In response to interest in App Inventor from Kentucky's Students Technology Leadership Program, the Kentucky Academy of Technology Education (KATE) hosted two, two day, App Inventor trainings. To demonstrate a range of App Inventor's features in a manner inclusive to teachers of a variety of subjects and grade levels, I selected two topics that arise in all classrooms: classroom management and vocabulary building.

On day one a slight tweak to the Hello Purr app, commonly used as an introduction to App Inventor, enabled educators to build a basic classroom management application within an hour. From there participants were introduced to several common CS concepts and App Inventor features to enhance their classroom management applications.

The following day educators built an application that helps students keep track of new vocabulary words and definitions. Participants learned how to store and retrieve data with a TinyDB component, as well as how an activity starter can link an app directly to a website, such as an online dictionary.

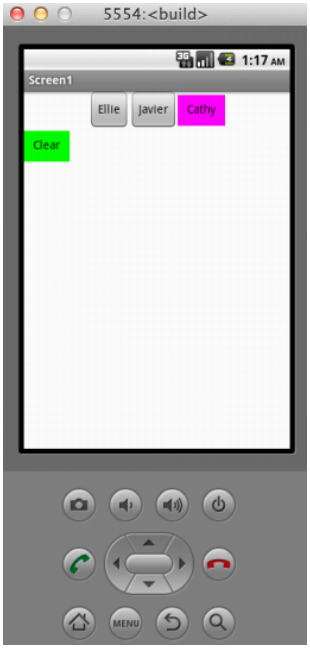
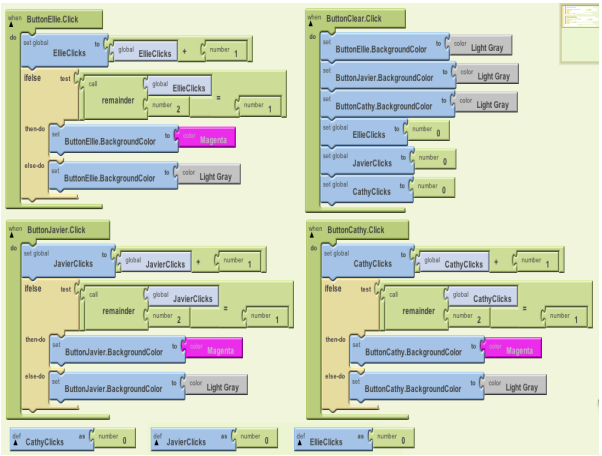
Participants left the conference with at least three functional apps for their Android phones. Perhaps more importantly, they returned to their sites with ideas on how to utilize App Inventor to engage students and assist teachers.

Introductory Classroom Manager Abstract

Teachers frequently find themselves walking around the classroom with a clipboard and class roster to note down the behavior and progress of their students. To facilitate this process, we will build an application that acts as a digital class roster for the teacher's phone. As a teacher circles the room, he/she can tap on a student's name to indicate that student deserves an extra point, is off task, arrived tardy, or any variety of other behaviors. This application is a twist on the commonly used introductory app, Hello Purr; instead of clicking a button to hear a sound, the teacher clicks on a button to change its background color. To increase the functionality of the app we continue on to create a toggle feature for each button, allowing a teacher to mark a student and then clear the mark if necessary.

<u>Main Concepts Covered:</u>

- Variables
- Branching
- Toggling
- Horizontal Arrangements
- Centering
- Programmatically change background color

Emulator	Block Editor
	

Instructions

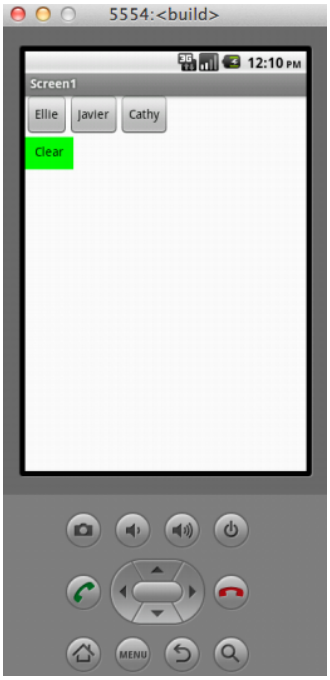
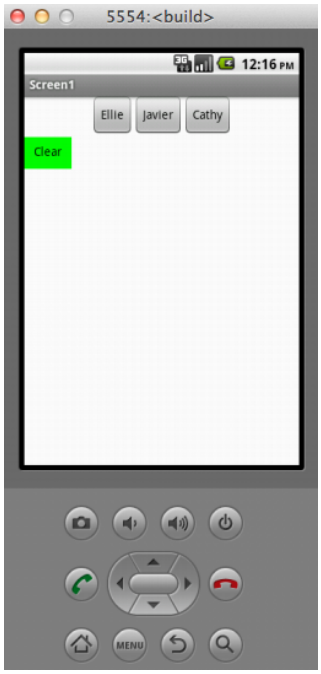
We begin by creating an app that has four buttons, one for each of three students and one button to clear the screen. When we click on any one of the three student buttons, the background color of that button will turn magenta. When we click the clear button, the background color of all three student buttons will turn light gray.

To polish off the app, we add a toggle feature that allows us to click on any button multiple times, alternating the background color of that button between magenta and light gray.

In the design window, drag out the following components:

- Horizontal Arrangement
- Four buttons (three names, one clear)

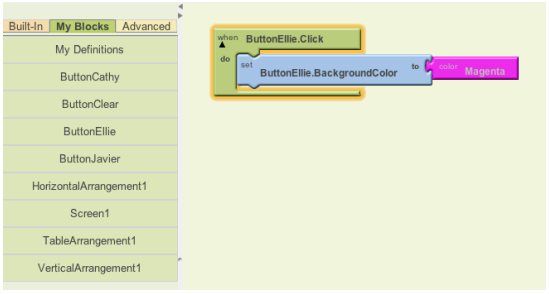
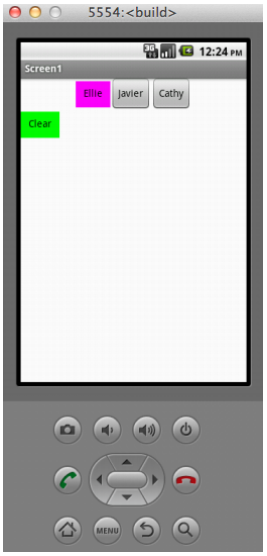
The phone will initially appear as shown in the left-hand image below. In the right-hand image we see the same features, with the name buttons centered.

Phone emulator before centering	**Phone emulator after centering**
	
The horizontal arrangement contains three buttons. The clear button is placed below the horizontal arrangement.	The horizontal arrangement contains three buttons and two vertical arrangements; one placed before the first button and another placed after the last button. Both vertical arrangements' widths and heights are set to Fill parent.

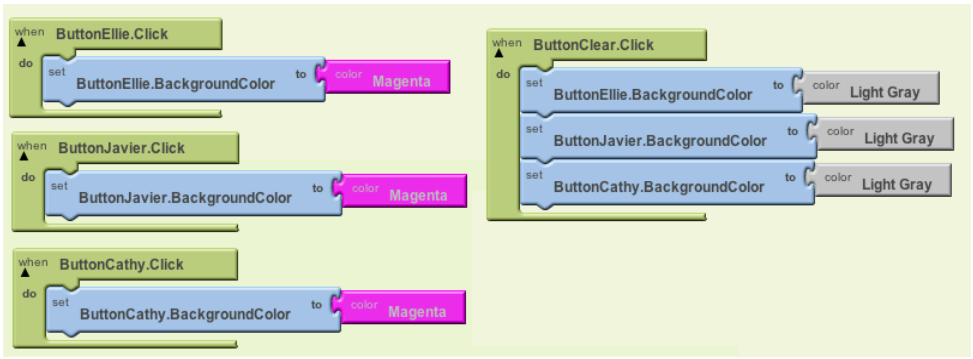
After arranging the components, open the blocks editor. We begin by programming the Ellie button. When we click on this button, we want it to turn magenta.

Next steps:

- In the blocks editor, from the **ButtonEllie** drawer, we drag out the block **when ButtonEllie.click**.
- In that same drawer we find the block **set ButtonEllie.BackgroundColor to**.
- In the **Colors** drawer (in the **Built-in** menu), drag out the **Magenta** block.
- Connect these three blocks in the block editor as shown below:

Block editor	Phone emulator
	
When the button we named **ButtonEllie** is clicked, the background color of that button will turn magenta.	Upon clicking the button **ButtonEllie**, we see that it is now magenta. This may indicate to us the Ellie was absent, tardy, deserves extra points, etc.

Repeat this process to program the remaining two student buttons so that they will also turn magenta when clicked. We follow a similar procedure to program the clear button.



Our app is now fully functional! However it is a bit limited; in order to clear one button, we must clear them all.

We would like to be able to alternate between magenta and gray on each button individually. In other words, for any odd number of clicks we want the button to turn magenta and for any even number of clicks we want the button to turn gray.

To program this feature, we use the idea that any even number divided by two has a remainder of zero, and any odd number divided by two has a remainder of one.

In the **Built-In** menu, within the **Math** drawer, we find the **Remainder** block. The top socket of the remainder block takes the number we wish to divide (the dividend) and the bottom socket takes the number we wish to divide it by (the divisor). If the dividend is odd and the divisor is two, the remainder will equal one. However if the dividend even the remainder will be zero, so long as the divisor remains two.

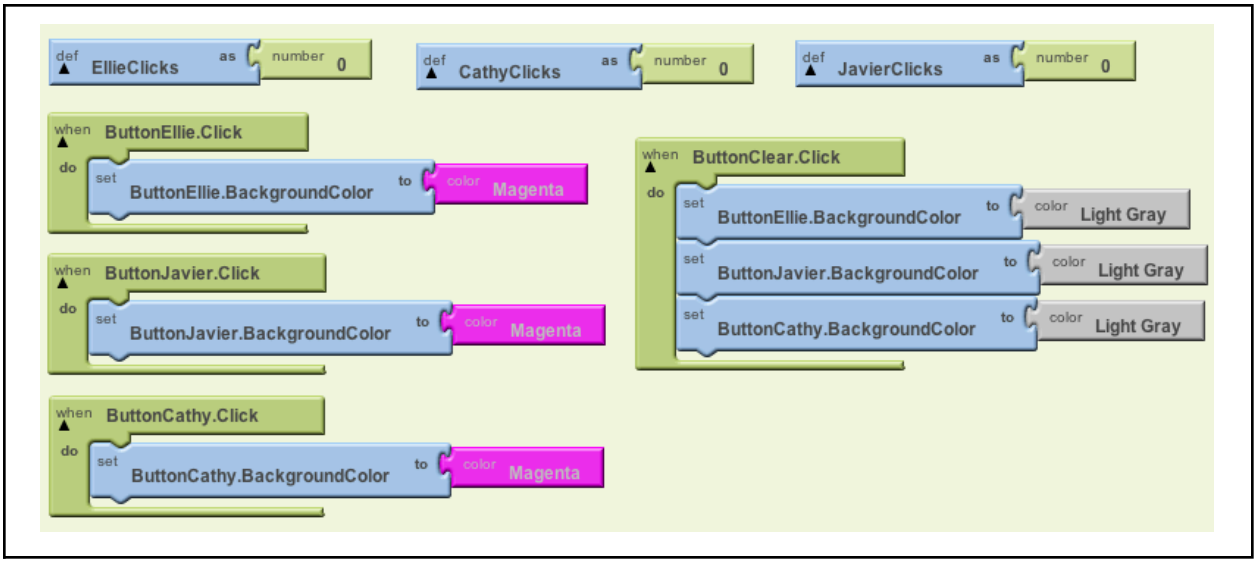
Refer to the example below:

Remainder equals one	**Remainder equals zero**
	

Note: *Within the **Math** drawer, we can also find the **Modulo** block. Modulo is similar to remainder, however it is a bit more general. If you anticipate dividing by both positive and negative numbers, it is best to use modulo instead of remainder.*

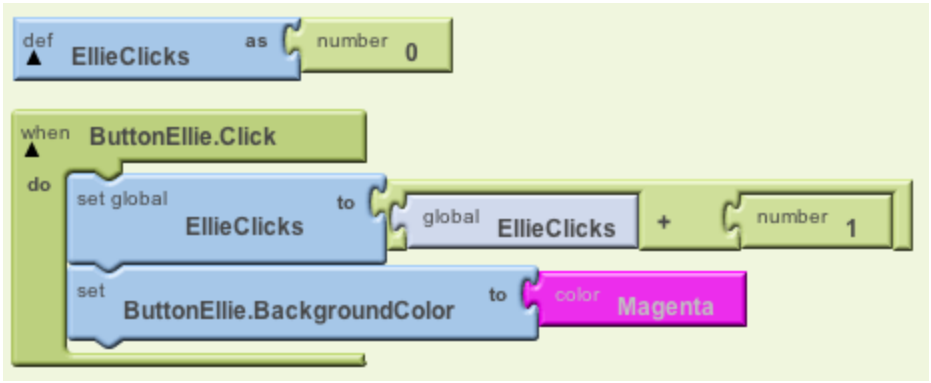
Let's start each student off with zero tardies (bonus points, absences, etc.). We can do this by dragging out one variable block for each student (**Built-In** → **Definition** → **variable**), and plugging the number zero into each variable.

These three variables will store the number of times we click each button. We renamed our variables so that the names are indicative of the values they store (**EllieClicks**, **CathyClicks**, and **JavierClicks**):



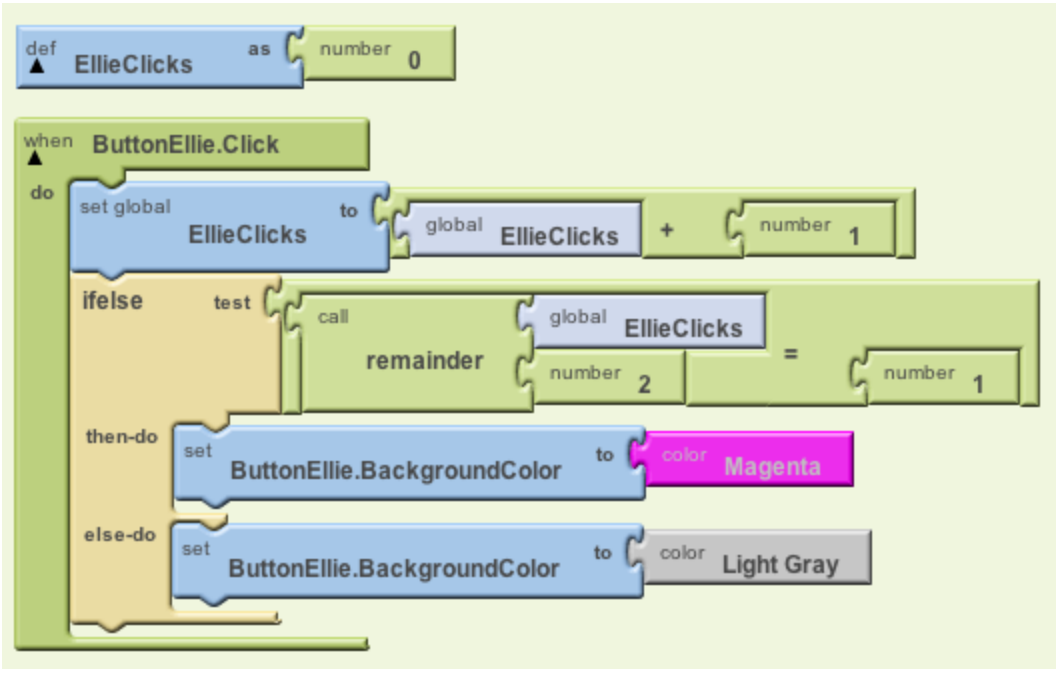
We want to increase the value of each variable by one each time the corresponding button is clicked. Using Ellie as an example, each time **ButtonEllie** is clicked we want to increase the value stored in the variable **EllieClicks** by one.

This is done as shown below:



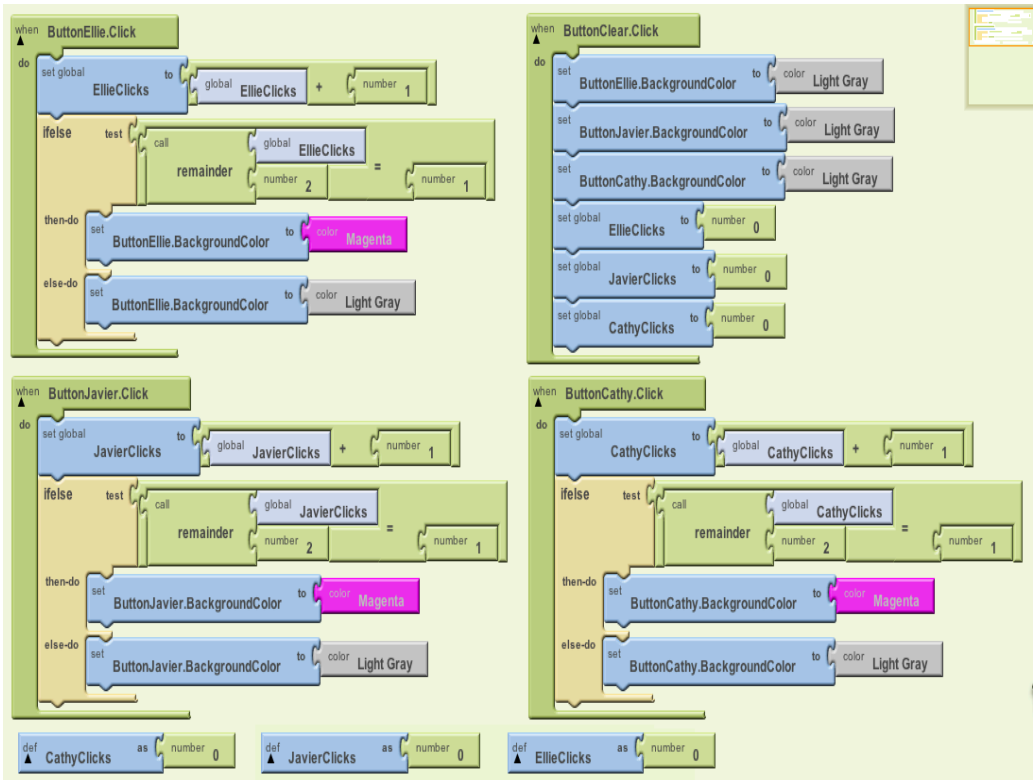
Lastly, we need to tell App Inventor that if we divide the value stored in **EllieClicks** by two and get a remainder of one, we wish to set the background color of the **ButtonEllie** to magenta. If not, we wish to set the background color to light gray.

This can be done with an **ifelse then-do else-do** block (found in the **Control** drawer), as shown below:



Repeating this process for the remaining two buttons, we end up with the following program:

Complete basic classroom manager app with toggle buttons



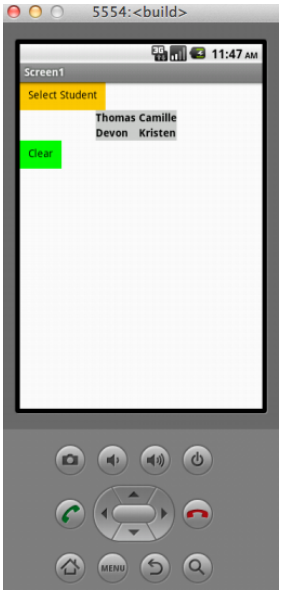
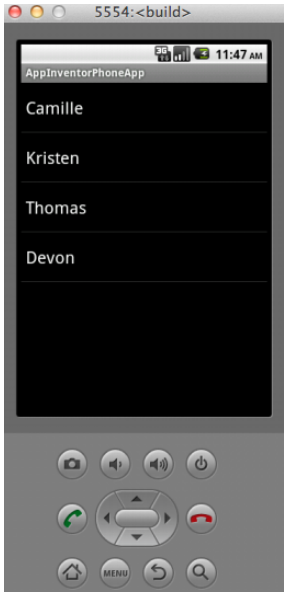
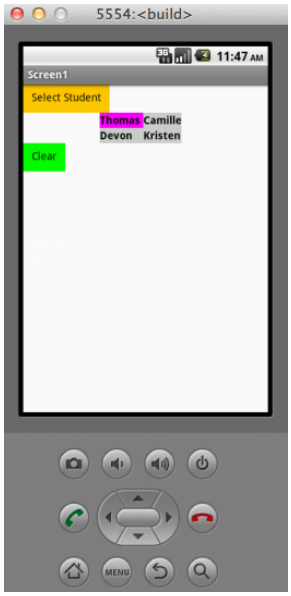
Note: *We reset the variables back to zero when the* ***ButtonClear*** *is clicked so that both the color and the number of clicks are cleared at the same time.*

Intermediate Classroom Manager Abstract

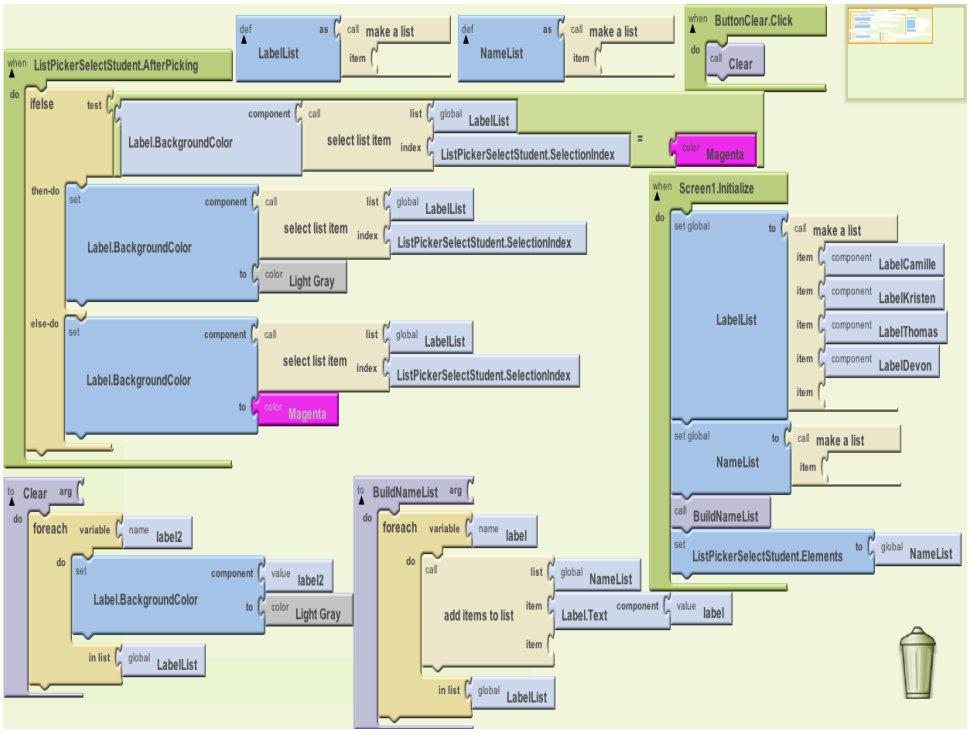
While working through the introductory classroom manager app, educators may notice that it does not scale well. Many teachers have over 100 students, and creating a new toggle button for each one of them is tedious work. To facilitate the process, we introduce the listpicker component and the concept of looping. Although the logic behind this app is more challenging than that of the introductory classroom manager, it allows teachers to add any number of students to the app with minimal extra work.

Main Concepts Covered:

- Lists and listpickers
- Initializing the screen
- Procedures
- Looping (for loop)

Step 1: Click Select Student	**Step 2:** Choose a student	**Step 3:** The background color changes
		

Block Editor



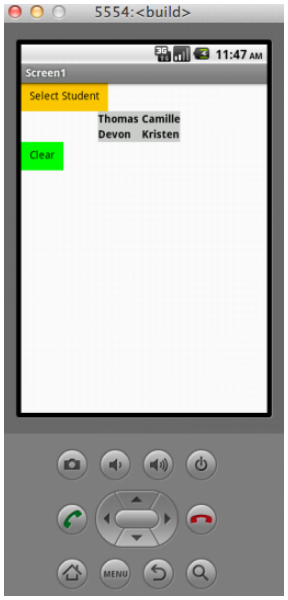
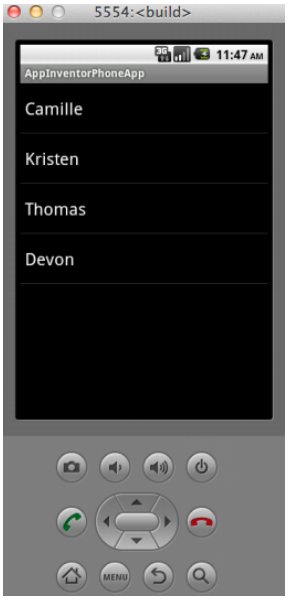
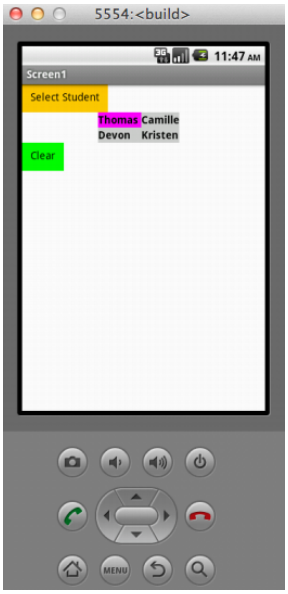
Instructions

This app assumes one has already completed the basic classroom manager app. Although the basic manager is fully functional, it does not scale well. Many teachers have over 100 students, and programming 100 + buttons as shown in the previous app would be very time consuming.

It would be nice if App Inventor allowed us to write a program along the lines of “when any button is clicked, change the color of that button to magenta”, as opposed to programming each button separately.

App Inventor does not allow us to do this directly, however we can get around that with the following solution:

We create a list of names and when we choose a name from the list, the corresponding label will turn magenta. Refer to the images below.

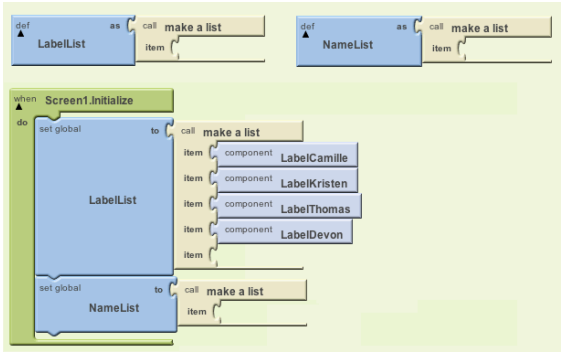
Initial screen	**Listpicker**	**After picking from the listpicker**
		
We click **Select Student**.	This list of name appears.	After we select a student from the list of names, the background of that student’s label will turn magenta. In this case we selected Thomas.

In the design window, drag out the following components:

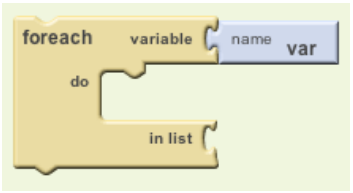
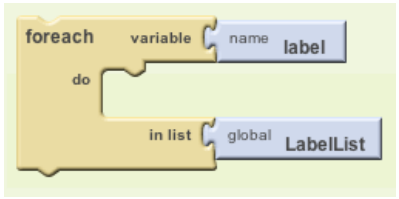
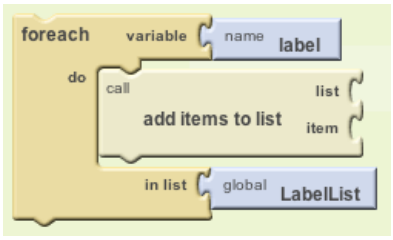
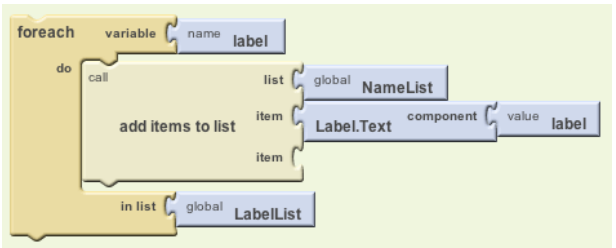
- Listpicker (**Select Student**)
- Table arrangement (to arrange the labels)
- Four labels (one for each student)
- One button (**Clear all**)

We still need to add each label individually in the design window, however we can program App Inventor to automatically take the name on each label and generate the list in the listpicker.

Refer to steps one and two below to get started:

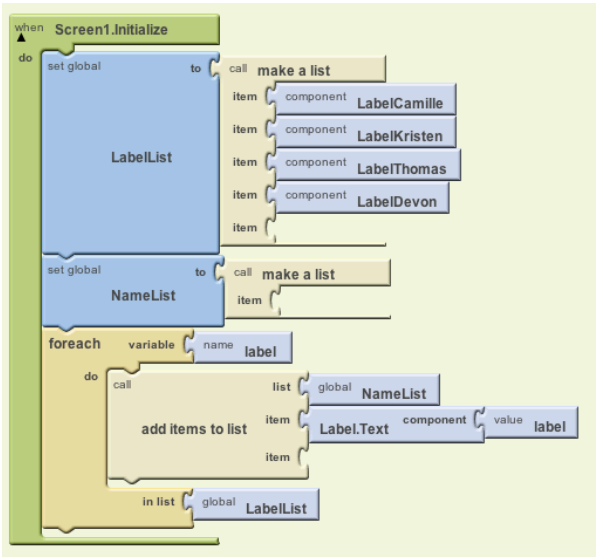
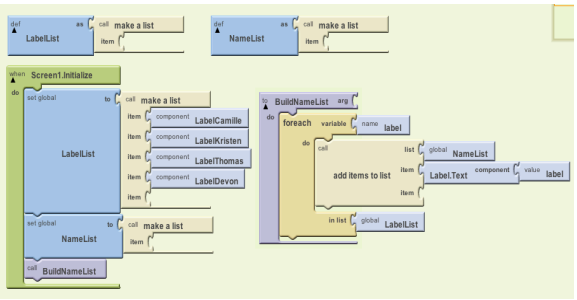
Step 1	Step 2
	
We tell App Inventor that we need it to carve out space for two lists. **LabelList** stores the names that we typed onto the labels in the design window. **NameList** will store these same names to be displayed in the listpicker.	When we open the app we tell App Inventor to fill the **LabelList** with the list of names from the labels on the design screen by plugging in the component blocks. For now, the **NameList** remains empty, but we must include it to indicate to App Inventor that we will soon be accessing it.

Next, we want App Inventor to take each name from the **LabelList** and add it to the **NameList**. Upon doing so we will be able to click **Select Student** and see a list of our students' names.

Step 1	**Step 2**
	
This block is found in the **Control** drawer. The **var** block does not appear until we drag out the **foreach** block.	We change the word **var** to a word that better describes our intention (**label**): For each **label** in the **LabelList** we want to "do" something.
Step 3	**Step 4**
	
What we wish to "do" is add the name from each label to the **NameList**.	Now App Inventor knows that for each **label** in the **LabelList**, we want to add the **text** from that label to the **NamesList**.

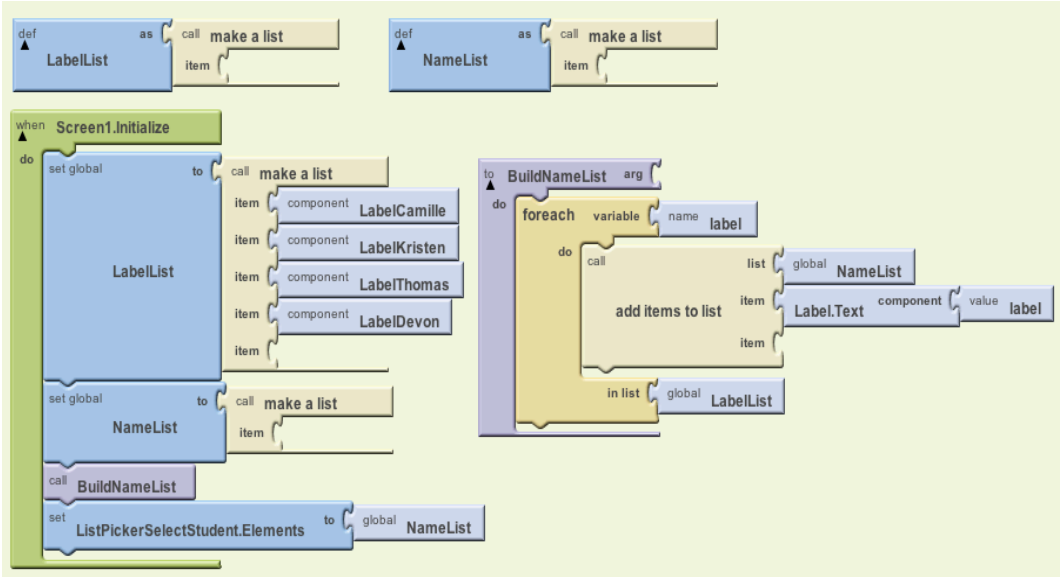
We want to generate the **NameList** as soon as we open the app, so we can drag the chunk of code in **Step 4** above into the **Screen1.Initialize** block.

Be sure to place it BELOW, the two variable blocks, otherwise App Inventor will not recognize the variable names when it goes through the for loop that we just created.

Option1: Place the for loop within the Screen1.Initialize block	**Option 2:** Make the for loop a separate procedure and call that procedure in the Screen1.Initialize block
	
When we initialize the screen (open the app), we immediately load both variables; for each label in the **LabelList** we add the text from that label to the **NameList**.	To break up these tasks and make the program more readable, we can use a **procedure** block (renamed as **BuildNameList**), and plug the “for loop” into it (**Built-In → Definition → procedure**). Within the **Screen1.initialize** block we can then simply “call” the **BuildNameList** procedure.

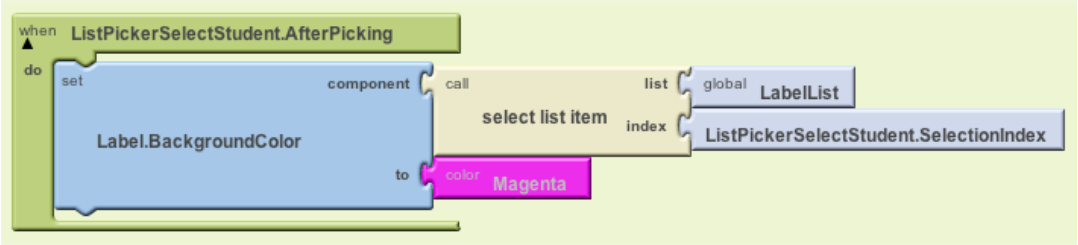
If we open the emulator and test out the above app, we see that upon clicking the **Select Student** button, we get a blank black screen; not the list of names that we just generated!

This is because we have not yet told App Inventor to *display* the names we generated in our loop. This can be done by adding the **ListPickerSelectStudent.Elements** block to the **Screen1.Initialize** block, and plugging the **global NameList** block into it as shown on the following page:



We now see our list of names when we click the **Select Student** button. However, upon selecting a name, the corresponding label does not turn magenta. We program the listpicker, as shown below, to solve this problem.

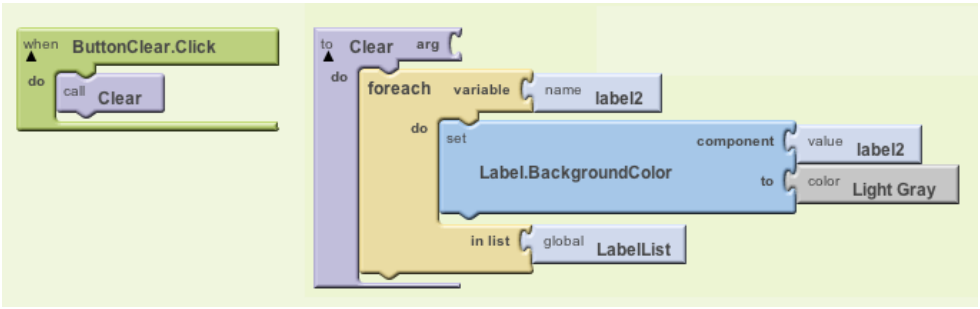
Refer to the image below:



What's left to do:

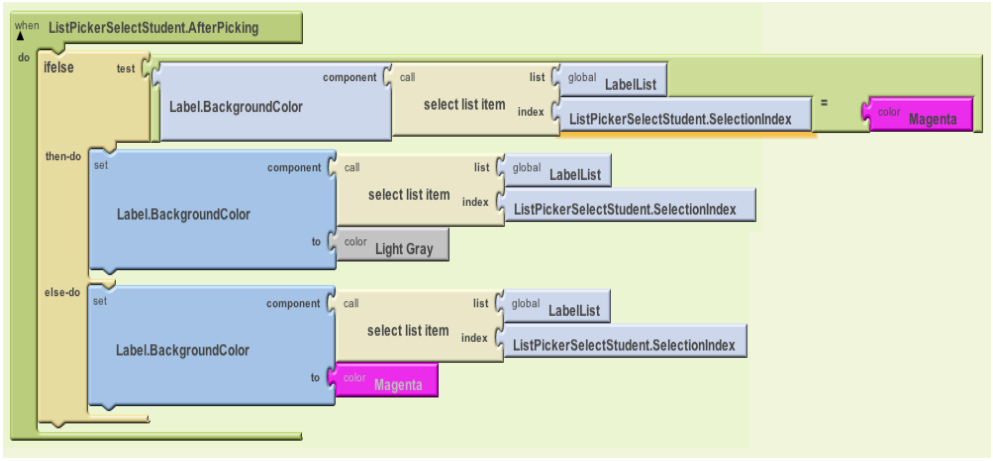
1. Program the clear button to set the background of each label gray.
2. Bonus: create a toggle feature that allows us to toggle between two background colors for each individual label.

1. We will use another loop to program the clear button. When we click the **ButtonClear**, for each label (**label2**) in the **LableList**, we want to set the background color of that label to light gray. This is done with the blocks of code as shown below:



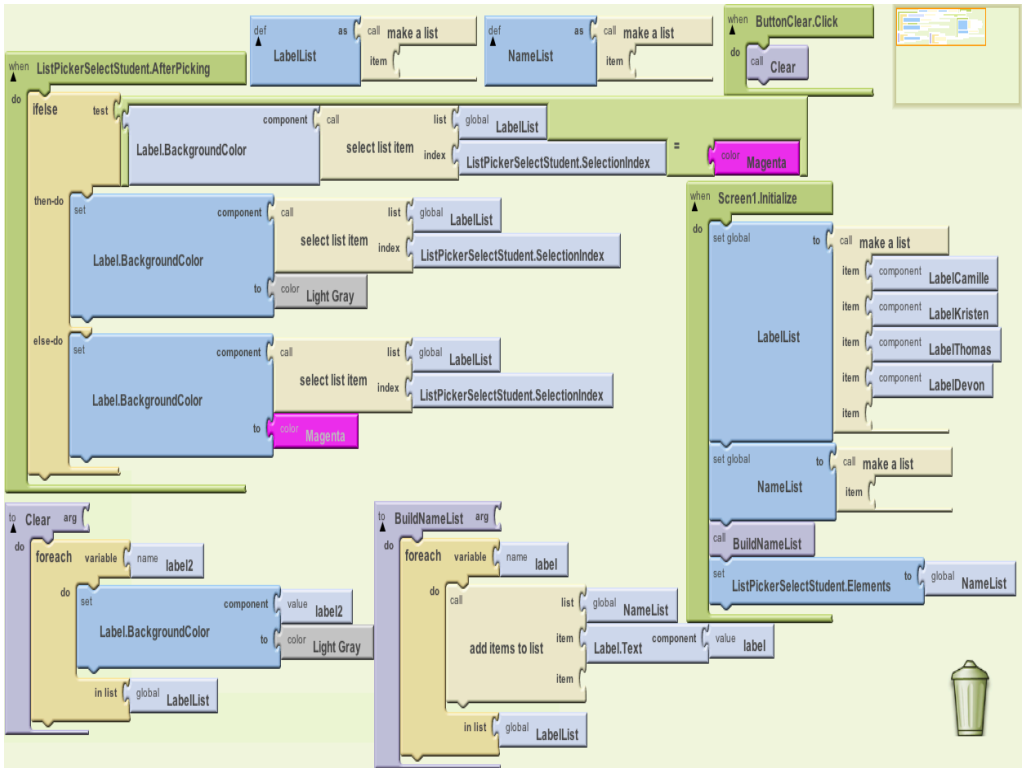
2. We can create a toggle button to clear each label individually. We add a block of code that tells App Inventor, when we select a name from the listpicker, if the corresponding label's background color is magenta, turn it light gray. Otherwise turn it magenta.

Listpicker with toggle feature



Our app is now complete. To add more students, we simply drag more label components into the design window and add the corresponding component block to the **LabelList** in the **Screen1.initialize** block. Our program does the rest for us!

Complete classroom manager app



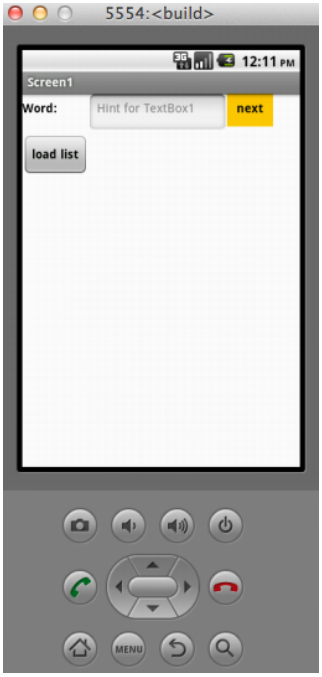
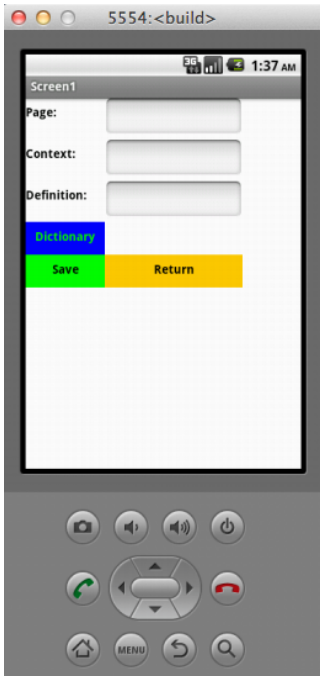
Vocabulary Tracker Abstract

Students are presented with new vocabulary words in every class. To help them keep track of these new words, we build an application that stores new words and their definitions. Students can add words to the list, delete words that they have since mastered, and link directly to an online dictionary. Additionally, students can use their mobile vocabulary lists to quiz themselves as an alternative to flashcards.

Note: *Although we centered this application around vocabulary building, the same concept may be used by a teacher to enter a student's name and behavior, or as a general to-do list.*

Main Concepts Covered

- Multiple screens (visibility settings)
- Storing and retrieving data (TinyDB)
- Lists and listpickers
- Branching
- Linking to a website

Enter a word	Enter the page number, context, and definition	The word is added to the word list
		

Instructions

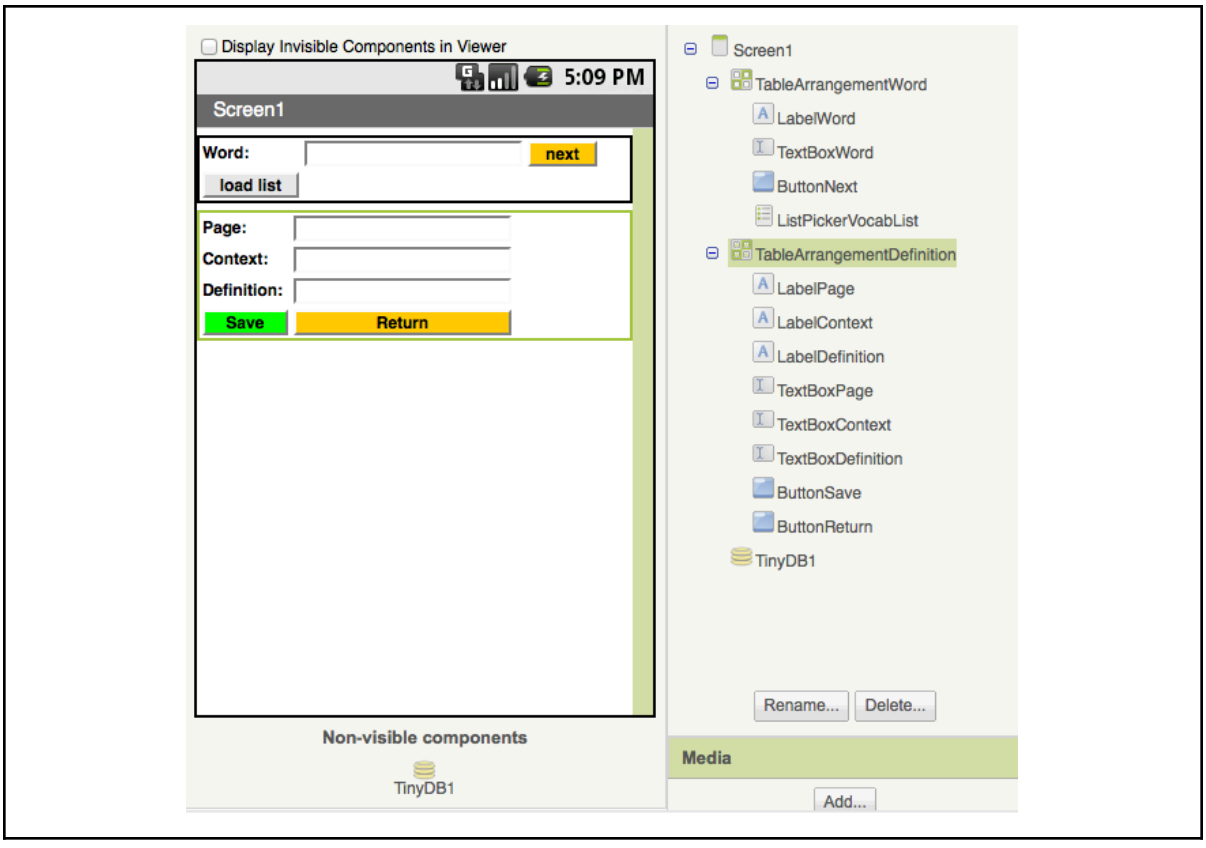
We want to create an app that allows students to build a vocabulary list on their phones.

Students will enter the following info:

- a vocabulary word
- the page they found the word on (if students are reading articles, this can be changed to the title of the article, or the paragraph that contains the word)
- the context they found the word in
- the definition of the word

The vocabulary words will be stored in variable and displayed in a listpicker. When the user selects a given vocabulary word from the list, the page number, context, and definition of that word will be displayed.

We'll design the app as shown in the image below:

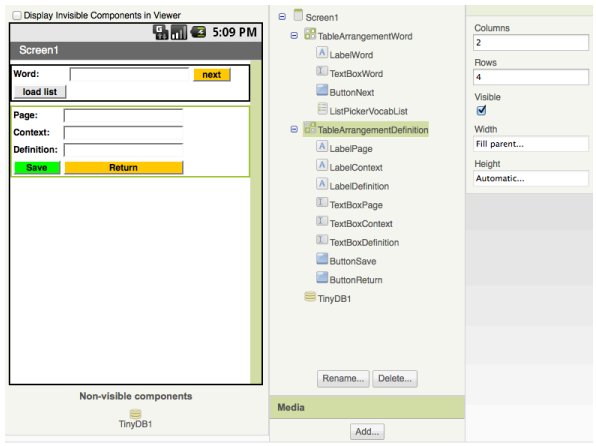
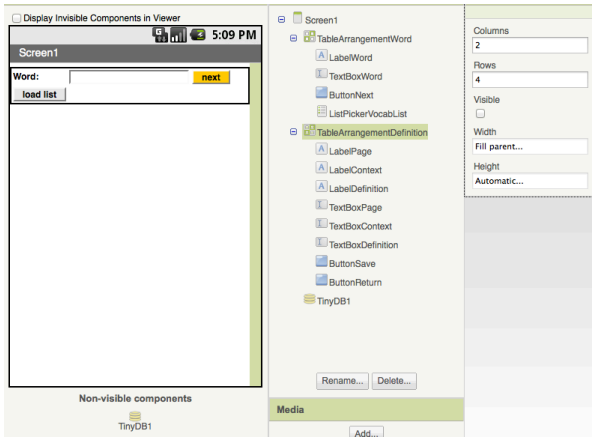


When we open the app, only the top table arrangement will be visible (**TableArrangementWord**). We will program our app such that when we click the **next** button, **TableArrangementDefinition** becomes visible, and **TableArrangementWord** is no longer seen.

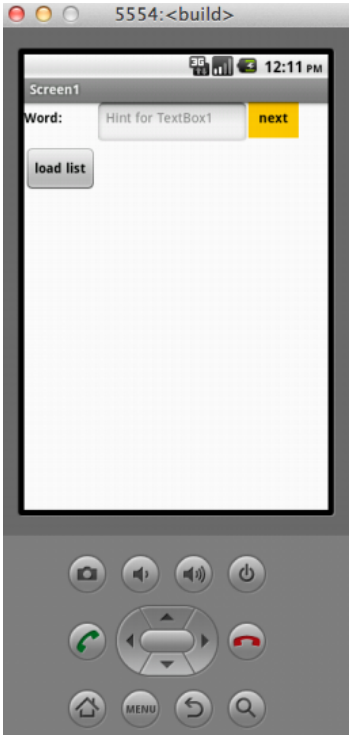
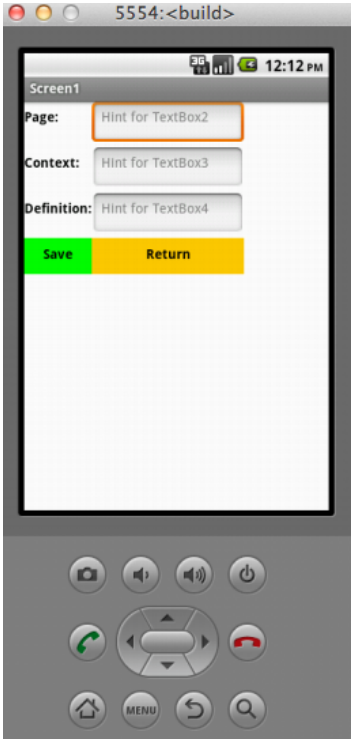
When we click the return button, **TableArrangementWord** will be visible again, and **TableArrangementDefinition** will not. This creates the effect of multiple screens.

By checking off the **Visible** box in the **Properties** column of the design window, we “hide” the second table arrangement (**TableArrangementDefinition**).

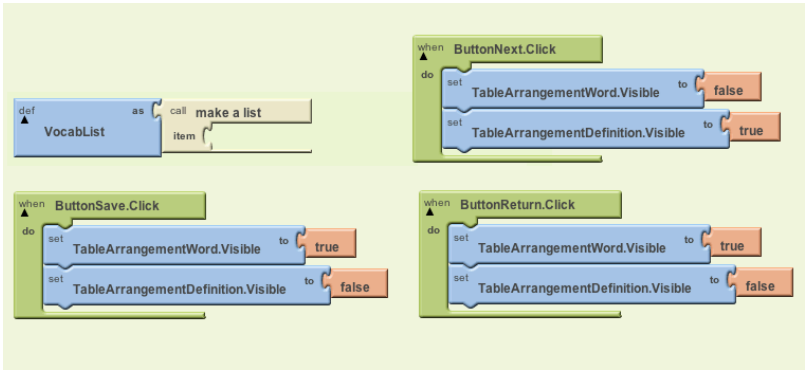
Refer to the two images below. Notice that in the properties column, the left hand image has the visibility box checked on, whereas the right hand image does not:

TableArrangementDefinition visibility on	**TabelArrangementDefinition visibility off**
	

Our goal is for the app to function as shown below:

TableArrangementWord	TableArrangementDefinition	listpicker
		
When we open the app, we only want to see **TableArrangementWord**	When we click **next**, we only want to see **TableArrangementDefinition**	When we click **load list**, we see our list of vocabulary words.

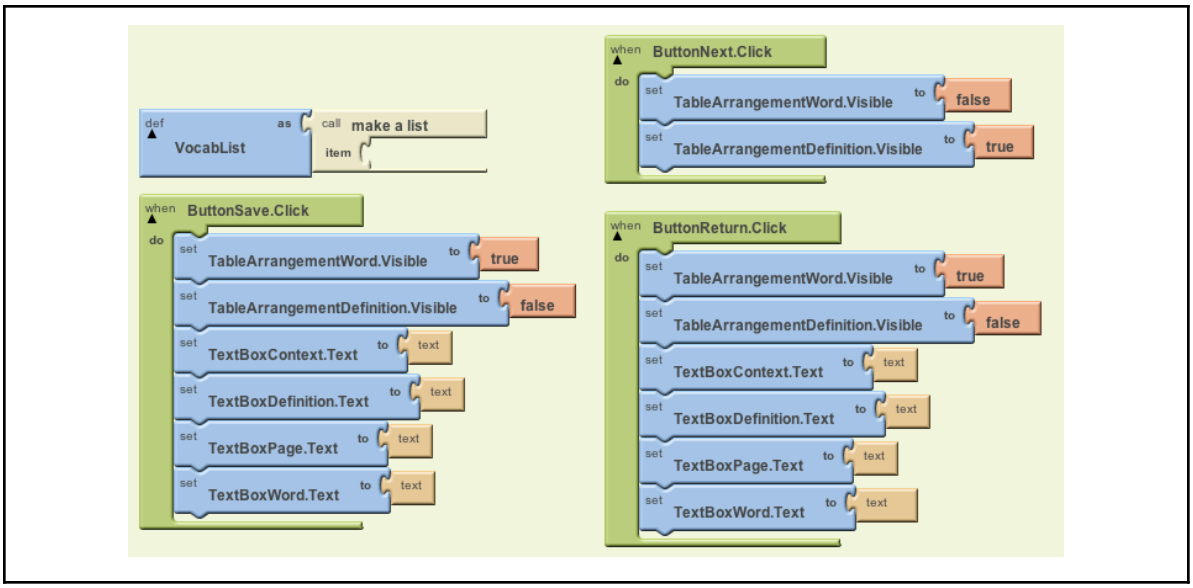
We begin by declaring the variable in which we will store our list of vocabulary words (**VocabList**), and programming App Inventor to switch between **TableArrangementWord** and **TableArrangementDefinition**, as shown below.



We now have the effect of multiple screens programmed, however our app is not very functional yet. Let's start by solving the following issues:

1. The text boxes are not cleared when we click the save button.
2. The save button is not programmed to save anything.
3. The listpicker is not programmed to display the page number, context, and definition of our word
4. We have not initialized the screen.

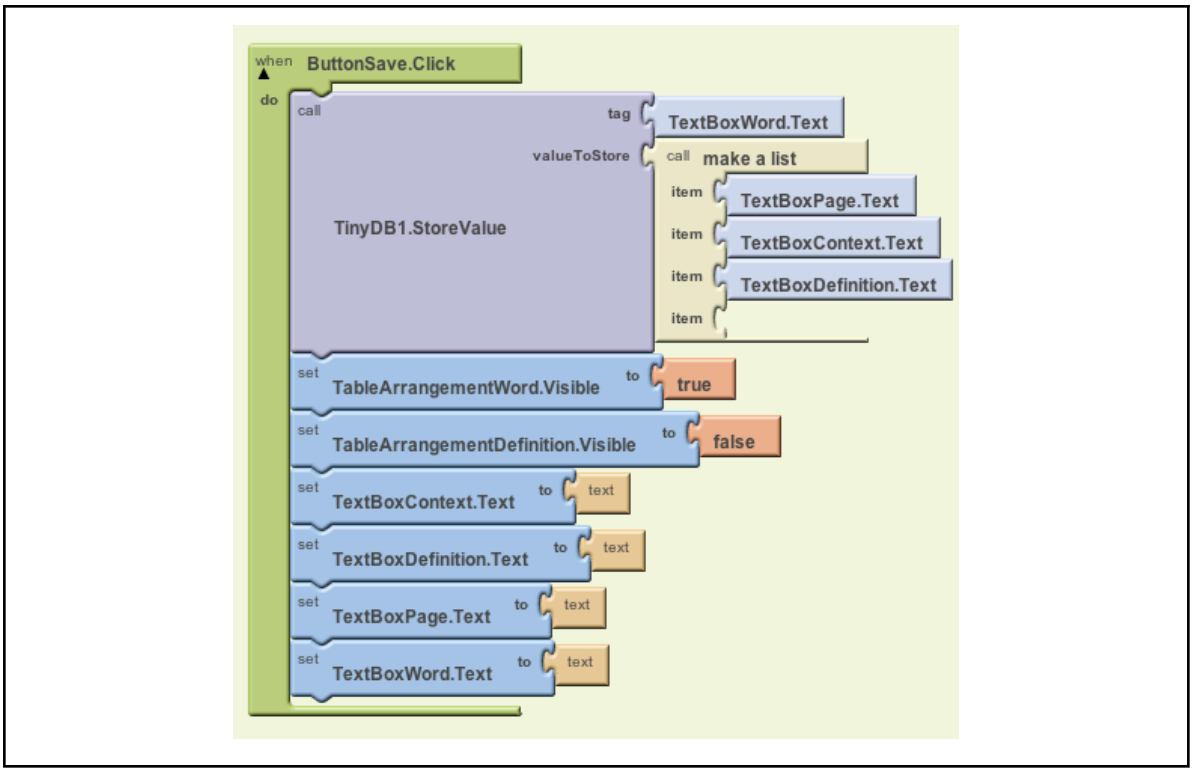
1. Clearing the text boxes - To clear a text box, simply plug a blank text box into it's **set XXX.text** block



2. Saving data - In order to save data when the phone is turned off, we need a data base. In the design window, we can click on the **Basic** drawer and drag out a **TinyDB** component. Notice this automatically snaps down to the **Non-visible components** section of screen.

In the blocks editor, open the **TinyDB1** drawer, and drag out the **TinyDB1.StoreValue** block. We want to store three values: the page number, the context, and the definition. As such, drag out a **make a list** block, plug it into the **valueToStore** socket, and plug in the **TexBoxPage.Text**, **TextBoxContext.Tex**t, and **TextBoxDefinition.Text** blocks. We will use **TextBoxWord.Text** as the tag, as the user will click on the word to find it's page number, context, and definition.

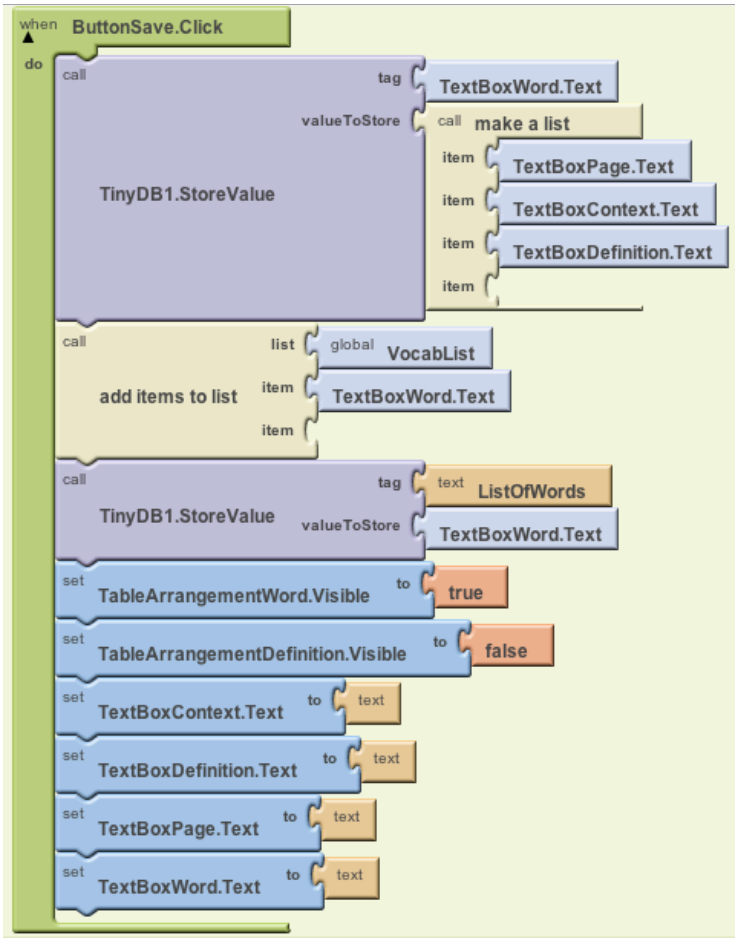
The **ButtonSave.click** should now appear as shown in the image below:



We also need to save each vocabulary word that is typed in. Drag out a second **TinyDB1.StoreValue** block, and plug **TextBoxWord.text** into the **valueToStore** socket. For the tag socket, we use a text block and type in a tag name that is indicative of what we wish to store (**ListOfWords**).

Finally, when we click the save button, we want to make sure we add our new vocabulary word to our variable, **VocabList**.

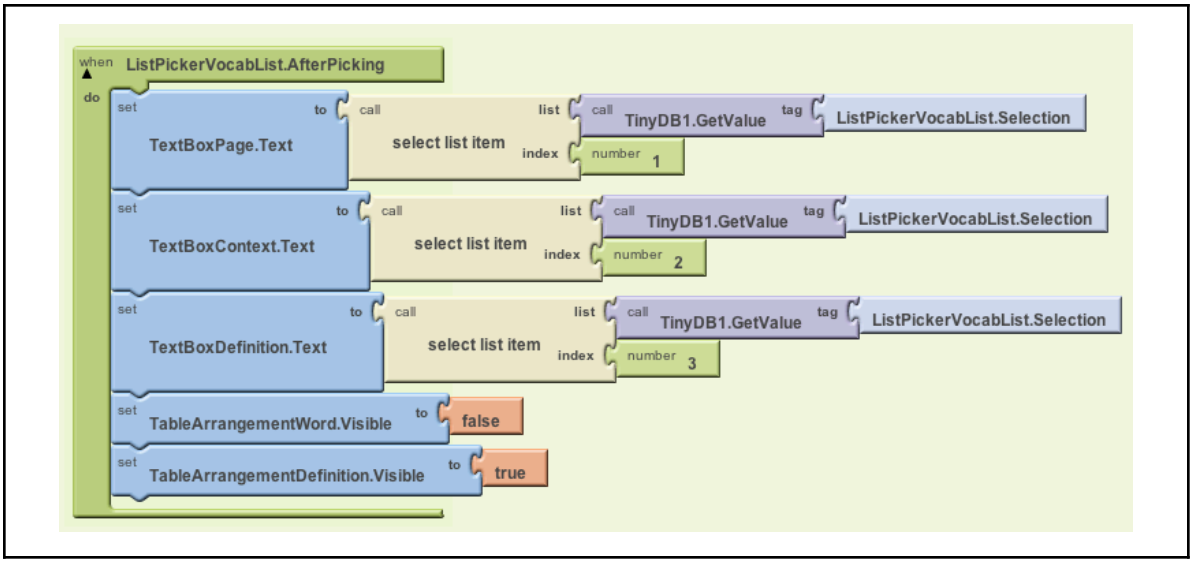
For now we are finished programming the **ButtonSave.click**, as shown below:



We can now save a new word, its corresponding page number, context, and its definition by clicking save. In order to display our saved data, we now need to program the listpicker.

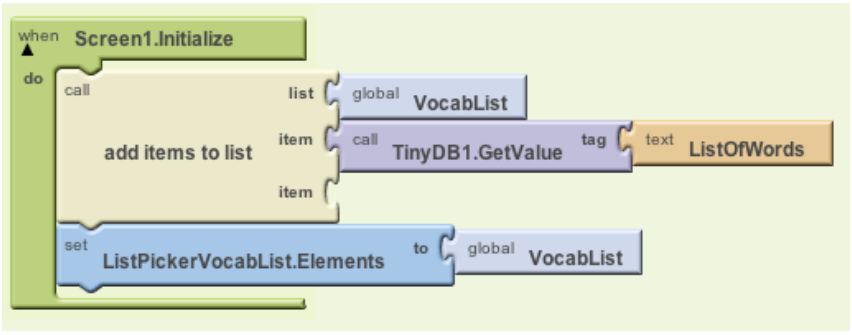
When we click on the listpicker (**Load List**), we want to see the list of words that we saved in the variable **VocabList**. After we click on any one of our vocabulary words, we wish to see the corresponding page number appear in the **TextBoxPage** box, the context appear in the **TextBoxContext** box, and the definition appear in the **TextBoxDefinition** box.

These text boxes are all in the **TableArrangementDefinition**, so we set this to **true**, and the **TableArrangementWord** to **false**.



Finally, we must initialize the screen. When we turn the phone off, any words stored in the variable **VocabList** get wiped out, however we can retrieve them from the **TinyDB1**, under the tag **ListOfWords**.

As soon as we open the app, we tell App Inventor to get all words saved under the tag **ListOfWords** and plug them into the variable **VocabList**. We can then tell App Inventor to display these words in the listpicker.



Our app is now completely functional! Go ahead and try it out.

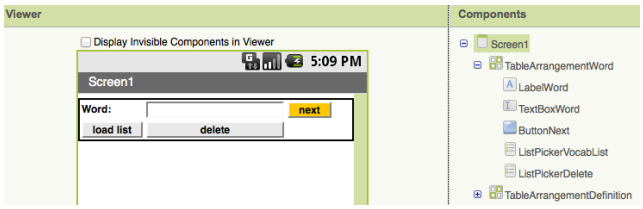
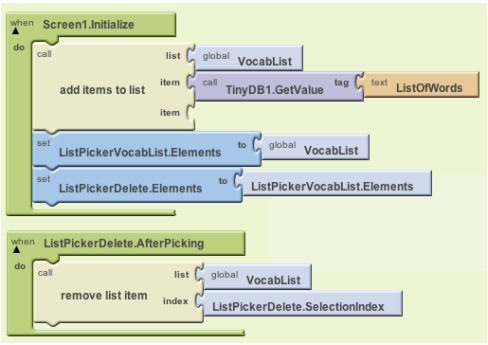
To spruce up our vocabulary tracker a bit, we will add the extra features listed below:

1. The ability to delete a word from the listpicker
2. The ability to save additional information to the text boxes in the TableArrangementDefinition at a later time.
3. The ability to launch an online dictionary from our app (we chose to use google.com as

our dictionary, however you can use any dictionary you like)

1. Deleting a word - We want to be able to click on a listpicker with the word “delete” on it to view our list of vocabulary words. Whatever word we select from our list will then be deleted.

- In the design window we will add a listpicker to the TabelArrangementWord.
- In the blocks editor we will add a procedure block, name it delete, and program it to delete whatever word the user selects.

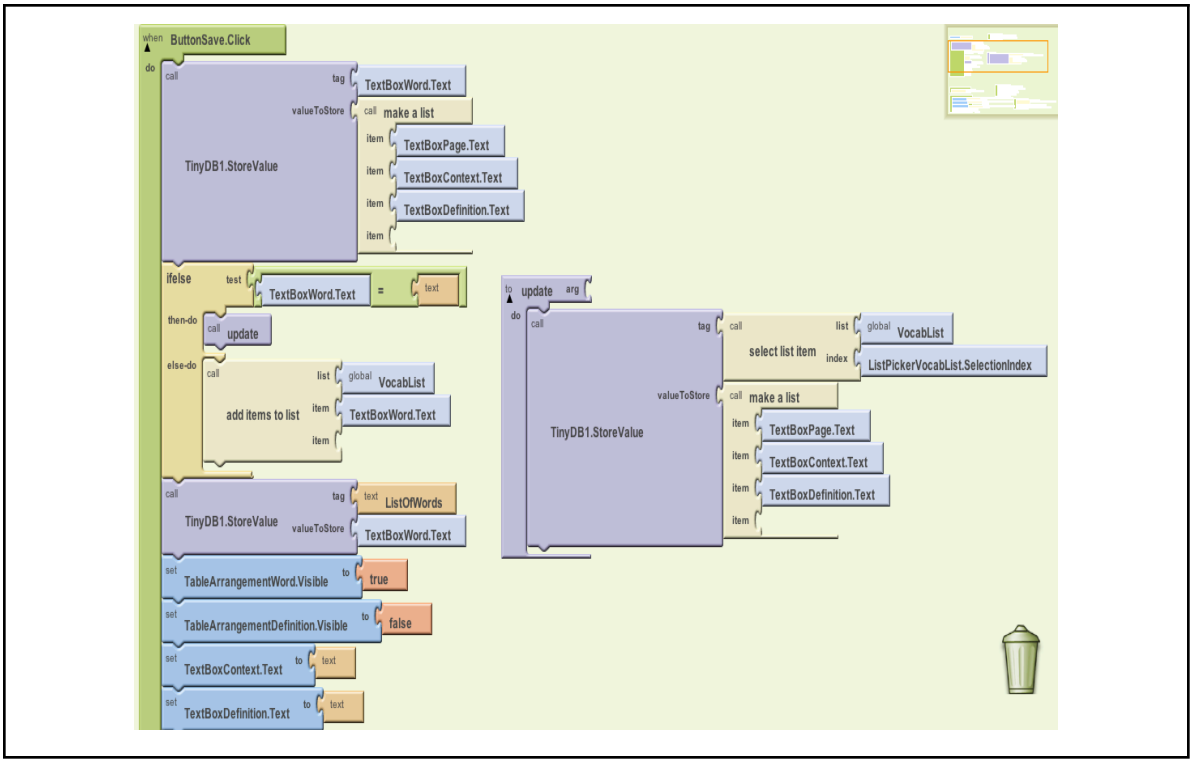
	
We added a listpicker, **ListPickerDelete**, and typed the word **delete** on it. We will program this listpicker such that when the user touches this listpicker, the list of vocabulary words will appear. When the user picks one of the vocabulary words, it will be deleted.	When the screen initializes we must load each element in the **ListPickerVocabList** into the **ListPickerDelete** llst so that we can select a word to delete. After we select the word we wish to delete, we use the **remove list item** block to tell App Inventor that want to remove the selected word from the variable **VocabList**.

2. As is, if we enter a word, click save, and come back later to type in the definition, the definition does not appear to be saved after we click the save button again.

This is because when we clicked save the fist time, we cleared the text box that contains the corresponding vocabulary word. Recall, we are using the text in that same text box as the tag. When we clear the **TextBoxWord**, the tag is blank and therefore it appears as a blank space in our listpicker.

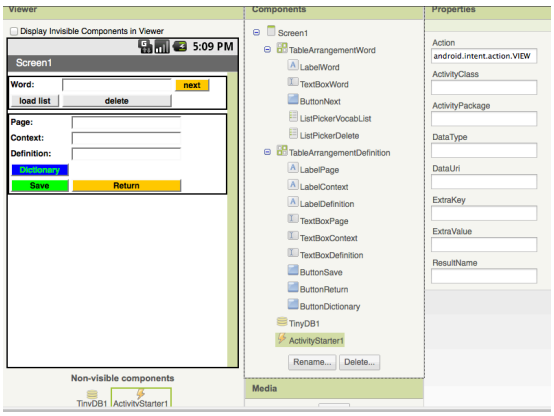
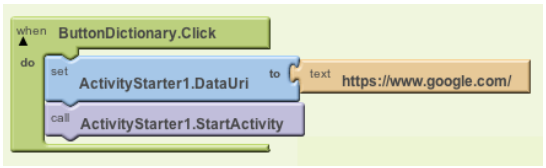
We can fix this with an **ifelse then do else do** block. We need to tell App Inventor that when we click save, if the tag is blank, add the newly input information to the text boxes that correspond to the word we originally chose from the listpicker.

This is a mouthful, and our **when ButtonSave.click** is already packed with instructions, so we will create a separate procedure to store the page number, context, and definition that we want to add under the selected word. We will name this procedure **update**.



3. Linking to an online dictionary - This requires dragging two new components onto the design screen:

- a button to click when we want to connect to an online dictionary; we chose to place this button on the **TableArrangementDefinition** section of the screen.
- an activity starter (found in the **other stuff** drawer); The activity starter automatically snaps down to the non-visible components section at the bottom of the screen.
    - In the design window, under the properties column, there are several blank text boxes. The only one we need to fill in is the **Action** box. In the **Action** box, type the following command and hit return: **android.intent.action.VIEW**

Design Window	Blocks Editor
	
We added an activity starter and a button (Button.Dictionary)	We programmed **ButtonDictionary** to connect to google.com so that we can look up unfamiliar words. You may replace google.com with the URL of your favorite online dictionary; be sure to include https://www in the url.

This is now a simple, but highly functional, app! Students can type in unfamiliar words as they read, and come back later to define them. They can also quiz themselves on words they entered in the past.