Assignment III: Графический калькулятор (Graphing Calculator)

Цель:

Вы должны усовершенствовать свой калькулятор Calculator в плане создания графика для “программы”, которую пользователь ввел в ваш калькулятор. Этот график может масштабироваться (zoom in) с помощью жеста pinch и перемещаться по экрану с помощью жеста pan. Ваше приложение теперь будет работать не только на iPhone, но также и на iPad.

Материалы

- У вас должно быть успешно законченное Задание 2. Задание 3 выполняется на его основе. Вы можете модифицировать существующий проект или создать новый проект (и повторно использовать написанные вами классы путем “перетягивания” их в новый проект). В любом случае убедитесь, что у вас есть копия прошлой работы прежде, чем приступать к выполнению Задания 3.
- Этот AxesDrawer класс будет вам очень полезен!

Обязательные пункты задания

1. Вы должны начать это Задание с кода вашего Задания 2, а не с какой-то демонстрации, находящейся на сайте. Изучение создания новых MVCs и **segues** требует опыта, поэтому не используйте copy / paste или редактирование уже существующей **storyboard**, на которой уже есть **segues**.

2. Переименуйте класс **ViewController**, с которым вы работали в Задании 1 и 2 в **CalculatorViewController**.

3. Добавьте новую кнопку к вашему пользовательскому интерфейсу калькулятора. Если вы ее нажимаете, то вы segues (“переезжаете”) на новый MVC (его вы должны будете написать). Этот MVC строит график для программы **program** в **CalculatorBrain** , сформированной во время нажатия кнопки, с использованием размещенной в стэке **M,** как независимой переменной. Например, если **CalculatorBrain** содержит **sin(M)**, то вы рисуете синусоиду. Последующий ввод информации в калькулятор не должен оказывать эффект на график (до тех пор, пока графическая кнопка не будет снова нажата).

4. Никакому из ваших MVCs в этом Задании не разрешается иметь в **non-private API CalculatorBrain**.

5. На **iPad** и в **ландшафтном** режиме на **iPhone 6+** устройствах, график должен быть (или может быть) на экране одновременно с вашим пользовательским интерфейсом существующего калькулятора (например, в **Split View**). На других **iPhones** график следует “выталкивать” (“push”) его на экран через **Navigation Controller**.

6. В любое время, пока график находится на экране, должно быть также показано описание (description) того, что рисуется на графике, например, если строится график **sin**(**M**), то строка “**sin**(**M**)” должна показываться где-то на экране.

7. Как часть вашей реализации (implementation), вам нужно написать обощенный (**generic**) **y (x)** графический **UIView**. Другими словами, **UIView**, который рисует графики, должен быть сконструирован таким способом, что он является полностью независимым от калькулятора (и мог бы использоваться в других совершенно различных приложениях для рисования графика **y (x)**).

8. Графический **V**iew **не должен владеть** (то есть запоминать) **данные**, представляемые графически. Он должен использовать **делегирование** для получения данных в случае необходимости.

9. Ваш графический калькулятор должен обладать способностью графически представлять разрывные свойства функций (то есть он должен рисовать линии только от/ к точкам, которые для фиксированного значения **M** в **program**, задающей рисование графика, оцениваются как **Double** (то есть не **nil**), что соответствует .**isNormal** или .**isZero**).

10. Ваш графический View должен быть **@IBDesignable**, и его масштаб должен быть **@IBInspectable**. Оси на графическом **V**iew должны появиться на **storyboard** в инспектируемом (inspected) масштабе.

11. Ваш графический **V**iew должен поддерживать следующие жесты:
    a. **Pinching** (изменение масштаба, zooming, целого графика, включая оси, в / за пределами графика)
    b. **Panning** (перемещение целого графика, включая оси, вслед за передвижениями пальцев по экрану)
    c. **Double-tapping** (перемещение **origin** графика в точку, где вы дважды “тапнули”)

Подсказки (Hints)

1. Не забывайте устанавливать класс для **UIViewController** или пользовательского **UIView** в **Инспекторе Идентичности** (Identity Inspector) в **Xcode** - это общая ошибка. Вам придется это сделать при переименовании **ViewController** в **CalculatorViewController**, а также для новых **UIViewController** и **UIView**, которые будут созданы в этом Задании.

2. Для того, чтобы существенно облегчить рисование графика, вам предоставляется класс (**AxesDrawer**), который рисует оси графика в текущем “рисующем” контексте . Заметьте, что “рисующий” оси метод класса (**drawAxesInRect**) берет в качестве аргументов **bounds** (границы), в которых осуществляется рисование, и еще два аргумента: **origin** и **pointsPerUnit** (это существенно для “масштаба” (“**scale**”) графика). Вы захотите иммитировать это (то есть иметь переменные vars для origin и scale) в вашем **generic** графическом **V**iew.

3. Кроме переменной **var program** из лекции 5, ваш **CalculatorBrain** не нуждается в каких-либо изменениях в этом Задании. Конечно, вам нужно усовершенствовать **var program**, чтобы она вобрала в себя все требования Задания 2.

4. Это предлагаемый порядок разработки … Заставьте ваш существующий **CalculatorViewController** работать внутри **Split View Controller** и **Navigation Controller** с графической кнопкой, которая **segues** (осуществляет “переезд”) к новому пустому MVC (вначале) с подходящим **UIViewController** subclass. Добавьте ваш **generic** графический **V**iew к новому MVC. Заставьте графический **V**iew рисовать по крайней мере оси графика. Добавьте жесты (**gestures**). Наконец, заставьте ваш новый MVC графически отображать **program**, которая находится в вашем главном MVC в момент нажатия графической кнопки. Вам необязательно следовать такому порядку, но это поможет вам организовать вашу работу.

5. **UIViewController** **subclass** для вашего нового MVC (того, который рисует график программы, находящейся в калькуляторе) и **generic** графический **UIView**

subclass - это единственные новые классы, которые вам придется написать “с нуля” в этом Задании. Если вы думаете, что вам придется написать еще и другие классы, то это, возможно, уже чрезмерно.

6. Убедитесь, что вы ясно понимаете, что является Моделью для вашего нового MVC.

7. Не сердитесь, когда вы вытянете **Split View Controller** из Палитры Объектов, а он принесет с собой другие дополнительные **View Controllers**. Это просто **Xcode** пытается вам помочь. Для большей безопасности удалите их, и используйте **CTRL**-перетягивание для “подцепления” ваших MVCs (внутри **Navigation Controllers**) к **Split View Controller** в нужных местах.

8. Было бы прекрасно, если бы **origin** для вашего графика по умолчанию разместился бы в середине **UIView**. Но будьте внимательны где/ когда вы рассчитываете это, потому что границы (**bounds**) вашего **UIView** не будут установлены до тех пор, пока **UIView** не расположится на устройстве. Вы можете быть уверены, что границы (**bounds**) вашего **UIView** уже установлены в вашем **drawRect**. Но, конечно, будьте внимательны, чтобы не переустановить заново **origin**, если он уже был кем-то установлен.

9. Прекрасным местом для вашего MVC, где можно установить себя в качестве **DataSource** делегата для вашего графического **V**iew, является **Наблюдатель Свойства (property observer) для вашего outlet** к графическому **V**iew.

10. Не переусложните ваш **drawRect**. Просто делайте итерации по каждой пикселе (**pixel**) (не **point**) по всей ширине (**width**) вашего **V**iew, и рисуйте **линию** (или просто перемещайтесь, если последняя точка (datapoint) была не “правильной”) к следующей “правильной” точке (datapoint), которую вы получаете из **Data Source** делегата.

11. Система координат, в которой вы рисуете внутри вашего **drawRect** не та же самая, что система координат вашего **DataSource**, обеспечивающего данными ваш график (из-за **origin** и **scale**). Проясните для себя при написании кода, какая из этих двух систем координат является переменной **var** или аргументом функции.

12. **AxesDrawer** также знает, как рисовать по **пикселям** (не **point**) границы (**boundaries**) (подобно как **drawRect** должна это делать), но только, если вы скажите ей **contentScaleFactor** для контекста рисования, на котором вы рисуете.

13. Не забывайте использовать **Наблюдателя Свойства (didSet)**, чтобы заставить ваше **V**iew заметить, что оно нуждается в перерисовке, когда свойство,

воздействующее на него, изменяется.

14. Убедитесь, что вы правильно установили свойство **UIViewContentMode** (это должно быть сделано на **storyboard**).

15. Ваши жесты, возможно, будут обрабатываться графическим **V**iew, но они захотят быть “подключенными” вашим **Controller**. По этой причине методам, которые обрабатывают (handle) жесты, не следует быть **private** в вашем графическом **V**iew.

16. Помните, что когда вы определяете **action**, который будет обрабатывать жест, как часть созданного распознавателя жестов, **gesture recognizer**, то если обработчик имеет аргумент, он должен иметь знак **“:”** в конце своего имени.

17. Ваша переменная **var description** в **CalculatorBrain** дает описание *целого стэка операндов*, но реально вы представляете на графике только последнее выражение, введенное в калькулятор, поэтому вам необходимо что-то сделать с этим, если вы хотите формировать корректный заголовок того, что показывается графически.□

Что нужно изучать

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

- Понимание boundaries (границ) MVC
- Создание нового subclass для UIViewController
- Универсальное приложение (то есть различные пользовательские интерфейсы (UI) для iPad и iPhone в одном и том же приложении)
- Split View Controller
- Navigation Controller
- Segues
- Property List
- Subclassing UIView
- UIViewContentMode.Redraw
- Делегирование
- Рисование с помощью UIBezierPath и/или Core Graphics
- CGFloat/CGPoint/CGSize/CGRect
- Gestures (жесты)
- contentScaleFactor (pixels и points)

Оценка (Evaluation)

Во всех заданиях требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений (without warnings or errors), следовательно вы должны тестировать полученное приложение (application) до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено (marked down):

- Приложение не создается (Project does not build).
- Приложение не создается без предупреждений (Project does not build without warnings).

- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания (A fundamental concept was not understood).
- 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. Вы можете предполагать, что читатель знает SDK, но вам не следует предполагать, что он уже знает решение проблемы.

Дополнительные задания (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. Поймите, как работают **Instruments**, анализируя производительность жестов **panning** и **pinching** в вашем графическом **V**iew. Что делает перемещение графика по экрану таким “вялым”? Объясните в комментариях к вашему коду, что вы обнаружили и что с этим можно сделать.

2. Используйте информацию, которую вы обнаружили в 1-ом дополнительном пункте, для улучшения производительности жеста **panning**. **НЕ** превращайте ваш код в “месиво”, выполняя это. Ваше решение должно быть простым и элегантным. Есть сильное искушение при оптимизации принести в жертву читабельность кода или преступить границы MVC, но вам **НЕ** разрешено это делать для этого дополнительного пункта!

3. Сохраните **origin** и **scale** между запусками вашего приложения. Где это следует сделать, оказывая наибольшее уважение MVC, как вы думаете?

4. При вращении прибора (или другом измении границ (**bounds**)), разместите **origin** вашего графика по отношению к центру графического **V**iew, а не по отношению к верхнему левому углу.

5. Добавьте **Popover** к вашему новому MVC, который будет сообщать о минимальном и максимальном **y**-значениях (и еще какую-нибудь статистику, если хотите) в области графика, показываемого в данный момент. Это потребует от вас создание еще одного MVC и **segue** к нему, используя **popover segue**. Это также потребует некоторого нового **public API** в вашем **generic** графическом **V**iew для предоставления статистики об области, в которой нарисован график.

6. Сделайте калькулятор, который реагирует на изменение **size class** путем совершенно различного расположения элементов на пользовательском интерфейсе в среде различных **size class** (например, кнопок в различной сетке или даже добавлением различного количества операций в одном или другом **size class**). Когда вы делаете это, то ничего не должно ломаться!