Assignment IV: Animated Set

Objective

The goal of this assignment is to understand the animation mechanisms described in lecture this week by adding animation to your Set assignment from last week.

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 before lecture 11.

Materials

- You will need your code from last week's Set assignment.
- You can use any of the code from lecture as well.

Required Tasks

1. Your assignment this week must still play a solo game of Set.
2. In this version, though, when there is a match showing and the user chooses a card, do not *replace* the matched cards, instead, *remove* (aka discard) them (leaving fewer cards showing in the game).
3. Add a "deck" and a "discard pile" to your UI. They can be any size you want and you can put them anywhere you want on screen, but they should not be part of your main grid of cards and they should each look like a stack of cards (for example, they should have the same aspect ratio as the cards that are in play).
4. The deck should contain all the not-yet-dealt cards in the game. They should be "face down" (i.e. you should not be able to see the symbols on them).
5. The discard pile should contain all the cards that have been discarded from the game (i.e. the cards that were discarded because they matched). These cards should be face up (i.e. you *should* be able to see the symbols on the last discarded card). Obviously the discard pile is empty when your game starts.
6. Any time matched cards are discarded, they should be animated to "fly" to the discard pile.
7. You don't need your "Deal 3 More Cards" button any more. Instead, *tapping on the deck* should deal 3 more cards.
8. Whenever more cards are dealt into the game for any reason (except when you first launch your application), their appearance should be animated by "flying them" from the deck into place. Once you get this working cleanly for dealing out 3 more cards, it should "just work" when you hit the "new game" button.
9. Note that dealing 3 more cards (by tapping on the deck) when a match is showing on the board still should replace those matched cards and that those matched cards would be flying to the discard pile at the same time as the 3 new cards are flying from the deck (see Extra Credit for even more on this). This Required Task might not take any extra effort if you implemented the above Required Tasks cleanly.
10. All the card repositioning and resizing that was required by Required Tasks 2 and 3 in last week's assignment must now be animated. If your cards from last week never changed their size or position as cards were dealt or discarded, then fix that this week so that they do.
11. When a match occurs, use some animation (your choice) to draw attention to the match.
12. When a mismatch occurs, use some animation (your choice) to draw attention to the mismatch. This animation must be very noticeably different from the animation used to show a match (obviously).

13. Add a shuffle button to your UI and animate the shuffling.

Hints

1. Your deck and discard pile have to look like a stack of cards. It's okay if they look like an extremely neatly stacked pile of cards (like Memorize's stack of undealt cards did).
2. You will start to see the importance of properly separating code out between View, ViewModel and Model when you do animation. Remember, changes to the game should be happening in the Model only and the View should just be reflecting those changes. Animation is just causing these changes to appear over a brief period of time instead of instantly (as was happening in A3, assuming you had no animation there).
3. You likely will want to shuffle only already-dealt and not-yet-discarded cards or you might get unexpected results (depending on how you've implemented the rest of your code).
4. Remember that `matchedGeometryEffect` doesn't really work properly in the Preview. Use the simulator.
5. The animations for match and mismatch do not necessarily have to repeat forever like we did in Memorize (though that could be cool depending on the animation) and it does not necessarily have to involve the symbols (though it's very easy to imagine the symbols being involved in these animations).
6. If you do use a `repeatForever` animation for your mismatches, be careful to have your implicit `.animation` `ViewModifier` go back to specifying the `.default` animation when the card returns to *not* being involved in a mismatch. Otherwise your `repeatForever` will, indeed, repeat forever which you obviously don't want in that case.
7. The "back" of a card can look like whatever you want (except that it can't give away what's on the front of the card, of course!).
8. There is no requirement for cards to "flip over". See Extra Credit.
9. None of the Required Tasks require you to create your own `ViewModifier`, but you'd probably learn a lot by doing so if you can fit it into your solution.
10. Since you'll be using `matchedGeometryEffect` to transition your cards in and out of existence (as they pass from deck to playing field to discard pile), you probably won't need to use `.transition` for that but depending on how you animate your matched and unmatched cards, it might be something you would want to use there.

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. Animation

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 the lectures works, but should <u>not</u> 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.

If you choose to tackle an Extra Credit item, mark it in your code with comments so your grader can find it.

1. Have your deck and/or discard pile be either "sloppy" (i.e. not a perfectly neat stack) or show the first few cards slightly offset (so that it looks more like a stack).
2. When a new game starts, animate the dealing out of the initial 12 cards.
3. When you deal 3 more cards and there is a match showing, start animating the matched cards flying to the discard pile *before* the animation of the 3 new cards flying in from the deck starts. In other words, give the user a better impression of "I just replaced these 3 cards for you" by *delaying* the dealing animation a short bit in this scenario. The animations can still overlap, but delaying the dealing one just a little bit can result in a pleasing effect.
4. Make the cards that you deal out flip from face down (as they are in the deck) to face up (as they are once they are in play). You can use/modify the `.cardify` `ViewModifier` from lecture if you want.
5. Add any other animation you can think of that would make sense.