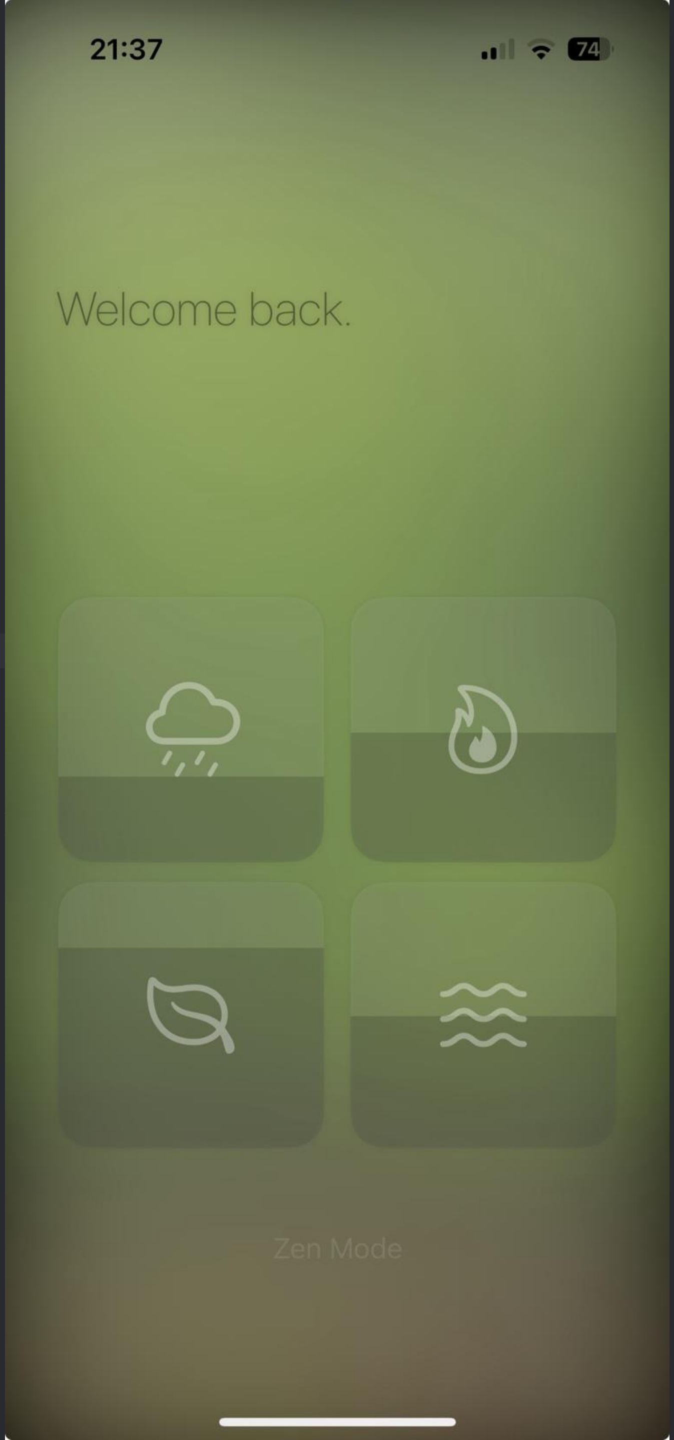
```
ct ContentView: View {
@StateObject var noiseRepo = NoiseRepo()
@State private var showMixer = true

var body: some View {
    ZStack {
        ZenView().environmentObject(noiseRepo)

        mixerView().environmentObject(noiseRepo)
            .opacity(showMixer ? 1 : 0)
            .animation(.smooth(duration: 0.4),value: showMixer)

        VStack{
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Spacer()
            Text("Zen Mode")
                .foregroundColor(.white.opacity(showMixer ? 0.1 : 0.32))
                .padding(30)
                .background(.clear)
                .contentShape(Rectangle())
                .onTapGesture {
                    showMixer.toggle()
                }
                .animation(.smooth(duration: 0.2),value: showMixer)

            Spacer()

        }
    }


}
```

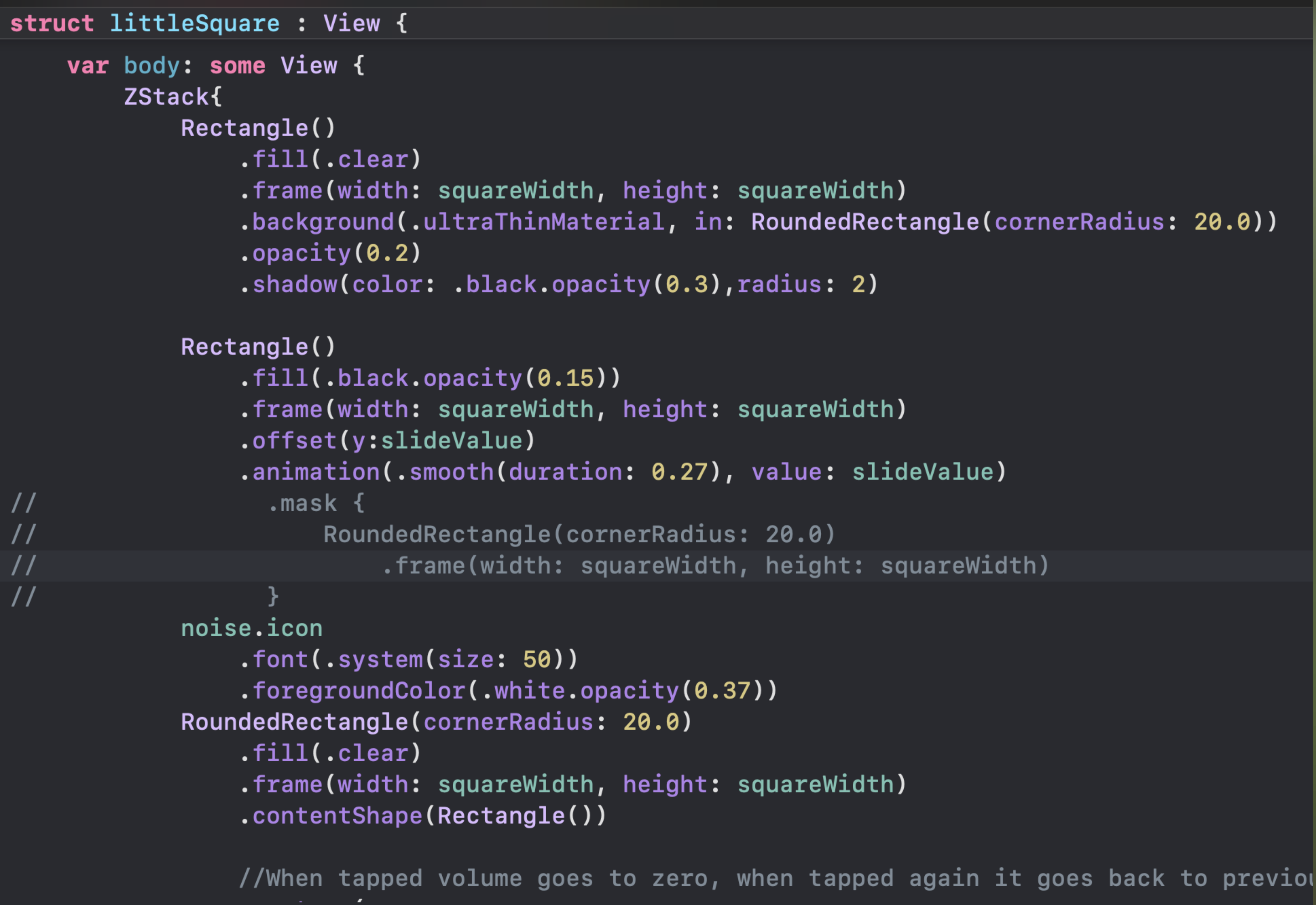


Master View (ContentView)

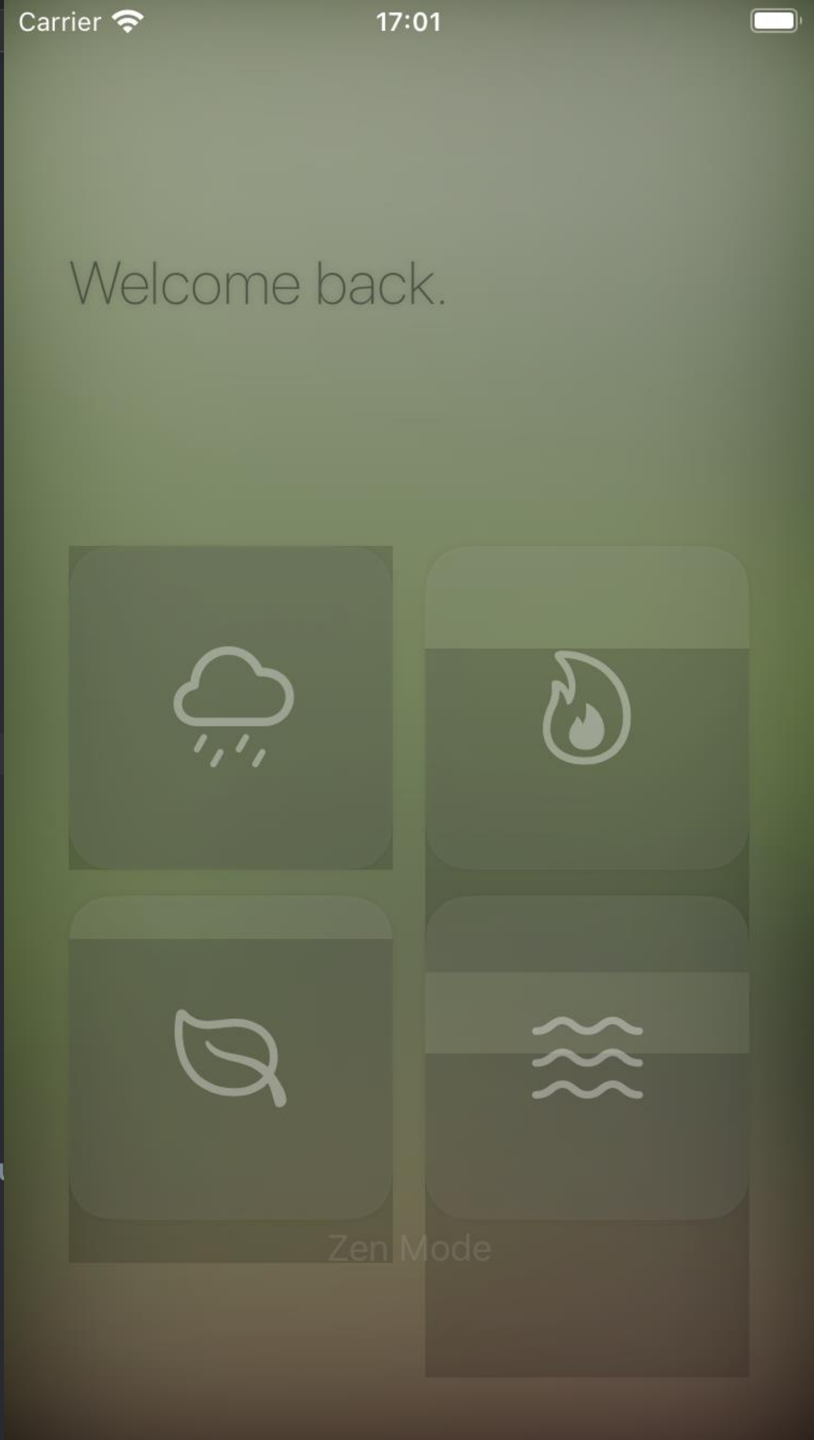
Zstack of mixer (mixerView), background(ZenView) and the Zen Mode button which can hide mixer.

Use EnvironmentObject() to pass noseRepo to the sub views.

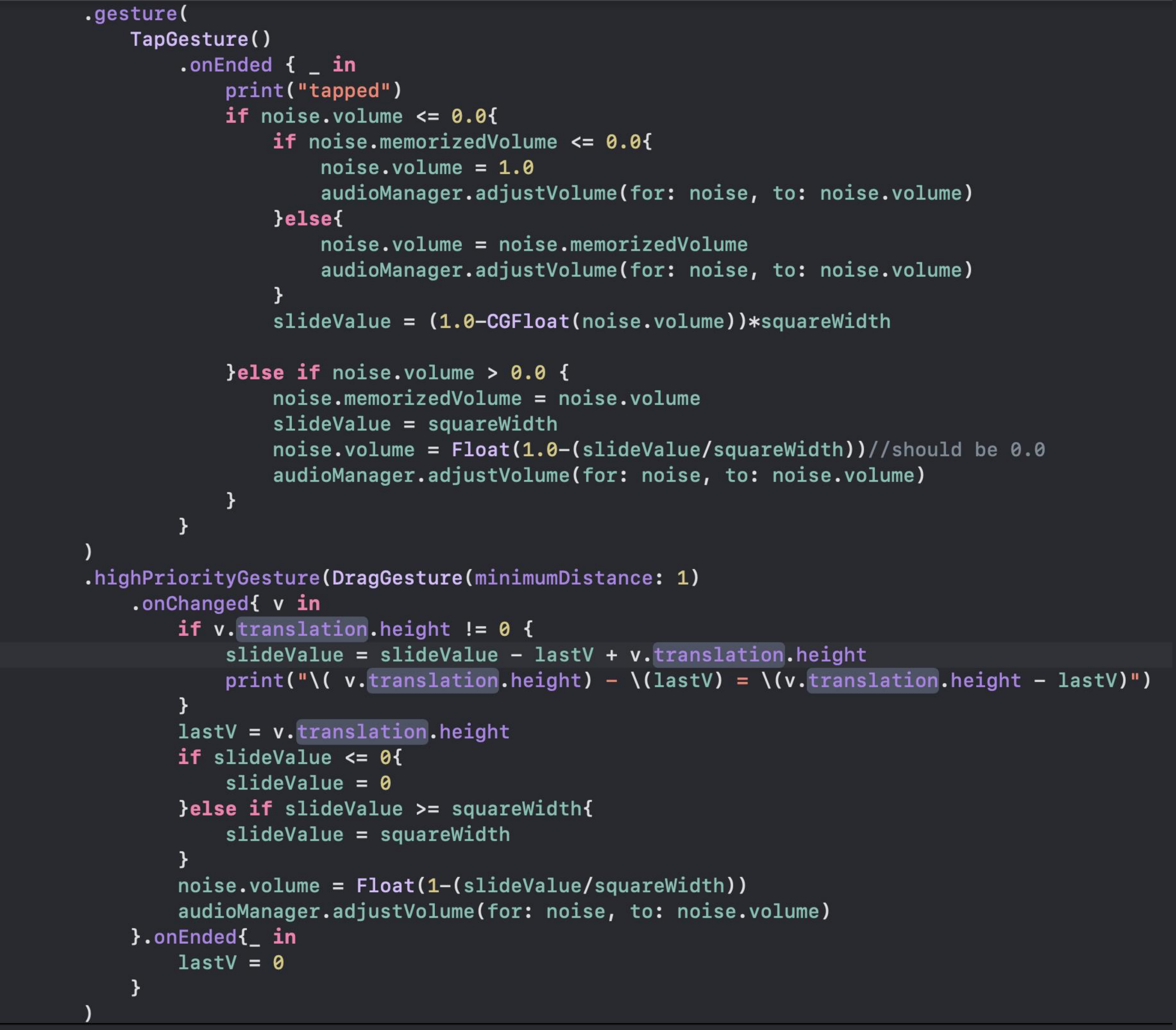
Spacer() to make layout responsive to the screen width and height.



```
struct littleSquare : View {
    var body: some View {
        ZStack{
            Rectangle()
                .fill(.clear)
                .frame(width: squareWidth, height: squareWidth)
                .background(.ultraThinMaterial, in: RoundedRectangle(cornerRadius: 20.0))
                .opacity(0.2)
                .shadow(color: .black.opacity(0.3),radius: 2)

            Rectangle()
                .fill(.black.opacity(0.15))
                .frame(width: squareWidth, height: squareWidth)
                .offset(y:slideValue)
                .animation(.smooth(duration: 0.27), value: slideValue)
//                 .mask {
//                     RoundedRectangle(cornerRadius: 20.0)
//                         .frame(width: squareWidth, height: squareWidth)
//                 }
            noise.icon
                .font(.system(size: 50))
                .foregroundColor(.white.opacity(0.37))
            RoundedRectangle(cornerRadius: 20.0)
                .fill(.clear)
                .frame(width: squareWidth, height: squareWidth)
                .contentShape(Rectangle())

                //When tapped volume goes to zero, when tapped again it goes back to previou
```



Each slider is a zstack of a background square, slide chunk, icon and a invisible square that take gesture input. Using “slideValue” to affect the offset of the sliding rectangle and transfer to volume.

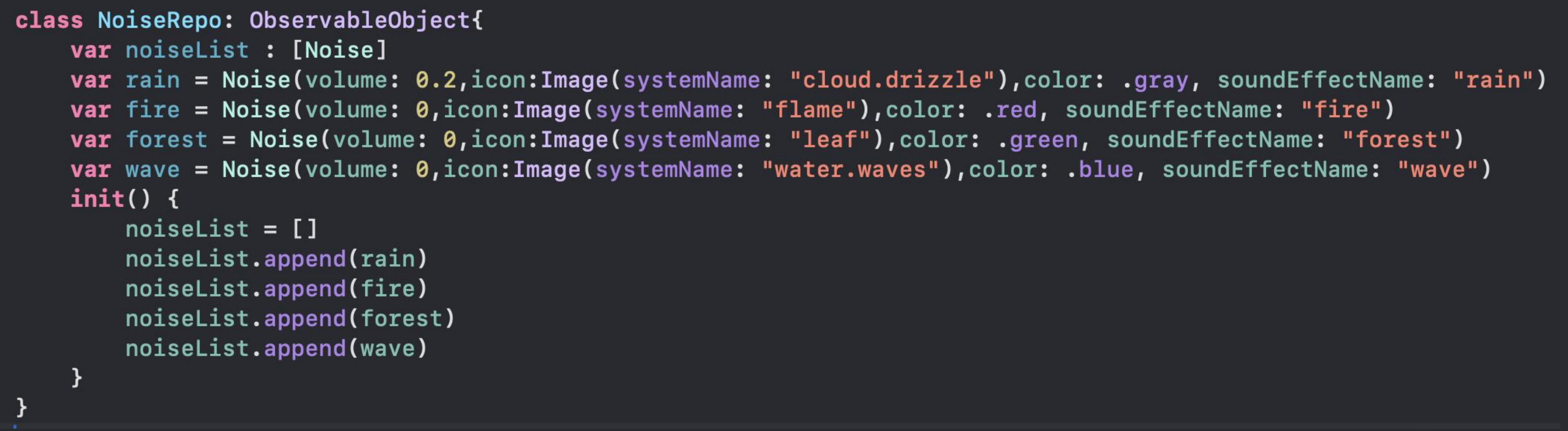


```swift
.gesture(
    TapGesture()
        .onEnded { _ in
            print("tapped")
            if noise.volume <= 0.0{
                if noise.memorizedVolume <= 0.0{
                    noise.volume = 1.0
                    audioManager.adjustVolume(for: noise, to: noise.volume)
                }else{
                    noise.volume = noise.memorizedVolume
                    audioManager.adjustVolume(for: noise, to: noise.volume)
                }
                slideValue = (1.0-CGFloat(noise.volume))*squareWidth

            }else if noise.volume > 0.0 {
                noise.memorizedVolume = noise.volume
                slideValue = squareWidth
                noise.volume = Float(1.0-(slideValue/squareWidth))//should be 0.0
                audioManager.adjustVolume(for: noise, to: noise.volume)
            }
        }
)
.highPriorityGesture(DragGesture(minimumDistance: 1)
    .onChanged{ v in
        if v.translation.height != 0 {
            slideValue = slideValue - lastV + v.translation.height
            print("\( v.translation.height) - \(lastV) = \(v.translation.height - lastV)")
        }
        lastV = v.translation.height
        if slideValue <= 0{
            slideValue = 0
        }else if slideValue >= squareWidth{
            slideValue = squareWidth
        }
        noise.volume = Float(1-(slideValue/squareWidth))
        audioManager.adjustVolume(for: noise, to: noise.volume)
    }.onEnded{_ in
        lastV = 0
    }
)
```

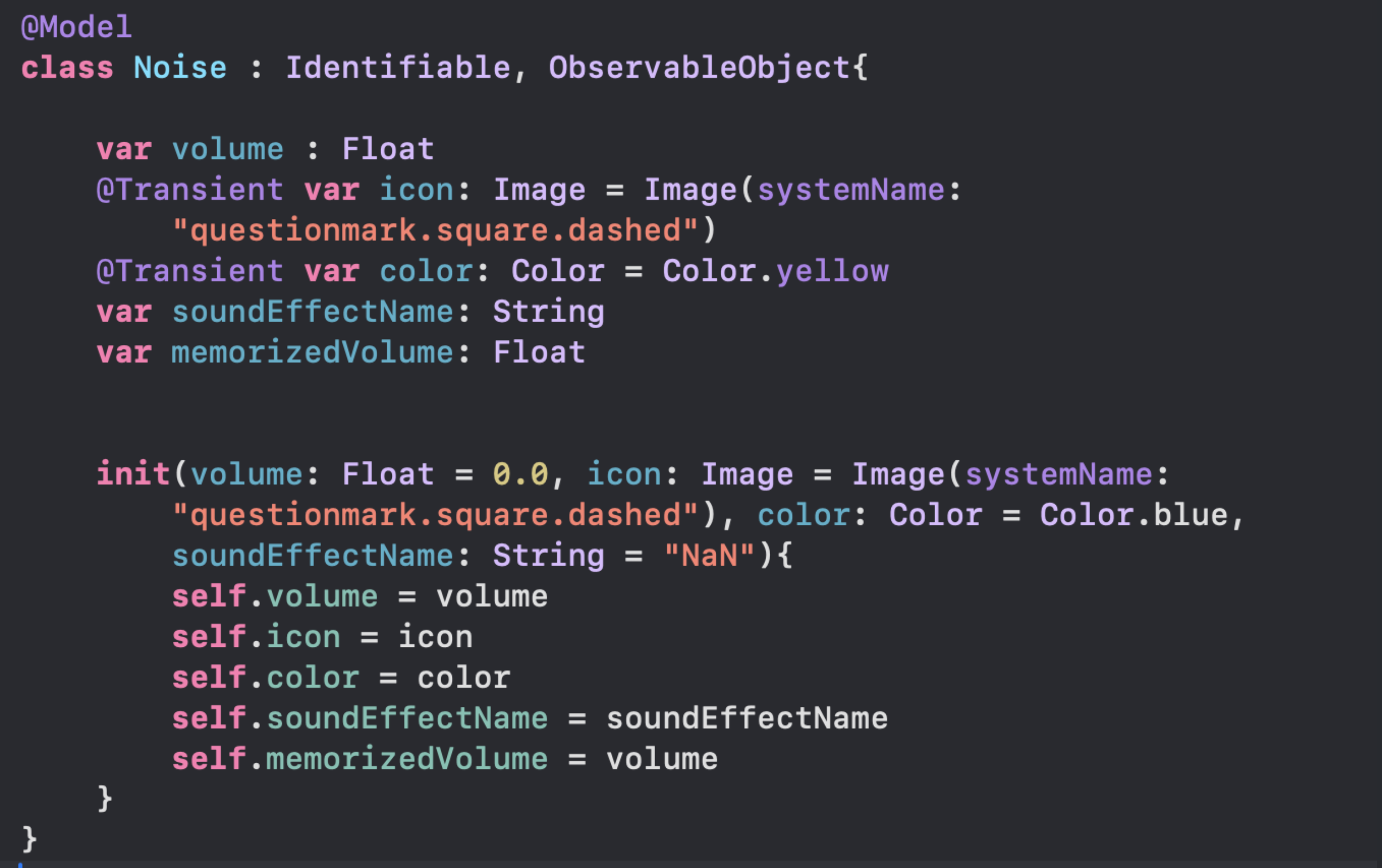
- By tapping the square, it memorized the current volume, then adjust the volume to zero. Tapping again will bring the slide and the volume back to the last position/volume

- adjusting the volume by dragging the silder.

Noise Model



```
class NoiseRepo: ObservableObject{
    var noiseList : [Noise]
    var rain = Noise(volume: 0.2,icon:Image(systemName: "cloud.drizzle"),color: .gray, soundEffectName: "rain")
    var fire = Noise(volume: 0,icon:Image(systemName: "flame"),color: .red, soundEffectName: "fire")
    var forest = Noise(volume: 0,icon:Image(systemName: "leaf"),color: .green, soundEffectName: "forest")
    var wave = Noise(volume: 0,icon:Image(systemName: "water.waves"),color: .blue, soundEffectName: "wave")
    init() {
        noiseList = []
        noiseList.append(rain)
        noiseList.append(fire)
        noiseList.append(forest)
        noiseList.append(wave)
    }
}
```



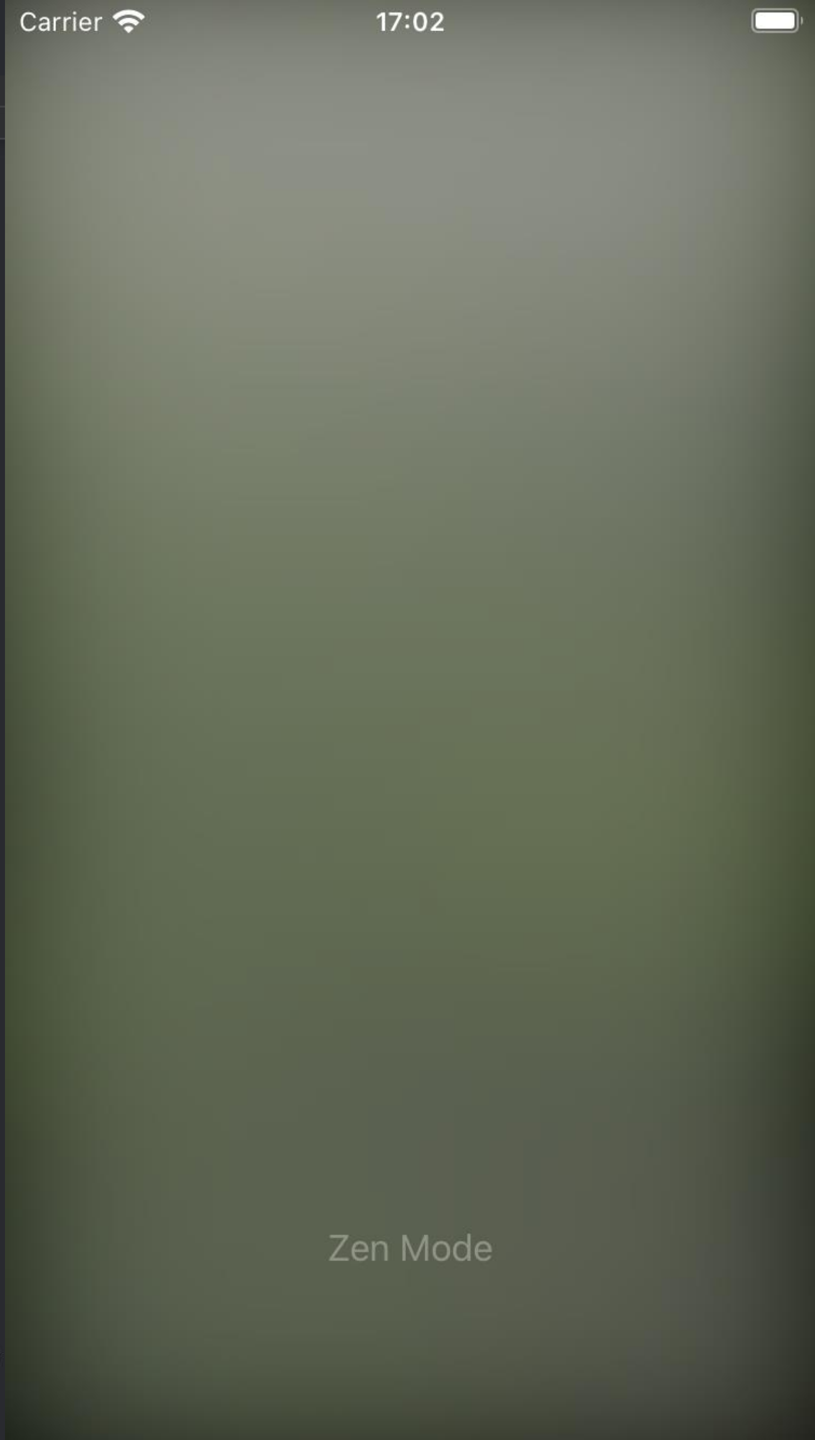
```
@Model
class Noise : Identifiable, ObservableObject{

    var volume : Float
    @Transient var icon: Image = Image(systemName:
        "questionmark.square.dashed")
    @Transient var color: Color = Color.yellow
    var soundEffectName: String
    var memorizedVolume: Float


    init(volume: Float = 0.0, icon: Image = Image(systemName:
        "questionmark.square.dashed"), color: Color = Color.blue,
        soundEffectName: String = "NaN"){
        self.volume = volume
        self.icon = icon
        self.color = color
        self.soundEffectName = soundEffectName
        self.memorizedVolume = volume
    }
}
```

Metal Wave Effect: Pixels with Sin Wave Behavior.

```
#include <metal_stdlib>
#include <SwiftUI/SwiftUI_Metal.h>
using namespace metal;

[[stitchable]] float2 wave(float2 position,float time,float speed,
    float frequency, float amplitude){
    float positionY = position.y + sin((time * speed) + (position.x /
        frequency)) * amplitude;
    return float2(position.x, positionY);
}
```

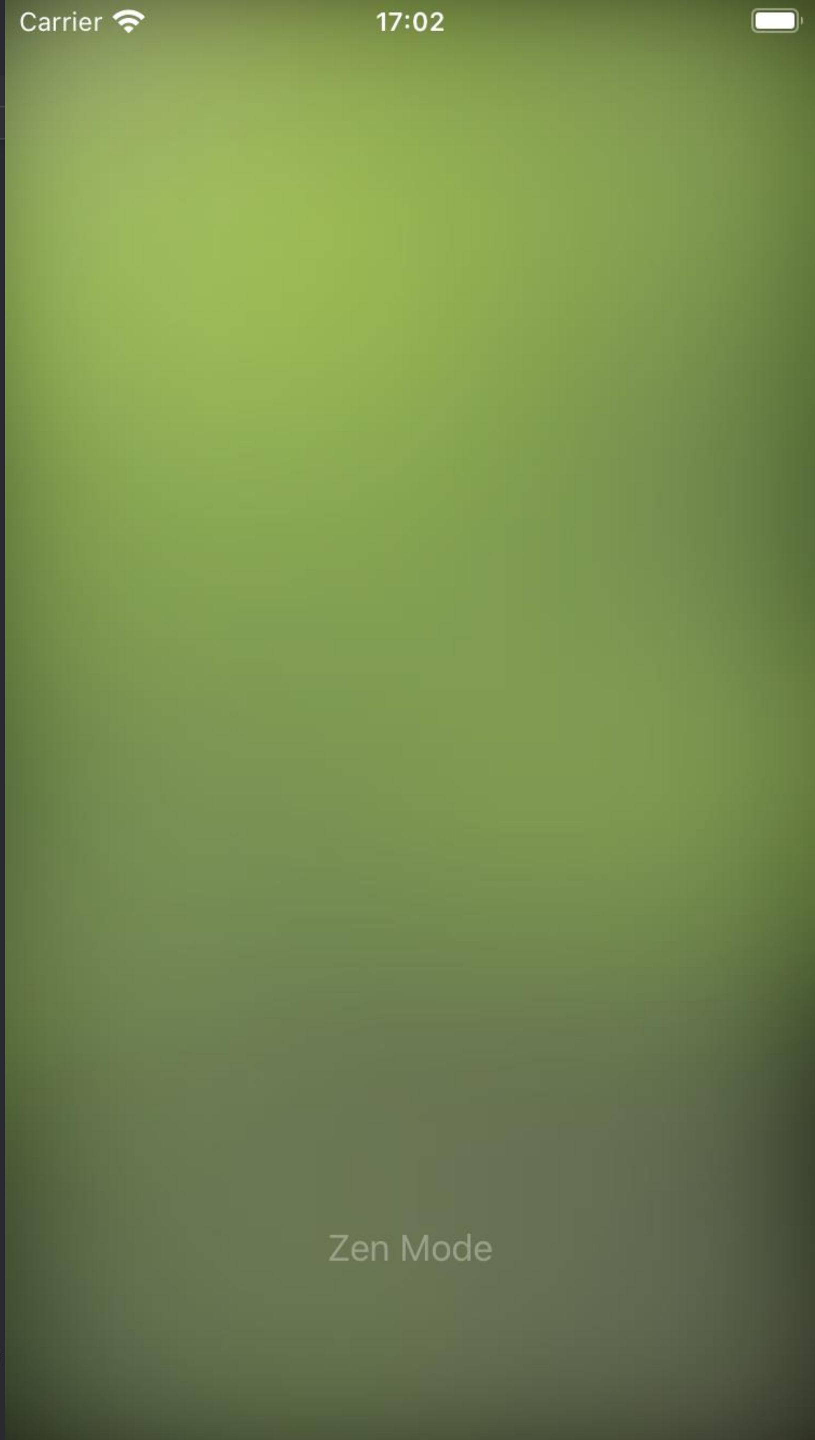
Dynamic Background

```swift
struct ZenView: View {
    func waveBackground() -> some View{
    }
    var body: some View {
        ZStack{
            ZStack {
                Image("background")
                    .resizable()
                    .scaledToFill()
                    .frame(width: UIScreen.main.bounds.width, height: UIScreen.main.bounds.height, alignment: .center)
                    .opacity(0.3)
                waveBackground()
                VStack{
                    RainRectangle()
                        .frame(height: UIScreen.main.bounds.height * 3/4)
                                    .edgesIgnoringSafeArea(.top)
                        .opacity(Double(self.noiseRepo.rain.volume))
                        .animation(.smooth(duration: 3), value: noiseRepo.rain.volume)
                    Spacer()
                }
                VStack{
                    Spacer()
                    FireRectangle()
                        .frame(height: UIScreen.main.bounds.height * 2/5)
                        .opacity(Double(self.noiseRepo.fire.volume))
                        .animation(.smooth(duration: 3), value: noiseRepo.fire.volume)
                }
            }
            VisualEffectView(effect: UIBlurEffect(style: .light))
        }
        .blur(radius: 40)
        .ignoresSafeArea()
        .background(.black)
    }


    struct VisualEffectView: UIViewRepresentable {
        var effect: UIVisualEffect?
        func makeUIView(context: UIViewRepresentableContext<Self>) -> UIVisualEffectView { UIVisualEffectView() }
        func updateUIView(_ uiView: UIVisualEffectView, context: UIViewRepresentableContext<Self>) { uiView.effect = effect }
    }
}
```

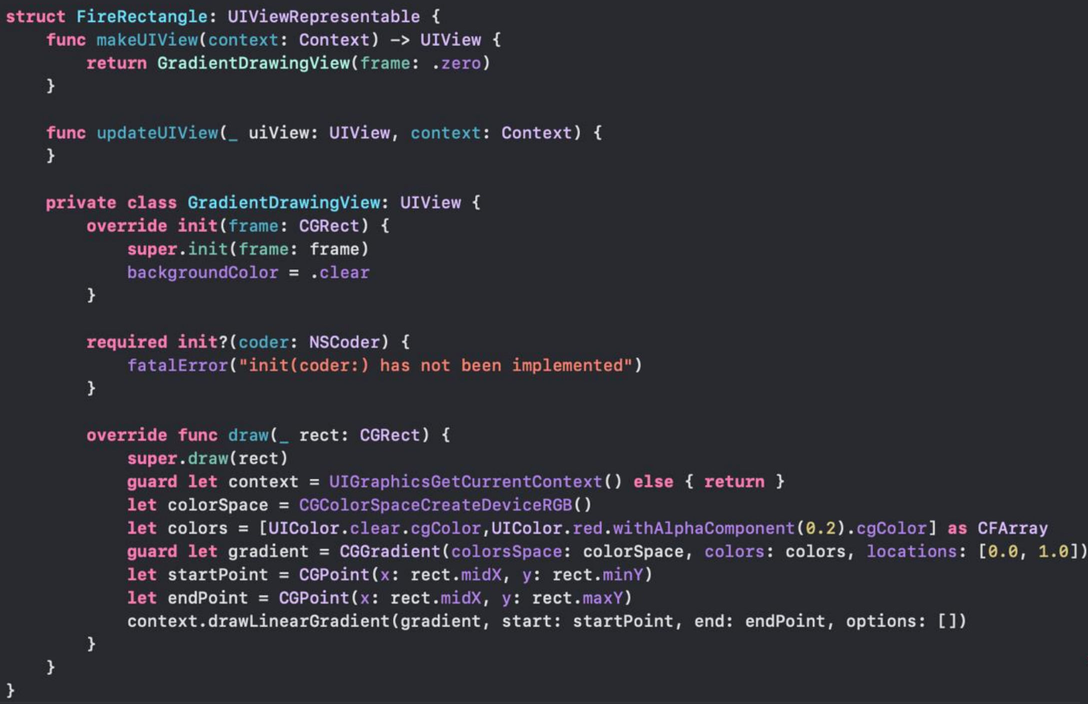


Dynamic Background

```
struct ZenView: View {
    func waveBackground() -> some View{
    }
    var body: some View {
        ZStack{
            ZStack {
                Image("background")
                    .resizable()
                    .scaledToFill()
                    .frame(width: UIScreen.main.bounds.width, height: UIScreen.main.bounds.height, alignment: .center)
                    .opacity(0.3)
                waveBackground()
                VStack{
                    RainRectangle()
                        .frame(height: UIScreen.main.bounds.height * 3/4)
                                    .edgesIgnoringSafeArea(.top)
                        .opacity(Double(self.noiseRepo.rain.volume))
                        .animation(.smooth(duration: 3), value: noiseRepo.rain.volume)
                    Spacer()
                }
                VStack{
                    Spacer()
                    FireRectangle()
                        .frame(height: UIScreen.main.bounds.height * 2/5)
                        .opacity(Double(self.noiseRepo.fire.volume))
                        .animation(.smooth(duration: 3), value: noiseRepo.fire.volume)
                }
            }
            VisualEffectView(effect: UIBlurEffect(style: .light))
        }
        .blur(radius: 40)
        .ignoresSafeArea()
        .background(.black)
    }


    struct VisualEffectView: UIViewRepresentable {
        var effect: UIVisualEffect?
        func makeUIView(context: UIViewRepresentableContext<Self>) -> UIVisualEffectView { UIVisualEffectView() }
        func updateUIView(_ uiView: UIVisualEffectView, context: UIViewRepresentableContext<Self>) { uiView.effect = effect }
    }
}
```

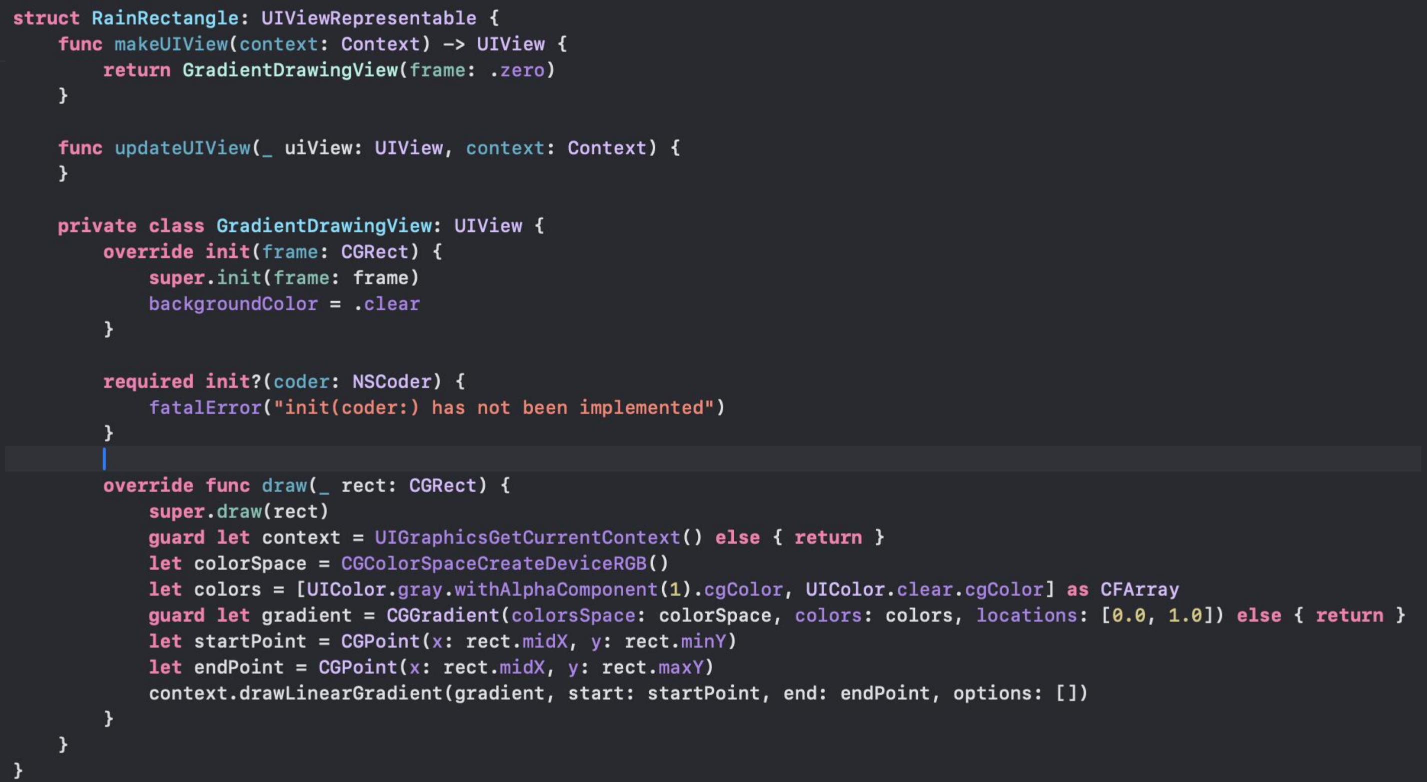


FireRectangle



```swift
struct FireRectangle: UIViewRepresentable {
    func makeUIView(context: Context) -> UIView {
        return GradientDrawingView(frame: .zero)
    }

    func updateUIView(_ uiView: UIView, context: Context) {
    }

    private class GradientDrawingView: UIView {
        override init(frame: CGRect) {
            super.init(frame: frame)
            backgroundColor = .clear
        }

        required init?(coder: NSCoder) {
            fatalError("init(coder:) has not been implemented")
        }

        override func draw(_ rect: CGRect) {
            super.draw(rect)
            guard let context = UIGraphicsGetCurrentContext() else { return }
            let colorSpace = CGColorSpaceCreateDeviceRGB()
            let colors = [UIColor.clear.cgColor,UIColor.red.withAlphaComponent(0.2).cgColor] as CFArray
            guard let gradient = CGGradient(colorsSpace: colorSpace, colors: colors, locations: [0.0, 1.0]) else { return }
            let startPoint = CGPoint(x: rect.midX, y: rect.minY)
            let endPoint = CGPoint(x: rect.midX, y: rect.maxY)
            context.drawLinearGradient(gradient, start: startPoint, end: endPoint, options: [])
        }
    }
}
```

RainRectangle



```swift
struct RainRectangle: UIViewRepresentable {
    func makeUIView(context: Context) -> UIView {
        return GradientDrawingView(frame: .zero)
    }

    func updateUIView(_ uiView: UIView, context: Context) {
    }

    private class GradientDrawingView: UIView {
        override init(frame: CGRect) {
            super.init(frame: frame)
            backgroundColor = .clear
        }

        required init?(coder: NSCoder) {
            fatalError("init(coder:) has not been implemented")
        }

        override func draw(_ rect: CGRect) {
            super.draw(rect)
            guard let context = UIGraphicsGetCurrentContext() else { return }
            let colorSpace = CGColorSpaceCreateDeviceRGB()
            let colors = [UIColor.gray.withAlphaComponent(1).cgColor, UIColor.clear.cgColor] as CFArray
            guard let gradient = CGGradient(colorsSpace: colorSpace, colors: colors, locations: [0.0, 1.0]) else { return }
            let startPoint = CGPoint(x: rect.midX, y: rect.minY)
            let endPoint = CGPoint(x: rect.midX, y: rect.maxY)
            context.drawLinearGradient(gradient, start: startPoint, end: endPoint, options: [])
        }
    }
}
```