Assignment 1: MatchMarkers

Objective

The goal of this assignment is to recreate the demonstration given in the first two lectures.

Mostly this is about experiencing the creation of a project in Xcode and typing code in from scratch. **Do not copy/paste any of the code from anywhere.** Type it in and watch what Xcode does as you do so.

Be sure to review the Hints section below!

Also, check out 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 the start of lecture 4.

Materials

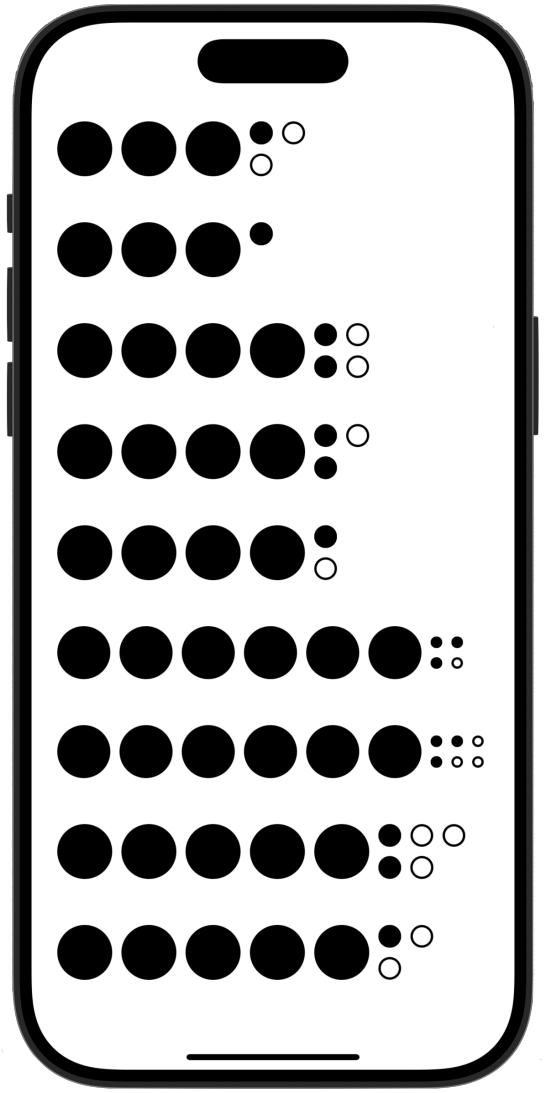
- You will need to install the (free) program called Xcode using the App Store on your Mac (versions of Xcode earlier than Xcode 16 will not work).
- In order to recreate the demo, you will certainly need to watch the first two lectures.

Required Tasks

1. Get the CodeBreaker app working as demonstrated in lecture. Type in all the code. Do not copy/paste from anywhere. Make sure the application runs successfully in the Simulator.
2. Enhance the `#Preview` of `MatchMarkers` to also show some "dummy pegs" next to the `MatchMarkers` so that the Preview Canvas is showing the `MatchMarkers` in more of it's "native environment." See the Screenshot below to get the general idea.
3. Enhance the `MatchMarkers View` to work properly when given an `Array` of anywhere between 3 and 6 `Match` values (we only ever pass 4 during lecture).
4. Update the `#Preview` of `MatchMarkers` you built above to exercise your code by trying out a bunch of different configurations of `Array<Match>` including `Array`s of 3, 4, 5 and 6 `Match` values and with a variety of `Match` (i.e. `.exact`, `.inexact`, `.nomatch`) values.
5. To properly preview what the `MatchMarkers` will look like when there is something other than 4 things in the `Array<Match>`, the `#Preview` will need to show 3, 4, 5 or 6 "dummy" pegs too (since each peg might be a `Match`, you'd never have, for example, 5 match markers unless there were 5 pegs capable of being matched).
6. Make sure your `#Preview` for `MatchMarkers` does something sensible in both Dark Mode and Light Mode.

Screenshot

1. Screenshots are only provided in this course to help if you are having trouble visualizing what the Required Tasks are asking you to do. Screenshots are **not** part of the Required Tasks themselves (i.e. your UI does **not** have to look exactly like what you see below).
2. Your `MatchMarkers` `#Preview` might look something like this (in Light Mode) …



Hints

1. We are only prototyping the components of our UI (specifically, `MatchMarkers`) this week. We'll implement our actual CodeBreaker game play next week.
2. A kind of ugly solution to Required Task 2 would be adding a lot of (repeated) lines of code to your `#Preview`, but consider instead creating your own "helicopter `View`" (maybe called something like `MatchMarkersPreview`?) which is used only by your `#Preview` and which shows the dummy pegs and the `MatchMarkers` together. Then your `#Preview` can consist only of uses of that "helicopter `View`" (inside a `VStack` or something). In other words "decompose" your `#Preview`. There's no rule that says you can't use whatever `Views` you want in your `#Preview`, even ones you create specifically for the purpose of making your `#Preview` better/cleaner.
3. Remember that you are allowed to use an `if` statement inside a `@ViewBuilder` (e.g. in the `var body` of `MatchMarkers`). This is a Hint, not a Required Task.
4. Required Task 3 can be accomplished in just a few lines of code. Very few.
5. Required Task 5 should require only a couple of lines of code to be changed. Literally.
6. If you do a good job on Required Tasks 2 and 5, Required Task 4 might take only 1 line of code per configuration you want to preview.
7. You'll likely find that Required Task 6 "just works" without any work at all on your part. That's one of the features of SwiftUI (i.e. that it adapts to the user's chosen environment with little work on your part—but we still always want to be double-checking this as we develop our application).
8. Assignment 2 will build directly on the work you are doing in this assignment. In other words, you will not be able to accomplish the Required Tasks in Assignment 2 if you have not successfully completed Assignment 1.
9. For fun, try to get your code to run on a physical device (i.e. on an iPhone).

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. Xcode 16
2. Swift 6
3. Understanding the syntax of a `@ViewBuilder` (e.g. "bag of Lego") `function`/`var`
4. `#Preview`
5. Putting `View`s together using `VStack`, `HStack`, etc.

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 SwiftUI API and knows how the CodeBreaker game code from lectures 1 and 2 works, but should not assume that they already know your (or any) solution to the assignment.

Assignment 1 may well not require any comments since it is mostly focussed on familiarizing yourself with Xcode and the process of entering, previewing and running code.