Project 1 - 15 Puzzle

CS 458

1 Introduction

For this project, you will create an iOS *Fifteen Puzzle* app: `https://en.wikipedia.org/wiki/15_puzzle` Section 2 describes how to use Interface Builder (IB) editor to layout your view components and programmatically control the size and position of the tiles. The methods for the internal game logic (which you will need to implement) are specified in Section 3. The details of the Model View Controller (MVC) design and the flow of events is outlined in Section 5.1. Information about grading and some opportunities for bonus points appear in Section 6. How to submit your solution is specified in Section 7.

1.1 Creating the Project

This project will use the iOS "Single View App" template in Xcode. Use the product name `FifteenPuzzle`. You may have Xcode build a git repository for you, if you like. Xcode should also be able to connect to a remote repo on the ENCS gitlab instance.

2 Laying Out the User Interface

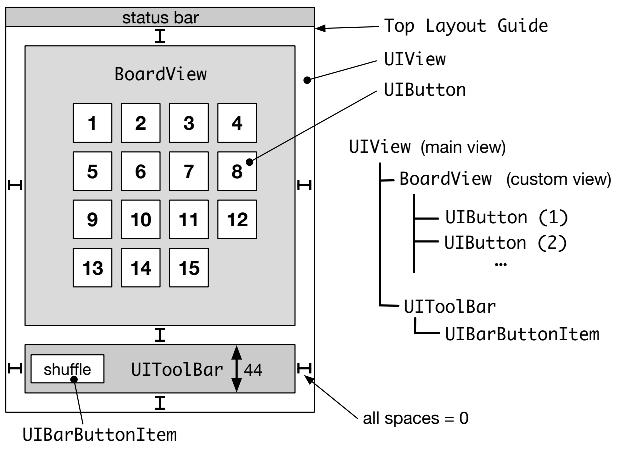

Figure 1: Layout and Hierarchy of Views

2.1 Adding Views and Controls

The interface for this project will consist of, at minimum, an additional `UIView` referred to here as `BoardView`, a `UIToolBar` with `UIBarButtonItem`, and a total of 15 `UIButtons`. See Figure 1 for the overall layout. Pin the leading, trailing, and bottom edges of the toolbar to the corresponding edges of the safe view, and fix its height to 44 pts. Pin all edges of `BoardView` so there is no space between it and the corresponding neighboring edges. (Use the safe view for top and sides, and the toolbar on the bottom.) Spaces represented by I-beams in the figure should all be zero.

Add the buttons to the `BoardView`, selecting colors so we can see them on the board view (e.g. make the buttons white and the board grey). Set the titles of the buttons to the values 1 through 15, and adjust their fonts to taste. Do not set any constraints on the buttons – we are going to set their size and position programmatically, as described in Section 4. Use the Attributes Inspect to set their view tags to 1 through 15 (matching their titles) so we can identify the buttons in our code.

2.2 Creating a Custom BoardView

In order to override UIView's layout mechanism, we create a custom subclass named `BoardView`. From the Xcode menu choose File → New → File and choose the iOS / Source / Cocoa Touch Class template in the presented dialog. Press Next and name the class `BoardView` and make it a subclass of `UIView` (do not create a XIB file; use Swift as the language). You should see the file `BoardView.swift` in the Navigator now. Using the Identity Inspector, change the class of the appropriate `UIView` object (labeled `BoardView` in Figure 1) to `BoardView`. Viola! This object is now an instance of your new class.

2.3 Adding a boardView outlet

Create an *outlet* so the controller can access the tiles of our board view:

```
@IBOutlet weak var boardView: BoardView!
```

You can do this with the Assistant Editor using the control-click-and-drag magic shown in class. Alternatively, you could just add the property above to `ViewController.swift` and use the Connections Inspector in IB to connect the `boardView` outlet with this property.

2.4 Adding action methods

We need to create appropriate callback methods in the `ViewController` class that will be invoked when the user selects a tile or wants a new game. You can either use the Assistant Editor to set these up, or create the methods and set them up with the Connection Inspector. Here are suitable prototypes:

```
@IBAction func tileSelected(_ sender: UIButton) { ... }
@IBAction func shuffleTiles(_ sender: AnyObject) { ... }
```

3 The Model

Define a class `FifteenBoard` that will encapsulate the puzzle's logic and support the methods below. (This will be a normal Swift file.) The puzzle tiles are encoded as integers from 1 to 15 and the space is represented with a zero. The puzzle tiles are assumed to lie in a 4×4 grid with one empty space, which I encode as a 2D array:

```
class FifteenBoard {
    var state : [[Int]] = [
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
        [13, 14, 15, 0]   // 0 => empty slot
    ]
...
}
```

`func scramble(numTimes n: Int)` Choose one of the "slidable" tiles at random and slide it into the empty space; repeat n times. We use this method to start a new game using a large value (e.g., 150) for n.

`func getTile(atRow r: Int, atColumn c: Int) -> Int` Fetch the tile at the given position (0 is used for the space).

`func getRowAndColumn(forTile tile: Int) -> (row: Int, column: Int)?` Find the position of the given tile (0 is used for the space) – returns tuple holding row and column.

`func isSolved() -> Bool` Determine if puzzle is in solved configuration.

`func canSlideTileUp(atRow r : Int, Column c : Int) -> Bool` Determine if the specified tile can be slid up into the empty space.

`func canSlideTileDown(atRow r : Int, Column c : Int) -> Bool`

`func canSlideTileLeft(atRow r : Int, Column c : Int) -> Bool`

`func canSlideTileRight(atRow r : Int, Column c : Int) -> Bool`

`func canSlideTile(atRow r : Int, Column c : Int) -> Bool`

`func slideTile(atRow r : Int, Column c: Int)` Slide the tile into the empty space, if possible.

It is important that the positions of the tiles are not simply chosen at random since you may create an unsolvable puzzle. Instead, "scramble" the tiles of a puzzle that we know is solvable when the user wants to start a new game.

3.1 Creating the Model Instance

We create a singleton instance of `FifteenBoard` class to hold the state of the puzzle. Since both the Board View and the View Controller will want access to this object, let the `AppDelegate` object own it:

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    var board : FifteenBoard?
    let numShuffles = 100 // used to scramble board
    ...
}
```

Create and initialize the the board when the application finishes launching:

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
                        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    self.board = FifteenBoard()
    self.board!.scramble(numTimes: numShuffles)
    return true
}
```

The app delegate exists for the entire lifetime of the app, so it is the only entity that needs to "own"the board object (i.e., uses strong reference). Placing the board object in the application delegate also makes it easy to save and load the state of the puzzle if you choose to add data persistence (as a bonus feature). We can obtain a reference to the "global" board object anywhere in our program as follows:

```
let appDelegate = UIApplication.shared.delegate as! AppDelegate
let board = appDelegate.board
```

4 Sizing and Positioning the Tile Buttons

```
override func layoutSubviews() {
    super.layoutSubviews() // let autolayout engine finish first

    let appDelegate = UIApplication.shared.delegate! as! AppDelegate
    let board = appDelegate.board  // get model from app delegate

    let boardSquare = boardRect()  // determine region to hold tiles (see below)
    let tileSize = boardSquare.width / 4.0
    let tileBounds = CGRect(x: 0, y: 0, width: tileSize, height: tileSize)

    for r in 0 ..< 4 {      // manually set the bounds, and of each tile
        for c in 0 ..< 4 {
            let tile = board!.getTile(atRow: r, atColumn: c)
            if tile > 0 {
                let button = self.viewWithTag(tile)
                button!.bounds = tileBounds
                button!.center = CGPoint(x: boardSquare.origin.x + (CGFloat(c) + 0.5)*tileSize,
                                         y: boardSquare.origin.y + (CGFloat(r) + 0.5)*tileSize)

            }
        }
    }
}
```

Figure 2: `BoardView` method that overrides `UIViews` `layoutSubviews` method to position the tile buttons to reflect the state of the board model.

We want to programmatically control the size and positions of the tile buttons so we override the values computed by the AutoLayout engine. This is accomplished by overriding `UIView`'s `layoutSubviews` method as listed in Figure 2. Once the AutoLayout is finished, we immediately reset by assigning to desired values of the bounds and center properties of each button. [1]

```
func boardRect() -> CGRect { // get square for holding 4x4 tiles buttons
    let W = self.bounds.size.width
    let H = self.bounds.size.height
    let margin : CGFloat = 10
    let size = ((W <= H) ? W : H) - 2*margin
    let boardSize : CGFloat = CGFloat((Int(size) + 7)/8)*8.0 // next multiple of 8
    let leftMargin = (W - boardSize)/2
    let topMargin = (H - boardSize)/2
    return CGRect(x: leftMargin, y: topMargin, width: boardSize, height: boardSize)
}
```

Figure 3: Auxiliary method that determines the largest square that fits in the center of the board view. We make sure the board size of is a multiple of 8 so the tile centers lie on the integer grid.

We first determine a suitably sized square in the center of the `BoardView` region where the tiles will be placed using the helper method `boardRect` listed in Figure 3. We then consult the `board` object to determine

[1] Note that we set the `bounds` and `center` properties rather than the `frame` property for each button since setting the `frame` property is ignored if the `transformation` property is *not* the identity (see Apple's UIView documentation).

the row and column of each tile in the puzzle. From this we determine the appropriate position and location of each button.

5 User Interaction

5.1 Model-View-Controller

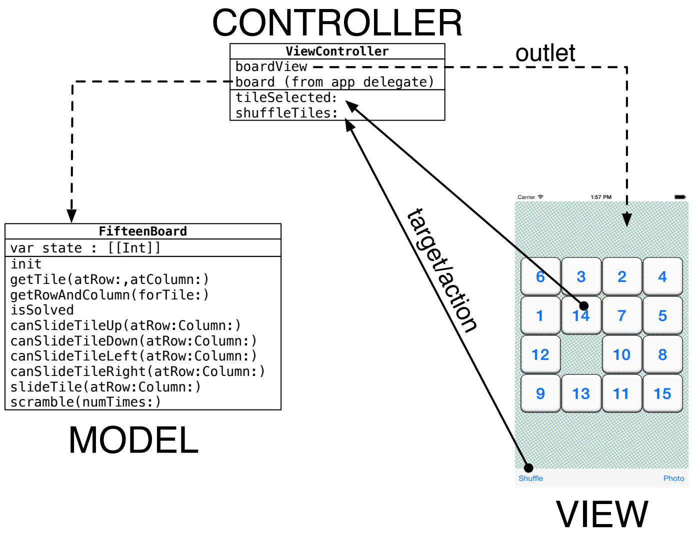

Figure 4: MVC elements of the 15 Puzzle App

Figure 4 illustrates the main components of the 15 Puzzle App classified according to the MVC pattern. The `UIButtons` and `UIViews` that are archived and reanimated at run time from the storyboard file are shown on the right. The controller references the `boardView` via an IB outlet so that it can manipulate the tiles on the board. The target/action mechanism is used to inform the controller when the user initiates an event. The controller accesses the sole model object stored in the app delegate which encodes the state of the puzzle and provides methods for querying and modifying the board. The model is not tied to any specific user interface and could be used in another application with a different user interface.

5.2 Sliding Tiles

Figure 5 shows the sequence of events that are triggered on a "touch-up" event when the user touches tile 14. We wired this event to send a `tileSelected:` message to the Controller (1). Whenever the user touches a tile we consult the board object to determine if there is a neighboring empty space for the tile to slide into (2). If so, we update the board state (3) and move the button accordingly (4). This is accomplished with the action method listed in Figure 6. We simply move the button by changing changing its center property.

Rather than move the tile directly to its new location, we can animate the change by replacing the line that moves the button's center with

```
UIView.animateWithDuration(0.5, animations: {sender.center = buttonCenter})
```

This uses a Swift closure for the `animations:` argument which sets the center property of the view. (All of the UIKit views are backed by a Core Animation layer and it is simple to animate many of their properties.)

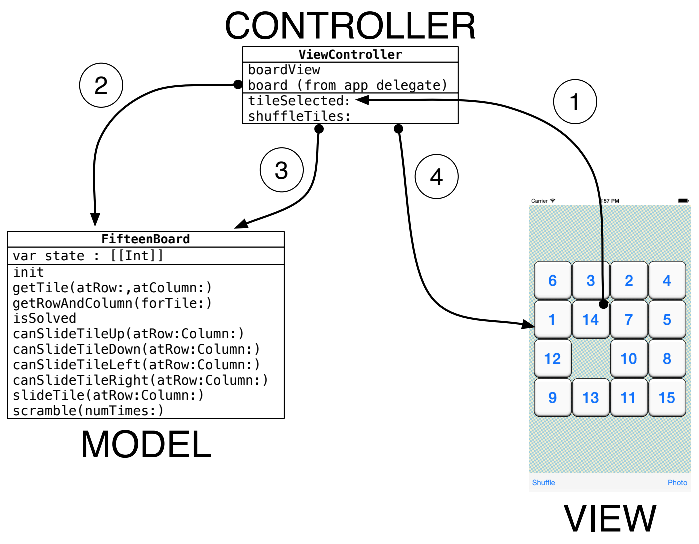

Figure 5: Communication sequence triggered when the user “touches up” on the 14-tile

5.3 Shuffling Tiles

When the user chooses to shuffle the tiles to start over, we simply tell the model to scramble the tiles and inform the boardView that it needs to layout the buttons again using the action method listed in Figure 7. Note that we do not invoke `boardView`’s `layoutSubviews` method directly – we simply mark the `boardView` as needing a layout which will eventually trigger the `layoutSubviews` message to be sent to the `boardView`.

6 Grading and Bonus Points

This assignment is worth a total of 50 points, broken up as follows:

10 pts	General Style / Compilation
05 pts	Interface is properly laid out
10 pts	Model is present and correctly structured / implemented
10 pts	Model, View, and Controller are properly separated
05 pts	Shuffle is handled appropriately
05 pts	Tiles which can be slid slide correctly
05 pts	Tiles which cannot be slid remain stationary

You can earn some bonus points for going above and beyond the requirements of the assignment, to a maximum of 5 bonus points. Here are a few examples of things you might do:

- Provide (reasonably nice) app icons, and customize the launch screen to be more appealing (1pt)
- Save/restore the state of the puzzle to local storage (2pts)
- Use images on the buttons (in addition to their title) so the puzzle shuffles / fixes the image. (2 pts)

7 What to Submit

Zip up your project’s directory, as usual. If you would like your project to be considered for extra credit, include a README file in the zip which indicates what else program does, and how to use the functionality.

```
@IBAction func tileSelected(_ sender: UIButton) {
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let board = appDelegate.board

    let pos = board!.getRowAndColumn(forTile: sender.tag)
    let buttonBounds = sender.bounds
    var buttonCenter = sender.center
    var slide = true
    if board!.canSlideTileUp(atRow: pos!.row, Column: pos!.column) {
        buttonCenter.y -= buttonBounds.size.height
    } else if board!.canSlideTileDown(atRow: pos!.row, Column: pos!.column) {
        buttonCenter.y += buttonBounds.size.height
    } else if board!.canSlideTileLeft(atRow: pos!.row, Column: pos!.column) {
        buttonCenter.x -= buttonBounds.size.width
    } else if board!.canSlideTileRight(atRow: pos!.row, Column: pos!.column) {
        buttonCenter.x += buttonBounds.size.width
    } else {
        slide = false
    }

    if slide {
        board!.slideTile(atRow: pos!.row, Column: pos!.column)
        sender.center = buttonCenter // or animate the change
        if (board!.isSolved()) {
            // celebrate victory
        }
    }
}
```

Figure 6: Action method that can trigger a tile to slide into a neighboring hole.

```
@IBAction func shuffleTiles(_ sender: AnyObject) {
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let board = appDelegate.board!
    board.scramble(numTimes: appDelegate.numShuffles)
    self.boardView.setNeedsLayout()
}
```

Figure 7: Shuffling tiles to generate a new puzzle.