Assignment IV: Animated Set

Objective

In this assignment you will add animation to your Set game and combine your first three assignments into one.

This assignment must be submitted using the submit script described here by the start of lecture a week from Monday (i.e. before lecture 11). 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 Assignments 1 and 3.

Required Tasks

1. Your application should continue to play a solo game of Set as required by Assignment 3 (with the caveats below).
2. You must animate the following actions in your set game:
   a. the rearrangement of cards. when cards are added or disappear from the game the cards should move smoothly (not jump instantly) to their new positions.
   b. the dealing of new cards. this includes both the initial 12 cards and any time 3 new cards are dealt. cards should fly across the screen from some "deck" somewhere on screen. the appearance of the deck is up to you. no two cards should be dealt at the same time though their animations can overlap a bit.
   c. the discovery of a match. matched cards should all fly away from where they were at the same time and bounce around on the screen for a couple of seconds before snapping to some "discard pile" somewhere on screen. the appearance of the discard pile is up to you.
   d. the flipping over of cards. cards should be dealt face down until they are in position, then they should be flipped over to reveal the set card contents and after cards have flown away to the discard pile, at least the top card on the discard pile should be flipped face down.
3. Your animation implementation must use `UIViewPropertyAnimator`, `UIDynamicAnimator`, and the `UIView` class method `transition(with:…)`. You will probably also need a `Timer`, but it's not strictly required.
4. Instead of using a swipe gesture to deal 3 more cards, allow users to deal 3 more cards by tapping on your deck.
5. Automatically perform a "deal 3 more cards" (i.e. simulate tapping on the deck) whenever a match is revealed.
6. You are not required to support your "rearrange cards" rotation gesture from the last assignment (see Extra Credit).
7. Add a theme-choosing MVC to your Concentration game in the same way that a theme chooser was added to Concentration in Lecture 7. This Required Task is essentially to reproduce Lecture 7, however, you'll be using your own Concentration (not the demo one) including your theme code. You are allowed to modify your Assignment 1 code if necessary. The point of this Required Task is to show us that you can do what was done with Multiple MVCs in the Lecture 7 demo.
8. Combine the above Set and Concentration games into a single application using a tab bar controller. This is the application you will submit.

9. 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. It is quite likely that all of your animations can originate from your `updateViewFromModel` equivalent. This is just a Hint, not a Required Task.
2. Whenever we want `layoutSubviews()` to get called immediately (rather than "at some time in the future that's convenient"), we don't call `layoutSubviews()`, we call `layoutIfNeeded()`. Note that this will only actually call `layoutSubviews()` to lay out the subviews if, since the last `layoutSubviews()` was called, a) the `bounds` of the view have changed, b) some subview has been added or removed, or c) someone called `setNeedsLayout()`. So feel free to call `layoutIfNeeded()` when you want a view's `layoutSubviews()` method to be called, but also be sure to call `setNeedsLayout()` any time you change anything in that view that would cause it to need to lay out its subviews again.
3. Here is a strategy you can use to attack your animation tasks. This is only a Hint, so you are welcome to ignore it completely, but if you are finding yourself a little at a loss for where to start, start here (after making sure you've read Hint #1) …
   - 3.a. Start off by simply animating the laying out of your cards. This is very easy to do with `UIViewPropertyAnimator` since laying out the cards only changes the `frame` property of those views which is one of the animatable properties. It's quite possible that you can implement this in 2 lines of code if you've properly implemented your view the holds your cards.
   - 3.b. Implement a placeholder for your "flyaway" animation by simply animating the transparency (`alpha`) of the matched cards down `0`. You'll use `UIViewPropertyAnimator` (for now) to do this, of course (since `alpha` is an animatable property), but eventually you'll be switching this animation to use `UIDynamicAnimator`.
   - 3.c. Once you have the "flyaway" fade working, implement a placeholder for your "dealing a card" animation by animating the fading of all cards with `alpha = 0` back to `alpha = 1`.
   - 3.d. You should not start this "dealing a card" animation until after your "rearrange the cards" animation has finished.
   - 3.e. You can even use this mechanism to "deal" out your initial 12 cards. Just put them out there with `alpha = 0` initially and your 3c animation should automatically "deal" them in.
   - 3.f. Now you have 3 of the 4 animations working (albeit in a primitive form).
   - 3.g. Remember that the "flyaway" and "deal" animations are prompted by different user actions. The "flyaway" happens when a card is touched that causes a 3-card match to be revealed. The "deal" animation is prompted when "deal 3 more

cards" happens or when the NEXT card AFTER a card that causes a 3-card match to be revealed is chosen.

3.h. Now go implement the real "deal" animation. All the cards that need to be dealt are already currently in the right position (but with `alpha = 0`, so the user can't see them). Since they're invisible, you can jump them over to your deck. Once they are there, set (no need to animate) their `alpha = 1` to make them reappear (hopefully your deck is drawn in a way that this won't be disconcerting having this card just appear out of nowhere). Now you can animate their `frame`s back to where they started (one by one).

3.i. If your deck is a different size than a card you're dealing, you can feel free to change the size of the card when you move it over to the deck (while its `alpha = 0` so the user won't see). Since you're animating it back using its `frame`, it will animate growing or shrinking to the proper size automatically.

3.j. Now get the "flyaway" animation working with `UIDynamicAnimator`.

3.k. The "flyaway" animation is a completely different animation that the "dealing" animation, so the timing of them does not have to be coordinated in any way. In fact, when you eventually implement automatically dealing 3 new cards, they'll start flying in at the same time the "flyaway" cards are bouncing around. Cool.

3.l. This also means that the `UIView`s that are used in the "flyaway" animation will have to be different than the `UIView`s being "dealt". It's probably best to create temporary cards for the flyaway, leaving the original cards behind with `alpha = 0` (which will cause the "dealing" animation to replace them on the next user interaction!).

3.m. Don't even try to implement the "automatic deal 3 new cards" until you have the above animations working. Once you have them working, this should be a trivial thing to implement. But if you don't have them working yet, it might just be confusing.

3.n. Once your `UIDynamicAnimator` settles down (and you then flip at least the top card over), you'll probably want to remove those cards from your `UIDynamicAnimator` so it doesn't waste its time trying to animate them anymore.

3.o. The flipping animation that `UIView.transition(with:…)` is doing is modifying the view's `transform` behind the scenes. Thus, you probably only want to do this on a view that is otherwise un-`transform`ed (i.e. it has the `identity` transform). You also don't want to be animating a view's `frame` at the same time you are animating its `transform`.

3.p. It might be easier to have your deck and discard pile just start out as fixed points on the screen at first, then upgrade them to being views later.

4. We haven't talked about doing autolayout in code (because it's a bit complicated), but there is a very easy "old style" autolayout mechanism if all you want to do is have a subview stay centered or pinned to the superview's edge(s). Just set the `autoresizingMask` `var` on the subview to, for example …

   ```
   [.flexibleTopMargin,.flexibleBottomMargin,.flexibleLeftMargin,.flexibleRightMargin]
   ```

   … if you want to keep the subview centered in its superview. You can read the documentation for `autoresizingMask` to find out more.

5. For code cleanliness, you'll probably want to collect all of your "flyaway" animations into a `UIDynamicBehavior` subclass.

6. It's perfectly legal to have a `UIViewPropertyAnimator` acting on a view at the same time that a `UIDynamicAnimator` is, but they should not be trying to animate the same property. Note that animating the `bounds` of a view is not the same as animating its `frame`. For example, you can animate the `bounds` of a view (basically, its size) with one animator at the same time a different animator is animating its `center`.

7. Note that you have a dynamic animation that starts off doing one thing (cards flying around) and then switches to doing another thing (snapping the cards to the discard pile). `Timer` might be useful to kick off the second half of this (although there are other ways to do it as well).

8. When your flyaway animation starts, you might well want the cards to be bouncing off each other, but when it comes time to snap them home to the discard pile, they definitely will not want to be bouncing off each other (since they have to stack up on each other). Dynamic animation is dynamic: you can change any aspect of any of the behaviors at any time and you can add or remove items from any behavior at any time and it will immediately react.

9. Your flyaway animation will look a lot cooler if you allow the cards to rotate as they bounce around. And you'll definitely want to adjust the elasticity of their collisions to get a pleasing amount of chaos.

10. Don't forget to always keep a strong pointer to your `UIDynamicAnimator` so that it does not leave the heap on you!

11. Your "flip the top card on the discard pile face down" animation has to happen after your flyaway animation finishes. You'll need `UIDynamicAnimator`'s `delegate` to detect that finishing.

12. There is an option in the Attributes Inspector for a view in InterfaceBuilder to either have a view clip its subviews to its `bounds` or not. Make sure you have that set correctly if you are trying to draw a subview outside the bounds of its superview (even if you are doing so via an animation).

13. As you do all of this, be careful to be operating in the right coordinate system. You have a lot of coordinate systems going on here. There is the coordinate system of your grid of cards, the coordinate system of your `UIDynamicAnimator`'s `referenceView`

(depending on which view you choose for that), the coordinate system of your deck and discard pile, etc. `UIView` has an awesome set of methods for converting points and rects between coordinate systems. Check out the many `convert(, to/from:)` methods in `UIView`.

14. Don't forget that the code in `prepare(for:sender:)` is executing before the outlets in the destination MVC have been set.

15. Depending on how you implement your animations, there is some small chance they might collide with the animations that happen when you rotate the device. Don't worry about this case for this assignment, but generally it's a good idea to design animations that head toward a position that always resolves to the same thing that calling `layoutSubviews()` does (since that's what happens when you rotate). Continuing animations sometimes have to pause or adjust themselves on a rotation so that, post rotation, all the items are on-screen and continuing to animate in a sensible way. Your application does not have any continuing animations (they are all transient), so hopefully this won't be a problem for you.

16. Don't forget the last Required Task!

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. `UIViewPropertyAnimator`
2. `UIDynamicAnimator`
3. `Timer`
4. `UIView.transition(with:duration:options:animations:completion:)`
5. `UINavigationController`
6. `UISplitViewController`
7. `UITabBarController`
8. Segues
9. 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.).
- 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 your rotation gesture from the last assignment work and be animated. You might get this "for free" if you implemented Required Task 2a well.
2. Take EC1 to another level by using some wacky animation (cards spinning or flying around or some such).
3. Make cards being dealt not only fly out, but also spin a couple of times. There is one trick to animating spinning (which you do with the view's `transform`). You have to make it be a chained animation where you spin 1/3 of the way around during the first step, then 1/3 more in the second step, and the final 1/3 in a third step. Why? Because if you just set your transform to be "rotate 2π", you'd be right back to where you started and thus no animation will happen because none will be necessary. Similarly, if you try to go halfway around and then the other half, it might well rotate back the way it came in the second half rather than continuing around to the end point. Remember that it is perfectly fine to have two view property animators going simultaneously if they are each animating different properties (e.g. `center` and `transform`). However, see Hint 3o above (you can't animate the `frame` and `transform` properties simultaneously). So if your card dealing animation is also, for example, animating the size of the view (because perhaps your deck is a different size than the cards sometimes), you'll have to either do that with the `transform` property (in your spin animations) or using the `bounds` property (which is only a transient animation property, so when your card arrives and stops spinning, you'd have to update its `frame`).
4. Make the name of the theme the user chooses in your Concentration game appear in the UI while the game is being played. On iPhone, this is easy since you're playing inside a `UINavigationController` and so you can just set your `ConcentrationViewController`'s `navigationItem`'s `title` to be the name of the theme. On the iPad, it's more difficult because the detail of the split view is not in a navigation controller and thus has no title bar. A simple solution to this is to embed the split view detail in a navigation controller too. But doing this will require you to modify your code because the detail will no longer be a `ConcentrationViewController`, it's going to be a `UINavigationController`. You'll have to use `UINavigationController` methods (probably `visibleViewController`) to "get at" the `ConcentrationViewController` inside to `prepare` it.

5. Create customized glyphs for your tab bar items. Apple's Human Interface Guidelines document covers the requirements for these.