Assignment V: JSON Memorize Theme

Objective

The goal of this assignment is to add some simple, yet important functionality to Memorize that you will need in Assignment 6 while the concept (making a custom `struct` persistent) is still fresh in your mind from Lecture 7.

Be sure to review the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Due

This assignment is due before you start watching Lecture 9. Yes, it is due at exactly the same time as Assignment 4 (they are being assigned in parallel, but this one is a lot easier).

Materials

- You must apply this assignment to the code from your solution to Homework 2. If you had problems with that assignment, you can (and should) repair them before proceeding to this assignment. You will probably want to also incorporate the changes from Lectures 5 and 6 (animation) so that your Memorize is awesome, but that is not required.

Required Tasks

1. Remove the "random number of cards" theme option from your Memorize game. Each theme will now have its own specific, pre-defined number of cards. In other words, how many cards are in a game is part of the theme for that game and it can no longer be "random".
2. Every time a new game starts, `print` a `JSON` representation of the theme being used for that game out to the console. All elements of the theme (its name, the emojis to choose from, how many pairs of cards to show and the color of the theme) must be included.

Hints

1. The easiest way to make any `struct` persistent is to use the `Codable` protocol. `Codable` lets you make any `struct` at all persistent, but it has fantastic built-in capabilities to *automatically* make `struct`s containing simple types (e.g. `Int`, `String`, `Double`, `CGFloat`, `URL`, `Array`, `Dictionary`, `Set`, etc.) persistent. So the best way to approach this assignment is to modify your theme struct to only contain simple types. It probably already is like that except for one notable exception, the theme's color …
2. You are likely storing your theme's color as a `Color`. As we learned in lecture, a `Color` is really just a way to *specify* a `Color` (or it can be a `View` or a `ShapeStyle` too). It's not truly a color representation. You'll want to switch to using `UIColor` in your theme. `UIColor` is an object that actually represents a specific color.
3. But even `UIColor` is not automatically `Codable`. So, we strongly recommend representing your color inside your theme as a `struct` with 4 floating point numbers: the color's red, green, blue and alpha (transparency) levels (aka `RGBA`). To aid you in this, we have included some simple code below that takes a `UIColor` and returns a `struct` with 4 such `CGFloat`s in it. Thankfully, `Codable` does know how to automatically encode/decode a `struct` with `CGFloat`s.
4. Since your theme `struct` is now going to represent its color internally as this `struct` of four `CGFloat`s, you'll want a way to reconstruct a `Color` or `UIColor` from this `struct`. The code below can do that as well. You'll probably want to provide some nice API in your theme so that other code in your app doesn't even have to know about this `RGBA` thing.
5. You're very likely to want the `.utf8` `var` we added in `EmojiArtExtensions.swift`.

UIColor+RGBA

```
extension Color {
    init(_ rgb: UIColor.RGB) {
        self.init(UIColor(rgb))
    }
}

extension UIColor {
    public struct RGB: Hashable, Codable {
        var red: CGFloat
        var green: CGFloat
        var blue: CGFloat
        var alpha: CGFloat
    }

    convenience init(_ rgb: RGB) {
        self.init(red: rgb.red, green: rgb.green, blue: rgb.blue, alpha: rgb.alpha)
    }

    public var rgb: RGB {
        var red: CGFloat = 0
        var green: CGFloat = 0
        var blue: CGFloat = 0
        var alpha: CGFloat = 0
        getRed(&red, green: &green, blue: &blue, alpha: &alpha)
        return RGB(red: red, green: green, blue: blue, alpha: alpha)
    }
}
```

Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. `Codable`
2. JSON Encoding

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Project does not build without warnings.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it.