Assignment III: Graphical Set

Objective

The goal of this assignment is to gain the experience of building your own custom view, including handling custom multitouch gestures.

Start with your code in Assignment 2.

This assignment must be submitted using the submit script described here by the start of lecture next Wednesday (i.e. before lecture 8). You may submit it multiple times if you wish. Only the last submission before the deadline will be counted.

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 your implementation of Assignment 2.
- You may find this Grid object useful for determining the frames of your Set cards.

Required Tasks

1. Your application should continue to play a solo game of Set as required by Assignment 2.
2. In this version, however, you are not to limit the user-interface to a fixed number of cards. You should always be prepared to Deal 3 More Cards. The only time the Deal 3 More Cards button will be disabled is if there are no more cards left in the deck.
3. Do not "pre-allocate" space for all 81 possible cards. At any given time, cards should be as large as possible given the screen real estate available to cards and the number of cards currently in play. In other words, when the game starts (with only 12 cards), the cards will be pretty big, but as more and more cards appear on screen (due to Deal 3 More Cards), they will have to get smaller and smaller to fit.
4. Towards the end of the game, when 3 cards are matched and there are no more cards in the Set deck, the matching cards should be removed from the screen entirely and the remaining cards should "re-form up" to use the space freed up by these departing cards (i.e. getting a bit larger again if space allows).
5. Cards must have a "standard" look and feel (i.e. 1, 2 or 3 squiggles, diamonds or ovals that are solid, striped or unfilled and are either green, red or purple). You must draw them using `UIBezierPath` and/or CoreGraphics functions. You may not use attributed strings nor `UIImage`s to draw your cards.
6. Whatever way you draw your cards must scale with the size of the card (obviously, to support Required Task 3).
7. On cards that have more than one symbol, you are allowed to draw the symbols on horizontally across or vertically down (or even have that depend on the aspect ratio of the card at the time it is being drawn).
8. A tap gesture on a card should select/deselect it.
9. A swipe down gesture in your game should Deal 3 More Cards.
10. Add a rotation gesture (two fingers rotating like turning a knob) to cause all of your cards to randomly reshuffle (it's useful when the user is "stuck" and can't find a Set). This might require a modification to your Model.
11. Your game must work properly and look good in both Landscape and Portrait orientations on all iPhones and iPads. It should efficiently use all the space available to it in all circumstances.

Hints

1. You will not be able to use stack views or an outlet collection for your cards in this assignment because you have to be able to put a lot more cards on screen at the same time (as many as 81 theoretically) and you can't preallocate space for them in any case. The included `Grid` utility object can help make it really easy to calculate the `frames` for your cards (which you can set from code).
2. You are not required to use `Grid`. You can also feel free to modify it if you want (though you shouldn't have to).
3. While you won't be using autolayout or stack views to place your cards themselves (you'll be setting their `frames` directly in code), you will still use autolayout and/or stack views to place the view that *contains* all of your cards as `subviews` as well as all other UI (score label, Deal 3 More Cards button, etc.).
4. Note that the third kind of "filling" of cards is striping (not shading like last week). You might find clipping to a path with `addClip` useful in implementing this one.
5. When you `addClip`, all future drawing will be clipped (for the rest of your `draw()` method's execution). If you want to "undo" the clip, you'll want to wrap calls to the Core Graphics functions `saveGState()` and `restoreGState()` around the part of your drawing you want clipped. The function `saveGState()` saves the entire graphics state (including the clipping) at the time you call it, then `restoreGState()` returns back to that saved graphics state. You send these functions to the context you get from calling `UIGraphicsGetCurrentContext()`.
6. Were you able to use your Model untouched from last week (other than the new requirement of shuffling in response to a rotation gesture)? If not, why not? Understanding that may help with your understanding of MVC in general.

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. Creating a custom `UIView` with a `draw(CGRect)` method
2. Gestures
3. Understanding the `UIView` hierarchy
4. Creating `UIViews` in code (rather than in Interface Builder)
5. Drawing with Core Graphics and `UIBezierPath`

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.).
- Violates MVC.
- UI is a mess. Things should be lined up and appropriately spaced to "look nice."
- Improper object-oriented design including proper use of value types versus reference types.
- Improper access control (i.e. `private` not used appropriately).
- 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, 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. Make the game be a two-player Set game. No need to go overboard here. Maybe just a button for each user that can be used to claim they see a Set on the board, then that player has a certain (fairly short) amount of time to actually choose the Set or the other person gets a turn (with twice as much time to find a Set)? Most Sets found wins. Hitting Deal 3 More Cards on one user's side might give the other person a medium amount of time to find a Set without penalty if they can't? Up to you. You'll probably need `Timer` (as mentioned in last week's extra credit) to do this one.