SwiftUI Tutorial

By: Christian Thompson, Connor Muma, Josh Hubbard

Overview

SwiftUI is a way to build user interfaces for applications on Apple devices. SwiftUI can be used on multiple Apple platforms such as iOS 13, macOS 10.15, tvOS 13 and watchOS 6, or any later version of these platforms. The SwiftUI framework provides all the layout structures and views for a user interface as well as event handlers for any kind of user input. SwiftUI views can be integrated with other objects from frameworks such as UIKit. In this tutorial we will be creating a demo randomizer application using SwiftUI. The function of the app is to create multiple choices and choose randomly between these choices a set number of times.
Scrapped due to time constraint and an unforeseen roadblock with populating Lists (The user input is stored in a dictionary and displayed in a tableview, and entered lists can be saved and loaded by the user. The application will keep track of selection history, and additionally the user can also delete their saved lists through the use of UserDefaults.)

Getting Started

This SwiftUI tutorial will be done all in Xcode. For this tutorial we will be using Xcode version 12.2. SwiftUI cannot be used on versions of Xcode older than Xcode1. When creating your project in Xcode, set the Interface to SwiftUI and set the life cycle to SwiftUI App. If you are trying to use an older version of Xcode, there may be a SwiftUI checkbox, so make sure to check it. For the language option, select Swift. Xcode can be installed for free through Apple's developer website: https://developer.apple.com/xcode/

Step-by-step coding instructions

```
ZStack {
    textColor
        .edgesIgnoringSafeArea(.all)
    VStack(alignment: .center, spacing: 10) {
        HStack() {
            TextField("Enter text here", text: $enteredText)
                .padding(.all, 5.0)
                .textFieldStyle(RoundedBorderTextFieldStyle
                    ())
                .ignoresSafeArea()
                .background(textColor)

            Button("Save/Load") {
                saveLoad()
            }
            .padding(.all, 10.0)
            .frame(alignment: .center)
            .background(accentColor)
            .foregroundColor(textColor)
            .font(Font.caption.weight(.heavy))
            .cornerRadius(4)

        }
        .padding(.all, 10.0)

        List(items) { item in
            Text(item.item)
        }
        .padding([.bottom, .trailing], 10.0)
        .background(textColor)
        .listRowBackground(textColor)
```

For our code we are using a ContentView which holds another view. That view encompasses several layers of vertical and horizontal stack that contain our buttons and text fields. This is where we are able to adjust the style and placement of the elements we put on the screen. We have all of our stacks stored inside of one main Z Stack which holds a Vertical Stack. At the top of the vertical stack we have a TextField and our Save/Load button which is placed inside of a horizontal stack. We have our TextField wired up so that whatever the user enters, it automatically gets displayed into our list element on the screen after hitting the Enter button.

Next we have an Enter button inside of a horizontal stack. this submits the users inputted string into the displayed list as long as there is text in it, otherwise it will close the keyboard. In the pictures below you will see the different options we were able to apply to all the elements on the screen (ex. Color, font, alignment, etc.)

```
HStack {
    Button(action: {            // enter button
        if enteredText != "" {
            enterButtonAction(addText:
                self.enteredText)
        }
        self.enteredText = ""
        UIApplication.shared.endEditing()
    }) {
        Text("Enter")
            .lineLimit(2)
            .multilineTextAlignment(.center)
            .frame(width:
                UIScreen.main.bounds.size.width -
                30, height: 50, alignment: .center)
            .background(accentColor)
            .foregroundColor(textColor)
            .font(Font.caption.weight(.heavy))
            .cornerRadius(4)
    }
}
```

Next we have another horizontal stack which houses several buttons which are also in horizontal stacks. We have the x1 button, x5 button, and x10 button. Once any of these buttons are pressed they call their own respective helper methods. Once again we have applied colors, fonts, curved edges, etc to the buttons for aesthetic purposes as well as how they line up on the screen.

```
HStack {
    HStack {
        Button(action: {            // choose 1 button
            choose1ButtonAction()
        }) {
            Text("Choose x1")
                .padding(.all, 10.0)
                .frame(width: (UIScreen.main.bounds.size.width - 45) / 3.0, height: 40, alignment: .center)
                .background(accentColor)
                .foregroundColor(textColor)
                .font(Font.caption.weight(.heavy))
                .cornerRadius(4)
        }
    }
    HStack {
        Button(action: {            // choose 5 button
            choose5ButtonAction()
        }) {
            Text("Choose x5")
                .padding(.all, 10.0)
                .frame(width: (UIScreen.main.bounds.size.width - 45) / 3.0, height: 40, alignment: .center)
                .background(accentColor)
                .foregroundColor(textColor)
                .font(Font.caption.weight(.heavy))
                .cornerRadius(4)
        }
    }
    HStack {
        Button(action: {            // choose 10 button
            choose10ButtonAction()
        }) {
            Text("Choose x10")
                .padding(.all, 10.0)
                .frame(width: (UIScreen.main.bounds.size.width - 45) / 3.0, height: 40, alignment: .center)
                .background(accentColor)
                .foregroundColor(textColor)
                .font(Font.caption.weight(.heavy))
                .cornerRadius(4)
        }
    }
}
```

Next we have another horizontal stack which holds the Clear button. The Clear button removes all items from the List and off of the screen.

```
HStack {
    Button(action: {            // choose 10 button
        clearButtonAction()
    }) {
        Text("Clear")
            .padding(.all, 10.0)
            .frame(width: UIScreen.main.bounds.size.width - 30, height: 40, alignment: .center)
            .background(accentColor)
            .foregroundColor(textColor)
            .font(Font.caption.weight(.heavy))
            .cornerRadius(4)
    }
}
```

```
func clearButtonAction() {
    print("clear")
    items.removeAll()
}
```

Next we have the code which controls all of the randomize buttons. This code essentially picks a random number and assigns that to a value in the list. Once an item is chosen, if you only pressed the x1 button, then a star will appear next to the item that got chosen. If you press the x5 or x10 button then a count will appear next to the list items to show how many times that item was picked in either 5 or 10 cycles.

```swift
func choose1ButtonAction() {
    if items.count > 0 {
        restoreArrayValues(list: displayArray)
        let chosen = items.randomElement()?.item
        for x in 0..<items.count {
            if items[x].item == chosen {
                items[x].item = "\(items[x].item) »"
            }
        }

        print(items.randomElement()!)
    }


}
func choose5ButtonAction() {    // this doesn't work yet
    if items.count > 0 {
        items = displayArray
        var chosenItem = items.randomElement()
        for _ in 0..<5 {
            chosenItem = items.randomElement()
            for x in 0..<items.count {
                if chosenItem?.item == items[x].item {
                    items[x].count += 1
                }
            }
        }

        for item in 0..<items.count {
            items[item].item = "\(items[item].item): Chosen \(items[item].count) times"
        }
    }

}
```

```swift
func choose10ButtonAction() {
    if items.count > 0 {
        items = displayArray
        var chosenItem = items.randomElement()
        for _ in 0..<10 {
            chosenItem = items.randomElement()
            for x in 0..<items.count {
                if chosenItem?.item == items[x].item {
                    items[x].count += 1
                }
            }
        }

        for item in 0..<items.count {
            items[item].item = "\(items[item].item): Chosen \(items[item].count) times"
        }
    }

}
```

Conclusion

SwiftUI is to be used as an alternative to the older UIKit development platform. Other alternative development frameworks are React Native and Flutter. These are both cross platform frameworks made by third party developers. For more tutorials, Apple has SwiftUI tutorials on app design and layout: https://developer.apple.com/tutorials/swiftui/working-with-ui-controls , and framework integration: https://developer.apple.com/tutorials/swiftui/interfacing-with-uikit .
Link to github repository with program source code: https://github.com/HubbardJosh/CIS357FinalProject