Assignment III: Set

Цель

Цель этого задания состоит в том, чтобы дать вам возможность создать свое первое приложение полностью с "нуля" и самостоятельно. Оно достаточно похоже на первые два Задания, которое помогло обрести вам уверенность, но достаточно отличающееся, чтобы дать вам полную свободу для накопления опыта!
НЕ называйте своё приложение Set (или что-то другое в вашем приложении именами, которые могли бы совпасть с ключевыми словами Swift типа struct или class). Например, имя "Set Game" вполне было бы подходящим.
Так как цель этого Задания - создать приложение с "нуля", **не начинайте с кода Задания 2, начинайте с New → Project в Xcode.**
Обязательно посмотрите ниже раздел подсказок **Подсказки (Hints)**!
Это задание - прекрасная возможность получить много дополнительных очков, если вы действительно хотите испытать себя.
Посмотрите также последние изменения в разделе **Оценка (Evaluation)**, чтобы вы понимали, что будет оцениваться в Домашнем Задании.

Срок сдачи

Это Домашнее Задание должно быть выполнено перед тем, как вы начнете смотреть Лекцию 7. Вы можете обнаружить, что это Задание значительно сложнее, чем предыдущие два.

Материалы

- Вы можете использовать любой код из Лекций (например, Grid).
- Вы захотите освежить в памяти правила игры Set.

Обязательные пункты задания

1. Реализуйте игру **Set** в версии соло (для одного игрока).

2. Когда ваша игра запускается в первый раз, карты на короткое время не должны отображаться, но как только они появляются, необходимо немедленно сдать 12 карт, заставляя их "прилетать" из случайных мест за пределами экрана.

3. В процессе игры эффективно используйте все пространство на экране. Карты должны становиться меньше (или больше) по мере того, как на экране одновременно появляется больше (или меньше) места, при этом всегда используется столько места, сколько доступно, и карты должны быть "красиво расположены". Все это делает Grid, и вы можете им пользоваться. Все изменения расположения и / или размеров карточек должны быть анимированы.

4. Карты могут иметь любое соотношение сторон (aspect ratio), которое вам нравится, но все они должны всегда иметь одно и то же соотношение сторон (независимо от их размера и независимо от того, сколько карт отображается на экране одновременно). Другими словами, карты, появляющиеся перед пользователем могут становиться большего и меньшего размера по мере того, как игра продолжается, но карты не могут "растягиваться" до других соотношений сторон (aspect ratio) во время игры.

5. Символы на картах должны быть пропорциональны размеру карты (т.е. большие карты должны иметь большие символы, а меньшие карты должны иметь меньшие символы).

6. Пользователи должны иметь возможность выбрать до 3 карт, прикоснувшись к ним, чтобы попытаться создать **Set** (т. е. 3 совпадающие карты (**matching**) в соответствии с правилами игры Set). Пользователю должно быть ясно видно, какие карты уже были выбраны.

7. После того, как были выбраны 3 карты (**selected**), вы должны показать, совпадают ли эти 3 карты (**match**) или нет (**mismatch**). Вы можете показать это как хотите (цветом, границами, фоном, анимацией, чем угодно). Каждый раз, когда выбраны 3 карты (**selected**), пользователю должно быть ясно, совпадают (**match**) они или нет

(**mismatch**) (и карты, входящие в не совпавшее трио, должны выглядеть иначе, чем карты, когда в случае выбора 1 или 2 карт).

8. Поддержите "отмену выбора" ("**deselection**"), путем повторного касания уже выбранных карт (но только если в данный момент выбраны 1 или 2 карты (не 3)).

9. Если вы касаетесь любой карты, когда уже выбраны 3 совпадающие (**matching**) карты, образующие **Set**, тогда …
    a. согласно правилам игры Set, замените эти 3 совпавших (**matching**) **Set** карты новыми из колоды
    b. совпадающие (**matching**) карты должны улетать (с анимацией) в случайные места за пределами экрана
    c. заменяющие карты должны прилететь (с анимацией) из других случайных мест за пределами экрана (или из "колоды" где-нибудь на экране, см. Дополнительные пункты)
    d. если колода пуста, то место, освобожденное совпадающими картами (которые не могут быть заменены), должно быть доступно для оставшихся карт (т.е. они, вероятно, станут больше)
    e. если карта, которую вы коснулись, не была частью совпадающих (**matching**) карт, образующих **Set**, выберите эту карту

10. Когда выбрана новая карта и есть уже 3 выбранных (**selected**) и не совпавших карты, сделайте эти 3 не совпавших карты не выбранными (**deselected**), а новую карту выбранной (**selected**) (независимо от того, была ли она частью не совпавшего трио карт).

11. Вам необходимо также иметь кнопку "Deal 3 More Cards" (Сдай еще 3 карты) (согласно правилам игры Set).
    a. при касании этой кнопки замените выбранные карты, если выбранные карты составляют **Set** (с анимацией прилета /улета, как описано выше)
    b. или, если выбранные карты не составляют **Set** (или если выбрано менее 3 карт, в том числе и ни одной), организуйте прилет (т. е. анимируйте прибытие) 3 новых карты, чтобы присоединиться к уже имеющимся на экране (и не делайте их выбранными (**selected**))
    c. Отключите эту кнопку, если колода пуста.

12. У вас также должна быть кнопка "Новая игра", которая запускает новую игру (т. е. происходит возврат к 12 случайно выбранным картам). Карты также должны прилетать и улетать, когда это происходит.

13. Чтобы немного упростить себе жизнь, вы можете заменить "волнистую линию" ("squiggle") в игре Set на прямоугольник.

14. Вы должны создать свою собственную структуру Shape для ромба (diamond).

15. Еще одно облегчающее жизнь изменение заключается в том, что вы можете использовать полупрозрачный цвет для представления "штриховки" в качестве заливки ("striped" shading) символа. Обязательно выберите подходящий уровень прозрачности, который четко отличит "штриховку" от "закрашивания" ("solid") символа.

16. Вы можете использовать любые 3 цвета, если они четко отличаются друг от друга.

17. Вы должны использовать перечисление enum как значимую часть вашего решения..

18. Вы должны использовать замыкание (т.е. функцию в качестве аргумента) как значимую часть вашего решения.

19. Ваш пользовательский интерфейс должен работать в портретной или ландшафтной ориентации на любом устройстве iOS. Это, вероятно, не потребует от вас какой-либо работы (это часть мощи SwiftUI), но обязательно поэкспериментируйте с запуском на разных симуляторах в Xcode, чтобы быть уверенным.

Подсказки (Hints)

1. Не стесняйтесь использовать "сетку" Grid для раскладки карт, если хотите. Однако вы не обязаны этого делать. Вы также можете изменить Grid, если хотите, но в этом нет необходимости.

2. Убедитесь, что вы четко представляете, что находится в вашей **Model**, что в вашей **ViewModel** и что находится в вашем **View**. Всегда спрашивайте себя: "Это о том, как играть в игру Set или о том, как она представлена пользователю на экране?"

3. Ваша **Model** должна четко отображать статус всех карт, которые есть или когда-либо были в колоде. Постарайтесь создать согласованный **API** для вашей **Model**.

4. При любом компромиссе между **View** и **ViewModel**, делайте более простым **View**.

5. Ваша **Model** на самом деле не требует сложных компромиссов, потому что она просто пытается представить интерфейс программирования (**API**), не зависящий от пользовательского интерфейса, который максимально эффективно играет в игру Set. **ViewModel** должна адаптироваться к дизайну вашей **Model** (если ваша **Model** хороша, это не должно быть слишком сложно для вашей **ViewModel**).

6. Не забывайте, что **View** всегда является отражением **Model**. Это "реактивное", "декларативное" программирование пользовательского интерфейса. **Model** изменяется, а **View** просто объявляется как нечто, полностью основанное на текущем состоянии **Model** (доступ к которому **View**, конечно, осуществляет через **ViewModel**). Постарайтесь освободиться от "императивной" стиля программирования, на котором вы, вероятно, выросли (т.е. вы вызываете функцию, и что-то происходит, а затем вы вызываете другую функцию, и происходит что-то еще, и т. д.). Интерфейс в **SwiftUI** мы делаем не так.

7. Помните, что любая происходящая анимация просто показывает пользователю то, что уже произошло. Также обратите внимание, что вам не нужно будет делать какие-либо "анимации того, что должно произойти в ближайшее время", как мы вынуждены были сделать для нашего "пирога" **Pie** в **Memorize**. Все анимации в этом приложении Set прямо показывают, что уже произошло (например, карты были сданы, или карты были выбраны, или карты "совпали" и сброшены).

8. Вероятно, было бы хорошим дизайном **MVVM** не встраивать "ориентированные на отображение" вещи, такие как цвета или даже имена символов (ромб, овал, прямоугольник) и видов заполнения символов (пустой, заштрихованный, закрашенный) в вашу **Model**. Представьте, что у вас есть темы для игры **Set**, как и для **Memorize**. Помните, что ваша **Model** почти ничего не знает о том, как игра будет представлена пользователю. Почему бы не **Set** карт с пингвинами?

9. Вот вам некоторая помощь с "летающими" картами …
    a. Все обязательные пункты этого Задания для "летающих" карт просто связаны с **Views**, представляющих карты, которые "приходят и уходят" из UI.
    b. "Приходы и уходы" **Views** в **SwiftUI** анимируются с помощью transitions (как описано в лекции).
    c. "Улет (прилет)" - это просто перемещение. Перемещение от того места, где они в конечном итоге будут (или были) на экране, в какое-то случайное место за пределами экрана. Так что вам нужно иметь простую функцию где-нибудь в вашем приложении, которая вычисляет случайное местоположение за пределами экрана (куда карта может улететь или откуда прилетать).
    d. В **SwiftUI** есть идеальный **transition** для перемещения под названием **AnyTransition.offset** (CGSize) (который анимирует тот же **ViewModifier**, что и модификатор **.offset** в **View**).
    e. Убедитесь, что вы помещаете модификатор **.transition** в **View**, которое действительно "приходит на экран и уходит с экрана", а, например, не в какой-то стек Stack, внутри которого находится этот **View**.
    f. Карты не будут "приходить на экран и уходить с экрана" должным образом, если ваша **Model** не будет четко представлять, какие карты в настоящее время участвуют в игре. Карта уже сдана? Карта уже "совпала и сброшена"? Подобные вещи должны быть где-то в вашей **Model**, иначе пользовательский интерфейс (**UI**) не будет знать, какие карты должны быть на экране в данный момент.
    g. Ваш **View**, как всегда, просто отображает состояние **Model**. Так, например, массив **Identifiables** в **Grid** (при условии, что вы используете **Grid** для отображения своих карт) не будет включать карты, которые ещё не были сданы или которые уже "совпали" и сброшены.
    h. Помните, что **transition** анимация не происходит, если вы не будете ее явно анимировать. Таким образом, вам потребуются любые действия пользователя,

которые могут вызвать явную анимацию летающих карточек с помощью **withAnimation**.

i. Не забывайте об **.onAppear**. Это может быть очень полезно для запуска игры, когда ваш пользовательский интерфейс впервые появляется на экране.

10. Будьте осторожны при тестировании "конец игры" (т.е. когда закончится колода). Чтобы упростить тестирование, возможно, вы захотите временно обхитрить вашу Model, заставив ее думать, что любые 3 карты "совпадают" в режиме тестирования - так вы сможете быстро добраться до конца игры. Или протестируйте с неполной колодой.

11. Не забудьте установить надлежащий уровень доступа для всех ваших **vars** и **funcs**.

12. Это Задание не требует, чтобы вы добавляли пользовательскую анимацию в пользовательский **ViewModifier** или **Shape** (то есть **Shape**, которую вы должны написать для этого Задания, не имеет пользовательской анимации, и вам не нужно писать свой собственный **ViewModifier**).

13. Когда вы используете ТИП **some View** для вычисляемой переменной **var** или в качестве возвращаемого типа функции **func**, помните, что вы просите компилятор **Swift** заглянуть внутрь вашего кода, выяснить, какой ТИП фактически возвращается, и эффективно заменить **some View** этим ТИПом (и, пока это происходит, убедитесь, что этот ТИП реализует протокол **View**). Это означает, что то, что возвращается, должно *определяться во время компиляции* (а не во время выполнения). Вот почему "условность" при построении **Views** выполняется с помощью **ViewBuilder** и вот почему мы не можем выполнять операторы **if-else** или **switch** на верхнем уровне функции, которая возвращает **some View**. Поэтому не забудьте использовать **ViewBuilder**, если вы возвращаете **some View** и внутри него есть **if-elses** (**ViewBuilder** не полностью поддерживает **switch** (по крайней мере, на момент написания), поэтому вам, вероятно, придется использовать **if-else**. Хотя в iOS 14 **ViewBuilder** полностью поддерживает **switch** и **if let**).

14. У вас также может возникнуть соблазн вернуть **some Shape** из функции. Это практически никогда не делается, потому что *не существует такой вещи*, как **ShapeBuilder** (то есть что-то вроде **ViewBuilder** для **Shapes**). Вместо этого передайте информацию, необходимую для обводки (*stroke*) и / или закрашивания

(*fill*) **Shape**, и верните **some View** из этой функции.

15. Это в значительной степени "середина" этого курса, поэтому потратьте время, чтобы убедиться, что вы действительно понимаете все, что было до сих пор. Это приложение на самом деле не требует от вас использования чего-либо совершенно нового. Если вы не можете заставить это приложение работать, у вас наверняка возникнут проблемы с финальным проектом.

Что нужно изучать

Это частичный список концепций, в которых это задание увеличивает ваш опыт работы или демонстрирует ваши знания.

1. Все, что было в Задании 1 и 2, но на этот раз с нуля.
2. Управление доступом (Access Control).
3. Геометрическую фигуру Shape.
4. Анимацию.
5. Перечисление **enum**.
6. Замыкания (Closures).

Оценка (Evaluation)

Во всех заданиях этого семестра требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений, следовательно вы должны тестировать полученное приложение до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено:

- Приложение не создается.
- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания.
- Приложение не создается без предупреждений.
- Код - небрежный или тяжелый для чтения (например, нет отступов и т.д.).
- Ваше решение тяжело (или невозможно) прочитать и понять из-за отсутствия комментариев, из-за плохого наименования методов и переменных, из-за непонятной структуры и т.д.

- Часто студенты спрашивают: "Сколько комментариев кода нужно сделать?" Ответ - ваш код должен легко и полностью быть понятным любому, кто его читает.

Дополнительные пункты (Extra Credit)

Мы постарались создать Дополнительные задания так, чтобы они расширили ваши познания в том, что мы проходили на этой неделе. Попытка выполнения по крайней мере некоторых из них приветствуется в плане получения максимального эффекта от этого курса..

1. Ваши карты прилетают из **View** колоды карт, расположенной где-нибудь на экране. Это немного сложнее, чем полет из / в случайное место, потому что выяснение того, где находится какой-то другой **View**, потребует некоторого исследования с вашей стороны.

2. Нарисуйте настоящую "волнистую линию" (squiggle) вместо прямоугольника.

3. Когда карты совпадают, обеспечьте захватывающую анимацию. Другими словами, используйте анимацию, чтобы показать, что карты совпадают в Обязательной задаче 7.

4. Сделайте "улет" карт более увлекательным (может быть, карты крутятся, когда улетают?).

5. Как-нибудь ведите счет в своей игре **Set**. Вы можете решить, какого рода начисление очков будет наиболее разумным.

6. Давайте больше очков игрокам, которые выбирают "совпадающие" **Sets** быстрее (т. е. включите временной компонент в вашу систему подсчета очков).

7. Придумайте, как наказать игроков, выбравших Deal 3 More Cards ("Сдать еще 3 карты"), когда на самом деле можно было выбрать **Set**.

8. Добавьте кнопку "мошенничество" в свой интерфейс. Если вы напишете алгоритм определения **Sets**, то сможете добавить "мошенническую" кнопку для того, чтобы испытывающий трудности пользователь смог использовать ее для нахождения **Set**!

9. Поддержите игру двух игроков. Здесь не нужно перегибать палку. Может быть, просто кнопка для каждого пользователя (может быть, одну перевернутую "вверху ногами" кнопку в самом верху экрана?), чтобы заявить, что они видят **Set** на игровом поле. Затем этот игрок получает (довольно короткое) количество времени, чтобы на самом деле выбрать **Set**, или другой человек получает столько времени, сколько они хотят попытаться найти **Set** (или, может быть, у них будет более длительное, но не неограниченное количество времени?) , Может быть, нажатие кнопки Deal 3 More Cards ("Сдать еще 3 карты") одним пользователем дает другому некоторое среднее количество времени, чтобы выбрать **Set** без штрафа? Вам нужно будет выяснить, как использовать таймер для выполнения этих ограниченных по времени вещей.

.