VIEWS

• What is a view?
  • Subclass of a `UIView`
  • defines coord space for drawing, touch events
• There is a hierarchy
  • Top is effectively `var view UIView`
    • `view`’s bounds can change on rotation
  • Each view has a parent: `var superview: UIView?`
  • Potentially children: `var subviews: [UIView]`
    • Order matters
    • Views can be clipped to their `bounds`, or (usually) not

VIEW_INITIALIZATION

• Avoid if possible, but:
  • `init(frame: CGRect)`     // from code
  • `init(coder: NSCoder)`   // from a storyboard
• If you have one, you must have both:

```swift
class PeteView: UIView {
  override init(frame: CGRect) {
    super.init()
    setup()
  }

  required init?(coder aDecoder: NSCoder) {
    super.init()
    setup()
  }

  func setup() {
    // do stuff
  }
}
```
CG TYPES

- **CGFloat**
  - Used everywhere in UI
  - Must convert other numbers: `let aCG = CGFloat(42)`
- **CGPoint**
  - `var pt = CGPoint(x: 42, y: 0)`
- **CGSize**
  - `var sz = CGSize(width: 42, height: 42)`

CGRect

```swift
struct CGRect {
    var origin: CGPoint
    var size: CGSize
}
```

// And other methods and properties:
```swift
let r = CGRect(x: CGFloat, y: CGFloat,
               width: CGFloat, height: CGFloat)
```

Many methods and properties:
```swift
    var minX: CGFloat
    var midY: CGFloat
    .intersects(CGRect) -> Bool
    .contains(CGPoint) -> Bool
    ...
```
VIEW COORDINATE SYSTEM

- origin in upper left
- units are points, not pixels
  - Pixels are min units per-device
  - Point is an integer multiple of pixel
    - 2 or 3 for retina
    - 1 for older
    - a view’s contentScaleFactor

- A view’s **view** of itself
  - var bounds: CGRect

- A view’s **place** in the superview
  - var center: CGPoint
  - var frame: CGRect

FRAME VS BOUNDS

frame is position in superview
bound is a view local view of itself

view.bounds = CGRect(x: 0, y: 0, width: 100, height: 100)
view.frame = CGRect(x: 100, y: 100, width: 241, height: 241)
CREATING VIEWS

• Two ways:
  • Xcode interface builder (storyboard)
    • pull out UIView from object library
    • use Identity Inspector to specify the (custom) class of the new view
  • From code:
    
    ```swift
    let r = CGRect(x: 0, y: 0, width: 42, height: 54)
    let button = UIButton(frame: r)
    button.setTitle("Hello, world", for: UIControl.State.normal)
    UIView.addSubview(button)
    ```

DRAW: UIColor

• Colors are of type UIColor
  • some special colors: UIColor.green
  • try typing “colorliteral” into Xcode, and then double-click on the white splotch (you get a ColorPicker)

• Colors can have transparency (alpha)
  • UIColor.orange.withAlphaComponent(0.25)

• Make drawing transparent, no effect on UIButton etc.
  • var opaque = false

• Make entire view transparent
  • var alpha = 0.25
**Draw: CORE ANIMATION LAYER**

- UIView.layer: CALayer
- Useful stuff:
  - .layer.cornerRadius
  - .layer.borderWidth
  - ...
- It’s own color nomenclature:
  - UIColor.green.cgColor
- Animation
  - ...soon...

**Draw: TRANSPARENCY**

- Subviews are ordered
  - front view occlude those in back
  - transparent front views let back views show through
- .isHidden: Bool
**Draw: Fonts**

- Can change fonts in views like UIButton etc.
- Get preferred font for a given text style:
  ```swift
  static func preferredFont(forTextStyle: UIFontTextStyle) -> UIFont {
    .headline
    .body
    .footnote
  }
  ```
- Or through more flexible ways:
  ```swift
  let font = UIFont(name: “Menlo”, size: 24)
  ```
  - or through a FontPicker (attributes pane)
  - or scale to user’s preferred size:
    ```swift
    let metrics = UIFontMetrics(forTextStyle: .body)
    let fontToUse = metrics.scaledFont(for: font)
    ```
  - or use system fonts
    ```swift
    static func systemFont(ofSize CGFloat) -> UIFont
    static func boldSystemFont(ofSize CGFloat) -> UIFont
    ```

**Draw: Drawing Text**

- UILabels are easy, but...
- Draw using NSAttributedString:
  ```swift
  let text = NSAttributedString(string: “hello”)  
  text.draw(at: CGPoint(x: 42, y: 11))  
  let textSize: CGSize = text.size
  ```
DRAWING

• How?
  • Subclass UIView and override draw()
    • `override func draw(_ rect: CGRect)`
    • You can draw outside of rect.
    • Rect is just a hint
    • `.bounds` describes a view’s entire area
  • You do not ever call `draw()` instead:
    • `.setNeedsDisplay()`
    • system gathers up multiple redraws into a single event

DRAWING

• How to implement drawing?
  • Either get a drawing context (printing, double-buffering), or
  • UIBezierPath (easy)
  • UIBezierPath
    • create `paths` from lines, arcs, rounded rects etc.
    • set colors, linewdiths, fonts, etc.
    • optionally set clipping to a closed path
    • stroke or fill
• Create a path
  let path = UIBezierPath()

• Move, add lines and arcs
  path.move(to: CGPoint(x: 40, y: 40))
DRAWING

• Create a path
let path = UIBezierPath()

• Move, add lines and arcs
path.move(to: CGPoint(x: 40, y: 40))
path.addLine(to: CGPoint(x: 80, y: 40))
**DRAWING**

- Create a path
  ```swift
  let path = UIBezierPath()
  ```
- Move, add lines and arcs
  ```swift
  path.move(to: CGPoint(x: 40, y: 40))
  path.addLine(to: CGPoint(x: 80, y: 40))
  path.addLine(to: CGPoint(x: 80, y: 80))
  path.addLine(to: CGPoint(x: 40, y: 80))
  ```
- Close (optional)
  ```swift
  path.close()
  ```
• Create a path

```
let path = UIBezierPath()
```

• Move, add lines and arcs

```
path.move(to: CGPoint(x: 40, y: 40))
path.addLine(to: CGPoint(x: 80, y: 40))
path.addLine(to: CGPoint(x: 80, y: 80))
path.addLine(to: CGPoint(x: 40, y: 80))
```

• Close (optional)

```
path.close()
```

• Set attributes

```
UIColor.blue.setStroke()
UIColor.red.setFill()
path.lineWidth = 3
```

• Fill/Stroke

```
path.stroke()
```
DRAWING

• Create a path
  let path = UIBezierPath()

• Move, add lines and arcs
  path.move(to: CGPoint(x: 40, y: 40))
  path.addLine(to: CGPoint(x: 80, y: 40))
  path.addLine(to: CGPoint(x: 80, y: 80))
  path.addLine(to: CGPoint(x: 40, y: 80))

• Close (optional)
  path.close()

• Set attributes
  UIColor.blue.setStroke()
  UIColor.red.setFill()
  path.lineWidth = 3

• Fill/Stroke
  path.stroke()
  path.fill()

SHAPES

• Add curves:
  path.addCurve(to: pt1, controlPoint1: pt2, controlPoint2: pt3)
  path.addQuadCurve(to: pt1, controlPoint: pt2)

• Add common shapes:
  path.append(UIBezierPath(ovalIn: r))
  path.append(UIBezierPath(rect: r))
  path.append(UIBezierPath(roundedRect: r, cornerRadius: 0))
  path.append(UIBezierPath(roundedRect: r, cornerRadius: r.width / 2.0))

• .intersects(), etc.
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

```swift
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, begunFromCurrentState, curveEaseIn, curveEaseInEaseOut, curveLinear, layoutSubviews, overrideInheritedCurve, overrideInheritedDuration, repeat ...
TRANSITIONS

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

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

DYNAMIC ANIMATIONS

Set up physics....and watch them go.

• 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 to the animator
  let gravity = UIGravityBehavior()
  animator.addBehavior(gravity)

  collisions = UICollisionBehavior()
  animator.addBehavior(collisions)
DYNAMIC ANIMATIONS

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

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

BEHAVIORS

• UIGravityBehavior

  // in radians, 0 to right, positive clockwise
  var angle: CGFloat

  // 1.0 is 1000 points/sec/sec (seems like 1G)
  var magnitude: CGFloat

BEHAVIORS

• UIAttachmentBehavior

  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
  • can be to another dynamic object, or to a point
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`

  *Note that objects can escape*

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\textsc{BEHAVIORS}

```swift
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.

---

MIDTERM

- Stats:
  - max: 93
  - min: 20
  - median: 62
  - stdev: 14
  - rough grades (entire class is curved, not single assignments)
    - A: 76
    - B: 62
    - C: 48

- Regrades open on gradescope:
  - due sunday (3/8) midnight