Welcome to the journey of testing Soundscape for RCOS! As someone who has delved into an issue within the project, I've compiled a set of tips aimed at enhancing your understanding of Soundscape, SwiftUI, and iOS app development. This document serves as your roadmap to navigate through the intricacies of our project, empowering you to contribute effectively to the testing efforts.

Within these pages, you'll find valuable insights derived from practical experience, aimed at accelerating your familiarity with Soundscape and SwiftUI. Whether you're a seasoned developer or just starting out, these tips are crafted to provide actionable guidance that will streamline your journey into the world of sound exploration and iOS development.

Think of this document as your personalized instruction manual, designed to equip you with the knowledge and skills needed to make meaningful contributions to Soundscape for RCOS

What is Soundscape?

Microsoft Soundscape [was] a product from Microsoft Research that explored the use of innovative audio-based technology to enable people to build a richer awareness of their surroundings, thus becoming more confident and empowered to get around. Unlike step-by-step navigation apps, Soundscape uses 3D audio cues to enrich ambient awareness and provide a new way to relate to the environment. It allows you to build a mental map and make personal route choices while being more comfortable within unfamiliar spaces. Soundscape is designed to be used by everyone and live in the background; therefore, feel free to use it in conjunction with other apps such as podcasts, audio books, email and even GPS navigation!

The issue that I worked on:

“Allow the user to configure the current setting of 15 metres for turning off the audio beacon. This distance was set in the original Soundscape and took into account the general level of accuracy experienced with GPS behaviour. However, based on user feedback, having the beacon turn off automatically can lead to frustration - hence the requirement to make it configurable by the user. This could be done via 5 metre increments or less.”

How to get started with building Soundscape?

1. Fork the repository from Soundscape Community to your own github, link: https://github.com/soundscape-community/soundscape?tab=readme-ov-file
2. Clone the repository that you forked from Soundscape Community to your own local machine.
3. Make sure that your MacOS and Xcode on your local machine have been updated to the newest version.
4. Open soundscape/apps/ios/GuideDogs.xcworkspace in Xcode.

***If you don't have an Apple Developer account and you want to build Soundscape, do the following:**

- Open GuideDogs.xcworkspace in xcode.
- press command 1, to open the project navigator.
- scroll all the way to the top, to the guide dogs project.
- Navigate to the "signing and capabilities" tab by hitting the button with the same label.
- Choose whatever kind of build you want to make (debug is probably fine)
- Expand the first signing option, (E.G. signing, debug), and change the team to your personal account. Make sure that "automatically manage signing" is checked.
- Either expand and change any other signing settings for build types if you need to (E.G. for release as well), or move on to the next step.
- Find the remove buttons for any capabilities. Specifically ones for notifications, cloud sync and associated domains. These require entitlements granted by a paid dev account.

There is more building tips on this page:https://github.com/soundscape-community/soundscape/wiki/build-tips

If new to SwiftUI and iOS development:

Here is some resource that would be useful to get familiar with SwiftUI and iOS development

- https://docs.swift.org/swift-book/documentation/the-swift-programming-language/guided tour/
  This is the official language guide.You don't need to memorize every chapter or anything like that, but it's a great resource to look over. It offers guided tours covering essential aspects of SwiftUI development, ranging from basic syntax to various data structures.

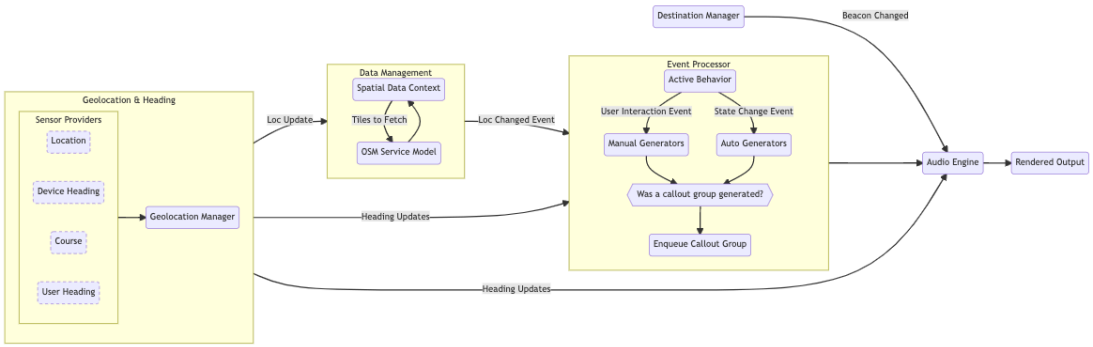
- https://developer.apple.com/tutorials/swiftui and https://developer.apple.com/tutorials/swiftui-concepts
  This is the official tutorial for SwiftUI and SwiftUI concepts from Apple. If you are new to programming and SwiftUI, this will be a great place to start.
- https://www.youtube.com/watch?v=09TeUXjzpKs
  This video teaches you how to get set up for iOS app development, how to use the tools required and how to write/read Swift code so that you can build your own app. If you are a visual learner, this will be a great place to start.
- https://www.youtube.com/watch?v=F2ojC6TNwws
  This video is a comprehensive tutorial for SwiftUI.

Tips for working on soundscape:

- Ask Professor Turner to add you into the Slack for Soundscape Community. If you have a question regarding iOS development, soundscape, and SwiftUI, you can feel free to ask developers in Slack.
- Additional documentation for Soundscape.
    - https://github.com/soundscape-community/soundscape/blob/main/docs/ios-client/overview.md

      This is an overview for additional documentation for Soundscape from Soundscape Community. You can find each document under the “docs” file in the repository.

    - These are documentation that I think that will be valuable to know before the contribution to the Soundscape Community.
        - Client Pipeline Architecture



        - **Event Processor**
          The core of Soundscape's audio user interface (UI) is managed and generated by two key components: the EventProcessor and SoundscapeBehavior, along with associated Generator objects. The EventProcessor performs three primary functions: maintaining the current "active behavior" object, transmitting events to this active behavior, and handling the output from events processed by the active behavior.

          Behaviors serve as containers for the primary experiences within Soundscape, encapsulating their respective logics. These logics are structured into sets of Generator objects, which can either implement the

AutomaticGenerator or ManualGenerator protocol alongside associated event types.

ManualGenerator-capable Generators handle events aligned with the UserInitiatedEvent protocol, reflecting direct user actions. Callouts generated as a result of such events override any ongoing callouts. Conversely, AutomaticGenerator-capable Generators process events associated with the StateChangeEvent protocol, reflecting state changes like alterations in the user's location. Typically, callouts generated by AutomaticGenerator objects queue up without interrupting ongoing callouts.

Generators specify the event types they respond to and can take various actions upon receiving an event:

Ignore the event (by returning nil).
Generate callouts based on the event (returned in a CalloutGroup object).
Update internal states without additional actions.
Update internal states and generate new events for processing.
By default, the SoundscapeBehavior initializes as the active behavior upon app launch. This behavior encompasses generators like AutoCalloutGenerator, IntersectionGenerator, and ExplorationGenerator. Custom behaviors such as PreviewBehavior, RouteGuidance, and OnboardingBehavior have also been developed.

When a custom behavior activates, the base SoundscapeBehavior becomes the parent behavior, with the custom behavior taking over as the active behavior. Events passed to the active behavior are first attempted to be handled by its generators. If none of these generators handle the event, it's then passed to the parent behavior for potential handling. The active behavior can control event distribution to specific parent generators and override event blocking properties as needed.

Generators can specify how callouts should behave, such as whether they should interrupt ongoing callouts or simply enqueue new ones. CalloutGroups provide mechanisms for determining the relevance of callouts before rendering, ensuring efficient audio cues delivery.

Overall, this intricate system enables Soundscape to dynamically generate audio cues based on user interactions and environmental changes, enhancing the overall user experience.

- **Continuous Dynamic Audio (Audio Beacons)**
  Rendering audio beacons entails a more intricate process compared to rendering discrete audio. The primary sound type utilized in this context is BeaconSound, which encapsulates a DynamicAudioEngineAsset. DynamicAudioEngineAsset implementations typically consist of enums where each case represents a component of the beacon. These cases must provide a selector function responsible for determining which component should play based on the user's orientation or location.

  BeaconSound assets are designed such that each component builds upon the previous one, creating a layered audio effect that intensifies as the user turns to face the beacon's direction.

  The DynamicAudioPlayer is tasked with initializing the necessary audio node objects for playing dynamic sounds, such as BeaconSound. As the user's orientation and location change, the DynamicAudioPlayer continually verifies if it's playing the correct component of the DynamicSound's DynamicAudioEngineAsset. Upon detecting a change in the asset that should be played, the DynamicAudioPlayer schedules the new asset to start playing on the next beat of the phrase. DynamicAudioEngineAsset implementations must specify the number of beats in the phrase for the represented audio, ensuring consistency across different components of the asset.

- **Data Management**

  ChatGPT
  When location updates occur, the SpatialDataManager is the initial recipient, enabling it to confirm whether map data surrounding the location has been downloaded. This data is fetched from the Soundscape services in segmented units known as "tiles". While these tiles do not contain traditional map vector data, they utilize the same system for determining data distribution within each tile, following the Mercator projection.

Soundscape employs tiles set at a fixed zoom level of 16, approximately covering an area of 600m x 600m per tile. To ensure seamless user experience, Soundscape caches all tiles within a 1000-meter radius of the user's location at any given time.

Once the required tiles are downloaded, the SpatialDataContext forwards location updates to the rest of the application, with the EventProcessor being a key recipient of these updates.

When other components within Soundscape necessitate access to nearby map data, they invoke the getDataView(for:searchDistance:) function on the SpatialDataContext. This function returns a SpatialDataView object, which consolidates map data from multiple tiles stored in the application's database into a unified view.

For further details on how Soundscape handles the download and storage of map data, refer to this resource.