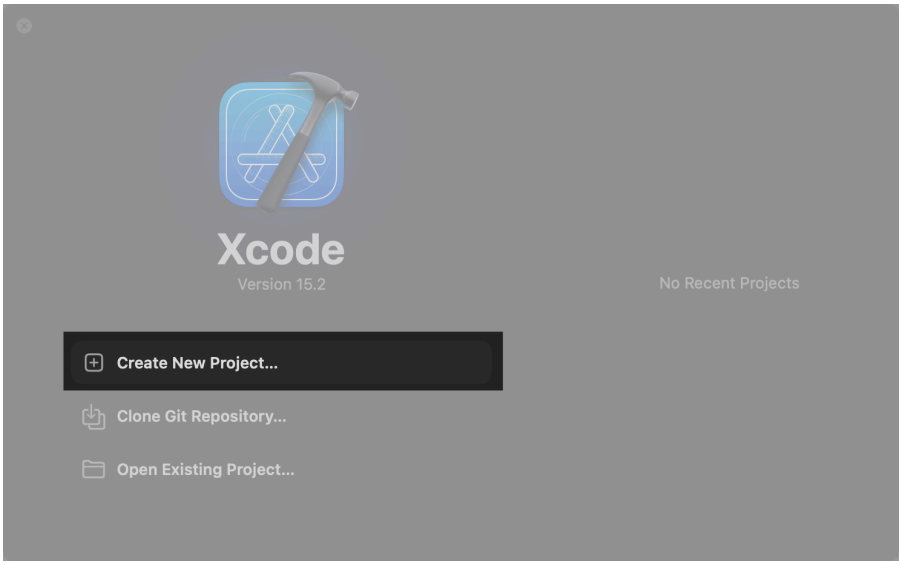
Sección 1

Crear una vista personalizada

En Personalizar vistas con propiedades, aprendiste a factorizar elementos repetidos de tu interfaz en una vista personalizada. Para hacer una aplicación con varios dados, crea una vista personalizada para representar un solo dado y luego úsala para hacer más dados en tu aplicación.

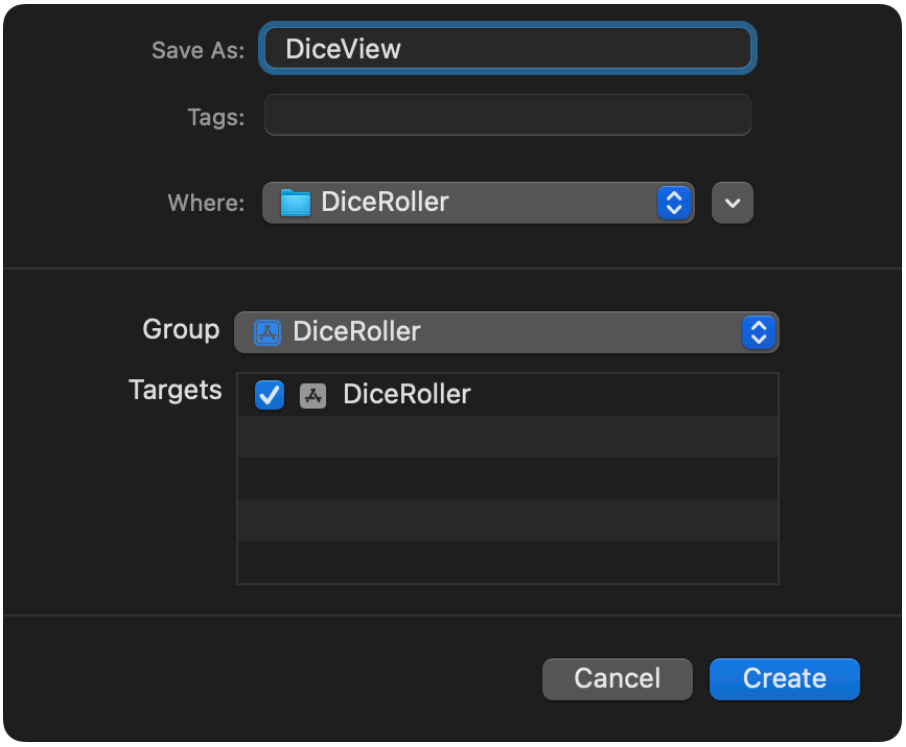
Paso 1

Crea un nuevo proyecto de aplicación iOS en Xcode llamado DiceRoller.



Paso 2

Crea un nuevo archivo de vista SwiftUI llamado DiceView.

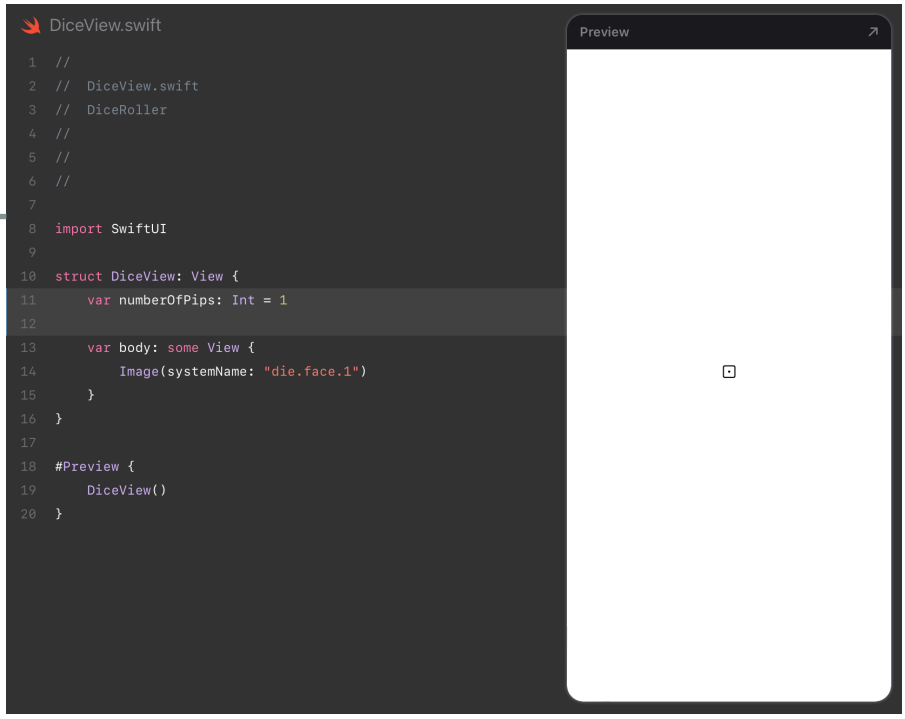


Paso 3

Reemplaza el código del cuerpo con una imagen de un dado.

Experimento

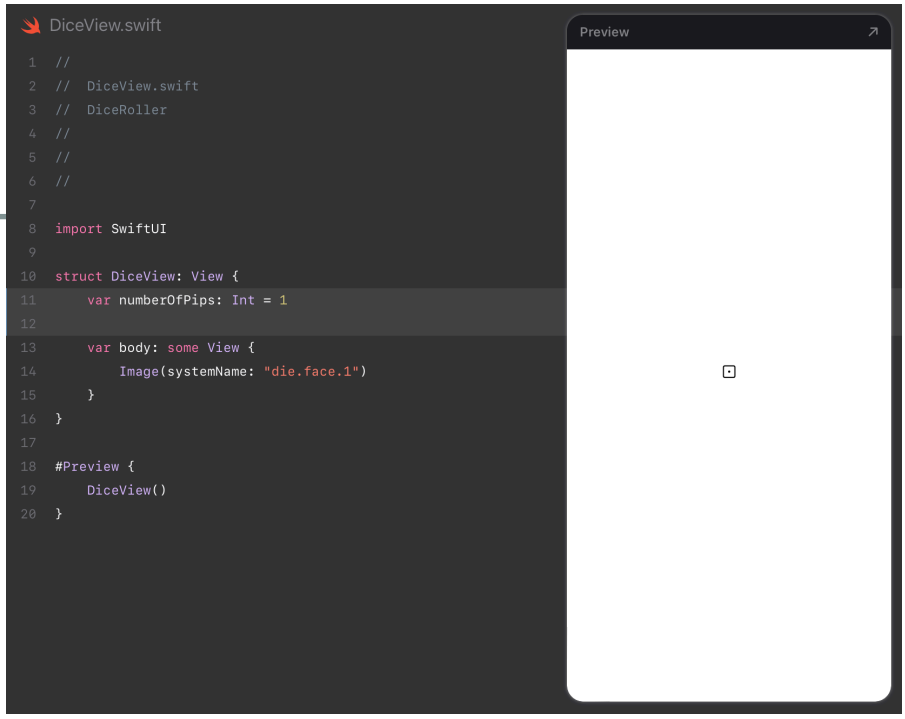
SF Symbols tiene imágenes para los seis lados de un dado. Cambia el número al final del nombre de la imagen para ver cada uno.



Paso 4

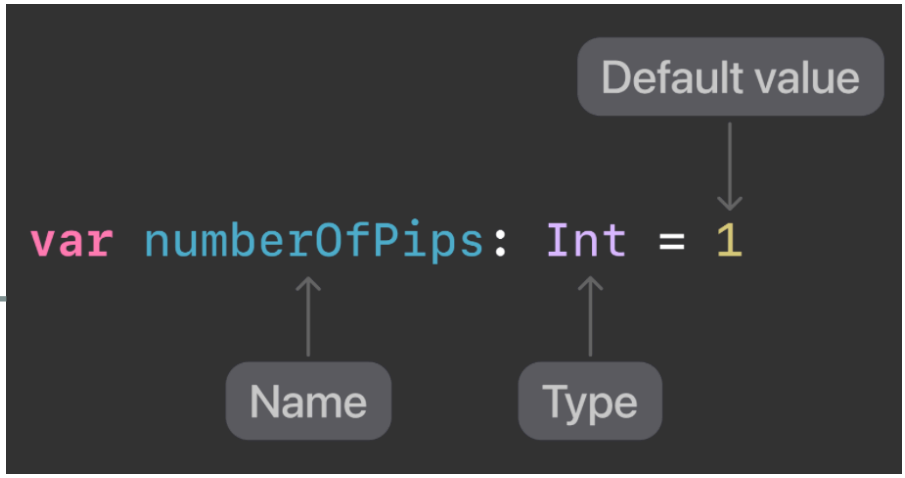
Añade una propiedad a la vista para representar el número de pips en los dados.

Utiliza el operador de asignación = para dar a la propiedad un valor predeterminado de 1.



Paso 5

Esta propiedad está marcada con la palabra clave var, lo que significa que puede asignarle nuevos valores. Esta vista es dinámica; cambiarás el valor de la propiedad cuando la gente tire los dados.



Cuando asigna un valor predeterminado a la propiedad de una estructura, no es necesario en el inicializador. Es por eso que no tuviste que cambiar la instancia de DiceView en la vista previa para incluir numberOfPips.

Paso 6

Utilice la interpolación de cadenas para mostrar la imagen de los dados utilizando el valor de su nueva propiedad.

Experimento

Cambie el valor predeterminado de numberOfPips para ver las imágenes de dados correspondientes.

DiceView.swift

```swift
//
//  DiceView.swift
//  DiceRoller
//
//
//

import SwiftUI

struct DiceView: View {
    var numberOfPips: Int = 1

    var body: some View {
        Image(systemName: "die.face.\(numberOfPips)")
    }
}

#Preview {
    DiceView()
}
```

Preview

Paso 7

Utilice modificadores para aumentar el tamaño de la imagen. El modificador .resizable le dice a la imagen que puede estirarse para llenar cualquier espacio disponible. No quieres que los dados llenen todo el espacio disponible, por lo que limitas la imagen estableciendo su tamaño de fotograma.

Por lo general, hace coincidir el tamaño de los símbolos de SF con su contenido circundante utilizando el modificador .font. En este caso, estás usando la imagen como contenido puramente gráfico, por lo que está bien usar .resizable y .frame.

DiceView.swift

```swift
//
//  DiceView.swift
//  DiceRoller
//
//
//

import SwiftUI

struct DiceView: View {
    var numberOfPips: Int = 1

    var body: some View {
        Image(systemName: "die.face.\(numberOfPips)")
            .resizable()
            .frame(width: 100, height: 100)
    }
}

#Preview {
    DiceView()
}
```

Preview

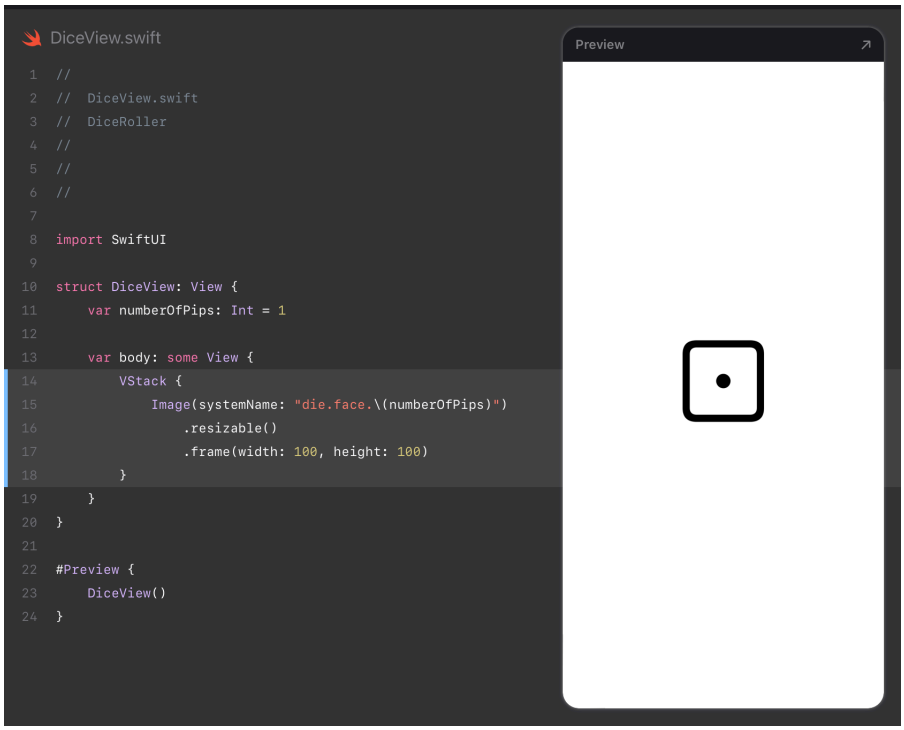
Sección 2

Añade un botón para tirar los dados

Usa un botón para cambiar la imagen de los dados. Un botón utiliza un cierre para ejecutar el código cuando una persona lo toca.

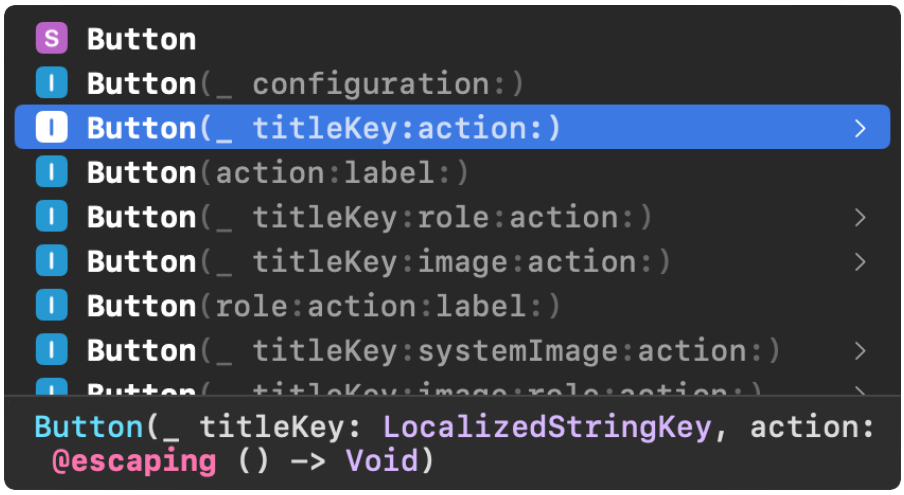
Paso 1

Inserta la imagen en un VStack para que puedas añadir el botón debajo de ella.



Paso 2

Dentro del VStack, debajo de la imagen y sus modificadores, comienza a escribir Button. La finalización del código le ofrecerá varias sugerencias; elija la que tenga una clave de título y una acción.



Si ves título en lugar de titleKey, elige ese inicializador.

Paso 3

Reemplace el primer marcador de posición con la cadena "Roll".

```
8  import SwiftUI
9
10 struct DiceView: View {
11     var numberOfPips: Int = 1
12
13     var body: some View {
14         VStack {
15             Image(systemName: "die.face.\(numberOfPips)")
16                 .resizable()
17                 .frame(width: 100, height: 100)
18
19             Button("Roll", action: () -> Void)
20         }
21     }
22 }
23
24 #Preview {
25     DiceView()
26 }
```

Paso 4

El segundo parámetro, acción: () -> Vacío, requiere un cierre que ejecuta el código cuando la gente toca el botón. Presione Tab para seleccionar el marcador de posición y, a continuación, pulse Retorno.

Xcode elimina el parámetro de acción por completo y lo reemplaza con un cierre contenido en un par de llaves. Hay un marcador de posición de código dentro de las llaves donde escribirás tu código para el botón.

```
8  import SwiftUI
9
10 struct DiceView: View {
11     var numberOfPips: Int = 1
12
13     var body: some View {
14         VStack {
15             Image(systemName: "die.face.\(numberOfPips)")
16                 .resizable()
17                 .frame(width: 100, height: 100)
18
19             Button("Roll") {
20                 code
21             }
22         }
23     }
24 }
25
26 #Preview {
27     DiceView()
28 }
```

Paso 5

Reemplace el último marcador de posición con un código para elegir un número aleatorio de pips para los dados.

Int.random selecciona un número entero aleatorio del rango dentro de sus paréntesis; el código 1...6 crea un rango de números enteros del 1 al 6.

Nota

Este es otro ejemplo de uso de un operador de asignación. Esta vez, estás actualizando el valor de numberOfPips cada vez que se ejecuta el código.

```
import SwiftUI

struct DiceView: View {
    var numberOfPips: Int = 1

    var body: some View {
        VStack {
            Image(systemName: "die.face.\(numberOfPips)")
                .resizable()
                .frame(width: 100, height: 100)

            Button("Roll") {
                numberOfPips = Int.random(in: 1...6)
            }
        }
    }
}

#Preview {
    DiceView()
}
```

Hay un error en su código que corregirá en la siguiente sección cambiando numberOfPips a una propiedad @State.

Sección 3

Usar el estado para actualizar una vista

Todas las aplicaciones tienen datos, o estado, que cambian con el tiempo. Cuando una aplicación cambia de estado, es posible que tenga que actualizar su interfaz. Sin embargo, SwiftUI no supervisa todas las propiedades de una aplicación de forma predeterminada.

En esta aplicación, debes actualizar la imagen cuando una persona toca el botón Roll. Para decirle a SwiftUI que supervise numberOfPips y actualice la interfaz de usuario cuando cambie, marque la propiedad con la palabra clave @State.

Paso 1

Haz de numberOfPips una propiedad de @State. A continuación, pulsa el botón Roll unas cuantas veces para comprobar que la imagen cambia.

El estado de la vista es propiedad de la vista. Siempre se marcan las propiedades estatales como privadas para que otras vistas no puedan interferir con su valor.

```

import SwiftUI

struct DiceView: View {
    @State private var numberOfPips: Int = 1

    var body: some View {
        VStack {
            Image(systemName: "die.face.\(numberOfPips)")
                .resizable()
                .frame(width: 100, height: 100)

            Button("Roll") {
                numberOfPips = Int.random(in: 1...6)
            }
        }
    }
}

#Preview {
    DiceView()
}
```

Paso 2

Añade un borde al botón para diferenciarlo de la imagen.

```
9
10  struct DiceView: View {
11      @State private var numberOfPips: Int = 1
12
13      var body: some View {
14          VStack {
15              Image(systemName: "die.face.\(numberOfPips)")
16                  .resizable()
17                  .frame(width: 100, height: 100)
18
19              Button("Roll") {
20                  numberOfPips = Int.random(in: 1...6)
21              }
22              .buttonStyle(.bordered)
23          }
24      }
25  }
26
27  #Preview {
28      DiceView()
29  }
```

Paso 3

Para hacer que la transición se desvanezca de la imagen de dados antigua a la nueva, usa conAnimation para animar el cambio.

Agregar withAnimation indica a SwiftUI que anime cualquier cambio de estado que se produzca dentro de su código. Utiliza un cierre final, similar a la forma en que funciona Button.

```
8   import SwiftUI
9
10  struct DiceView: View {
11      @State private var numberOfPips: Int = 1
12
13      var body: some View {
14          VStack {
15              Image(systemName: "die.face.\(numberOfPips)")
16                  .resizable()
17                  .frame(width: 100, height: 100)
18
19              Button("Roll") {
20                  withAnimation {
21                      numberOfPips = Int.random(in: 1...6)
22                  }
23              }
24              .buttonStyle(.bordered)
25          }
26      }
27  }
28
29  #Preview {
30      DiceView()
31  }
```

Sección 4

Crea una visualización dinámica de dados

Añade funcionalidad para elegir el número de dados creando otra propiedad. Marcar la propiedad con @State garantiza que la interfaz se actualice cuando cambia el número de dados.

Paso 1

En ContentView, reemplace el contenido de VStack con un título.

Puedes usar modificadores en el tipo de fuente de la misma manera que los usas con las vistas.

Experimento

Encadena otros modificadores de fuente en .largeTitle para ver los efectos.

Si quieres, puedes poner los modificadores en líneas separadas:

```
.font(.largeTitle
.lowercaseSmallCaps()
.Bold()
)
```

```swift
import SwiftUI

struct DiceView: View {
    @State private var numberOfPips: Int = 1

    var body: some View {
        VStack {
            Image(systemName: "die.face.\(numberOfPips)")
                .resizable()
                .frame(width: 100, height: 100)

            Button("Roll") {
                numberOfPips = Int.random(in: 1...6)
            }
        }
    }
}

#Preview {
    DiceView()
}
```

Paso 2

Añade un HStack con tres instancias de DiceView. Intenta tirar cada dado.

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                DiceView()
                DiceView()
                DiceView()
            }
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Paso 3

Para poder mostrar cualquier número de dados, utilice una vista ForEach. Repite DiceView tres veces usando un rango de 1 a 3. Escriba el código manualmente; ForEach tiene opciones de finalización de código para muchos usos diferentes.

La vista ForEach es dinámica; calcula sus subvistas en función de su entrada, que puede cambiar con el estado de la aplicación. Se crea un rango usando 1...3, tal como lo hizo en Int.random (en: 1...6). La vista ForEach crea una vista de dados para cada valor del rango.

```
8  import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         VStack {
13             Text("Dice Roller")
14                 .font(.largeTitle.lowercaseSmallCaps())
15
16             HStack {
17                 ForEach(1...3, id: \.self) { _ in
18                     DiceView()
19                 }
20             }
21         }
22         .padding()
23     }
24 }
25
26 #Preview {
27     ContentView()
28 }
```

Paso 5

Utilice la nueva propiedad para hacer que el rango sea dinámico.

Experimento

Intenta cambiar el valor predeterminado de numberOfDice para ver cómo cambia la interfaz.

```
8  import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         VStack {
13             Text("Dice Roller")
14                 .font(.largeTitle.lowercaseSmallCaps())
15
16             HStack {
17                 ForEach(1...3, id: \.self) { _ in
18                     DiceView()
19                 }
20             }
21         }
22         .padding()
23     }
24 }
25
26 #Preview {
27     ContentView()
28 }
```

Paso 6

Debajo de la vista ForEach y HStack, añade dos botones para aumentar y disminuir el número de dados.

Mira el código dentro del cierre de cada botón. En Swift, puedes usar += y -= para sumar o restar del valor actual de una propiedad. En este caso, estás sumando o restando 1.

Experimento

Prueba los botones para asegurarte de que funcionan.

```swift
import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice") {
                    numberOfDice -= 1
                }

                Button("Add Dice") {
                    numberOfDice += 1
                }
            }
            .padding()
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Paso 7

Reduzca el número de dados a uno, luego toque el botón Eliminar dados. La vista previa se bloquea, porque el rango 1...0 no es válido.

Paso 8

Para evitar que la gente toque el botón Eliminar dados cuando solo hay un DiceView, puede desactivar el botón para evitar el bloqueo y dar a la gente una señal visual de que el botón no responde. Utilice el modificador .disabled para desactivar el botón Eliminar dados cuando numberOfDice tenga un valor de 1.

Utilizas el operador == para comprobar si dos números son iguales. El resultado de esta comparación es un Bool, que tiene uno de dos valores: verdadero o falso. Cuando numberOfDice == 1 es verdadero, el modificador desactiva el botón.

```
8  import SwiftUI
9
10 struct ContentView: View {
11     @State private var numberOfDice: Int = 1
12
13     var body: some View {
14         VStack {
15             Text("Dice Roller")
16                 .font(.largeTitle.lowercaseSmallCaps())
17
18             HStack {
19                 ForEach(1...numberOfDice, id: \.self) { _ in
20                     DiceView()
21                 }
22             }
23
24             HStack {
25                 Button("Remove Dice") {
26                     numberOfDice -= 1
27                 }
28                 .disabled(numberOfDice == 1)
29
30                 Button("Add Dice") {
31                     numberOfDice += 1
32                 }
33             }
34             .padding()
35         }
36         .padding()
37     }
38 }
39
40 #Preview {
41     ContentView()
42 }
```

Paso 9

Las imágenes de los dados se fijan en 100x100 puntos, por lo que solo caben tres imágenes de dados en la pantalla. Utilice el modificador .disabled con el botón Añadir dados para evitar que las personas tengan más de tres dados.

```
9
10 struct ContentView: View {
11     @State private var numberOfDice: Int = 1
12
13     var body: some View {
14         VStack {
15             Text("Dice Roller")
16                 .font(.largeTitle.lowercaseSmallCaps())
17
18             HStack {
19                 ForEach(1...numberOfDice, id: \.self) { _ in
20                     DiceView()
21                 }
22             }
23
24             HStack {
25                 Button("Remove Dice") {
26                     numberOfDice -= 1
27                 }
28                 .disabled(numberOfDice == 1)
29
30                 Button("Add Dice") {
31                     numberOfDice += 1
32                 }
33                 .disabled(numberOfDice == 3)
34             }
35             .padding()
36         }
37         .padding()
38     }
39 }
40
41 #Preview {
42     ContentView()
43 }
```

Sección 5

Adapta la interfaz para obtener más dados

Utilice anchos y alturas flexibles para permitir que las imágenes de los dados se redimensionen dinámicamente en función del número de dados en la pantalla.

Paso 1

Aumenta el límite de dados a cinco y pulsa Añadir dados hasta que haya cinco dados.

Una vista HStack siempre le da a sus subvistas los tamaños solicitados, incluso si necesita extenderse más allá de los límites de la pantalla.

```swift
//

import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Paso 2

Fija la vista previa de ContentView y luego cambia a DiceView. Usa un marco flexible para permitir que las imágenes de los dados se encojan.

```swift

import SwiftUI

struct DiceView: View {
    @State private var numberOfPips: Int = 1

    var body: some View {
        VStack {
            Image(systemName: "die.face.\(numberOfPips)")
                .resizable()
                .frame(maxWidth: 100, maxHeight: 100)

            Button("Roll") {
                withAnimation {
                    numberOfPips = Int.random(in: 1...6)
                }
            }
            .buttonStyle(.bordered)
        }
    }
}

#Preview {
    DiceView()
}
```

Paso 3

Ahora que las imágenes de los dados tienen anchos y alturas flexibles, el HStack puede reducirlas para que quepan en la pantalla. Pero cuando hay cuatro o cinco dados, se estiran verticalmente porque no hay nada por encima o por debajo del HStack que limite su altura. Para evitar esto, establezca la relación de aspecto de la imagen de los dados en 1.

```
7
8  import SwiftUI
9
10 struct DiceView: View {
11     @State private var numberOfPips: Int = 1
12
13     var body: some View {
14         VStack {
15             Image(systemName: "die.face.\(numberOfPips)")
16                 .resizable()
17                 .frame(maxWidth: 100, maxHeight: 100)
18                 .aspectRatio(1, contentMode: .fit)
19
20             Button("Roll") {
21                 withAnimation {
22                     numberOfPips = Int.random(in: 1...6)
23                 }
24             }
25             .buttonStyle(.bordered)
26         }
27     }
28 }
29
30 #Preview {
31     DiceView()
32 }
```

Una relación de aspecto 1:1, o cuadrado, tiene el mismo ancho y altura. El modo de contenido .fit significa que si la imagen no tiene la misma relación de aspecto que el espacio disponible, se reducirá al eje más pequeño y dejará espacio vacío en el otro.

Sección 6

Utilice imágenes en las etiquetas de los botones

Personaliza los botones de añadir y quitar dados para mostrar imágenes en lugar de texto.

Paso 1

En ContentView, añade otro argumento al botón Añadir dados para añadir una imagen.

La vista que se muestra un botón se llama su etiqueta. En muchos casos, utilizarás una combinación de imágenes y texto para las etiquetas de los botones.

```
7
8  import SwiftUI
9
10 struct ContentView: View {
11     @State private var numberOfDice: Int = 1
12
13     var body: some View {
14         VStack {
15             Text("Dice Roller")
16                 .font(.largeTitle.lowercaseSmallCaps())
17
18             HStack {
19                 ForEach(1...numberOfDice, id: \.self) { _ in
20                     DiceView()
21                 }
22             }
23
24             HStack {
25                 Button("Remove Dice") {
26                     numberOfDice -= 1
27                 }
28                 .disabled(numberOfDice == 1)
29
30                 Button("Add Dice", systemImage: "plus.circle.fill") {
31                     numberOfDice += 1
32                 }
33                 .disabled(numberOfDice == 5)
34             }
35             .padding()
36         }
37         .padding()
38     }
39 }
40
41 #Preview {
42     ContentView()
43 }
```

Paso 2

Para este botón, una imagen es suficiente para decirle a la gente lo que hace. Pero un botón siempre debe tener una etiqueta de texto, visible o no, para que esté disponible para las personas que dependen de funciones como VoiceOver. Oculta el texto del botón usando el modificador .labelStyle.

A pesar de que el modificador afecta a ambos botones, el botón Eliminar dados todavía tiene una etiqueta de texto porque no tiene un icono para mostrar.

```
import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Paso 3

Aumenta el tamaño del botón usando una fuente de título.

```
import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Paso 4

Añade una imagen al botón Eliminar dados.

```swift
import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice", systemImage: "minus.circle.fill") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

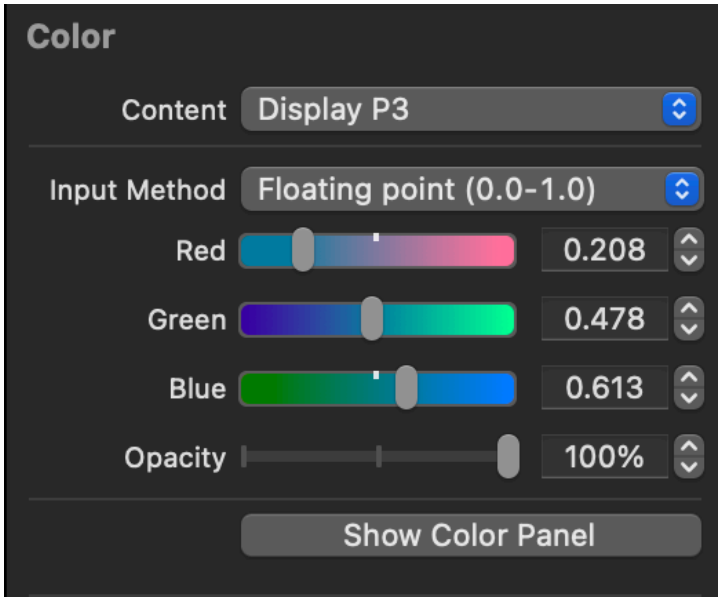
Sección 7

Mejora el diseño de tu aplicación

Dale estilo a los botones y los dados y dale a tu aplicación un color de fondo.

Paso 1

Añade un conjunto de colores de fondo personalizado llamado Fondo de la aplicación.



Para un repaso sobre cómo hacer esto, consulte esta sección en Diseñar una interfaz.

Propina

Elige los nombres de tus conjuntos de colores con cuidado. Algunos nombres, como Fondo y Azul, entran en conflicto con los colores definidos por el sistema existentes.

Paso 2

Utilice el color personalizado para el fondo de ContentView..

```swift

import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice", systemImage: "minus.circle.fill") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
        .background(.appBackground)
    }
}

#Preview {
    ContentView()
}
```

Paso 3

Utilice el modificador .tint en el VStack para aplicar un tinte blanco global a la aplicación.

El modificador .tint solo afecta a las vistas que dependen del color de acento, normalmente controles como botones y conmutadores. Las vistas usan el tinte de diferentes maneras. Por ejemplo, tenga en cuenta que el botón Roll adquirió un fondo blanco sutil.

```swift

import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                }
            }

            HStack {
                Button("Remove Dice", systemImage: "minus.circle.fill") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
        .background(.appBackground)
        .tint(.white)
    }
}

#Preview {
    ContentView()
}
```

Paso 4

Cambia el color de las otras vistas a blanco usando .foregroundStyle.

```swift

import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())
                .foregroundStyle(.white)

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                        .foregroundStyle(.white)
                }
            }

            HStack {
                Button("Remove Dice", systemImage: "minus.circle.fill") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
        .background(.appBackground)
        .tint(.white)
    }
}

#Preview {
    ContentView()
}
```

Paso 5

Para hacer que el fondo se expanda para llenar la pantalla, usa un marco.

Pase .infinity para los parámetros maxWidth y maxHeight para que el marco se expanda en ambas direcciones en la medida de lo posible.

```swift
import SwiftUI

struct ContentView: View {
    @State private var numberOfDice: Int = 1

    var body: some View {
        VStack {
            Text("Dice Roller")
                .font(.largeTitle.lowercaseSmallCaps())
                .foregroundStyle(.white)

            HStack {
                ForEach(1...numberOfDice, id: \.self) { _ in
                    DiceView()
                        .foregroundStyle(.white)
                }
            }

            HStack {
                Button("Remove Dice", systemImage: "minus.circle.fill") {
                    numberOfDice -= 1
                }
                .disabled(numberOfDice == 1)

                Button("Add Dice", systemImage: "plus.circle.fill") {
                    numberOfDice += 1
                }
                .disabled(numberOfDice == 5)
            }
            .padding()
            .labelStyle(.iconOnly)
            .font(.title)
        }
        .padding()
        .frame(maxWidth: .infinity, maxHeight: .infinity)
        .background(.appBackground)
        .tint(.white)
    }
}

#Preview {
    ContentView()
}
```

Paso 6

Los dados clásicos son blancos con calas negras. Para hacer este cambio, abra el archivo DiceView y utilice la variante rellena del nombre del símbolo de los dados para rellenar los dados con un color sólido.

De forma predeterminada, SwiftUI renderiza la mayoría de los símbolos de SF con un solo color determinado por el estilo de primer plano. Por eso los dados son blancos. Los pips son del mismo color que el fondo porque son transparentes de forma predeterminada.

```
7
8  import SwiftUI
9
10 struct DiceView: View {
11     @State private var numberOfPips: Int = 1
12
13     var body: some View {
14         VStack {
15             Image(systemName: "die.face.\(numberOfPips).fill")
16                 .resizable()
17                 .frame(maxWidth: 100, maxHeight: 100)
18                 .aspectRatio(1, contentMode: .fit)
19
20             Button("Roll") {
21                 withAnimation {
22                     numberOfPips = Int.random(in: 1...6)
23                 }
24             }
25             .buttonStyle(.bordered)
26         }
27     }
28 }
29
30 #Preview {
31     DiceView()
32 }
```

Paso 7

Para hacer que los pips sean negros, usa dos colores de primer plano: uno primario y otro secundario. Utilice el modificador .foregroundStyle para establecer el color primario en negro y el color secundario en blanco.

Estás viendo el efecto del modo de renderizado de la paleta de símbolos SF. Para los símbolos de dados rellenos, el pip asume el color primario, mientras que el resto toma el color secundario.

Nota

Algunos símbolos de SF están compuestos en una jerarquía, con diferentes elementos que pueden tomar colores primarios, secundarios y terciarios. Hay cuatro modos de renderizado de símbolos SF, que puedes configurar manualmente usando el modificador .symbolRenderingMode. Puedes leer más sobre los modos de renderizado, y mucho más sobre los símbolos de SF, en las Directrices de la interfaz humana.

```
7
8  import SwiftUI
9
10 struct DiceView: View {
11     @State private var numberOfPips: Int = 1
12
13     var body: some View {
14         VStack {
15             Image(systemName: "die.face.\(numberOfPips).fill")
16                 .resizable()
17                 .frame(maxWidth: 100, maxHeight: 100)
18                 .aspectRatio(1, contentMode: .fit)
19                 .foregroundStyle(.black, .white)
20
21             Button("Roll") {
22                 withAnimation {
23                     numberOfPips = Int.random(in: 1...6)
24                 }
25             }
26             .buttonStyle(.bordered)
27         }
28     }
29 }
30
31 #Preview {
32     DiceView()
33 }
```