Assignment 1: Memorize

Objective

The goal of this assignment is to recreate the demonstration given in the first two lectures and then make some small 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.

Mostly this is 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 start watching Lecture 3.

Materials

- In order to recreate the demo, you will certainly need to watch the first two lectures
- You will need to install the (free) program Xcode 11.4 using the App Store on your Mac (previous versions of Xcode will not work, though you could maybe give it a try with 11.3.1 and see how it goes). It is highly recommended that you do this immediately so that if you have any problems getting Xcode to work, you have time to get help from Piazza. Xcode 11.4 requires Catalina (macOS 10.15.4).
- If you are using an even newer version of Xcode, it may be that improvements to SwiftUI cause some of the lecture to not make sense anymore. You'll just have to adapt!

Required Tasks

1. Get the Memorize game working as demonstrated in lectures 1 and 2. Type in all the code. Do not copy/paste from anywhere.
2. Currently the cards appear in a predictable order (the matches are always side-by-side, making the game very easy). Shuffle the cards.
3. Our cards are currently arranged in a single row (we'll fix that next week). That's making our cards really tall and skinny (especially in portrait) which doesn't look very good. Force each card to have a width to height ratio of 2:3 (this will result in empty space above and/or below your cards, which is fine).
4. Have your game start up with a random number of pairs of cards between 2 pairs and 5 pairs.
5. When your game randomly shows 5 pairs, the font we are using for the emoji will be too large (in portrait) and will start to get clipped. Have the font adjust in the 5 pair case (only) to use a smaller font than `.largeTitle`. Continue to use `.largeTitle` when there are 4 or fewer pairs in the game.
6. Your UI should work in portrait or landscape on any iOS device. In landscape your cards will be larger (but still 2:3 aspect ratio). This probably will not require any work on your part (that's part of the power of SwiftUI), but be sure to experiment with running on different simulators in Xcode to be sure.

Hints

1. Economy is valuable in coding. The easiest way to ensure a bug-free line of code is not to write that line of code at all. Required Tasks 2, 3 and 4 (and possibly even 5) can each be done with an addition or change to **a single line of code** (this is a Hint, not a Required Task, so you are in no way required to do that).
2. We haven't implemented the "reactive" part of SwiftUI yet (i.e. when the View automatically updates every time the Model changes), so you'll be re-launching your application over and over in the simulator (and observing it in the Preview pane) as you test your code.
3. Shuffling the cards might be easier than you think. Be sure to familiarize yourself with the documentation for `Array`.
4. Be sure to do your shuffling in the proper place in your MVVM. Is shuffling a UI thing (i.e. View or ViewModel) or a Model thing?
5. Another area of the documentation you will want to become very familiar with is the documentation for the `View` protocol. There's a lot there (`View` is very powerful), and you are of course not required to understand every function in `View` (yet), but at least scanning/searching through it will both give you an idea of what's there and likely be helpful in solving your aspect ratio problem.
6. The function `Int.random(in: ClosedRange<Int>)` can generate a random integer in any range, for example, `let random = Int.random(in: 15...75)` would generate a random integer between 15 and 75 (inclusive).
7. You'll definitely need to look at the documentation for `Font`. You are welcome to use one of the other built-in fonts listed there (i.e. you don't have to call any of `Font`'s functions like `system` or `custom`). Just pick whichever built-in gives you the size you want since the font does not affect what an emoji looks like (other than size).
8. Required Task 5 can also be implemented in a single line of code. But you'll likely need to use Swift's "ternary" operator (e.g. `let x = truthTest ? foo : bar`) to do so. It is very common to use ternary operators to affect the arguments to modifiers of `Views`.
9. One thing we are not addressing in this assignment is that our emoji font looks **too small** in landscape. We'll be fixing this in a much more powerful way next week.

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. Xcode 11.4
2. Swift 5.2
3. MVVM
4. Looking things up in the documentation (`Array`, `Font` and `View`)
5. `Int.random(in:)`
6. Running your application in different simulators

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.

This week there's only one Extra Credit item, sorry! If we think of some more this week, we'll post them on the class forums.

1. Have the emoji on your cards be randomly chosen from a larger set of emoji (at least a dozen). In other words, don't always use the same five emoji in every game.