Due Date
11:59pm Thursday March 8th, 2012

Assignment

The last assignment gave us the basic structure of an MVC application. The model was the PolygonShape class, the view had some basic controls to let the user set the number of sides of the polygon and the controller mediated between the model and the view keeping everything in sync.

In the next lab we'll delve deeper into the view side of the design by adding a custom view to display the polygon in all its glory. We'll also use the user defaults system for saving some application state.

The biggest part of this assignment will be creating a custom view and doing the drawing of the polygon. Since this isn't a geometry class, you will be given code to calculate the points of a polygon given a rectangle (see the hints section below). A general outline of what you'll need to do would be as follows:

1. Create a custom UIView subclass that will display your PolygonShape object

2. Give your view class access to the PolygonShape object so that it can retrieve the details of the polygon as needed

3. Implement the code to draw the polygon in the custom view

4. Respond to the user input changing the number of sides and update the view accordingly

5. Add a subview to your custom view to display the name of the polygon

6. Save the number of sides in the user defaults, and restore that value when the user launches the application

Testing

In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the behavior of the application, and also on the code.

We will be looking at the following:

1. Your project should build without errors or warnings and run without crashing

2. All of the features from last week's assignment should continue to function correctly

3. After launching, your custom view should display a polygon with the correct number of sides (matching what's in the text label showing the number of sides, use the viewDidLoad method in the ViewController.m to initialize the number of sides in the view)

4. Tapping the increase and decrease buttons should cause the custom view to update as expected (both the polygon and the name should update)

5. The background of the custom view should be filled with a solid color

6. The border of the custom view should be stroked with a solid color (e.g. black)

7. Add a label as a subview of your custom view to display the name (e.g. Square, Pentagon) of the polygon

8. When the user exits and relaunches the application the number of sides should be preserved. That is, if the user changes the sides to 7 then exits/relaunches the app, the number of sides should still be set to 7.

Extra Credit

• Add a slider as another way of inputting the number of sides
• Add a segmented control allowing the user to select from solid or dashed lines for the sides of the polygon
• Allow the user to interact with the custom view to rotate the polygon

• Fill the background of the custom view or the interior of the polygon with a gradient

Hints

Here is a block of code to help with calculating the points of a polygon given a rectangle and a number of sides.

```
+ (NSArray *)pointsForPolygonInRect:(CGRect)rect numberOfSides:(int)numberOfSides {
    CGPoint center = CGPointMake(rect.size.width / 2.0, rect.size.height / 2.0);
    float radius = 0.9 * center.x;
    NSMutableArray *result = [NSMutableArray array];
    float angle = (2.0 * M_PI) / numberOfSides;
    float exteriorAngle = M_PI - angle;
    float rotationDelta = angle - (0.5 * exteriorAngle);

    for (int currentAngle = 0; currentAngle < numberOfSides; currentAngle++) {
        float newAngle = (angle * currentAngle) - rotationDelta;
        float curX = cos(newAngle) * radius;
        float curY = sin(newAngle) * radius;
        [result addObject:[NSValue valueWithCGPoint:CGPointMake(center.x + curX,
        center.y + curY)]];
    }
    return result;

}
```

It will return an array of NSValue objects that wrap the CGPoints (because an NSArray has to have objects as its contents). You can retrieve the CGPoint contents by sending the CGPointValue message to the NSValue object. For example, if you have an NSValue object you can do:

```
NSValue *theValue = .... // retrieve an object from the array
CGPoint thePoint = [theValue CGPointValue];
```