OTHER STUFF

Programming Handheld Systems—iOS

CMSC 436
Spring 2019

TODAY

• Gestures
• Animation
• TabBarController
• NavigationController
GESTURES

- We can get raw touches, or predefined gestures.
- Captured by instances of UIGestureRecognizer
- Two parts:
  - Add recognizer to a UIView
    - Usually done by a UIViewController
  - Define method to handle gesture
    - Either UIViewController or UIView

If we want a UIView to recognize a pan:
```swift
@IBOutlet weak var panViewOutlet: UIView {
    didSet {
        let panRecognizer = UIPanGestureRecognizer(
            target: self, action: #selector(doPan))
        panViewOutlet.addGestureRecognizer(panRecognizer)
    }
}
```

UIPanGestureRecognizer has three methods:
- `func velocity(in: UIView?) -> CGPoint`    // how fast
- `func translation(in: UIView?) -> CGPoint` // since start
  - `func setTranslation(CGPoint, in: UIView?)`
    - incremental translations, snap-to-grid
- `var state UIGestureRecognizerState { get }`
  - .possible
  - .began - .changed - .changed - .changed - .ended
  - .failed, .cancelled
GESTURES

- So the pan handler looks like the following:
  ```swift
  @objc func doPan(gesture: UIGestureRecognizer) {
    switch gesture.state {
      case .changed: fallthrough
      case .ended:
        let howFar = gesture.translation(in: panViewOutlet)
        // update things
      default:
        break
    }
  }
  ```
- We get notifications whenever finger moves or goes away, our action is often the same in either case.
- We end by getting the translation in the target view’s coordinates.

- WeIRotationGestureRecognizer
  - var rotation: CGFloat { get set }
  - var velocity: CGFloat { get }
- UIPinchGestureRecognizer
  - var scale: CGFloat { get set }
  - velocity: CGFloat { get }
- UISwipeGestureRecognizer (assign3)
  ```swift
  let gesture = UISwipeGestureRecognizer(target: self, action:@selector(doSwipe))
  gesture.direction = .right
  board.addGestureRecognizer(gesture)
  ```
- UITapGestureRecognizer
- UILongPressRecognizer
ANIMATION

- UIView property animations
- Transitions
- Dynamic Animator

UIVIEW ANIMATION

- Easy animation of subview properties:
  - frame, center
  - transform (translation, rotation, scale)
  - alpha (transparency)
  - color
- Done by:
  - making changes in a closure
  - passing closure to `UIViewPropertyAnimator`
    - changes are immediate
    - become visible over time
  - completion closure allows more changes at animation end
UIView.animate(withDuration: LEN, animations: {
    but.frame = self.butRect(position: i)
    but.backgroundColor = self.colors.tileBG(value: val)
}, completion: { finished in
    but.setTitleColor(self.colors.tileFG(value: val),
    for: UIControl.State.normal)
    but.setTitle((val == 0) ? " " : "\(1<val)"),
    for: UIControl.State.normal)
})

- other possible parameters (other initializers)
  - delay: before start
  - options:
    - allowAnimatedContent, allowUserInteraction, autoreverse,
      beginFromCurrentState, curveEaseIn, curveEaseInEaseOut, curveLinear,
      layoutSubviews, overrideInheritedCurve, overrideInheritedDuration, repeat
      ...

TRANSITIONS

- Modify entire view at once
  - flip view over
  - curl up or down

UIView.transition(with: tileView,
    duration: 1.25,
    options: [.transitionFlipFromRight],
    animations: { redrawMe() };
    completion: nil)
DYNAMIC TRANSLATIONS

- Create a UIDynamicAnimator
  - var animator = UIDynamicAnimator(referenceView: UIView)
  - If animating views, all views must be in a view hierarchy with referenceView at the top.
- Create and add UIDynamicBehavior instances
  let gravity = UIGravityBehavior()
  animator.addBehavior(gravity)

  collisions = UICollisionBehavior()
  animator.addBehavior(collisions)

- Add UIDynamicItems to a behavior
  - implemented by UIView, UICollectionViewLayoutAttributes
    - gravity.addItem(button1)
    - gravity.addItem(button2)
    - collisions.addItem(button1)

- differing behaviors
  - button1 affected by both gravity and collisions
  - button2 affected only by gravity
DYNAMIC ANIMATION

- **UIDynamicItem protocol**
  - Must be implemented by any animatable item

- **UIView** implements this protocol

- You must call this method in **UIDynamicAnimator** ...
  
  ```swift
  func updateItemUsingCurrentState(item: UIDynamicItem)
  ```

```swift
protocol UIDynamicItem {
    var bounds: CGRect { get }
    var center: CGPoint { get set }
    var transform: CGAffineTransform { get set } // e.g. rotation
    var collisionBoundsType: UIDynamicItemCollisionBoundsType { get set }
    var collisionBoundingPath: UIBezierPath { get set }
}
```

- If center is translated or transformed while animator is running, call:

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

BEHAVIORS

- **UIAttachmentBehavior**
  
  ```swift
  init(item: UIDynamicItem, attachedToAnchor: CGPoint)
  init(item: UIDynamicItem, attachedTo: UIDynamicItem)
  init(item: UIDynamicItem, offsetFromCenter: CGPoint, attachedTo[Anchor]...)
  ```

  - var length: CGFloat // distance between, change on fly
  - var anchorPoint: CGPoint // also change on fly
  - Attachments can oscillate like a spring, control both frequency and damping

```swift
CMSC 436 (iOS) • Spring 2019 • Pete Keleher
```
BEHAVIORS

• UICollisionBehavior  
  • var collisionMode: UICollisionBehaviorMode // .items, .boundaries, .everything
  • Views can bounce off each other with .item
  • Define .boundaries through UIBezierPath():
    func addBoundary(withIdentifier: NSCopying, for: UIBezierPath)
    func addBoundary(withIdentifier: NSCopying, from: CGPoint, to: CGPoint)
    func removeBoundary(withIdentifier: NSCopying)

    // referenceView’s edges
    var translatesReferenceBoundsIntoBoundary: Bool

• NSCopying means NSString or NSNumber, but you can cast to String or Int
  • using as

BEHAVIORS

• UISnapBehavior  
  • init(item: UIDynamicItem, snapTo: CGPoint)
  • Spring-like dampening

• UIPushBehavior  
  var mode: UIPushBehaviorMode // .continuous or .instantaneous
  var pushDirection: CGVector
  ... or ...
  var angle: CGFloat // in radians
  // 1.0 moves a 100x100 with density 1.0 view at 100 pts/s^2
  var magnitude: CGFloat
META BEHAVIORS

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

• and others...

UIDynamicBehavior

• create subclass from multiple simple behaviors
• add all items of a class to the same combination
  • override init(), or addItem() etc.
TAB BAR CONTROLLER

• Draw out in IB and pretty much done!
• But...navigation...

• Your view controller has the following properties:
  • var viewControllers: [UIViewController]? { get set }
  • // in order (left-to-right (tab), master-detail (split), root etc. (nav)

• References to enclosing view controllers
  • var tabBarController: UITabBarController? { get }
  • var splitViewController: UISplitViewController? { get }
  • var navigationController: UINavigationController? { get }

• Navigation
  • if let highController =
    tabBarController?.viewControllers?[1] as? HighController {
    • tabBarController?.selectedIndex = 1

NAVIGATION CONTROLLER

Pushes and pops off of a stack. “Pushed” view controllers are always brand new, and are destroyed as they are popped off.
NAVIGATION CONTROLLER
NAVIGATION CONTROLLER

Can add buttons via UIViewController’s toolbarItems to