Custom Layout

CMSC 10500

Overview

A fair amount of the layout process is handled automatically by SwiftUI. This makes layout super simple, but sometimes we will want to use a different layout than SwiftUI expects.

In today's demo we will look a bit about conditional layout and custom layout for these cases.

Container Views

Container views are views which hold other views. We have seen a couple so far.

- `ZStack { … }`
- `HStack { … }`
- `VStack { … }`
- `ForEach(...) { … }`

Other views are built-in, like `Text(...)`, `Rectangle()`, and `Spacer()`.

Complex views are constructed by combining built-in views using containers views.

We call the container view containing another view its **parent view**.

The Automatic Layout Process

1. Container views *offer* space to each of the views they contain in order of priority (more on that in the next slide).
2. When a view is offered space, it take as much of that space as it wants.
3. Container views layout the contained views according to their definition. `HStack`s positions each view in a horizontal line. `ForEach` defers the layout to *its parent view.*

The `ContentView` is offered the entire screen (more precisely the entire safe-area in the screen, which excludes things like the top banner) by the application when it starts running.

Priority of Views

Views are order in priority according to their *flexibility*. Here is a short list of views from greatest to least priority.

- `Image(...)` (we will see these in the demo today)
- `Text(...)`
- `Rectangle()`, `Spacer()`, other container views

This priority can be changed, using the `.layoutPriority(_:)` method.

View Methods for Layout

Some methods of views actual do a bit of layout.

- `.padding()`
- `.aspectRatio(...)` (important for Tic Tac Toe to make a square board)
- `.frame(...)`
- `.position(...)`

Conditional Layout

It turns out we can use if statements inside Views. This helps if we want the view to depend on the state of the model or other views. (It will be useful for Tic Tac Toe to make the symbols appear when positions are tapped).

```
ZStack {
    Rectangle()
    if <condition> {
        Text(“X”)
    else {
        Text(“O”)
    }
}
```

Custom Layout with `GeometryReader`

`GeometryReader` is a container view that gives access to its view its size.

This makes explicit the amount of space that a parent view offers its views

```
GeometryReader { geometry in
    …
}
```

geometry (a `GeometryProxy`) has a property .size (a `CGSize`) which has properties .width and .height (`CGFloat`s).

Using these values we can use the `.frame(...)` and `.position(...)` methods to place views where we want in the screen.