Mobile Application Programming: iOS

CS4952 Spring 2016
Project 2
Due: 11:59PM Monday, February 29th

Abstract

Create a painting application that allows the user to paint a picture using their finger, tracking the locations the user's finger touches in a paint area. It also allows the user to keep a hierarchal organization of these paintings with 2 levels: Collection and Painting. The Painting level is simply the UI provided to make paintings. The Collection level keeps related paintings together and exhibits them in a grid. The paintings are stored in a vector format that is independent of the screen size used to create them, but aware of the aspect ratio at it was created in. The application should be written in Swift with no Interface Builder files.

The application must take full advantage of rotation, allowing the user to rotate the device without losing their painting. When the paint area changes shape (e.g. by going from portrait to landscape), the picture contained should be redrawn to fill the new view shape.

See the Components section for a description of the required program parts, and the Considerations section for suggested methods to accomplish this.

Components

- **Painting View:** Collects touch points in an array using the touch-related methods in UIResponder, building up a list of points that compose a poly-line. A poly-line is a set of points with straight lines connecting them. When the user puts their finger down, create an array to collect points. When they move their finger, add to collected points to the array. When they lift their finger, package the points to form a complete poly-line. The poly-line should also have properties for the brush used to draw the polyline, specifically stroke width, end caps, join, and brush color. The view should maintain an array of poly-lines in addition to the current line being built. When asked to redraw itself, the view should draw the array of poly-lines as well as the current poly-line. By calling "setNeedsDisplay" in the touch methods, the current line will be drawn as the user drags their finger. The view should expose properties for the brush type that can be set outside the class. These properties will apply to any poly-lines created from that point forward. A button to delete the painting should be provided. The easiest way is using the navigationItem property of a view controller object containing this view as its content. Another button should also be made available that opens the brush chooser you built for project 1. Use this view to change the brush properties of the painting view.
- **Collection View:** Shows the paintings the user has created in a grid format. Each painting shown is a cell whose content is a preview of the painting. Tapping the painting should show the painting view populated with the polylines already created for that painting. Using a UICollectionView and UICollectionViewFlowLayout is suggested. A button allowing the user to add a painting to the collection should be provided. Again, the easiest way is using the navigationItem property of a view controller containing this view as its content.
- **Brush Chooser:** The view you created for project 1 that allows the user to select brush characteristics. It is opened from the painting view, temporarily replacing it while the user selects the brush properties. Add a control to allow the user to indicate that they have finished

choosing their brush that, when selected, closes the view, restoring the painting view, and transferring the brush properties to the painting view.

Considerations

- **Data Model:** The paintings created by the application need to be shown in a collection view, and recalled for further editing if the user selects them again. The views used to create the UI of the application should rely on a backend object that organizes the paintings and their contents. A suggested format for this backend is:
  - A PaintingCollection object that exposes an interface giving the number of paintings and a way to add and remove paintings from the collection.
  - A Painting object that contains a private array of Stroke objects, methods to determine the number of strokes and add / remove strokes from the painting, and has other needed properties like the painting aspect ratio.
  - Stroke object that contains an array of points that make up the polyline used to draw the stroke and the brush settings used to draw it. The coordinate system for points and stroke width in this object should be resolution independent. See the description below for rotations for one way to do this.
- **Screen Independence / Rotation Support:** This is best accomplished by using a painting view, an adapter object (a view controller), and a data model. The adapter object receives completed poly-line and brush information from the paint view every time the user lifts one of their fingers, by way of a delegate relationship. This information is converted by the adapter from the paint view coordinate system to a view-independent coordinate system used by the model object, then added to the model object. Such a coordinate system is a unit floating-point coordinate system (points take on values between 0 and 1 in x and y). When the paint view needs this information back (e.g. when the view has been resized due to rotation), the controller retrieves the information from the model object, converts the data back to the paint view's coordinate system, and gives it to the paint view. The painting has an aspect ratio property that allows this to be converted back and forth from the unit coordinate system without stretching. In this way, the adapter has access to both the painting view and the data model, and has conversion methods to convert data from view to model and back from model to view.
- **Changing Screens:** Unlike projects 0 and 1, this project requires several screens of UI to accomplish the full task of a painting collection. Each screen should be organized into a UIView subclass that acts as a container, exposing its subviews as read-only properties of the view. That view should be assisted by a controlling object that takes care of coordinating the changing of screens. This is typically a set of UIViewController subclasses and a unifying UINavigationController instance. The controller also fills an adapter roll, converting data from the format the model uses and the format the views use. In this way, the views and model never directly share information, but instead rely on the controller object to accomplish the transmission of data between them. This allows the controller to account for differences in the view's coordinate system and the model's data storage.

Extra Credit

- **Undo / Redo (10%):** Allow the user to remove polylines from the painting using with an Undo button. Lines removed should be able to be restored using a Redo button. When a new line is created by the user, any items in the redo stack should be removed (no branching undo/redo tree is needed).

Handin

You should hand in your zipped project, including any supporting files, using the CADE Lab handin system on the website, or on the command line:
handin cs4962 project2 your_project_zip_file.zip