COMP47390: CocoaTouch

Programming Test (HelloPoly) – February 27, 2019 (9:00 – 10:50)

Student Name:	Student Number:

Table of Contents

In this programming test you will build an iOS application for displaying a polygon. In Part 1, the user interface will allow the user to set the number of sides for a polygon and display the name of the polygon. In Part 2, the label will be replaced by a custom view that draws the shape of the polygon. You will make use of some key design patterns used in Cocoa Touch applications.

Important notes

- Keep a bullet style log of your coding effort using Pages in a file called `readme-part<#>`. Insert screenshots when appropriate using the Grab application.
- Ensure your code is adequately documented.
- Commit changes regularly (at least every 15 mins).
- For simplicity, use the iOS Simulator. When building your App, you need to tell Xcode which platform to build against. There is a handy toolbar popup for selecting the active scheme. Set the scheme to the iPhoneSE to test your App.

Part 1

Create and connect a Model, View, and Controller to create the App shown in the screenshots of Figure 1. Here is a rough outline of the tasks required to complete this part successfully:

1. Create a new single view application for iOS named `HelloPoly`.
2. Layout the user interface in your storyboard and add constraints so that it is responsive to various device orientations and sizes.
3. 'Wire up' your UI elements to the view controller as shown in class.
4. Create a new CocoaTouch Swift class, called `PolygonShape`, and implement an appropriate model for your MVC using the following public API (ensure the number of vertices for your polygon never drops below 3 nor exceeds 12):
   - public `Int` property called `numberOfSides`
   - public `String` property called `name`
5. Add your polygon model to your view controller using lazy instantiation.
6. Implement action methods that are triggered by user interactions.
7. Update the text label in response to user increasing or decreasing the number of sides of the polygon.
8. Enable or disable user interface elements appropriately (i.e. when the number of sides of the polygon reaches the minimum or maximum values allowed by the model).
9. Add swipe gesture recognisers to increase/decrease the number of polygon sides.

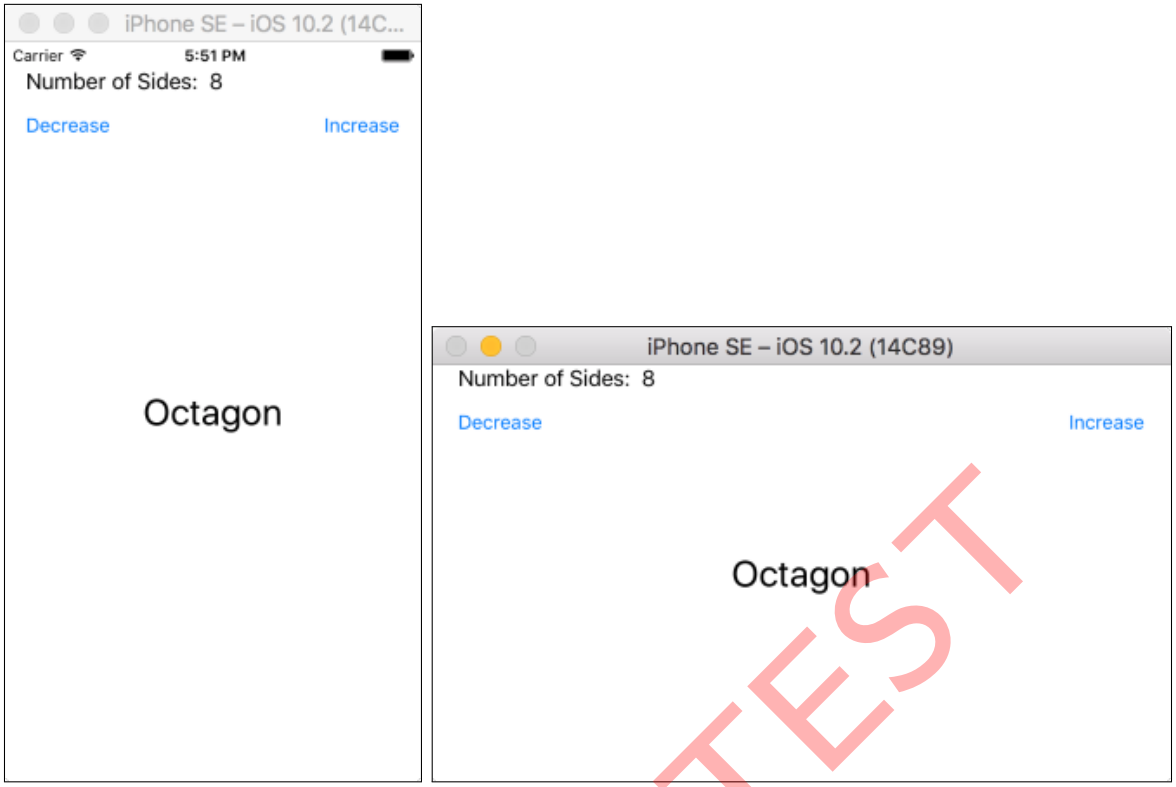

Figure 1: HelloPoly App – Screenshots of Part 1

Build & Test

Before you write the guts of the `@IBAction` methods, you may want to put in a simple logging statement and test your App. If you have hooked up all the target/actions correctly, clicking the buttons should display the log message in the console. Recall that outputting simple log statement in Swift can be achieved with `print("I'm in the increase/decrease method")` in your code. To view the console in Xcode, select the middle icon in the top right of Xcode (option also available in `View > Debug Area > Show Debug Area`).

Remember your App should always build and run without warning. You must also ensure it is responsive (test for various screen sizes and all device orientations).

Submission

Once completed with this part, commit your code with the name `solution-part1` and create a zip archive of your project including your log file and called:
`<Firstname>_<Surname>_<StudentNumber>_part1.zip`

Ensure your code compiles with no errors and no warnings (penalties for these will be large).

Part 2

The previous part gave us the basic structure of a single MVC application. Your model was coded in the `PolygonShape` class, your view had some basic controls to let the user set the number of sides of the polygon and your controller mediated between the model and the view keeping everything in sync. You will now delve deeper into the view side of your app design, replacing the text label by a custom view to display the polygon in all its glory as shown in the screenshots of Figure 2. You will use the delegation pattern to communicate between the view and the model in the view controller. Finally you will use the user defaults system for saving and restoring our application state.

Figure 2: HelloPoly App – Screenshots of Part 2

Most of the work is in creating a custom view and doing the drawing of the polygon. Since this is not a geometry class, you will be provided with some code to calculate the vertices of a polygon given a rectangle (see closure in the hint section below). A general outline of what you'll need to do is as follows:

1. Create a `UIView` Swift subclass called `PolygonView` that will display your `PolygonShape` object.
2. Add a new protocol called `PolygonProtocol` to the start of your Swift model file with a delegation method called `func pointsInRect(rect: CGRect) -> [Array]`.
3. Make your `PolygonShape` model conform this new protocol and implement the delegation method using the hint below.
4. Add a delegate property of type `PolygonProtocol` to your custom view.
5. Implement the `func drawRect(rect: CGRect)` in your custom view so it draws the polygon using the delegate property to get the vertex coordinates
6. Edit your storyboard and replace the text label with a custom `UIView` of type `PolygonView`. Wire it up to an instance property in your controller, called `polygonView`, and update your controller code appropriately. Update constraints so that your UI is responsive, supporting all device sizes and orientations.
7. In your view controller, edit `func viewDidLoad()` and ensure your `PolygonView` delegate is initialised with your `PolygonShape` model.
8. Edit your view controller and add persistence to your App saving the number of sides of your model in the user defaults, and restoring its value on App restart.

Hints

Here is a closure to help you with the calculation of the coordinates of the polygon vertices:

```swift
var polygonPoints = { (rect: CGRect, numberOfSides: Int) -> [CGPoint] in
    let center = rect.center
    let radius = min(rect.size.width, rect.size.height) / 2.0
    let arc = 2 * CGFloat.pi / CGFloat(numberOfSides)

    var vertexArray = [CGPoint]()
    for i in 0..<numberOfSides {
        var vertex = center
        vertex.x += cos(arc * CGFloat(i) - 2 * CGFloat.pi) * radius
        vertex.y += sin(arc * CGFloat(i) - 2 * CGFloat.pi) * radius
        vertexArray.append(vertex)
    }
    return vertexArray
}
```

In this closure you will be relying on a `CGRect` extension providing the read-only property `centre` to the structure (to be implemented in your solution).

Submission

Once completed with this part, commit your code with the name `solution-part2` and create a zip archive of your project including your log file and called:
`<Firstname>_<Surname>_<StudentNumber>_part2.zip`

Ensure your code compiles with no errors and no warnings (penalties for these will be large).

— END OF PROGRAMMING TEST —