Assignment V: Emoji Art

Objective

The goal of this assignment is to learn about how to deal with multi-touch gestures.

Be sure to review the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Due

This assignment is due before lecture 13.

Materials

- Start your work from the version of EmojiArt from Lecture 11.

Required Tasks

1. Do not break anything that is working in Emoji Art as of Lecture 11 as part of your solution to this assignment.
2. Support the selection of one or more of the emojis which have been dragged into your Emoji Art document (i.e. you're selecting the emojis in the document, *not* the ones in the palette at the bottom). You can show which emojis are selected in any way you'd like. The selection does not need to be persistent (in other words, restarting your application does not have to preserve the selection).
3. Tapping on an unselected emoji should select it.
4. Tapping on a selected emoji should unselect it.
5. Single-tapping on the background of your Emoji Art (i.e. single-tapping anywhere except on an emoji) should deselect *all* emoji.
6. Dragging a selected emoji should move the *entire selection* to follow the user's finger.
7. If the user makes a dragging gesture when there is no selection, pan the entire document (i.e., as it does in lecture). There is an exception to this if you do the Extra Credit.
8. If the user makes a pinching gesture anywhere in the Emoji Art document and there is a selection, all of the emojis in the selection should be scaled by the amount of the pinch.
9. If there is *no selection* at the time of a pinch, the entire document should be scaled (again, like it does in lecture).
10. Make it possible to delete emojis from the Emoji Art document. This Required Task is intentionally not saying what user-interface actions should cause this. Be creative and try to find a way to delete the emojis that feels comfortable and intuitive.

Hints

1. Get selection working first (by adding a tap gesture that adds/removes an emoji from the selection), then start in on the resizing and moving gestures.
2. Your selection is *not part of your Model*. It is purely a way of letting the user temporarily express which emoji they want to resize or move. Thus you will want to maintain your selection as temporary state in your UI code somewhere.
3. A `Set` might be a good data structure to store the set of selected emoji since there's no "order" to the selection (like an `Array` would imply) and a single emoji cannot be selected twice (again, as an `Array` would allow, but a `Set` does not).
4. If you do choose to use a `Set` for your selection, you might want to store `Emoji`'s `id`'s in there because, by definition, an `Identifiable`'s `id` is `Hashable` and anything put in a `Set` must also be `Hashable`.
5. Whatever view modifier(s) you are using on your `Text` to show whether it is selected need to be applied *before* you apply the `.position()` view modifier (because you want your selection-showing modifier to get positioned too).
6. Check out the view modifier called `.border` to see if it's something you want to use. Totally not required, but we didn't get a chance to mention its existence yet in lecture, so we're mentioning it here.
7. You might want to hide whatever UI you are using to show which emojis are selected when a gesture is in flight. It's unnecessary clutter in that circumstance.
8. When it comes to pinching, all pinches (whether zooming the document or resizing the selected emojis) are always recognized on the entire document (even though what happens on such a pinch depends on whether there's a selection at the time). So you can likely piggy-back your selection resizing on the existing `MagnificationGesture`. But you'll need to be sure that if there is something in the selection, your existing zoom view modifier is not paying any attention to `gestureZoom`. Also, you'll be doing something different in `.onEnded` depending on whether there is a selection or not.
9. We are using `scaleEffect` and `offset` to zoom and pan on our entire document. You can use the same strategy to resize and move your selected emojis (but only while gestures are in motion—the rest of the time your View is just drawing what it sees in the Model, as normal).
10. A general rule of thumb for gestures is that you are modifying `@GestureState` in `.updating` and you are modifying your Model (or some `@State`) in your `.onEnded`. Then you are using both that `@GestureState` and that Model/`@State` as arguments to your view modifiers throughout your View as appropriate.
11. You will not be able to piggy-back your emoji-moving onto the whole-document panning drag gesture as you can with zoom/resize. A drag gesture that starts on a selected emoji does a very different thing than a drag gesture that does not. So a new

`DragGesture` (attached to the view that is drawing each emoji) will be required to support moving the selection around.

12. You are allowed to pass `nil` to the `.gesture` view modifier. You might want to do this as part of a ternary expression which chooses whether to add a gesture to a view or not. For example, `Text().gesture(includeGesture ? theGesture : nil)`.
13. As always, we try to make "conditionality" in our SwiftUI views be embedded into the *arguments* to view modifiers rather than by using `if-else` statements (whether inside or outside `@ViewBuilder` constructs). This often takes the form of a ternary operator (`? :`) but may involve calling a `function` or getting the value of a computed `var` as part of the conditional decision-making.
14. This entire assignment can be done in fewer than 50 lines of code (possibly fewer than 30 lines of code in the View itself). So if you find yourself writing a lot more than that, maybe rethink your approach.

Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. Gestures

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Project does not build without warnings.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the iOS API and knows how the Memorize game code from lectures 1 and 2 works, but should <u>not</u> assume that they already know your (or any) solution to the assignment.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least one of these is highly recommended to get the most out of this course.

If you choose to tackle an Extra Credit item, mark it in your code with comments so your grader can find it.

1. Allow dragging *unselected* emoji separately. In other words, if the user drags an emoji that is part of the selection, move the entire selection (as required above). But if the user drags an emoji that is not part of the selection, then move only that emoji (and do not add it to the selection). You will find that this is a much more comfortable interface for placing `Emoji`s. Doing this will very likely require you to have a more sophisticated `@GestureState` for your drag gesture.