Assignment I: Concentration

Objective

The goal of this assignment is to recreate the demonstration given in lecture 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 the enhancements.

Another goal is to get experience creating 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.

This assignment must be submitted using the submit script described here by the start of lecture next Wednesday (i.e. before lecture 4). You may submit it multiple times if you wish. Only the last submission before the deadline will be counted. For example, it might be a good idea to go ahead and submit it after you have reproduced what was shown in lecture and gotten that part working (even before you attempt the enhancements). If you wait until the last minute to try to submit and you have problems with the submission script, you'll likely have to use one of your valuable free late days.

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.

Materials

- You will need to install the (free) program Xcode 9 using the App Store on your Mac (previous versions of Xcode will NOT work). 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 and/or the TAs in their office hours.
- A link to the video of the lectures can be found in the same place you found this document.

Required Tasks

1. Get the Concentration game working as demonstrated in lectures 1 and 2. Type in all the code. Do not copy/paste from anywhere.
2. Add more cards to the game.
3. Add a "New Game" button to your UI which ends the current game in progress and begins a brand new game.
4. Currently the cards in the Model are not randomized (that's why matching cards end up always in the same place in our UI). Shuffle the cards in `Concentration`'s `init()` method.
5. Give your game the concept of a "theme". A theme determines the set of emoji from which cards are chosen. All emoji in a given theme are related by that theme. See the Hints for example themes. Your game should have at least 6 different themes and should choose a random theme each time a new game starts.
6. Your architecture must make it possible to add a new theme in a single line of code.
7. Add a game score label to your UI. Score the game by giving 2 points for every match and penalizing 1 point for every previously seen card that is involved in a mismatch.
8. Tracking the flip count almost certainly does not belong in your Controller in a proper MVC architecture. Fix that.
9. All new UI you add should be nicely laid out and look good in portrait mode on an iPhone X.

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. This entire assignment (not including Extra Credit) can be done in a few dozen lines of code beyond what was shown in lecture (probably less than two dozen), so if you find yourself writing more than 100 lines of code, you are almost certainly on the wrong track.
2. Example themes: animals (🐼🐔🦄), sports (🏀🏈⚾️), faces (😀😢😉).
3. 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.
4. If you'd like an `Array` containing all the keys in a `Dictionary` (called `foo` in this example), use …

```
let fooKeys = Array(foo.keys)
```

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 9
2. Swift
3. MVC
4. `UIViewController` subclass
5. `UILabel` and `UIButton`
6. Target/Action (`@IBAction`)
7. Outlets (`@IBOutlet`) and Outlet Collections
8. `func`tions and properties (instance variables)
9. `let` versus `var`
10. Value type (`struct`, `enum`) versus reference type (`class`)
11. Strong typing and type inference
12. `didSet`
13. `for in` (and `..<` `CountableRange` syntax)
14. `Array<Element>` and `Dictionary<Key, Value>`
15. `[Element]` and `[Key:Value]` syntax
16. `init`ialization of `struct` and `class`
17. `viewDidLoad`
18. Optionals (including implicitly-unwrapped Optionals)
19. `??` optional defaulting operator
20. `// TODO`
21. `arc4random()`
22. Type conversion (e.g. from `UInt` to `Int`)
23. Stack View and (simple) autolayout

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 Concentration 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. Change the background and the "card back color" to match the theme. For example, our Halloween theme has a black background and orange card backs. Maybe a "winter" theme might have blue and white colors. A "construction" theme could be black and yellow. `UIViewController` has a property called `view` which is connected to the top-level view in the scene (i.e. the view that was black in lecture).
2. You can find out what time it is using the `Date` `struct`. Read the documentation to figure out how it works and then use it to adjust your scoring so that the more quickly moves are made, the better the user's score is. You can modify the scoring Required Task in doing this, but the score must still somehow be dependent on matches being rewarded and mismatches of previously-seen cards being penalized (in addition to being time-based). It's okay if a "good score" is a low number and a "bad score" is a high number.