Assignment 2: Matchismo 2

Objective

This assignment extends the card matching game Matchismo we started last week to get experience understanding MVC, modifying an MVC's View in Xcode, creating your own actions and outlets, interacting with `UILabel`, `UIButton` and other iOS 7 SDK elements, and generally getting more experience with Xcode and Objective-C.

Be sure to check out the **Hints** section below!

Materials

- You will need to have completed last week's assignment before starting this one.
- You will also need the lecture slides from Lecture 3.
- The slides for all lectures can be found in the same place you found this document.

Required Tasks

1. Follow the detailed instructions in the lecture slides (separate document) to reproduce the latest version of Matchismo we built in lecture (i.e. the one with multiple cards) and run it in the iPhone Simulator. Do not proceed to the next steps unless your card matching game functions as expected and builds without warnings or errors.
2. Add a button which will re-deal all of the cards (i.e. start a new game). It should reset the score (and anything else in the UI that makes sense). In a real game, we'd probably want to ask the user if he or she is "sure" before aborting the game in process to re-deal, but for this assignment, you can assume the user always knows what he or she is doing when they hit this button.
3. Drag out a switch (`UISwitch`) or a segmented control (`UISegmentedControl`) into your View somewhere which controls whether the game matches two cards at a time or three cards at a time (i.e. it sets "2-card-match mode" vs. "3-card-match mode"). Give the user appropriate points depending on how difficult the match is to accomplish. In 3-card-match mode, it should be possible to get some (although a significantly lesser amount of) points for picking 3 cards of which only 2 match in some way. In that case, all 3 cards should be taken out of the game (even though only 2 match). In 3-card-match mode, choosing only 2 cards is never a match.
4. Disable the game play mode control (i.e. the `UISwitch` or `UISegmentedControl` from Required Task 3) when a game starts (i.e. when the first flip of a game happens) and re-enable it when a re-deal happens (i.e. the Deal button is pressed).
5. Add a text label somewhere which desribes the results of the last consideration by the `CardMatchingGame` of a card choice by the user. Examples: "Matched J♥ J♠ for 4 points." or "6♦ J♣ don't match! 2 point penalty!" or "8♦" if only one card is chosen or even blank if no cards are chosen. Do not violate MVC by building UI in your Model. "UI" is anything you are going to present to the user. This must work properly in either mode of Required Task 3.
6. Change the UI of your game to have 30 cards instead of 12. See Hints about the size of cards.
7. Do not change the method signature of any public method we went over in lecture. It is fine to change private method signatures (though you should not need to) or to add public and/or private methods.

Hints

These hints are not required tasks. They are completely optional. Following them may make the assignment a little easier (no guarantees though!).

1. `NSString`'s `stringWithFormat:` method will be very helpful.
2. Don't forget that `NSString` constants start with `@`. Constants without out the `@` (e.g. `"hello"`) are `const char *` and are rarely used in iOS.
3. Think carefully about where the code to generate the strings in Required Task 5 above should go. Is it part of your Model, your View, or your Controller? Some combination thereof? Sometimes the easiest solution is a violation of MVC, but **do not violate MVC**! Justify your decision in comments in your code. A lot of what this homework assignment is about is your understanding of MVC. This aspect of this assignment is very often missed by students, so give it special attention!
4. You will have to read the documentation for `UISwitch` and/or `UISegmentedControl` to figure out how to use them. A switch is probably (a little bit) easier to understand, but a segmented control might be more appropriate to the task. Being able to figure a class out solely from its documentation is crucial to being a good iOS developer. That's what that Required Task is about.
5. The logic in your Model will have to be *configurable* for the two different game play modes. And your `PlayingCard` class will also have to know how to match itself against two other cards (it already knows how to match itself against one other card).
6. Often in computer science we talk about there only being 3 numbers: 0, 1 and n. The Matchismo we developed in class matches against 1 other card. You are being asked to match against 2 other cards, but perhaps (at least in your `CardMatchingGame` class, but maybe not in `PlayingCard`) you should choose to match against n other cards instead of just 2 other cards? This is **not** a Required Task, just something to consider.
7. You can feel free to adjust the scoring of 2-card-match mode if you want it to be consistent with your 3-card-match mode's scoring. In other words, consider the difficulty of matching 2 out of 2 cards of the same suit (medium) versus 2 out of 3 (easy) or 3 out of 3 (hard).
8. To make 30 cards fit, you'll have to make them smaller (oh, maybe 40x60 or so). The cards will strongly want to be their natural size (based on the background image sizes) while working with them in Xcode, so you'll have to hold down the Command key while resizing them.
9. Economy is valuable in coding: the easiest way to ensure a bug-free line of code is not to write the line of code at all. But not at the expense of violating MVC! This assignment requires more lines of code than last week, but still not an excessive amount. So if you find yourself writing many dozens of lines of code, you are on the wrong track.

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. Connecting outlets.
2. Connecting action methods.
3. Making Outlet Collection connections.
4. Gaining more experience using Xcode 5.
5. Understanding MVC (keeping UI out of Model).
6. Start to gain some experience laying out UI elements in a pleasing arrangement.
7. Using the documentation to learn a new class (e.g. `UISwitch` or `UISegmentedControl`).
8. Gaining more experience with Objective-C.
9. Challenging your programming skills a bit more (multi-card-matching/choice result).
10. Designing simple data structures (Extra Credit).
11. Loading and using images in Xcode (Extra Credit).

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.
- Project does not build without warnings.
- One or more items in the **Required Tasks** section was not satisfied.
- A fundamental concept was not understood.
- Code is 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, etc.
- Solution violates MVC.
- **UI is a mess. Things should be lined up and appropriately spaced to "look nice." Xcode gives you those dashed blue guidelines so there should be no excuse for things not being lined up, etc. Get in the habit of building aesthetically balanced UIs from the start of this course.**
- Assignment was turned in late (you get 3 late days per quarter, so use them wisely).

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 SDK, but should not assume that they already know the (or a) solution to the problem.

Extra Credit

Here is an idea for something you could do to get some more experience with the SDK at this point in the game.

1. Add a `UISlider` to your UI which travels through the history of the results of the currently-being-played game's card choices and display it to the user (moving the slider will modify the contents of the text label you created for Required Task 5 to show its state over the course of the game). When you are displaying past results, you probably want the text label to be grayed out or some such (take a look at the `UIView` method `alpha` and note that `UISlider` inherits from `UIView`) so it's clear that it's "the past." And every time a new choice happens, you probably want to "jump to the present" in the slider. Implementing this extra credit item will require you to familiarize yourself with `UISlider`'s API and to add a data structure to your Controller to keep track of the history. It can be implemented in fewer than a dozen lines of code.
2. Change the image on the back of the cards (i.e. something other than a Stanford logo). Also, set the application and launch image icons for Matchismo. Try to get the resolutions right for each application icon and launch image. Let your creativity run wild when it comes to designing an icon!