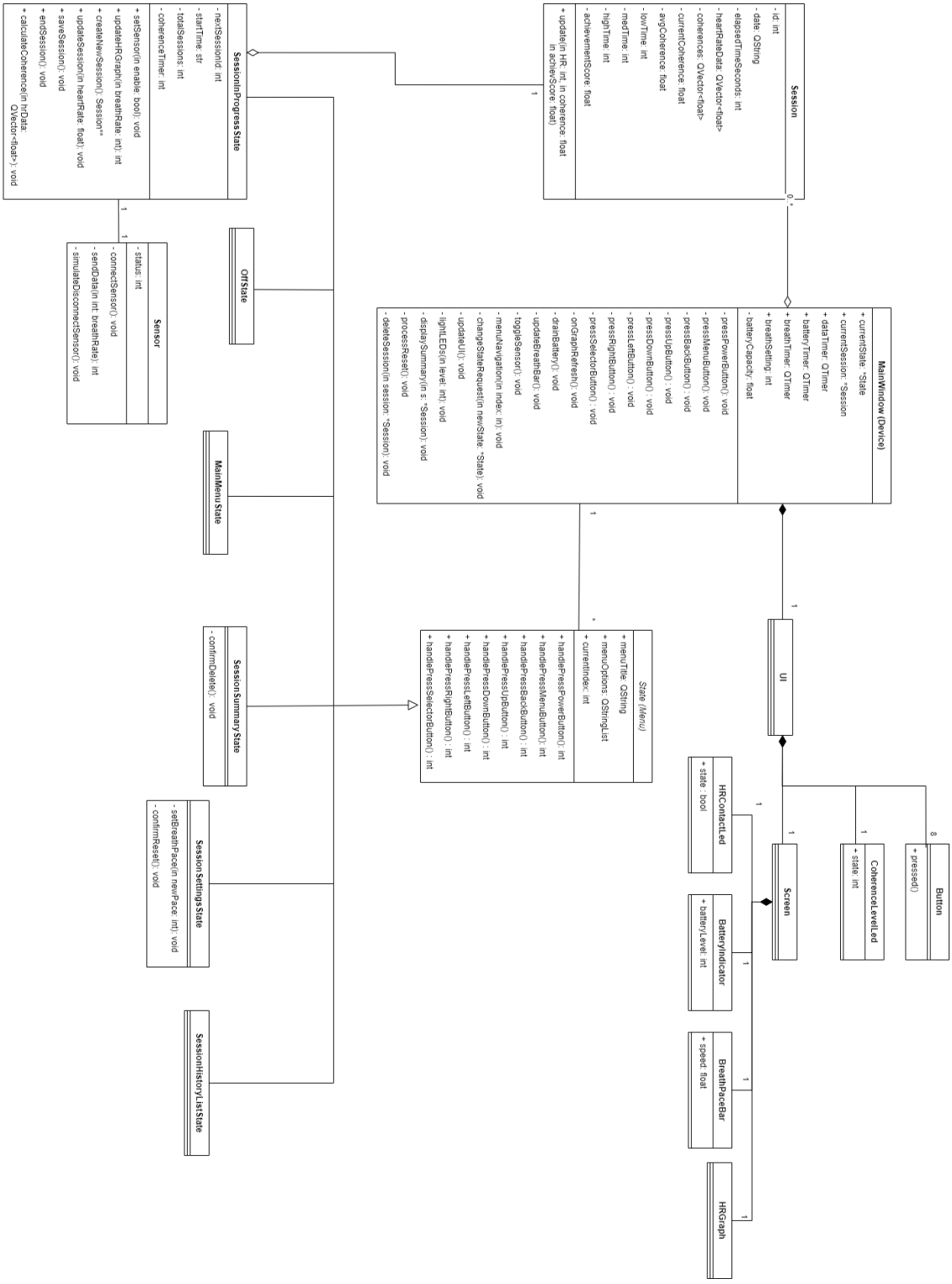
Full UML Diagram



List of Classes

- **MainWindow (Device)**
- **UI**
    - **Button**
    - **CoherenceLevelLed**
    - **Screen**
        - **BreathPaceBar**
        - **HRContactLed**
        - **BatteryIndicator**
    - **HRGraph**
- *State (abstract class)*
    - **OffState**
    - **MainMenuState**
    - **SessionInProgressState**
    - **SessionSummaryState**
    - **SessionSettingsState**
    - **SessionHistoryListState**
- **Sensor**
- **Session**

Overall design explanation

The user of the HeartWave device must be able to navigate menus to start a session. Once a session is started, the HeartWave device must record and display the user's heart rate information on a screen in real-time. Our design treats each menu as a separate state that the device is in. In order to go from one state to the next, the user must press a button (there are two exceptions). The device has 8 buttons, each of which can be pressed by the user when the device is in any state (there is no touchscreen). Some buttons will not have an effect when pressed in a certain state. Some buttons may also have different effects in two different states. This varying behaviour of the button depending on the state of the device is best captured by a state design pattern.

In the state design pattern, there are 6 concrete implementations of the abstract state class, each of which represents a menu (there are also 3 sub-menus). These concrete classes will have their own implementation of 8 handlePressButton functions, which are virtual functions of the abstract State class. These 6 concrete state classes are stored in the MainWindow (Device) class, which has a current state member (pointer to a State object) to indicate the device's active state.

When a button is pressed on the UI. A signal will be sent to the MainWindow which will call a slot to indicate that a specific button has been pressed. This slot will call the current state's corresponding handlePressButton function. Ideally, this allows us to implement state-specific behaviour in the respective concrete state class and reduce the amount of clutter in

MainWindow. This also keeps all of the possible state change interactions organized and easy to understand (see button state change/effect matrix).

The downsides of using the state design pattern (we would need to implement a new state for every new type of menu and menu item) are less apparent as there are not too many Menus and MenuItems to implement.

The data collection and reporting is mainly done in the SessionInProgressState class. A Session object is created and tied to the SessionInProgressState. The Session class acts as a storage class for all session data while the Data is fed to the device through the Sensor class (simulated data in this case).Once a session is over the SessionInProgressState saves the Session object to the MainWindow which stores a collection of saved Sessions.

Button State Change/Effect Matrix

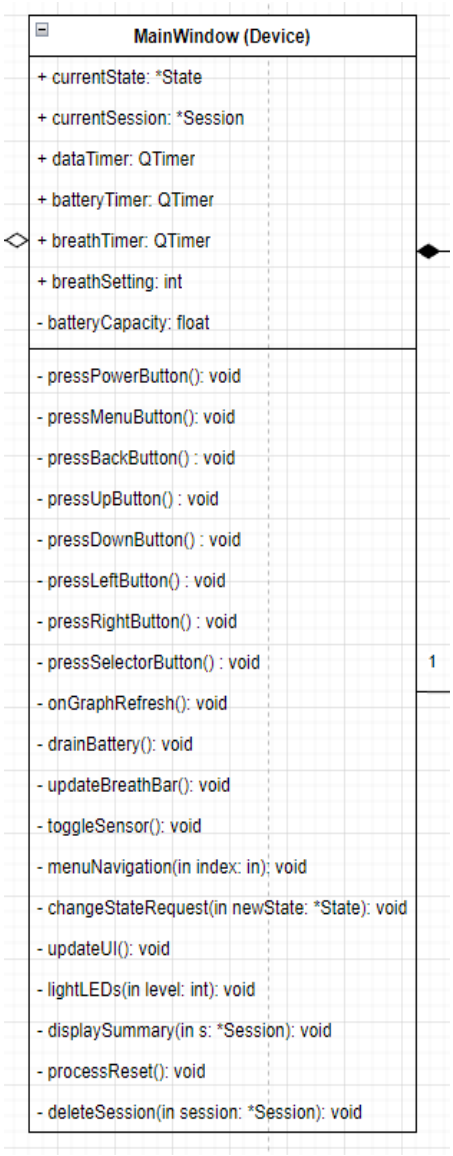
The logic capture in this table can be cleanly organized by using a state design pattern

Menu (ID)	power	menu	back	select	select setting	select history	select breath	select reset	select yes	select no
Off (0)	1	--	--	--	--	--	--	--	--	--
MainMenu (1)	0	1	--	2,3	2	3	--	--	--	--
Session Settings (2)	0	1	1	9,10	--	--	10	9	--	--
Session History (3)	0	1	1	4	--	--	--	--	--	--
Session Summary (4)	0	1	3	7	--	--	--	--	--	--
SessionIn Progress (5)	0	--	--	6	--	--	--	--	--	--
SessionEnd Summary(6)	0	1	1	8	--	--	--	--	--	--
Summary Delete (7)	0	1	4	1,4	--	--	--	--	1	4
SummaryEnd Delete (8)	0	1	6	1,6	--	--	--	--	1	6
Reset Confirm (9)	0	1	2	1,2	--	--	--	--	1	2
Breath Pace (10)	0	1	2	1	--	--	--	--	--	--

Note 1: The selector button can initiate changes to different states depending on what menu item is currently being selected

Note 2: The Up, Down, Left, and Right buttons are for menu navigation and not involved in state changes

Individual Class Documentation



MainWindow (Device)

The MainWindow class handles any input received from the UI (slots and signals), handles any change to be made to the UI (QTimers), stores a list of saved sessions, and handles anything related to the battery. The MainWindow also plays a major role in navigating the menus as it stores a QVector of device State concrete classes as well as a **currentState** which dictates the behaviour of button presses.

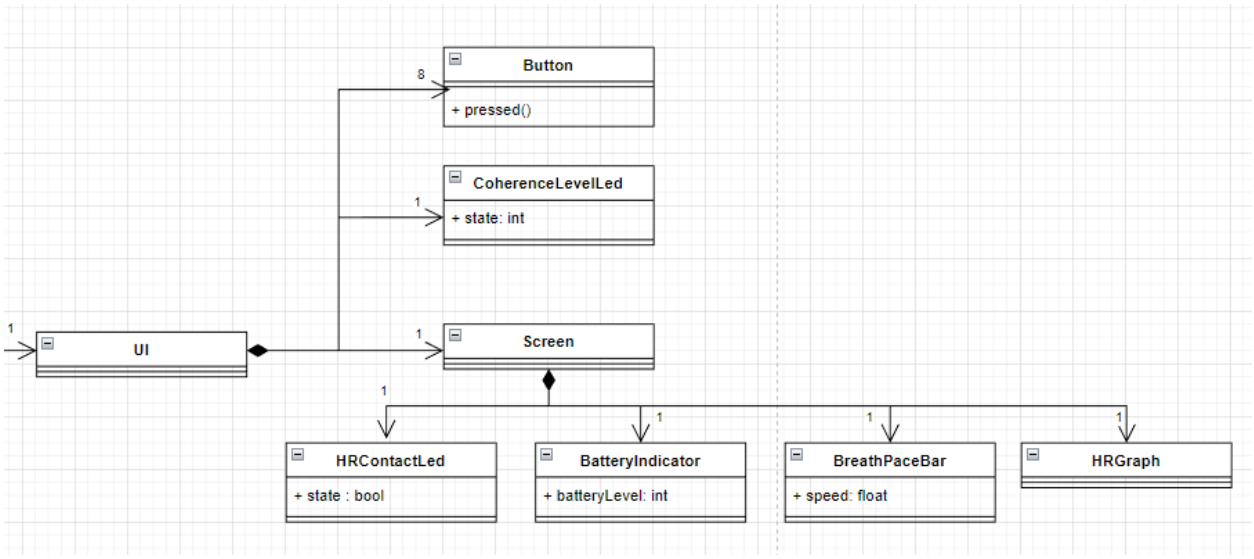
The general flow of button presses is as follows:

1. The user presses a button
2. MainWindow pressXButton() method is called
3. currentState's handlePressXButton() method is called (and any state-specific interactions are handled)
4. The return value of currrentState's handler is the newState that the device should be in (if applicable)
5. MainWindow updates the currentState
6. MainWindow performs any UI updates associated with the newState

The general flow QTimer timeout is as follows:

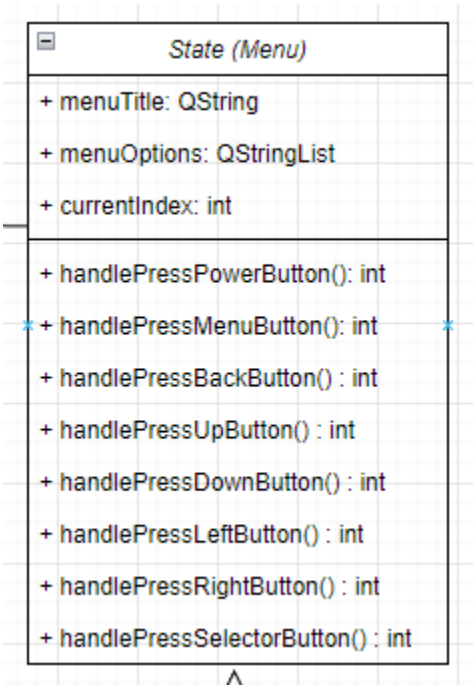
1. Timer is started (either when MainWindow starts or at the start of a session)
2. Upon timer timeout, will call a timeout handler (i.e. onGraphRefresh(), drainBattery(), updateBreathBar())
3. The handler will update the UI and perform any additional processing if necessary
4. A new timer is started with the same duration as the previous one.

UI



The UI is solely controlled by the MainWindow. Many of the slots that are implemented in MainWindow receive signals from the UI (the others are from QTimers). MainWindow's updateUI() method is a generalization for updating the UI based on the currentState. The actual implementation may choose to split this update UI task across many different methods and they may be placed in the concrete State classes instead.

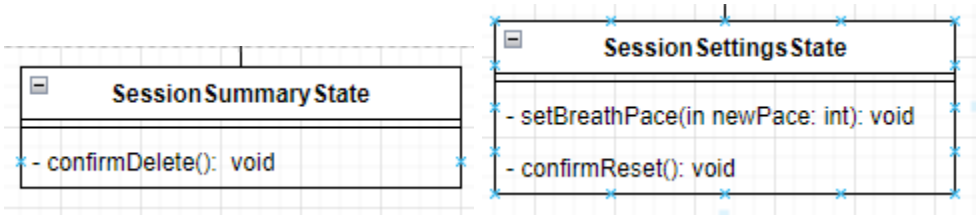
State (abstract class)



The handlePressXButton() methods return an int value representing the new state (see state change matrix). The behaviour implemented in the abstract class serves as the 'default' behaviour for each button (e.g. Power button turns the device off as it is applicable to more states). Each handle button method can be overridden in the concrete State classes if specific behaviour needs to be defined.

Certain States will have a list of menuOptions, implemented as a QStringList inside a QComboBox. States need to be able to track which option is currently being selected for specific selector button interactions.

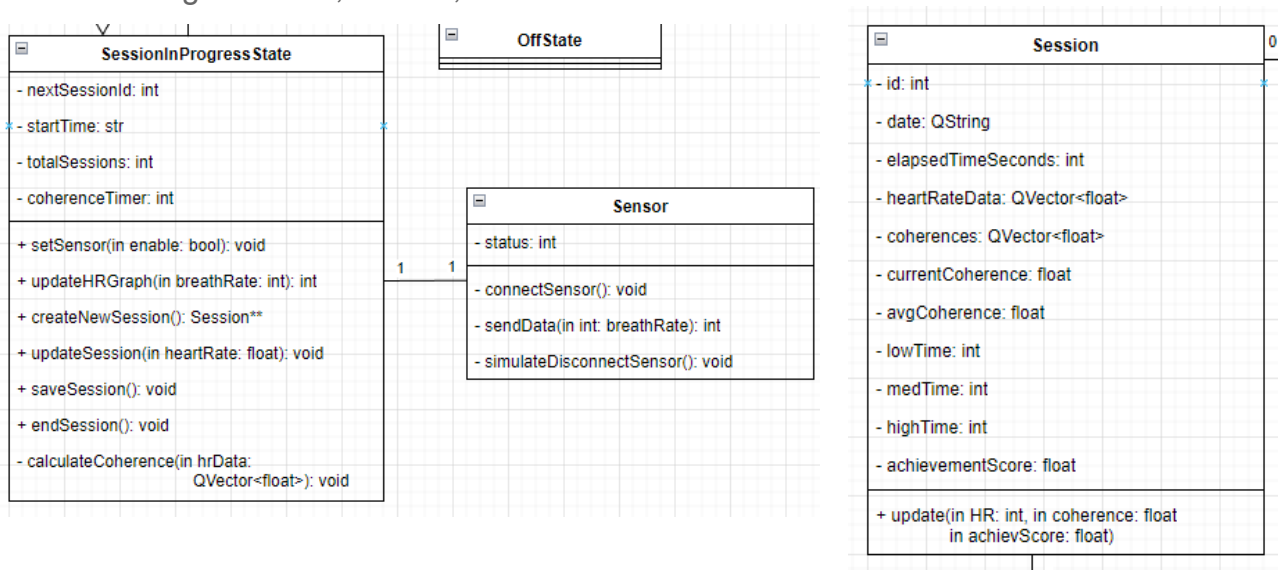
Concrete State Classes



For the button-related differences in the concrete state classes see the state change matrix. Some concrete state classes have specific functionality within them such as the Summary state and the ability to delete the session in the session summary state and the ability to set breath pace and perform a data reset in the settings state.

The SessionInProgress state has many differences from other concrete state classes and will be discussed in its own section

SessionInProgressState, Sensor, and Session

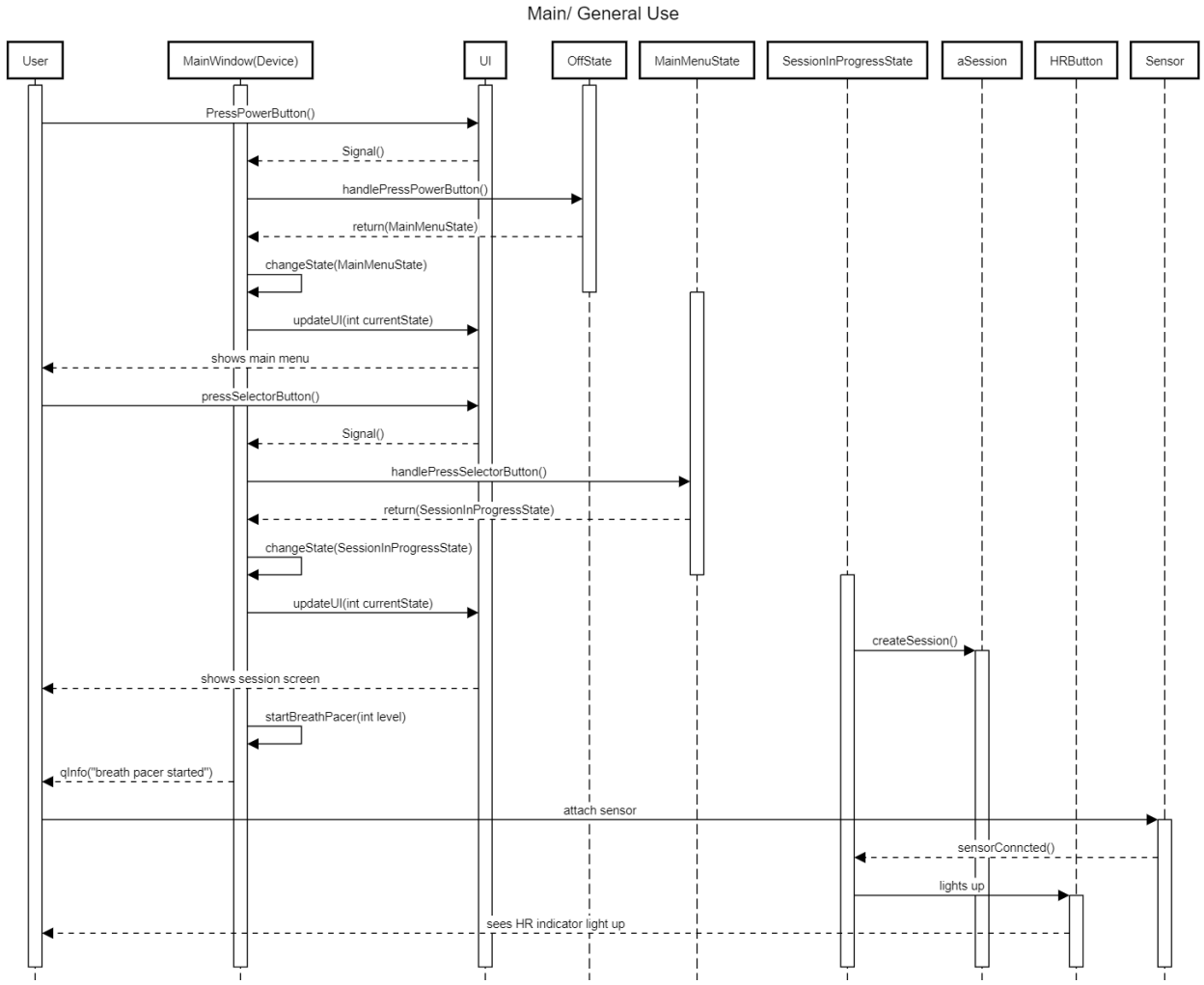


The session in progress state represents the state where the device is actively recording a session and displaying up-to-date information about the user's HR, coherence, and other metrics. Whenever we enter this state, a new Session object is created which stores all of the data related to the Session. This Session object is accessible both by the SessionInProgressState, which updates the Session's data using information fed from a Sensor and the MainWindow state which takes data from the Session and uses it to update the UI. Data is fed from the Sensor to the SessionInProgressState on a 1-second QTimer which is implemented in MainWindow.

When a session is stopped, a pointer to the Session object will be stored in the MainWindow's session history Qvector. The SessionHistoryListState is also updated with the newest saved session.

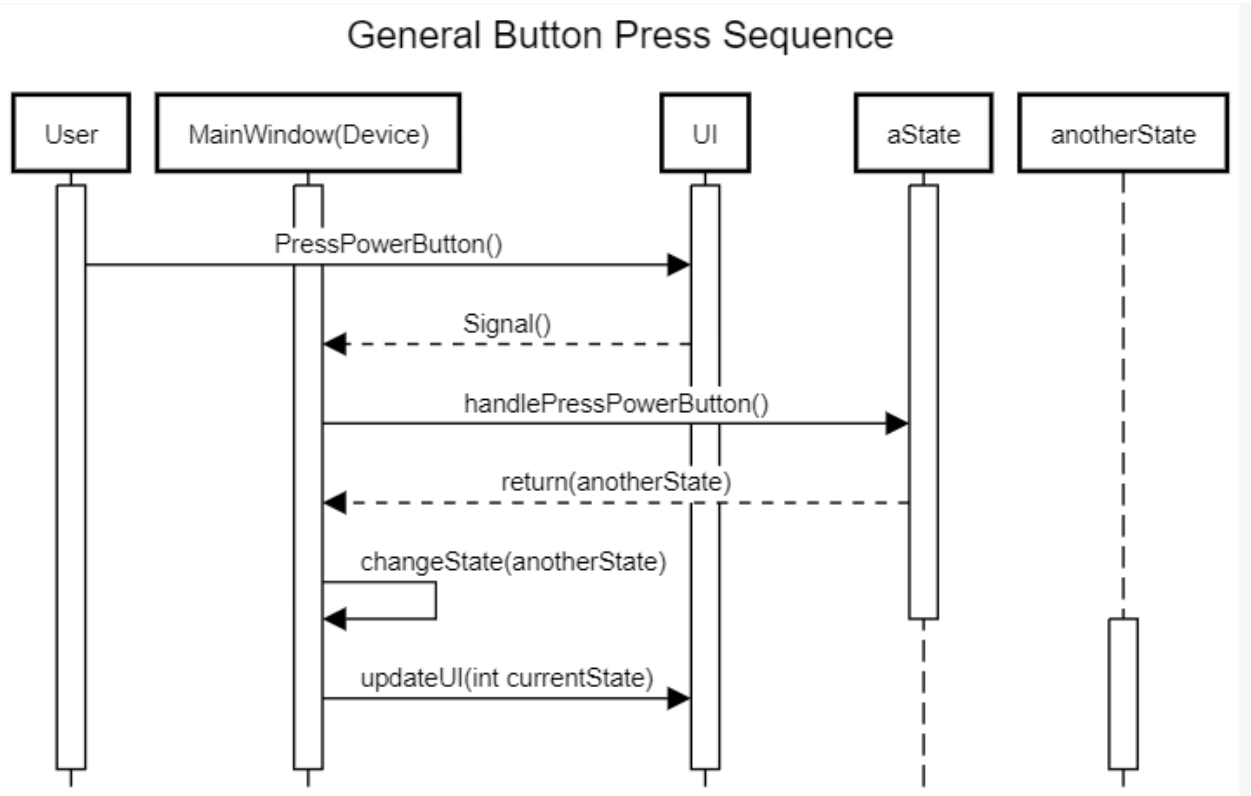
Sequence Diagrams

Use Case #1: Basic Use Case Pt.1 Starting a Session

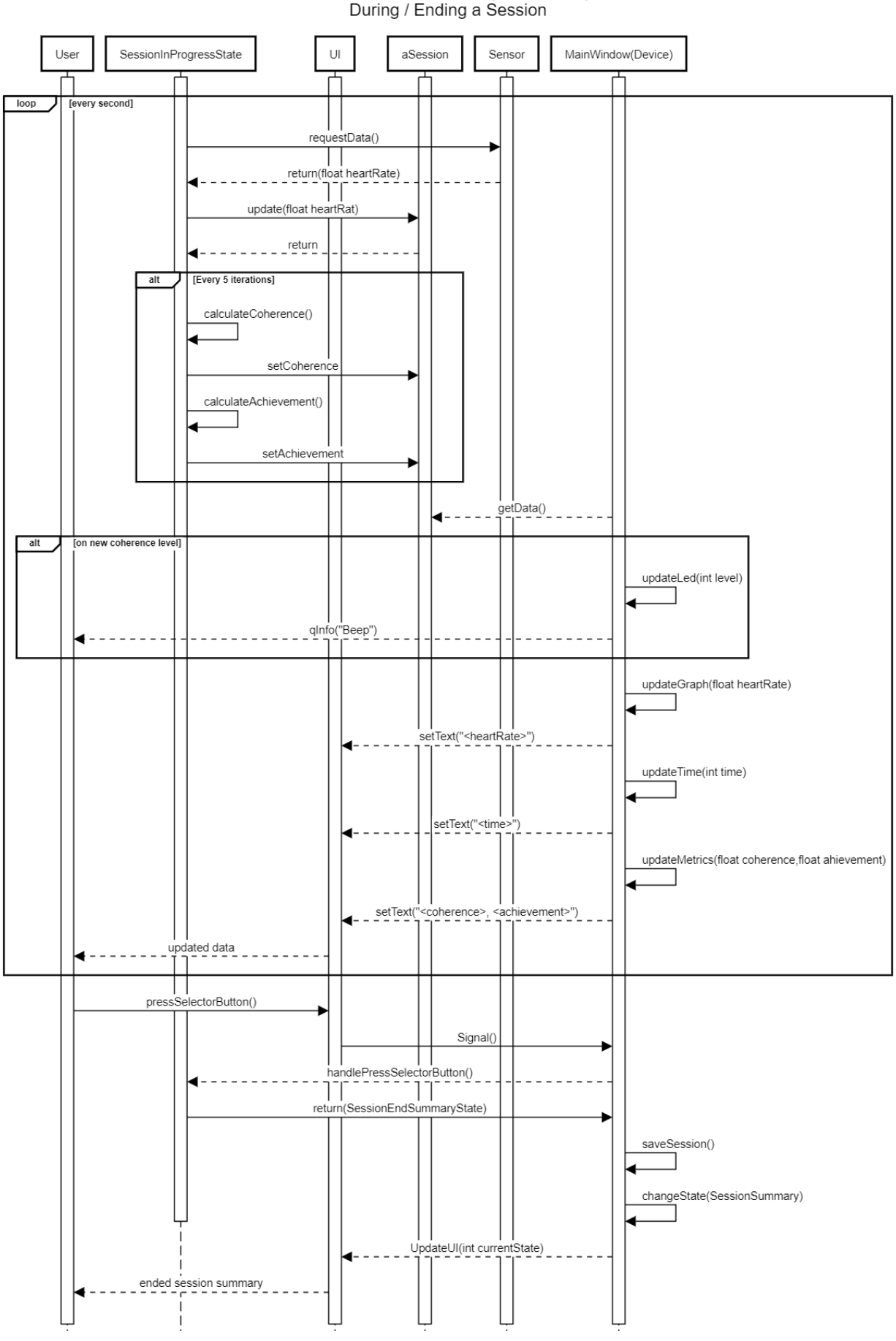


The basic use case illustrates the button-press interaction which is a pattern that appears multiple times in other sequence diagrams (see the general button press sequence below). Notably even though the changeState occurs in the MainWindow, the new state is already listening and ready to respond to new signals. UpdateUI is an abstraction of the many methods that can occur when the UI actually needs to be updated to a new state.

Attach sensor done right before the


General Button Press Sequence
User
MainWindow(Device)
UI
aState
anotherState
PressPowerButton()
Signal()
handlePressPowerButton()
return(anotherState)
changeState(anotherState)
updateUI(int currentState)

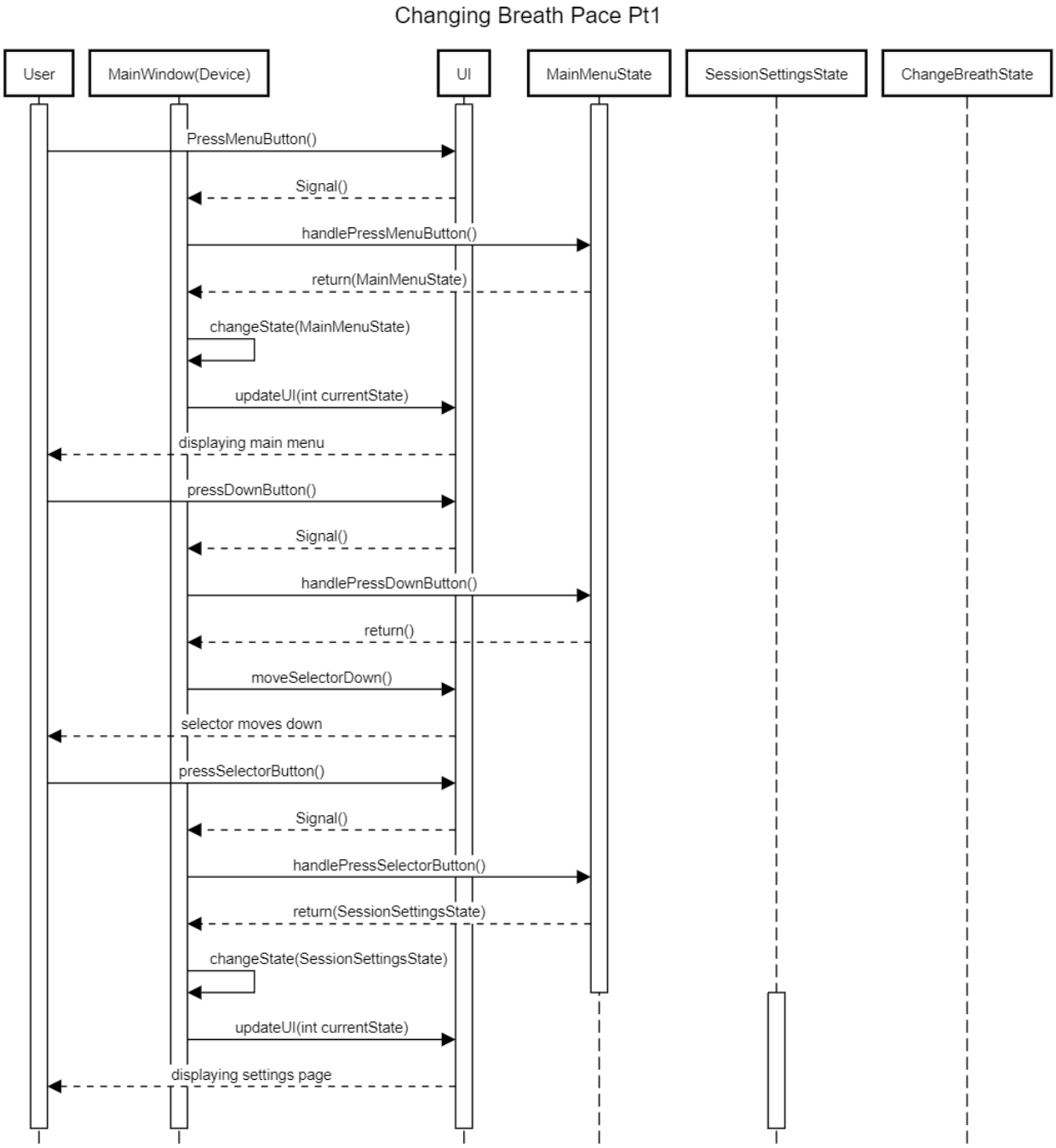
Use Case #2: Basic Use Case Pt.2 During/Ending a Session



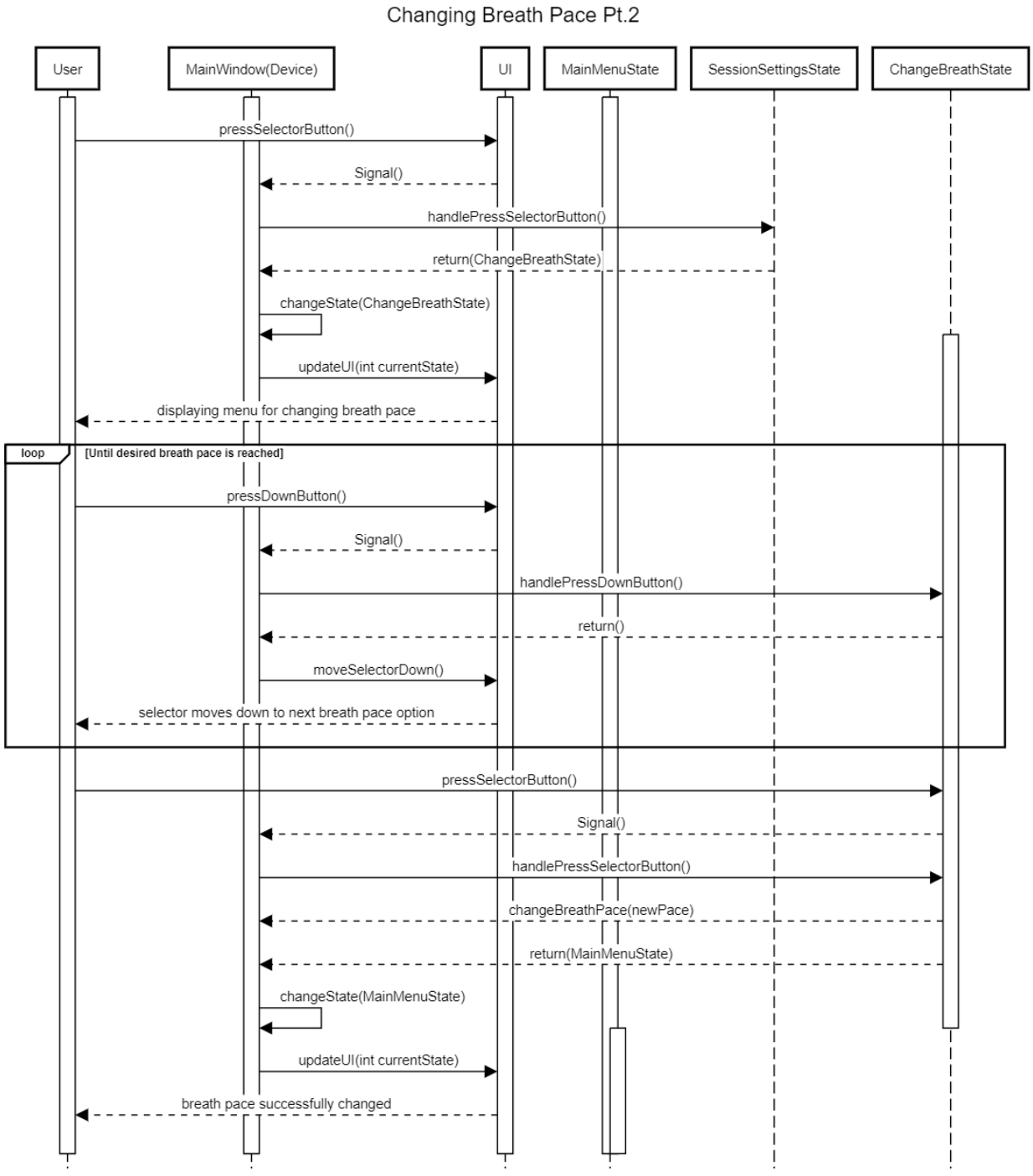
UC2 describes the data collection loop. Every second the SessionInProgressState will request data from the Sensor and update its associated Session. Every 5 seconds, coherence and achievement will also be calculated. The MainWindow then requests this data from the active Session every second and updates the UI accordingly.

When a Session is terminated (normally by pressing the selector button), the Session will be saved to the MainWindow and the state will change to the SessionSummary state. Viewing this saved session will be discussed in the sequence diagram for UC5.

Use Case #3: Changing the Breath Pace in Settings

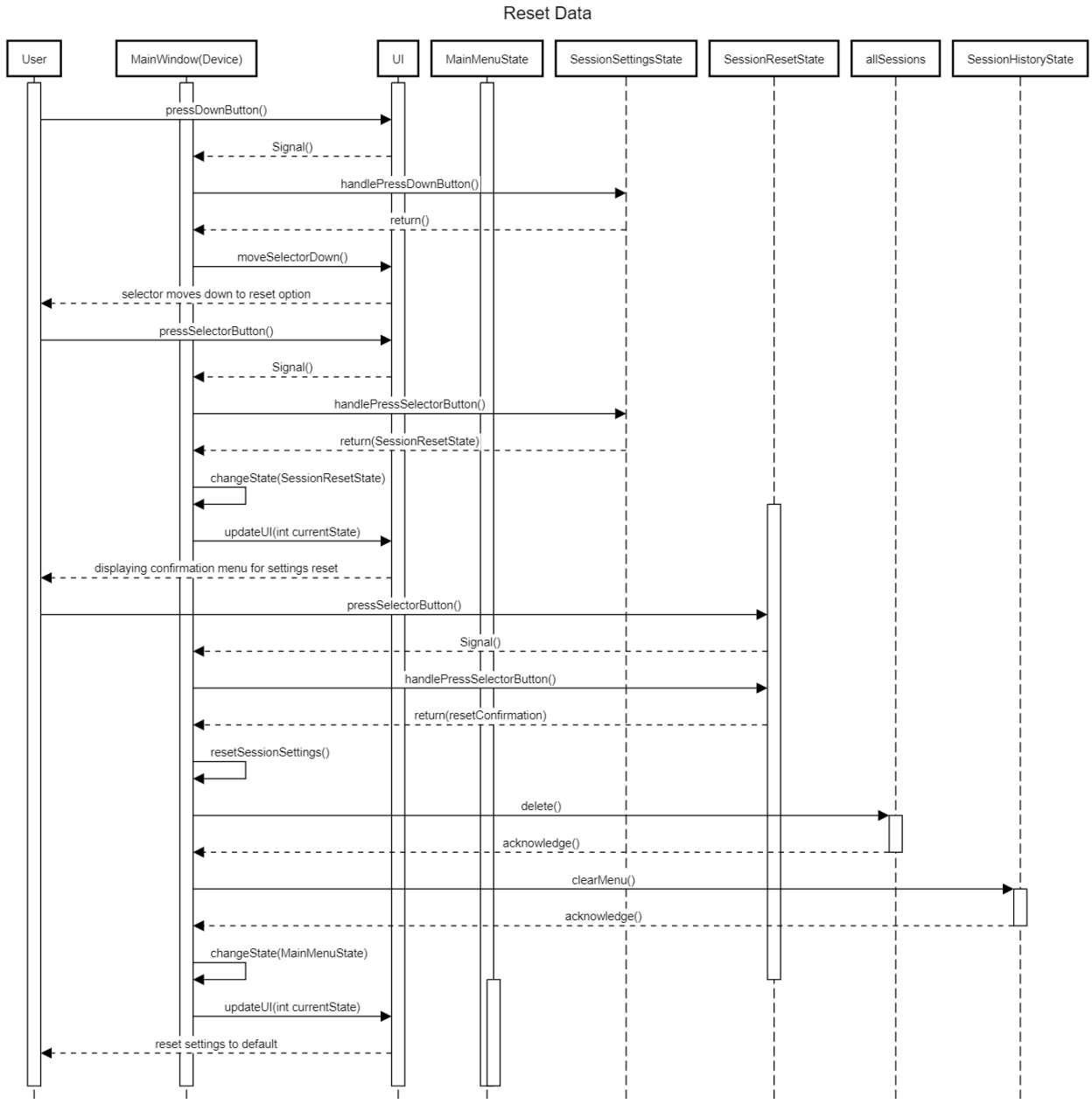


The first part of UC3 details how to navigate from the MainMenu to the Settings Menu (and can also be thought of as a sequence diagram for navigating menus using the down button).



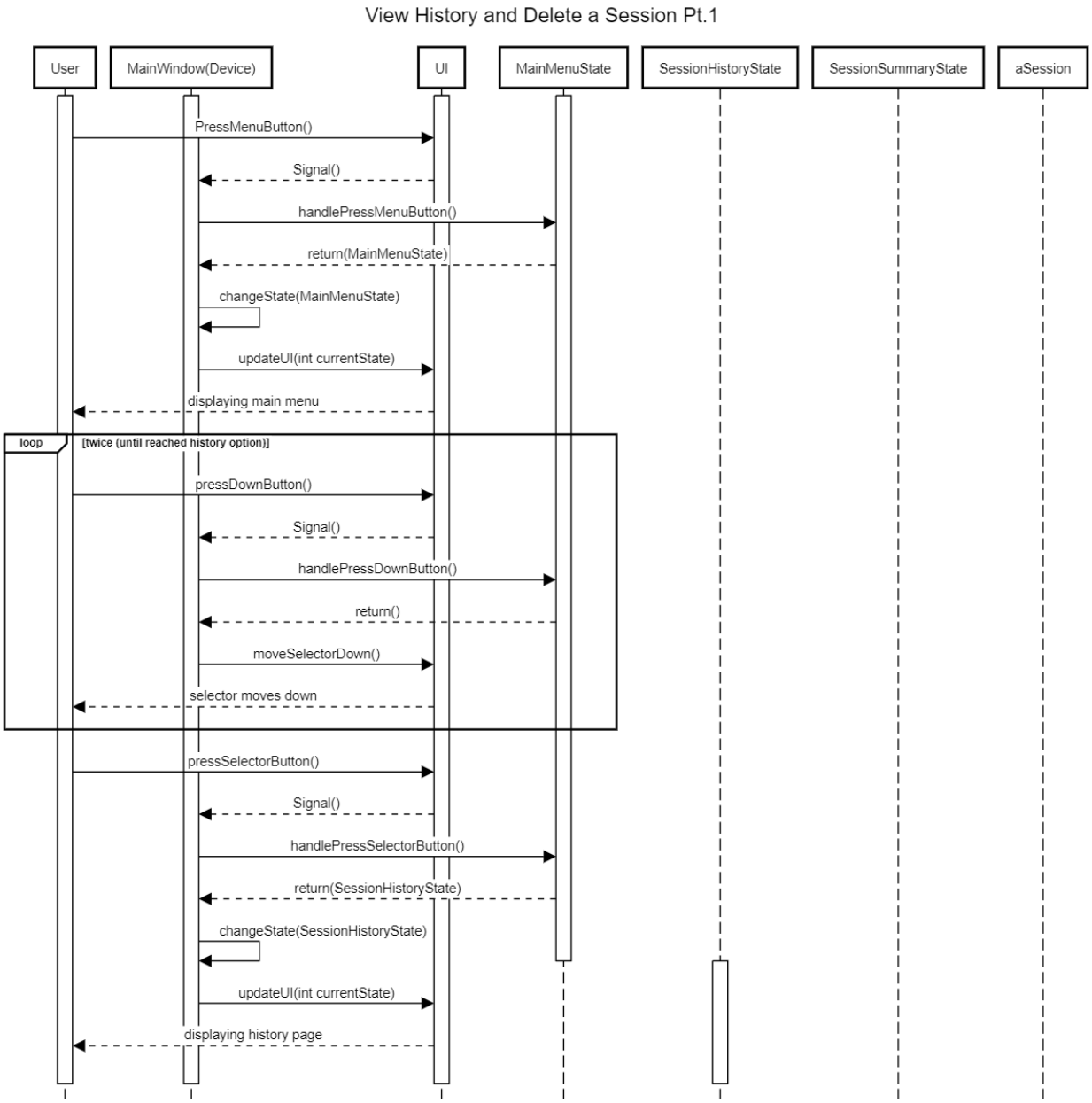
The second part of the sequence diagram outlines how to access the change breath pace menu and how to reach your desired breath pace through more menu navigation.

Use Case #4: Reset Data Through Settings

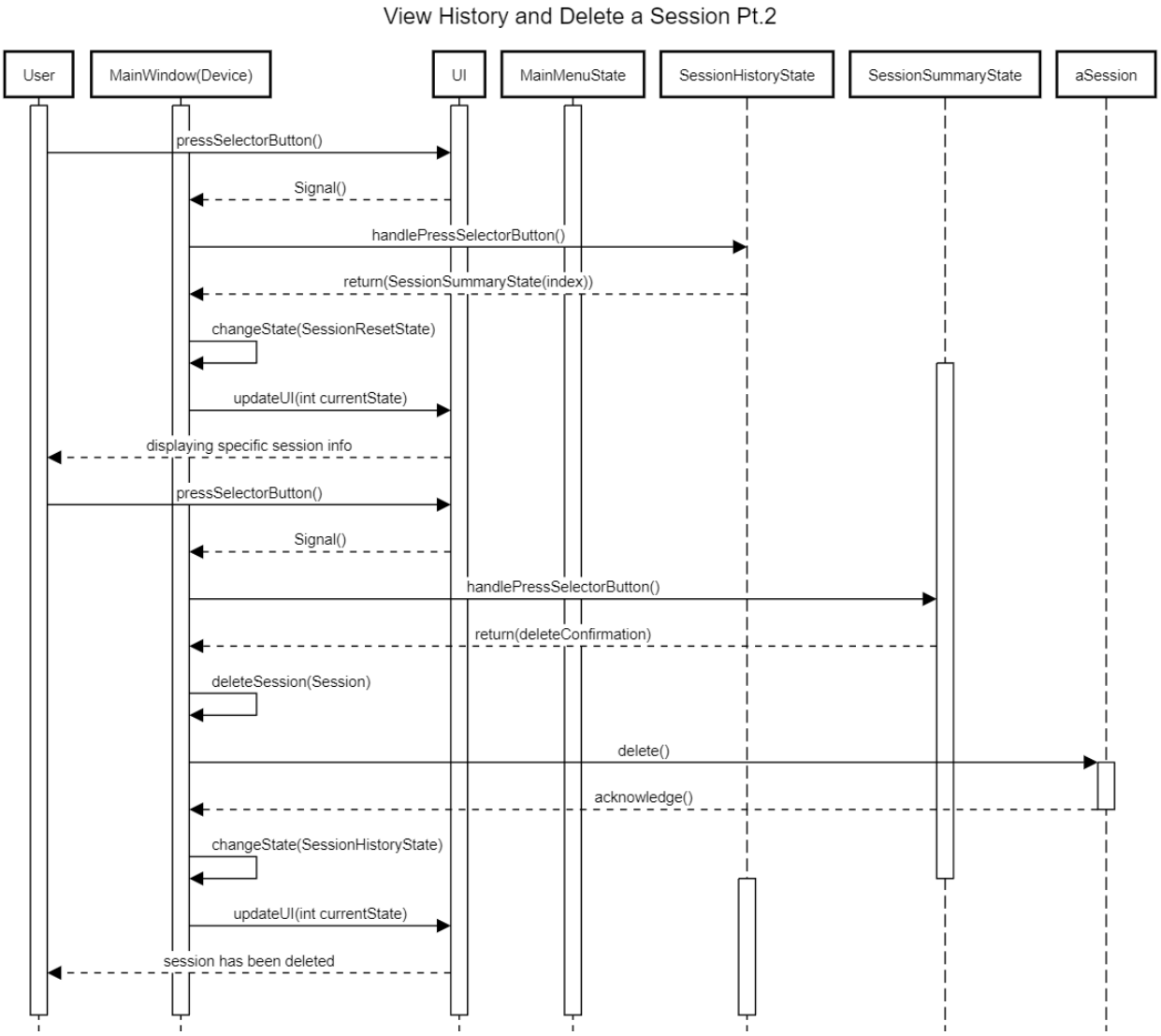


The first part of UC4 is identical to the first part of UC3 so it will be omitted. UC4 outlines how to reset all data and preferred settings on the device. It is very similar to UC3 except with a bit of different menuing as the user would need to navigate through a reset confirmation menu before being able to reset device data.

Use Case #5: Viewing Session History and Deleting a Session

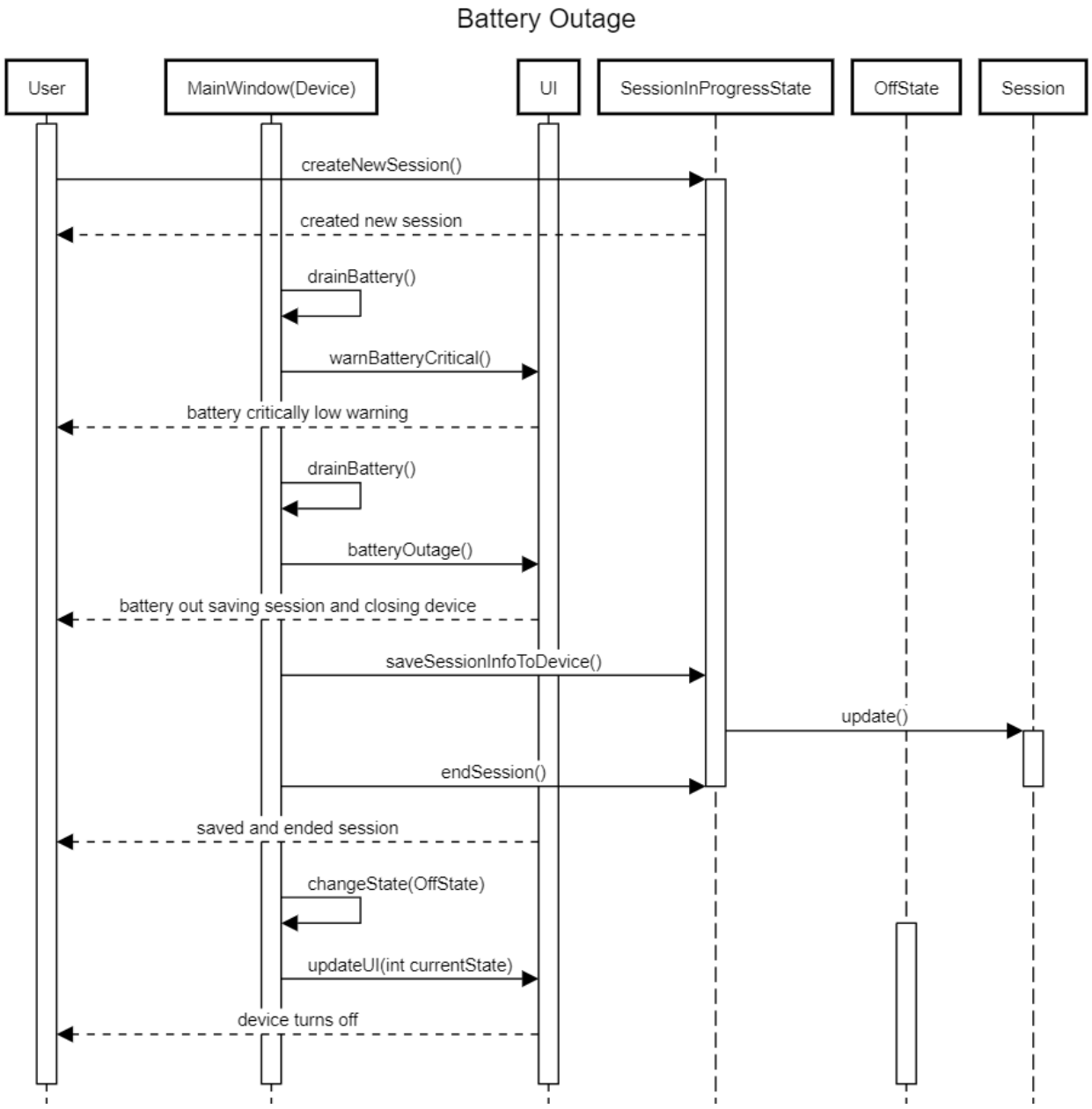


This is part one of viewing session history which involves navigating to the history menu from the main menu. It is similar to Pt.1 of UC3 and UC4 with the exception that the user will need to press the down button one additional time.



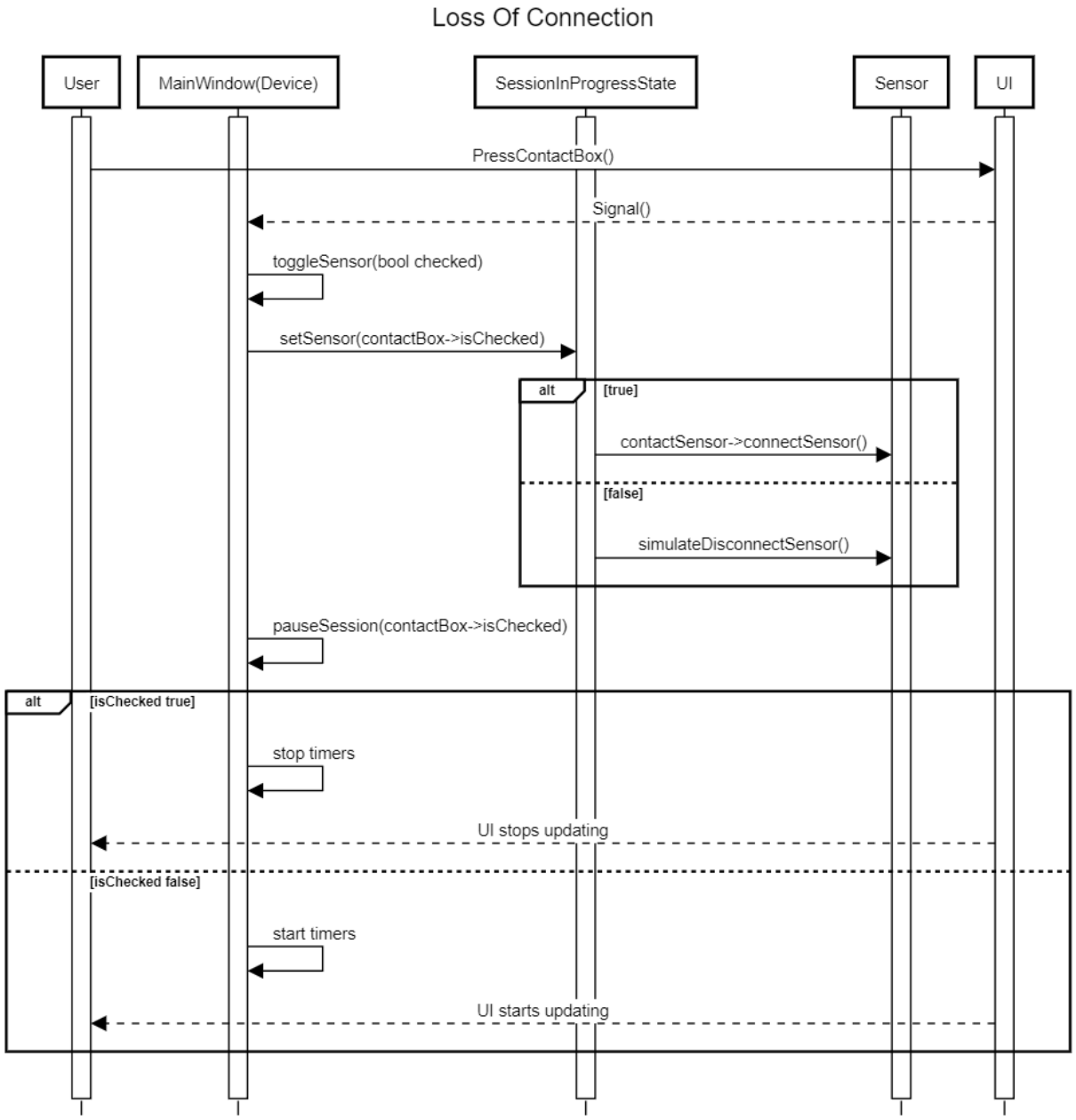
The second part of UC5 involves navigating the SessionHistoryState and viewing and deleting the actual session. Again, this mainly involves menuing. Similar to UC4's reset confirmation there is a deletion confirmation screen when a user attempts to delete a session.

Use Case #6: Battery Critical and Battery Outage



UC6 details the scenario where the battery is draining during the middle of a session (SessionInProgress). When the battery levels are critically low, then a warning will be sent to the user to charge the device. If the battery drains further to 0%, then the device will attempt to end the current session and save what ever data has already been collected before shutting off.

Use Case #7: Disconnecting Sensor During a Session



The Sensor being disconnected during a Session is handled by the MainWindow and SessionInProgressState. Since this is a simulation, sensor connection/disconnection is handled by a UI checkbox. If the sensor is disconnected, session data collection will pause and the MainWindow will pause updating the UI. These activities will resume if the sensor is reconnected.