App-Entwicklung für iOS und OS X

SS 2016

Stephan Gimbel

Unwind Segue

- Segue der keinen neuen MVC erstellt

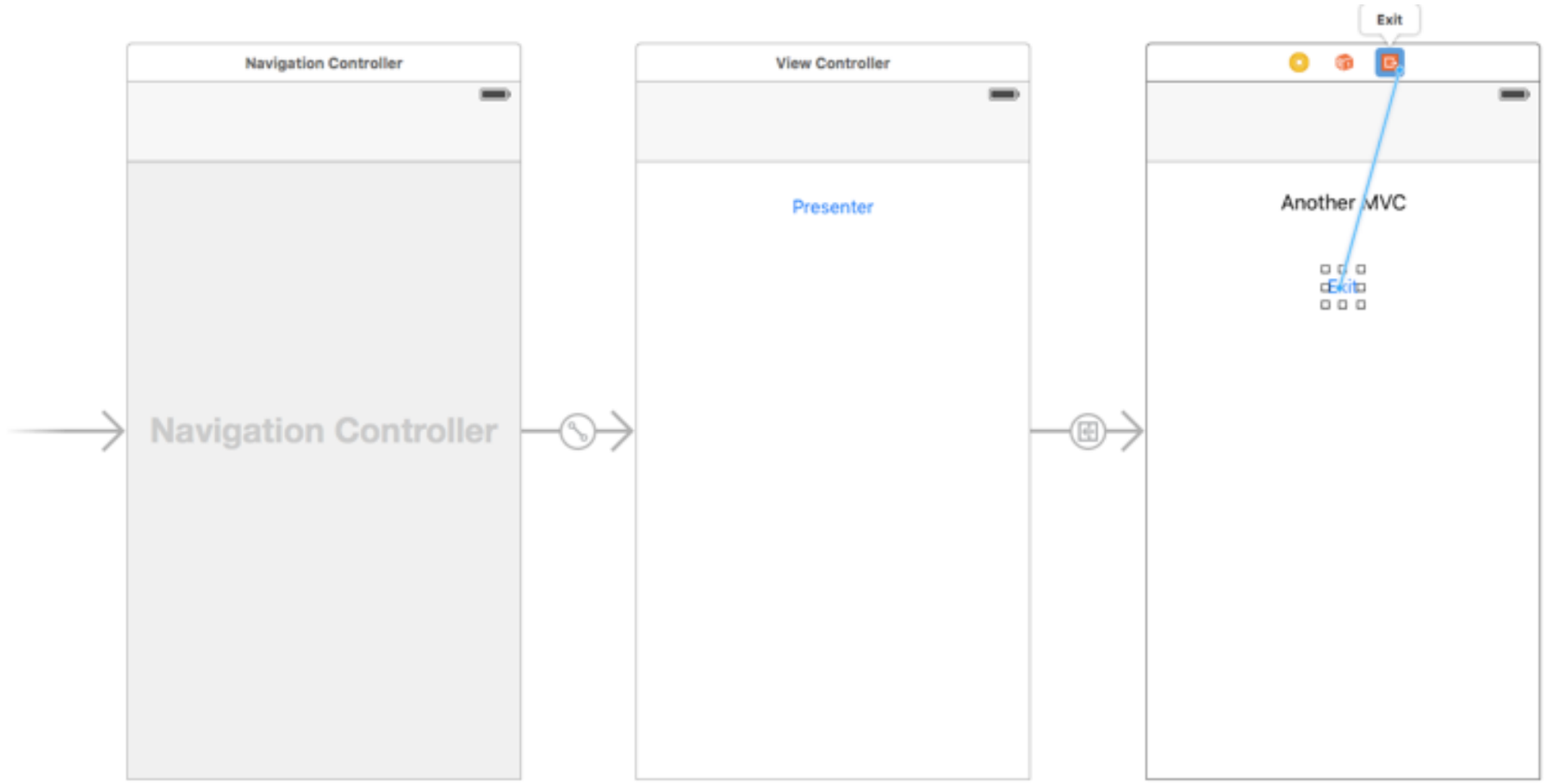
  Kann nur einen Segue zu anderen MVCs durchführen die direkt oder indirekt den aktuellen MVC erstellt haben.

- Wozu ist dieser gut?

  Springen innerhalb des Navigation Controller Stack.
  Ausblenden eines modalen MVC und Rückgabe von Informationen zum Presenter.

Unwind Segue

- Wie funktioniert es?
  Statt wie bisher durch ctrl-dragging zu einem anderen MVC, ctrl-drag zu “Exit” im selben MVC.

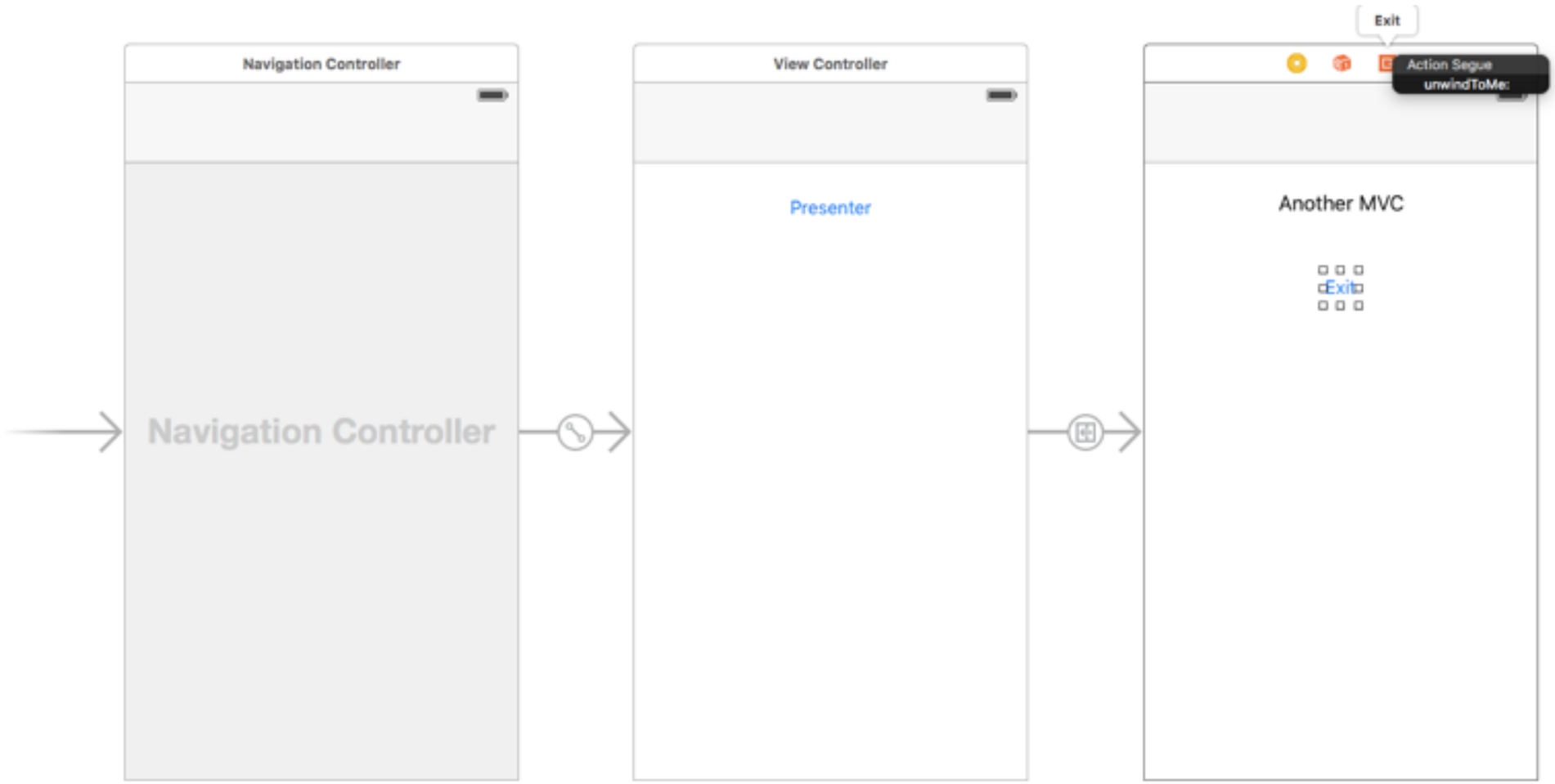


Unwind Segue

- Wie funktioniert es?

  Statt wie bisher durch ctrl-dragging zu einem anderen MVC, ctrl-drag zu “Exit” im selben MVC.

  Dann die `@IBAction` Methode auswählen, die wir in einem anderen MVC erstellt haben.



Unwind Segue

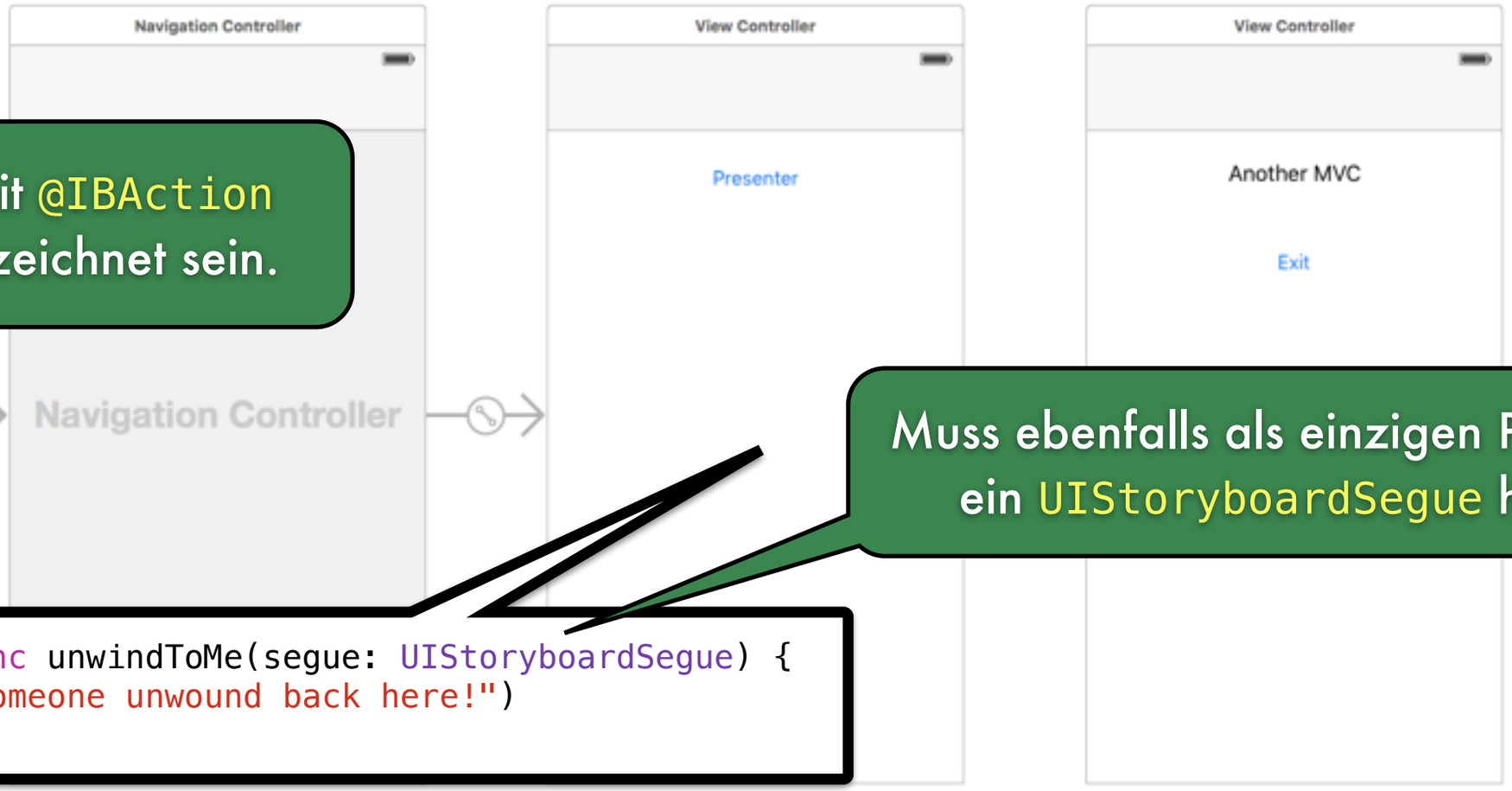
- Wie funktioniert es?

  Statt wie bisher durch ctrl-dragging zu einem anderen MVC, ctrl-drag zu “Exit” im selben MVC.

  Dann die `@IBAction` Methode auswählen, die wir in einem anderen MVC erstellt haben.

  Dies bedeutet “Segue durch Exit und finde einen Presenter der diese Methode implementiert”.

  Wenn kein Presenter diese Methode implementiert, wird auch kein Segue durchgeführt



Muss mit `@IBAction` gekennzeichnet sein.

Muss ebenfalls als einzigen Parameter ein `UIStoryboardSegue` haben.

```
@IBAction func unwindToMe(segue: UIStoryboardSegue) {
    print("Someone unwound back here!")
}
```

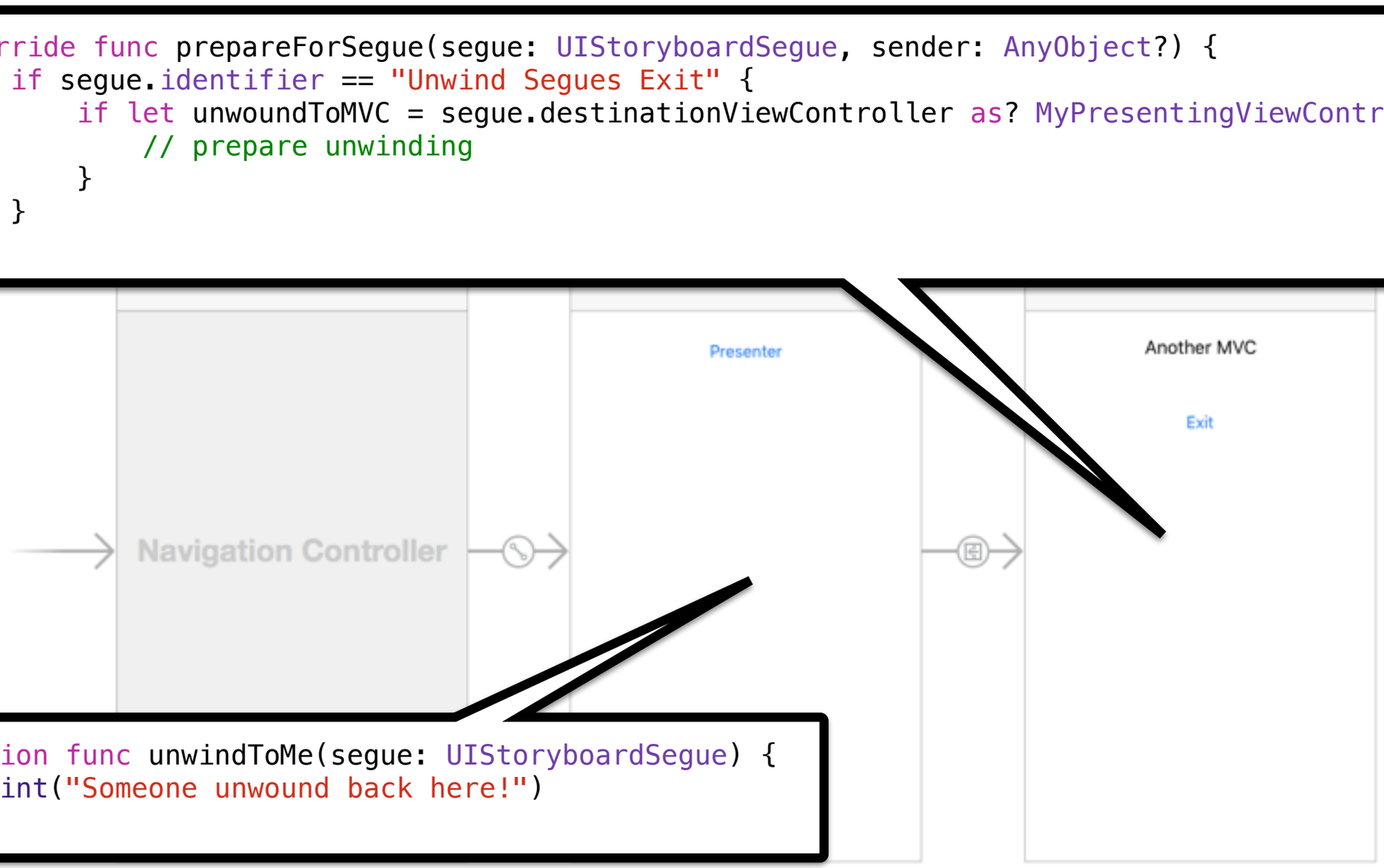
Unwind Segue

- Wie funktioniert es?

  Wenn die @IBAction gefunden wird, wird wie immer `prepareForSegue` aufgerufen.

  Dann wird die `@IBAction` Methode im anderen MVC aufgerufen und dieser MVC wird angezeigt.

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "Unwind Segues Exit" {
        if let unwoundToMVC = segue.destinationViewController as? MyPresentingViewController {
            // prepare unwinding
        }
    }
}
```




```
@IBAction func unwindToMe(segue: UIStoryboardSegue) {
    print("Someone unwound back here!")
}
```

Unwind Segue

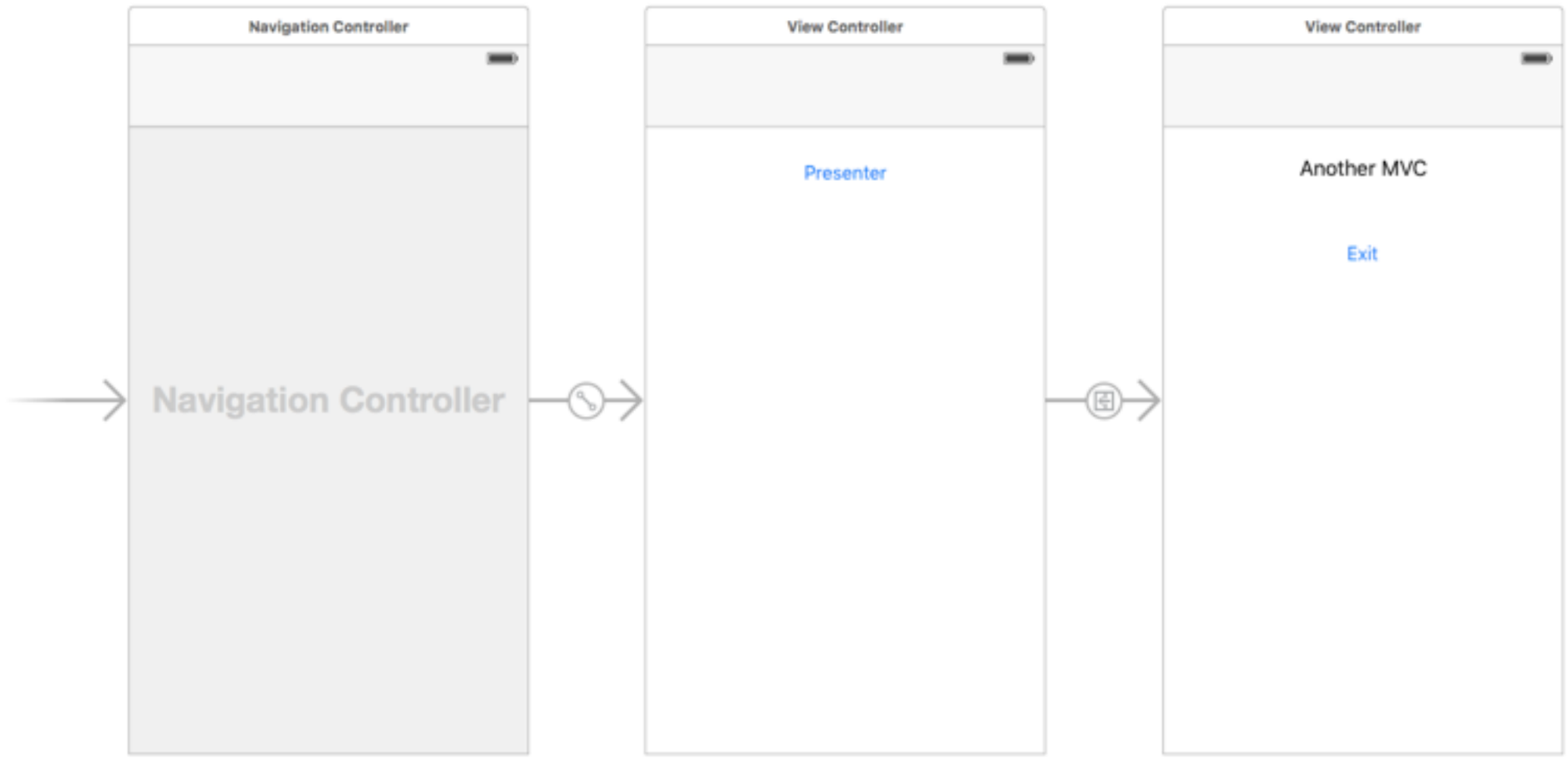
- Wie funktioniert es?

  Wenn die @IBAction gefunden wird, wird wie immer `prepareForSegue` aufgerufen.

  Dann wird die `@IBAction` Methode im anderen MVC aufgerufen und dieser MVC wird angezeigt.

  In diesem Prozess wird der präsentierte MVC ausgeblendet und gelöscht.

  Wir haben modale Segues noch nicht besprochen, diese funktionieren aber genau aus diese Art und Weise.



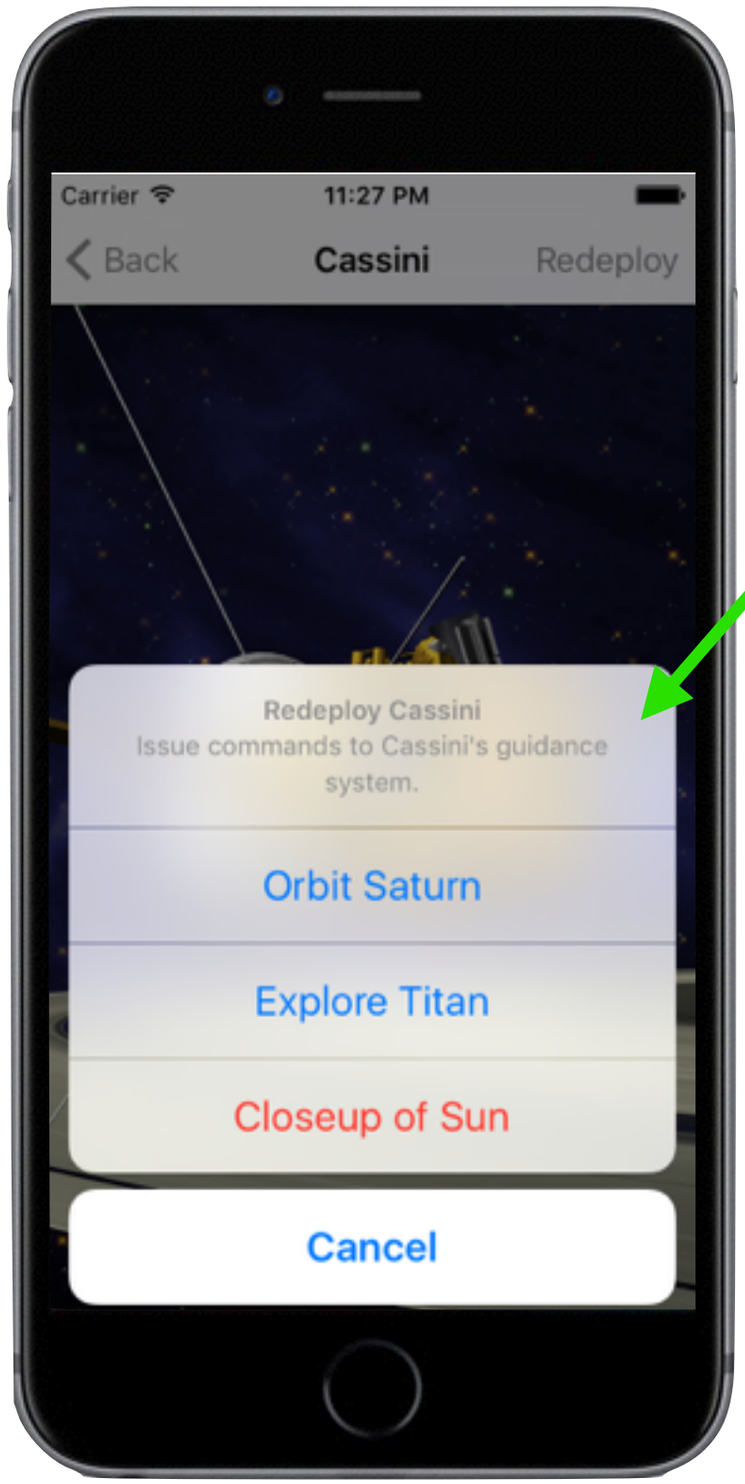
Alerts & Action Sheets

- Zwei Arten von “Pop up und Frage” Mechanismen
  - Alerts
  - Action Sheets

- Alerts
  - Popup in der Mitte des Screens.
  - Normalerweise wird dem User eine Frage mit nur zwei (oder einer) Antworten gestellt (z.B. OK/Cancel, Ja/Nein, etc.).
  - Oft für “asynchrone” Probleme verwendet (“Connection Reset”, “Verbindungsfehler”).
  - Kann ein Textfeld für schnelle Antwort enthalten.

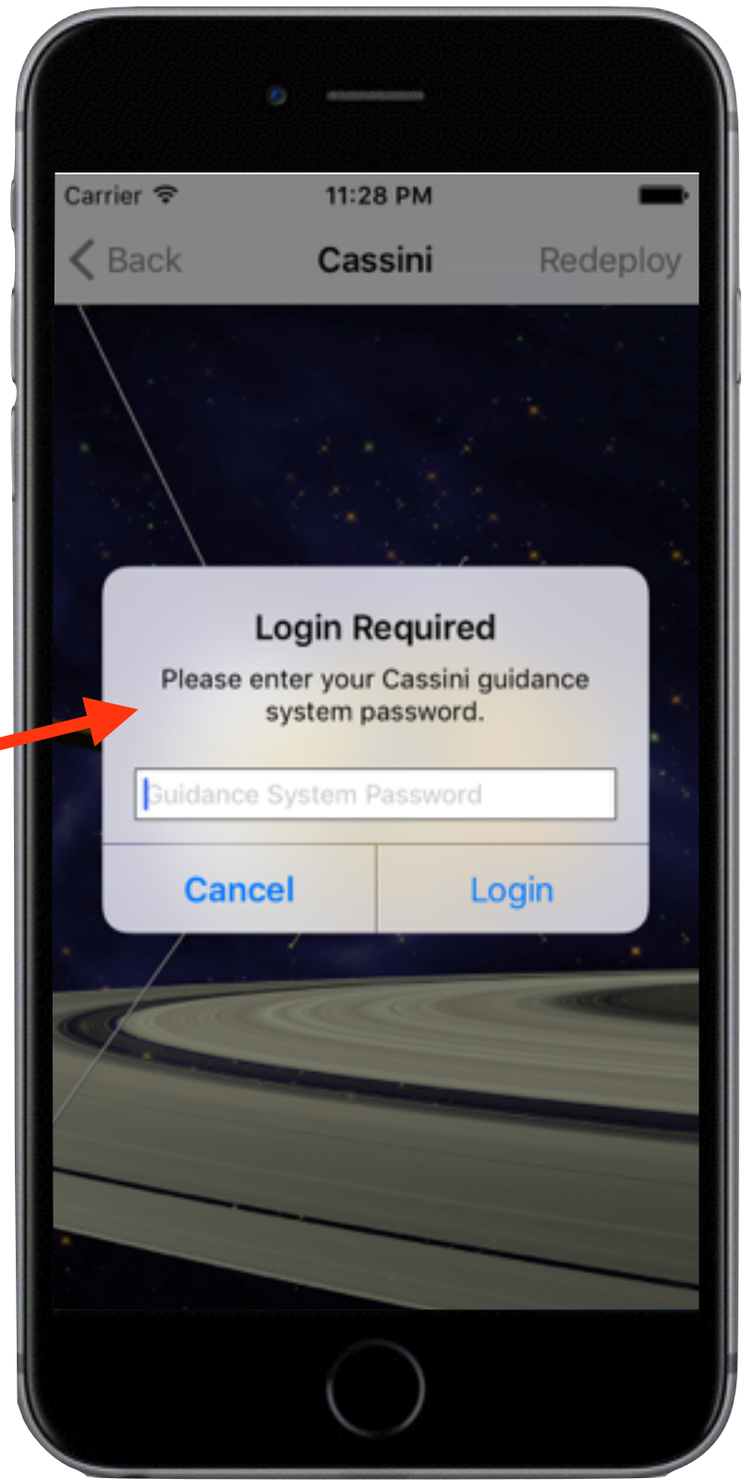
- Action Sheets
  - Bewegt sich normalerweise von unten in den Screen bei iPhone/iPod Touch und als Popover bei einem iPad.
  - Kann durch ein Bar Button Item oder von einer rechteckigen Fläche des View angezeigt werden.
  - Die Fragen haben normalerweise mehr als zwei Antworten.
  - Kann als Branching Entscheidung verwendet werden (“was als nächstes?”).

Alerts & Action Sheets

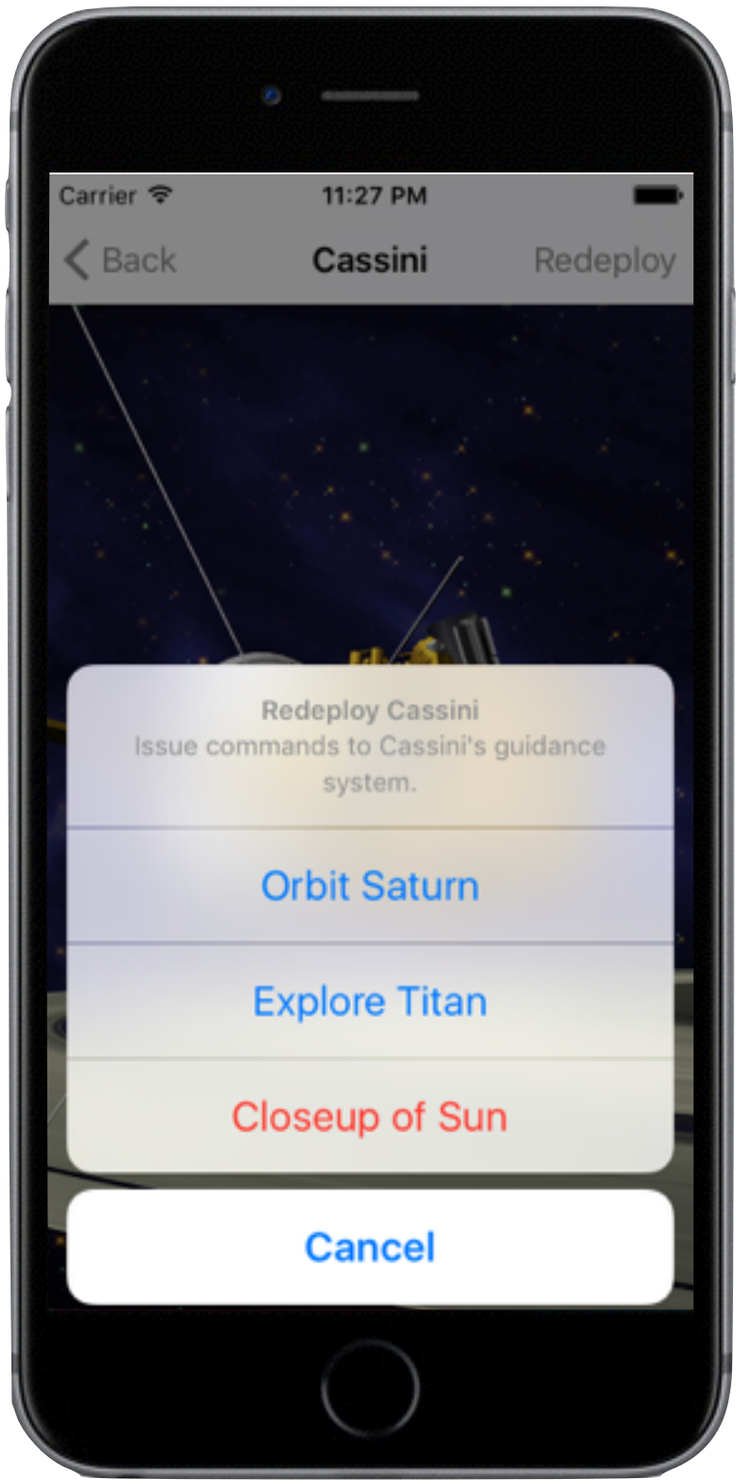


Action Sheet

Alert

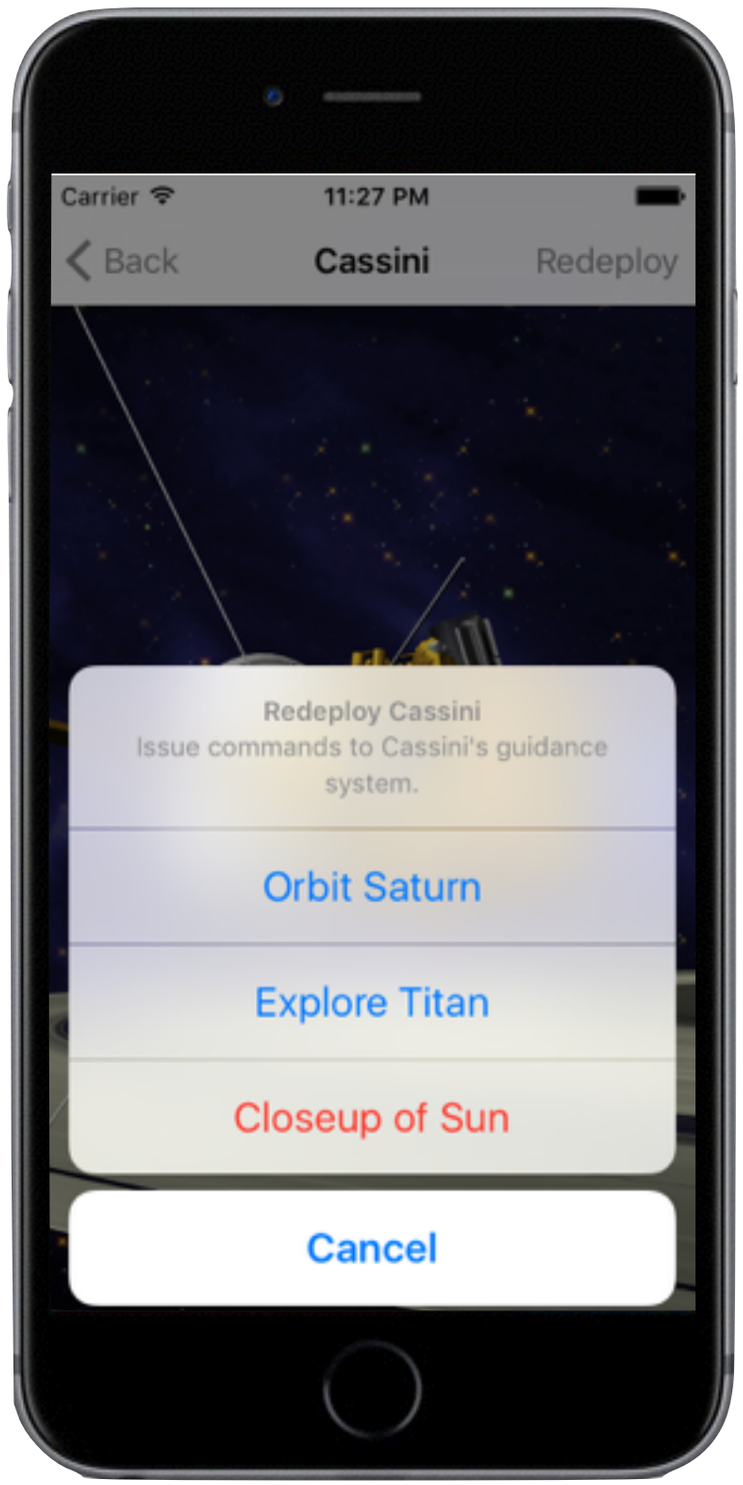


Alerts & Action Sheets



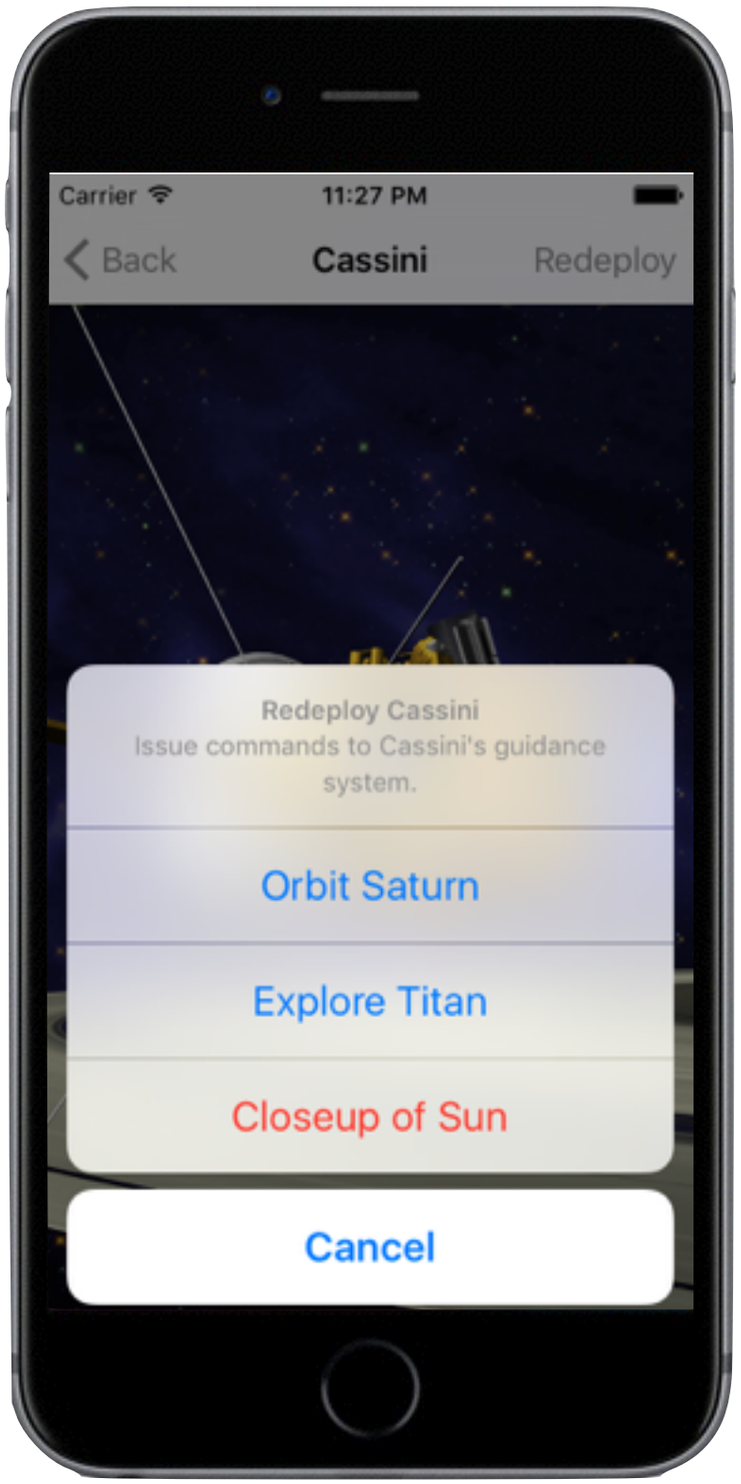

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)
```

Alerts & Action Sheets



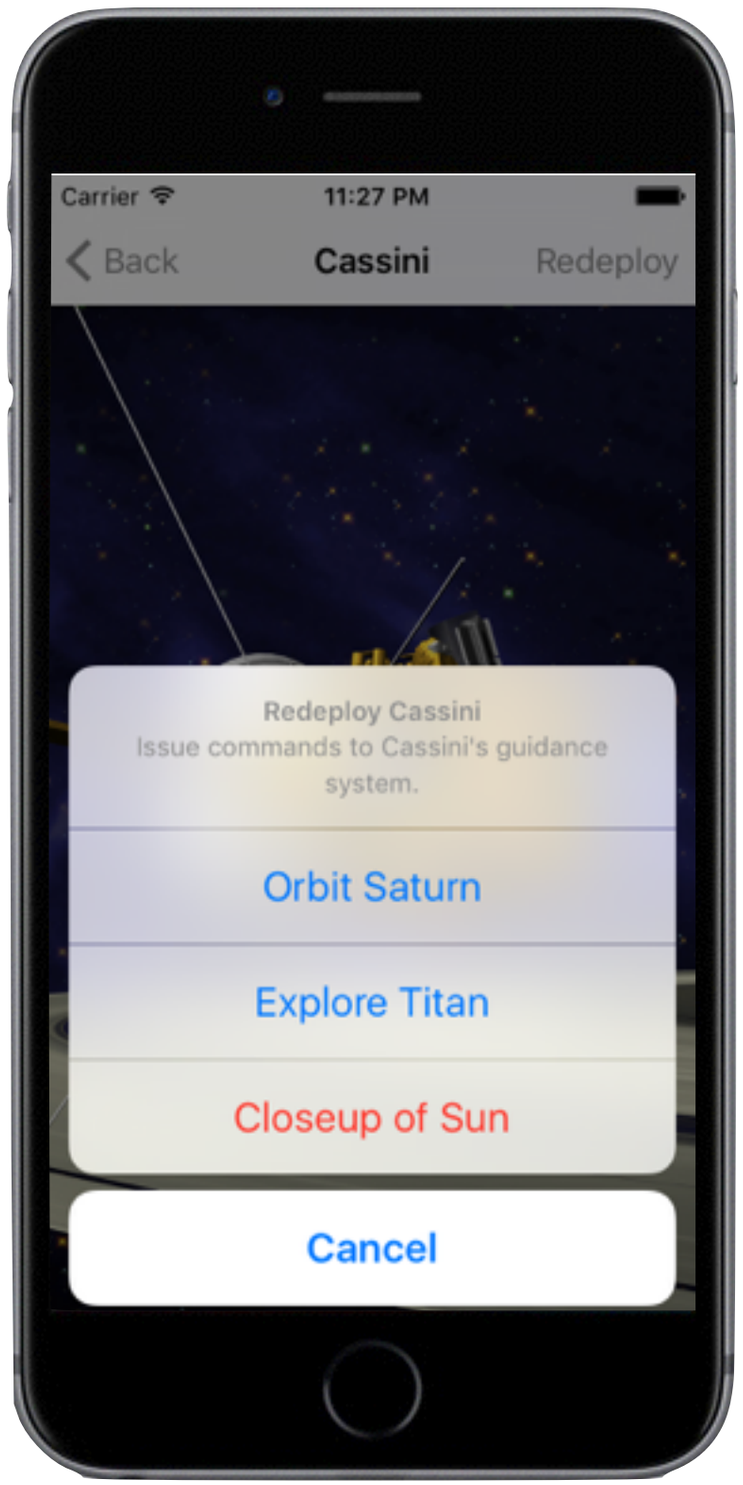

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)
```

Alerts & Action Sheets



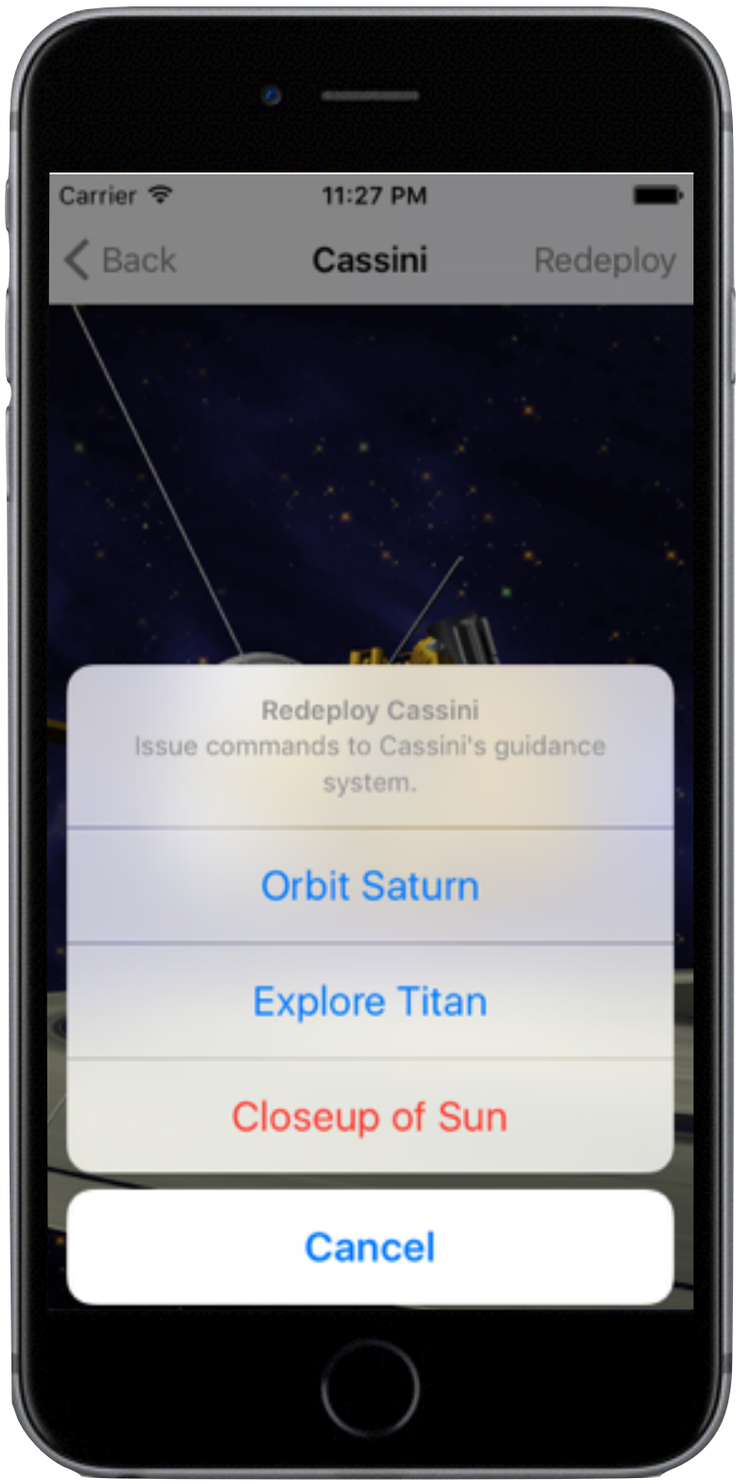

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(...)
```

Alerts & Action Sheets



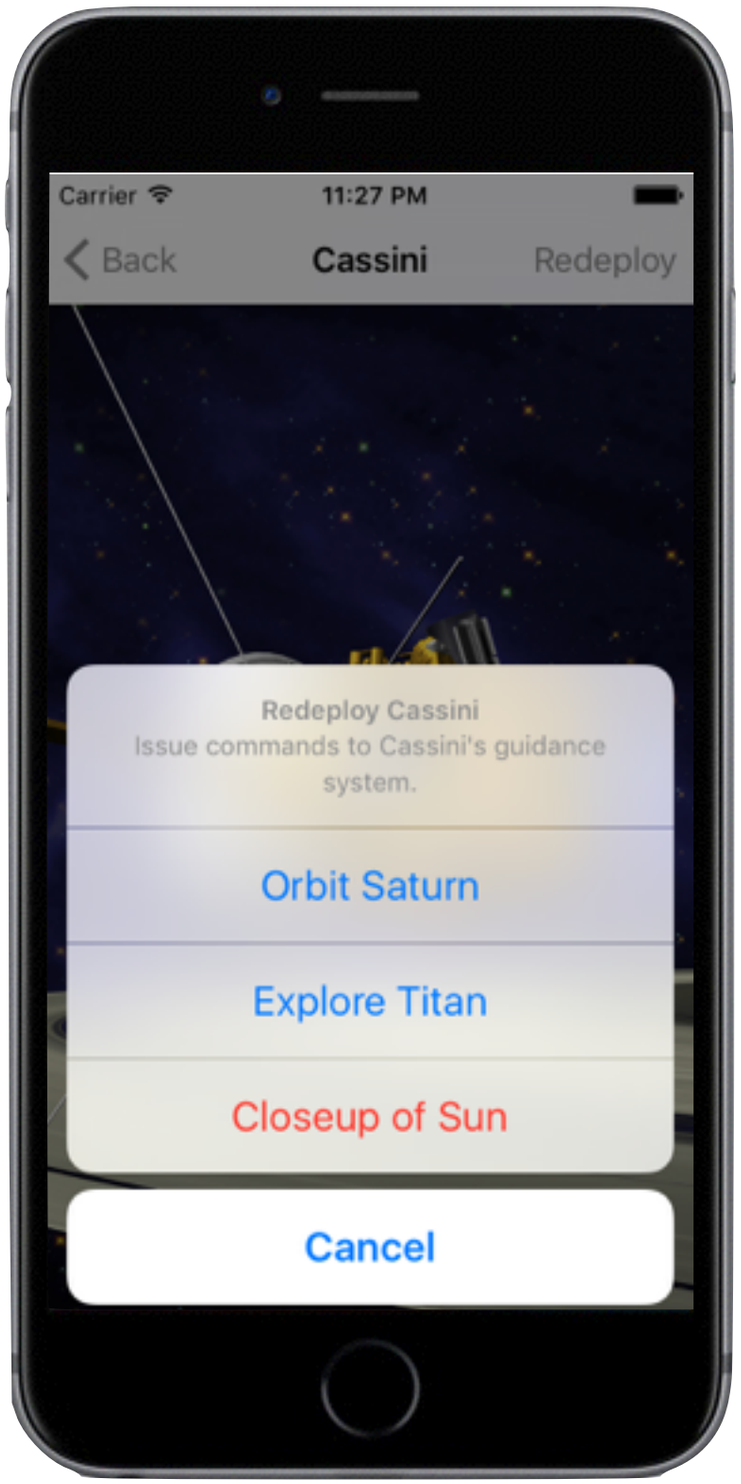

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(...))
```

Alerts & Action Sheets



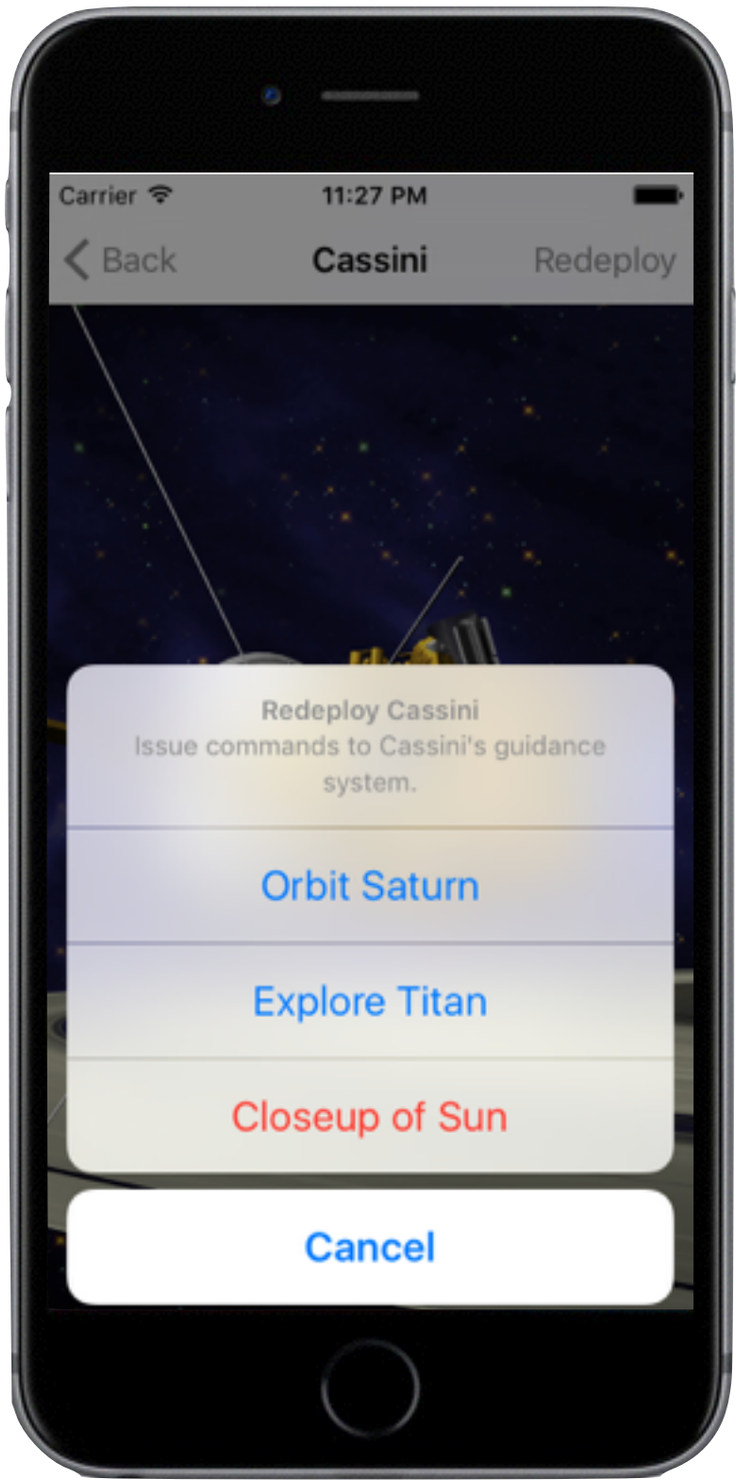

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(
      title: String,
      style: UIAlertActionStyle,
    handler: (action: UIAlertAction) -> Void
))
```

Alerts & Action Sheets



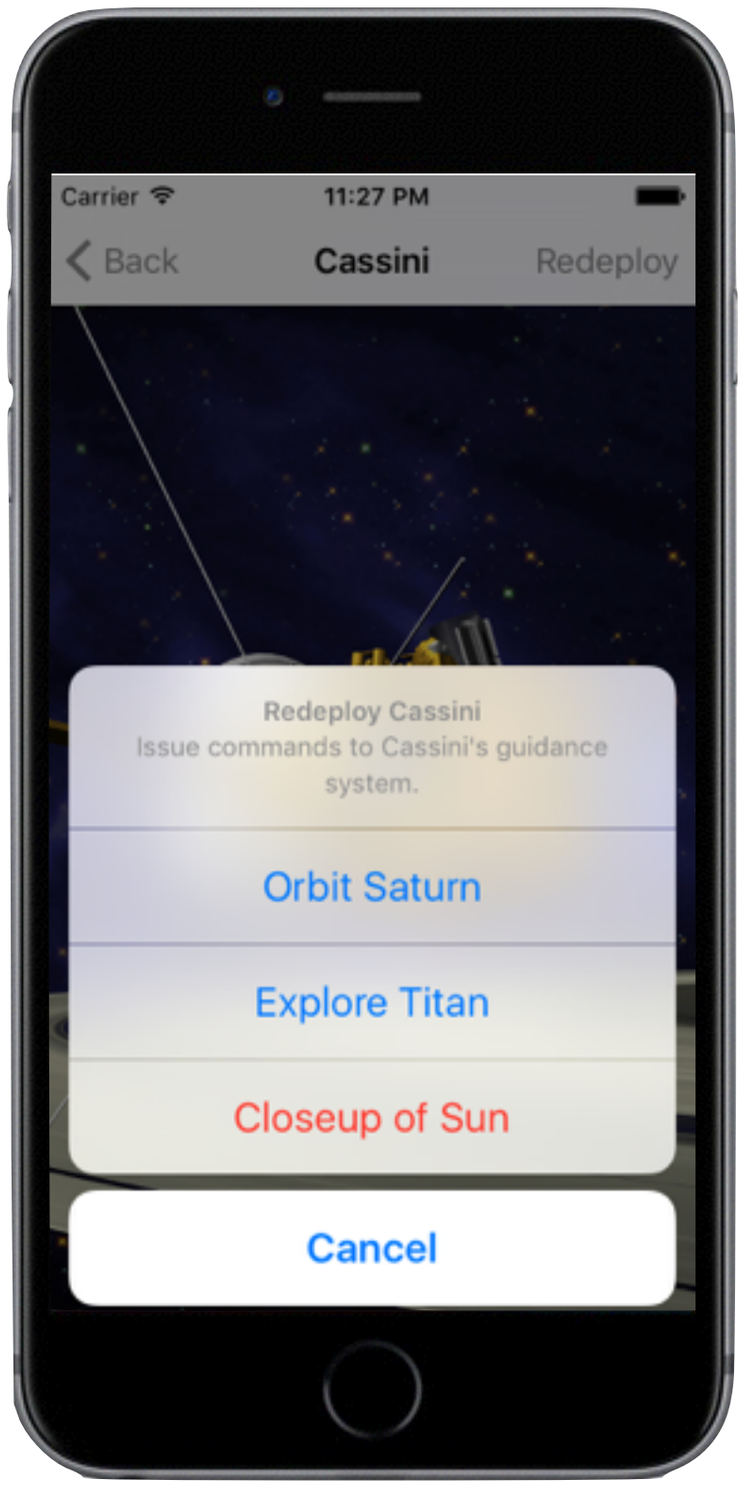

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(
      title: “Orbit Saturn”,
      style: UIAlertActionStyle.Default)
      { (action: UIAlertAction) -> Void in
          // orbit saturn
      }
)
```

Alerts & Action Sheets



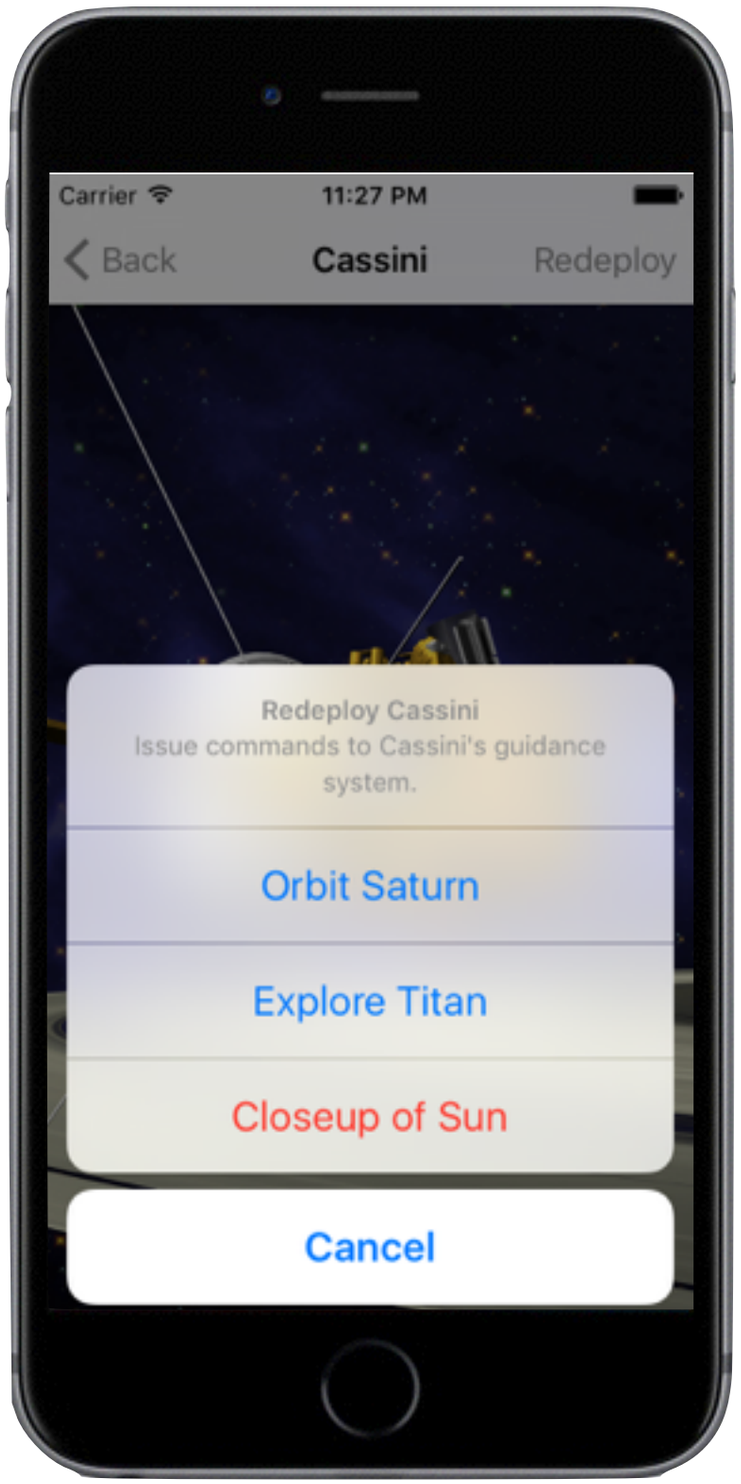

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(UIAlertAction(
      title: “Orbit Saturn”,
      style: UIAlertActionStyle.Default)
      { (action: UIAlertAction) -> Void in
          // orbit saturn
      }
)

alert.addAction(UIAlertAction(
      title: “Explore Titan”,
      style: .Default)
      { (action: UIAlertAction) -> Void in
          if !self.loggedIn { self.lohin() }
          // if loggedIn go to titan
      }
)
```

Alerts & Action Sheets



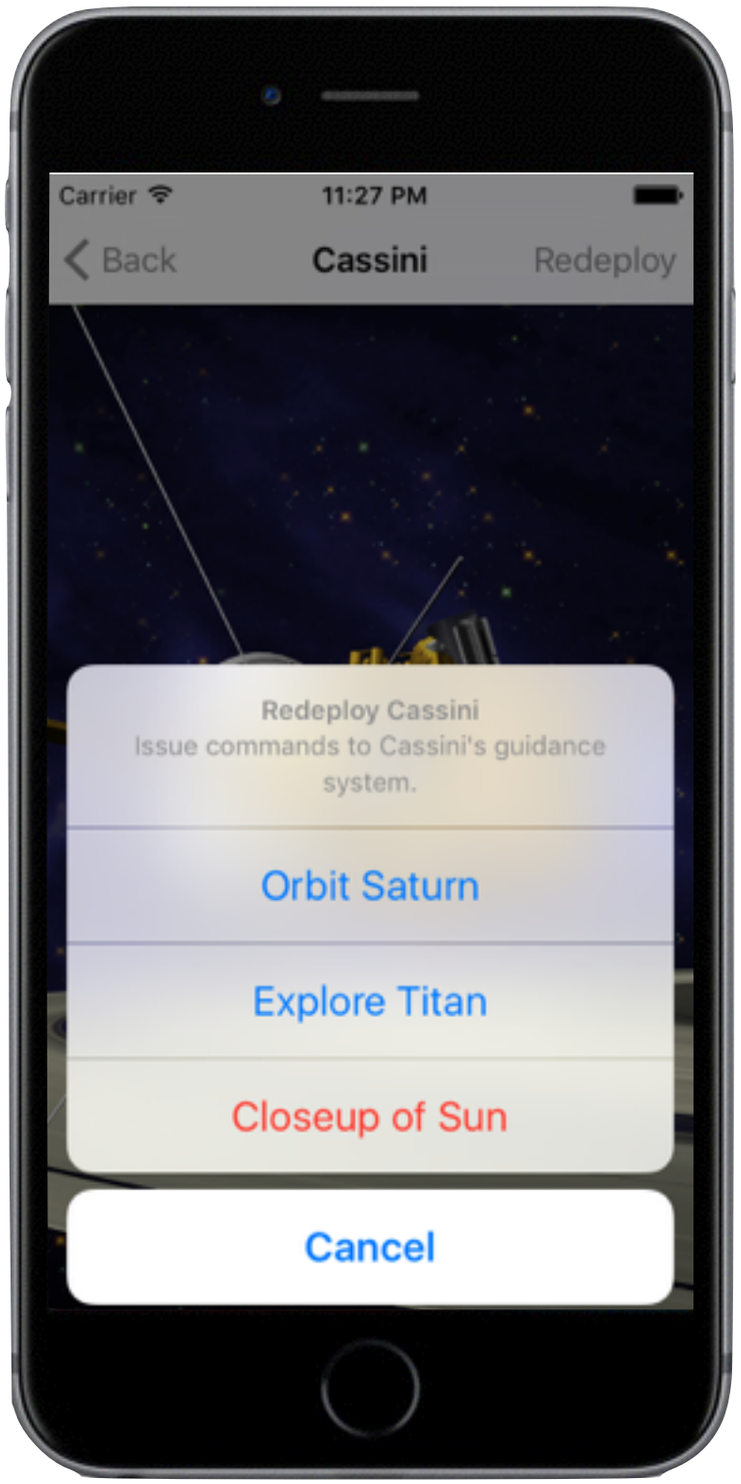

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
```

Alerts & Action Sheets



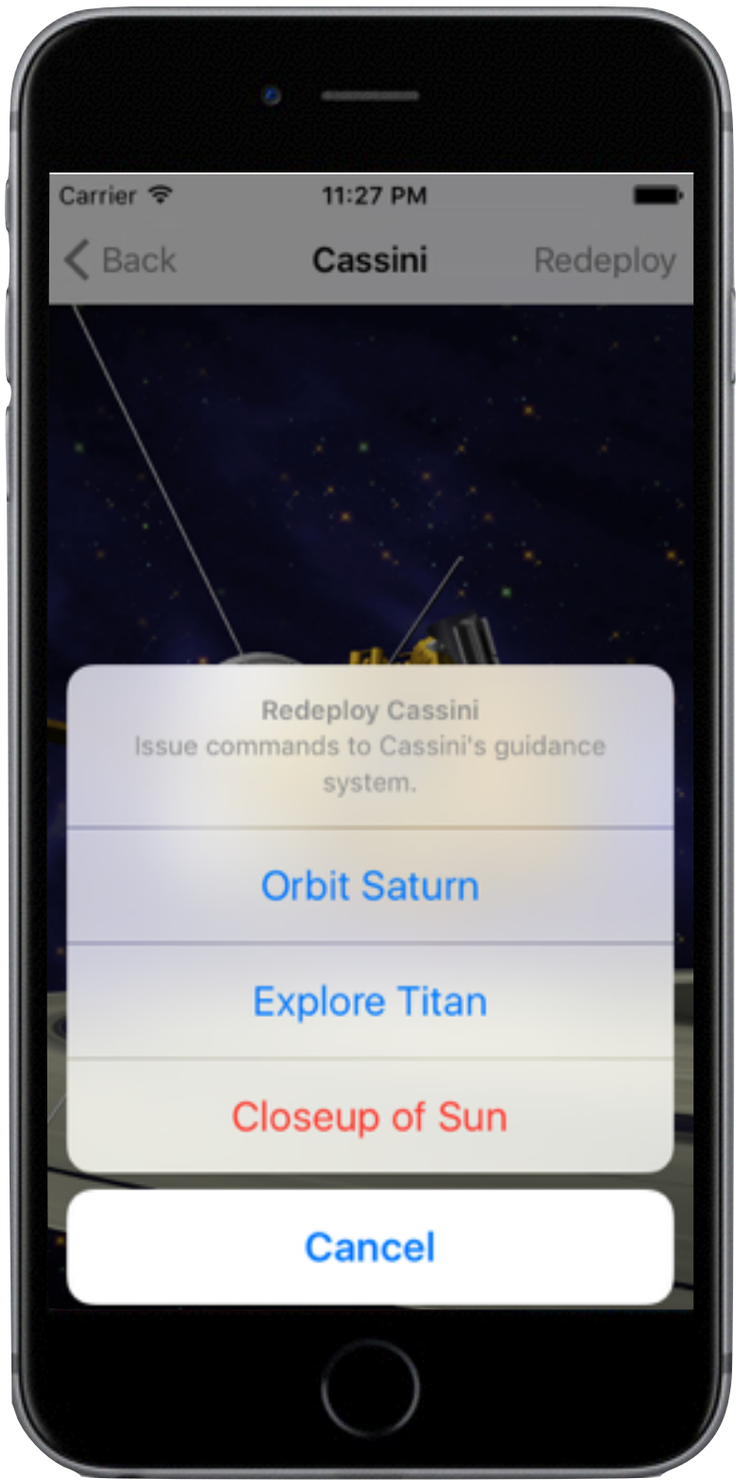

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)

alert.addAction(UIAlertAction(
    title: "Closeup of Sun",
    style: .Destructive)
    { (action:UIAlertAction) -> Void in
        if !loggedIn {self.login() }
        // if logged in destroy cassini
    }
)
```

Alerts & Action Sheets



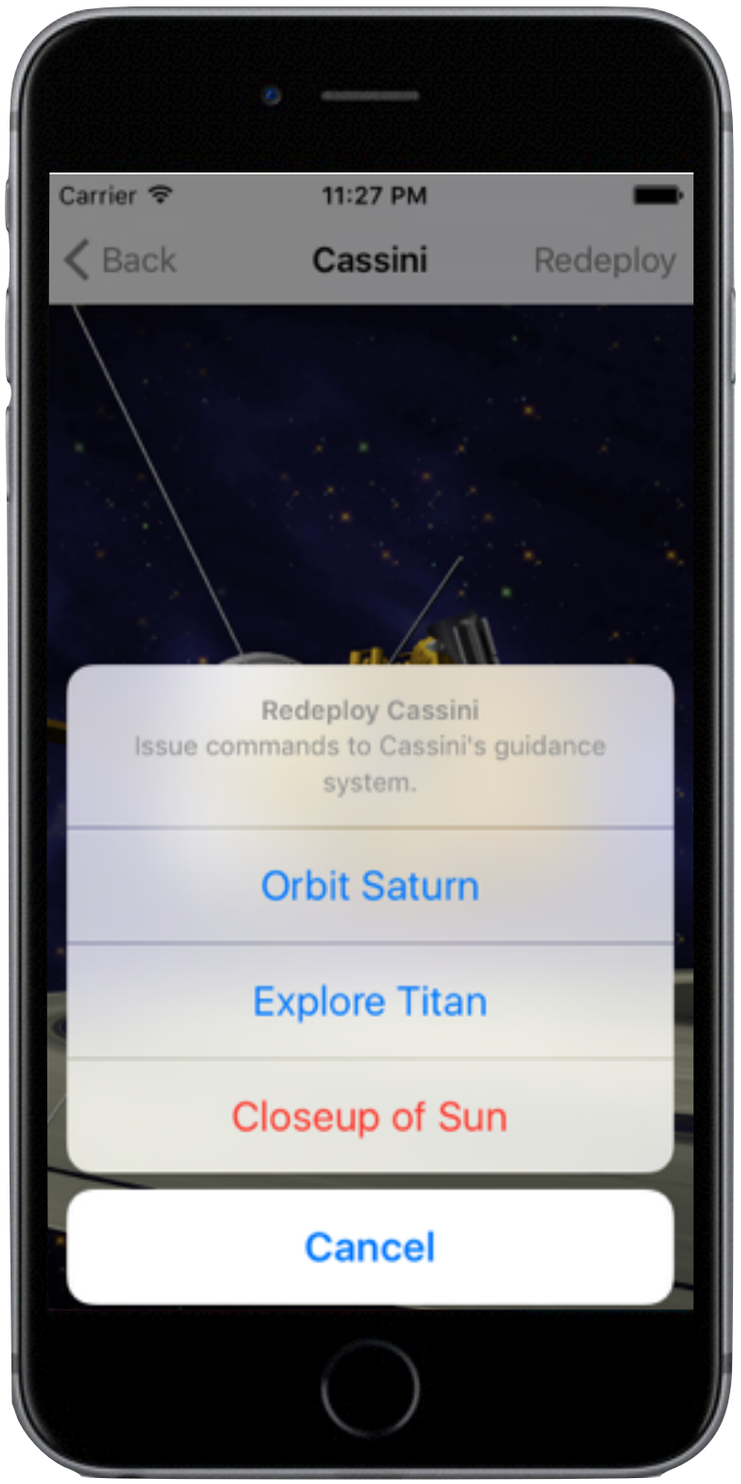

```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)

alert.addAction(UIAlertAction(
    title: "Closeup of Sun",
    style: .Destructive)
    { (action:UIAlertAction) -> Void in
        if !loggedIn {self.login() }
        // if logged in destroy cassini
    }
)

alert.addAction(UIAlertAction(
    title: "Cancel",
    style: .Cancel)
    { (action:UIAlertAction) -> Void in
        // do nothing
    }
)
```

Alerts & Action Sheets




```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy cassini (sun) action */)
alert.addAction(/* do nothing cancel action */)
```

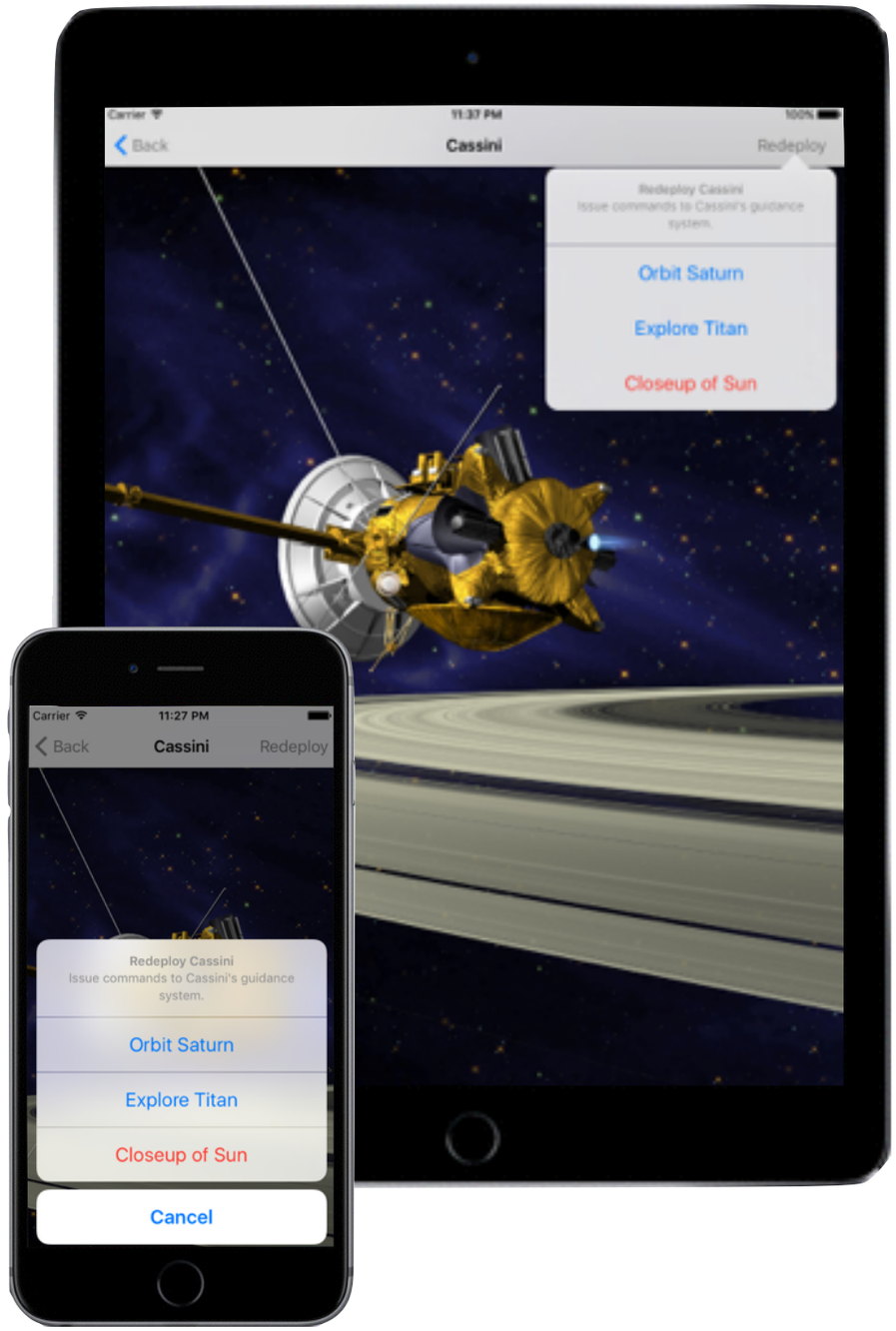
Alerts & Action Sheets




```
let alert = UIAlertController(
              title: "Redeploy Cassini",
            message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy cassini (sun) action */)
alert.addAction(/* do nothing cancel action */)

presentViewController(alert, animated: true, completion: nil)
```

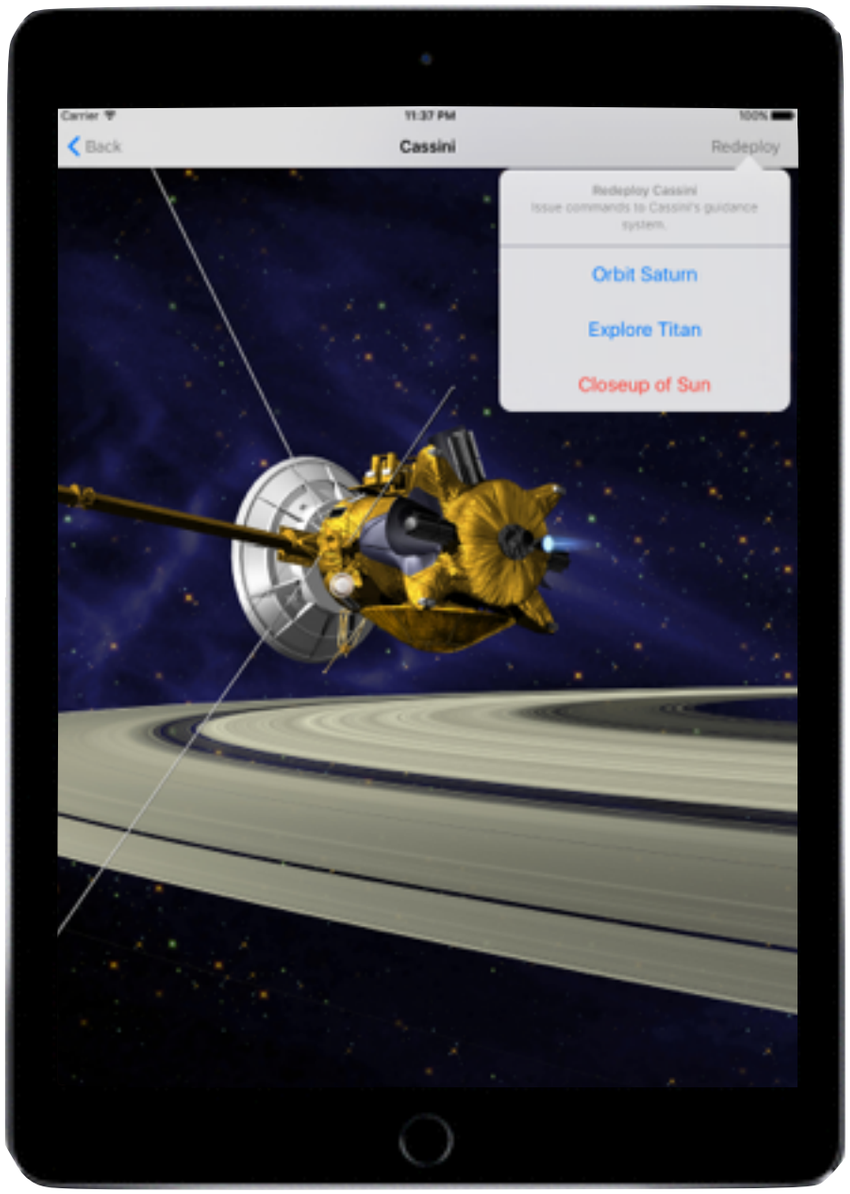
Alerts & Action Sheets

```
let alert = UIAlertController(
             title: "Redeploy Cassini",
           message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy cassini (sun) action */)
alert.addAction(/* do nothing cancel action */)

presentViewController(alert, animated: true, completion: nil)
```



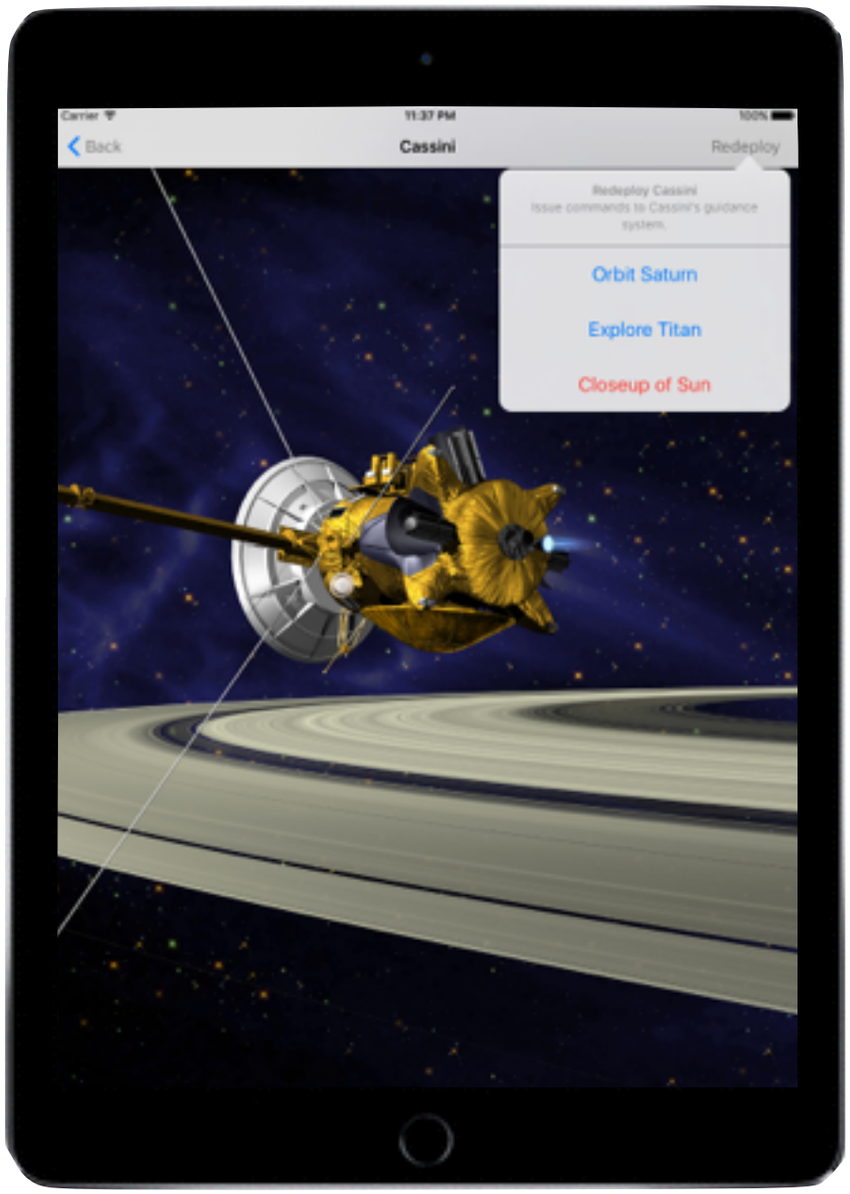
Alerts & Action Sheets

```
let alert = UIAlertController(
             title: "Redeploy Cassini",
           message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy cassini (sun) action */)
alert.addAction(/* do nothing cancel action */)

alert.modalPresentationStyle = .Popover



presentViewController(alert, animated: true, completion: nil)
```



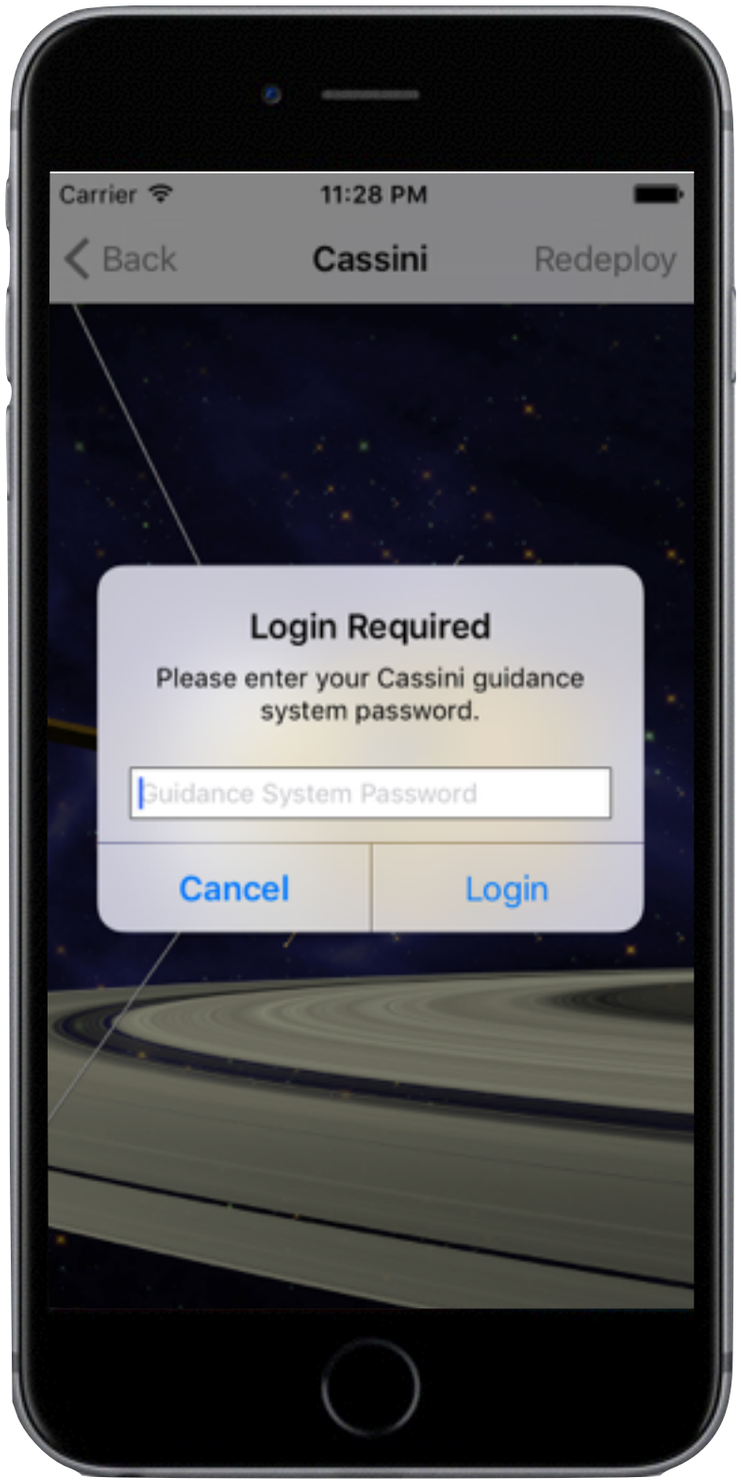
Alerts & Action Sheets

```
let alert = UIAlertController(
             title: "Redeploy Cassini",
           message: "Issue commands to Cassini's guidance system.",
    preferredStyle: UIAlertControllerStyle.ActionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy cassini (sun) action */)
alert.addAction(/* do nothing cancel action */)

alert.modalPresentationStyle = .Popover
let ppc = alert.popoverPresentationController
ppc?.barButtonItem = self.navigationItem.rightBarButtonItem

presentViewController(alert, animated: true, completion: nil)
```



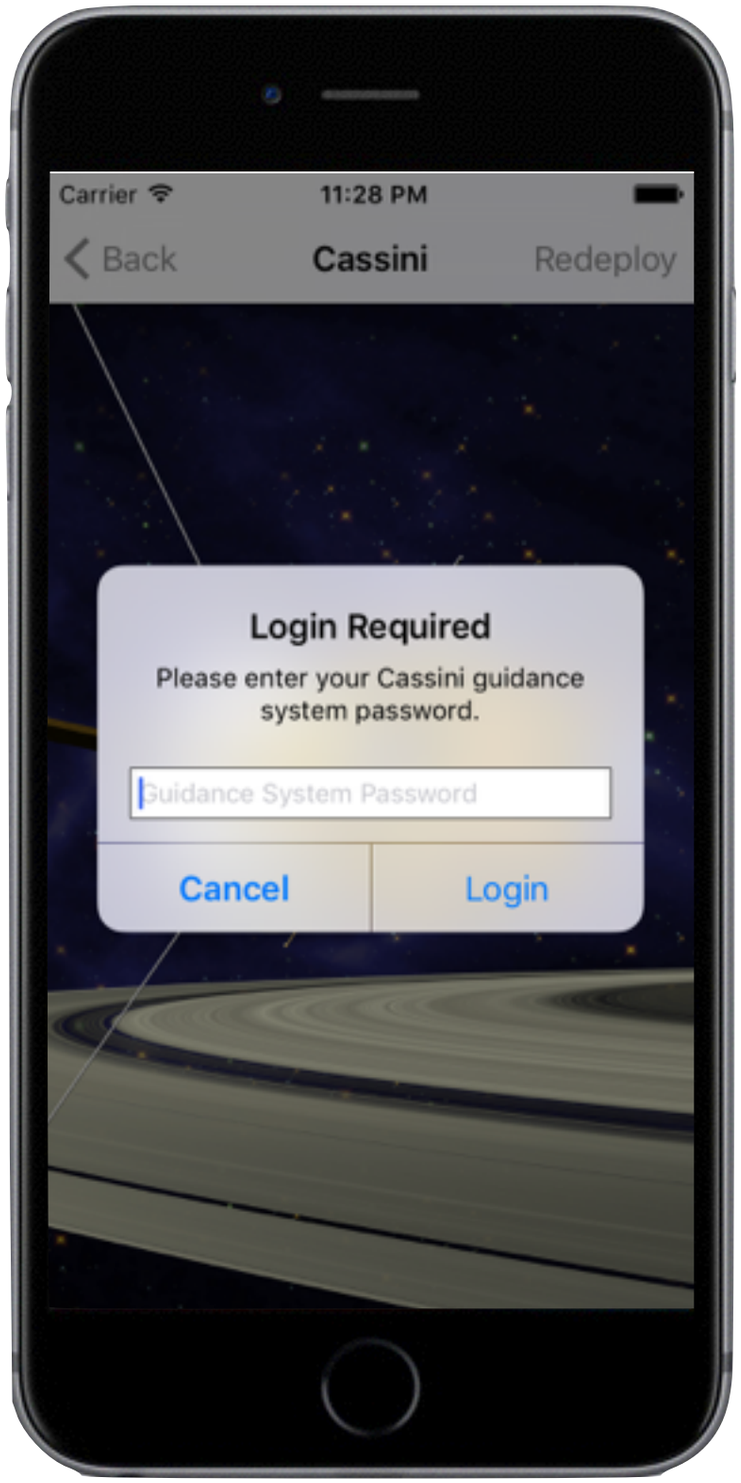
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)
```



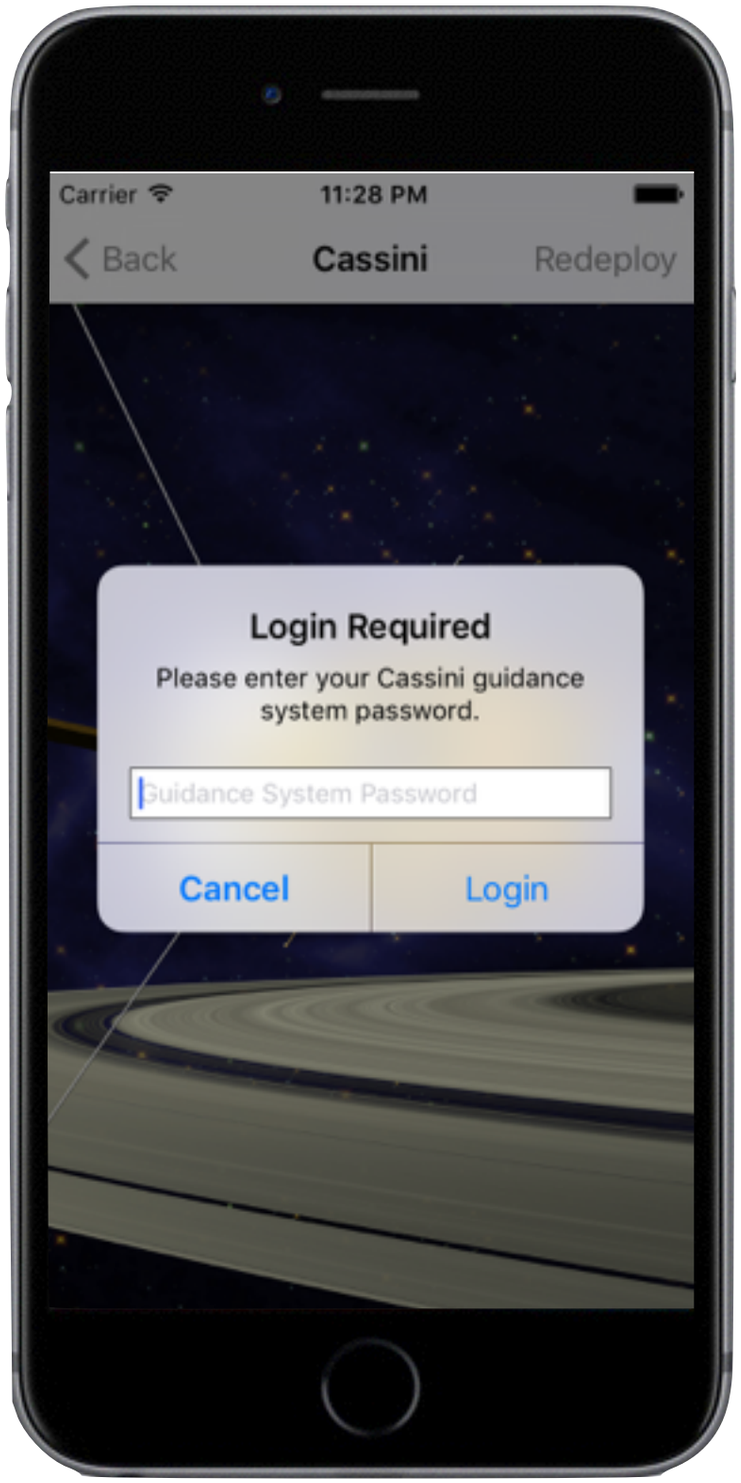
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(UIAlertAction(
    title: "Cancel",
    style: .Cancel)
    { (action: UIAlertAction) -> Void in
        // do nothing
    }
)
```



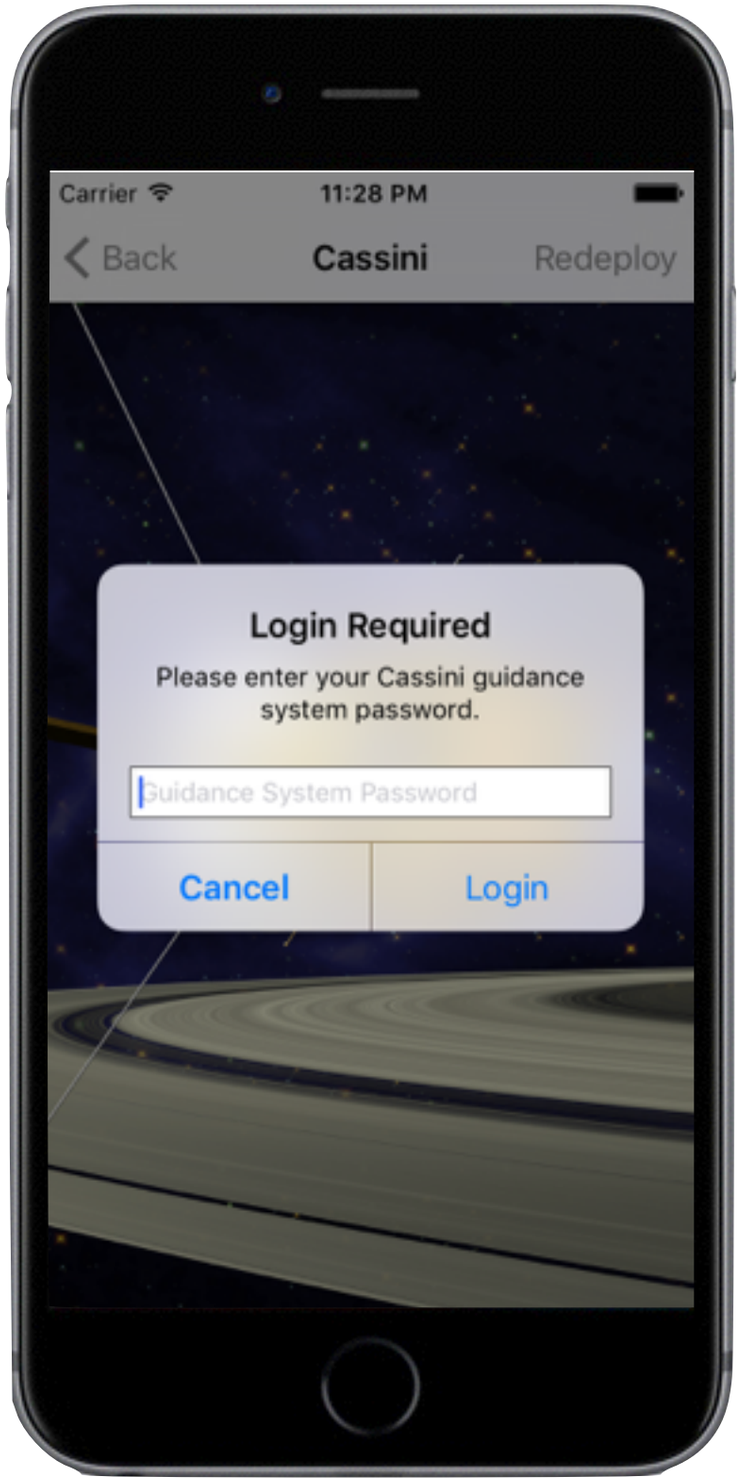
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(/* cancel button action */)
```



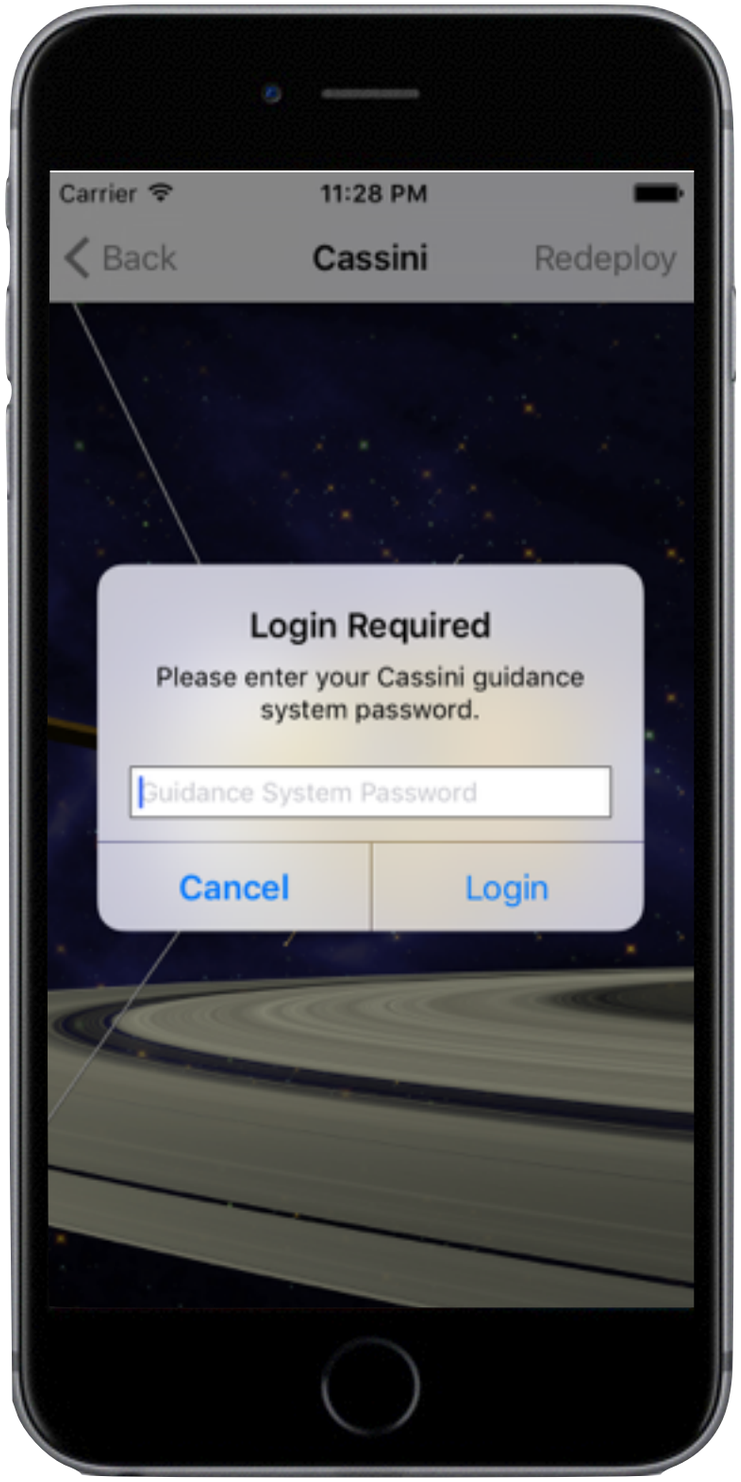
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in


    }
)
```



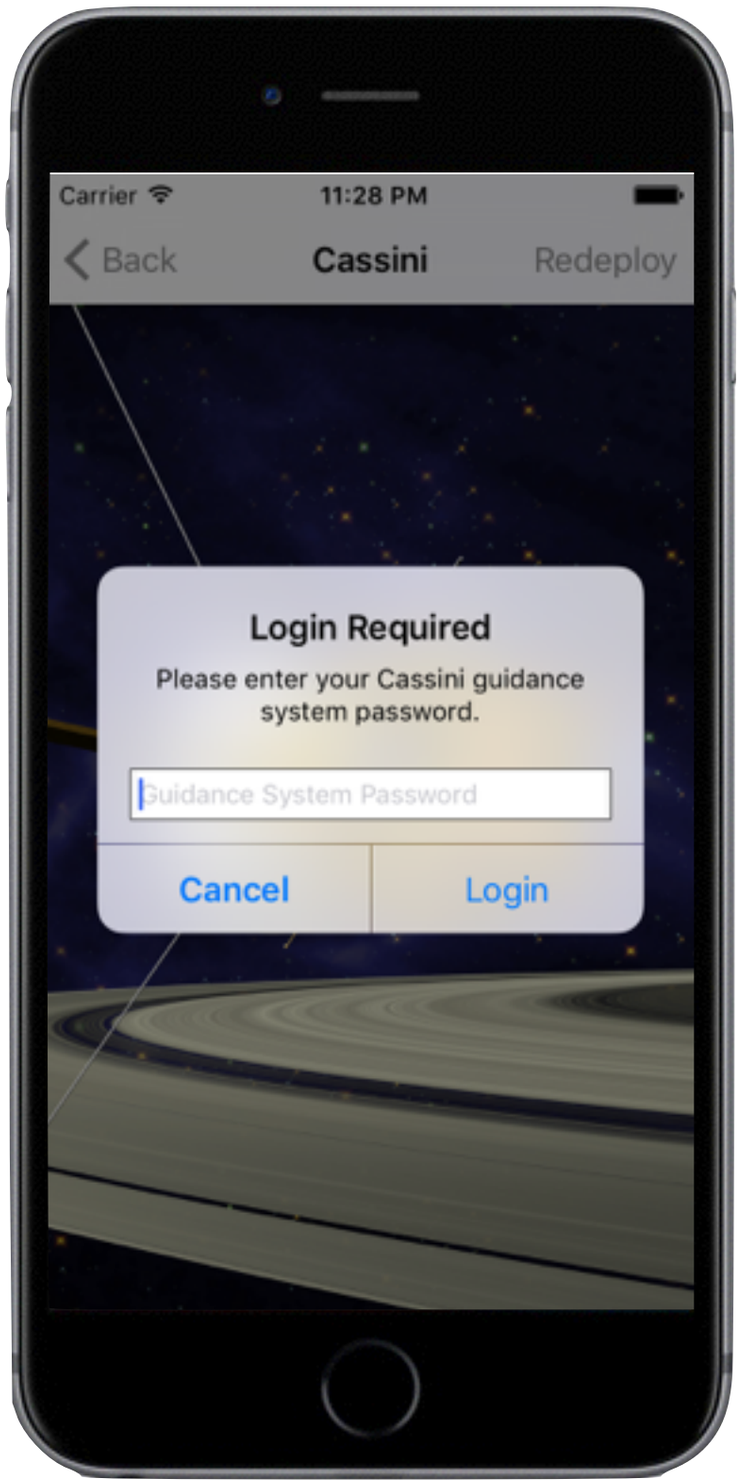
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in


    }
)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = "Guidance System Password"
}
```



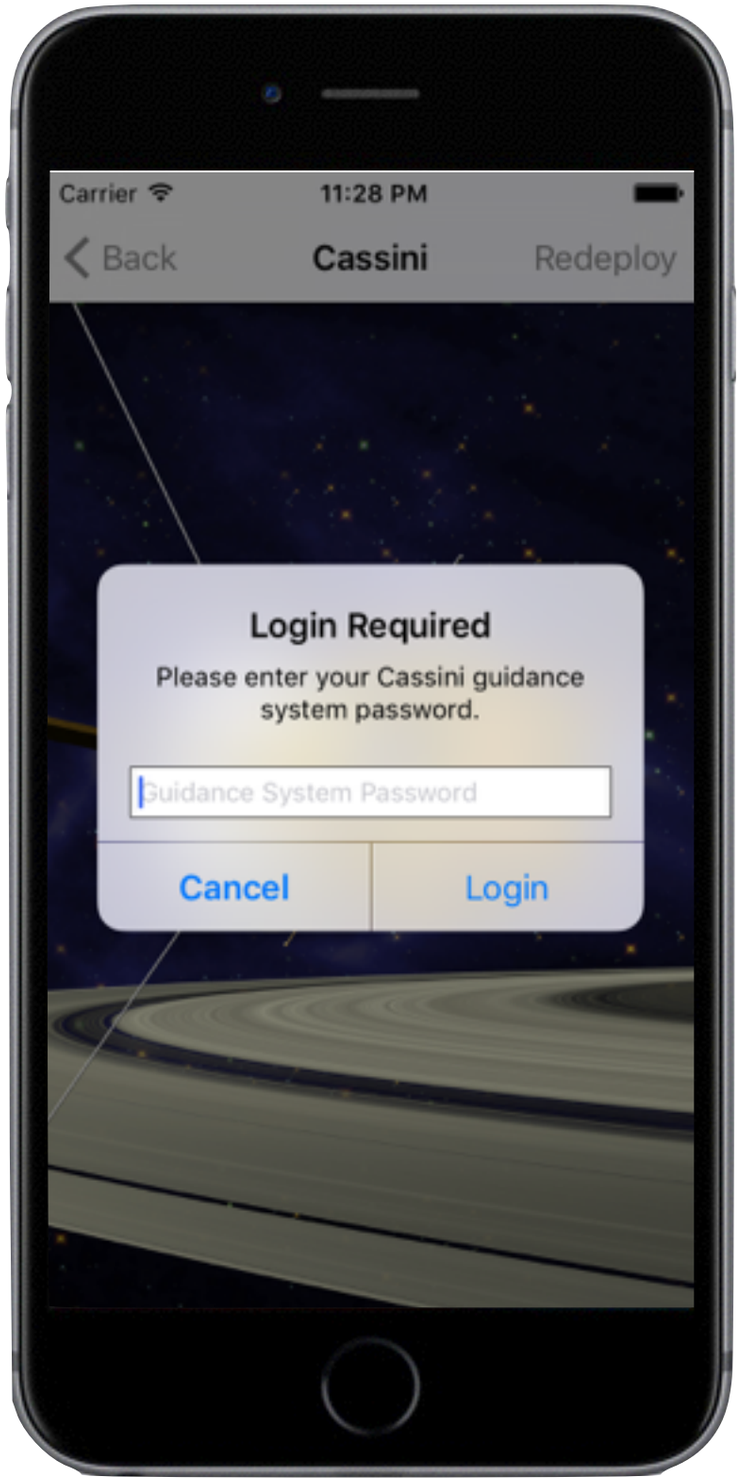
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in
        let tf = self.alert.textFields?first as? UITextField
        if tf != nil { self.loginWithPassword(tf.text) }
    }
)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = "Guidance System Password"
}
```

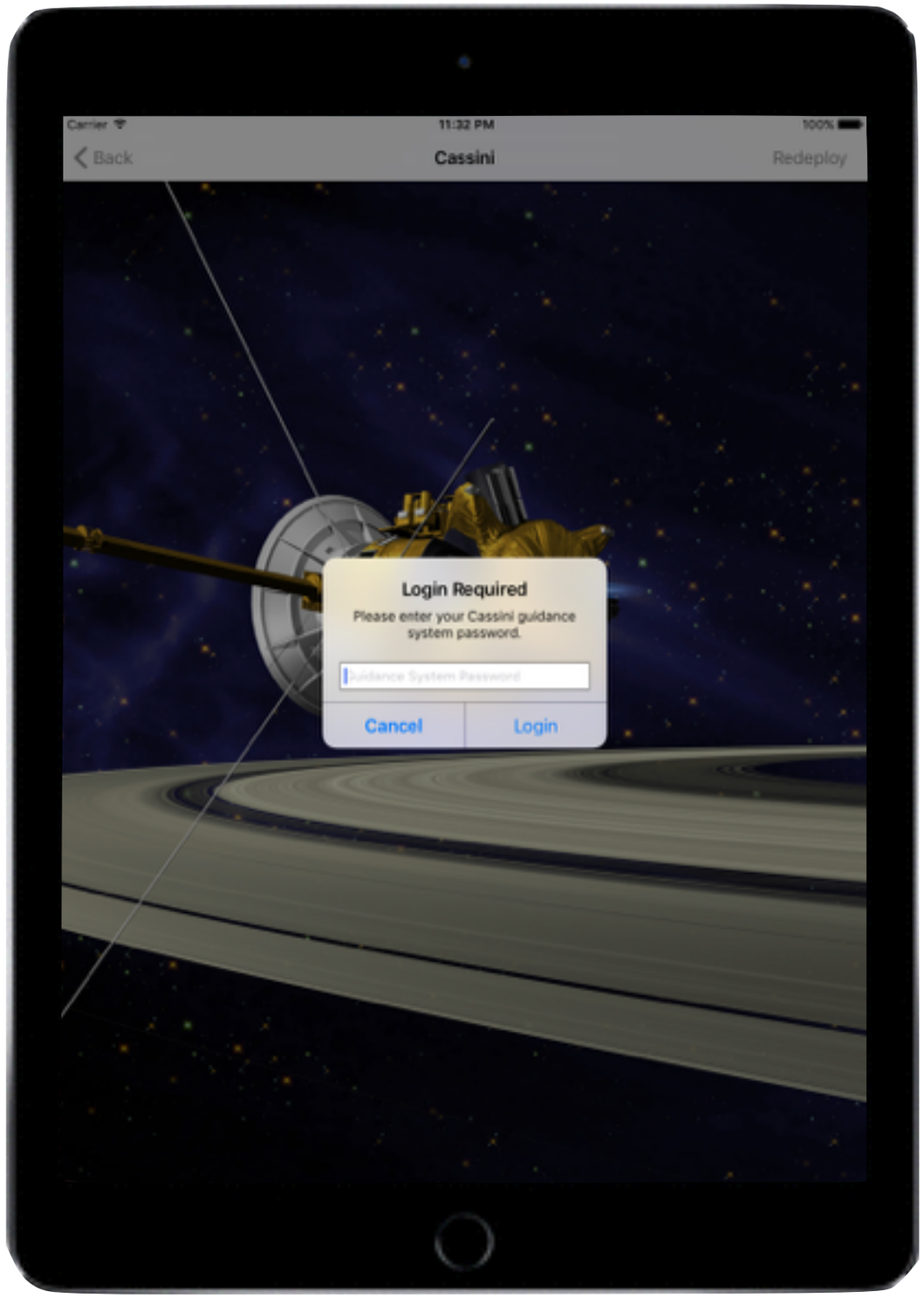


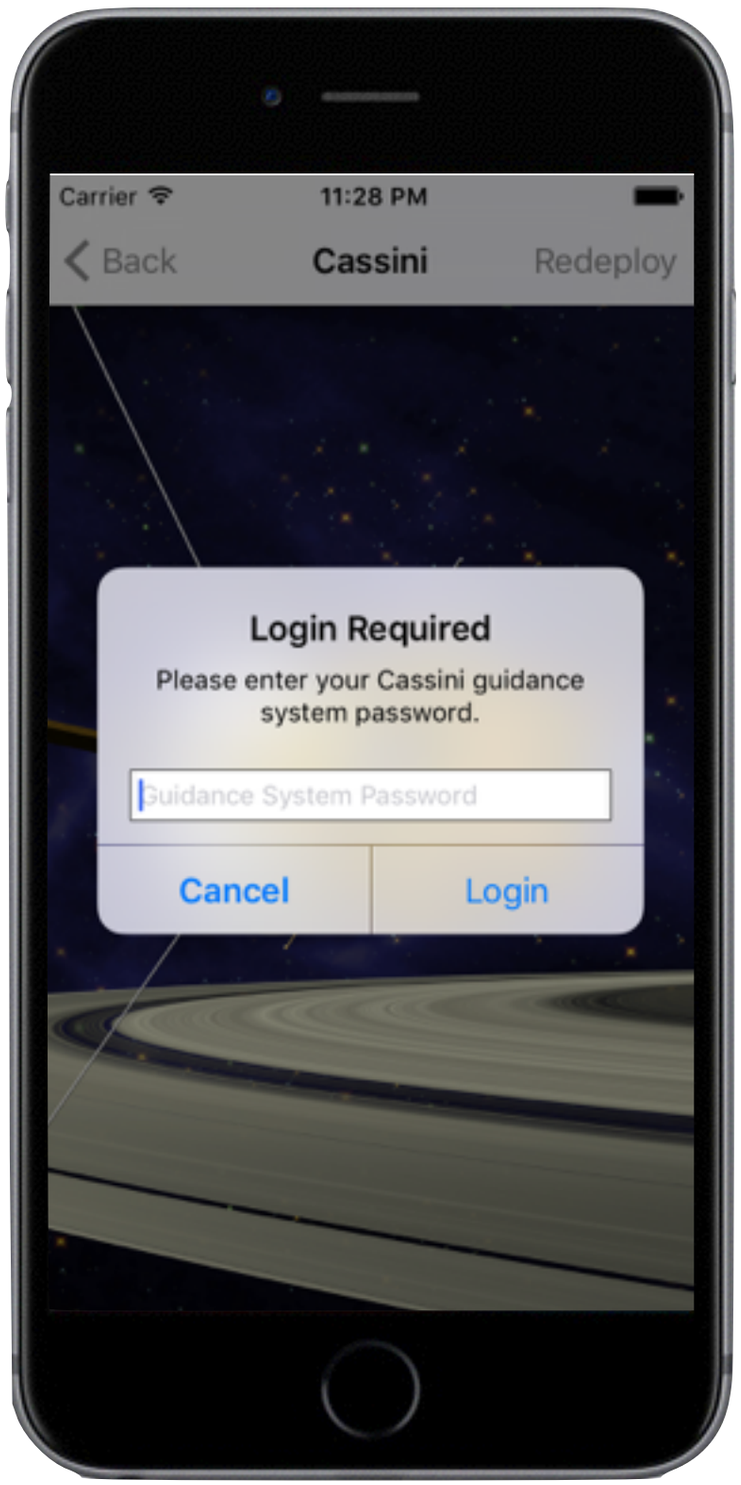
Alerts & Action Sheets

```
let alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...”,
    preferredStyle: UIAlertControllerStyle.Alert)
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: "Login",
    style: .Default)
    { (action: UIAlertAction) -> Void in
        // get password and log in
        let tf = self.alert.textFields?first as? UITextField
        if tf != nil { self.loginWithPassword(tf.text) }
    }
)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = "Guidance System Password"
}

presentViewController(alert, animated: true, completion: nil)
```



Alerts & Action Sheets





NSTimer

- Erstellen eines Timers um eine Methode periodisch aufzurufen

  Kann so erstellt werden, dass er einmal zu einem bestimmten Zeitpunkt in der Zukunft wiederholt ausgeführt wird.

  Wenn wiederholend garantiert das System nicht wann er wieder ausgeführt wird, also nicht "Echt-Zeit".

  Aber für die meisten UI (oder ähnliche) Aktivitäten ist das ok.

  Wird nicht für Animationen verwendet.

  Ist für gröbere Aktivitäten gedacht.

- Run Loops

  Timer arbeiten mit Run Loops (welche wir uns nicht mehr anschauen werden).

  Für unsere Zwecke verwenden wir `NSTimer` in der Main Queue.
  Siehe Doku zum erlernen von Run Loops und Timer in anderen Queues.

NSTimer

- Ausführen eines Timers...

```
class func scheduledTimerWithInterval(
    _ seconds: NSTimerInterval,
    target: AnyObject,
    selector: Selector,
    userInfo: AnyObject?,
    repeats: Bool
)
```

- Beispiel

```
let timer = NSTimer.scheduledTimerWithTimeInterval(2.0,
    target: self, selector: @selector(fire(_:)),
    userInfo: nil,
    repeats: true
)
```

Alle 2 Sekunden wird die Methode `fire(NSTimer)` in `self` aufgerufen.

- Wie sieht die fire Methode aus?

```
func fire(timer: NSTimer) {
    // irgendwas hier machen was alle 2 Sekunden ausgeführt werden soll
    // aber nicht zu lange, findet in der Main Queue statt
    let theTimersUserInfo = timer.userInfo // die userInfo von oben
}
```

NSTimer

- Stoppen eines wiederholenden Timers

  Einfach `invalidate()` für den Timer aufrufen um ihn zu stoppen...

```
func timer(timer: NSTimer) {
    if imDoneWithThisTimer {
        timer.invalidate()
    }
}
```

- Toleranz

  Es ist gut für die Systemperformance eine Toleranz zu setzen für “late firing”.

  Zum Beispiel, wenn wir einen Timer haben der jede Minute ausgeführt wird, ist eine Toleranz von 10 Sekunden eine gute Wahl.

```
myOneMinuteTimer.tolerance = 10 // in Sekunden
```

  Die firing Time ist relativ zum Start des Timers (nicht das letzte mal als er gefired hat), also kein Drift.

Arten von Animationen

- Animation von `UIView` Properties
  Änderung von Dingen wie Bounds oder Transparenz.

- Animationen von View Controller Transitions (wie UINC)
  Außerhalb des Umfangs des Kurses, grundsätzliche Prinzipien sind aber gleich.

- Core Animation
  Unterliegendes sehr mächtiges Animation Framework (auch außerhalb des Umfangs des Kurses).

- OpenGL
  3D

- SpriteKit
  "2.5D" Animation (überlappende Bilder die sich übereinander bewegen, etc.)

- Dynamic Animation
  "Physik"-basierte Animationen.

UIView Animationen

- Änderungen für bestimmte UIView Properties können über die Zeit animiert werden
  `frame`
  `transform` (Translation, Rotation und Skalierung)
  `alpha` (Deckkraft)

- Geschieht mittels UIView Klassenmethoden und Closures
  Die Klassenmethoden nehmen Animation Parameter und einen Animation Block als Parameter entgegen.
  Der Animation Block enthält Code welcher die Änderungen des UIViews durchführt.
  Die Änderungen im Block werden unmittelbar durchgeführt (auch wenn diese erst über die Zeit auftauchen).
  Die meisten haben ebenfalls einen weiteren "Completion Block", der ausgeführt wird wenn die Animation beendet ist.

UIView Animationen

- Animation Klassenmethode in UIView

```
class func animateWithDuration(duration: NSTimeInterval,
                                        delay: NSTimerInterval,
                                      options: UIViewAnimationOptions,
                                   animations: () -> Void,
                                   completion: ((finished: Bool -> Void)?)
```

- Beispiel

```
if myView.alpha == 1.0 {
    UIView.animateWithDurarion(3.0,
                               delay: 2.0,
                             options: [UIViewAnimationOptions.CurveLinear],
                          animations: { myView.alpha = 0.0 },
                          completion: { if $0 { myView.removeFromSuperView() } })
    print("myView.alpha = \(myView.aplha)")
}
```

Dies führt dazu dass myView über 3 Sekunden ausgeblendet wird (beginnend nach 2 Sekunden).

Dann wird myView aus der View Hierarchie entfernt (aber nur, dann das ausblendet abgeschlossen wurde).

Wenn, innerhalb von 5 (3+2) Sekunden, jemand den Alpha Wert auf nicht 0 animiert, würde das entfernen nicht passieren.

Die Ausgabe auf der Konsole zeigt... `myView.alpha = 0.0`

... obwohl der Alpha Wert auf dem Screen noch nicht 0 ist für 5 weitere Sekunden.

UIView Animationen

- `UIViewAnimationOptions`

```
BeginFromCurrentState       // Unterbrechen anderer Animationen für dieses Property
AllowUserInteraction        // Gesten werden verarbeitet, während der Animation
LayoutSubviews              // Animiert Re-Layout von Subviews mit Parent Animation
Repeat                      // unendliche Wiederholung
Autoreverse                 // Animation vorwärts, dann rückwärts
OverrideInheritedDuration   // Wenn n.gesetzt, verwende Dauer der laufenden Anim.
OverrideInheritedCurve      // Wenn n.gesetzt, verwende Curve der laufenden Anim.
AllowAnimatedContent        // Wenn n.gesetzt, interpoliere zwischen akt. und End-"Bits"
CurveEaseInEaseOut          // Anfang und Ende langsam, in der Mitte normal schnell
CurveEaseIn                 // Anfang langsamer, dann normal
CurveLinear                 // Immer gleiche Geschwindigkeit
```

UIView Animationen

- Manchmal soll eine vollständige View Modifikation auf einmal stattfinden

  In einem solchen Fall sind wir nicht auf spezielle Properties wie `alpha`, `frame` oder `transform` beschränkt.

  Flip des gesamten Views `UIViewAnimationOptions.TransitionFlipFrom{Left, Right, Top, Bottom}`

  Auflösen von alten hin zu neuem Zustand `UIViewAnimationOptions.TransitionCrossDissolve`

  Curl up/down `UIViewAnimationOptions.TransitionCurl{Up,Down}`

- Wieder Nutzung von Closures mit diesen `UIView` Klassenmethoden

```
UIView.transitionWithView(view: UIView,
                         duration: NSTimeInterval,
                          options: UIViewAnimationOptions,
                       animations: () -> Void,
                       completion: ((finished: Bool) -> Void)?)
```

UIView Animationen

- Beispiel

  Spielkarte umdrehen…

```
UIView.transitionWithView(view: myPlayingCardView,
                  duration: 0.75,
                   options: [UIViewAnimationOptions.TransitionFlipFromLeft],
                animations: {cardIsFaceUp = !cardIsFaceUp }
                completion: nil)
```

  Unter der Annahme, dass sich `myPlayingCardView` sich selbst Face Up oder Down zeichnet und Abhängigkeit von `cardIsFaceUp`.

  Dies führt dazu, dass die Karte sich dreht (von der linken Kante der Karte).

UIView Animationen

- Animieren von Änderungen in der View Hierarchie ist etwas anders

  Anders ausgedrückt möchten wir das Hinzufügten/Entfernen (oder auch Verstecken/Zeigen) von Subviews animieren.

```
UIView.transitionFromView(fromView: UIView,
                            toView: UIView,
                          duration: NSTimeInterval,
                           options: UIViewAnimationOptions,
                        completion: ((finished: Bool) -> Void)?)
```

  `UIViewAnimationOptions.ShowTransitionViews` wenn wir das `hidden` Property verwenden wollen.

  Ansonsten wird `fromView` aus der View Hierarchie entfernt und `toView` hinzugefügt.

Dynamic Animation

- Etwas anders als UIView-basierte Animationen

  Physik für animierbare Objekte setzen und diese laufen lassen, bis sie zum Stillstand kommen.

  Es ist möglich dies so zu setzen dass sie nie zum Stillstand kommen, kann aber ein Performance Problem sein.

- Schritte

  Erstellen eines `UIDynamicAnimator`.

  Hinzufügen vom `UIDynamicBehavior` dazu (Schwerkraft, Kollision, etc.).

  Hinzufügen von `UIDynamicItem`s (normalerweise `UIView`s) zu den `UIDynamicBehavior`s (`UIDynamicItem` ist ein Protokoll welches von `UIView` implementiert wird).

  Das war's. Animationen starten unmittelbar!

Dynamic Animation

- Erstellen eines UIDynamicAnimator

```
var animator = UIDynamicAnimator(referenceView: UIView)
```

  Wenn Views animiert werden müssen alle Views in einer View Hierarchie sein mit dem `referenceView` ganz oben.

- Erstellen und Hinzufügten von `UIDynamicBehavior` Instanzen

  z.B.
```
let gravity = UIGravityBehavior()
animator.addBehavior(gravity)
```

  z.B.
```
let collider = UICollisionBehavior()
animator.addBehavior(collider)
```

- Hinzufügen von `UIDynamicItems` zu `UIDynamicBehavior`

```
let item1: UIDynamicItem = ... // normal ein UIView
let item2: UIDynamicItem = ... // normal ein UIView
gravity.addItem(item1)
collider.addItem(item1)
gravity.addItem(item2)
```

  `item1` und `item2` werden beide durch die Schwerkraft beeinflusst.
  `item2` kollidiert mit anderen Collider Items oder Boundaries, aber nicht mit `item2`.

Dynamic Animation

- `UIDynamicItem` Protokoll

  Jedes animierbare Item muss dies implementieren…

```
protocol UIDynamicItem {
    var bounds: CGRect { get }                    // Size kann nicht anim. werden
    var center: CGPoint { get set }               // Position kann
    var transform: CGAffineTransform { get set } // und auch Rotation
}
```

  `UIView` implementiert dieses Protokoll.

  Wenn `center` oder `transform` geändert wird während der Animator läuft, muss diese Methode in `UIDynamicAnimator` aufgerufen werden…

```
func updateItemUsingCurrentState(item: UIDynamicItem)
```

Behavior

- `UIGravityBehavior`

```
var angle: CGFloat      // in Radian; 0 = rechts, CCW
var magnitude: CGFloat // 1.0 ist 1000 Points / s^2
```

- `UIAttachmentBehavior`

```
init(item: UIDynamicItem, attachedToAnchor: CGPoint)
init(item: UIDynamicItem, attachedToItem: UIDynamicItem)
init(item: UIDynamicItem, offsetFromCenter: CGPoint, attachedToItem/Anchor…)
var length: CGFloat // Distanz zw. attachten Dingen (settable während Animation!)
var anchorPoint: CGPoint // Ebenfalls jederzeit settable (auch bei Animationen)
```

  Attachment kann oszillieren (wie eine Feder) und Frequenz und Dämpfung kann kontrolliert werden.

Behavior

- `UICollisionBehavior`

  `var collisionMode: UICollisionBehaviorMode` // `.Items`, `.Boundaries` oder `.Everything`

  Wenn `.Items`, dann prallen alle Items die zu einer `UICollisionBehavior` hinzugefügt wurden voneinander ab.

  Wenn `.Boundaries`, dann müssen wir für ein Item `UIBezierPath` Boundaries hinzugefügt werden…

  ```
  func addBoundaryWithIdentifier(identifier: NSCopying, forPath: UIBezierPath)
  func removeBoundaryWithIdentifier(identifier: NSCopying)
  var translatesReferenceBoundsIntoBoundary: Bool // Referece View Edges verw.?
  ```

Behavior

- `UICollisionBehavior`

  Herausfinden wann/ob eine Kollision stattfindet?

  ```
  var collisionDelegate: UICollisionBehaviorDelegate
  ```

  ... diese Methoden werden ausgeführt...

  ```
  func collisionBehavior(behavior: UICollisionBehavior,
        began/endedContactForItem: UIDynamicItem,
           withBoundaryIdentifier: NSCopying) // auch withItem:atPoint:
  ```

  Der `withBoundaryIdentifier` ist derjenige der an `addBoundaryWithIdentifier()` übergeben wird.

  Ist ein `NSCopying` (`NSString` & `NSNumber` sind beide `NSCopying`, also funktionieren `String` & `Int/Double`).

  In dieser Delegate Methode müssen wir das `NSCopying` casten mit (`as` oder `as?`) in was wir erhalten wollen.

Behavior

- UISnapBehavior
  `init(item: UIDynamicItem, snapToPoint: CGPoint)`
  Vorstellen wie vier Federn in vier Ecken um das Item im neuen Spot.
  Die Dämpfung der “Federn” kann kontrolliert werden… `var damping: CGFloat`

- UIPushBehavior
  `var mode: UIPushBehaviorMode // .Continuous oder .Instantaneous`
  `var pushDirection: CGVector`
  … oder…
  `var angle: CGFloat` // in Radian und …
  `var magnitude: CGFloat` // 1.0 bewegt ein 100x100 View mit 100 Points/s^2

Wenn der Push `.Instantaneous` ist, was passiert nachdem er fertig ist?
“Sitzt herum” und verschwendet Speicher.
Wir schauen uns gleich an, wie wir dies beheben können.

Behavior

- `UIDynamicItemBehavior`

  Spezielle “Meta” Behavior.

  Kontrolliert Behavior von Items, wenn diese von anderen Behaviors beeinflusst werden.

  Jedes Item welches zu dieser Behavior hinzugefügt wird (mittels addItem) ist betroffen durch…

```
var allowsRotation: Bool
var friction: CGFloat
var elasticity: CGFloat
```

  … und andere, siehe Doku.

  Wir können ebenfalls Informationen über Items mit dieser Behavior erhalten…

```
func linearVelocityForItem(UIDynamicItem) -> CGPoint
func addLinearVelocity(CGPoint, forItem: UIDynamicItem)
func angularVelocityForItem(UIDynamicItem) -> CGFloat
```

  Mehrere `UIDyanmicItemBehaviors` die das gleiche Item beeinflussen ist ein “fortgeschrittenes” Thema.

Behavior

- `UIDynamicBehavior`

  Superclass von Behaviors.

  Wir können unsere eigene Subclass erstellen, welches eine Kombination von anderen Behaviors ist.

  Normalerweise überschreiben wir `init` Methoden und `addItem` sowie `removeItem`...

  ```
  func addChildBehavior(UIDyanmicBehavior)
  ```

  Gute Art und Weise physikalisches Verhalten zu kapseln, welches eine Kombination von anderen Behaviors ist.
  Vielleicht möchten wir auch API um das Konfigurieren unserer Subclass für dessen Children zu erlauben.

- Alle Behaviors kennen den `UIDyanmicAnimator` vom dem sie ein Teil sind

  Sie können nur Teil von einem zu einem Zeitpunkt sein.

  ```
  var dynamicAnimator: UIDynamicAnimator { get }
  ```

  Und die Behavior wird benachrichtigt wenn der Animator sich ändert...

  ```
  func willMoveToAnimator(UIDynamicAnimator)
  ```

Behavior

- `UIDynamicBehavior`'s `action` Property

  Jedes mal wenn die Behavior sich auf ein Item auswirkt, wird dieser Block von Code ausgeführt...

  ```
  var action: (() -> Void)?
  ```

  (`action`, die keine Parameter annimmt und nichts zurück gibt)

  Kann so gesetzt werden, dass beliebige Dinge ausgeführt werden.

  Wird sehr häufig ausgeführt, sollte also sehr effizient sein.

  Wenn die `action` auf Properties in der Behavior selbst zugreift, Memory Cycles beachten!

Stasis

- `UIDynamicAnimators`'s Delegate benachrichtigt, wenn die Animation pausiert

  Einfach Delegate setzen…

  ```
  var delegate: UIDynamicAnimatorDelegate
  ```

  … und wird können herausfinden wenn die Animation pausiert und weiter geht…

  ```
  func dynamicAnimatorDidPause(UIDynamicAnimator)
  func dynamicAnimatorWillResume(UIDynamicAnimator)
  ```

Memory Cycles

- Beispiel für Verwendung von `action` und Vermeidung von Memory Cycles

  Zurück zum Beispiel einer `.Instantaneous UIPushBehavior`

  Wenn sie damit fertig ist auf ihre Items auszuwirken, wäre es hilfreich diese aus dem Animator zu entfernen.

  Dies können wir über die `action` Methode erreichen, müssen aber vorsichtig bzgl. Memory Cycles sein...

```
if let pushBehavior = UIPushBehavior(items: [...], mode: .Instantaneous) {
    pushBehavior.magnitude = ...
    pushBehavior.angle = ...
    pushBehavior.action = {
        pushBehavior.dynamicAnimator!.removeBehavior(pushBehavior)
    }
    animator.addBehavior(pushBehavior) // pushed unmittelbar
}
```

Das obige hat einen Memory Cycle da die action einen Pointer zu sich selbst captured.

Also kann weder das Action Closure noch die PushBehavior je den Heap verlassen.

Memory Cycles

- Beispiel für Verwendung von `action` und Vermeidung von Memory Cycles

  Zurück zum Beispiel einer `.Instantaneous UIPushBehavior`

  Wenn sie damit fertig ist auf ihre Items auszuwirken, wäre es hilfreich diese aus dem Animator zu entfernen.

  Dies können wir über die `action` Methode erreichen, müssen aber vorsichtig bzgl. Memory Cycles sein...

```
if let pushBehavior = UIPushBehavior(items: [...], mode: .Instantaneous) {
    pushBehavior.magnitude = ...
    pushBehavior.angle = ...
    pushBehavior.action = { [unowned pushBehavior] in
        pushBehavior.dynamicAnimator!.removeBehavior(pushBehavior)
    }
    animator.addBehavior(pushBehavior) // pushed unmittelbar
}
```

Die `pushedBehavior` wird nun nicht länger gecaptured.

Es ist safe als `unowned` zu markieren da solange das `action` Closure existiert auch die `pushBehavior` existiert.

Wenn die `pushBehavior` sich selbst entfernt aus dem Animator, wird die `action` nicht im Speicher gehalten.

Sie verlassen also beiden den Heap da der Animator nicht länger auf die Behavior zeigt.