The note you're looking for was deleted.

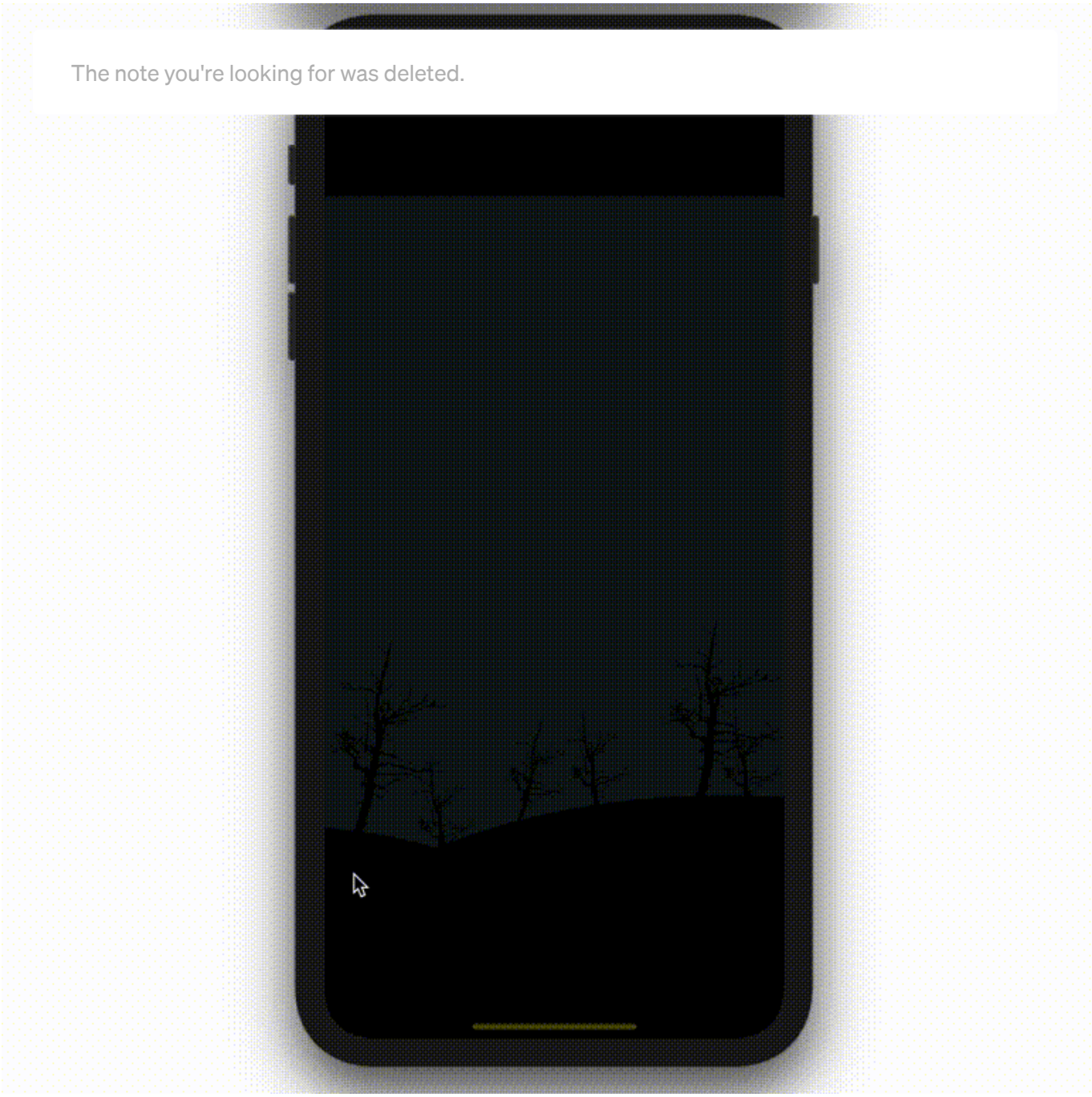
CODEX

How to Generate Lightning in Swift [Recursively]

Generate realistic-looking lightning strikes in Swift

Artturi Jalli · Follow

Published in CodeX · 7 min read · Jan 16, 2021



Learn how to generate realistic lightning in Swift.

Disclaimer: This post contains affiliate links.

The Setup

To get started, create a Game App in Xcode. Remove `Actions.sks` file and remove the hello label from `GameScene.sks`. Then erase excess code in

`GameScene.swift` file to make it look like this:

The note you're looking for was deleted.

```
import SpriteKit
import GameplayKit

class GameScene: SKScene {
    override func didMove(to view: SKView) {
        self.backgroundColor = .black
    }
}
```

Drawing a Single Line

A lightning strike consists of small paths or line segments that are connected to form the lightning strike.

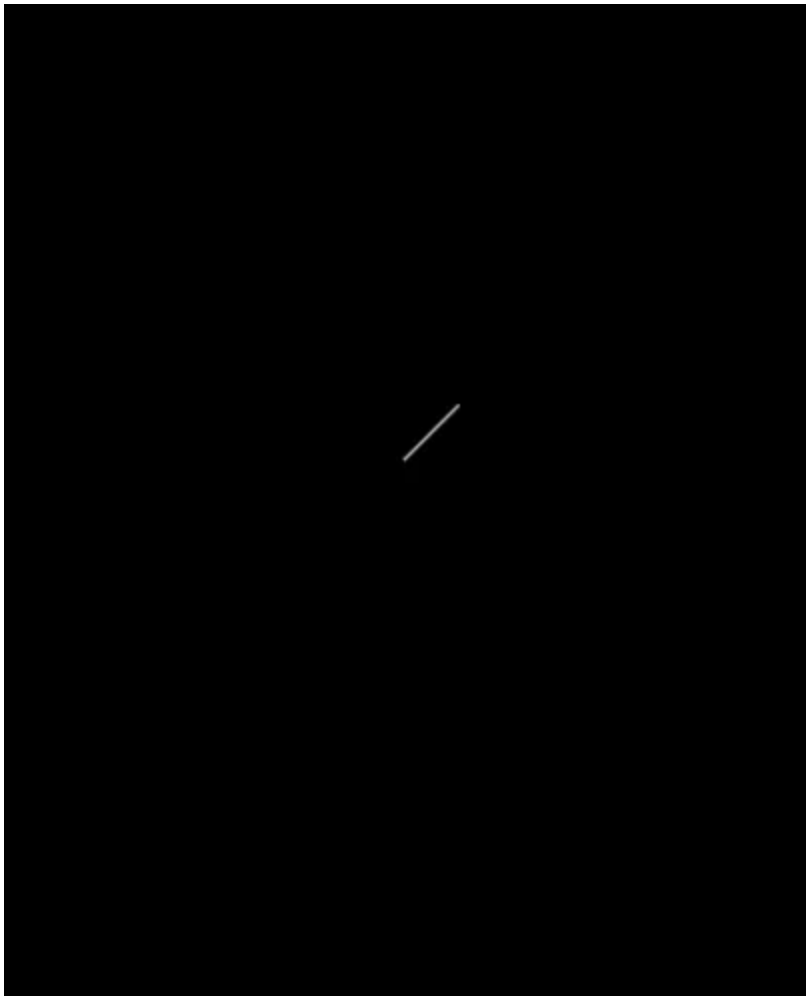
First off, we want to be able to draw a line from point A to point B. The following code snippet will return a line:

```
func createLine(pointA: CGPoint, pointB: CGPoint) -> SKShapeNode {
    let pathToDraw = CGMutablePath()
    pathToDraw.move(to: pointA)
    pathToDraw.addLine(to: pointB)
    let line = SKShapeNode()
    line.path = pathToDraw
    line.glowWidth = 1
    line.strokeColor = .white
    return line
}
```

Note that this function returns a shape node. It does not automatically add the line segment to the screen. You can add a line to the scene by adding the following to the `didMove` function:

The note you're looking for was deleted.

```
self.addChild(line)
```



A line from point (0, 0) to point (50, 50)

Feel free to tweak `x` and `y` to draw different kinds of lines.

Connecting the Lines

Next up, to create a lightning strike, we need to connect the lines. This can be done by generating the path for lightning by utilizing the above `createLine` method:

The note you're looking for was deleted.

```
    var startPoint = startingFrom
    var endPoint = startPoint
    let numberOfLines = 120

    var idx = 0
    while idx < numberOfLines {
        strikePath.append(createLine(pointA: startPoint, pointB:
endPoint))
        startPoint = endPoint
        let r = CGFloat(10)
        endPoint.y -= r

        idx += 1
    }
    return strikePath
}
```

In the above, we create an array of lines by looping in such a way that the previous line's endpoint is the starting point for the next one. This way the lightning strike is a continuous line.

To draw a set of lines let's create a function called `lightningStirke` that draws a lightning strike to the scene based on a path input to it:

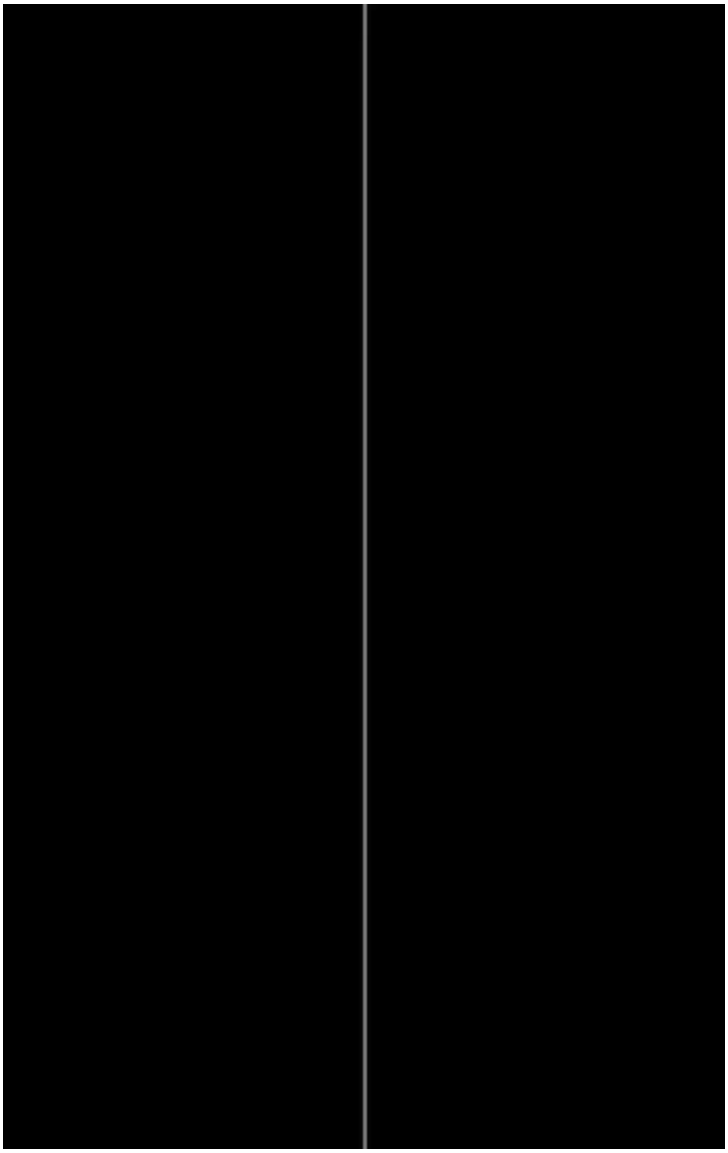
```
func lightningStrike(throughPath: [SKShapeNode]) {
    for line in throughPath {
        self.addChild(line)
    }
}
```

Now, in `didMove` function add a lightning strike to the scene by first generating a path for the strike and then lighting up the path by calling the `lightningStrike`:

The note you're looking for was deleted.

```
lightningStrike(throughPath: path)
```

We now see that the lightning strike is a long line from the top to the bottom.

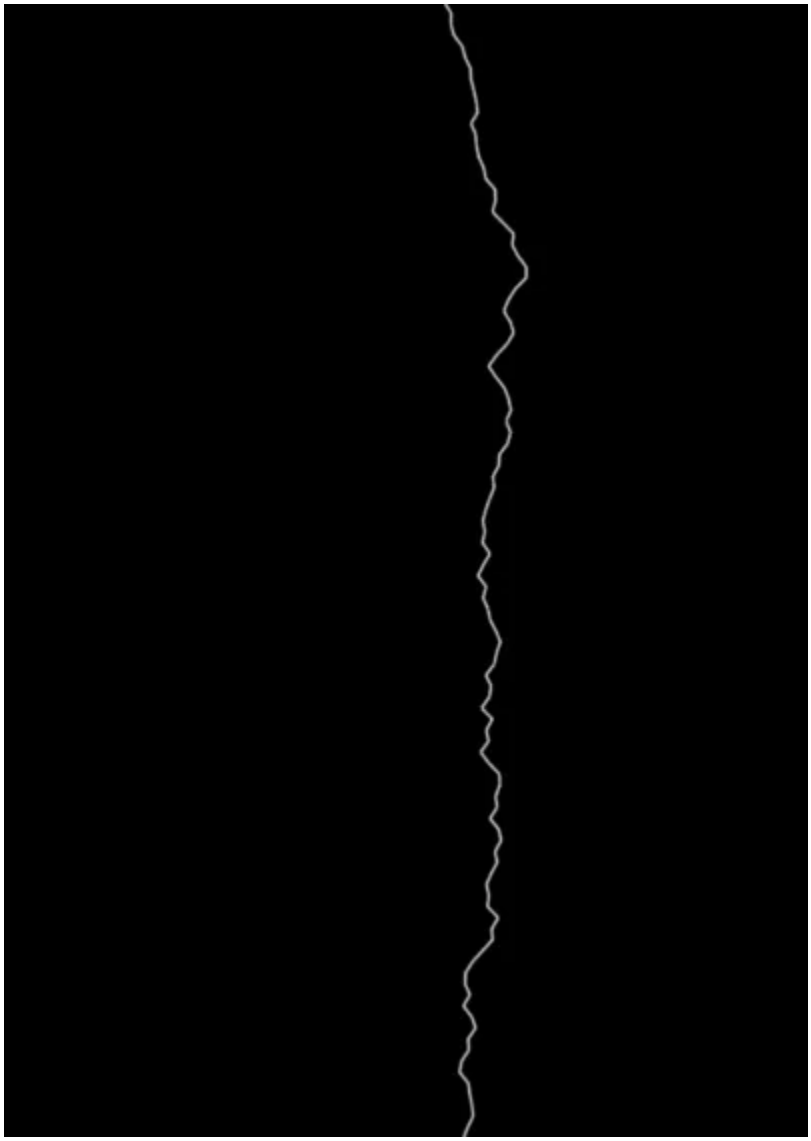


We need to add some randomness to the lines constructing the strike to make it look more realistic. This can be done by randomizing the x position `endPoint` for each line. Update `generateLightningPath` function by adding

The note you're looking for was deleted.

after `endPoint.y -= r`.

Result:



Fading Out the Strike

With the current implementation, the lightning strike stays in the scene forever. But we want it to disappear in a split second. To do this, run a

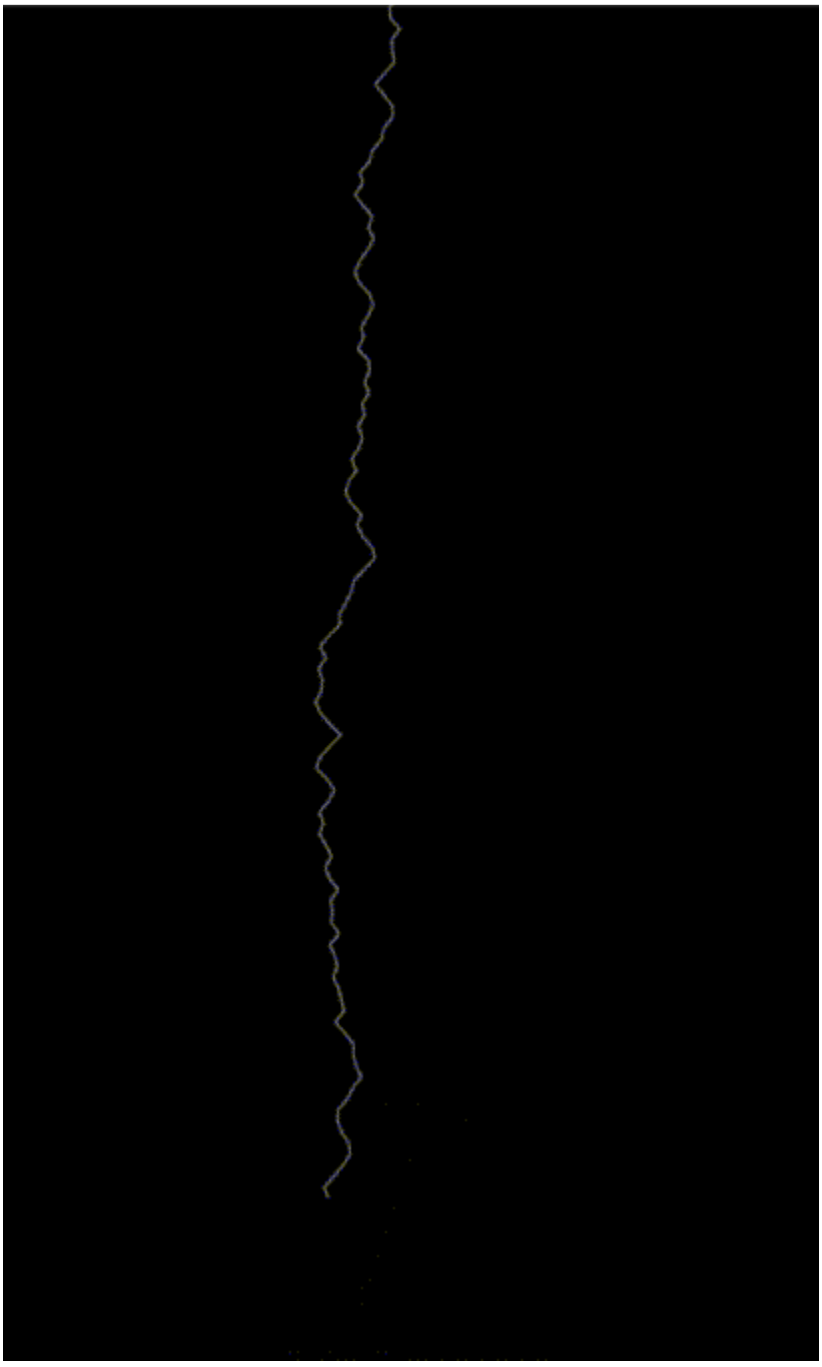
`fadeAlpha` action on the lightning strike:

The note you're looking for was deleted.

```
func lightningStrike(throughPath: [SKShapeNode]) {
    let path = genrateLightningPath(startingFrom: CGPoint(x:
frame.midX, y: frame.size.height / 2))

    for line in throughPath {
        self.addChild(line)
        line.run(SKAction.fadeAlpha(to: 0, duration: 1))
    }
}
```

Now the lightning strike fades away 1 second after appearing.

The note you're looking for was deleted.



Let’s make it possible to trigger lightning strikes by touching the screen. We move the functionality from `didMove` to `touchesBegan` as follows:

```
override func touchesBegan(_ touches: Set<UITouch>, with event:
UIEvent?) {
```

```
    let path = genrateLightningPath(startingFrom: CGPoint(x:
```

The note you're looking for was deleted.

```
}
```

Flickering

We don't want the lightning strike to disappear, but we want it to flicker in the screen for a second before vanishing.

We create an action sequence for all of the lightning strike's lines. It repeatedly fades the lines in and out. Let's create the *action sequence* for the lightning strike's lines, by extending the `lightningStrike` function:

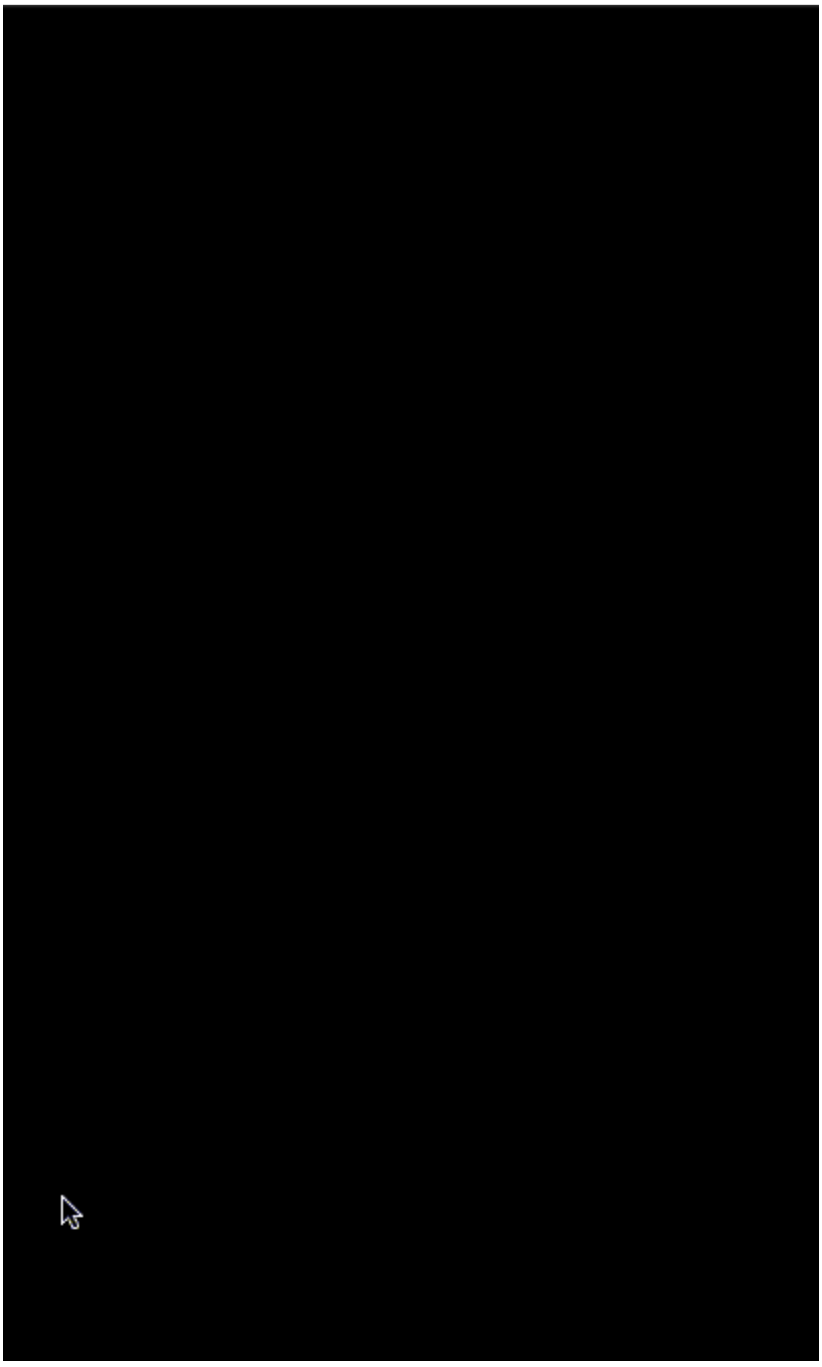
```
func lightningStrike(throughPath: [SKShapeNode], maxFlickeringTimes:
Int) {

    let fadeTime = TimeInterval(CGFloat.random(in: 0.005 ... 0.03))
    let waitAction = SKAction.wait(forDuration: 0.05)
    let reduceAlphaAction = SKAction.fadeAlpha(to: 0.0, duration:
fadeTime)
    let increaseAlphaAction = SKAction.fadeAlpha(to: 1.0, duration:
fadeTime)
    let flickerSeq = [waitAction, reduceAlphaAction,
increaseAlphaAction]

    var seq: [SKAction] = []
    let numberOfFlashes = Int.random(in: 1 ... maxFlickeringTimes)

    for _ in 1 ... numberOfFlashes {
        seq.append(contentsOf: flickerSeq)
    }

    for line in throughPath {
        seq.append(SKAction.fadeAlpha(to: 0, duration: 0.25))
        seq.append(SKAction.removeFromParent())
        line.run(SKAction.sequence(seq))
        self.addChild(line)
    }
}
```

Note the last `for` loop completes the flickering action sequence by fading ou

Note that we also added a new parameter `maxFlickeringTimes` to the function. This determines how many times the flickering sequence is run. Also, remember to modify the `lightningStrike` function call inside the `touchesBegan` function to pass in the new parameter by giving it some nice value, such as 5.

Result:

The note you're looking for was deleted.



Flickering the Background

Flashes of lightning light up the sky. In this case, we can achieve it by rapidly changing the color of the background. This is done with action sequences:

1. We light up the background

2 Wait for a split second

The note you're looking for was deleted.

3

4. Repeat 1–3 multiple times to achieve the flickering effect

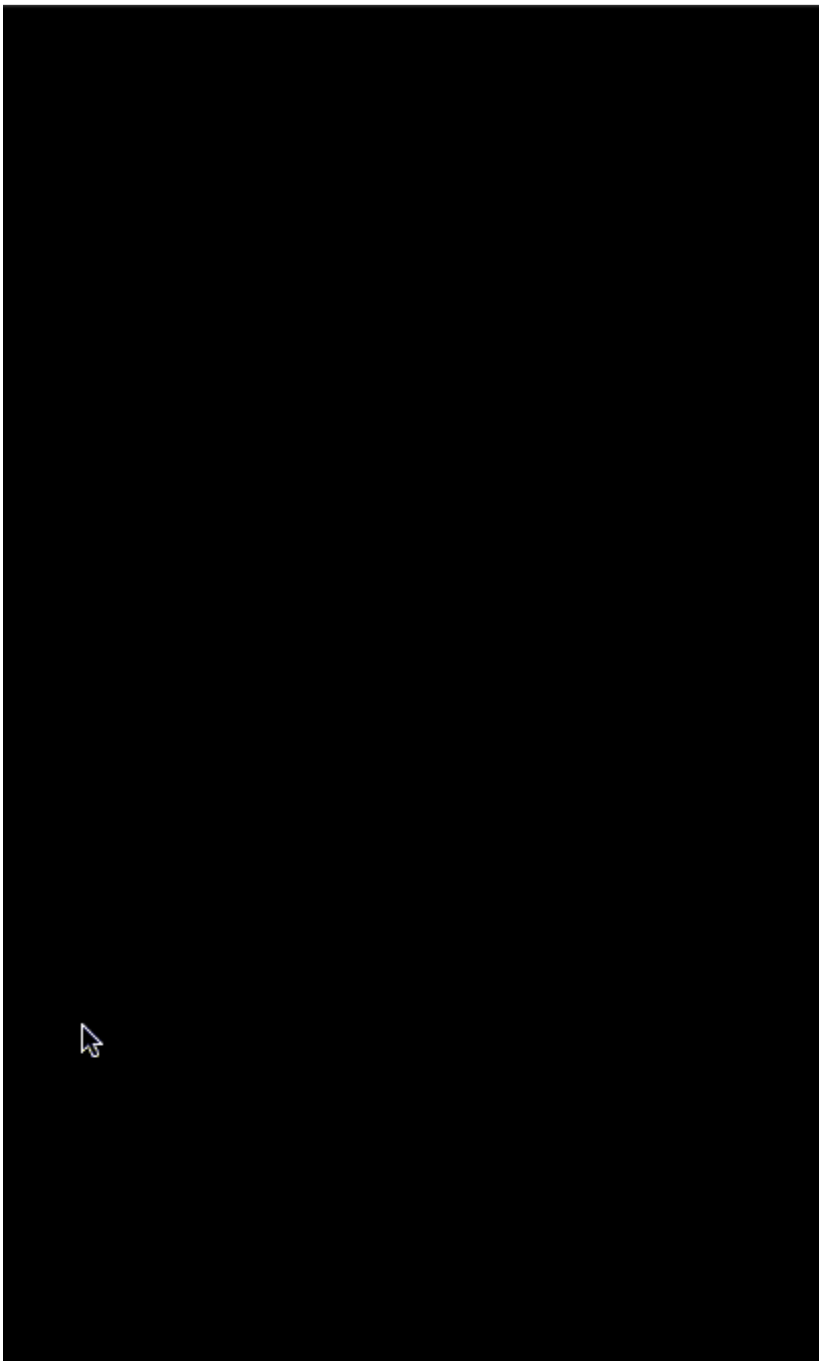
```
func flashTheScreen(nTimes: Int) {
    let lightUpScreenAction = SKAction.run { self.backgroundColor =
UIColor.gray }
    let waitAction = SKAction.wait(forDuration: 0.05)
    let dimScreenAction = SKAction.run { self.backgroundColor =
.black}

    var flashActionSeq: [SKAction] = []
    for _ in 1 ... nTimes + 1 {
        flashActionSeq.append(contentsOf: [lightUpScreenAction,
waitAction, dimScreenAction, waitAction])
    }
    self.run(SKAction.sequence(flashActionSeq))
}
```

Now call this function at the end of `lightningStrike` function to flicker the screen as the lightning strike occurs:

```
flashTheScreen(nTimes: numberOfFlashes)
```

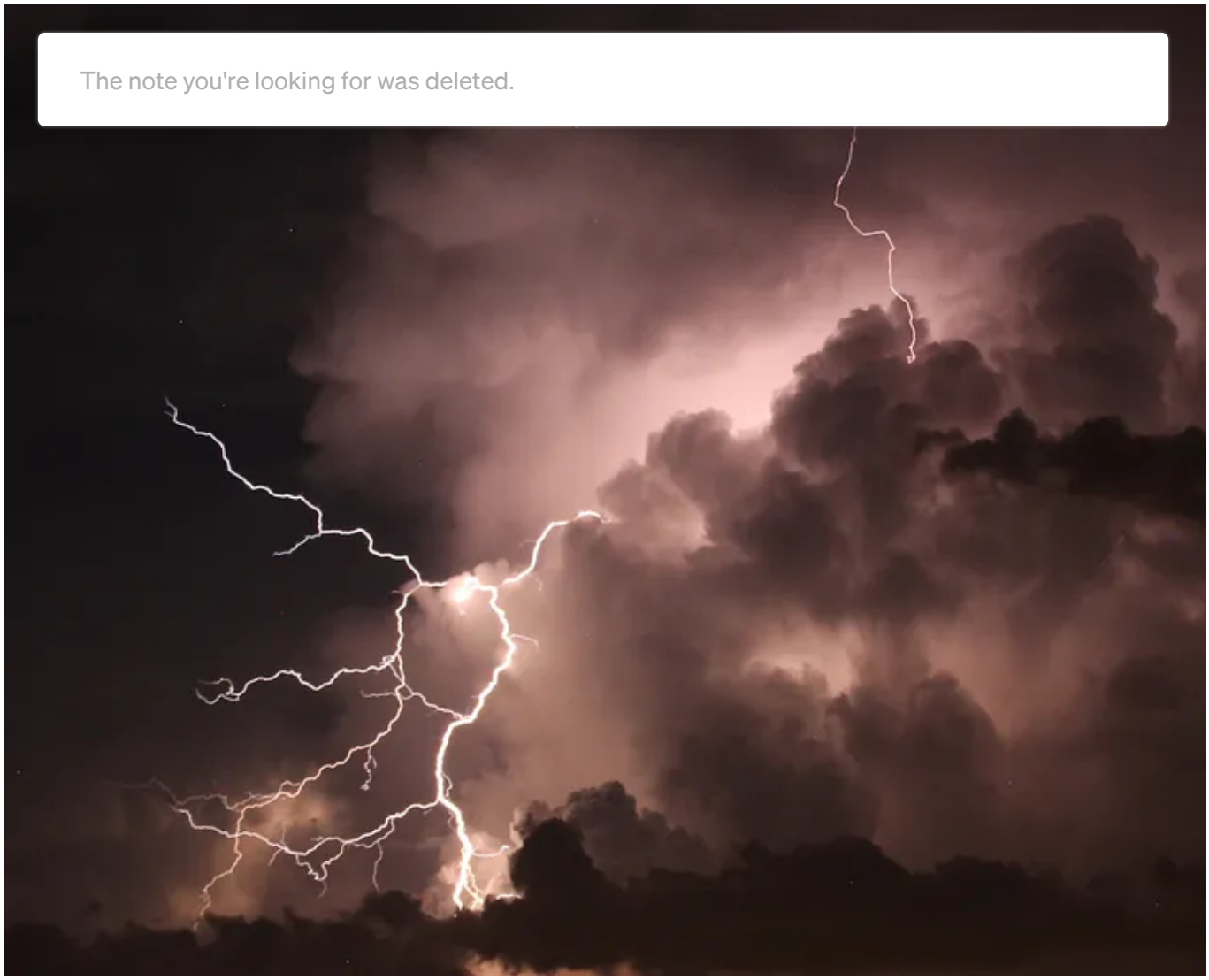
Result:

The note you're looking for was deleted.



Branching the Lightning Strike

The lightning strike is not always a single strike but it branches out too:



Controlling the Striking Angle

To make the lightning strike branch out, we need to be able to control the direction of the lightning strike.

To do this, add an `angle` parameter to `generateLightningPath` function:

```
func genrateLightningPath(startingFrom: CGPoint, angle: CGFloat) ->
[SKShapeNode] {

    ...

}
```

Let's modify the implementation of `generateLightningPath` by changing the wa The note you're looking for was deleted. es in `generateLightningPath`:

```
endPoint.y -= r
endPoint.x += CGFloat.random(in: -r ... r)
```

with


```
endPoint.y -= r * cos(angle) + CGFloat.random(in: -r ... r)
endPoint.x += r * sin(angle) + CGFloat.random(in: -r ... r)
```

Here we used trigonometry to determine the next position based on the `angle`.

Now the lightning strikes in the direction of the `angle` passed into the `generateLightningPath` function.

Recursion for Branching

For branching, we need to know if we are working with the main lightning path or if we are in a lightning strike's branch. For this, we extend the implementation of `generateLightningPath` by adding an `isBranch` boolean to the parameters of the function.

```
func genrateLightningPath(startingFrom: CGPoint, angle: CGFloat,
isBranch: Bool) -> [SKShapeNode] {

    ...
```

```
}
```

The note you're looking for was deleted.

Let's then make `numberOfLines` smaller in case we are working with a branch:

```
func genrateLightningPath(startingFrom: CGPoint, angle: CGFloat,
isBranch: Bool) -> [SKShapeNode] {
    let numberOfLines = isBranch ? 50 : 120
    ...
}
```

As we generate the lightning path in `generateLightningPath` by adding lines together we want that with a small chance the path creates a branch:

Right after the lines in `generateLightningPath`

```
endPoint.y -= r * cos(angle) + CGFloat.random(in: -10 ... 10)
endPoint.x += r * sin(angle) + CGFloat.random(in: -10 ... 10)
```

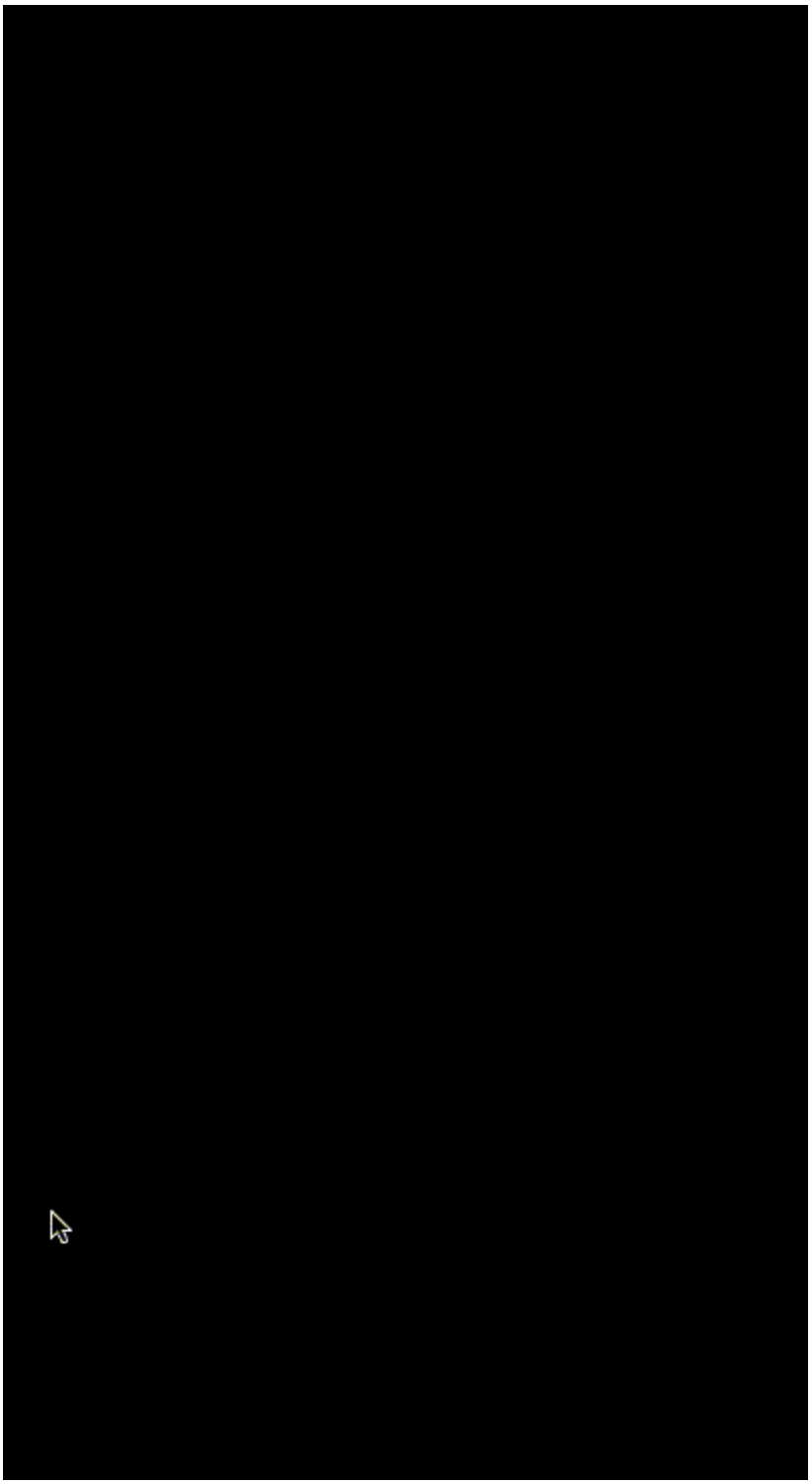
Add this piece of code:

```
if Int.random(in: 0 ... 100) == 1 {
    let branchingStartPoint = endPoint

    let branchingAngle = CGFloat.random(in: -CGFloat.pi / 4 ...
CGFloat.pi / 4) // the angle to make the branching look natural

    strikePath.append(contentsOf: genrateLightningPath(startingFrom:
branchingStartPoint, angle: branchingAngle, isBranch: true))
}
```

This part generates a branch from the lightning strike with a probability of

0. The note you're looking for was deleted.

Now run the app and enjoy the effect.



Feel free to copy the full implementation from my Github Repo. It includes a so The note you're looking for was deleted.

Conclusion

Thanks for reading. Happy coding!

Become a Genius

Did you like this article?

Become a member at Medium.com to read top stories by experts in the field.

Disclaimer: By joining via the provided link, I earn a small commission at no extra cost for you :)

More Insightful Stories

iOS Development Tips in 2021 | Code Like a Pro | Codingem

Adapt some super useful skills to your iOS development routine to significantly improve your efficiency and...

www.codingem.com

50 Swift Interview Questions and Answers for 2021

Ace your iOS developer job interview by knowing the answers to

The note you're looking for was deleted.

Swift — Upgrade Your Skills with These 8 Tips

1. Shorter If...Else Statements with Ternary Operators

medium.com

Programming | Ios Tutorial | Swift Tutorial | IOS Development | Swift

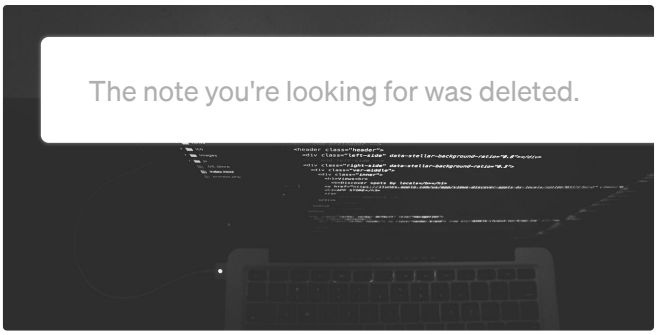
Written by Artturi Jalli

3.7K Followers · Writer for CodeX

Follow

Youtube: @jalliartturi—learn blogging without SEO

More from Artturi Jalli and CodeX



Artturi Jalli in Better Programming

8 Neat C++ Programming Tricks You Should Know

Master some of the crucial C++ concepts and save a few lines of code

✦ · 4 min read · Mar 24, 2021

318 6

Enigma of the Stack in CodeX

Why Are JavaScript Pros Saying Goodbye to TypeScript?

TypeScript's Triumph and the Unexpected Turn

✦ · 7 min read · Sep 12

653 43

Mochamad Kautzar Ichramsyah in CodeX

Automate the exploratory data analysis (EDA) to understand the...

What is EDA?

11 min read · Jul 11

1K 15

Artturi Jalli

Blogging Just Became Easier—Do This to Win

Forget about SEO and growth hacks!

✦ · 4 min read · Sep 11

79 4

The note you're looking for was deleted.

Recommended from Medium

Jakir Hossain

Dynamic list, Custom Views and NavigationStack in SwiftUI

In this article, we will learn how to use NavigationStack in SwiftUI and pass data to...

4 min read · 2 days ago

2

Bryn Bodayle in The Airbnb Tech Blog

Unlocking SwiftUI at Airbnb

How Airbnb adopted SwiftUI in our iOS app

10 min read · Sep 21

2K 19

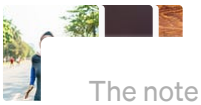
Lists

Staff Picks

457 stories · 305 saves

Stories to Help You Level-Up at Work

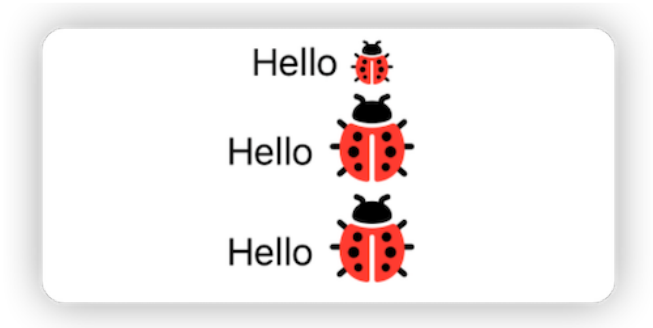
19 stories · 227 saves



Self-Improvement 101

The note you're looking for was deleted.

Productivity 101



fatbobman (东坡肘子)

Mixing Text and Image in SwiftUI

SwiftUI offers powerful layout capabilities, but these layout operations are performed...

14 min read · Apr 9

1

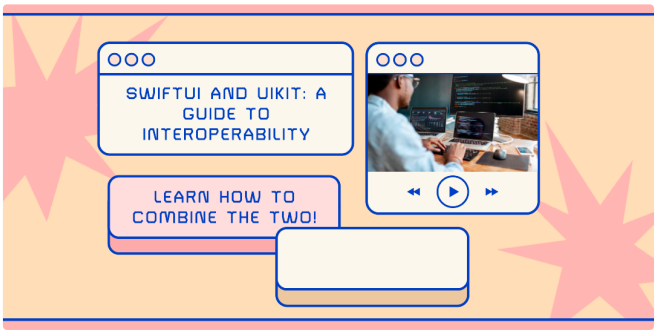
AsyncLearn

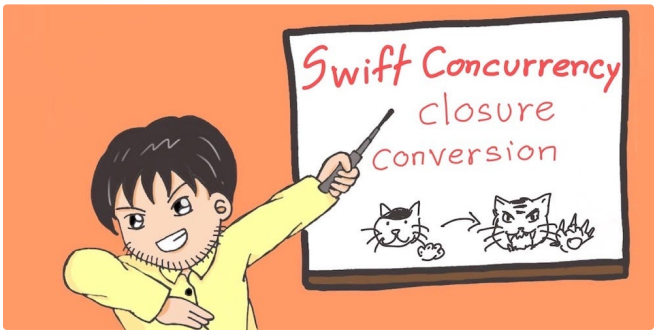
How to hide a TabBar in SwiftUI

If you want to read the Spanish version of this article, you can find it here...

2 min read · 4 days ago

8





Computer Science Diaries in Dev Genius

Combining SwiftUI and UIKit: A guide to interoperability

SwiftUI and UIKit are two powerful frameworks that enable you to create...

7 min read · Apr 21

Oudy1st

Swift Concurrency (closure-conversion)

When we start learning Swift Concurrency, the first thing that should come in mind is how w...

3 min read · 6 days ago

The note you're looking for was deleted.

See more recommendations