Assignment IV:

CodeWord Breaker History

Objective

This week you will enhance your existing CodeWord Breaker application to remember all the games you play (well, for this week, it will only remember until you restart your application—next week, we'll make it remember forever). You'll also be making your application multi-platform and adding some settings to it.

Be sure to review the Hints section below!

Due

This assignment is due before the start of lecture 13.

Materials

You will build directly on top of your Assignment 3 code.

As always, you are welcome to use any code from lecture.

You can use the internet (AI or whatever) to figure things out if you need to, but if you do, you must **cite it** in comments in the corresponding code. **You should not need that for the Required Tasks.** The Challenge Modes, however, might be a different story.

Required Tasks

1. Your existing A3 CodeWord Breaker's game-playing `View` must continue to work and satisfy all the Required Tasks from A3 except that **restarting a game is no longer allowed**.
2. The main `View` of your CodeWord Breaker application (i.e. the one specified in your `@main` code) should now show a `List` of all the games the user has ever played. This list will start out empty (unless you tackle one of the Challenge Modes) each time you launch your application, but will grow as the user plays more games.
3. The games in the `List` must be sorted with the most-recently-played game (i.e. the game with the most recently made attempt) first in the list.
4. The `List` of games should show **the last word attempted** in each game **along with its proper peg-matching indication**, ideally looking exactly like that attempt looked when the game was being played.
5. Games in the `List` that have no attempts yet (newly created games) should show blank or "redacted" pegs or something similar to indicate that no attempts have been made yet.
6. The `List` of games should also show the number of attempts that have been tried (so far) for each game.
7. Tapping on a game in the `List` must navigate to a `View` where the user can continue to play the game (i.e. with the `View` you wrote in A3). If the game is over, this `View` should simply show the state of the game when it ended.
8. There must be some way to start a new game in your application.
9. You are allowed to restrict your application to only playing games with words that are 5 letters in length, but see Challenge Modes for more on that.
10. Your application should work on iPhone, iPad and Mac (Designed for iPad). On iPad, the list of games and the currently-being-played game should be able to be on screen at the same time.
11. Support some "settings" for your application. See Hints for examples. You can postpone this Required Task to Assignment 5 if you want (since we did not cover things like putting up a `sheet` until lecture 12).
12. Your Model must utilize an `@Observable class`.

Hints

1. Fewer hints this week. It's time to start to be able to figure things out without any jumpstarts.
2. This Hint is just a reiteration that all of the Required Tasks should be doable with stuff you've learned in lecture only. If you think at some point that you need AI or internet searches to figure something out for the Required Tasks, you might be going off-track a bit. Some of the Challenge Modes, on the other hand, are specifically asking you to go outside of what you learned in lecture. In that case, internet resources (but also the documentation in Xcode) might be valuable. Don't forget to cite the resources you use.
3. Be sure to think about code reuse, especially with Required Task 4.
4. Here are some example "settings" for the last Required Task (note that this is a Hint, not a Required Task). Maybe the user can choose the colors for an `.exact` match peg, an `.inexact` match peg and a `.none` matching peg? Or whether the pegs are circular or rounded rectangles or some such? If you allow playing with 3, 4, 5 or 6 letter words (see Challenge Modes below), maybe there's a way for the user to choose a default word length? Have fun with this one.
5. Probably the easiest way to bring up your settings UI is with a "gear" icon in the toolbar above the main chooser `View`. The point of the settings Required Task is to show us you know how to put up a `sheet` with a `Form` full of controls like we did in lecture.
6. You might be sorely tempted to try to figure out how to persist your games between launchings of your application. We certainly won't hold it against you if you do, however, we are going to do that (in a specific way that you don't know how to do yet) in Assignment 5, so you'll probably want to wait until then to add that feature.
7. You'll probably find yourself wanting to share your "settings" across many of your `Views` (e.g. the `View`(s) that are setting the settings and all the `View`(s) that are using the settings). One approach to accomplish this sharing might be to use `@Environment var` in the same way `Words.swift` shares the dictionary of words across all `Views`. You could put your settings into an `@Observable class` (this `class` would *not* be part of your Model because these are probably mostly UI settings and `@Observable` is perfectly legal to use in your UI code too) and then create an `@Entry` in `EnvironmentValues` to hold an instance of that `class` which can then be used throughout your UI using `@Environment`. You'll probably want to do as `Words.swift` does, which is to create a `static let shared` on itself and use that as the `@Entry`'s default value.

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 CodeBreaker game code from the lectures works, but should not assume that they already know your (or any) solution to the assignment.

Challenge Mode

Challenge Mode is an opportunity to expand on what you've learned this week and push yourself with more difficult exercises. Attempting one or more of these each week is highly recommended to get the most out of this course.

1. 1 pt. Pre-load your list of games at application launch with a few sample games that are in various stages of completion for testing purposes. You'll have to think about how to represent these sample games in your code. There are lots and lots of ways to approach this.
2. 1 pt. Show how long each game has taken to play so far somewhere in the game-playing UI. It should only show the total amount of time the user has spent *with that game in front of them on screen*. This requires adding something like the elapsed time feature we added in lecture.
3. 1 pt. Show how long each game has taken to play so far *in the list of games* too. This should not be "ticking" unless the game in question is also on screen.
4. 1 pt. Support deleting games from the list of games.
5. 1 pt. Let the user choose whether to play with 3, 4, 5 or 6 letter words (not just 5 letters) when they create a new game. You'll have to figure out a way to ask the user what they want here each time they want to start a new game. It's okay to have it default to something (maybe a word length that is set in your "settings" or the last length the user played) but then have some alternate UI to choose a different length. `contextMenu` might be valuable here? Or maybe you can figure out how to use `confirmationDialog`?
6. 1 pt. Make your "settings" persist between launches of your application. Check out `UserDefaults` in the documentation as a way to store things like settings.