Assignment 2: Smiley 

Ziele

Im zweiten Assignment lernen Sie die Implementierung von Protokollen zum Austausch von Daten zwischen unterschiedlichen Komponenten kennen.

Weiterhin sollen zwei standardisierte Gesture Recognizer implementiert werden, deren Nutzung Auswirkungen darauf haben wie etwas gezeichnet und was gezeichnet wird.

Lesen Sie zuerst die Aufgabenstellung komplett durch, verstehen Sie die darin geforderten Anforderungen, planen Sie danach Ihre Vorgehensweise und machen sich danach an die Implementierung!

Hinweis: Das gesamte Praktikum lässt sich durch das schreiben von weniger als 30 Zeilen Code lösen. Ggf. brauchen Sie etwas mehr, sofern Sie ein paar Zwischenschritte einfügen. Sollten Sie deutlich mehr als 50 Zeilen Code brauchen, überdenken Sie Ihren Ansatz noch einmal!

Materialien

- Laden Sie sich die *Happiness* Demo aus Moodle herunter. Diese dient als Ausgangspunkt für dieses Praktikum. Es soll also nur Code/Funktionalität in der Demo ergänzt werden. Sie sollen kein neues Projekt erstellen.

- Sprechen Sie mit Ihren Kommilitonen, ggf. ist einer Ihrer Kommilitonen auf das gleiche oder ein ähnliches Problem gestoßen und kann Ihnen behilflich sein. Sie sollten keinen Code kopieren, stattdessen zusammen ein Problem verstehen, dafür eine Lösung finden und diese dann in Ihrer Praktikumsgruppe implementieren.

- In jedem Fall sollten Sie den Code den Sie schreiben, auch vollständig verstanden haben. Rechnen Sie damit, dass Sie bei einem Testat detailiert nach Ihrer Implementierung gefragt werden.

Aufgaben

1. Öffenen Sie das Projekt in Xcode und schauen Sie sich das Storyboard an. Was sehen Sie dort? (Wahrscheinlich nicht allzu viel...). Fügen Sie **vor** die Klassendefinition von `FaceView` das Schlüsselwort `@IBDesignable` ein und schauen sich das Storyboard nochmal an. Was sehen Sie nun? (warten Sie ggf. ein paar Sekunden). Ändern Sie die `happiness` Ihres Smileys und beobachten Sie die Auswirkung.
2. In `FaceView` gibt es drei Attribute, die Sie vielleicht modifizieren möchten. Um dies im Attribute Inspector und nicht in Code zu machen, fügen Sie jeweils vor `lineWidth`, `color` und `scale` das Schlüsselwort `@IBInspectable` ein. Schauen Sie sich anschließend für Ihren `FaceView` den *Attribute Inspector* an und probieren unterschiedliche Einstellungen aus.
3. Implementieren Sie das Model für die App. Zum jetzigen Zeitpunkt möchten wir kein großes Datenmodel, sondern behelfen uns mit einer Implementierung des Models im Controller. Das Model für die App ist relativ einfach, es ist lediglich ein *Property* vom Typ `Int` und hat einen Default Wert von `50`. Wir wollen nicht den kompletten Wertebereich von `Int` ausnutzen, sondern den Bereich von `0` (sehr traurig) bis `100` (super glücklich) nutzen. Achten Sie also darauf, dass das Model nur Werte in diesem Bereich speichert. Realisieren können Sie dies mit einem *Property Observer* (`didSet`), der die Werte setzt und für alle Werte kleiner 0 den Wert 0 bzw. für Werte größer 100 den Wert 100 verwendet.

   Jedes mal, wenn sich das Model ändert, soll der View neu gezeichnet werden. Implementieren Sie die Funktion `updateUI()` im `HappinessViewController`. Lassen Sie die Funktion vorläufig leer! Rufen Sie die Funktion aber aus Ihrem *Property Observer* auf (auch wenn Sie vorläufig nichts macht).

   Zum debuggen möchten Sie vielleicht im *Property Observer* die aktuelle `happiness` auf der Konsole ausgeben: `println("happiness = \(happiness)")`. Dies ist jedoch optional.
4. Da Sie das Model nun implementiert haben, können Sie nun das *Protokoll* implementieren, damit der generische `FaceView` mit einem beliebigen Controller verwendet werden kann.

   Implementieren Sie das Protokoll `FaceViewDataSource` in `FaceView.swift`. Das Protokoll hat die Funktion `happinessForFaceView(sender: FaceView) -> Double?`, welche des FaceView selbst als Parameter annimmt und ein Optional Double (die happiness) zurück gibt.

   Die Klasse FaceView braucht weiterhin eine (public) Variable `weak var dataSource: FaceViewDataSource?`, die als Quelle für die Daten diehnt. Da diese Variable `weak` ist, muss das Protokoll nur für Klassen (nicht für Structs und Enums) implementiert werden. Ihre Protokoll-Deklaration muss also wie folgt aussehen: `protocol FaceViewDataSource: class`.

   Modifizieren Sie `drawRect` nun so, dass statt einem festen Wert die Datenquelle (`dataSource`) für das Zeichnen des Mundes verwendet wird. Beachten Sie dabei, dass `dataSource` auch `nil` sein kann. In einem solchen Fall verwendeten Sie als Wert `0.0`. Diese Fallunterscheidung können Sie mit dem `??` Operator machen.
5. Ihr `FaceView` ist nun in der Lage Daten von einer externen Quelle (einem Controller) zu verarbeiten. Als nächstes müssen Sie dafür sorgen, dass `HappinessViewController` das `FaceViewDataSource` Protokoll implementiert.

   Implementieren Sie dazu die im Protokoll definierte Funktion `happinessForFaceView`. Beachten Sie dabei, dass Ihr Model `Int` Werte zwischen 0 und 100 speichert, `FaceView` aber `Double` Werte zwischen -1 und +1 erwartet. Konvertieren Sie also den Datentyp und Wertebereich entsprechend (Dreisatz-Rechnung).
6. Als letztes muss `dataSource` noch für den `FaceView` gesetzt werden. Ein guter Ort um dies zu machen, ist wenn der View vom Controller gesetzt wird. Dies geschieht normal automatisch aus dem Storyboard, da wir aber Code einfügen wollen, kann dies ebenfalls über ein *Outlet* im

`HappinessViewController` auf den `FaceView` erfolgen. Dies machen Sie wie bisher auch über Ctrl-Drag aus dem Storyboard in den Code. Benennen Sie das Outlet `faceView` und fügen einen Property Observer `didSet` hinzu. Darin setzen Sie die `dataSource` des `faceView` auf sich selbst, also den `HappinessViewController`.

Der Smiley wird nun immer mit dem aktuellen Model-Wert gezeichnet. Sie müssen nun nur noch den `faceView` neu zeichnen lassen. Machen Sie dies in der bisher leeren Funktion `updateUI()`.

Testen Sie Ihre App mit verschiedenen Werten im Model. Zum jetzigen Zeitpunkt sollte der Protokoll Teil der Aufgabe vollständig funktionieren. **Machen Sie nicht mit der Aufgabe weiter, bevor das bis hier geforderte Protokoll funktioniert! Vielleicht möchten Sie sich den bisheren Teil auch schon mal Teil-testieren lassen.**

7. Als nächstes fügen Sie einen Pinch Gesture Recognizer hinzu, der es Ihnen erlaubt den Smiley zu zoomen (bzw. dessen Zeichengröße zu ändern). Dies hat nichts mit dem Model zu tun! Es gibt lediglich an wie groß der Smiley gezeichnet wird.

   Sie müssen als erstes Ihrem `faceView` einen `UIPinchGestureRecognizer` hinzufügen. Überlegen Sie sich wo diese Geste behandelt werden soll (Hinweis: die Antwort steckt im letzten Absatz). Spezifieren Sie das `target` und `action`. Wobei `action` ein *Selector*, also eine Funktion ist, die aufgerufen wird sobald die Geste erkannt wird. Benennen Sie diese Funktion `scale(gesture: UIPinchGestureRecognizer)`. Die Funktion nimmt die Geste selbst als Parameter an und hat keinen Rückgabewert. Der dazu passende Selector beim hinzufügen des Gesture Recognizers sieht so aus: `action:#selector(FaceView.scale)`.

   Die `sclae` des `FaceView` soll nur dann geändert werden, wenn der Status der Pinch-Geste sich ändert (`.state == .Changed`). Denken Sie daran nach der Änderung die Scale der Geste wieder auf 1 zu setzen.

   Hinweis: Sie können das Hinzufügen des Pinch Gesture-Recognizers und das behandeln der Geste in der Funktion `scale` mit etwa 5 Zeilen Code realsieren! Standard Gesture Recognizer sind i.d.R. sehr einfach.

8. Als letzte Aufgabe müssen Sie nun noch einen Pan Gesture Recognizer implementieren. Dieser soll Ihr Model, also die `happiness` Ihres Smileys ändern (Mundkrümmung nach oben bzw. unten verschieben). Da Sie einen Gesture Recognizer bereits in Code hinzugefügt haben, machen Sie dies nun im Storyboard.

   Ziehen Sie einen *Pan Gesture Recognizer* aus der *Object Library* in den `FaceView`. Danach erstellen Sie wie im ersten Praktikum per Ctrl-Drag eine Target-Action in den Code, wo die Geste bearbeitet wird. Überlegen Sie sich, wo dieser Code dieses mal stehen muss (Hinweis: die Geste beeinflusst Ihr Model!). Benennen Sie die Funktion `changeHappiness(sender: UIPanGestureRecognizer)`, die auch hier wieder die Geste als Parameter entgegen nimmt.

   Innerhalb der Funktion behandeln Sie dieses mal drei Fälle mittels eines `switch` Statements.

   - `.Ended: fallthrough`
   - `.Changed:`
   - `default: break`

   `.Changed` behandelt dabei die eigentliche Änderung von `happiness`, die sich aus einer Translation in y-Richtung der Geste ergibt. Ermitteln Sie die Änderung der Translation aus der Geste und teilen diese durch eine Konstante (z.B. 4). Sie können dafür **optional** auch ein privates Struct mit Konstanten einführen, damit es ein wenig modularer wird. Setzten Sie den neuen Wert für `happiness` nur dann, wenn auch tatsächlich eine Änderung stattgefunden hat (`happinessChange != 0`).

   Testen Sie die Implementierung und lassen sich Ihre Aufgabe testieren.

Links

- The Swift Programming Language
- UIView Class Reference
- UIViewController Class Reference
- UIPinchGestureRecognizer Class Reference
- UIPanGestureRecognizer Class Reference