F.2 Computer Literacy
Coding with App Inventor 2 - Assignment 6

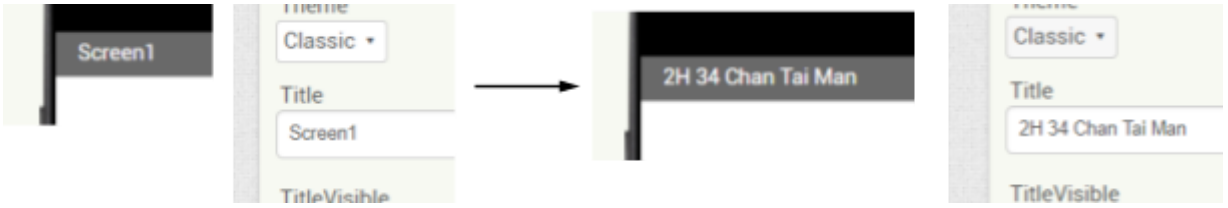
Project idea:

Build a musical instrument app (Xylophone) with recording and playback functionality based on the tutorial on the MIT App Inventor website, and improve on the program structure with the use of Generic components blocks (part 1) and a different data structure (part 2) for the recordings.

Requirements (Part 2):

1. Use "Save project as …" to save the project of part 1 in the last assignment with another name "Xylo_part2_***class_class no***", using your actual class and class number.

2. Check that the screen1 title has Your class, class no. and name



3. Read the pdf file **Ch9-part2.pdf** on the assignment page (starting from the Recording part) (An online-version is also available at the http://www.appinventor.org/Xylophone2 but sometimes the site is not available) to get an idea of how to make recordings and playback with proper delay for each notes. **Pay attention to the required modifications below.**

4. Read the notes below on the alternative way to store the song inside the app, and follow the preliminary tasks below to create the additional functions and procedures required for the data structure we are going to use in the app to store the notes and durations of each note to be recorded.

5. The end product should at least have the same functions as the one shown in the book; i.e. allow the user to playback the sequence of notes played so far with the correct time delay between notes, and the user can clear the recorded song to start again.

6. After adding the playback button and the reset button, you have to modify the generic handler to check whether the component is actually a note button (since the generic handler will also be invoked when the playback or reset buttons are clicked) to avoid errors.

7. You are free to add more functionalities to make a better app (e.g. allow playback at different rates, record multiple songs, etc)

Notes on alternative way to store a song

In the second part of the chapter, in the making of the recording and playback part, it is suggested to use parallel arrays to store the notes and the clock-times at which each note is clicked.

Excerpt from the book:

> For example, if you play "Row, Row, Row Your Boat" [C C C D E], your lists would end up having five entries, which might appear as follows:
>
> - `notes`: 1.wav, 1.wav, 1.wav, 2.wav, 3.wav
> - `times` [dates omitted]: 12:00:01, 12:00:02, 12:00:03, 12:00:03.5, 12:00:04

Only when the song is played back, the duration of each note is calculated.

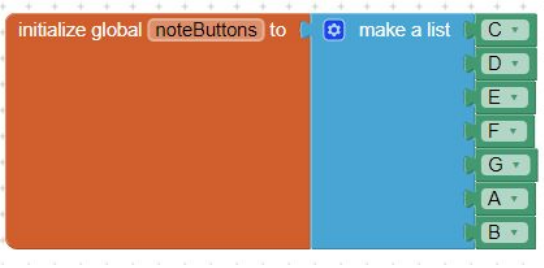
In other words, each time the song is played, it has to repeat these calculations!

To avoid such repetition of calculations, an alternative way to store these notes is used.

In our app, there are several differences in the data representation:

1. Notes are represented by their index (i.e. position) inside the list of buttons (the "noteButtons" list made in the last assignment that stores the buttons of the notes) instead of their sound file names.

   e.g. suppose your noteButtons list is defined as

   Then note-C is represented by the number 1, note-D represented by the number 2, etc. This value can be obtained by using the [index in list (thing, list)] block inside the generic handler of the Button.Click:

   So, **instead of storing "C.wav" for note-C or "D.wav" for note-D, etc, we store 1 for note-C, 2 for note-D, etc** (your value can be different; the exact value in your app depends on the list of noteButtons you had constructed. But actually you do not have to worry about the exact values as they are always found by using the [index in list] block)

2. Instead of storing the clock-times of the clicking of each note, the **duration** of each note is stored.

   Using the above examples, instead of storing

   12:00:01, 12:00:02, 12:00:03, 12:00:03.5, 12:00:04

   We want to store the durations 00:00:01, 00:00:01, 00:00:00.5, 00:00:00.5 ... etc (i.e. the difference between the clock-times of a note *and the next note*)

3. The note and the duration are put together into a single unit as a pair (i.e. a list of 2 elements)

   e.g. note-F and duration of 00:00:03 is stored as the pair [4, 00:00:03]

   Let's call it an ***NDpair*** (Note-Duration pair)

   However, *the duration of a note cannot be found until the next note is clicked by the user*. How can we store the note and its duration once a note is clicked?

   One way to solve this problem is to first store an invalid value (e.g. 0) as the duration for the note just clicked, and only update its value when the next note is clicked by the user.

   e.g. suppose five notes are just clicked and their clock-times are in sequence

   C, C, C, D, E
   12:00:00, 12:00:02, 12:00:04, 12:00:05, 12:00:06

   We want to store the following NDpairs (the last NDpair has an invalid value "0" as duration since it has no "next note" yet)

   [1, 00:00:02], [1, 00:00:02], [1, 00:00:01], [2, 00:00:01], [3, 0]

4. The NDpairs are stored inside a list in the global variable "song".

Algorithm to construct the data to store into the list "song"

The following steps should be performed inside the generic handler of Button.Click:

1. Whenever the user clicks a note, a new NDpair containing the note value and a temporary invalid value 0 as duration is constructed, and this new NDpair is added to "song".

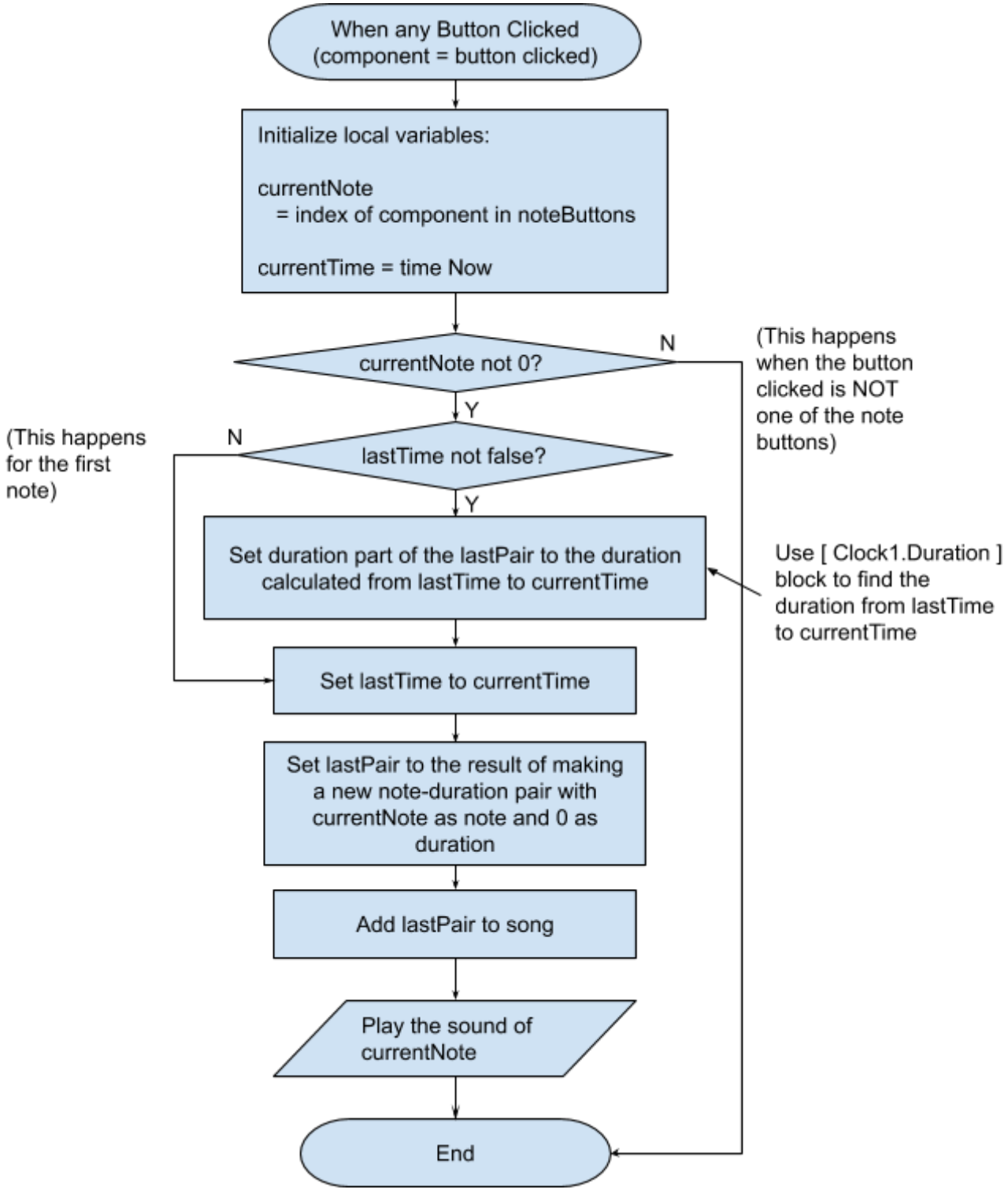
2. Then the duration value of the last stored NDpair is updated with the value of the difference between the current clock-time and the clock-time of the last note clicked (if any)

The following variables would be required:

global variable	**purpose**	**Initial value**
lastPair	hold the last stored pair	empty list
lastTime	hold clock-time of the last note clicked	false
song	Store the NDpairs in sequence of the notes clicked	empty list

(Note that you can still modify the last stored NDpair through the variable "lastPair" even after it has already been added to the "song" list)

The following flowchart for the generic Button.Click handler is given for reference:



Algorithm to playback with proper delay for each note-duration pair in song list

The playback mechanism is the same as the one shown in the book, except that we are not using the parallel arrays “notes” and “times”.

The notes and duration are instead extracted from each NDpairs stored in the “song” list.

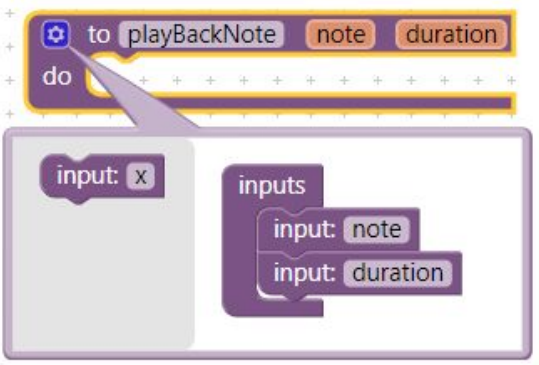
Make the following modifications to the [to playBackNote] procedure:

1. Add a parameter, *note* and *duration*, to the playBackNote procedure

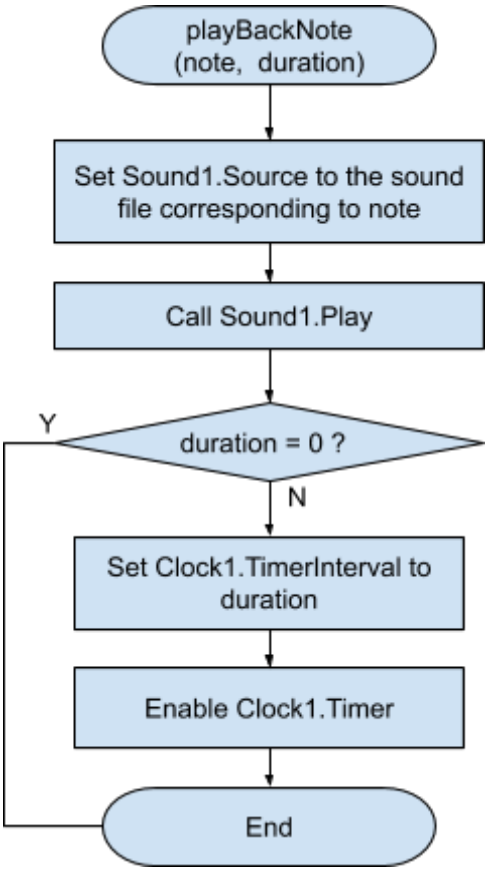
2. Simplify the playBackNote procedure to just

- Play the note on Sound1 (setting its source to the appropriate sound file, and call Sound1.Play)

- If the duration is not zero (i.e. there are more notes to play later), then set the TimerInterval to the duration and enable the timer.

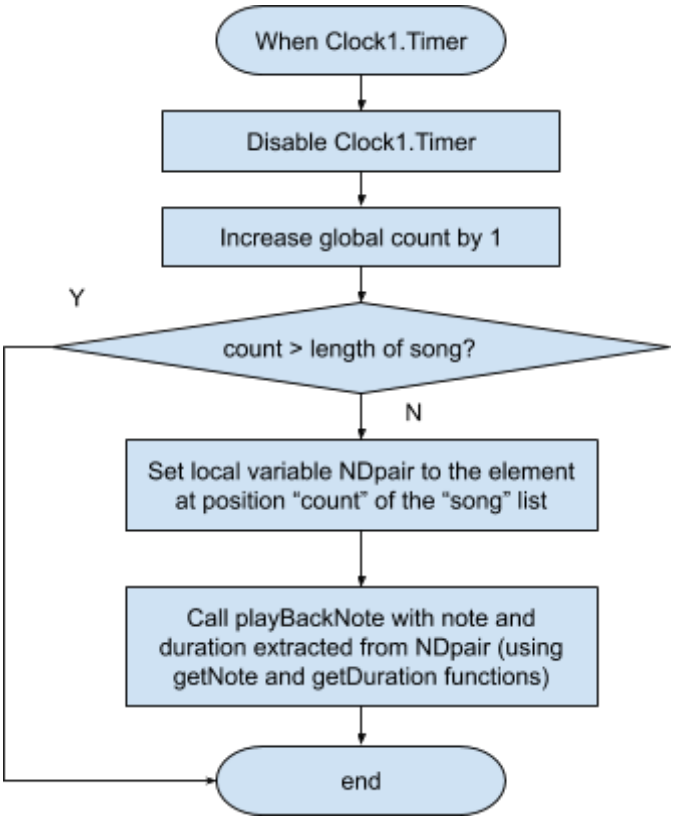


Flowchart for playBackNote



The job of increasing the global count and extracting the note and duration from the NDpairs are off-loaded to the [when Clock1.Timer] event handler.

Flowchart for [when Clock1.Timer] event handler



Actions of the playback button

In the event handler of the playback button, just set global count to 0, set the Clock1.TimerInterval to 0 and enable the timer to invoke the handler to play the song.

Preliminary tasks during lesson:

Make the following functions and procedures for the data structure of note-duration pair

1. Constructor - make the note-duration pair of the given note and duration

2. Getters - get the component values from the pair (to be used later in the playback part)

3. Setters - change the component value of a given pair