ANIMACIONES

ANIMACIONES:

- En el ejemplo como queremos trackear 3 cambios de estado empezamos declarando 3 **variables** con 3 **estados**.
- Seguimos en la **Zstack** colocando ahora el icono.
- Para poder cambiar los **estados** al pulsar en los que acabamos de crear sin haber hecho ningún **botón** podemos hacerlo aplicándole a todo el Zstack el comando **.onTapeGesture{}**, que es un **closure** en el que tenemos que ingresar un método o un código que nos permita intercambiar los estados de la vista. Recordar que dentro de un **closure** para acceder a una variable de la clase hay que hacerlo con la palabra **self** delante.
- Para hacer las animaciones implícitas por defecto solo nos quedaría añadile tanto al círculo como a la imagen el modificador .animation(.default)
- Una página para ver las velocidades de las animaciones es: easings.net

ANIMACIONES EXPLÍCITAS:

- En vez de usar el modificador de SwiftUI .**animation**(), utilizaremos un **Completion Hunler** (un bloque de código) el cual actúa en nuestro ejemplo sobre los 3 cambios que deben ocurrir, es decir, cambiar el **color** del botón, cambiar el **color** del icono y cambiar el **tamaño** de icono. Y para ello usaremos el **withAnimation**{}, en el cual incluimos los tres nos deja ponerle entre paréntesis el (.default)
- La gracia de las animaciones explícitas es que podemos separar el que no vayan todas las animaciones(las 3 en este ejemplo) de la mano, dicho de otra forma, que no afect a los tres caos a la vez, ya que si queremos podemos sacar alguno de ellos del closure para que no le afecte.

```
struct ContentView: View {

   @State private var buttonColorChanged = false
   @State private var iconColorChanged = false
   @State private var iconSizeChanged = false

   var body: some View {
      ZStack{
      Circle()
         .frame(width: 180, height: 180)
         .foregroundColor(buttonColorChanged ? Color(.systemGray4) : Color(.systemGreen))

      Image(systemName: "keyboard")
            .font(.system(size: 80))
            .foregroundColor(iconColorChanged ? Color(.systemGreen) : Color(.systemGray6))
            .scaleEffect(iconSizeChanged ? 1.0 : 0.5)
      }
      //.animation(.spring(response: 1, dampingFraction: 0.4, blendDuration: 2), value: iconColorChanged)
      .onTapGesture {
         withAnimation(.spring(response: 0.5, dampingFraction: 0.2, blendDuration: 0.5)) {
            self.buttonColorChanged.toggle()
            self.iconColorChanged.toggle()

         }
         self.iconSizeChanged.toggle()
      }
   }
}

struct ContentView_Previews: PreviewProvider {
   static var previews: some View {
      ContentView()
   }
}
```

INDICADOR DE CARGA:

```
struct LoadingView: View {

   @State private var isLoading = false
   @State private var isMoving = false

   var body: some View {

      VStack{
      ZStack{
      Circle()
            .stroke(Color(.systemGray4), lineWidth: 20)
            .frame(width:150, height: 150)
      Circle()
         .trim(from: 0.0, to: 0.60)
         .stroke(Color.green, lineWidth: 20)
         .frame(width: 150, height: 150)
         .rotationEffect(Angle(degrees: isLoading ? 360 : 0))
         .animation(Animation.linear(duration: 1.5).repeatForever(autoreverses: false), value: isLoading)
         .onAppear {
            isLoading = true

         }
      }
      Text("Cargando")
            .font(.system(.largeTitle, design: .rounded))
            .bold()
         ZStack{
      RoundedRectangle(cornerRadius: 4)
            .stroke(Color(.systemGreen), lineWidth: 20)
            .frame(width:350, height: 8)
      RoundedRectangle(cornerRadius: 4)
            .stroke(Color(.systemYellow), lineWidth: 20)
            .frame(width:50, height: 8)
            .offset(x: isMoving ? 150 : -150)
            .animation(Animation.linear(duration: 2).repeatForever(autoreverses: false), value: isMoving)
            .onAppear {
               self.isMoving = true
            }
         }
   }
}

struct LoadingView_Previews: PreviewProvider {
   static var previews: some View {
      LoadingView()
   }
}
}
```

INDICADOR DE PROGRESO

```
struct ProgressView: View {

   @State private var progress: CGFloat = 0.0
   @State private var colorCircle = Color(.systemYellow)

      var body: some View {

         VStack{
      ZStack{
         Text("\(Int(progress*100)) %")
            .font(.system(.title, design: .rounded))
            .bold()

         Circle()
            .stroke(Color(.systemGreen), lineWidth: 15)
            .frame(width: 150, height: 150)

         Circle()
```

```
            .trim(from: 0, to: progress)
            .stroke(colorCircle, lineWidth: 15)
            .frame(width:150, height: 150)
            .rotationEffect(Angle.init(degrees: -60))
            .onAppear {
               Timer.scheduledTimer(withTimeInterval: 0.1, repeats: true) { timer in
                  self.progress += 0.01
                  if self.progress > 1{
                     timer.invalidate()
                     self.colorCircle = Color(.systemBlue)
                  }
               }
            }
         }
      }
    }
  }
struct ProgressView_Previews: PreviewProvider {
  static var previews: some View {
    ProgressView()
  }
}
```

RETRASAR UNA ANIMACIÓN

```
struct DotsLoadingView: View {

  @State private var isLoading = false

  var body: some View {
    HStack{
      ForEach(0...8, id: \.self){ index in
        Circle()
          .frame(width:16, height: 16)
          .foregroundColor(Color(.systemGreen))
          .scaleEffect(self.isLoading ? 0 : 1)
          .animation(Animation.linear(duration: 0.5).repeatForever().delay(Double(index)/8), value: isLoading)
          .onAppear {
            isLoading = true
          }
      }
    }
  }
}
struct DotsLoadingView_Previews: PreviewProvider {
  static var previews: some View {
    DotsLoadingView()
  }
}
```

EJERCICIO DE SIMULAR UN PAGO

```
enum Payment {
case Welcome, procesing, finalized
}

struct PaymentView: View {

  @State private var progress : CGFloat = 0.0
  @State private var isMoving = false
  @State  var step: Payment = .Welcome

  var body: some View {

    switch step {
    case .Welcome:
      VStack {
        let mano = Image(systemName: "hand.point.down.fill")
```

```
            Text("Haz clic aquí \(mano) para pagar")
              .font(.title2)

            HStack{
              Image(systemName: "applelogo")
                . font(.system(size: 40))

              Text("Pay")
                .font(.system(size: 50))
            }
            .onTapGesture {
              withAnimation {
                step = .procesing
              }
            }
            .frame(width: 250, height: 80)
            //.padding(.maximum(20, 0))
            .background(Color.white)
            .foregroundColor(.black)
            .cornerRadius(30)
            .overlay(
              RoundedRectangle(cornerRadius: 25)
                .stroke(lineWidth: 10)
            )

    }
      case .procesing:
        VStack{
          Text("Cargando...")
            .font(.system(.title, design: .rounded).bold())
          ZStack{
          RoundedRectangle(cornerRadius: 20)
            .stroke(Color.black, lineWidth: 15)
            .frame(width:300, height: 8)

          RoundedRectangle(cornerRadius: 20)
              .stroke(Color(.white), lineWidth: 15)
              .frame(width: 50, height: 8)
              .offset(x: isMoving ? 125 : -125)
              .animation(Animation.linear(duration: 2).repeatForever(autoreverses: true), value: isMoving)
              .onAppear {
                self.isMoving = true
                Timer.scheduledTimer(withTimeInterval: 5, repeats: true) { timer in
                  if self.progress > 1{
                    self.isMoving = false

                  }
                  timer.invalidate()
                  step = .finalized
                }
              }
        }
      }
      case .finalized:
        VStack{
          Text("Pago completado")
            .font(.system(.title, design: .rounded).bold())
          Image(systemName: "checkmark.circle")
            .font(.system(.largeTitle).bold())
            .transition(AnyTransition.scale.animation(Animation.easeOut))

            }.onTapGesture {
              step = .Welcome
          }
      }
    }
struct PaymentView_Previews: PreviewProvider {
    static var previews: some View {
      PaymentView()
    }}}
```