Abstract

Create a simple finger painting application that includes a dynamic paint-mixing painter’s palette. The top the of the screen is a paint area where the user can draw pictures using their finger, using the selected color of paint from the palette. The bottom of the screen shows a painter’s palette with paints. One paint is the selected paint, selected by the user by touching a finger on the screen over it, and is indicated with an outline, glow, or other highlighting mechanism. A paint may be combined with another paint to form a new paint by dragging it on top of another paint. This causes a new paint to be dynamically added to the palette which is a combination of the two paints. This new paint then should then be selected automatically so it is the selected paint color.

Make sure to test your application on both a phone-type simulator (e.g. Nexus 4) and a tablet-type simulator (e.g. Nexus 7) in both portrait and landscape orientations to ensure that the UI is both functional and reasonable-looking on both device types and both orientations.

Components

- **Paint Area View**: A subclass of View that overrides onDraw, and allows the user to paint using their finger. This is accomplished by overriding the onTouch method of View and using the data given to form poly-lines. The view must be resizable, even after paint has been applied to the view. To accomplish this, store the poly-lines that the user draws, including the color used to draw them. Then, whenever the view needs to be redrawn, use the stored locations and colors to draw all of the poly-lines. Storing the points in a unit coordinate space (0.0 -> 1.0 in both x and y) allows a similar picture to be drawn whatever the view’s size.

- **Palette View**: A subclass of ViewGroup that overrides onDraw to draw a painter’s palette board, and maintains a collection of paints. The class should offer addColor and removeColor methods that cause paint views to be added to and removed from the palette, updating the layout. Override onLayout to layout the paint views around the palette. The palette view must be either **circular or shaped like a traditional painter’s palette**, and the paints should be laid out along the edge of the palette in a circle or following a painter’s palette shape. Do **not** lay the paints out in a grid. When the user touches a paint view, select it (see below). When the user then drags their finger, continually set the x and y properties to move the paint under the user’s finger, creating a drag action. When the user releases the paint, if it is over another paint view, mix the two paints and add the result to the palette (see below). If it is not over another paint, simply return the paint to its original position by setting the x and y properties equal to the left and top properties, respectively (see View documentation on x, y, left, top). If the paint view is dragged off the edge of the palette, remove the paint view, from the set.

- **Paint View**: A subclass of View that depicts a blob of paint of a particular color. Override onDraw to draw the paint blob to fill the view, respecting the padding of the view. Allow the paint to be highlighted using an outline or a glow effect by offering a boolean property “active” that, when changed, causes the view to redraw itself, either adding or removing the highlight effect.
• **Color Mixing:** When one paint view is dragged over another and released, the two paints should be mixed to get a 3rd color. Then that color should be added to the palette. You may use RGB averaging or true subtractive pigment mixing to determine the mixed color. It is actually somewhat tricky to get a properly mixed color, so do whatever you like to get the mixed color, as long as it produces something somewhat resembling a combination of the two input colors. The easiest is to isolate the red, green, and blue channels of each color by bitwise and-ing with a mask for each channel, average each channel individually, then combine the averaged channels into a single integer by bitwise or-ing them together.

**Extra Credit**

• **Palette (10%):** Provide a painter’s palette that is non-circular, instead using an oval-like shape with a smooth notch taken out to accommodate the user’s hand (see Google image search for “painter’s palette”). Make sure your paints line up along the palette’s edge, taking the notch into account. This will likely require using some kind of custom circle with a specialized radius generation function to get the right shape, similar to our paint splotch example in class. Use that function to generate the locations for the paint splotches.

• **Differentiated Paint Blots (5%):** Draw your paint views in such a way that their shape is generated randomly, making each paint blot have a different shape. This should **not** be just 10 different custom shapes you cycle through. It should be a truly procedurally generated shape for each new paint. The blots should have somewhat smooth borders, like real paint blots would.

• **Aesthetic (5%):** While all submissions should look visually pleasing, anyone going the extra mile to make the application look very good will get extra credit up to 5%. Unfortunately this is an inherently subjective requirement, so just do your best to achieve it. Try out wood-grain, nice background gradients or patterns, shadows under the palette, depth added to the paints using shadows or glossy highlights, etc.

**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 project1 your_project_zip_file.zip`