# Assignment V: EmojiArt

## 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 in one week.

## Materials

- Download the version of EmojiArt from Lecture 10 and start working on this assignment from there.
- You will need to have watched L9 and L10 before doing this assignment.

## Required Tasks

1. Download the version of EmojiArt from Lecture 10. Do not break anything that is working there 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 EmojiArt 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 is not persistent (in other words, restarting your app will not 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 EmojiArt (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 EmojiArt does in L10).
8. If the user makes a pinching gesture anywhere in the EmojiArt 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.
10. Make it possible to delete emojis from the EmojiArt 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. Note that your EmojiArt now supports both single tap and double tap gestures on the background. You will likely want to use the `Gesture` function `.exclusively(before:)` to make sure double taps don't get missed because SwiftUI recognizes the single tap part of a double tap and doesn't give an opportunity for the second tap to be recognized as part of a double tap gesture.
2. Your selection is *not part of your Model*. It is purely a way of letting the user express which emoji they want to resize or move. Thus you will want to maintain your selection 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, adding a `toggleMatching` function via `extension` (that adds/removes an element to/from the `Set` based on whether it's already there based on `Identifiable`) might be nice.
5. When it comes to pinching, all pinches 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` in EmojiArt.
6. You will not be able to piggy-back your emoji-dragging onto the whole-document panning drag gesture. however. 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` will be required to support moving the selection around.
7. 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.

## 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 emojis. Doing this will very likely require you to have a more sophisticated `@GestureState` for your drag gesture.
2. Zooming in to high `zoomScale`s starts to make the emoji look a bit "grainy". This is because we are using `scaleEffect` to scale the `Text` up. These emoji would look quite a bit sharper if we scaled the font size itself. In other words, a font of size 100 is going to look sharper than a font of size 20 with a `scaleEffect` of 5. But as we learned back in Memorize, trying to zoom a font by just changing its size results in poor animation because the `.font` modifier is not animatable. See if you can figure out how to make an `AnimatableSystemFontModifier` that *will* animate the size of a system font and use that instead of the combination of `.font` and `.scaleEffect` we are using now. This `ViewModifier` can be written in 6 lines of code (not saying you have to do it that efficiently, but just so you know what's possible).