Handover Document

Overview

The vision for our project is to develop an app that makes using Google TalkBack easier and more intuitive for vision-impaired users.

High-level view

Please refer to the Potential Solution Architecture document.

Risks

Please refer to the Risk Register document.

Communication

Discord is used as the main source of communication for this project. A server has been set up with general chats as well as role/team based text/voice chat. For the discord server link please contact the relevant people in charge.

Getting started

Hardware requirements

Phone

You or your development team must have at least one physical Android phone in order to test real-world gestures. The emulator has different behaviours as specified in the additional notes section.

As TalkBack does not significantly change in its behaviour with each new version, you can run a newer version of TalkBack, such as those bundled with newer Android versions. Preferably, you should run a phone with at least Android 9.0 (Pie) or later, as these have TalkBack installations that can be updated through the Play Store.

If your phone does not have TalkBack, we recommend using the Android Accessibility Suite from the Play Store as it is the only up-to-date distribution of TalkBack outside of the OS-provided version in Android itself.

Development computer

Your development computer should support Android Studio, which is the recommended IDE. This app does not depend on features exclusive to Android Studio, but was developed with it in mind.

Development requirements

SDK and emulator

For development, the following minimums are required:

- The latest version of Android Studio
- An Android SDK level above 26
- An Android emulator on Android 8.0 (Oreo) or later

We strongly recommend the following, as they will account for changes in TalkBack that have been added since Android 9 (Pie):

- An Android SDK with an API level of at least 28
- An Android emulator on Android 9.0 (Pie) or later

TalkBack

There are multiple TalkBack versions available. Since Android 9.0, they are no longer tied to the Android version you are running and are instead updated independently as an app. Using the latest version should allow you to work with new changes as they are released, as most old behaviours and gestures are still maintained in later versions.

Some minor changes have occurred, and you should consult Google's changelog for details. Beware; you cannot trivially check the TalkBack version being used.

TalkBack in the emulator

TalkBack is not installed in any emulator image provided in Android Studio. You **must** manually sideload a TalkBack version. For this, we recommend installing the Android Accessibility Suite from the Play Store. There are no modern stand-alone versions of TalkBack available to download.

Running the project

The project is a standard Android app and can be run through Android Studio to a connected device or emulator. Simply open the cloned project folder in Android Studio, build the app, and run it on a device.

We recommend using Project view instead of Android view as the latter has issues with our nested layout folders.

There are debugging settings that can be used to make the app easier to develop:

- Build flavours are available for disabling the intro, the TalkBack requirement, and the progression locks. There is a flavour for each combination of these settings.
- DebugSettings (a Kotlin object in the project source code) also configures skipping TTS and the database wipe procedure (read its comments - reinstalling/wiping the app on the device does the same thing)

External apps lesson

If you work on External Apps, you must also access the two additional repositories, calc-fork and svr-fork, which are also part of the GitHub organisation. To run these, you must change the run configuration in Android Studio to not launch any activities, as these are hidden apps and not making this change will cause the deployment to fail.

Current issues, bugs, and limitations

Bugs/Testing

Testing must be done manually using a physical device if possible. Emulators can be used for small fixes and temporary testing but not all features can be captured using an emulator.

Common bugs include:

- The Android Studio IDE being unable to find relevant files or values. Steps to fix this include:
    - File -> Sync project with Gradle Files
    - File -> Invalidate caches
    - Delete your local repository and re-download the repository from Github

- In-built Android TalkBack overwriting our custom TextToSpeechEngine. Steps to fix this include:
    - Intercept events such that TalkBack does not speak until the TextToSpeechEngine is finished speaking
    - Create a time delay long enough for TalkBack to finish speaking before using TextToSpeechEngine
    - Use TalkBack as the Text To Speech option by creating text boxes

- Database bug
    - After pulling from the GitHub repository, two of each module shows up on the lesson page. Steps to fix this include:
        - Go to DebugSetting.kt file
        - Set wipeDatabase to true and run the program (the program will automatically exit after finishing running)
        - Set wipeDatabase back to false

- After pulling from the GitHub repository, the program keeps crashing when it starts running. Steps to fix this include:
  - Go to TMTDatabase.kt file
  - Set the database version to 1

Additional notes for future developers

Emulator TalkBack differences

The Android emulator's TalkBack behaves quite differently to a physical phone's TalkBack:

- The gesture set is slightly different
- Normal movement on emulators is affected by reading modes, whereas phones have their own navigation gesture that uses reading modes
- Audio cues such as clicks and list-item ticks (which change in pitch) are missing from the emulator
- Haptic feedback is missing, and it is used by most gestures

TalkBack on reading modes

After the user switches between different text reading modes, swiping up or down would make TalkBack read the whole content of the text item box on the screen once before activating the specific reading mode. For example, when the user is in the "Character" reading mode and swipes down the first time to move to the next item, TalkBack will read the whole text content in the item box first. Then after swiping down the second time only then will TalkBack start reading through each character from the item box

TalkBack on untrackable gestures

Some gestures are untrackable since we do not have full access to the android developer API. When a user changes reading modes, we do not have the flexibility to track the reading modes that they have changed. Thus, it's difficult to come up with logics to track users that use different reading modes to tackle certain lessons. For example in Lesson 3 Module 2 about "jump text" reading modes, users can use "lines" reading modes to jump through items that were supposed to be learning "character" reading modes.

AccessibilityEvents Usage Documentation

We make use of accessibility events and the provided documentation does not provide enough detail for our tutorial. We have written our own documentation on how TalkBack uses these event types.

Event: `TYPE_VIEW_ACCESSIBILITY_FOCUSED`

This type of event is intercepted by binding a custom `View.AccessibilityDelegate()` to a view.

By detecting this event we can detect when TalkBack has focused on the `View` and can then perform specific actions, such as forcing TalkBack to focus on another element by sending an accessibility event.

Usage

- `Lesson3ChallengePart1Fragment.InterceptorDelegate` (L121)
- `JumpNavigationPart1Fragment.InterceptorDelegate` (L221)
- `AdjustSliderDelegate` (L45)

Event: `TYPE_VIEW_FOCUSED`

Generally sent via `View.sendAccessibilityEvent` to force TalkBack to focus on a specific View. This moves the cursor (the green box) to the View that sent this event.

Since this event hijacks TalkBack focus, it should be avoided in normal circumstances where it is not strictly needed in Teach Me TalkBack. All other apps should avoid explicitly sending this event.

Usage

- `Lesson1Part2Fragment` (L64)
- `Lesson1Part3Fragment` (L65)
- `ExploreMenuByTouchPart1Fragment` (L66)
- `Lesson3ChallengePart1Fragment.InterceptorDelegate` (L124, L126)
- `JumpNavigationPart1Fragment.InterceptorDelegate` (L224, L226)
- `ScrollPart1Fragment` (L43)
- `ScrollPart2Fragment` (L50)

Event: `TYPE_WINDOW_CONTENT_CHANGED`

Occurs when the content of a window (see Glossary) changes. All Accessibility Delegates attached within that window (often, there is one activity to a window) receive this event.

Any change to views, layouts, and so on will trigger this event. Examples include:

- Adding a new `TextView` to a LinearLayout
- Changing a `TextView`'s content
- Moving a slider
- Changing a view's size

To investigate: does this event require user interaction, or can it be sent if content changes automatically?

Usage

- `AdjustSliderDelegate` (L45, L52)
    - In this case represents when the value of a slider is changed
    - also used to disable TalkBack from reading the progress of the slider

Glossary

Window

On phones, windows are the full screen space taken up by an app (e.g. an activity with a layout). Each app in split mode is considered a window of its own.

The following are also treated as windows:

- The status bar
- The navigation bar (if not using full gesture navigation)
- Picture-in-Picture pop-ups
- Floating apps (e.g. those used on tablets)