Mobile Application Programming: iOS

CS4530 Spring 2017
Project 2
Due: 11:59PM Monday, February 20th

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. A painting is stored in a vector format that is independent of the size of the view used to create it, but is aware of the aspect ratio of the view at it was created in. The application also allows the user to keep a collection of such paintings. A painting can be created, closed, and subsequently opened again and modified. The application should be written in Swift and have an Xcode project 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 fit in the new view shape, but keep its previous aspect ratio. This means that a painting drawn in portrait mode would be shown in only a portion of the screen when viewed in landscape mode. The reverse is also true.

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 the collected points to the array. When they lift their finger, take the accumulated points and save them as a completed poly-line. While drawing, the view has a set of brush settings, including line width, end caps, join setting, and color. When a poly-line is completed, ensure that the brush settings used to draw it are included with the poly-line. The view should maintain an array of poly-lines in addition to the current poly-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. This approach is somewhat wasteful, as every poly-line will be redrawn every time a point is added to the current poly-line. In practice, though, this does not turn to be a performance issue. The view should expose properties for the brush type that can be set programmatically from outside the class. These properties will apply to any poly-lines created from that point forward. A button to delete the current painting should be provided. When pressed the user is returned to the painting collection. Also provide a back button to allow the user to exit the painting without deleting it. By placing this view in a view controller, then adding that view controller to a UINavigation controller, you can use the navigationItem property of the view controller object to create these buttons.
- **Brush Chooser**: Another button similar to the delete and back buttons should be accessible from the painting interface that opens the brush chooser you built for project 1. Pressing the button should open a view controller who's view is configured to contain your brush chooser. When the user activates the user interface elements there, the selections should be passed to the painting view. Include a button that allows the user to dismiss the brush chooser.

- **Collection View:** Shows the paintings the user has created in a grid-like format. This view is the view the user should initially see when entering the application, though it will be empty, as no paintings have been created yet. 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. Tapping the add button should open the painting view with a blank painting. After creating a painting, the user can return to this view and see a preview of the painting(s) they created. Each painting is shown in a cell whose content is a preview of the painting it represents. Using a UICollectionView and UICollectionViewFlowLayout is suggested. Configure items that present the painting preview. Because paintings can be created in portrait or landscape, not all the cells will be the same shape. The cell should show the painting they represent in a way that preserves the aspect ratio of the painting view used to create them. The painting previews can be made by creating instances of the painting view that are configured in a read-only mode and contain the poly-lines in the painting they represent. Tapping a painting preview cell should open the painting view populated with the poly-lines already created for that painting.
- **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 data model object that organizes the paintings and their contents. A suggested format for this object 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. It should also have other needed properties, like the painting aspect ratio.
  - A Stroke object that contains an array of points that make up the poly-line used to draw the stroke as well as 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.

Considerations

- **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 poly-lines 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 command line using the command:

handin cs4530 project2 your_project_zip_file.zip