Assignment III: Graphing Calculator

Ziele

In diesem Assignment erweitern Sie Ihren Taschenrechner, so dass er einen Graphen erstellen kann, von dem, was der User in Eingegeben hat. Weiterhin unterstützt der Graph Zoom und Pan. Die App soll nun nicht nur auf iPhones, sondern auch iPads laufen.

Material

- Sie müssen erfolgreich Assignment II beendet haben. Dieses Assignment baut darauf auf. Sie können versuchen Ihr existierendes Programm zu modifizieren oder ein neues Projekt zu erstellen (und die bereits erstellen Klassen wieder zu verwenden). In jedem Fall sollten Sie eine Kopie von Assignment II erstellen.
- Die bereitgestellte Klasse zum zeichnen von Achsen soll Ihnen helfen.

Aufgaben

1. Ändern Sie den Namen der `ViewController` Klasse aus Assignment I und II in `CalculatorViewController`.
2. Erstellen Sie einen vollständig neuen MVC, welcher eine beliebige x-y Funktion graphisch darstellt. Er kann jede beliebige public API haben die Sie möchten (mit den Einschränkungen von unten). Wählen Sie Ihr Model sorgfälltig aus.
3. Ihr neuer graphischer MVC muss vollständig generisch und wiederverwendbar sein. Er darf nirgends in seiner Implementierung Calculator-bezogen sein (nicht in seinem Model, nicht in seinem Controller und nicht in seinem View).
4. Als Teil Ihrer Implementierung dieses neuen MVCs, müssen Sie ebenfalls eine *generische*, *wiederverwendbare* x-y grapische Subclass von `UIView` schreiben.
5. Der graphische View darf die Daten die er darstellt nicht besitzen (d.h. speichern), auch nicht temporär. Er muss nach den Daten, die er braucht fragen und zwar Punkt für Punkt. Ihr graphischer View zeichnet eine x-y Funktion, er zeigt kein Array von Punkten, also übergeben Sie auch kein Array von Punkten.
6. Fügen Sie einen neuen Button zum User-Interface des Taschenrechners hinzu, welcher einen Segue zum neuen MVC durchführt und das was in `CalculatorBrain` ist, zu der Zeit als der Button gedrückt wurde, mit dem Speicher `M` als unabhängige Variable, graphisch darstellt. Wenn das CalculatorBrain also z.B. `sin(M)` beinhaltet, zeichnen Sie eine Sinuskurve. Darauf folgende Eingabe in den Taschenrechner, darf keinen Effekt darauf haben was der graphische MVC anzeigt (bis der Button neu gedrückt wird). Ignorieren Sie Versuche des Users etwas zu zeichnen, wenn das Ergebnis im `CalculatorBrain` noch nicht vollständig ist (z.B. existiert eine pendingBinaryOperation) zu diesem Zeitpunkt.
7. Auf einem iPad und dem iPhone 6 plus, sowie 7 plus, muss der Graph (oder sollte dazu fähig sein) zur gleichen Zeit on-screen sein, wie das existierende Calculator User-Interface (d.h. in einem Split View). Auf anderen iPhones sollte der Graph via einem Navigation Controller auf den Screen “gepusht” werden.
8. Immer wenn ein Graph on-screen ist, muss eine Beschreibung von dem was gerade gezeichnet wird, irgendwo auf dem Screen zu sehen sein. Wenn also z.B. `sin(M)` gezeichnet wird, muss irgendwo auf dem Screen der String “sin(M)” zu sehen sein.
9. Ihr graphischer View muss `@IBDesignable` und seine Scale `@IBInspectable` sein. Die Achen des graphischen Views sollten im Storyboard auftauchen in der entsprechenden Scale.
10. Ihr graphischer View muss die folgenden drei Gesten unterstützen:
    a. Pinch (zoomt den gesamten Graphen, inkl. Achsen)
    b. Pan (verschiebt den gesamten Graphen, inkl. Achsen)
    c. Double-Tap (bewegt den Ursprung des Graphen zur Position des Double-Taps)
11. Sie sollten keine neue public API zu Ihrem CalculatorBrain in diesem Assignment hinzufügen und dürfen ebenfalls noch immer nicht die Swift Typen `Any` oder `AnyObject` in Ihrer Lösung verwenden.

Tipps

1. Ein häufig gemachter Fehler ist es, zu vergessen die Klasse des `UIViewController`s oder einem custom View im Identity Inspector in Xcode zu setzen. Sie müssen dies tun, wenn Sie `ViewController` in `CalculatorViewController` umbenennen und für den neuen `UIViewController` und neuen `UIView`, welche Sie in diesem Assignment erstellen. Wenn Sie eine `class` oder `struct` umbenennen, vergessen Sie nicht das File welches diese beinhaltet ebenfalls umzubenennen (sofern dies angemessen ist).
2. Um das Zeichnen des Graphen viel einfacher zu machen, wird Ihnen eine Klasse welche die Achsen des Graphen im aktuellen Context zeichnet zur Verfügung gestellt. Beachten Sie, dass die Zeichenmethode (`drawAxes`) dieser Klasse die `bounds` in denen gezeichnet wird und zwei weitere Parameter annimmt: `origin` und `pointsPerUnit` (dies ist essenziell die "Scale" des Graphen). Sie möchten dies wahrscheinlich nachahmen (d.h. `var`s für origin und scale haben) in Ihrem generischen graphischen View.
3. Es sollte keine Notwendigkeit bestehen, den Code Ihres `CalculatorBrains` für dieses Assignment zu verändern.
4. Zum jetzigen Zeitpunkt sollten Sie ein solides Verständnis von MVCs haben. Seien Sie sich darüber im klaren, was das Model für Ihre beiden MVCs in diesem Assignment ist. Stellen Sie sicher, die Regeln von MVC nicht zu verletzen (zum Beispiel indem Ihr View direkt mit Ihrem Model spricht).
5. Erinnern Sie sich daran, dass das Model das UI-unabhängige Herz von dem ist, was Ihr MVC tut oder zeigt. Ein gutes Beispiel ist das `CalculatorBrain`, welches offensichtlich das Herz eines MVCs ist, der ein Taschenrechner ist. Wenn Sie also darüber nachdenken, was das Model für Ihren neuen, graphischen View Controller ist, fragen Sie sich "was zeigt dieser MVC (im Grunde) an?". Dann wählen Sie etwas aus, was dies einfach repräsentiert.
6. Ein Model muss keine `class` oder `struct` sein, welches Sie erstellen (wie `CalculatorBrain`). Es kann **jeder beliebige Typ** sein.
7. Denken Sie sorgfälltig darüber nach, welcher Code in Ihren neuen graphischen `UIView` muss, gegenüber welcher Code in den Controller des graphischen MVCs muss. Generell gilt, Sie möchten wahrscheinlich dass der `UIView` so mächtig wie möglich, auch alleinstehend, ist.
8. Evtl. ist dieses Assignment verfügbar, bevor wir in der Vorlesung mehrere MVCs in einer App behandeln. Da Sie jedoch rechtzeitig mit dem Assignment anfangen sollten, überlegen Sie, ob Sie nicht vielleicht eine neue App mit nur einem MVC (dem neuen graphischen MVC) erstellen wollen. Suchen Sie sich einfach eine angenehme Funktion zum zeichnen (z.B. `cos(x)`). Nachdem wir dann in der Vorlesung mehrere MVCs besprochen haben, können Sie den so erstellen graphischen View per Drag-and-Drop in das Projekt von Assignment 3 einfügen.
9. Wenn Sie den graphischen View per Drag-and-Drop in die Szene des neuen MVC einfügen, können Sie "Reset to Suggested Contrains" verwenden (im Menü welches durch den Button in der unteren rechten Ecke des Interface Builders in Xcode gezeigt wird) um die "Pin to the Edges" Contraints zu setzen, solange Sie die gestrichelten blauen Linien verwendet haben um den graphischen View in der MVC Szene zu positionieren. Dann können Sie den Size Inspector in den Utilities verwenden um die Auswirkungen zu untersuchen. Erinnern Sie sich daran, dass Xcode eine Undo-Funktion hat.
10. Es ist möglich dieses Assignment ohne Verwendung von weiterem Autolayout, bis auf das oben beschriebene zu lösen.
11. Die `UIViewController` Subclass für den neuen graphischen MVC und die generische `UIView` Subclass, sind die einzigen neuen Klassen, die Sie von Grund auf neu schreiben müssen in diesem Assignment. Wenn Sie das Gefühl haben weitere Klassen zu benötigen, machen Sie es sich wahrscheinlich komplizierter als nötig.
12. Erschrecken Sie sich nicht, wenn Sie einen `UISplitViewController` einfügen und dieser viele andere View Controller mitbringt. Xcode versucht Ihnen nur zu helfen. Sie können diese einfach löschen und Ihren MVC mittels ctrl-drag entsprechend verbinden.

13. Es wäre schön, wenn der Ursprung des Graphen per Default im Zentrum des `UIView` erscheint. Seien Sie aber vorsichtig wo/wann Sie dies berechnen, da die bounds Ihres `UIView`s nicht gesetzt sind, bis das Layout für das Device auf dem es läuft durchgeführt wurde. Sie können sich natürlich sicher sein, dass die `bounds` in `draw(CGRect)` gesetzt sind, aber seien Sie vorsichtig und setzen Sie den `origin` nicht neu, wenn er bereits durch den User gesetzt wurde.
14. Ihr graphischer MVC sollte wahrscheinlich etwas sinnvolles machen, wenn Sie unstetige Funktionen zeichnen (z.B. sollte er nur versuchen Linien zu oder von Punkten zu zeichnen, deren y-Wert `.isNormal` oder `.isZero` ist). Um die Dinge etwas einfacher zu machen, ist es ok wenn Ihr graphischer View eine Funktion falsch zeichnet, wenn deren Wert sich zwischen einzelnen Pixeln stark ändert und zwar durch zeichnen einer fast vertikalen Linie zwischen den Punkten, auch wenn die Funktion wahrscheinlich nicht stetig an dieser Stelle ist (z.B. `tan(x)`). Es wäre allerdings eine coole Funktion, dies innerhalb einer bestimmten Toleranz zu erkennen und dann keine Linie an der Stelle zu zeichnen (dies ist Ihnen überlassen).
15. Verkomplizieren Sie nicht Ihr `draw(CGRect)`. Iterieren Sie einfach über jeden Pixel (nicht Point) über die Breite des Views (oder noch besser, über die Fläche in der Sie zeichnen sollen mittels des Parameters zu `draw(CGRect)`) und zeichnen Sie eine Linie zum nächsten Datenpunkt (oder "move to", wenn der letzte Datenpunkt ungültig war) den Sie erhalten (sofern dieser gültig ist).
16. Das Koordinatensystem in das Sie in Ihrem `draw(CGRect)` zeichnen, ist nicht das gleiche wie das Koordinatensystem in dem Ihre Daten sind (weil, z.B. Ihre Zeichenkoordinaten den Ursprung oben links haben, der Datenursprung aber wahrscheinlich irgendwo anders im View ist; von der scale erst gar nicht zu reden).
17. Der `AxesDrawer` weiss wie auf Pixel (nicht Point) boundaries gezeichnet wird (so wie Ihr `draw(CGRect)` dies sollte), aber nur wenn Sie ihm den `contentScaleFactor` des Zeichen-Context verraten, in den Sie zeichnen.
18. Vergessen Sie nicht Property Observing (`didSet`) zu nutzen, um zu veranlassen dass Ihr View sich neu darstellt, wenn ein Property welches beeinflusst wie er aussieht gändert wird.
19. Stellen Sie sicher Ihre `UIViewContentMode` richtig zu setzen (kann im Storyboard gemacht werden).
20. Ihre Gesten werden wahrscheinlich vom graphischen View behandelt, werden aber wahrscheinlich vom Controller "aktiviert". Aus diesem Grund sollten die Methoden, welche die Gesten behandeln nicht `private` sein im graphischen View.
21. Dieses Assignment verlangt wahrscheinlich etwas mehr Code als die vorherigen Assignments, können aber noch immer mit weniger als 100 Zeilen Code gelöst werden.

Lernziele

Hier ist eine partielle Liste von Konzepten, die Sie in diesem Assignment lernen sollen. :

- Verstehen von MVC
- Werte Semantik
- Erstellen einer neuen Subclass von UIViewController
- Universal Application (d.h. unterschiedliche UIs auf iPad und iPhone in der gleichen App)
- Split View Controller
- Navigation Controller
- Segues
- Subclass von UIView
- UIViewContentMode.redraw
- Zeichnen mit UIBezierPath und/oder Core Graphics
- CGFloat/CGPoint/CGSize/CGRect
- Gesten
- contentScaleFactor (Pixel vs Points)

Testate

Für alle Testate in diesem Semester ist das Ziel qualitativ hochwertigen Code zu schreiben, der ohne Warnings und Errors baut. Ebenfalls sollte die so entstandene App auf Funktion und Fehler gestestet werden und dies iterativ so lange, bis alle Fehler beseitigt sind und die App so funktioniert, wie gedacht.

Hier sind die am meisten vorkommenden Gründe, weshalb Sie vielleicht kein Testat bekommen:

- Projekt baut nicht
- projekt baut nicht ohne Warnings
- Eine oder mehrere Aufgaben wurden nicht zufriedenstellend gelöst
- Ein fundamentales Konzept wurde nicht verstanden
- Code ist optisch mangelhaft und schwer zu lesen (z.B. ist das Einrücken nicht konsistent, etc.)
- Programm kann zum Absturz gebraucht werden (z.B. wurde ein Optional das `nil` ist mit `!` unwrapped)
- Ihre Lösung ist schwer (oder unmöglich) für jemanden zu lesen, durch mangelhafte Kommentare, schlechte Variablen/Methoden Namen, schlechte Lösungsstruktur, zu lange Methoden, etc.
- UI ist chaotisch, Dinge sollten ausgerichtet sein und angemessene Abstände haben um “nett auszusehen”
- Private API wurde nicht korrekt abgegrenzt
- Die Grenzen von MVC wurden verletzt
- Die Swift API Design Guidelines wurden nicht eingehalten

Häufig fragen Studenten wie viele Kommentare im Code nötig sind. Die Antwort ist einfach. Der Code muss einfach und vollständig lesbar sein. Sie können davon ausgehen, dass die iOS API und der Code aus der Vorlesung bekannt sind. Sie sollten nicht davon ausgehen, dass bereits eine Lösung für die Aufgabe bekannt ist.