Assignment II:

More Memorize

Objective

The goal of this assignment is to continue to recreate the demonstrations given in the next two lectures (first four lectures in total) and then make some bigger enhancements. It is important that you understand what you are doing with each step of recreating the demo from lecture so that you are prepared to do those enhancements.

This continues to be about experiencing the creation of a project in Xcode and typing code in from scratch. **Do not copy/paste any of the code from anywhere.** Type it in and watch what Xcode does as you do so.

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 watch Lecture 5.

Materials

Same materials as Assignment 1.

Required Tasks

1. Get the Memorize game working as demonstrated in lectures 1 through 4. Type in all the code. Do not copy/paste from anywhere.
2. Your game should still shuffle the cards.
3. Architect the concept of a "theme" into your game. A theme consists of a name for the theme, a set of emoji to use, a number of cards to show (which, for at least one, but not all themes, should be random), and an appropriate color to use to draw (e.g. orange would be appropriate for a Halloween theme).
4. Support at least 6 different themes in your game.
5. A new theme should be able to be added to your game with a single line of code.
6. Add a "New Game" button to your UI which begins a brand new game. This new game should have a randomly chosen theme. You can put this button anywhere you think looks best in your UI.
7. Show the theme's name somewhere in your UI.
8. Keep score in your game by giving 2 points for every match and penalizing 1 point for every previously seen card that is involved in a mismatch.
9. Display the score in your UI in whatever way you think looks best.
10. Your UI should work in portrait or landscape on any iOS device. The cards can have any aspect ratio you'd like. This probably will not require any work on your part (that's part of the power of SwiftUI), but be sure to continue to experiment with running on different simulators in Xcode to be sure.

Hints

1. Economy is still (and always) valuable in coding, but this week is not a collection of one-liners (though we do pay homage to the one-liner in Required Task 5).
2. Be careful to think carefully about what parts of your code are Model, what parts are ViewModel and what parts are the View. This is a significant part of this assignment. Remember, nothing about how the game is *displayed* should be in the Model (it's UI independent), but all of the mechanics about how the game plays should be.
3. One way to think of the number of cards in a theme is that if it is a number specified in the theme, then it's that number of (pairs of) cards, but if it's not specified somehow, then it's random (but always less than the number of emoji in the theme, of course).
4. If you think of it that way, what sort of type would be good to represent the number of cards in a theme? Swift has a great type for something that is normally specified, but sometimes is not.
5. Your New Game button is allowed to be a `Text` with an `onTapGesture` (since you know how to do that) or you can use a `Button` (see the documentation for how to use that). The difference is that a `Button` automatically adjusts itself to look like what a button is supposed to look like on the platform it is on (iOS, Apple TV, Apple Watch), whereas a `Text` is always going to look like a piece of text on all platforms. Because of this, `Button` is way better.
6. Example themes: animals (🐼🐔🦄), sports (🏀🏈⚾), faces (😀😢😉).
7. A card has "already been seen" only if it has, at some point, been face up and then is turned back face down. So tracking "seen" cards is probably something you'll want to do when you turn a card face down.
8. If you flipped over a 🐠 + 👻, then flipped over a ✏️ + 🏀, then flipped over two 👻s, your score would be 2 because you'd have scored a match (and no penalty would be incurred for the flips involving 🐠, ✏️ or 🏀 because they have not (yet) been involved in a mismatch, nor was the 👻 ever involved in a mismatch). If you then flipped over a 🐠 + 🐼, then flipped 🏀 + 🐠, your score would drop 3 full points down to -1 overall because the 🐠 had already been seen (on the very first flip) and subsequently was involved in two separate mismatches (scoring -1 for each mismatch) and the 🏀 was also involved in a mismatch after already having been seen (-1). If you then flip 🐠 + 🐠, you'd get 2 points for a match and be back up to 1 total point.

9. You are allowed to remove all of your code from assignment 1 (since the Required Tasks don't require you to preserve them). You especially will probably want to remove your aspect ratio code until you have the `Grid` working. But then it would be a great exercise to put it back and if it breaks your application, *go figure out why*. Remember that the order in which modifiers are applied to `Views` matters in SwiftUI. The slides in Lecture 3 that mention `aspectRatio` would be a great thing to review if you're having problems with this.

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. Reactive UI
2. Swift 5.2
3. Arranging UI with `HStack`, `VStack`, `Spacer`, `Divider`, etc.
4. Handling a tap gesture (or using `Button`)
5. MVVM (putting code in the proper place in your application)
6. Optionals

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. You can assume that the reader knows the iOS API and knows how the Memorize game code from lectures 1 and 2 works, but should not assume that they already know your (or any) solution to the assignment.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least some of these each week is highly recommended to get the most out of this course. How much Extra Credit you earn depends on the scope of the item in question.

If you choose to tackle an Extra Credit item, mark it in your code with comments so your grader can find it.

1. Support a gradient as the "color" for a theme. Hint: `fill()` can take a gradient as its argument rather than a color.
2. Modify the scoring system to give more points for choosing cards more quickly. For example, maybe you get max(10 - (number of seconds since last card was chosen), 1) x (the number of points you would have otherwise earned or been penalized with). (This is just an example, be creative!). You will definitely want to familiarize yourself with the `Date` struct.