del tiempo de reconocimiento de la prensa se ha descrito anteriormente sería **UIGestureRecognizerStatePossible** , entonces **UIGestureRecognizerStateBegan** , y finalmente **UIGestureRecognizerStateEnded** .

Cuando un reconocedor de movimientos transiciona a cualquier estado que no sea el estado posible , envía su mensaje de acción a su objetivo . Esto significa que el reconocedor de “long press” recibe el mismo mensaje cuando una pulsación larga comienza y cuando termina. Estado del reconocedor de movimientos permite que el objetivo de determinar por qué se ha enviado el mensaje de acción y tomar la acción apropiada.

Nuestro plan para la implementación de nuestro método de acción **LongPress:** será:

Cuando la vista recibe **LongPress:** y la pulsación larga acaba de empezar , vamos a seleccionar la línea más cercana al lugar donde se produjo el gesto. Esto permite al usuario seleccionar una línea , manteniendo el dedo sobre la pantalla. Cuando la vista recibe **LongPress:** y la pulsación larga ha terminado, vamos a cancelar la selección de la línea.

En **TouchDrawView.m**, implementamos **LongPress**:.

```
- (void)longPress:(UIGestureRecognizer *)gr
{
    if ([gr state] == UIGestureRecognizerStateBegan) {
        CGPoint point = [gr locationInView:self];
        [self setSelectedLine:[self lineAtPoint:point]];

        if ([self selectedLine]) {
            [linesInProcess removeAllObjects];
        }
    } else if ([gr state] == UIGestureRecognizerStateEnded) {
        [self setSelectedLine:nil];
    }
    [self setNeedsDisplay];
}
```

Compilamos y ejecutamos la aplicación. Dibujamos una línea, y luego mantenemos pulsado sobre ella,; la línea se pondrá de color verde y se seleccionará; y se mantendrá así hasta que la dejemos.

UIPanGestureRecognizer

Una vez que se selecciona una línea durante una pulsación larga, queremos que el usuario sea capaz de mover la línea alrededor de la pantalla arrastrándola con el dedo. Así que necesitamos un reconocedor de movimientos para un dedo que se mueve alrededor de la pantalla. Este gesto se llama "**panning"**, y su subclase reconocedor de gesto es **UIPanGestureRecognizer**.

Normalmente, un reconocedor de movimientos no comparte los toques que intercepta. Una vez que se ha reconocido su gesto, se "come" ese toque, y ningún otro reconocedor tiene la oportunidad de manejar la situación. En nuestro caso, esto es malo: queremos reconocer gesto "pan" que sucede dentro de una larga pulsación. Vamos a ver cómo hacerlo.

En **TouchDrawView.h**, declaramos un delegado de protocolo **UIGestureRecognizerDelegate** y declaramos **UIPanGestureRecognizer** como una variable de instancia.

```
@interface TouchDrawView : UIView
    <UIGestureRecognizerDelegate>
{
    NSMutableDictionary *linesInProcess;
    NSMutableArray *completeLines;

    UIPanGestureRecognizer *moveRecognizer;
}
```

En **TouchDrawView.m**, agregamos código en **initWithFrame:**

Una instancia a **UIPanGestureRecognizer**, fijamos dos de sus propiedades, y lo agregamos en **TouchDrawView**.

```
[self addGestureRecognizer:pressRecognizer];

moveRecognizer = [[UIPanGestureRecognizer alloc]
                  initWithTarget:self action:@selector(moveLine:)];
[moveRecognizer setDelegate:self];
[moveRecognizer setCancelsTouchesInView:NO];
[self addGestureRecognizer:moveRecognizer];
```

Hay una serie de métodos en el protocolo **UIGestureRecognizerDelegate**, pero a nosotros sólo nos interesa uno - **gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:**.

Un reconocedor de movimientos enviará este mensaje a su delegado cuando se reconoce su gesto, pero se da cuenta que otro gesto de reconocimiento ha reconocido su gesto, también. Si este método devuelve YES, el reconocedor compartirá sus toques con otros reconocedores gesto.

En **TouchDrawView.m**, devuelve YES cuando el "moveRecognizer" envía el mensaje a su delegado.

```
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
    shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer
*)other
{
    if (gestureRecognizer == moveRecognizer)
        return YES;
    return NO;
}
```

Ahora, cuando el usuario inicia una pulsación larga, el **UIPanGestureRecognizer** le permitirá hacer un seguimiento de ese dedo también.
Cuando el dedo comienza a moverse, el reconocedor del “pan” pasará al estado “began” . Si estos dos reconocedores no podían trabajar de forma simultánea , el reconocedor de pulsación larga comenzaría , y el reconocedor de “pan” nunca pasará al estado “began” o envíar el mensaje de acción a su objetivo .

Además de los estados que hemos visto anteriormente , un reconocedor de movimientos “pan” soporta el estado “change” . Cuando un dedo comienza a moverse , el reconocedor e movimientos “pan” entra en el estado “began” y envía un mensaje a su objetivo.
Mientras que el dedo se mueve alrededor de la pantalla , el reconocedor de transiciones cambia de estado y envía su mensaje de acción a su objetivo repetidamente.
Por último , cuando el dedo deja la pantalla , el estado del reconocedor se establece en “end”, y el mensaje final es entregado al objetivo/target .

Ahora necesitamos implementar el método **MoveLine:** que el reconocedor “pan” envía su objetivo. En esta implementación, se va a enviar el mensaje **translationInView:** al reconocedor “tap”. Este método **UIPanGestureRecognizer** devuelve hasta qué punto el “pan” se ha movido como un CGPoint en el sistema de coordenadas de la vista paseo como argumento .

Por lo tanto , cuando se inicia el gesto “pan” , esta propiedad se establece en el punto cero (donde tanto x como y son igual a cero). A medida que el “pan” se mueve, este valor se actualiza - si el pan va muy lejos a la derecha, que tiene un alto valor de x ; si el “pan” vuelve a donde empezó; su traslación regresa al punto cero.

En **TouchDrawView.m** , implementamos **MoveLine:**.

Observemos que debido a que se enviará el reconocedor de movimientos de un método de la clase **UIPanGestureRecognizer** , el parámetro de este método debe ser un puntero a una instancia de **UIPanGestureRecognizer** en lugar de **UIGestureRecognizer** .

```
- (void)moveLine:(UIPanGestureRecognizer *)gr
{
    // If we haven't selected a line, we don't do anything here
    if (![self selectedLine])
        return;

    // When the pan recognizer changes its position...
    if ([gr state] == UIGestureRecognizerStateChanged) {
        // How far has the pan moved?
        CGPoint translation = [gr translationInView:self];

        // Add the translation to the current begin and end points of the
line
        CGPoint begin = [[self selectedLine] begin];
        CGPoint end = [[self selectedLine] end];
        begin.x += translation.x;
        begin.y += translation.y;
        end.x += translation.x;
        end.y += translation.y;

        // Set the new beginning and end points of the line
        [[self selectedLine] setBegin:begin];
        [[self selectedLine] setEnd:end];

        // Redraw the screen
        [self setNeedsDisplay];
    }
}
```

Compilamos y ejecutamos la aplicación.

Tocamos sin soltar una línea y comenzamos a arrastrarla - notaremos inmediatamente que la línea y el dedo están muy fuera de sincronía. Esto tiene sentido porque va a añadir la traslación actual una y otra vez a los puntos extremos originales de la línea. Realmente necesitamos el reconocedor de movimientos para informar de la traslación desde la última vez que éste método fue llamado. Afortunadamente, podemos hacer esto. Podemos configurar la traslación de un reconocedor de movimientos "pan" de regreso sobre el punto cero cada vez que se informa de un cambio. Entonces, la próxima vez que se informa de un cambio, tendremos la traslación desde el último evento.

Cerca de la parte inferior de **MoveLine:** en **TouchDrawView.m**, añadimos la siguiente línea de código.

```
        [self setNeedsDisplay];

        [gr setTranslation:CGPointZero inView:self];
    }
}
```

Compilamos y ejecutamos la aplicación.

Antes de continuar, vamos a echar un vistazo a una propiedad se establece en el reconocedor de movimientos pan - **cancelsTouchesInView** . Cada **UIGestureRecognizer** tiene esta propiedad y , de forma predeterminada está fijada a YES. Esto significa que el reconocedor de movimientos captará cualquier cambio de manera que la vista no tendrá la oportunidad de manejar la situación a través de los métodos tradicionales **UIResponde**, como **touchesBegan:withEvent:** .

Por lo general, esto es lo que queremos, pero no siempre. En nuestro caso, el gesto que el reconocedor pan reconoce es el mismo tipo de toque que en la vista se encarga de dibujar líneas utilizando los métodos **UIResponder** . Si el reconocedor de movimientos omite estos toques , los usuarios no serán capaces de dibujar líneas .

Cuando se establece **cancelsTouchesInView** en NO, los toques que el reconocedor de movimientos reconoce también llegan a su destino a la vista a través de los métodos **UIResponder** . Esto permite que tanto el reconocedor y métodos **UIResponde**r de la vista puedan manejar los mismos toques .

Podríamos comentar la línea que establece **cancelsTouchesInView** NO y, compilar y ejecutar de nuevo para ver los efectos.

UIMenuController and UIResponderStandardEditActions

El **UIMenuController** es típicamente responsable de mostrar al usuario un menú "editar " cuando se muestra , pensemos en un campo de texto o un “text view” al presionar y mantener. Por lo tanto , un controlador de menú sin modificar (uno que no tiene establecidos los elementos de menú) ya cuenta con unos elementos de menú por defecto tales como Cortar , Copiar , entre otros . Cada elemento tiene un mensaje de acción asociado . Por ejemplo , el corte : se envía a la vista que presenta el controlador de menú cuando el elemento de menú Cut se toca .

Todos los **UIResponder**s implementar estos métodos, pero , de manera predeterminada , estos métodos no hacen nada . Las subclases como **UITextField** anulan estos métodos para hacer algo apropiado para su contexto , como cortar el texto seleccionado en ese momento . Los métodos se declaran en el protocolo **UIResponderStandardEditActions** .

Si reemplaza un método de **UIResponderStandardEditActions** en una vista, su elemento de menú aparecerá automáticamente en cualquier menú que muestra para esa vista . Esto funciona debido a que el controlador de menú envía el mensaje **canPerformAction:withSender:** a su vista , la cual devuelve SÍ o NO en función de si la vista implementa este método.

Si desea implementar uno de estos métodos , pero no quiere que aparezca en el menú , se puede reemplazar **canPerformAction:withSender:** para volver NO.

```
- (BOOL)canPerformAction:(SEL)action withSender:(id)sender
{
    if (action == @selector(copy:))
        return NO;

    // The superclass' implementation will return YES if the method is in
the .m file
    return [super canPerformAction:action withSender:sender];
}
```

Clearing Lines

Ahora mismo, el **TouchDrawView** borra sus líneas siempre que usemos dobles “taps”.

Disponemos de el código siguiente desde **touchesBegan:withEvent:**.

```
if ([t tapCount] > 1) {
    [self clearAll];
    return;
}
```

Bug. Líneas Misteriosas

Hay un error (bug) en la aplicación. Si pulsamos sobre una línea y luego empezamos a dibujar unos línea nueva, mientras que el menú es visible, se arrastra la línea seleccionada y se dibuja una nueva línea al mismo tiempo. (Habría que corregir este error).

UIGestureRecognizer - curiosidades

Sólo hemos visto superficialmente **UIGestureRecognizer.**

Hay más subclases, más propiedades y más métodos de delegado , incluso podemos crear reconocedores propios.

Cuando un reconocedor de movimientos está sobre una vista, en realidad está manejando todos los métodos **UIResponder** , como por ejemplo **touchesBegan:withEvent:** .

Los reconocedores de gesto son bastante codiciosos, así que por lo general no permiten a la vista recibir eventos de toque, o al menos retrasan la entrega de esos eventos.

Podemos establecer propiedades en el reconocedor de gestos , como **delaysTouchesBegan** , **delaysTouchesEnded** y **cancelsTouchesInView** , para cambiar este comportamiento . Si necesitamos un control más fino que este enfoque de todo o nada , se puede implementar métodos delegados para el reconocedor .

A veces, es posible que tengamos dos reconocedores de gesto buscando gestos muy similares. Podemos encadenar los reconocedores de manera que uno está obligado a fallar, para en el siguiente empezar a utilizar el método **requireGestureRecognizerToFail** : .

En total, hay siete estados en que un reconocedor puede entrar:

- UIGestureRecognizerStatePossible
- UIGestureRecognizerStateBegan
- UIGestureRecognizerStateChanged
- UIGestureRecognizerStateEnded
- UIGestureRecognizerStateFailed
- UIGestureRecognizerStateCancelled
- UIGestureRecognizerStateRecognized

La mayor parte del tiempo, un reconocedor se quedará en el estado “Possible” . Cuando un reconocedor reconoce su gesto, se entra en el estado “Began” . Si el gesto es algo que puede continuar , como un “tap” , se entra y permanece en el estado “Changed” hasta el final. Cuando ninguna de sus propiedades cambian , envía otro mensaje a su objetivo. Cuando el gesto termina (típicamente cuando el usuario levanta el dedo) , entra en el estado de “Ended” .

No todos los reconocedores realizan comienzo, cambio, y final. Para reconocedores de gesto que recogen en un gesto discreto como un “tap”, sólo veremos el estado “Recognized” (que tiene el mismo valor que el estado final “Ended”) .

Por último , un reconocedor se puede cancelar “Cancelled” (por una llamada telefónica entrante , por ejemplo) o por un fallo (mal gesto de los dedos) . Cuando se transiciona a este estado, se envía el mensaje de acción del reconocedor , y la propiedad “state” puede ser revisado para ver por qué.