

<b>Xylophone</b>	<b>1</b>
What You'll Build	2
What You'll Learn	3
Getting Started	3
Designing the Components	4
Creating the Keyboard	4
Creating the First Note Buttons	5
Adding the Sound Component	6
Connecting the Sounds to the Buttons	7
Telling Android to Load the Sounds	10
Implementing the Remaining Notes	11
Recording and Playing Back Notes	13
What to play ?	13
Change buttons	16
<b>Appendix 1.</b>	<b>20</b>
Variables	20
Built-in blocks	21
Procedure Blocks	22
Designer	23
Source	23

## Xylophone

*It's hard to believe that using technology to record and play back music only dates back to 1878, when Edison patented the phonograph. We've come so*



# CoderDojo Castleknock

*far since then-with music synthesizers, CDs, sampling and remixing, phones that play music, and even long-distance jamming over the Internet. In this class, you'll take part in this tradition by building a [Xylophone](#) app.*



## What You'll Build

With the app shown in Figure 1 (originally created by Liz Looney of the App Inventor team), you can:

- Play eight different notes by touching colored buttons on the screen.





*Figure 1. The Xylophone app UI*

## What You'll Learn

This tutorial covers the following concepts:

- Using a single **Sound** component to play different audio files.
- Deciding when to create a procedure.

## Getting Started





Connect to the App Inventor website and start a new project. Name it "Xylophone", and also set the screen's title to "Xylophone". Open the Blocks Editor and connect to your phone or emulator.

## Designing the Components

This app has 13 different components (8 of which compose the keyboard), listed in Table 1. Since there are so many, it would get pretty boring to create all of them before starting to write our program, so we'll break down the app into its functional parts and build them sequentially by going back and forth between the Designer and the Blocks Editor.

<i>Component Type</i>	<i>Palette Group</i>	<i>What you will name it</i>	<i>Purpose</i>
<i>Button</i>	<i>Basic</i>	<i>Button1</i>	<i>Play Low C key</i>
<i>Button</i>	<i>Basic</i>	<i>Button2</i>	<i>Play D key</i>
<i>Button</i>	<i>Basic</i>	<i>Button3</i>	<i>Play E key</i>
<i>Button</i>	<i>Basic</i>	<i>Button4</i>	<i>Play F key</i>
<i>Button</i>	<i>Basic</i>	<i>Button5</i>	<i>Play G key</i>
<i>Button</i>	<i>Basic</i>	<i>Button6</i>	<i>Play A key</i>
<i>Button</i>	<i>Basic</i>	<i>Button7</i>	<i>Play B key</i>
<i>Button</i>	<i>Basic</i>	<i>Button8</i>	<i>Play High C</i>

*Table 1. All of the components for the Xylophone app*

## Creating the Keyboard





Our user interface will include an eight-note keyboard for a pentatonic (seven-note) major scale ranging from Low C to High C. We will create this musical keyboard in this section.

## Creating the First Note Buttons

Start by creating the first two xylophone keys, which we will implement as buttons.

1. From the Basic category, drag a Button onto the screen. Leave its name as **Button1**. We want it to be a long magenta bar, like a key on a xylophone, so set its properties as follows:
  - Changing its **BackgroundColor** property to Magenta.
  - Changing its **Text** property to "C".
  - Setting its **Width** property to "Fill parent" so it goes all the way across the screen.
  - Setting its **Height** property to 40 pixels.
2. Repeat for a second **Button**, named **Button2**, placing it below **Button1**. Use **Width** and **Height** property values, but set its **BackgroundColor** property to Red and its **Text** property to "D".

(Later, we will repeat step 2 for six more note buttons.)

The view in the Component Designer should look something like Figure 9-2.



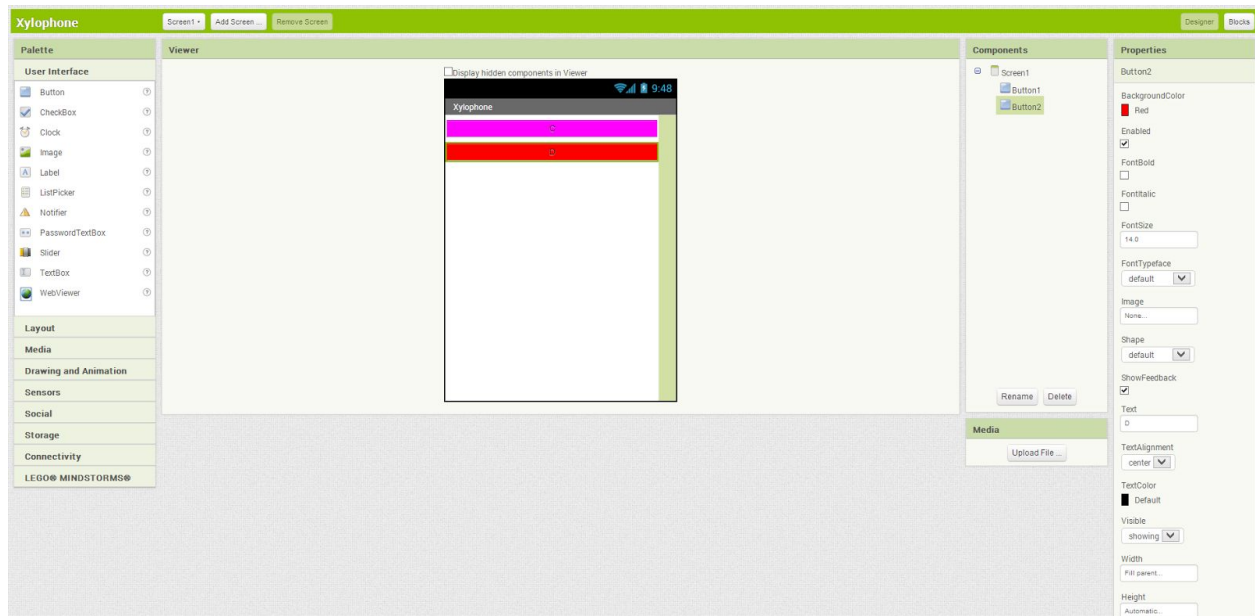


Figure 9-2. Placing buttons to create a keyboard

The display on your phone should look similar, although there will not be any empty space between the two colored buttons.

## Adding the Sound Component

We can't have a xylophone without sounds, so create a **Sound** component, leaving its name as **Sound1**. Change the **MinimumInterval** property from its default value of 500 milliseconds to 0. This allows us to play the sound as often as we want, instead of having to wait half a second (500 milliseconds) between plays. Don't set its **Source** property, which we will set in the Blocks Editor.

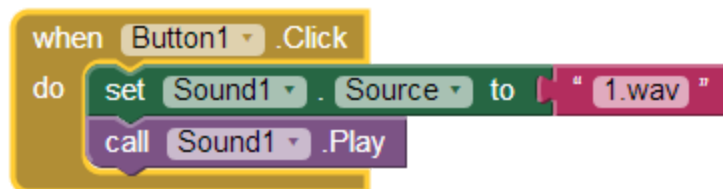
Upload the sound files [1.wav](#) and [2.wav](#) by clicking the sound link. It is important to use these exact names for reasons that will soon become clear. You can upload the remaining six sound files when directed to later.



## Connecting the Sounds to the Buttons

The behavior we need to program is for a sound file to play when the corresponding button is clicked. Specifically, if **Button1** is clicked, we'd like to play *1.wav*; if **Button2** is clicked, we'd like to play *2.wav*; and so on. We can set this up in the Blocks Editor as shown in Figure 9-3 by doing the following:

1. From the My Blocks tab and Button1 drawer, drag out the **Button1.Click** block.
2. From the Sound1 drawer, drag out the set **Sound1.Source** block, placing it in the **Button1.Click** block.
3. Type "text" to create a text block. (This is quicker than going to the Built-In tab and then the Text drawer, although that would work too.) Set its **text** value to "1.wav" and place it in the **Sound1.Source** block.
4. Add a **Sound1.Play** block.



*Figure 3. Playing a sound when a button is clicked*

We could do the same for **Button2**, as shown in Figure 4 (just changing the text value), but the code would be awfully repetitive.



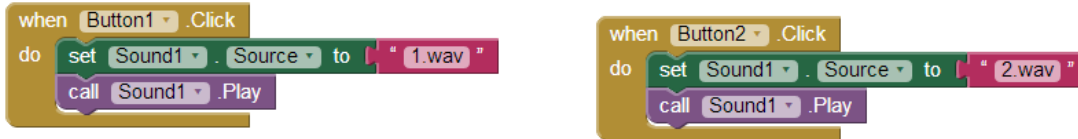


Figure 4. Adding more sounds

Repeated code is a good sign that you should create a procedure, specifically, we'll create a procedure that takes a number as an argument, sets **Sound1's Source** to the appropriate file, and plays the sound. This is another example of *refactoring*-improving a program's implementation without changing its behavior. We can use the Text drawer's **join** block to combine the number (e.g., 1) and the text ".wav" to create the proper filename (e.g., "1.wav"). Here are the steps for creating the procedure we need:

1. Under the Built-In tab, go to the Definition drawer and drag out the **to procedure** block.
2. Go back to the Definition drawer and drag a **name** block into the "arg" socket of **to procedure**.
3. Click the rightmost "name" and set the name to "number".
4. Click **procedure** and set the name to "PlayNote".
5. Drag the **Sound1.Source** block from **Button1.Click** into **PlayNote** to the right of the word "do". The **Sound1.Play** block will move with it.
6. Drag the **1.wav** block into the trash can.
7. From the Text drawer, drag the **join** block into **Sound1.Source's** socket.
8. Type "number" and move it to the left socket of the **join** block (if it is not already there).





# CoderDojo Castleknock

9. From the Text drawer, drag the **text** block into the right socket of the **join** block.
10. Change the text value to ".wav". (Remember not to type the quotation marks.)
11. Under the My Blocks tab, go to the My Definitions drawer and drag a **call PlayNote** block into the empty body of **Button1.Click**.
12. Type "1" and put it in the "number" socket.

Now, when **Button1** is clicked, the procedure **PlayNote** will be called, with its number argument having the value 1. It should set **Sound1.Source** to "1.wav" and play the sound.

Create a similar **Button2.Click** block with a call to **PlayNote** with an argument of 2. (You can copy the existing **PlayNote** block and move it into the body of **Button2.Click**, making sure to change the argument.) Your program should look like Figure 5.

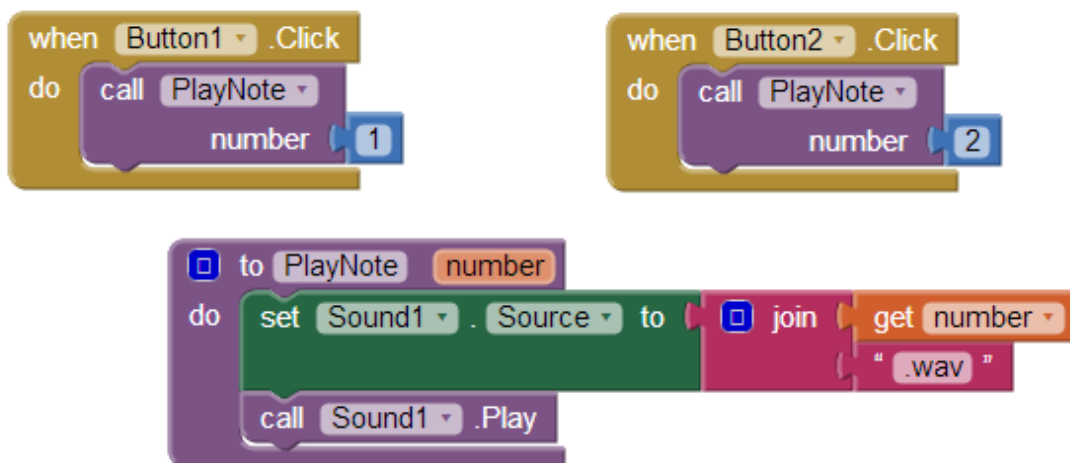


Figure 5. Creating a procedure to play a note



## Telling Android to Load the Sounds

If you tried out the preceding calls to **PlayNote**, you may have been disappointed by not hearing the sound you expected or by experiencing an unexpected delay. That's because Android needs to load sounds at runtime, which takes time, before they can be played. This issue didn't come up before, because filenames placed in a **Sound** component's **Source** property in the Designer are automatically loaded when the program starts. Since we don't set **Sound1.Source** until *after* the program has started, that initialization process does not take place. We have to explicitly load the sounds when the program starts up, as shown in Figure 6.

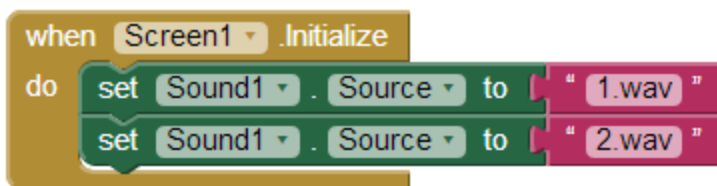


Figure 6. Loading sounds when the app launches



**Test your app.** Now if you restart the app by clicking on "Connect to Device..." in the Blocks Editor, the notes should play without delay. (If you don't hear anything, make sure that the media volume on your phone is not set to mute.)



## Implementing the Remaining Notes

Now that we have the first two buttons and notes implemented and working, add the remaining six notes by going back to the Designer and uploading the sound files [3.wav](#), [4.wav](#), [5.wav](#), [6.wav](#), [7.wav](#), and [8.wav](#). Then create six new buttons, following the same steps as you did before but setting their **Text** and **BackgroundColor** properties as follows:

- Button3 ("E", Pink)
- Button4 ("F", Orange)
- Button5 ("G", Yellow)
- Button6 ("A", Green)
- Button7 ("B", Cyan)
- Button8 ("C", Blue)

You may also want to change **Button8's TextColor** property to White, as shown in Figure 7, so it is more legible.

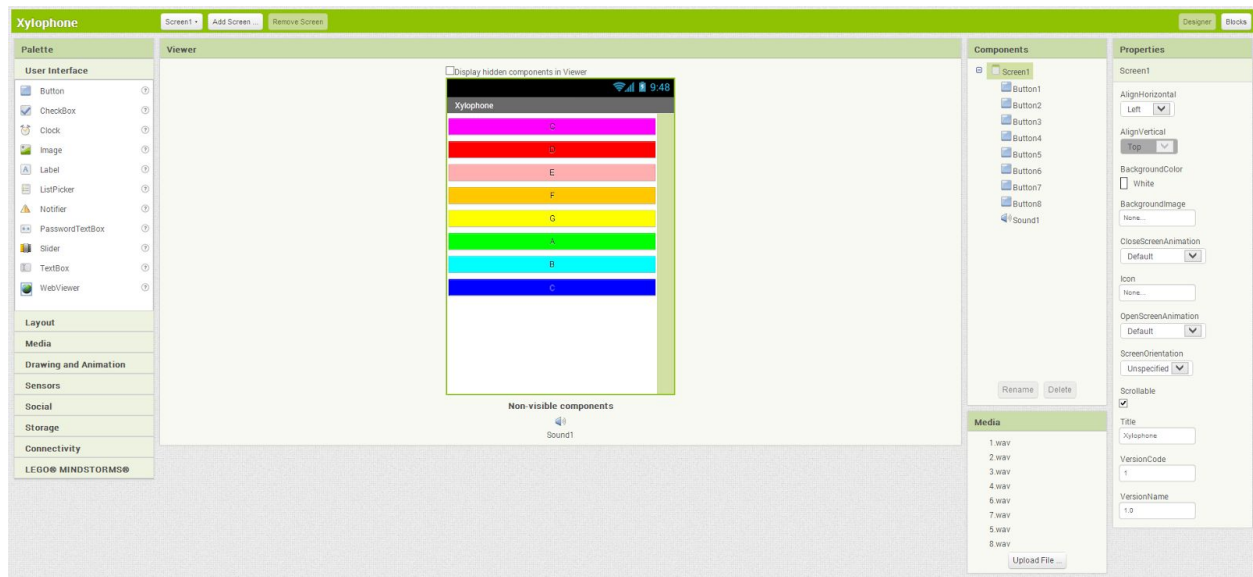


Figure 7. Putting the remaining buttons and sounds in the Component Designer



# CoderDojo Castleknock

Back in the Blocks Editor, create **Click** blocks for each of the new buttons with appropriate calls to **PlayNote**. Similarly, add each new sound file to **Screen.Initialize**, as shown in Figure 8.

With your program getting so large, you might find it helpful to click the white minus signs near the bottom of the "container" blocks, such as **PlayNote**, to minimize them and conserve screen space.



**Test your app.** You should now have all the buttons, and each one will play a different note when you click it.

```
when Button1 .Click
do
  call PlayNote
  number 1

when Button2 .Click
do
  call PlayNote
  number 2

when Button3 .Click
do
  call PlayNote
  number 3

when Button4 .Click
do
  call PlayNote
  number 4

when Button5 .Click
do
  call PlayNote
  number 5

when Button6 .Click
do
  call PlayNote
  number 6

when Button7 .Click
do
  call PlayNote
  number 7

when Button8 .Click
do
  call PlayNote
  number 8

when Screen1 .Initialize
do
  set Sound1 . Source to "1.wav"
  set Sound1 . Source to "2.wav"
  set Sound1 . Source to "3.wav"
  set Sound1 . Source to "4.wav"
  set Sound1 . Source to "5.wav"
  set Sound1 . Source to "6.wav"
  set Sound1 . Source to "7.wav"
  set Sound1 . Source to "8.wav"

to PlayNote number
do
  set Sound1 . Source to join get number
  ".wav"
  call Sound1 .Play
```

Figure 8. Programming the button click events to correspond to all the keyboard keys



## Recording and Playing Back Notes

Playing notes by pressing buttons is fun, but being able to record and play back songs is even better. To implement playback, we will need to maintain a record of played notes. In addition to remembering the pitches (sound files) that were played, we must also record the amount of time between notes, or we won't be able to distinguish between two notes played in quick succession and two played with a 10-second silence between them.

We will not cover this here, but we use lists to manage this. If you want to look at this, see the tutorial <http://www.appinventor.org/Xylophone2>.

## What to play ?

Eric Morecambe was a comedian, who said [Here](#).

“I'm playing all the right notes, but not necessarily in the right order. “

Eric Morecambe

This is actually quite important. As having a musical instrument but not knowing what to play is not great fun.

Music is a small bit like coding in that a musical instrument is like a computer, it can do lots of things, but you need to tell it exactly what to do.

In Coding we create programs to tell the computer what to do.

In Music, we use sheet music to know what notes to play, like this



## Jingle Bells

Arranged by John Eidsvoog

Crisp gallop ♩ = 280

Figure 9. Jingle Bells - musical composition

But for our xylophone app we have a easier way to know what to play. We have Letters for all the notes, like;



# CoderDojo Castleknock

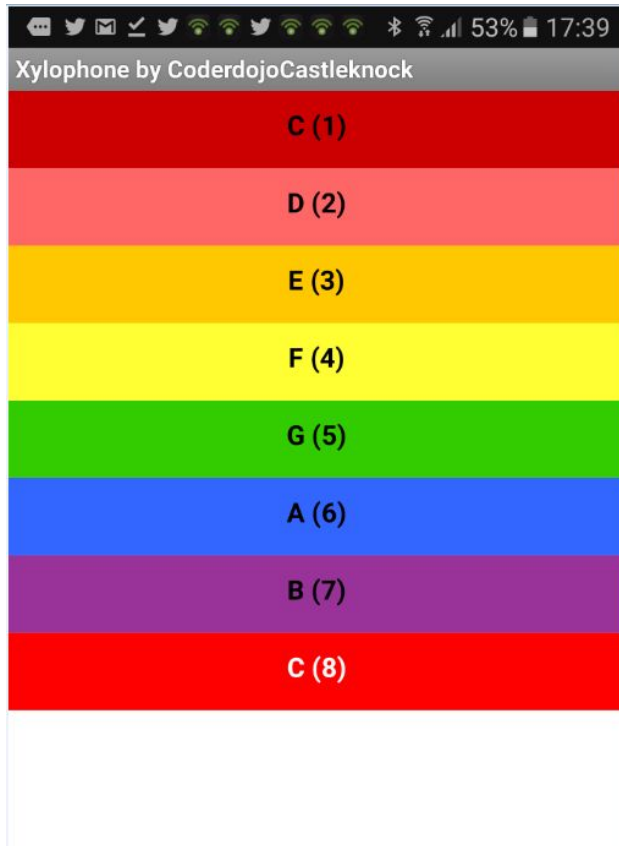


Figure 10. Final App

So the following says what notes to press like



# CoderDojo Castleknock



## Jingle Bells

J.S. Pierpont

Jin - gle bells, Jin - gle bells, Jin - gle all the way,  
 3 3 3 3 3 3 3 3 5 1 2 3

Oh, what fun it is to ride in a one - horse o - pen sleigh!  
 4 4 4 4 4 3 3 3 3 3 2 2 3 2 5

Jin - gle bells, Jin - gle bells, Jin - gle all the way,  
 3 3 3 3 3 3 3 5 1 2 3

Oh, what fun it is to ride in a one - horse o - pen sleigh!  
 4 4 4 4 4 3 3 3 3 5 5 4 2 1

Figure 11. Jingle Bells - Xylophone composition

### Change buttons

In order to use the scheme above, we need to change the button names and colors, so;

In the blocks we add colors for the buttons in the blocks screen;





# CoderDojo Castleknock

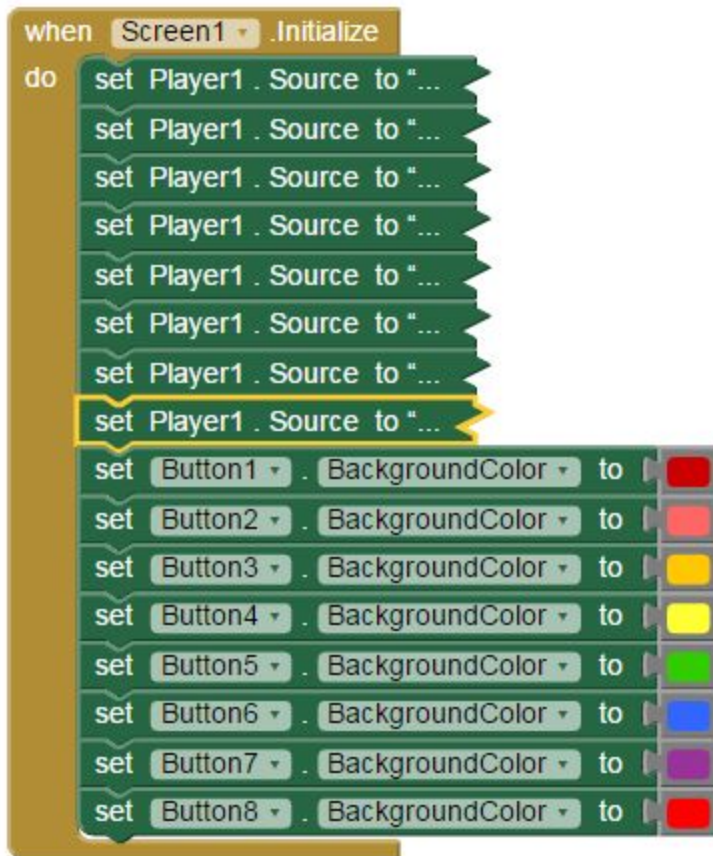


Figure 12. Change colours

And change the Button labels like below in the designer screen;



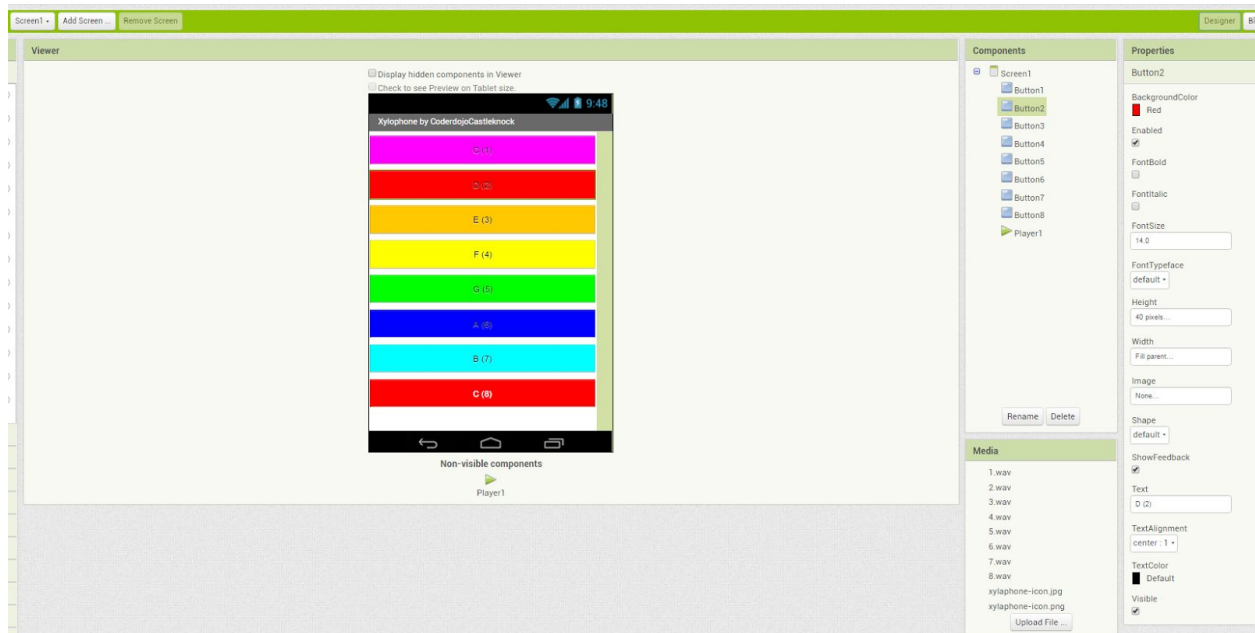


Figure 12. Change labels.

A list of songs we can play are;

- [Camptown Races](#)
- [He's Got The Whole World In His Hands](#)
- [Oh Susannah](#)
- [Up On The Housetop](#)
- [When The Saints Go Marching In](#)
- [All Things Bright And Beautiful](#)
- [Arirang](#)
- [Au Clair De La Lune](#)
- [Baby Bumble Bee](#)
- [Barnyard Song](#)
- [Buffalo Gals](#)
- [Come Dance With Me](#)
- [Debka Hora](#)
- [Drink To Me Only With Thine Eyes](#)
- [Hear Me Play My Chimalong](#)
- [Home, Sweet Home \(longer version\)](#)
- [I'm A Nut](#)
- [Jesu, Joy of Man's Desiring](#)



The logo features a black and white yin-yang symbol with a small white square in the top-left and bottom-right quadrants. To the right of the symbol, the text "CoderDojo" is written in a bold, black, sans-serif font, and "Castleknock" is written below it in a larger, bold, black, sans-serif font.

# CoderDojo Castleknock

- [Jingle Bells](#)
- [Kum Bachur Atzel](#)
- [Little Jack Horner](#)
- [Lullaby](#)
- [Mary Had A Little Lamb](#)
- [Michael, Row The Boat Ashore](#)
- [My Hat](#)
- [Night Herding Song](#)
- [Ninety-Nine Bottles](#)
- [Nobody Knows The Trouble I See](#)
- [Old Brass Wagon](#)
- [Old MacDonald Had A Farm](#)
- [O L'il 'Liza Jane](#)
- [Once](#)
- [Over The River And Through The Woods](#)
- [Reveille](#)
- [Ring Around The Rosie](#)
- [Row, Row, Row Your Boat](#)
- [Short'nin' Bread](#)
- [Skye Boat Song](#)
- [Sweet Betsy From Pike](#)
- [The Bear Went Over The Mountain](#)
- [The Farmer's Boy](#)
- [The Muffin Man](#)
- [The Mulberry Bush](#)
- [The Orchestra](#)
- [This Old Man](#)
- [Three Blind Mice](#)
- [Tinga Layo](#)
- [Twinkle, Twinkle Little Star](#)
- [What Did Delaware?](#)





Ps

We created this tutorial standing on the shoulders of giants. We want to acknowledge the original app created by Liz Looney of the App Inventor team and the book <http://www.appinventor.org/book2>

## Appendix 1.

### Variables

- notes
- count



# CoderDojo Castleknock

initialize global notes to create empty list

initialize global count to 1

## Built-in blocks

- When Screen1 Initialize
- When Player1.Completed
- When Button1.Click
- When Button2.Click
- When Button3.Click
- When Button4.Click
- When Button5.Click
- When Button6.Click
- When Button7.Click
- When Button8Click

The image displays a collection of Scratch code blocks organized into two main sections. The top section contains eight 'when clicked' blocks for buttons 1 through 8, each calling a 'PlayNote' block with a corresponding number. The bottom section contains a 'when Screen1 Initialize' block and a 'when Player1.Completed' block. The 'when Screen1 Initialize' block sets the source of 'Player1' to eight different WAV files. The 'when Player1.Completed' block calls a 'highlightnote' block, which then uses a series of blocks to retrieve the current note from a global list and update its bold state.

```
when Screen1 Initialize
do
  set Player1 Source to 1.wav
  set Player1 Source to 2.wav
  set Player1 Source to 3.wav
  set Player1 Source to 4.wav
  set Player1 Source to 5.wav
  set Player1 Source to 6.wav
  set Player1 Source to 7.wav
  set Player1 Source to 8.wav

when Player1 Completed
do
  call highlightnote
  note index in list thing 1
  list split text
  select list item list index
  get global notes length of list list
  get global notes
  boldstate false
```



## Procedure Blocks

- To PlayNote number
- To highlightnote note boldstate

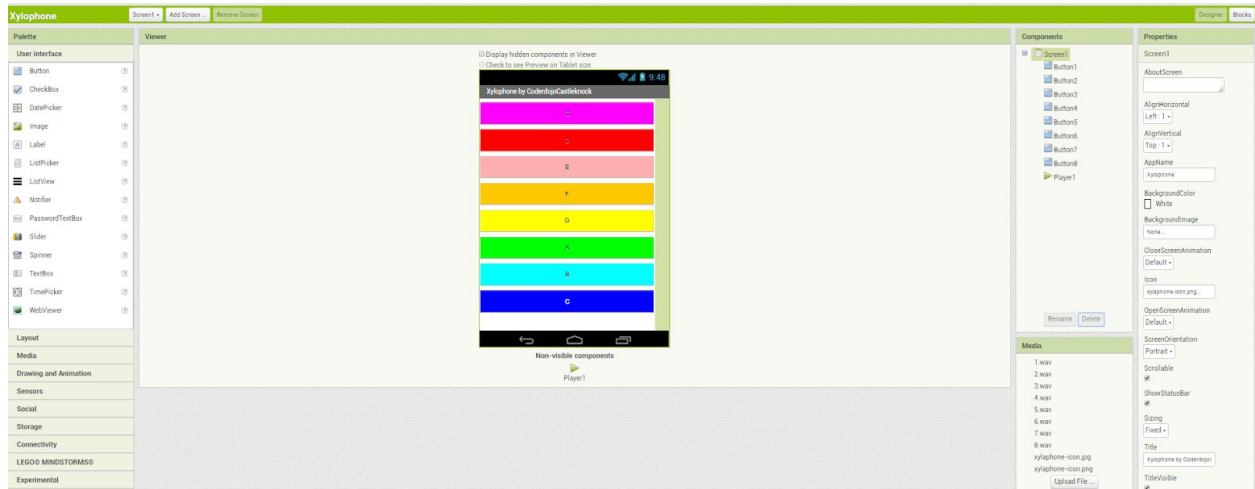
```
to PlayNote number
do
  call highlightnote
  note get number
  boldstate true
  set Player1 . Source to join get number .wav
  add items to list list get global notes
  call Player1 .Start
```

```
to highlightnote note boldstate
do
  if get note = 1
  then set Button1 . FontBold to get boldstate
  if get note = 2
  then set Button2 . FontBold to get boldstate
  if get note = 3
  then set Button3 . FontBold to get boldstate
  if get note = 4
  then set Button4 . FontBold to get boldstate
  if get note = 5
  then set Button5 . FontBold to get boldstate
  if get note = 6
  then set Button6 . FontBold to get boldstate
  if get note = 7
  then set Button7 . FontBold to get boldstate
  if get note = 8
  then set Button8 . FontBold to get boldstate
```



# CoderDojo Castleknock

## Designer



## Source

[zylophone](#)

