CSC 581: Mobile App Development

Spring 2018

Unit 3: Optionals & segues

- Swift Optionals
- other Swift features:
  - guards, variable scope, enumerations
- multiscreen apps with segues
  - passing data, Navigation Controllers
  - popover windows
- Example app: Concentration

Optionals

recall: Swift uses Optionals when a value may or may not be present

```
var ages = ["Pat": 18, "Chris": 20, "Kelly": 19]
print(ages["Chris"])                → Optional(20)

outputLabel.text = "Hello world"
print(outputLabel.text)             → Optional("Hello world")
```

- you can unwrap an optional using ! or (more elegantly) if-let

```
print(ages["Chris"]!)               → 20

if let msg = outputLabel.text {
    print(msg)                      → Hello world
}
```

every type in Swift has a corresponding optional type (with ? at end)

```
var age: Integer? = ages["Chris"]

var msg: String? = outputLabel.text
```

Intentional Optionals

structs/classes can have optional fields; functions can return Optionals

```
struct GradeList {
    private var grades: [Int] = []
    var lowest: Int? {
        return self.grades.min()  // note: min returns nil if empty
    }

    mutating func recordGrade(of grade: Int) {
        self.grades.append(grade)
    }

    func average() -> Double {
        if grades.count == 0 {
            return 0.0
        }
        else {
            var sum = 0
            for g in self.grades {
                sum += g
            }

            if self.grades.count > 2 {
                return Double(sum-self.lowest!)/Double(self.grades.count-1)
            }
            else {
                return Double(sum)/Double(self.grades.count)
            }
        }
    }
}
```

3

Guards

when defining functions, will often have an initial validity test or tests

- a guard is a cleaner notation for handling validity tests

```
guard condition else {  }
```

```
func average() -> Double {
    guard grades.count > 0 else { return 0.0 }

    var sum = 0
    for g in self.grades {
        sum += g
    }

    if self.grades.count > 2 {
        return Double(sum-self.lowest!)/Double(self.grades.count-1)
    }
    else {
        return Double(sum)/Double(self.grades.count)
    }
}
```

4

Variable scope

as in most languages, a block (code enclosed in { }) defines a new scope for variables

- note: a function defines a new scope as well as control statements

```
var x = 100
if true {
    var y = 3
    print("\(x) \(y)")          → 100 3
}
print(x)                        → 100
print(y)                        → ERROR: Use of unresolved identifier 'y'
```

variable shadowing

- unlike Java, you can override an existing variable inside a new scope

```
var x = 100
if true {
    var x = 3
    print(x)                    → 3
}
print(x)                        → 100
```

5

Enumerations

as in Java, you can define a new type by enumerating all of the possible values of that type

- useful when you want a limited range of values to be assignable

```
enum Answer {
    case yes, no, maybe
}


enum DayOfWeek {
    case Monday, Tuesday, Wednesday, Thursday,
         Friday, Saturday, Sunday
}


let response = Answer.yes

if (response == .yes || response == .no) {    // Swift infers the
    print("That's a decisive answer")         // enum type
}


let today: DayOfWeek = .Tuesday

print(today)
```

6

Multiscreen apps

a Single View app automatically has one screen on the storyboard

- can add a screen by dragging a View Controller from the Object Library
- can then specify segues (transitions) between the screens

- segues are defined in Interface Builder by connecting an action to a screen
  e.g., place a button in a screen, control-drag to the other screen
  alternatively, could use a tap or swipe gesture to initiate a segue

- can select the presentation method for the transition
  e.g., `Show` performs a sliding transition between screens
  e.g., `Present Modally` allows you to customize (dissolve, curl, …)

the initial screen (a.k.a. entry point) is identified with an arrow

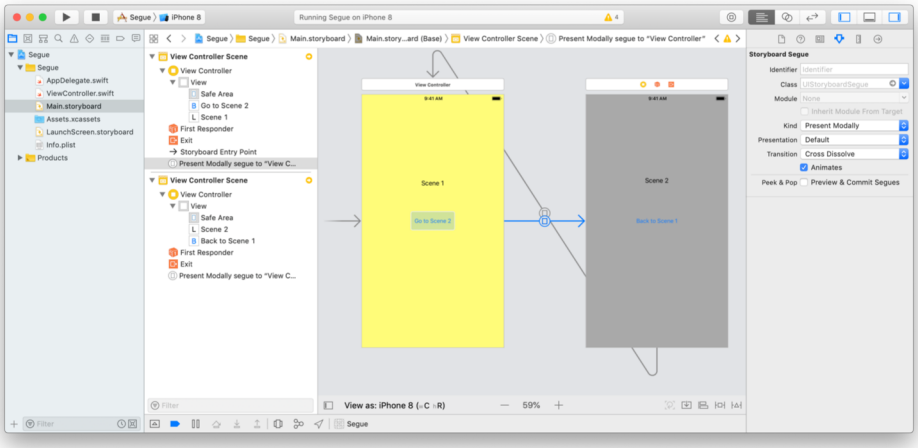
- can drag it from one screen to another to select the entry point

7

Simple segue

2 screens (view controllers), each with a button that segues to the other screen

- arrows in the storyboard show the segues between the screens
- can select an arrow and inspect or change its properties (e.g., transition type)
- here, scene 1 is the entry point



8

Controlling each screen

technically, each screen is a View Controller element

- each can have its own UIViewController class to specify its behavior
- Single View app automatically creates one UIViewController class that can be associated with a screen (default name ViewController.swift)
- if each screen has its own behavior, then each needs its own UIViewController

to add a new UIViewController to the project

- ✓ under the File menu, select New → File → Cocoa Touch Class
- ✓ enter the desired class name (e.g., Scene1Controller) and specify that it is a subclass of UIViewController
- ✓ associate that new class with the appropriate screen by clicking on the bar at the top of the screen and selecting the class name in the Identity Inspector

the default name was fine when there was only one UIViewController

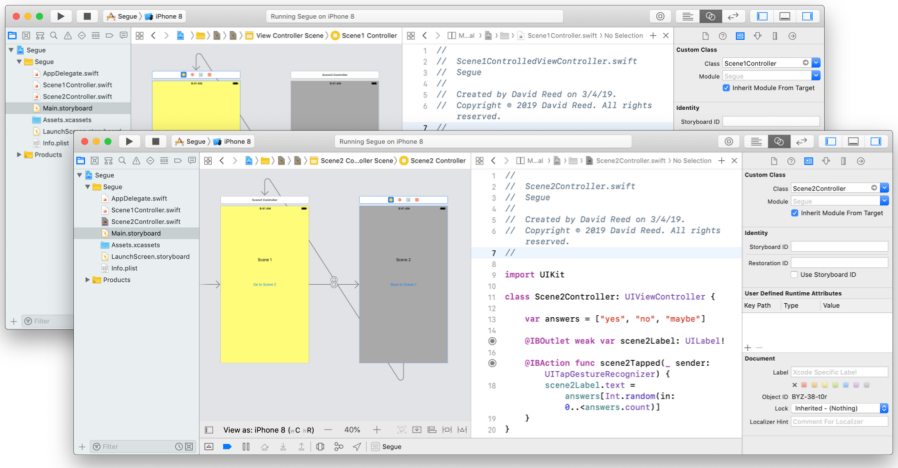
- if you are going to have multiple controllers, give them reasonable names
- you can always rename a variable/class by highlighting its name, right-clicking, and selecting Refactor

9

Multiple controllers

supposed we wanted each scene to have its own behavior

- when you tap on scene 1, the background color changes at random
- when you tap on screen 2, the label displays a random word

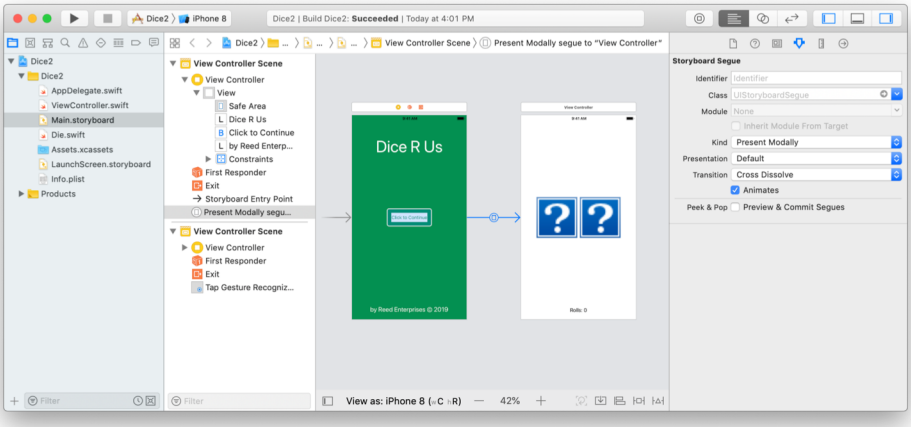


10

Welcome screen

a common feature in an app is to have a welcome screen

- may contain developer info, instructions, initial settings, …
- if the welcome screen does nothing other than present info and segue to the main screen, it does not require its own UIViewController class

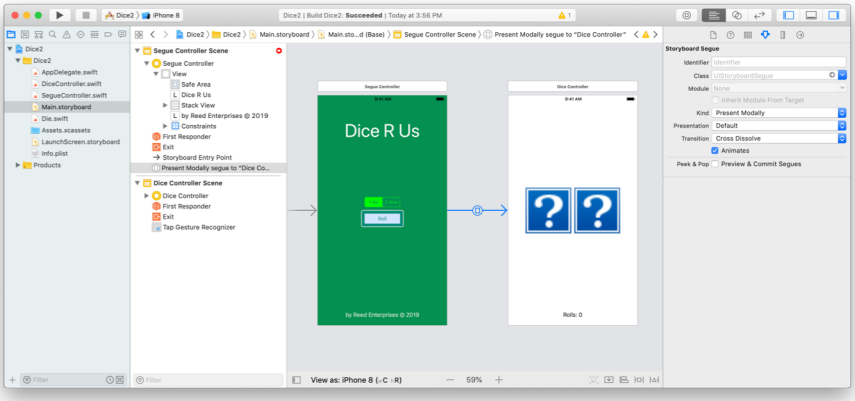


11

Passing data between screens

suppose we wanted to be able to select settings in the welcome screen

- e.g., select whether rolls one or two dice
- could have a UIControl to specify the desired number of dice, then must somehow pass that data to the other screen as part of the segue

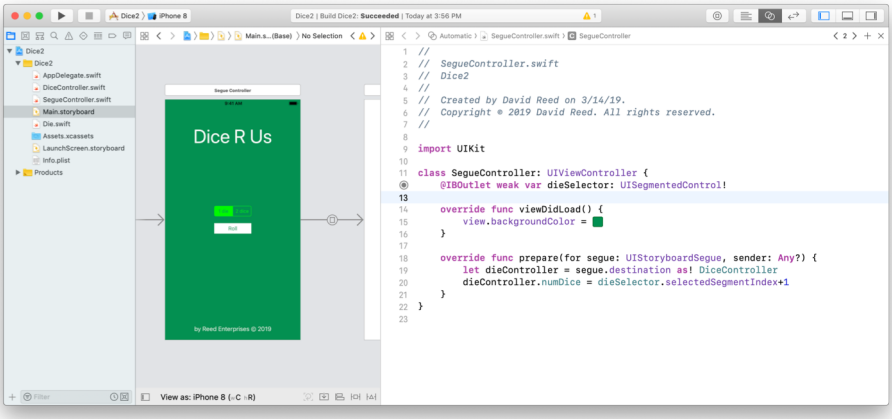


12

SegueController

need a UIViewController for the welcome screen

- it has an outlet for the segmented control
- plus a method that prepares for the segue by accessing the destination controller and passing the number of dice to its `numDice` field

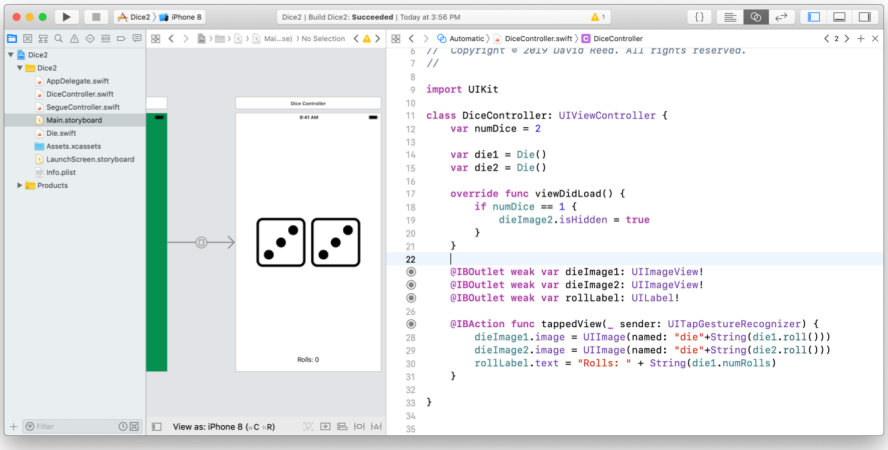


13

DiceController

since now more than one UIViewController, rename the main one

- have a field that stores the number of dice (default is 2)
- recall, viewDidLoad (if defined) is automatically called after the view loads
  here, check to see if this field was changed via the segue, if so hide the 2nd die

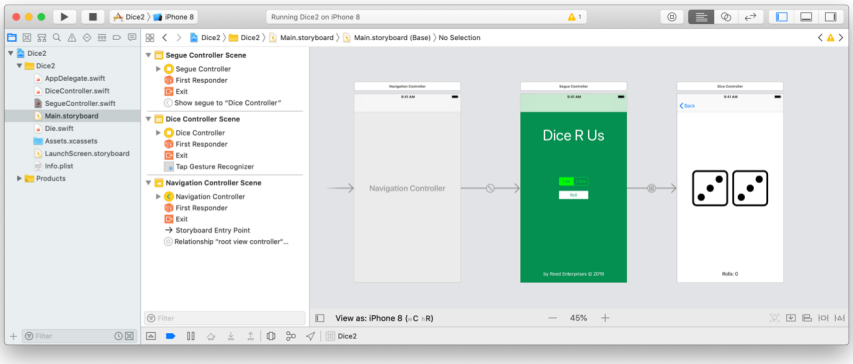


14

Navigation Controller

would like to be able to go back a screen

- e.g., after clicking one die, go back top the settings screen and switch to two dice
- select the controller of the entry-point screen (here, SegueController)
  go to Editor → Embed In → Navigation Controller
  this adds a navigation controller to the storyboard
- note: need to make sure that the segue is a Show (e.g., Push)



15

Popovers

sometimes, a separate screen is overkill

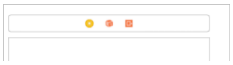
- a popover window is a less weighty alternative (e.g., for showing help info)

a popover is a View embedded in an existing View Controller

- in particular, it does not have its own UIViewController class

to open a popover (at the click of a button)

1. add a popover View to the storyboard
   - ✓ drag a View from the Object Library onto the bar at the top of the ViewController



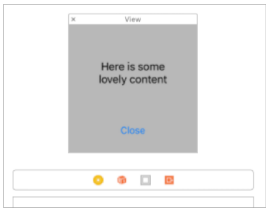
   - ✓ this will add a View icon to the bar and also display the View in the storyboard

16

Popovers (cont.)

2. specify the background & size for the popover
   - ✓ select the View, go to the Attributes Inspector and set background color
   - ✓ select the View, go to the Size Inspector and set X & Y

3. add content to the popover View
   - ✓ e.g., a label for displaying text and a button for closing



4. connect the popover View to the existing ViewController
   - ✓ i.e., control-drag from the View to create an outlet in the ViewController code (be careful to connect the UIView, not the contained elements)

```
@IBOutlet var popoverView: UIView!
```

17

Popovers (cont.)

5. create an Action for the button in the ViewController (to open the popover)
   - ✓ add Swift statements that open & position the popover

```
@IBAction func openPopover(_ sender: UIButton) {
    self.view.addSubview(popoverView)
    popoverView.center = self.view.center
}
```

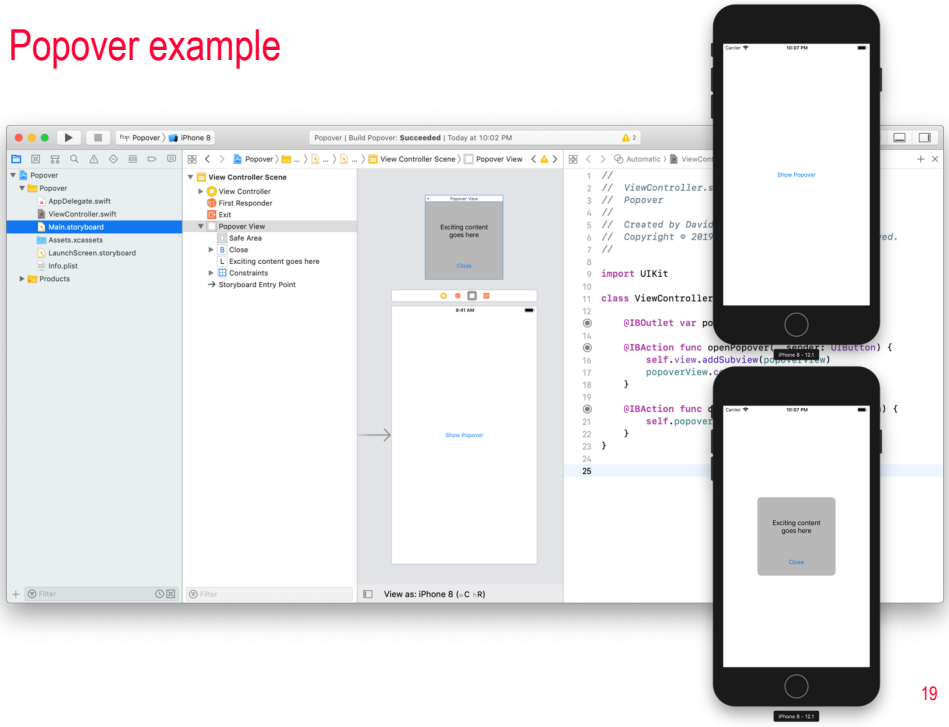
6. create an Action for the button in the popover View (to close the popover)
   - ✓ add a Swift statement to close the popover

```
@IBAction func closePopover(_ sender: UIButton) {
    self.popoverView.removeFromSuperview()
}
```

7. if desired, you can round the edges of the popover
   - ✓ select the View and go to the Identify Inspector
   - ✓ create a new User Defined Runtime Attribute (by clicking on +)
   - ✓ Key Path = `layer.cornerRadius`, Type = `Number`, Value = `10`

18

Popover example

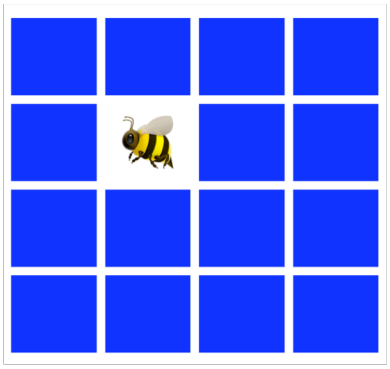


19

Complete example

suppose we wanted to create a Concentration game app

- Concentration is played with rows of cards, dealt face down
- a player picks two cards & flips them
  if they match, the cards are removed; otherwise, they are flipped back over
- the goal is to find all the matching pairs in the fewest number of flips

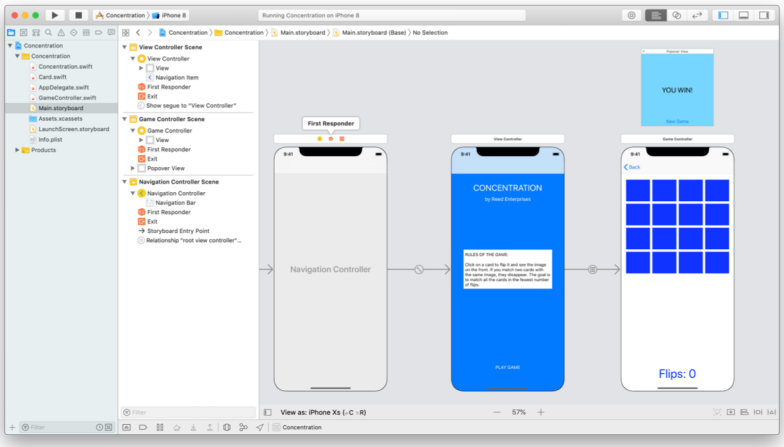


20

Concentration View

the project will contain

- a welcome screen with instructions
- a game screen with the grid of cards & a popover when the player wins
- a navigation controller to allow the user to go back to the welcome screen



21

Concentration model

2 structs:

- `Card` represents a card

```
struct Card {
    var isFaceUp = false
    var isMatched = false
    var identifier: String

    init(cardContent: String) {
        self.identifier = cardContent
    }
}
```

- `Concentration` contains the game logic

  new feature: filter

```
struct Concentration {
    private let theme = ["🐶", "🐝", "🐬", "🦕", "🦗", "🦀", "🐟", "🐘", "🕷", "🐹"]
    private(set) var cards = [Card]()
    private(set) var flipCount = 0

    var pairsRemaining: Int {
        let nonMatches = self.cards.indices.filter { !self.cards[$0].isMatched }
        return nonMatches.count/2
    }

    init(numberOfPairsOfCards: Int) {
        var emojiChoices = self.theme.shuffled()
        for i in 0..<numberOfPairsOfCards {
            let card = Card(cardContent: emojiChoices[i])
            self.cards += [card, card]
        }
        self.cards.shuffle()
    }

    mutating func chooseCard(at index: Int) {
        if !self.cards[index].isMatched {
            self.flipCount += 1

            let faceUpCardIndices = self.cards.indices.filter { self.cards[$0].isFaceUp && !self.cards[$0].isMatched }
            if faceUpCardIndices.count == 1 && faceUpCardIndices[0] != index {
                if self.cards[faceUpCardIndices[0]].identifier ==  self.cards[index].identifier {
                    self.cards[faceUpCardIndices[0]].isMatched = true
                    self.cards[index].isMatched = true

                }
                self.cards[index].isFaceUp = true
            }
            else {
                for i in self.cards.indices {
                    self.cards[i].isFaceUp = (i == index || self.cards[i].isMatched)
                }
            }
        }
    }
}
```

22

Concentration controller

uses several new features:

- OutletCollection
- lazy field
- alpha property
- animation

```
class GameController: UIViewController {
    @IBOutlet private weak var flipCountLabel: UILabel!
    @IBOutlet private var cardButtons: [UIButton]!
    @IBOutlet var popoverView: UIView!

    private lazy var game = Concentration(numberOfPairsOfCards: cardButtons.count/2)

    @IBAction private func restart(_ sender: UIButton) {
        popoverView.removeFromSuperview()
        game = Concentration(numberOfPairsOfCards: cardButtons.count/2)
        updateView()
    }

    @IBAction private func selectCard(_ sender: UIButton) {
        if let cardNumber = cardButtons.index(of: sender) {
            game.chooseCard(at: cardNumber)
            updateView()
        }
    }

    private func updateView() {
        flipCountLabel.text = "Flips: \(game.flipCount)"
        for index in cardButtons.indices {
            let button = cardButtons[index]
            if button.alpha != 0.0 {
                let card = game.cards[index]
                if card.isFaceUp {
                    button.setTitle(card.identifier, for: UIControl.State.normal)
                    button.backgroundColor =
                    if card.isMatched {
                        UIViewPropertyAnimator.runningPropertyAnimator(withDuration: 0.5, delay: 0.2,
                            animations: { button.alpha = 0.0 })
                    }
                }
                else {
                    button.setTitle("", for: UIControl.State.normal)
                    button.backgroundColor =
                }
            }
        }

        if game.pairsRemaining == 0 {
            for index in cardButtons.indices {
                cardButtons[index].alpha = 1.0
            }
            self.view.addSubview(popoverView)
            popoverView.alpha = 0.8
            popoverView.center = self.view.center
        }
    }
}
```

23