CSC 581: Mobile App Development

Spring 2019

App design & development

- MVC pattern
- development process
  View → Model → Controller
- HW5: tic-tac-toe

Model-View-Controller pattern

recall: designed apps utilize the *Model-View-Controller* software pattern:

- **Model** defines the logic underlying the app
  e.g., for a poker app, model cards, decks of cards, poker hands, …
- **View** defines the look of the app
  e.g., images of cards, button for dealing cards, …
- **Controller** defines actions of the app - it connects the View with the Model
  e.g., when the user clicks on a button to deal a card, it calls the appropriate method on a deck of cards object, then displays the resulting image

utilizing the MVC pattern leads to code that is

- easier to develop (especially when working in teams)
- easier to read & debug
- easier to reuse

App development process

Step 0: come up with an idea

- what problem are you trying to solve?
- who is your target audience?
- what makes it different from other apps?

Step 1: identify its features

- what functionality will it provide?
- what features are essential to its usability? → focus on these first
- what features would make it stand out? → design with these in mind

Step 2: create a storyboard (View)

- how many different views? what type of navigation?
- what are the visual elements? what are the controls?
- for common tasks, what is the user experience like?

- can use Interface Builder in Xcode to draw the storyboard

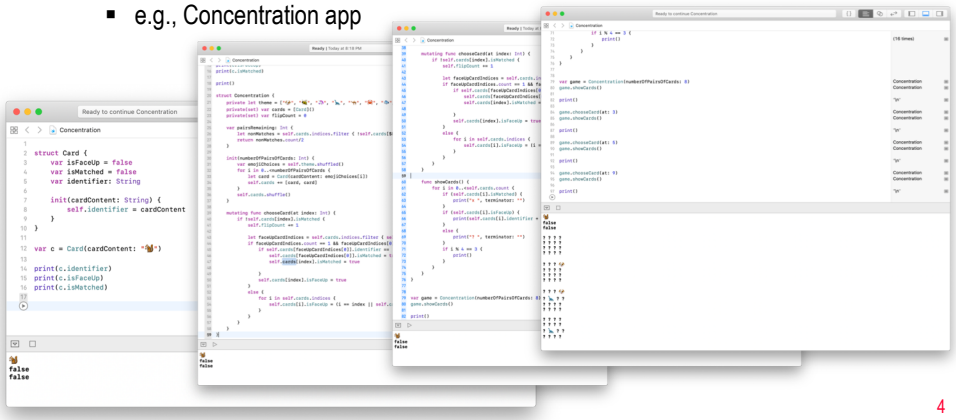
3

App development advice

Step 3: design & develop the underlying logic (Model)

- identify the objects that need to be modeled with structs/classes
  for each struct/class, identify the state that must be maintained + operations
- develop the Swift structs/classes in a playground
  use print statements to debug the code

- e.g., Concentration app



4

App development advice

Step 4: create the basic app (Controller)

- add the model classes to the project (can leave the print functions for now)
- create controller(s) to connect the view(s) with the model

- implement and test one feature at a time
- get the basic app working first (with the essential features)

Step 5: revise and add features as time allows

- as you test/debug your app…
  - ✓ you may find that the user experience needs to change – change the flow
  - ✓ you may find that some features are unnecessary – cut them
  - ✓ you may find that some missing features are needed – add them

- before revising or adding a feature, save the current app so you can restore

- when done revising, clean up (remove prints, make sure names make sense, …)

5

HW5: tic-tac-toe

for your final HW, you will complete the development of a tic-tac-toe app

STEP 0: a simple tic-tac-toe game, the user plays against the app

- i.e., user clicks on a board spot to place an X, the app selects a spot for O
- target audience: young children (or really bored adults)

STEP 1: features

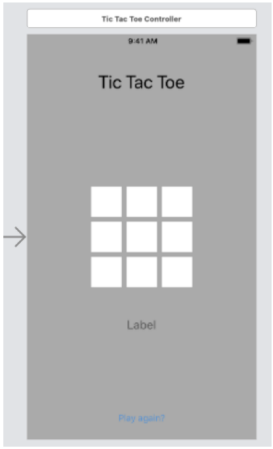
- displays the board (initially blank)
- user can click on a spot to make an X appear (O follows automatically)
- ideally, the app uses a smart strategy for O (but not too smart)
- winner (or draw) is displayed when game is over

6

HW5: tic-tac-toe (cont.)

STEP 2: create a storyboard (View)

- ideally, would have a welcome screen with author info & rules of the game
- for now, you are given a single view with the game board



- ✓ label for the app title
- ✓ a 3x3 grid of buttons for the board spots
- ✓ clicking on a button changes the button text to X
- ✓ label to display winner or draw (initially hidden)
- ✓ button to play new game (initially hidden)
- ✓ when the game is over, the label (with appropriate message) and button appear
- ✓ once a new game starts, they disappear

7

HW5: tic-tac-toe (cont.)

STEP 3: design & develop the underlying logic (Model)

- the basic design has been done for you: you must implement a struct named TicTacToeGame that has the following:

1. a private(set) field named `board` that represents the tic-tac-toe board
   e.g., print(game.board[0][1]) should display piece at row 0, col 1
2. a method named `place` that places a piece at a specific row/column
   e.g., game.place(piece: "X", inRow: 0, andColumn: 1)
3. a method named `place` that places a piece in a random empty spot
   for 80% credit, can pick any empty spot at random
   for full credit, should prioritize wins & non-losses
   e.g., game.place(piece: "O")
4. a method named `status` that returns the status of the game
   can return "X wins!", "O wins!", "It's a draw.", or "Play on…"
   e.g., print(game.status())

8

HW5: tic-tac-toe (cont.)

STEP 4: create the basic app (Controller)

- this has been done for you
- note: your Model must match the names used here *exactly*

STEP 5: revise & add features

- you are to add at least one significant feature
- e.g., a welcome screen with rules, a help popover, persistent stats (games won & lost), …

```
11  class TicTacToeController: UIViewController {
12
13      var game = TicTacToeGame()
14
15      @IBOutlet var boardSpots: [UIButton]!
16
17      @IBOutlet weak var resultLabel: UILabel!
18      @IBOutlet weak var againButton: UIButton!
19
20      @IBAction func boardClicked(_ sender: UIButton) {
21          if let index = boardSpots.index(of: sender) {
22              if game.place(piece: "X", inRow: index/3, andColumn: index%3) {
23                  if game.status() == "Play on..." {
24                      game.place(piece: "O")   ⚠ Result of call to 'place(piece:)' is unused
25                  }
26                  updateView()
27              }
28          }
29      }
30
31
32      @IBAction func playAgain(_ sender: UIButton) {
33          resultLabel.isHidden = true
34          againButton.isHidden = true
35          game = TicTacToeGame()
36          updateView()
37      }
38
39      func updateView() {
40          for i in 0..<boardSpots.count {
41              boardSpots[i].setTitle(game.board[i/3][i%3], for: .normal)
42          }
43
44          let result = game.status()
45          if result != "Play on..." {
46              resultLabel.text = result
47              resultLabel.isHidden = false
48              againButton.isHidden = false
49          }
50      }
51  }
```

9