

# CSE 438: Mobile Application Development

## Lab 3: Drawing App



### Overview

In this lab, you will be exploring UIKit and Core Graphics through the creation of a simple drawing app. This will be the first complete app you create, and will show how to use the Swift language to get the most out of the built-in Cocoa Touch frameworks.

### Details

**Due date:** Wednesday, October 4th, 11:59pm

**Grading:** This lab is out of 100 points total. The exact point distribution is described in the “Requirements” section below.

**Submission:** Zip the entire project folder and email it to [cse438ta@gmail.com](mailto:cse438ta@gmail.com). Please name the file “FirstNameLastName-Lab3.zip” and include a brief summary of your creative portion in the email body.

### Description

This lab requires you to create your own app from scratch. Although many of the implementation details are left to you, there are some helpful guidelines and code snippets to get you started in the “Helpful Advice and Code Snippets” section below. Many of the topics covered in this lab have been discussed in lecture as well. Additional resources are available online in the Swift documentation and UIKit documentation that Apple provides.

### Requirements

[40 points] Users can draw continuous lines by tapping and dragging.

[10 points] Lines drawn must be smooth (no jagged edges).

[5 points] Users can “undo” to erase the previous line drawn. They should be able to “undo” repeatedly until there are no lines on the screen.

[5 points] Users can selected from multiple (at least 5) color options.

[5 points] There is a slider to adjust the line thickness.

[10 points] Single taps result in dots.

[5 points] Users can easily erase all lines on the page.

[20 points] Creative portion: Add 2 other small features. Be creative!

## Helpful Advice and Code Snippets

The first challenge will most likely be getting data from the user's input. The way to do this is by overriding the `touchesBegan`, `touchesMoved`, and `touchesEnded` functions in your view controller. After these are in place, use the following code to get the location of a user's touch.

```
guard let touchPoint = touches.first?.location(in: view) else { return }
```

This function takes the first touch from a set of `UITouch` objects provided by the system, and finds its location in the current view. This limits the user to only inputting a single touch at a time, but it should suffice for the purposes of this lab.

An easy way to implement custom drawing is to override the `drawRect` function of a custom `UIView`. Create a new class which subclasses `UIView`, and override the `drawRect` function. This is where the custom drawing will take place, and is called automatically. If you need to refresh what is displayed in the view, the `setNeedsDisplay` function will call `drawRect` for you. Do not call `drawRect` directly. This allows the graphics system to optimize how often the screen is redrawn, and it will only execute `drawRect` if the display needs to be updated for the next frame to be displayed.

As for the lines themselves, I recommend using a `UIBezierPath` object, which has a lot of convenient methods to incrementally build and store complex paths. It is a higher-level wrapper object for the underlying `CGPath` object, and integrates with `UIKit` easily. It is quite straightforward to draw a `UIBezierPath` in a `UIView` (search for the `UIBezierPath` class reference online to find out how). If you are digging into the depths of Core Graphics, you have probably missed an easier solution.

The `UIBezierPath` class also comes in handy for smoothing out your lines (which otherwise may look jagged and unpolished). There are many path smoothing algorithms, but one of the simplest is given below. It uses the original points as control points for the bezier curve, and the midpoints between each pair of points as the actual points. (If you have worked with the pen tool in any vector drawing application the concept of anchor points and control points might be familiar to you.) In any case, below is a function which takes in an array of `CGPoint`s, and returns a smooth `UIBezierPath`. It requires the use of a midpoint helper function, which I will leave as an exercise for the reader.

```

private func midpoint(first: CGPoint, second: CGPoint) -> CGPoint {
    // implement this function here
}

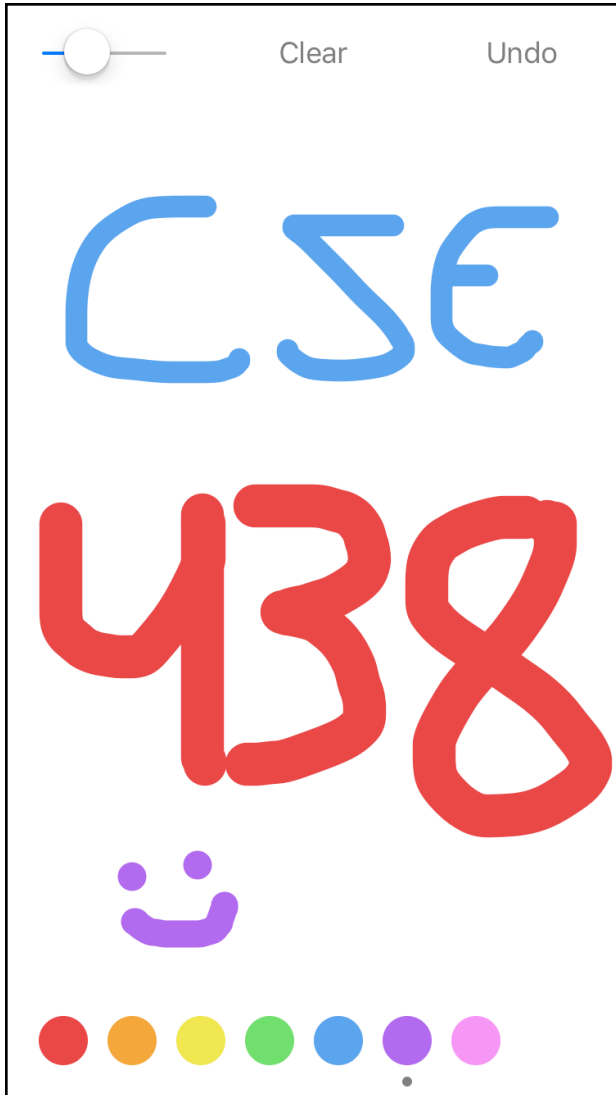
func createQuadPath(points: [CGPoint]) -> UIBezierPath {
    let path = UIBezierPath()
    if points.count < 2 { return path }
    let firstPoint = points[0]
    let secondPoint = points[1]
    let firstMidpoint = midpoint(first: firstPoint, second: secondPoint)
    path.move(to: firstPoint)
    path.addLine(to: firstMidpoint)
    for index in 1 ..< points.count-1 {
        let currentPoint = points[index]
        let nextPoint = points[index + 1]
        let midPoint = midpoint(first: currentPoint, second: nextPoint)
        path.addQuadCurve(to: midPoint, controlPoint: currentPoint)
    }
    guard let lastLocation = points.last else { return path }
    path.addLine(to: lastLocation)
    return path
}

```

Lastly, this application provides a great opportunity to leverage object-oriented programming principles to make your code easier to read and maintain, as well as save you time. If you are going to make a button for seven different colors, consider creating a UIButton subclass so you only have to write your button styling code once. Or, if you need to store additional information along with a UIBezierPath (like its color or thickness, for example), consider creating your own custom class/struct to keep all that relevant data together.

Think about structuring your app according to the model-view-controller paradigm. What is the model? What is the view? How do we connect those pieces in a reusable way? Which object should be responsible for the createQuadPath function? Contemplating these questions may seem trivial for a simple app, but becomes critical as your codebase scales.

Good luck!



An example of a working drawing app. Notice that the lines are smooth (not just connected line segments) and the user can tap to draw dots.