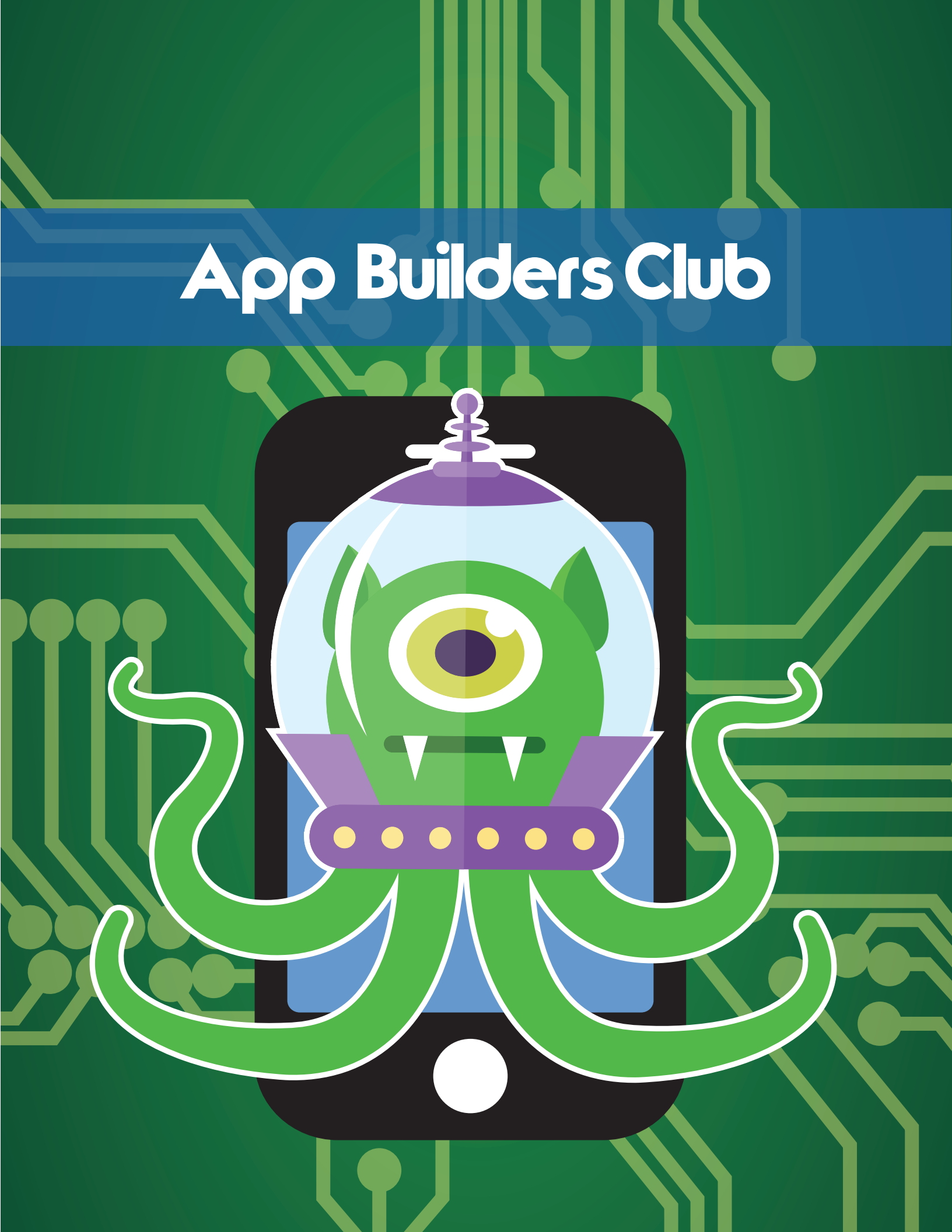
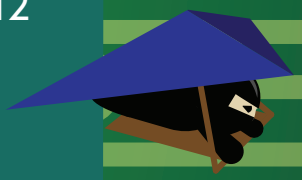
App Builders Club



Contents



Welcome to App Builders Club!

As you will soon discover, making useful and fun apps for your tablet or smart phone is as easy as surfing the web.

What's an App?

An app (or *application*) is a programed set of directions designed to run on a mobile device such as a smart phone or tablet. Currently, most mobile devices use apps that are designed to work either with an Android or iOS operating system. At this time, App Inventor can create apps only for Android devices. Over the next few days, you will be creating apps that you can either try out on your own android device (if available) or on an emulator on your PC.

Emulators

When making apps to run on other devices, designers frequently use a program designed to simulate or act like that device. This is known as an ***emulator***. An emulator can mimic some, but not all of the capabilities of the device on your PC. Whenever possible, all of the apps you will be designing work on either an emulator or an actual android device.

Photo by Tim Gouw on Unsplash

What is App Inventor?

App Inventor is an easy-to-use set of tools for you to quickly design and build actual apps for mobile devices. Drag the components that you want to use and adjust their properties. Then use a drag and drop method of scripting to create code to make those components do things. Even if you've never tried to write a program before, App Inventor makes it easy to create apps.

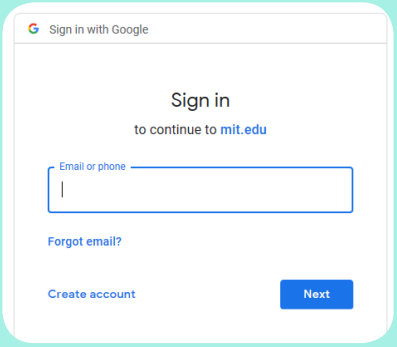
Before you begin:

To use App Inventor, you will need two things: a browser to access App Inventor on the internet and an android device or emulator. If either is missing, follow the steps below.

Using App Inventor

1 Go to the App Inventor website at http://ai2.appinventor.mit.edu. You will be asked to sign in.

2 Follow the instructions for signing in or creating an account.



Setting Up App Inventor

1 Before you can run any of the apps you build in App Inventor, you have to set up a device to test your apps on. Your PC should already be configured to use the Android emulator. To do so, be sure that aiStarter is running before using the emulator.

2 If you wish to use an Android device with App Inventor, that device needs to have the "MIT AI2 Companion" installed. This app is available for free on the Google Play site. Visit http://appinventor.mit.edu/explore/ai2/setup-device-wifi.html for additional information.

Assets

1 In the Exercises folder are all of the assets needed for each activity as well as a folder containing some example files. You are also free to substitute your own images and sound files.

1 Your First App

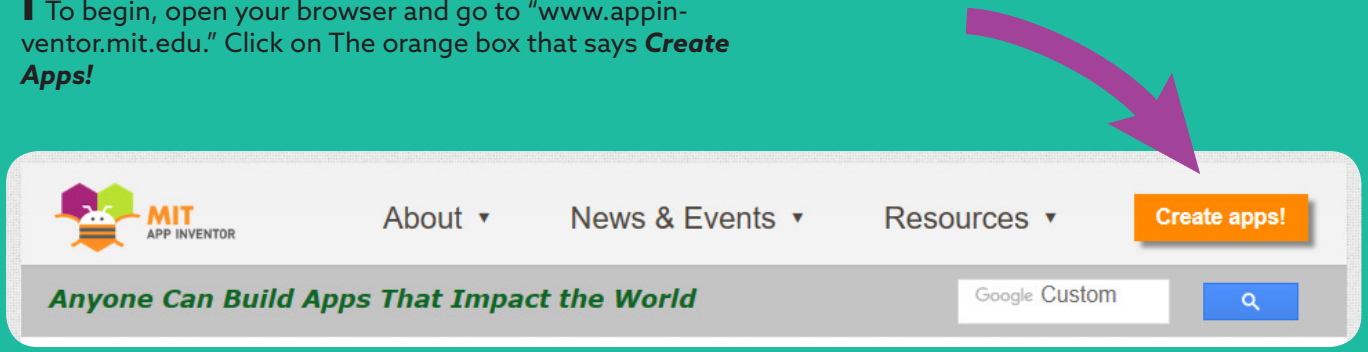
If you've always wanted to make an app for a phone or tablet, you're in the right place. Not only will you learn how to make an app and run it on an Android device, you can do it completely on the internet without downloading anything to your PC.

Talk to Me

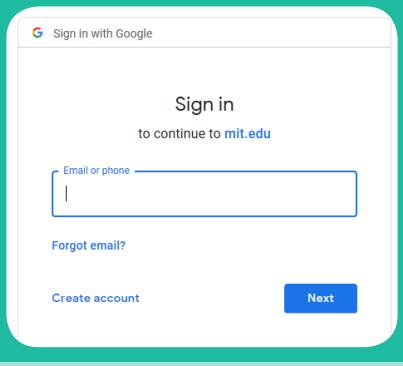
Are you ready to start making apps? This activity will show you how to make your way through App Inventor and we'll make a fun little app in the process. Are you ready? Let's begin!

Activity 01-01: Talk To Me

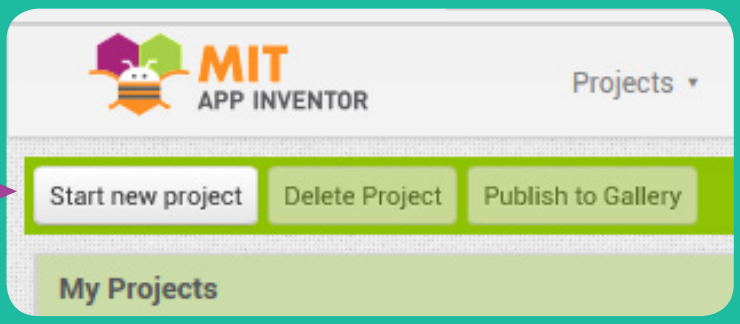
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



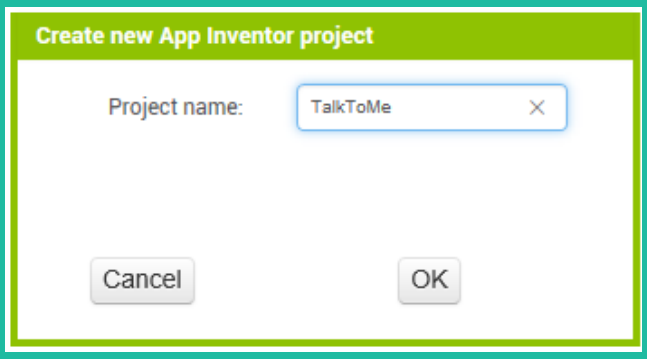
2 When you first start App Inventor (or if you've been away for a while) you are going to need to sign in so that App Inventor can keep track of what you create.



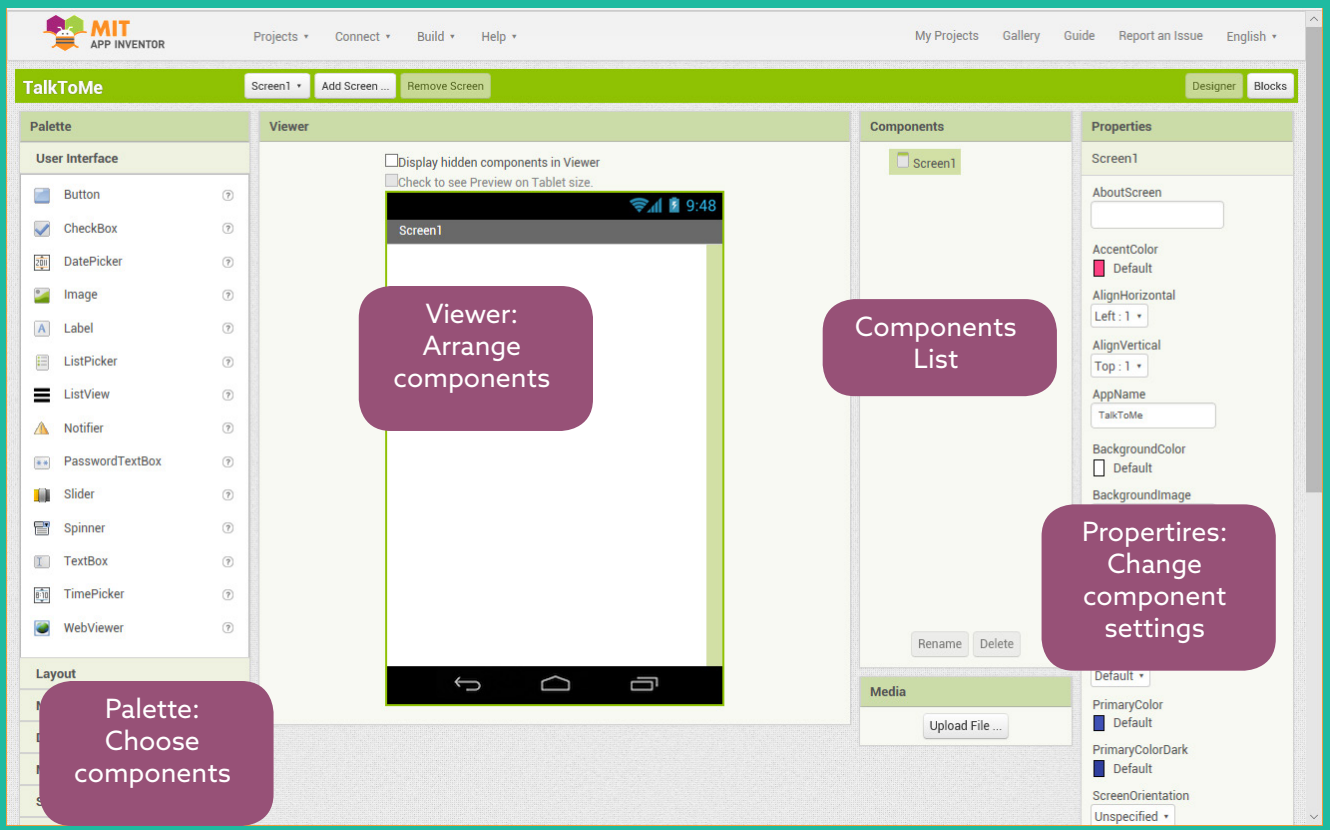
3 Click on the START NEW PROJECT button.



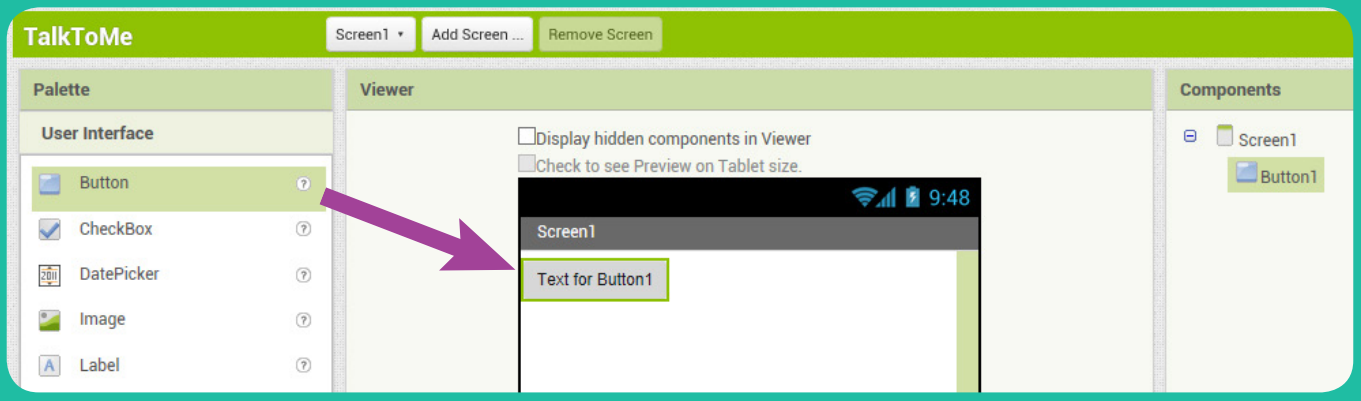
4 Name your new project "TalkToMe" (no spaces).



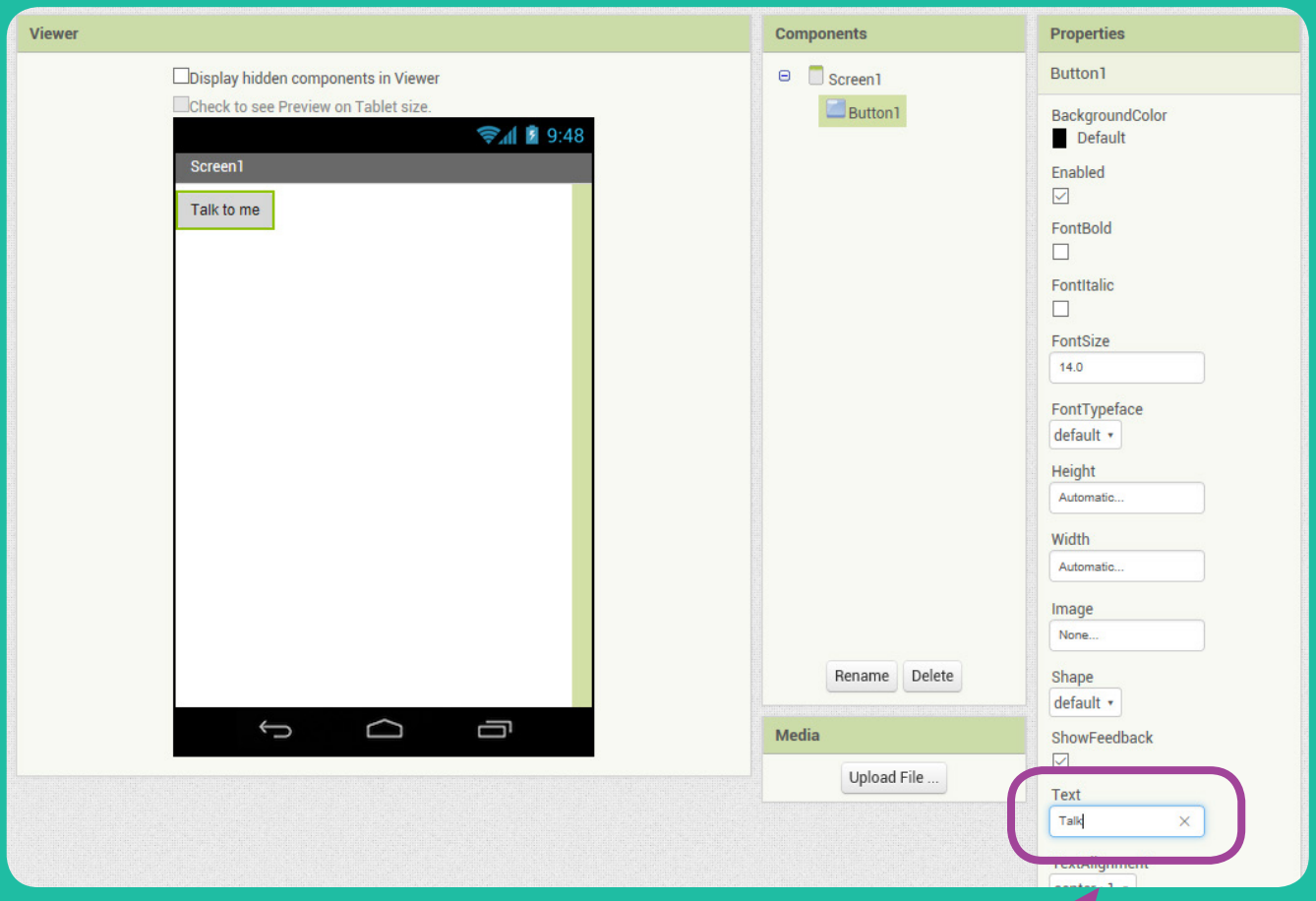
5 Once you've created a new project, App Inventor opens the designer window where you will design the interface for your app.



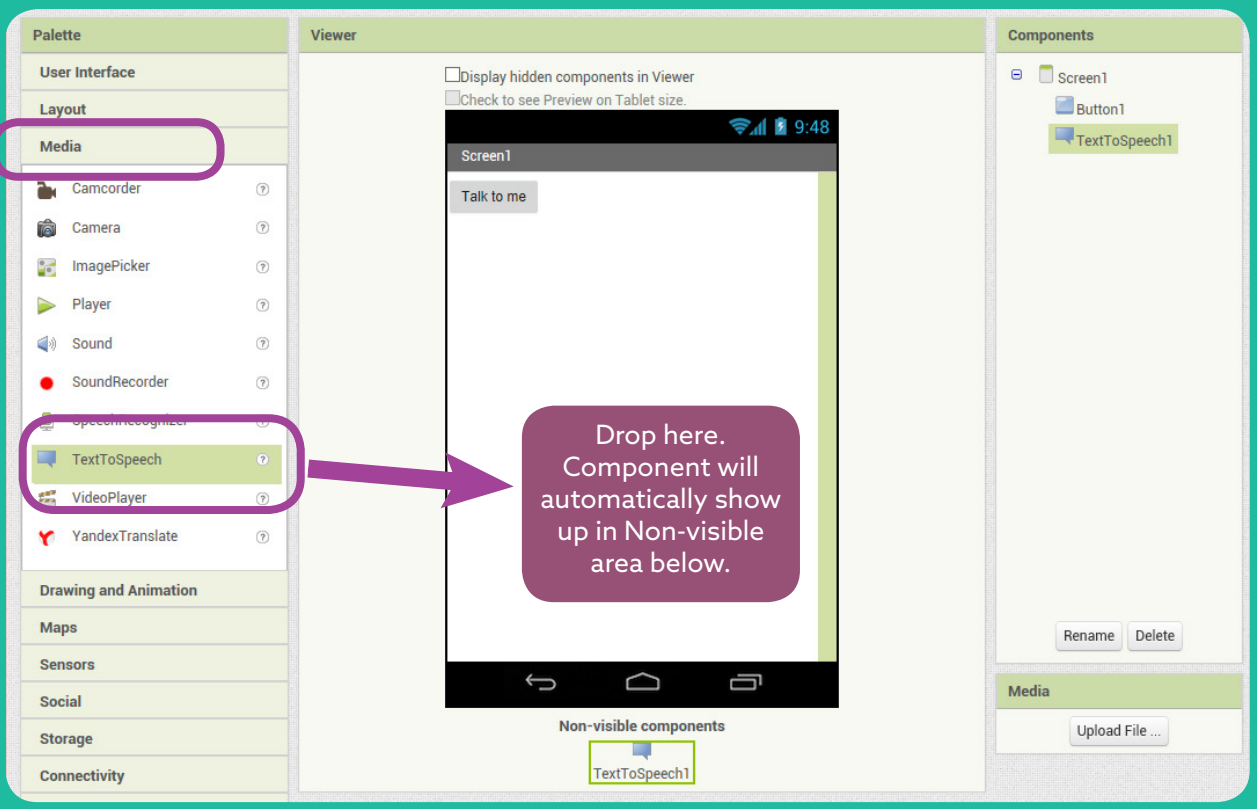
6 Add a button by clicking and holding the word "button" in the Palette and drag it over into the Viewer.



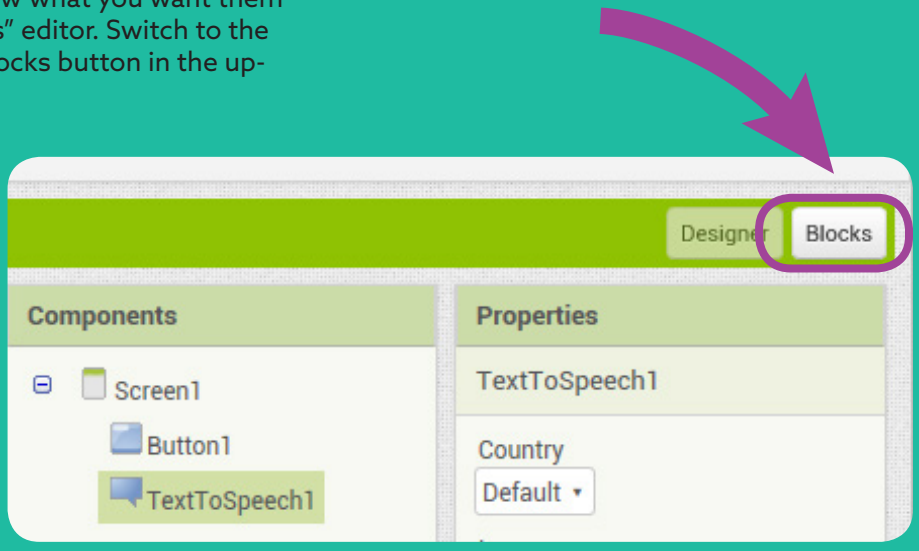
7 Change the text on the button. In the properties panel, find the "text" property and change it to something appropriate.



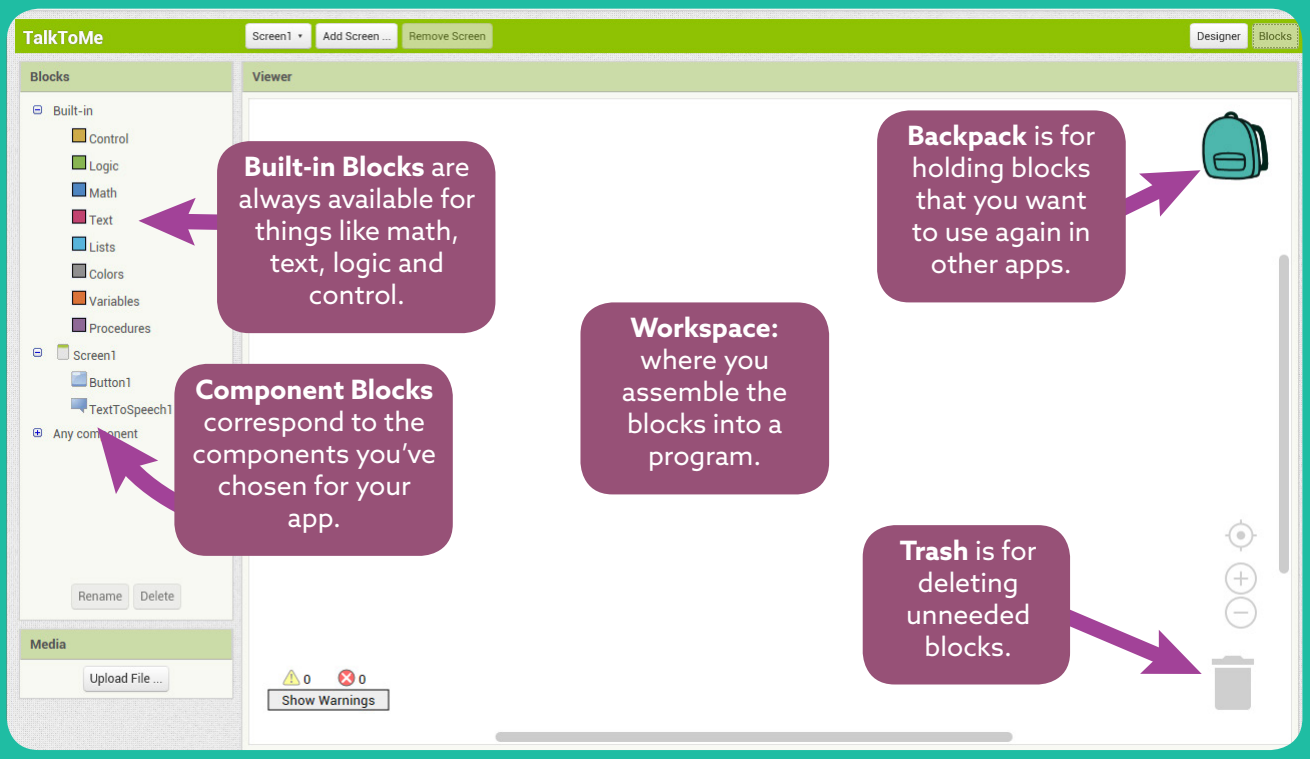
8 At the moment, the button doesn't do anything. Let's add a Text-to-Speech component. Select "media" from the component palette and drag the TextToSpeech component into the viewer. This component is non-visible, so it immediately drops into the Non-visible components section at the bottom.



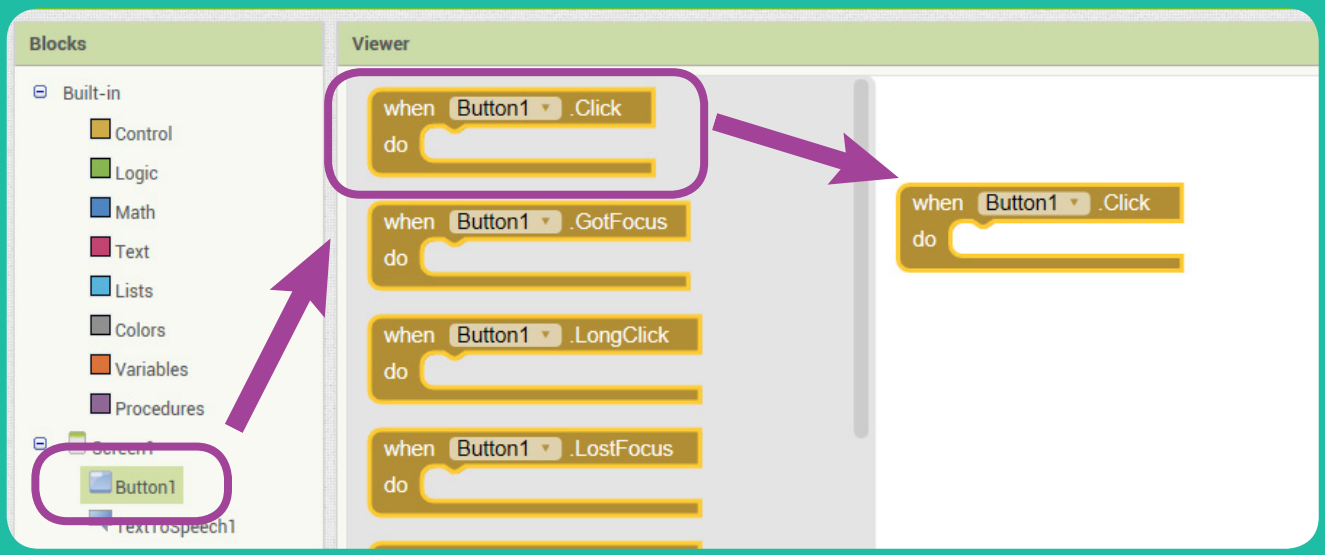
9 The components still don't know what you want them to do. That happens in the "Blocks" editor. Switch to the Blocks editor by clicking on the Blocks button in the upper right corner.



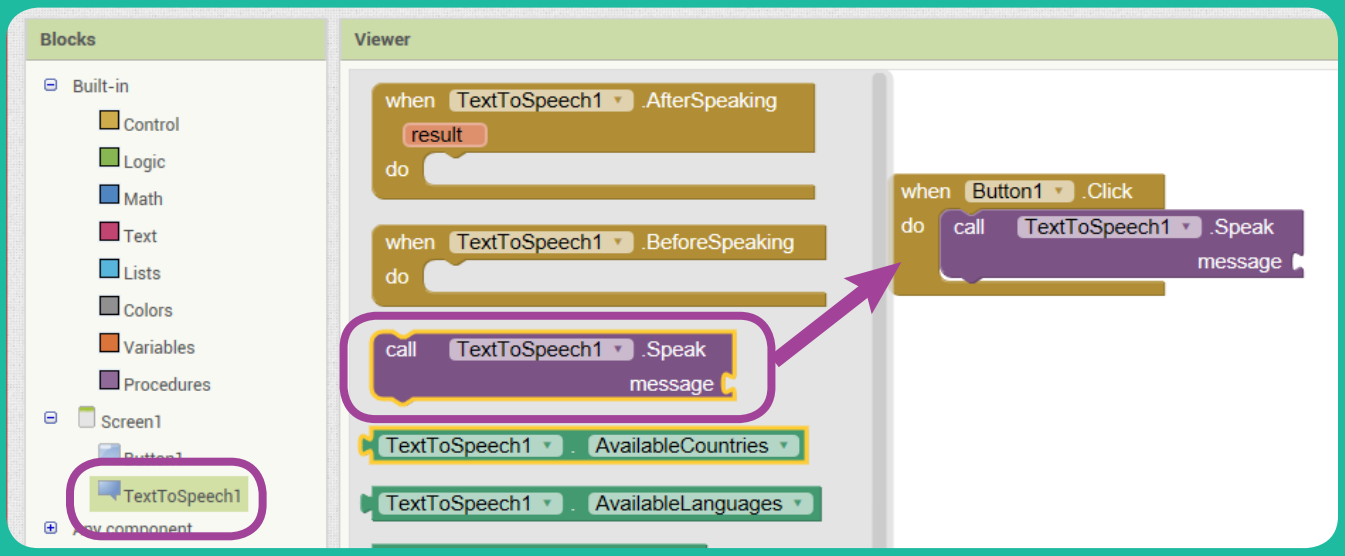
10 The Blocks editor has two sections. The first has the blocks which contain the directions for what you want your app to do. The second is the viewer where you put the blocks together.



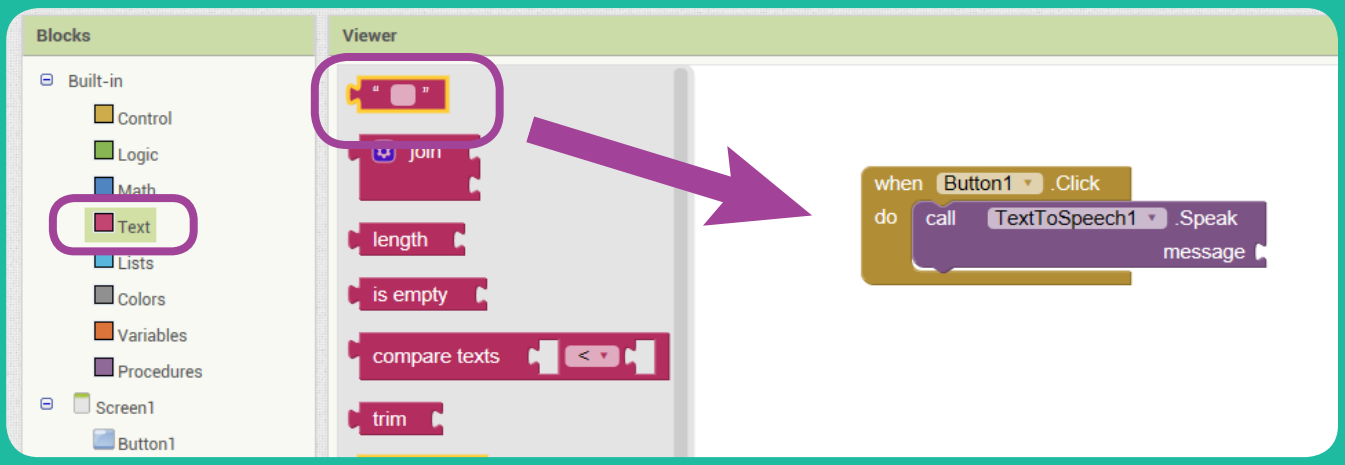
11 Select the Button1 component by clicking on it. This opens many possible blocks to be used with the button component. We ***want when Button1.Click do***. Drag it over into the viewer and drop it. Now when the user clicks on Button1, it will do something.



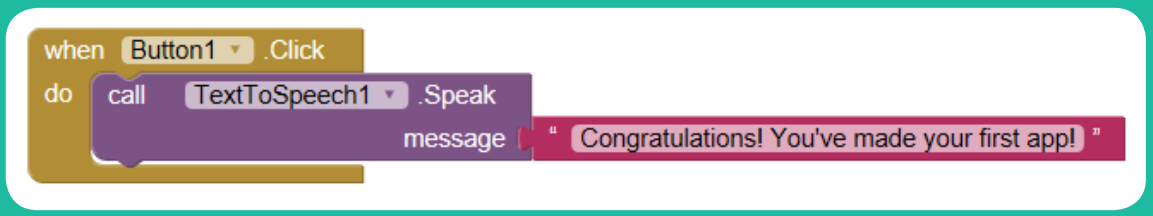
12 But what will it do? Select the TextToSpeech1 component to get the possible blocks for that. We want the ***call TextToSpeech1.Speak message*** block. Drag it so that it fits inside the Button1.Click event.



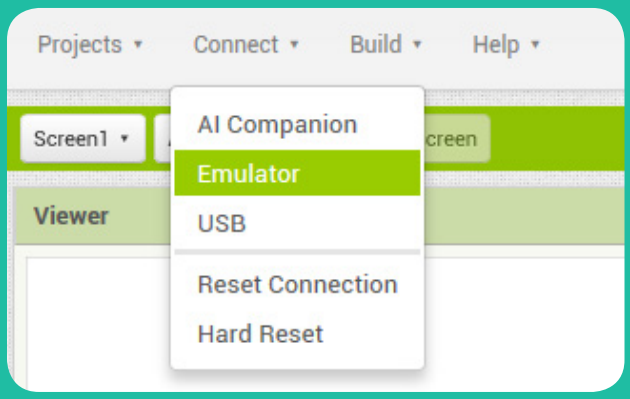
13 There is a socket in the TextToSpeech block for a message. We need some actual text! Select text and drag a text block over to that socket so that your event is finally complete.



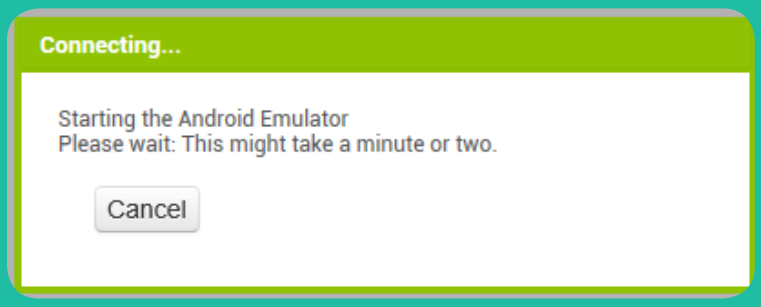
14 What will it say? Type in any message that you want your app to say when the button is clicked.



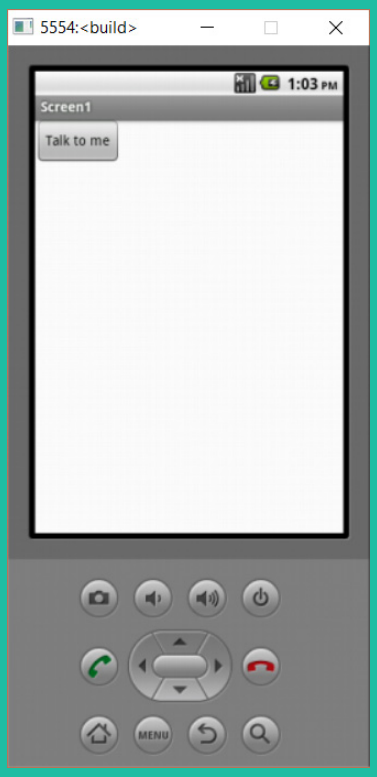
15 Now test your app! Select the ***Connect*** tab from the top of the window and from that, select ***Emulator*** (or ***AI Companion*** if you have an actual Android device).



16 Wait for App Inventor to launch the emulator and connect to it.



17 Once your app is running in the emulator, click on the Talk to Me button to hear what it has to say! (Make sure that your speakers are turned up.)

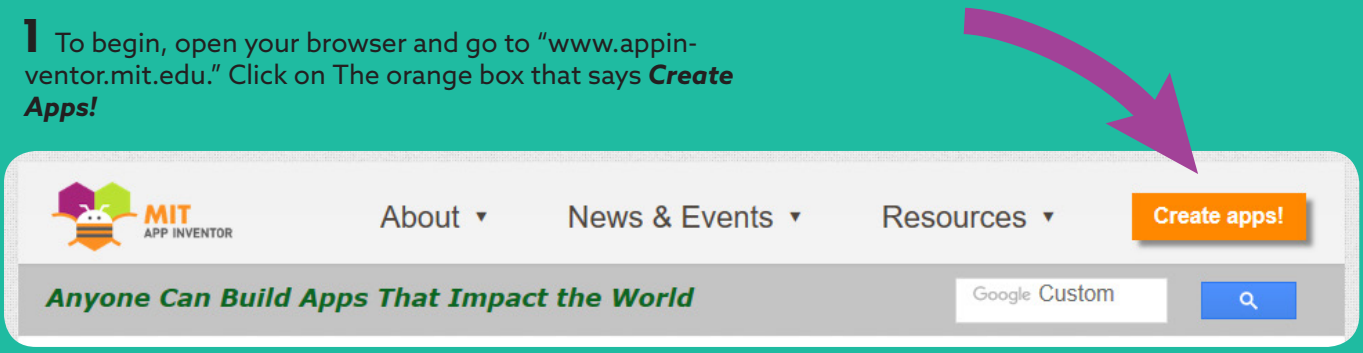


Keep Talking

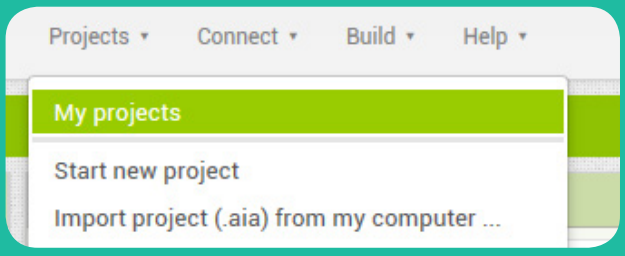
Did you notice that "Talk to Me" could say most anything you want it to? If only there was a way to make it easier to change what the app says when you press the button. But why stop there? Let's give the app a little personality, too.

Activity 01-02: TalkToMe 2

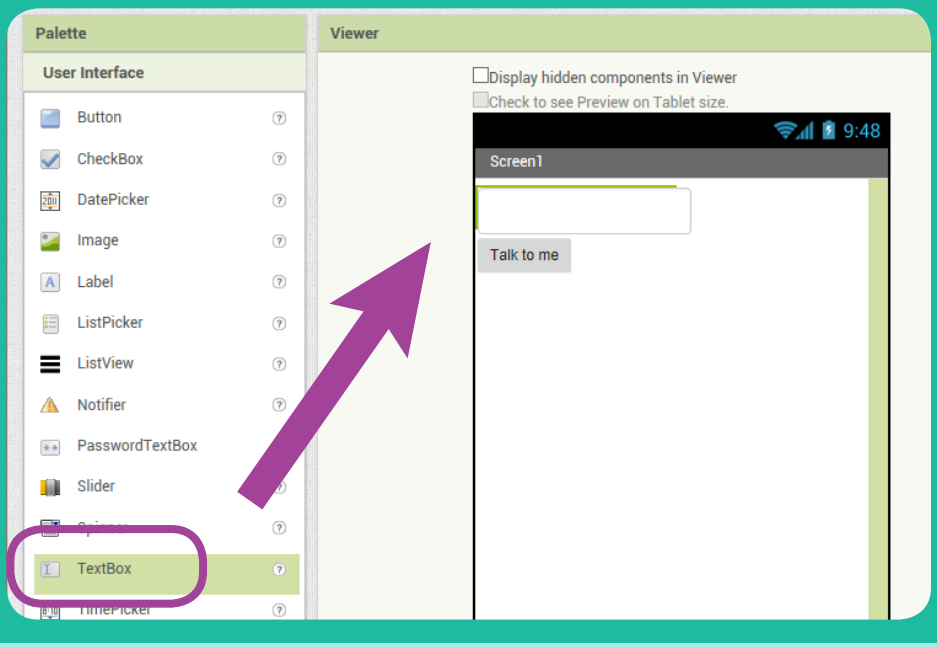
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



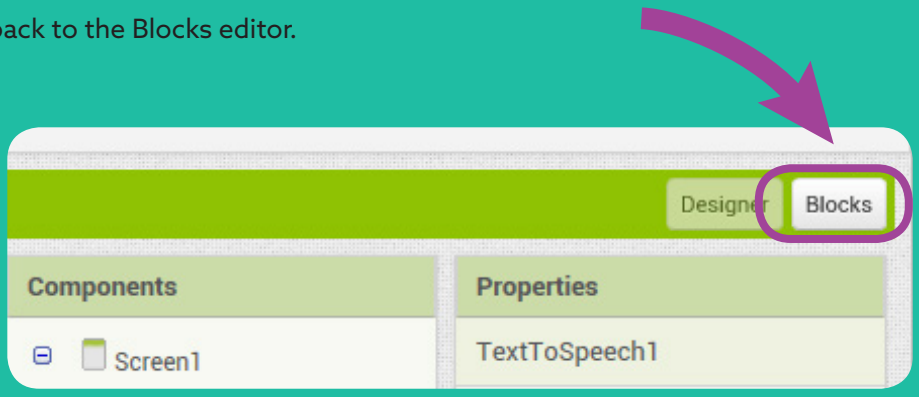
2 If your "TalkToMe" app doesn't open automatically, click on the ***Projects*** tab, select ***My Projects***, and select TalkToMe from the menu.



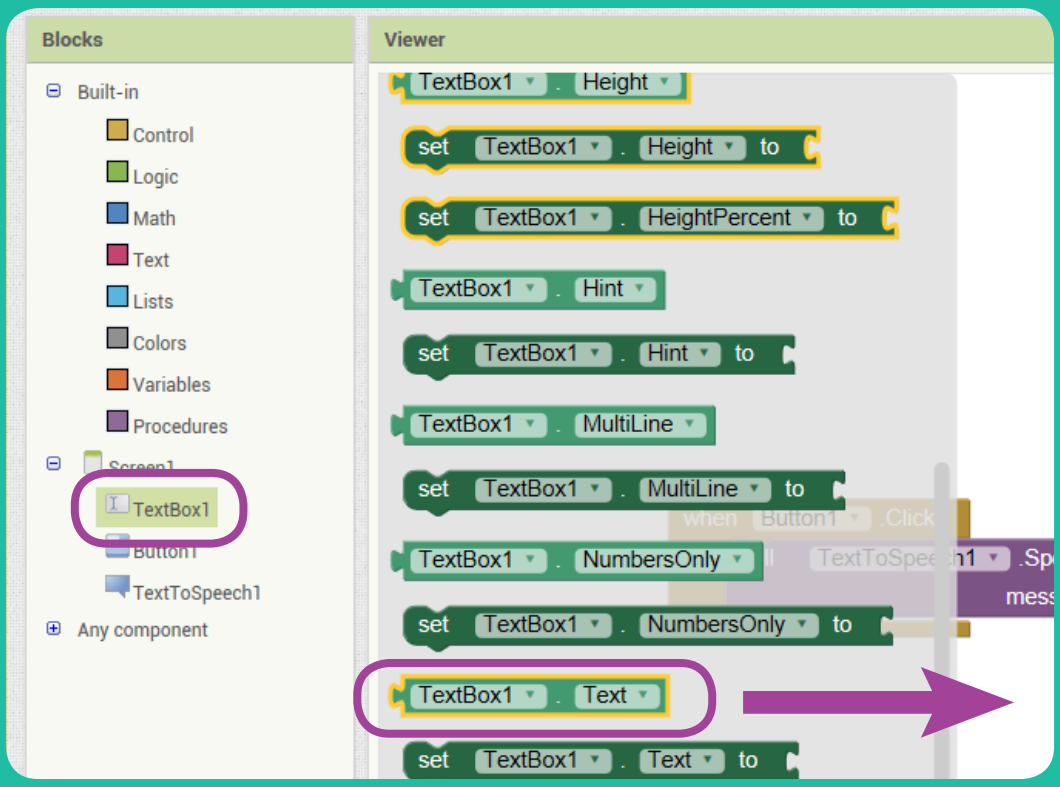
3 Make sure you're in the Designer section and then select a TextBox component from the User Interface part of the Palette. Drag it into the viewer and place it above the button.



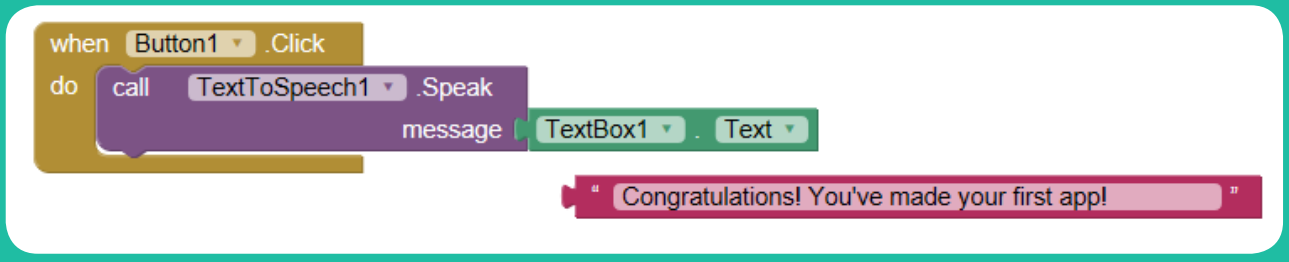
4 Once you have a text box, go back to the Blocks editor.



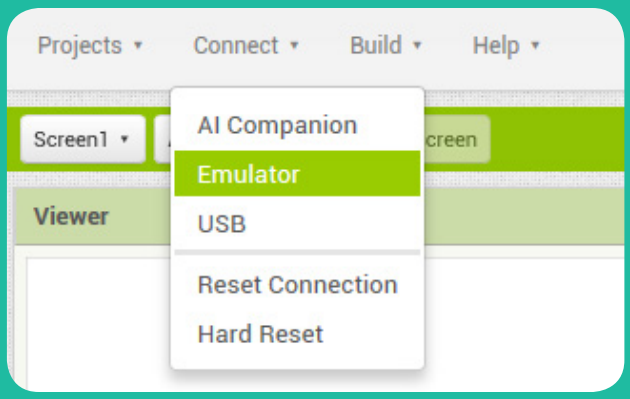
5 Any text that the user puts into the text box has the property of "text." Can we use that as our new message? Yes! Click on the TextBox1 component and select the ***TextBox1.Text*** parameter and drag it into the viewer.



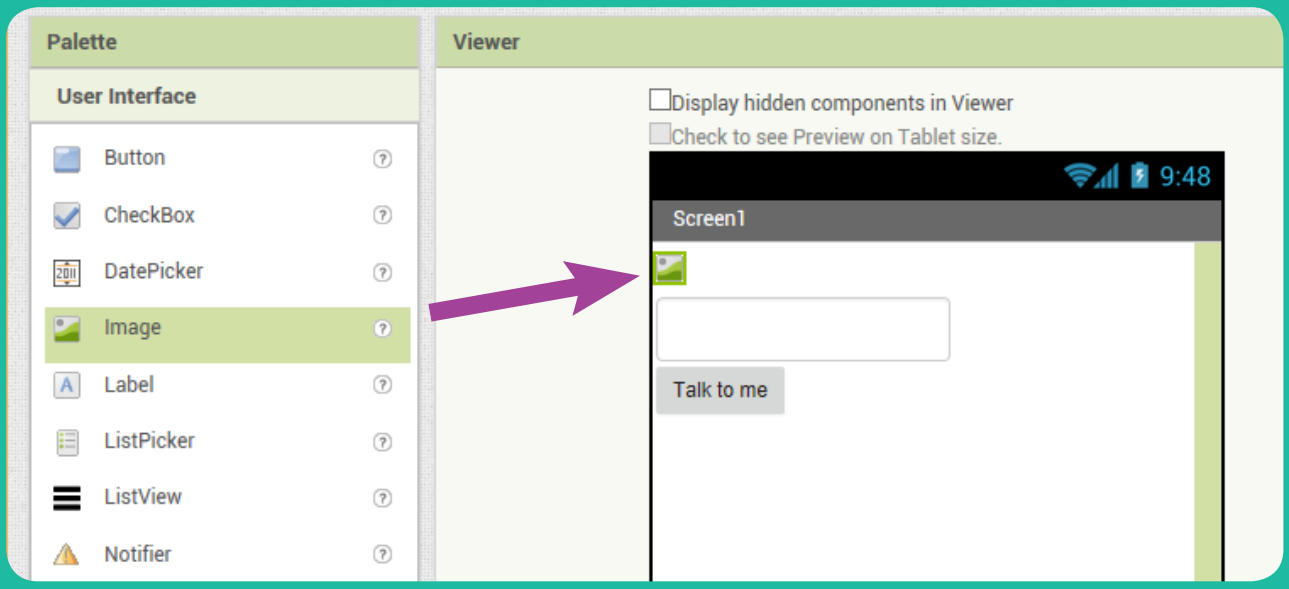
6 Replace the previous text parameter with the new ***TextBox1.Text*** block. You can get rid of the old block by dragging it into the trash in the corner.



7 Go ahead and test your app (you may need to click on "Reset Connection" and "Emulator" again if the emulator isn't responding).

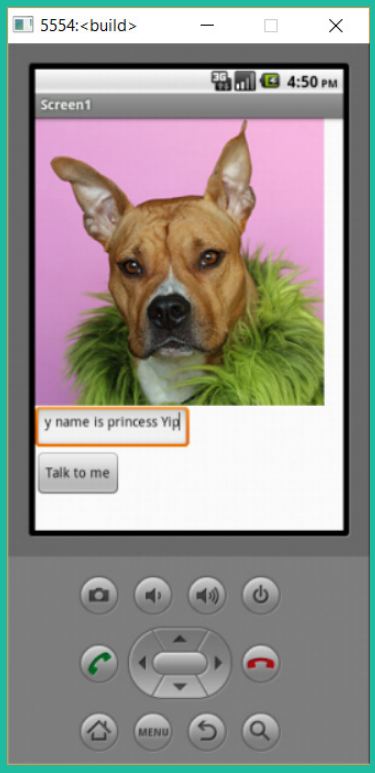


8 Our app has a voice, but it should have a pretty face, too. Go back to the designer section and select the ***Image*** component from the ***User Interface Palette*** and drag it into the viewer.



9 The image component doesn't show anything by default, because we haven't told it what to show. Make sure that the image component (***Image1***) is selected and then click on the ***Picture Property.*** Select ***Upload File*** and select the picture "Dog.png" from the assets folder. Click ***OK.***

10 Test your app. What does the app want to tell you?



More About Apps

All applications ("apps") are simply a series of directions that tell the device what to do. They may be called "smart" phones, but they really can't do anything until they are told what to do. Even the part that turns the thing on and shows you a menu is actually a set of instructions that is often referred to as a "script" or "code."

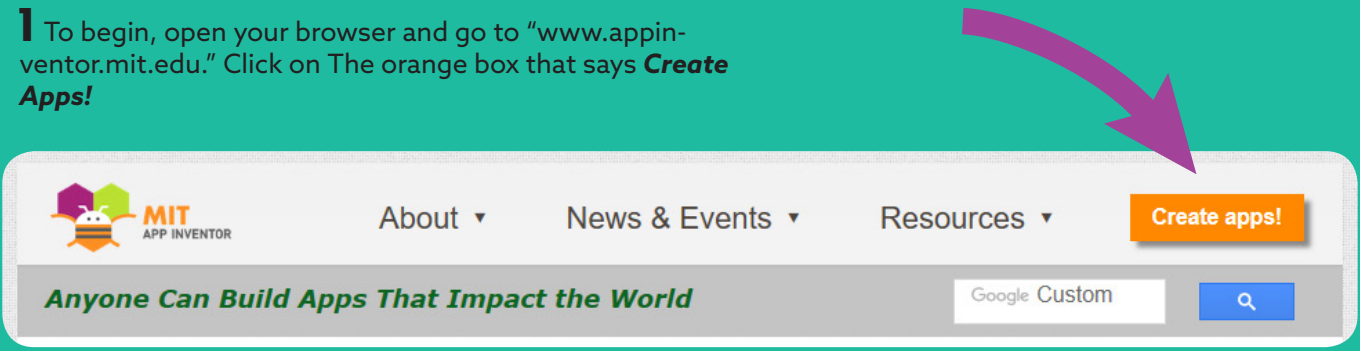
I Have A Dream

This app has two parts. The first part has a picture of Martin Luther King that, when clicked, plays a clip from the famous speech he gave at the Lincoln Memorial in 1963.

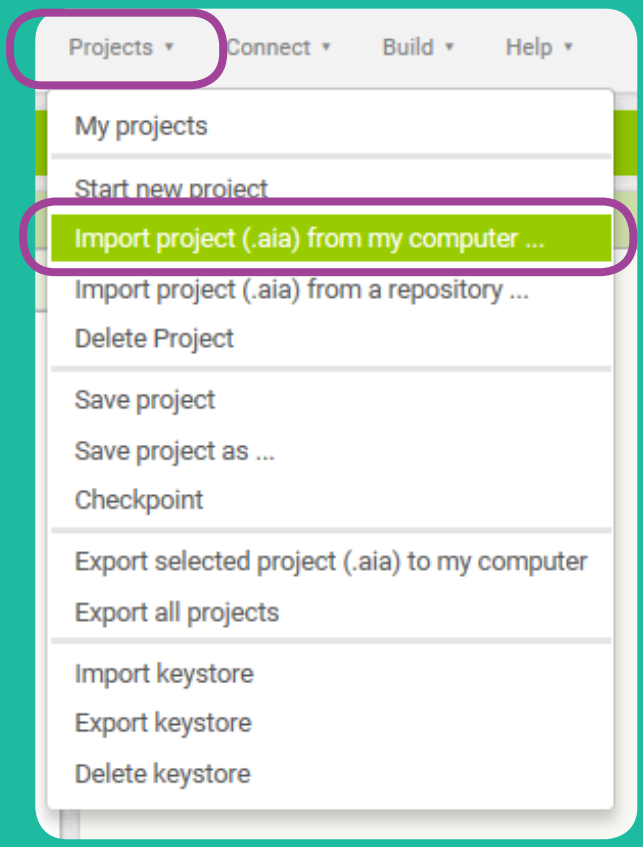
The second part has you adding another historical figure, Malcolm X, and plays one of his speeches as well. This second part will teach you a bit about conditional "if" blocks and how to use them.

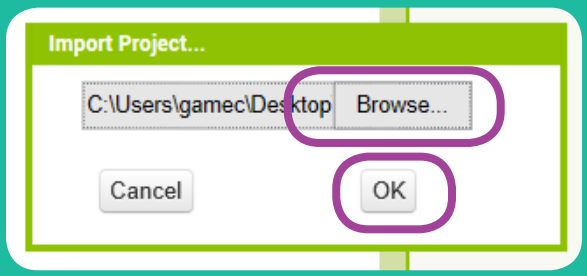
Activity 01-03: I Have a Dream

1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***

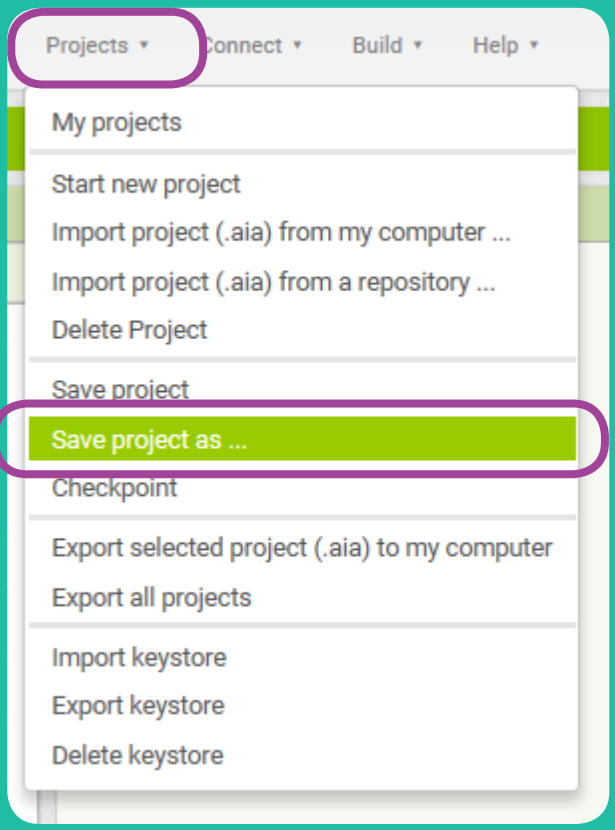


2 To save time, we will be loading a partially built project from the Assets folder. Click on the ***Projects*** tab at the top of the screen, then click on ***Import project (aia) from my computer.*** Click on ***Browse*** and from the Assets folder, select "IHaveADreamStarter.aia" and click ***OK***.

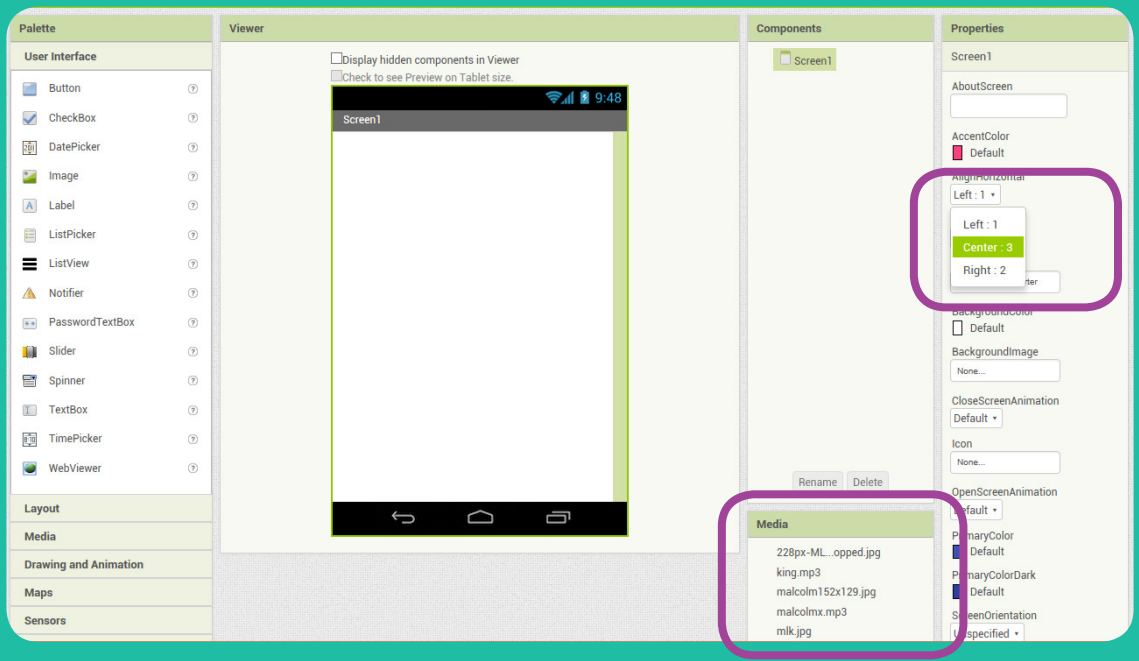




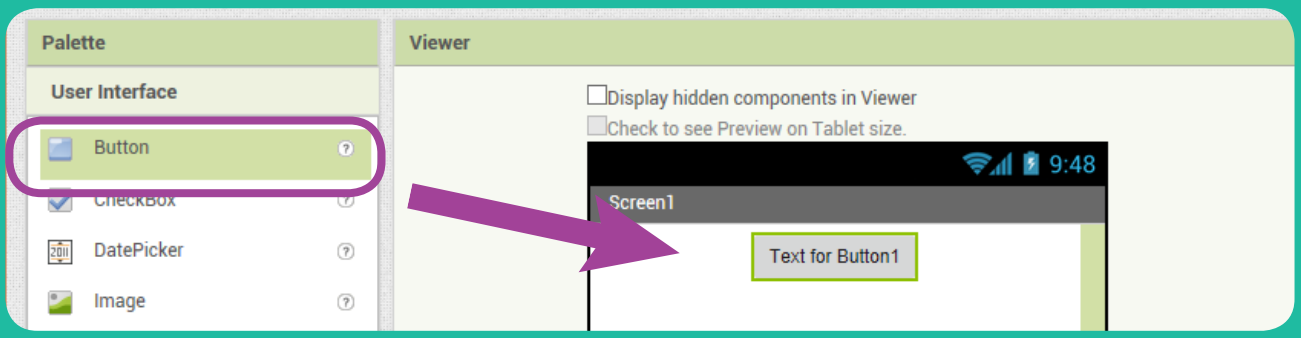
3 Once the file is loaded, give it a new name. Click on the ***Projects*** tab again and select ***Save project as***. Choose a name like "IHaveADream" for your project name.



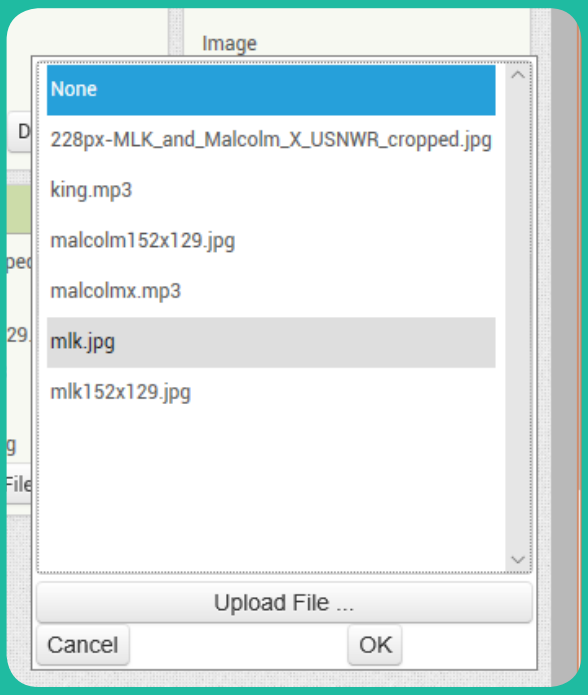
4 Once you have chosen a name for your project, it will open the App Inventor Designer. As you can see, the project already contains all of the media files that you will use. Let's get started by selecting the Align Horizontal property for the screen and change it to "Center."



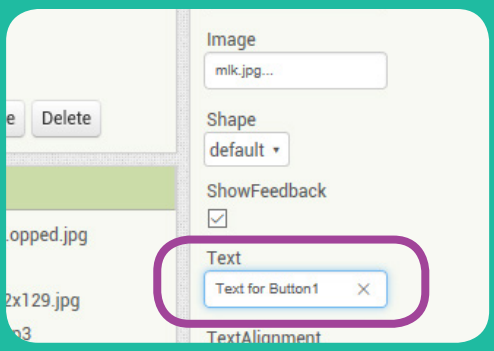
5 Add a button by dragging it from the ***User Interface Palette*** into the ***Viewer***.



6 Click on the button's ***Image*** property and select the file "mlk.jpg" as the image.

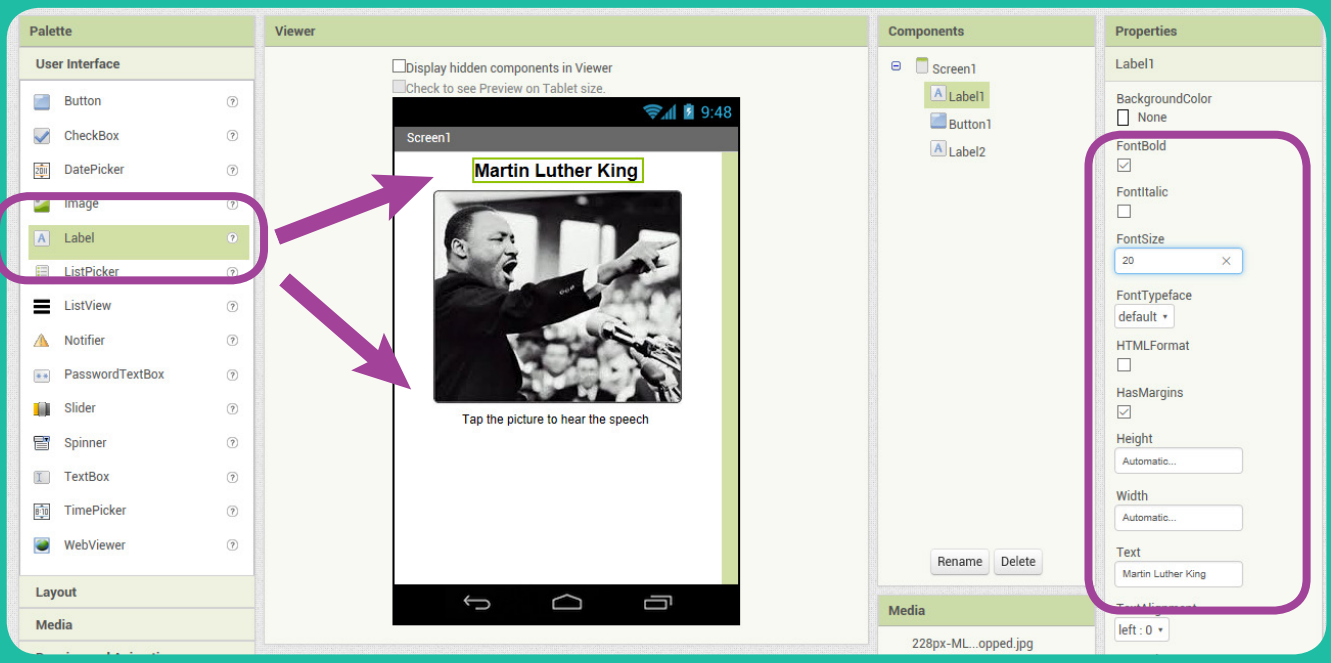


7 We don't need text for this button, so click on the button's ***Text*** property and delete the default text.

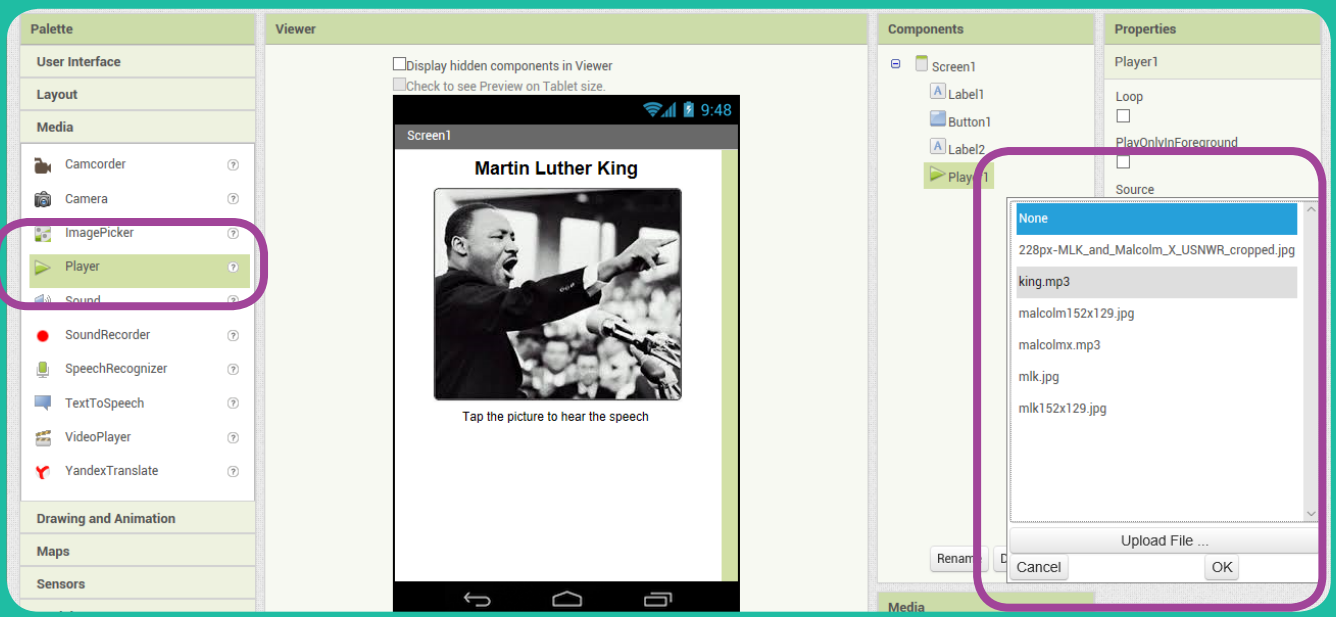


8 We need some text to explain what this App is about. Drag a label component from the User Interface Palette and place it above the button. Change the default text of the label to "Martin Luther King." Click on the "FontBold" property checkbox and change the FontSize to 20.

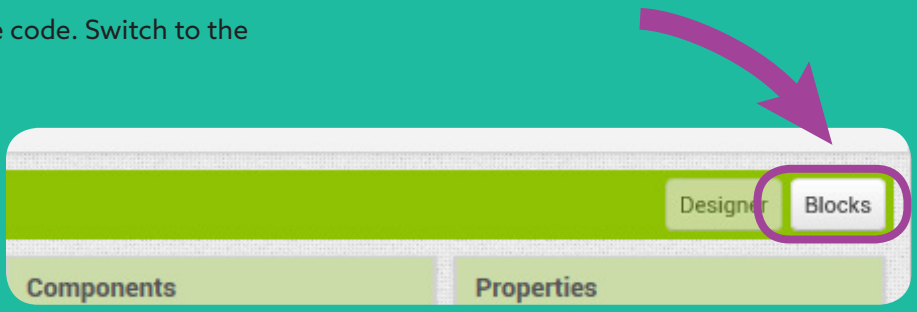
Drag a second label into the viewer and place it below the button. Change the label text to "Tap the picture to hear the speech." Leave the other label properties at their default settings.



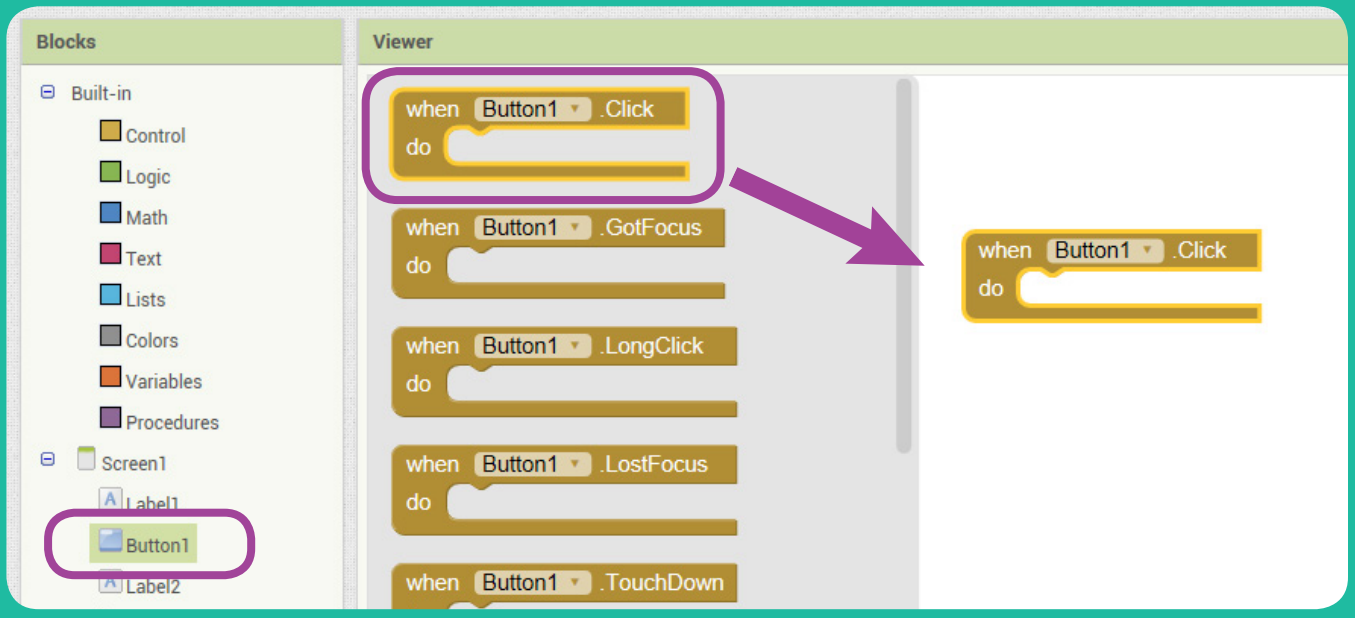
9 Select the ***Media Palette*** and drag a player component into the ***Viewer***. The player component is "non-visible" and positions itself below. With the player component selected, set its ***Source*** property to "king.mp3"



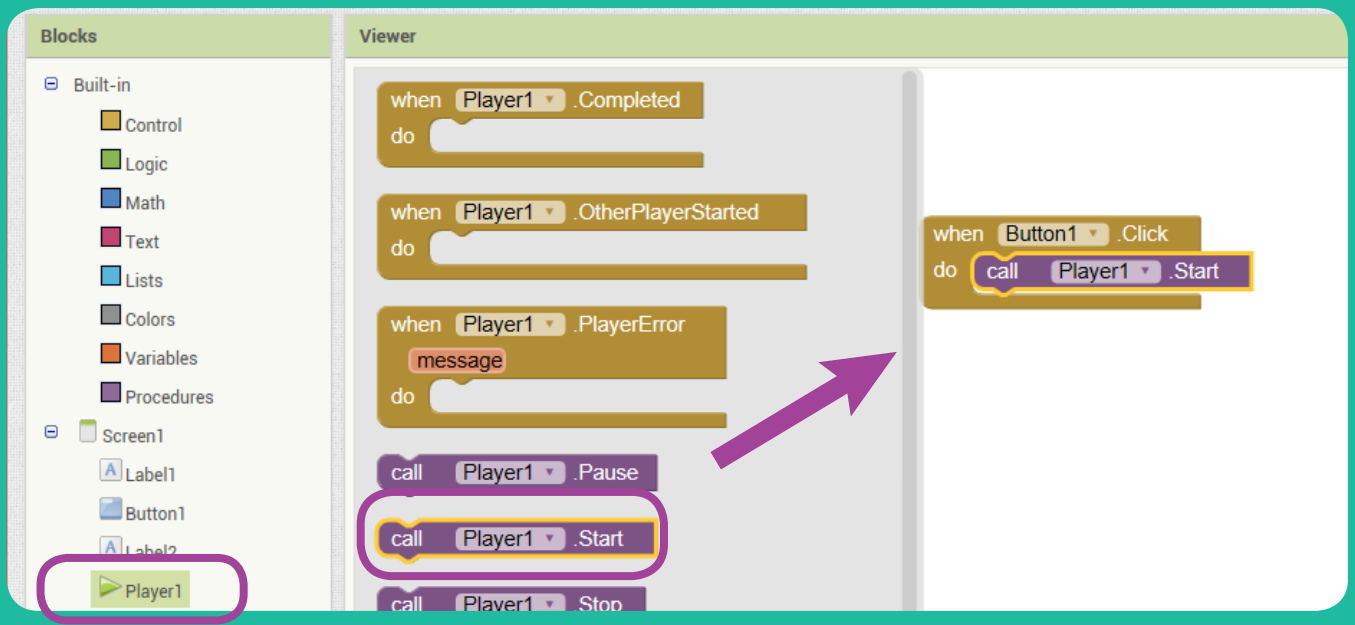
10 Now it's time to make some code. Switch to the Blocks Editor.



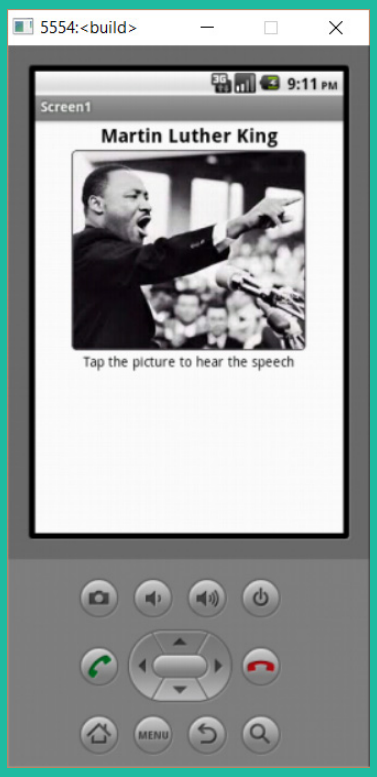
11 Select ***Button1*** from the Blocks menu and drag a ***when Button1.Click do*** event into the viewer.



12 Now click on the ***Player1*** component and drag a ***call Player1.Start*** block into the ***Button1.Click*** event.



13 Test your app. Does the sound play correctly when you tap the picture?



Expanding the App

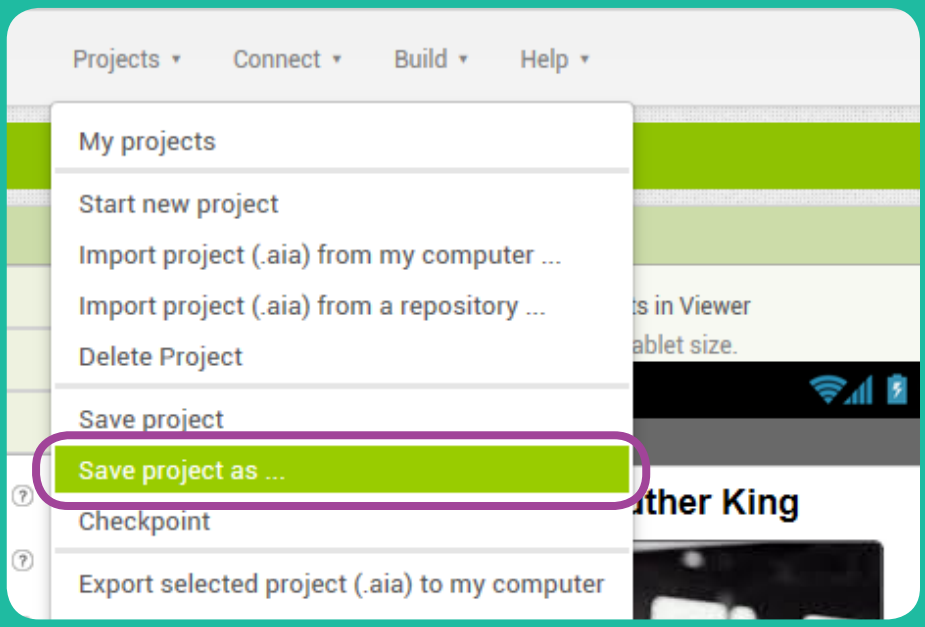
This next activity has us expanding the app to include a speech from another civil rights leader, Malcolm X.

Naturally, we will need another button and an additional player for the other sound file. But here is where it gets tricky. We don't want one sound file playing while the other one is playing.

It may sound like a big deal, but you'll find that it isn't too difficult. Let's begin.

Activity 01-04: I Have a Dream 2

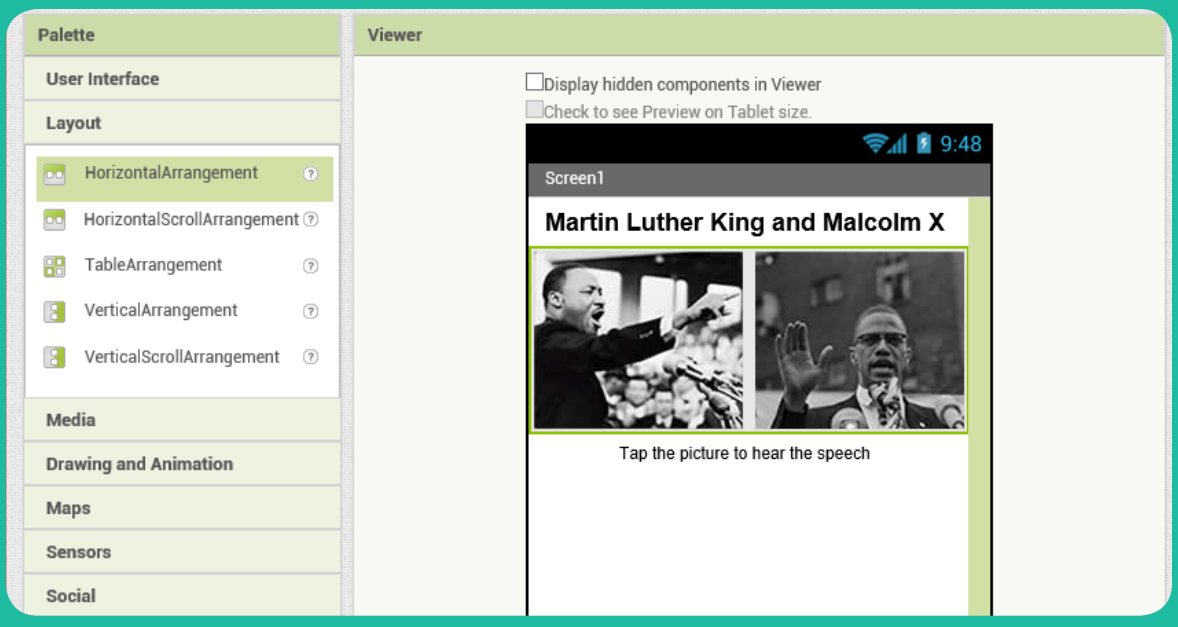
1 Before continuing, let's make sure that you have saved your progress so far. Click on the ***Projects*** tab and choose ***Save project as*** to select a new name for the current project. Choose any name you like, but make sure it is a name that makes it easy to recognize the file. Something like "IHaveADream2."



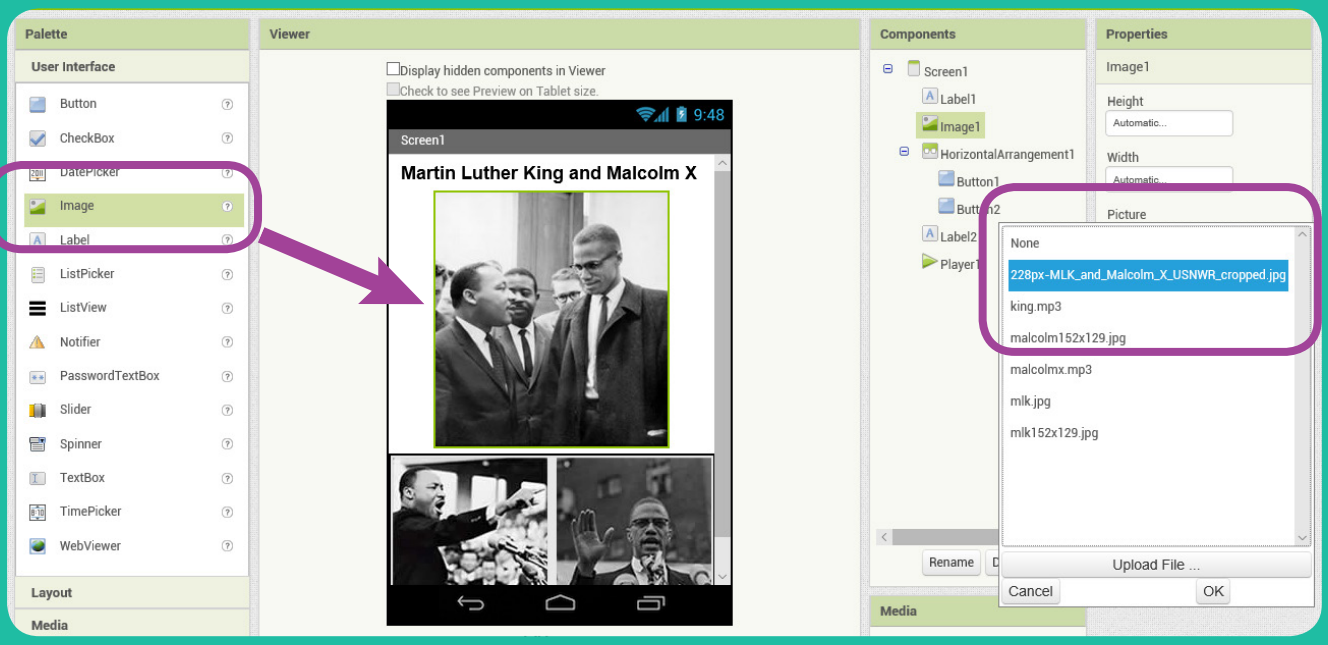
2 We want to add a second button for the other speech next to the first one. To do this, we'll need to do two things: replace the current image for the first button with a smaller one and add a horizontal layout component so that both buttons are side-by-side.

Select ***Button1*** and change the image to "mlk152x129.jpg" Drag out a second button from the ***User Interface Palette*** and set its image to "malcom152x129.jpg" and remove the default text. Finally, open the ***Layout Palette*** and drag the ***HorizontalArrangement*** component into the ***Viewer***. Drag both buttons into the ***HorizontalArrangement*** component.

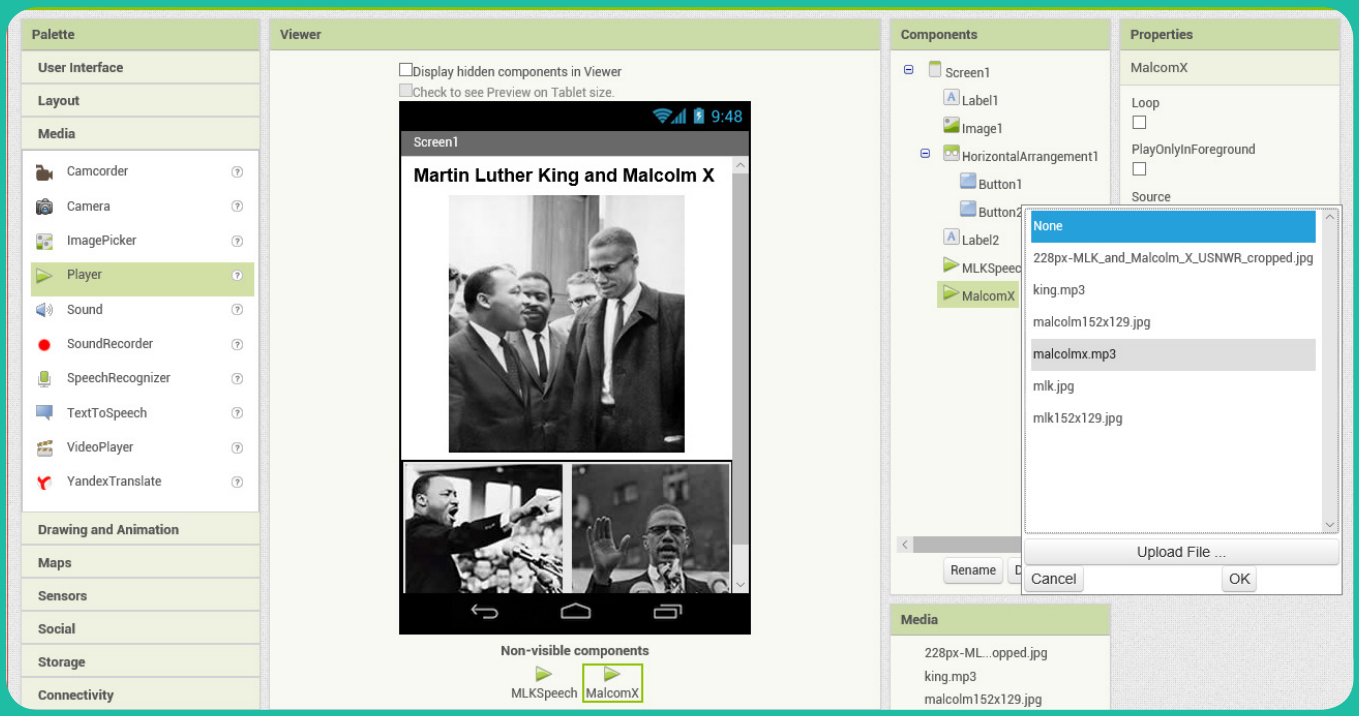
Change the top label to "Martin Luther King and Malcom X."



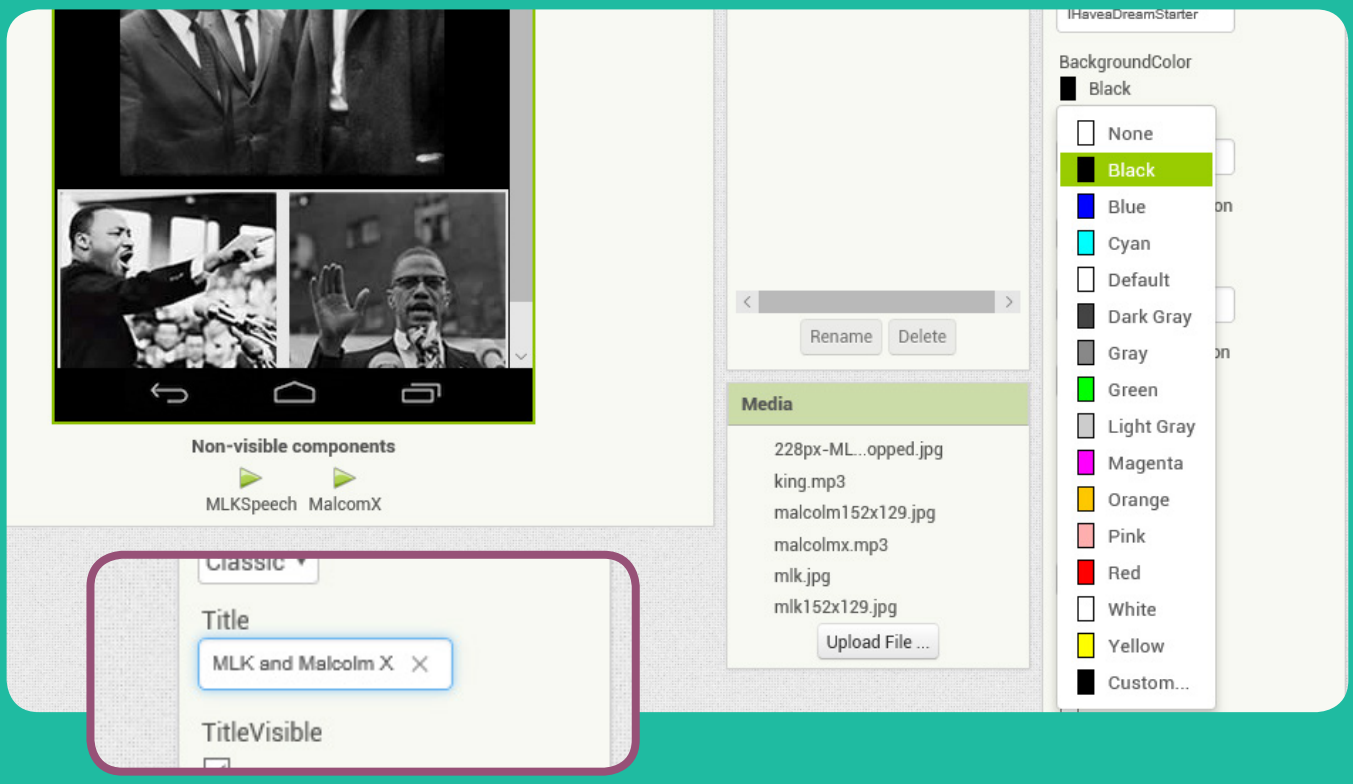
3 Since the focal point of our app (the large picture of MLK) is gone, we need to replace it. From the ***User Interface Palette***, drag out an ***Image*** component and place it underneath the top label. Choose "228px-MLK_and_Malcolm_X_USNWR_cropped.jpg" for the picture property. We are using the ***Image*** component because we don't need anything to happen if the user taps on it.



4 From the ***Media Palette***, drag a second ***Player*** component into the ***Viewer***. Set the source of this player to "malcolmx.mp3." Notice that in the ***Components*** section you now have ***Button1*** and ***Button2*** components and ***Player1*** and ***Player2*** components. This could get confusing. Select each of the four components in the section and click on Rename at the bottom to change the names to something more descriptive like "MLKButton" and "MLKPlayer."

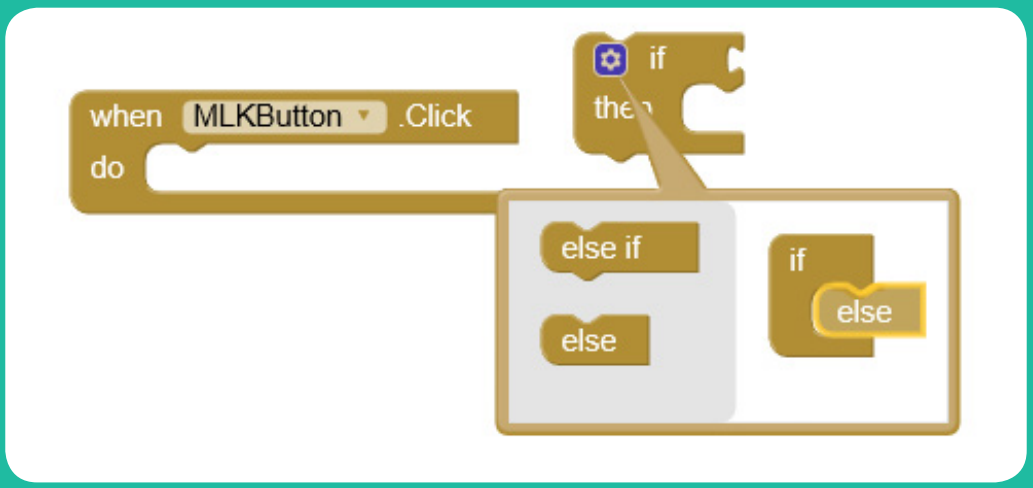


5 In the ***Components*** section, select ***Screen1*** and in the properties, change the ***BackgroundColor*** to black. Also in the properties, select the ***Title*** of the screen and change it to "MLK and Malcolm X." Now that the background is black, you can no longer read the labels. Select each label and change the ***TextColor*** to white.

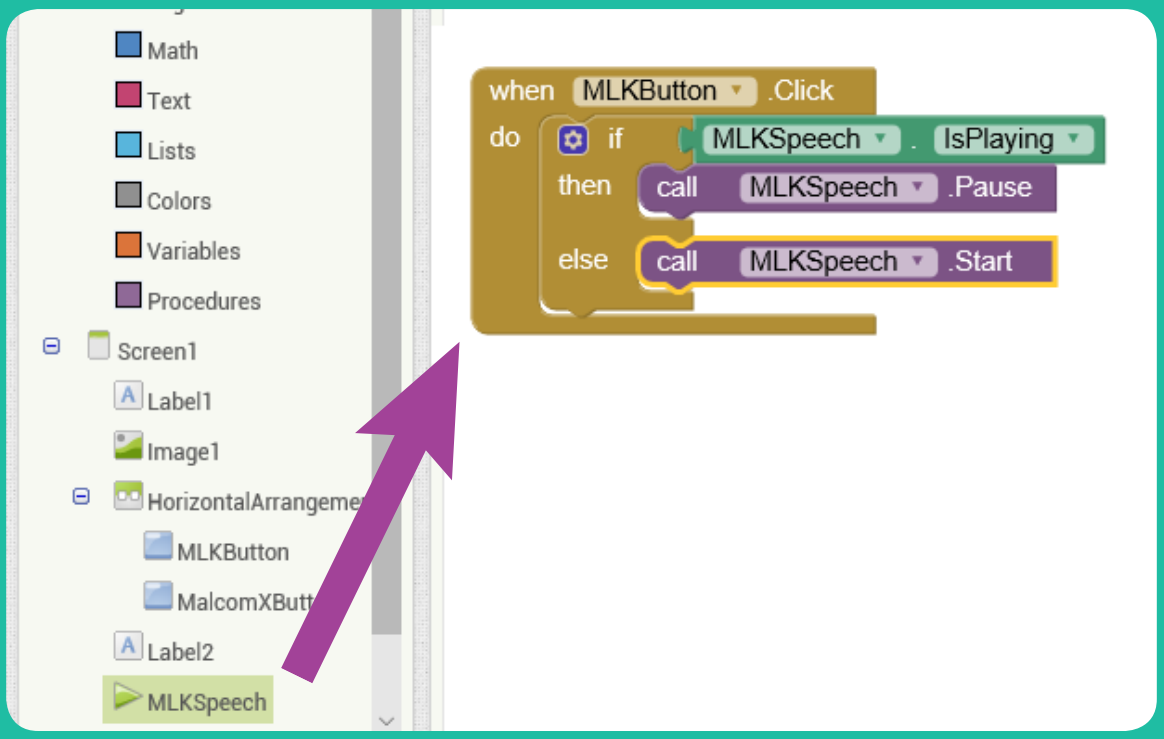


6 Switch over to the ***Blocks*** Editor. The event for ***Button1*** now has the new name that you gave to that component. We want to set this up so that when the user clicks on the button, it will check to see if the Player component is playing and either pause the player (if already playing) or start the player (if not.)

To do this, we need a *conditional*. In the ***Control*** section, drag an ***if then*** block into the ***Viewer.*** We need to add an additional choice - an "else" to do something else if the condition isn't true. To do this, click on the blue "modifier" icon in the upper left corner of the block. This opens a menu to let you add components to the block. In this case, drag the "else" modifier so that it is inside the if block in the modifier menu. This changes the ***if then*** block into an ***if then else*** block.

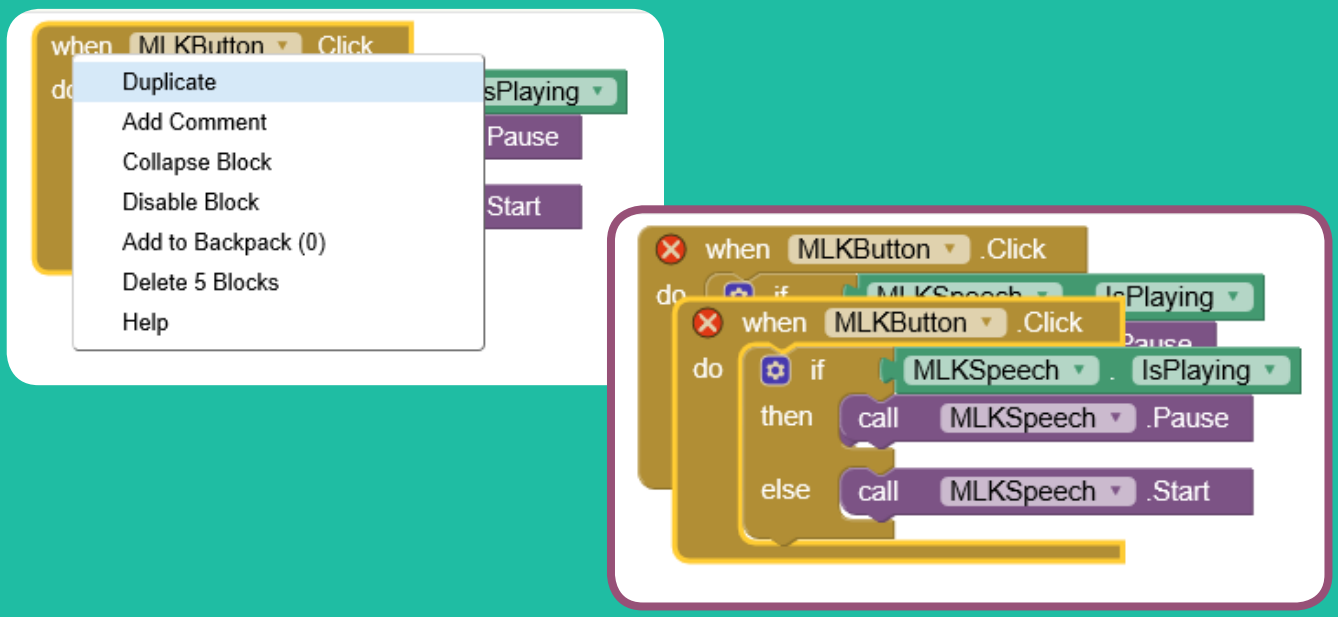


7 Place the ***if then else*** block inside the button event handler. We will then add three blocks from the player associated with that button. First, drag a ***MLKSpeech.IsPlaying*** (or whatever name you gave to Player1) block to the ***Viewer*** and connect it to the "if" connector. So if the Player *is playing*, then do something. In this case, we want to pause the player, so drag a ***call MLKSpeech.Pause*** block out and insert it next to "then." Finally, drag a ***call MLKSpeech.Start*** block into the "else" part of the conditional block to complete the event.

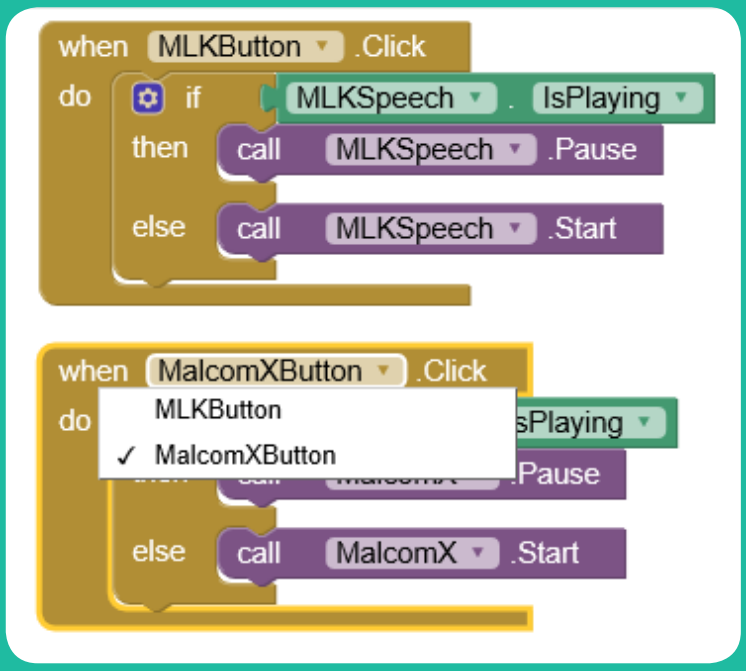


8 Your device should still be connected to App Inventor, so test it to see if the button starts and stops the speech.

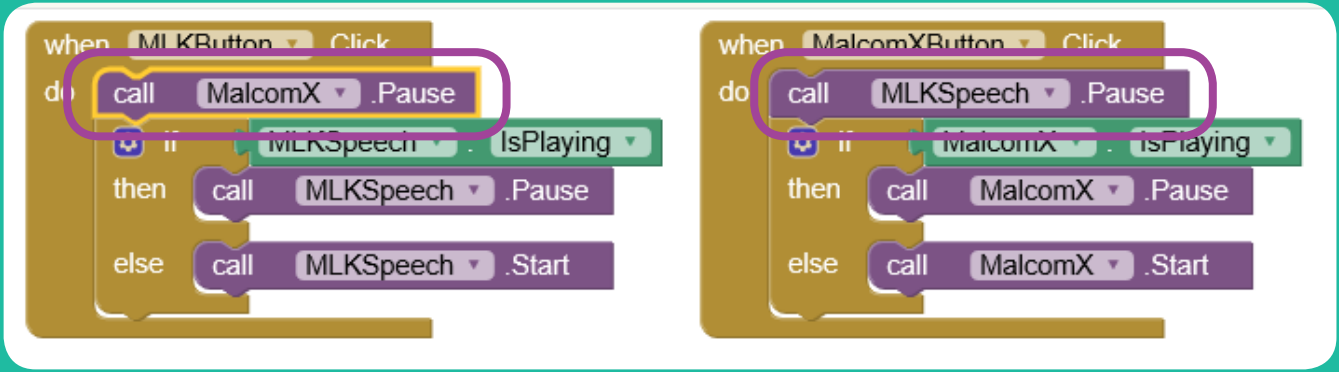
9 We want the same behavior for the other button, but rather than drag all of the blocks out one at a time, we can right-click on the ***when MLKButton.Clicked*** event and select ***Duplicate*** to make a copy of everything. Since we can only have one ***when MLKButton.Clicked*** event, App Inventor puts a red icon next to both to let you know that you need to fix something.



10 Click on the small arrow next to the parameter of the second button and change it to the ***MalcomXButton***. Do the same for the parameters of the MalcomX player.



11 We need to do one more thing. When the user presses on one of the buttons, we need to automatically pause the other player. (Don't worry if the player doesn't happen to be playing at the time. Pausing something that's already paused still won't start it). Duplicate the call MLKPlayer.Pause block and drag the copy over to place it inside the MalcomXButton event. Do the same with the MalcomX player pause block.



12 Test your app. Do the players start and stop when they are supposed to? This is an example of how many apps work. There is an event and a response to that event in the form of a sequence of blocks that are executed in the order in which they are stacked. Some of the blocks are only executed if the proper conditions are true.

Events

Think of an app like an event-response machine. The user clicks a button, the app responds. A text comes in, the app sends an auto-reply. A flying saucer hits an asteroid, it blows up. Your job as a designer is to program such event-responses. In App Inventor, you do this directly - your code is essentially a set of event-responses, of when-do blocks.

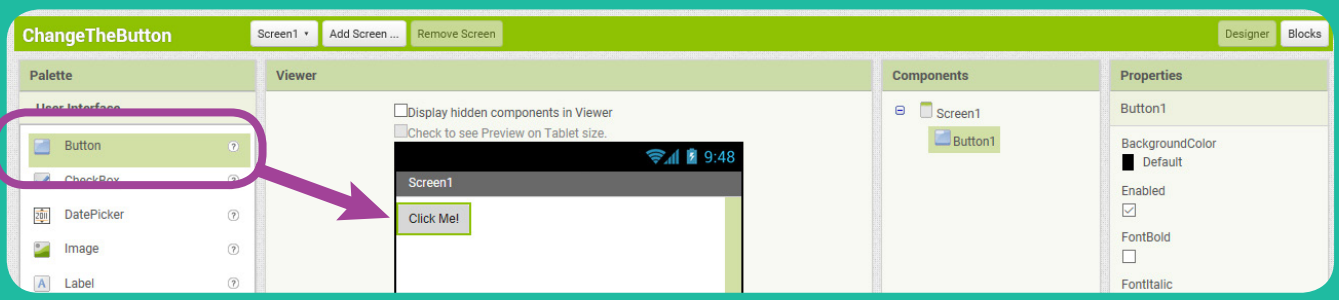
Change the Button

In this activity, try to do it on your own before checking the steps below. You will need to ***Start a New Project*** and simply put a button in it. You can change the button text to anything you like.

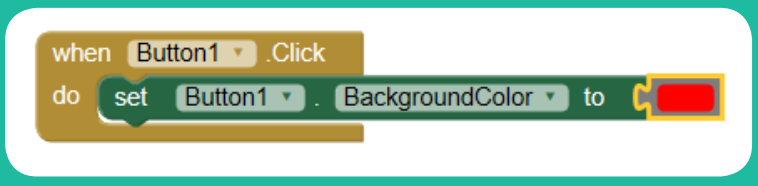
Edit the Blocks so that when the user taps on the button, the ***BackgroundColor*** property of that button changes to *red*.

Activity 01-05: Change the Button

1 Begin with a new project. In the Designer section, add a button from the User Interface Palette to the Viewer. Change the default button text to anything you want.



2 Switch to the Blocks Editor and add a when Button1.Clicked event to the Viewer. Drag a set Button1.BackgroundColor to block inside the event handler. Open the colors blocks menu and drag a red color block to the end of the block.



3 Feel free to explore other properties that you can change with the event handler.

Mediaboard

Now it's up to you to build a simple app called a "media-board." A mediaboard app has some image buttons and when each button is clicked, some media is played. For instance, showing several pictures of animals and plays the sound they make when the button is pressed. Or maybe a counting app. It doesn't have to be too sophisticated, but it must have at least:

- Four different buttons each playing a different media file (such as four different sounds).
- Use layout components such as Horizontal or VerticalArrangement
- Some decision making (an ***if then*** block).

2 Drawing and Animation

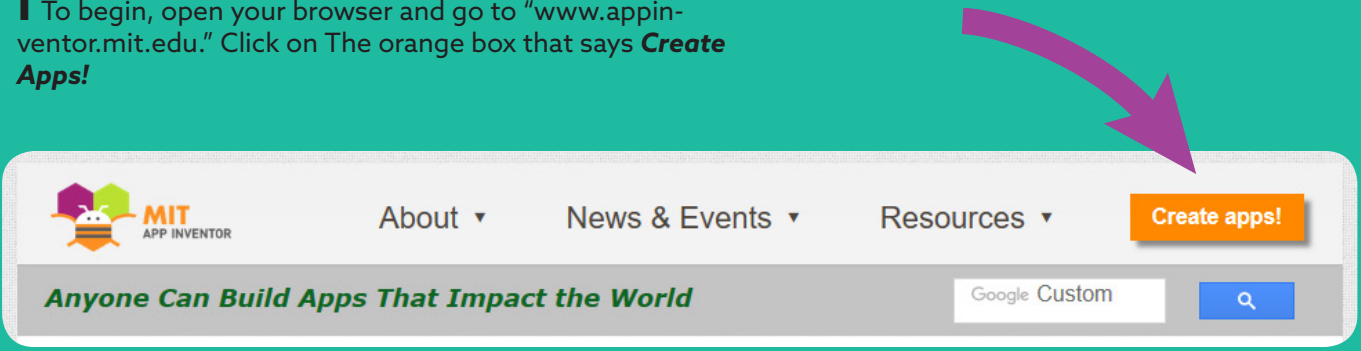
So far, we've been adding components to our screen in the form of buttons, images and labels. For the most part, these objects are static, meaning that changing them takes a bit of work and those changes are rather limited. What we need is a component that is designed for constant interactivity.

Introducing the Canvas

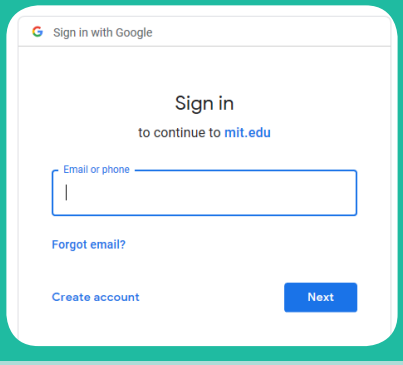
The canvas is a component that is designed specifically to allow you to draw shapes and images on your app screen. It's a lot like a special drawing board for drawing and interaction.

Activity 02-01: The Canvas

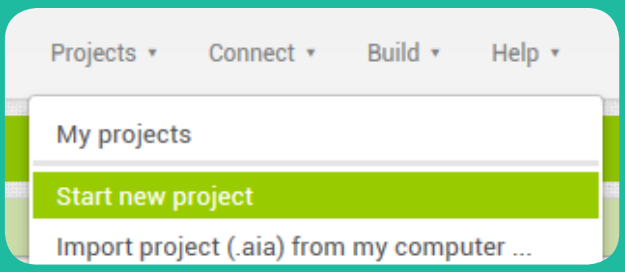
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



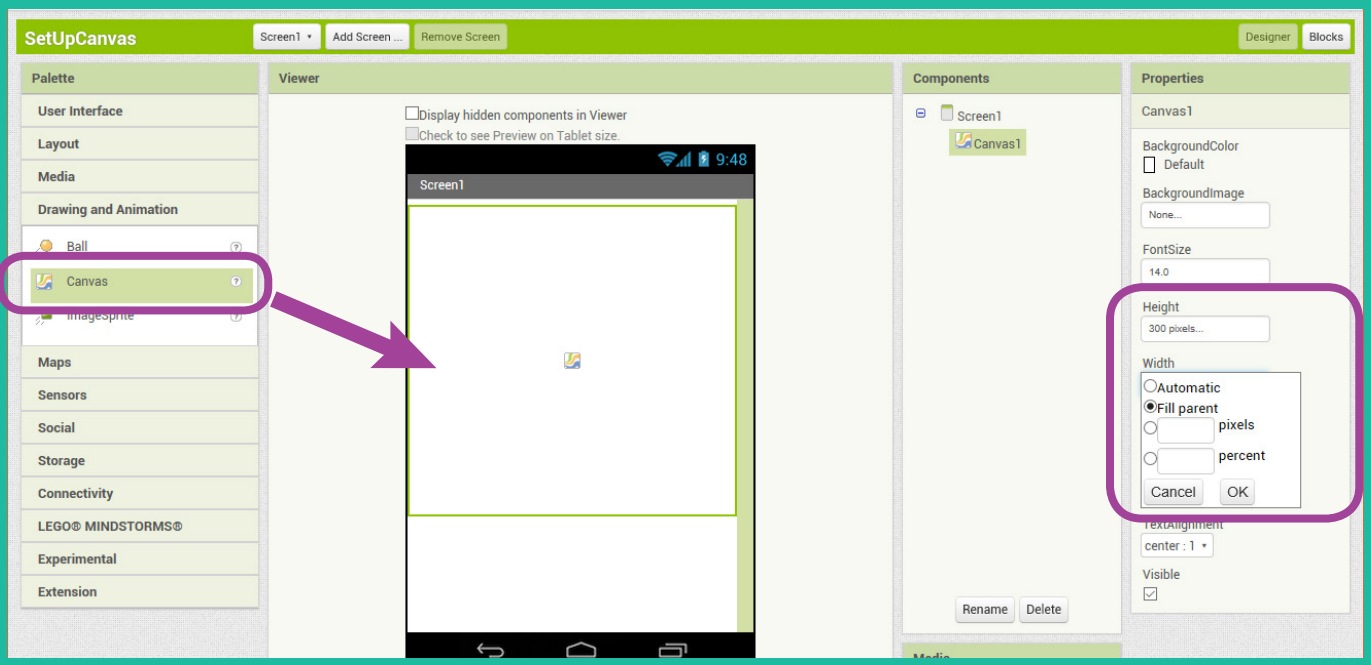
2 Remember, if you've been away for a while, you are going to need to sign in so that App Inventor can keep track of what you create.



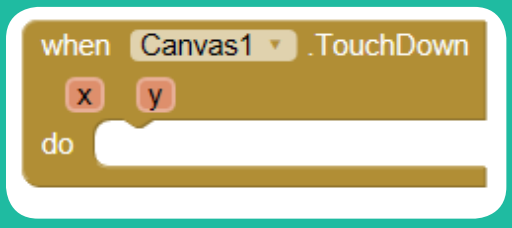
3 Select the ***Projects*** tab and click on ***Start New Project***. Give it a name like "MyCanvas."



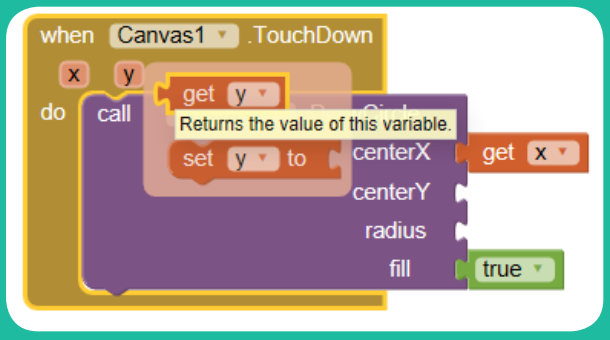
4 From the ***Drawing and Animation Palette***, drag a ***Canvas*** component into the ***Viewer***. Set the width property of the component to "fill parent" and the height to 300 pixels.



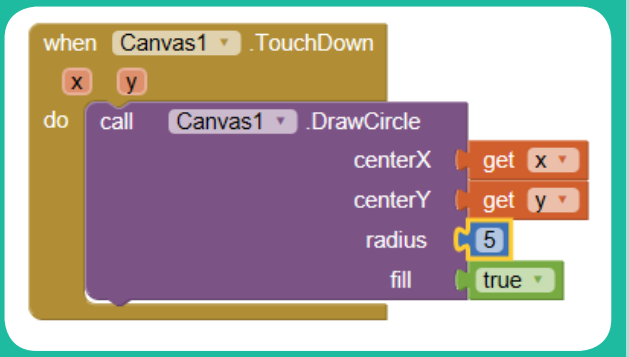
5 Switch to the ***Blocks*** Editor. Select the Canvas component and add a ***when Canvas1.TouchDown do*** event to the ***Viewer***. Notice the x and y parameters in the event? When the user touches the canvas, the x and y coordinates of where they touched is passed to the event.



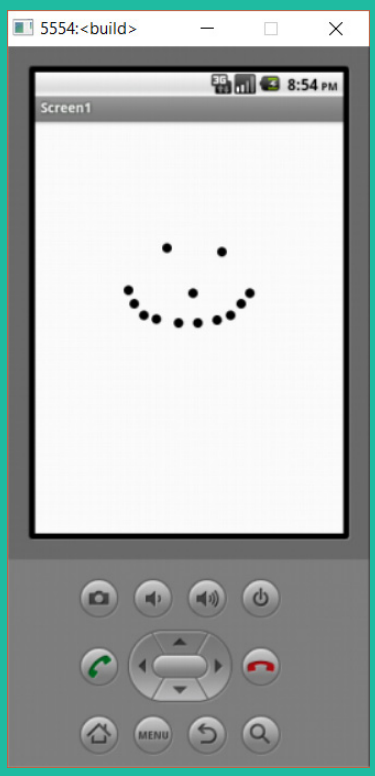
6 Select the Canvas component and add a ***call Canvas1.DrawCircle*** procedure to the ***Viewer***. The procedure needs to know where to draw the circle. In this case, we want to draw the circle where they just touched, so click on each the x and the y parameters in the orange boxes and select the ***get x*** and ***get y*** blocks and put them in the appropriate slots in the procedure.



7 The procedure needs to know how big to make the circle, so click on ***Math*** and drag a number block (it has a 0 in it) into the slot for radius. Change the number to something larger than 0.



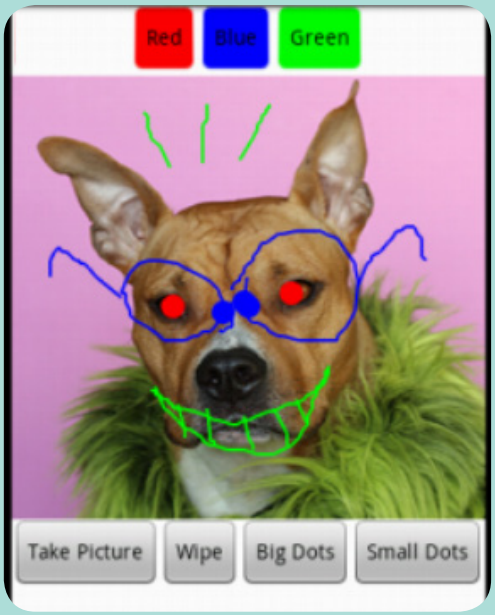
8 Now test your app. Are you able to make circles? What would you do if you wanted to make circles of different sizes?



Doodling

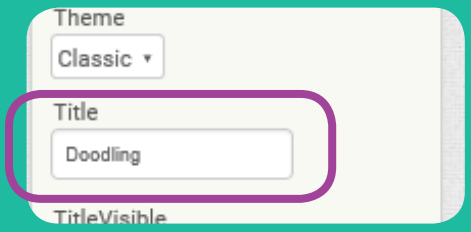
Now that you've gotten your feet wet with the canvas, let's create an app for drawing and making funny pictures.

This app lets you change colors, draw lines and make dots in the size you want. You can clear all of your doodles with the touch of a button and even use the camera (on your mobile device) to take a new picture to doodle on.



Activity 02-02: Doodling

1 Begin with a new project. Give it a name like "Doodling." App Inventor opens in the Designer section. Make sure that ***Screen1*** is selected and go to properties and change the ***Title*** to "Doodling." Giving the screen a specific title helps confirm what app the user has open. If you are simply looking at a white background and "Screen1," you may not remember what app you opened.



2 This app will use the following components:

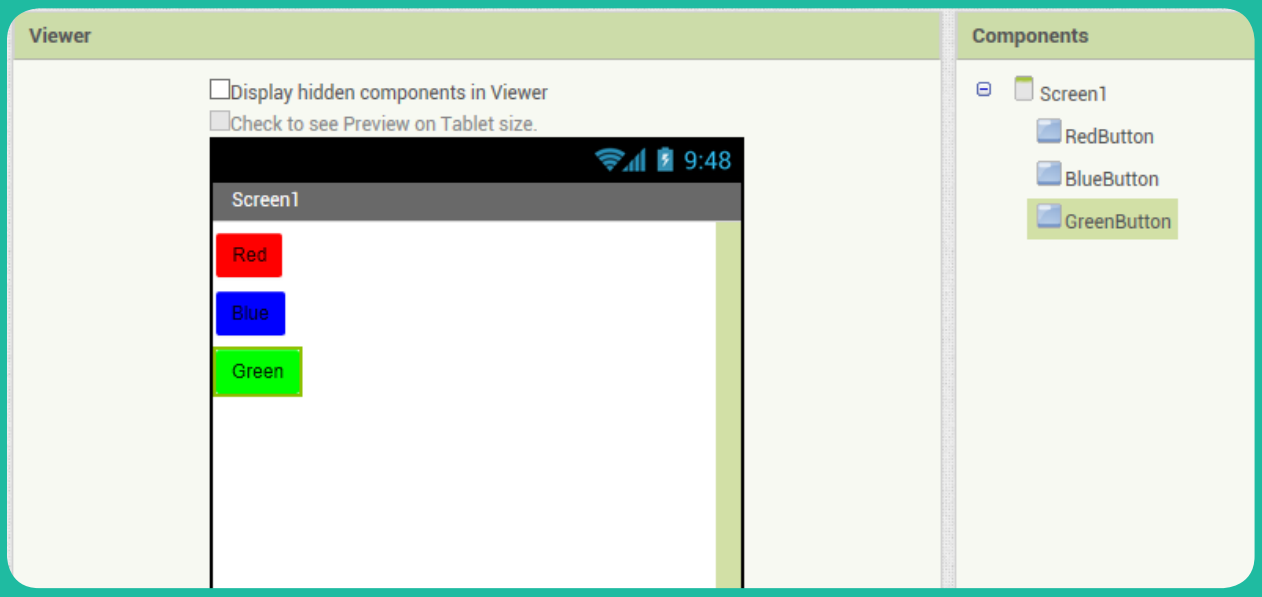
- Three buttons for selecting red, blue or green paint in a horizontal layout.
- A single button for clearing the screen, two buttons for changing dot size and one for using the camera to take a picture.
- A canvas component for drawing. The canvas has a background image (we're using the dog from the Talk to Me app, but you can use anything you want.

Names

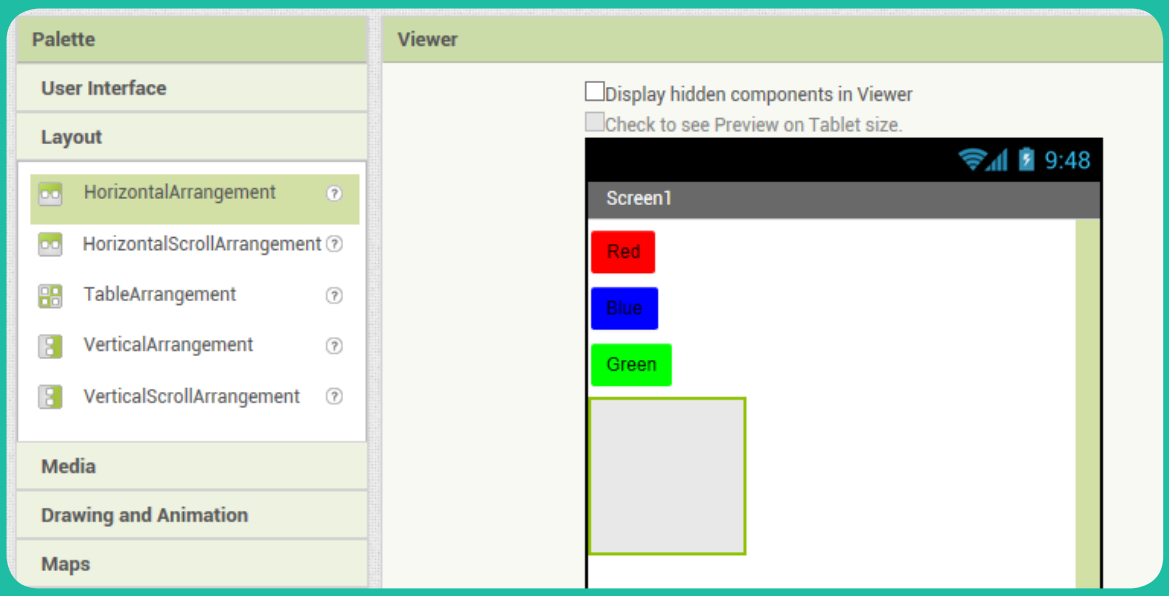
Don't worry about confusing your project name and the screen name. There are three key names in App Inventor:

- The name you chose for your project as you work on it
- The component name, Screen1 (which cannot be changed)
- The title of the screen which is what you will see in the App's title bar.

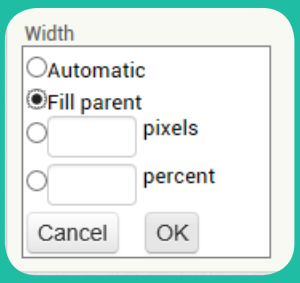
3 First, let's create the color buttons. Drag a button from the ***User Interface Palette*** into the viewer. Change the name of the button component to "RedButton." Change the button text property to "Red" and the button background color property to the color red. Do this two more times for the blue and green buttons.



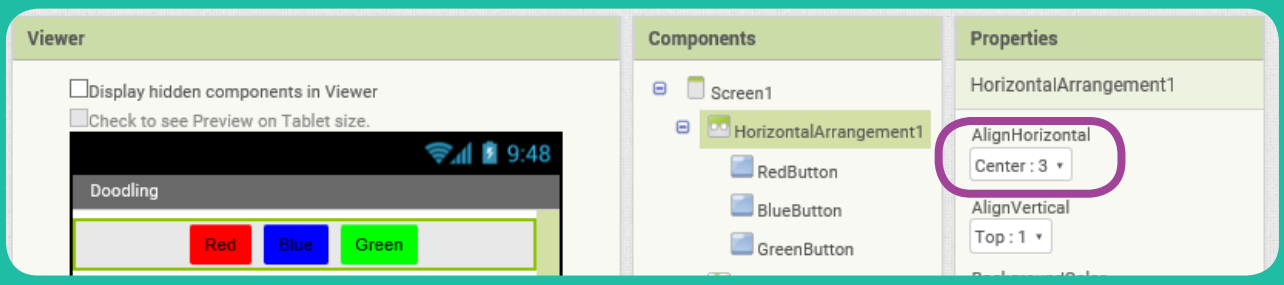
4 The only problem with the buttons is that they're taking up a lot of space stacked up like that. We need to set them side by side. From the ***Layout Palette***, drag a ***HorizontalArrangement*** component into the viewer.



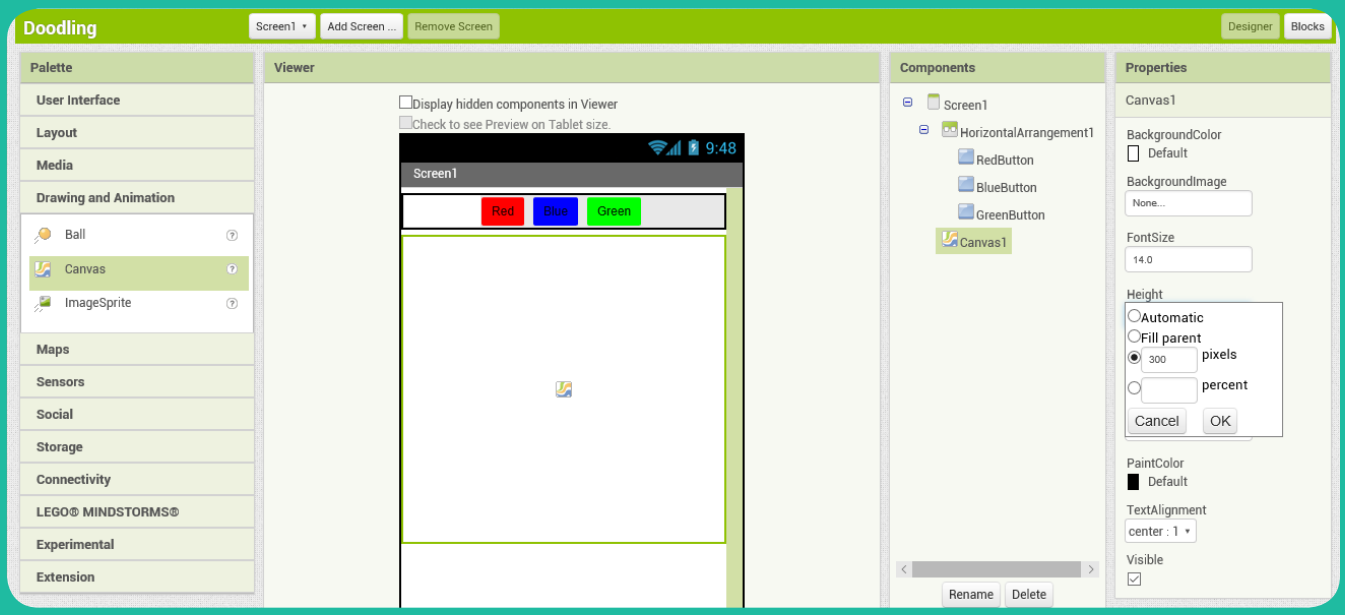
5 Change the ***width property*** of the horizontal arrangement component to "Fill parent."



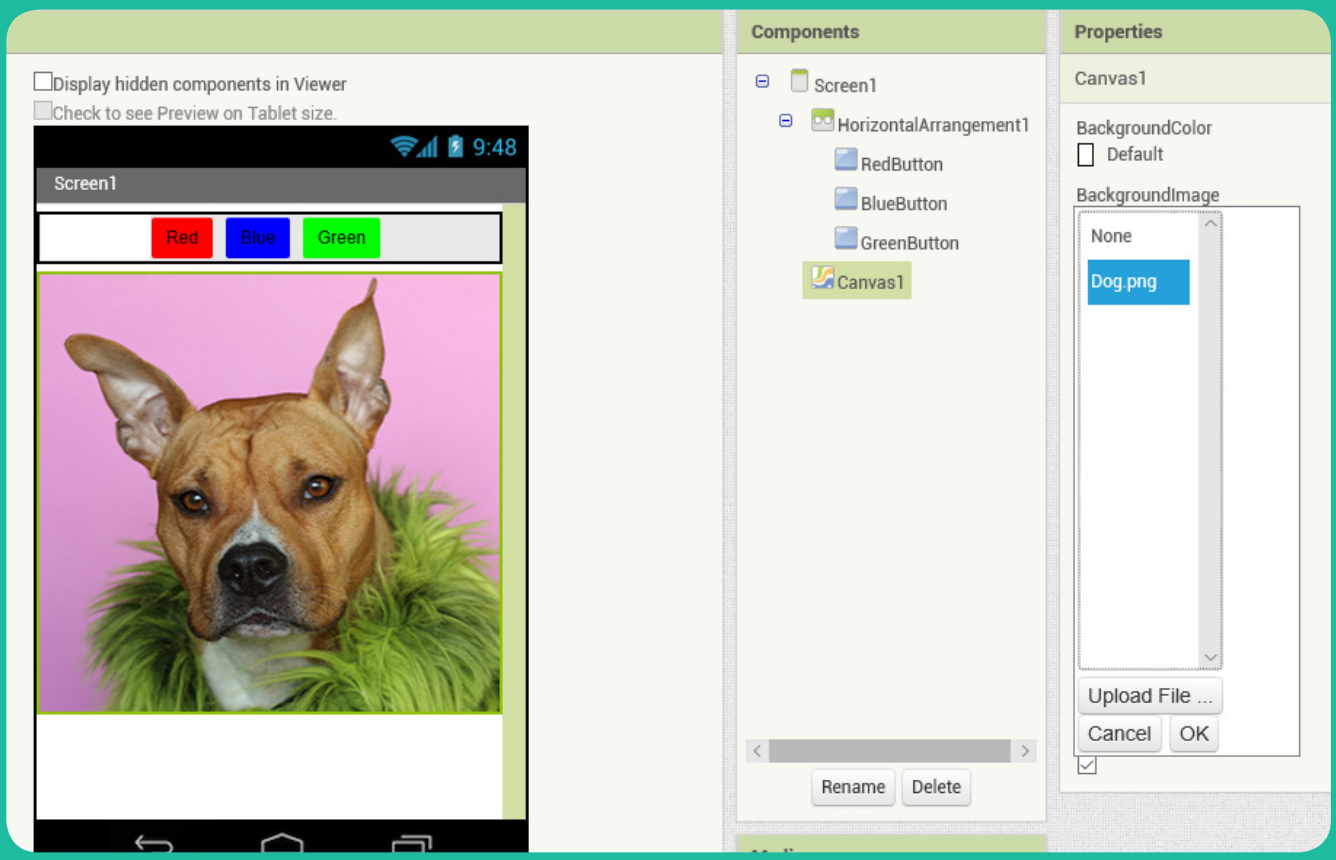
6 Drag each of the color buttons into the ***HorizontalArrangement*** component. You can center the row of buttons by selecting the ***HorizontalArrangement*** component and selecting the ***AlignHorizontal*** property and clicking on ***Center***.



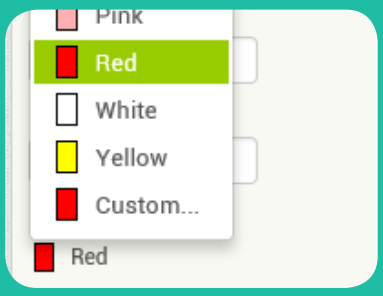
7 The next step is to set up the canvas for your drawing. Open the ***Drawing and Animation Palette*** and drag the ***Canvas*** component into your viewer beneath the row of buttons. Set the width property of the component to "Fill parent" and the height to 300 pixels. Change the name to "DrawingCanvas."



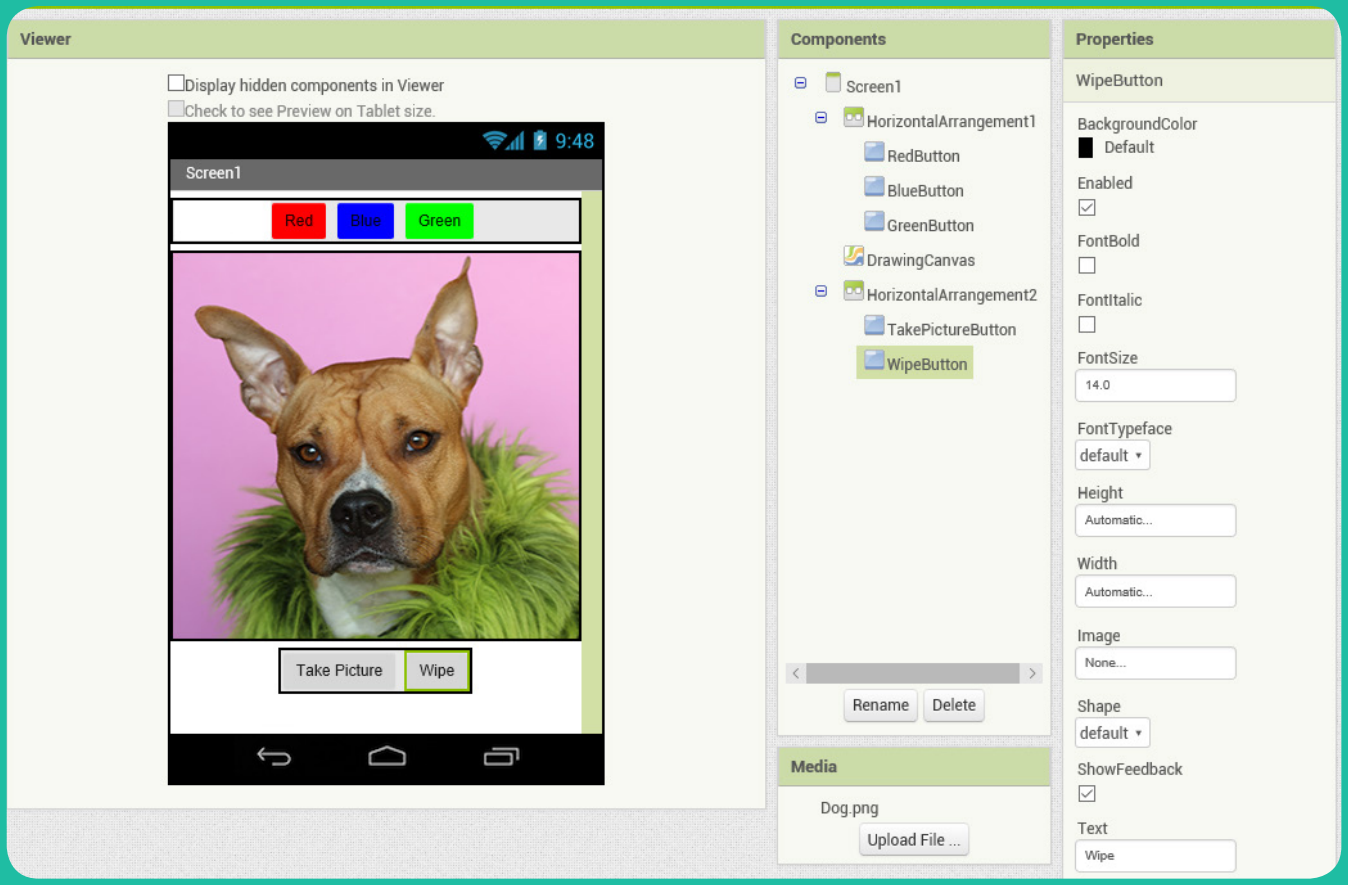
8 Click on the BackgroundImage property of the Canvas component to open the upload menu and upload an image for the Canvas. We're using "dog.png"



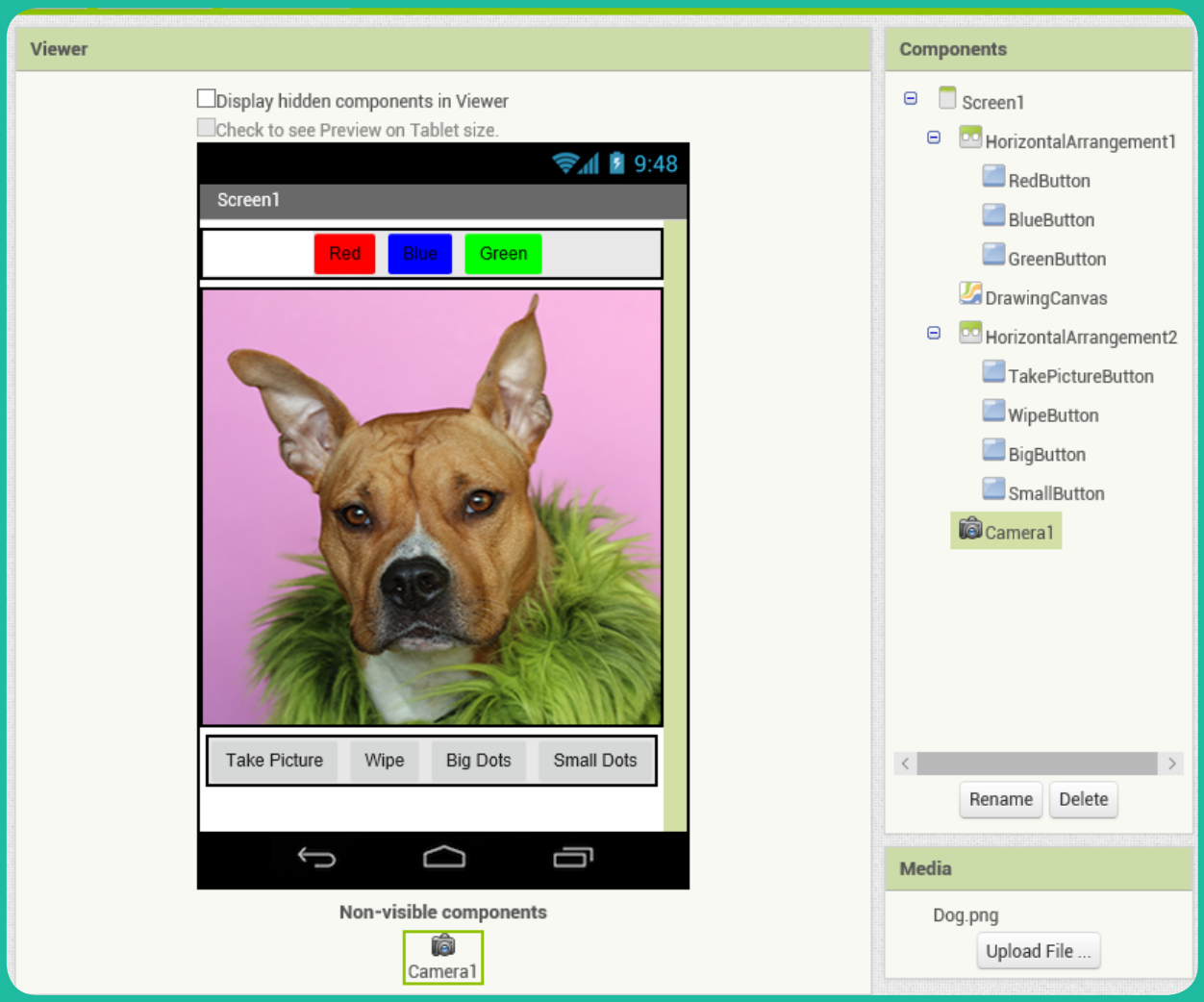
9 Set the ***PaintColor*** property of the Canvas to red so that when the app is first launched, but no color has been selected, the default color will be red.



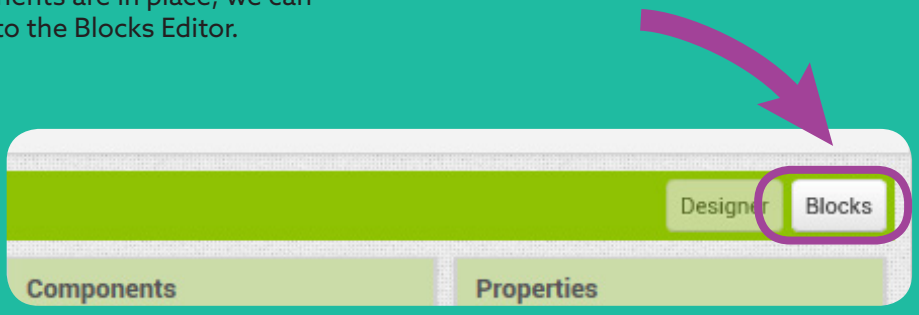
10 Now let's add the rest of the components. From the ***Layout Palette***, drag a new ***HorizontalArrangement*** component into the viewer and place it under the canvas. Drag two buttons from the ***User Interface Palette*** into the ***HorizontalArrangement*** component. Rename the first button to "TakePictureButton" and set its text property to "Take Picture." Rename the second button to "WipeButton" and change its text property to "Wipe."



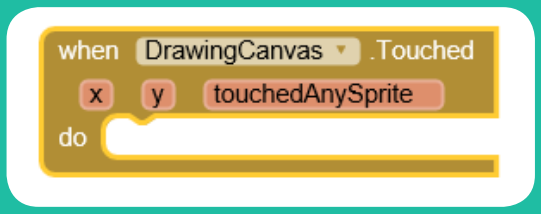
11 Drag two more buttons into the ***HorizontalArrangement***, placing them next to the Wipe button. Rename the first button to "BigButton" and change the text property to "Big Dots." Rename the second button to "SmallButton" and change its text to "Small Dots." Finally, drag a ***Camera*** component from the ***Media Palette*** into the viewer. Your project should look like this:



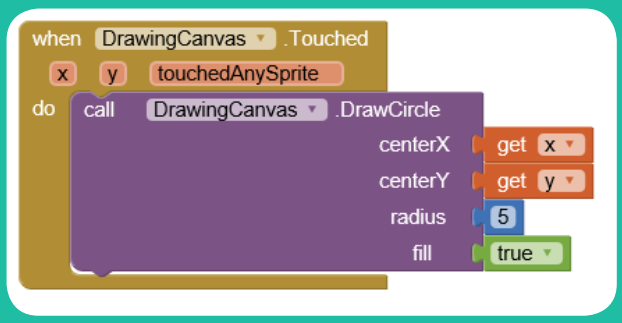
12 Now that all of our components are in place, we can set up their behaviors. Switch to the Blocks Editor.



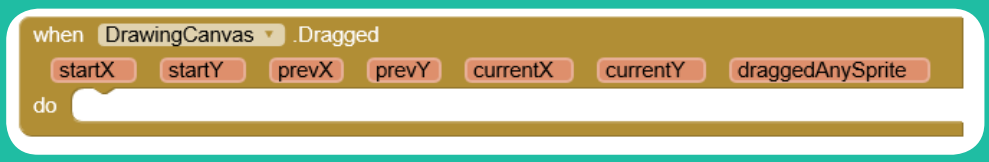
13 All of the events will affect our ***DrawingCanvas*** component. When the canvas is touched, we will place a dot. When the user drags on the canvas, we will create a line. To get started, select the ***DrawingCanvas*** component and drag a ***when DrawingCanvas.Touched*** event into the viewer.



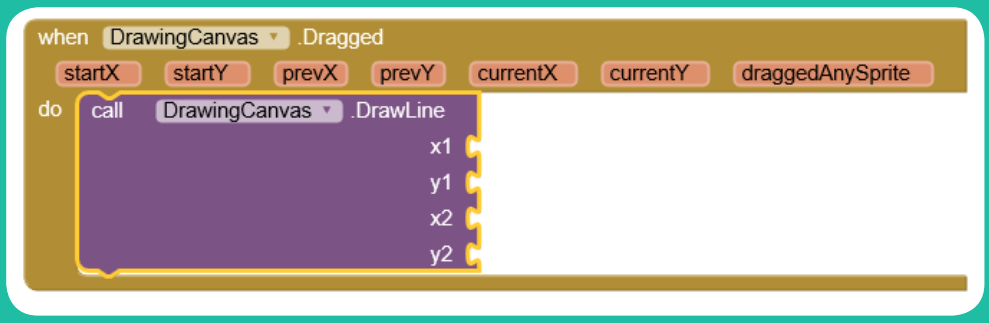
14 Select the ***DrawingCanvas*** component again and drag a ***call DrawingCanvas.DrawCircle*** block into your event handler. You're going to set this up exactly like you did with activity 02-01, using the x and y parameters for the coordinates of the circle and a value of "5" for the radius.



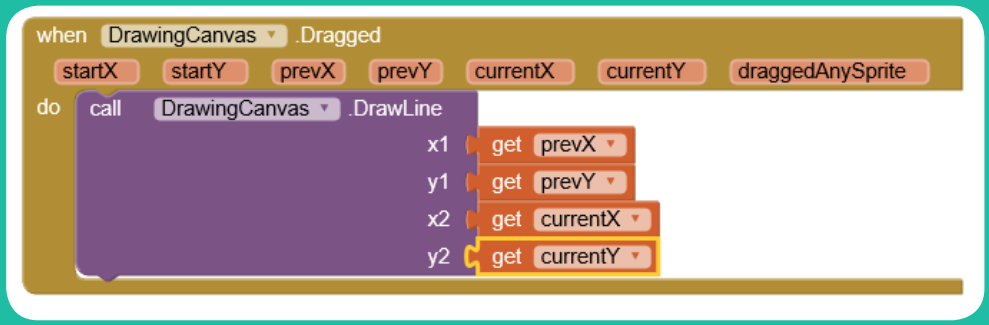
15 We also want to be able to draw a line in the canvas. Select the ***DrawingCanvas*** component again and drag a ***when DrawingCanvas.Dragged*** event into the ***Viewer***. This is similar to the Touched event, but it has a few more parameters. It has ***startX*** and ***startY*** to remember the point where you started your drag. There is also ***currentX*** and ***currentY*** to indicate the point where the drag is currently at. Finally, there are parameters for ***prevX*** and ***prevY*** to record the last position you were dragging from.



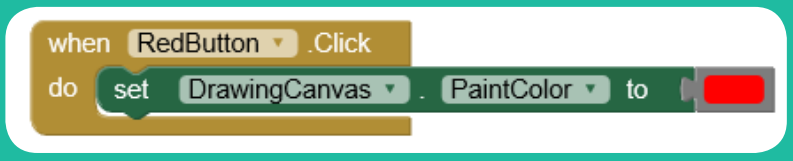
16 Select the ***DrawingCanvas*** component again and drag a ***call DrawingCanvas.DrawLine*** block into your event handler.



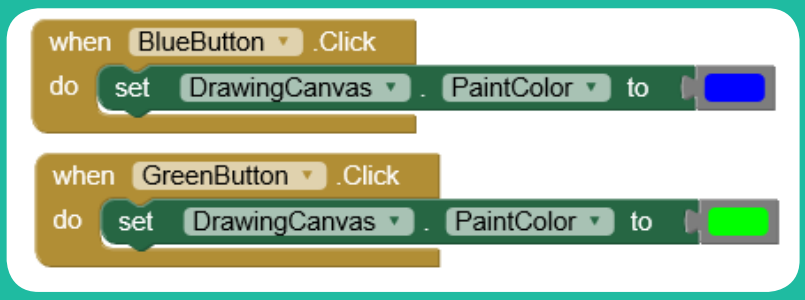
17 When the user is dragging across the canvas, we want to draw a line from where they just were (not the orignal start point) to where they are now. So for the x1 and y1 slots we want ***prevX*** and ***prevY*** and we want ***currentX*** and ***currentY*** for the last two slots.



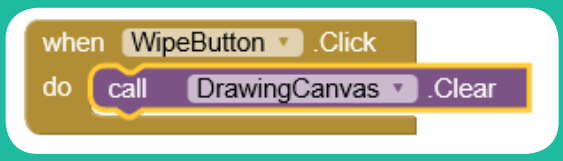
18 Now our app can draw dots and lines, but only in the default color of red. We should set up the other colors as well so that when the user clicks on one of the color buttons, the ***DrawingCanvas PaintColor*** property is changed. Start by selecting the ***RedButton*** component and dragging out a ***when RedButton.Clicked*** event. Add to that a ***set DrawingCanvas.PaintColor to*** block and add a ***red*** block from the ***Color*** components.



19 Duplicate this block two times by right-clicking on it. Change the parameters for Blue and Green as seen below.

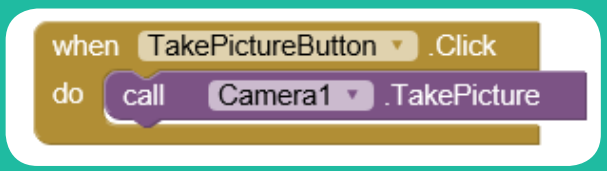


20 Finally, select the ***WipeButton*** component and add a ***when WipeButton.Clicked*** event. Add ***call DrawingCanvas.Clear*** block to this event to clear all of your drawing (but not the background image).

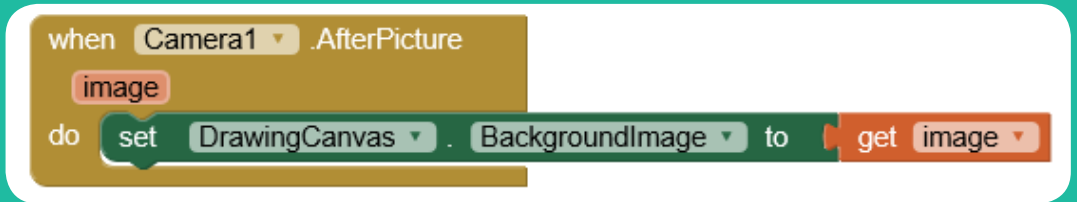


NOTE: EVEN THOUGH THE EMULATOR CANNOT USE THE CAMERA, WE ARE INCLUDING THE STEPS FOR TAKING A PICTURE WITH THE APP SO YOU KNOW HOW TO DO IT.

21 Select the ***TakePictureButton*** component and drag a ***when TakePictureButton.Clicked*** event. Add ***call Camera1.TakePicture*** block to this event to activate the camera component and have it take a picture.

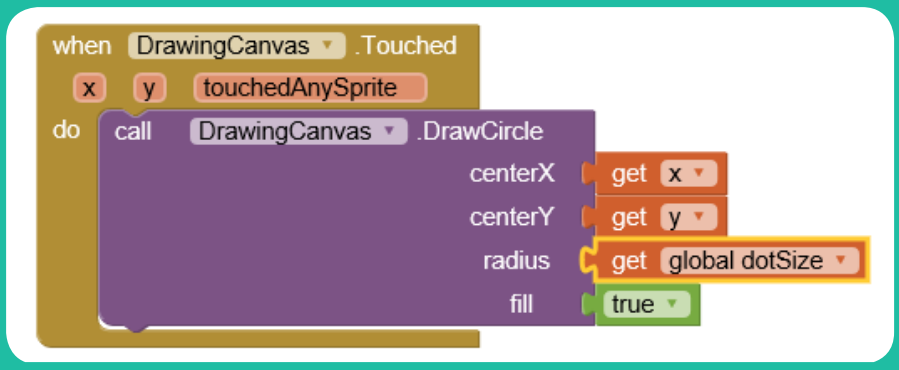


22 Once the camera component has taken a picture, we need to tell it what to to with the picture it just took. Select the ***Camera1*** component and drag a ***when Camera1.AfterPicture do*** event into the viewer. Select the ***DrawingCanvas*** component and drag a ***set DrawingCanvas.BackgroundImage to*** block into the event. The Camera event has a parameter for the image that the camera took, so drag that block over to the slot in the set block.



23 All that's left to do is set up the blocks that change the dot size. Remember, the dot size is determined by the radius parameter of the DrawCircle procedure. What we need is a placeholder for the current radius that we can change to any number we need. For that, we need a variable. Click on the ***Variables*** component and drag an ***initialize global name to*** block into the ***Viewer***. The first thing we need to do is change the default name to something just for our variable. Let's call it ***dotSize*** and give it a value of 2.

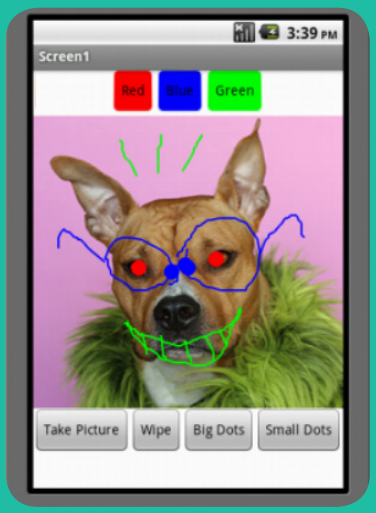
24 We need to apply that variable to our ***DrawCircle*** procedure. Remove the current block in the radius slot and replace it with a ***get global dotSize*** block.



25 Select the ***SmallButton*** component and drag a ***when SmallButton.Clicked*** event into the viewer. When the user clicks on this button, we want a small number for the radius. Add a ***set global dotSize to*** block and give it a number of 2.

26 Duplicate the event and change the parameters so that when the ***BigButton*** is clicked, the ***dotSize*** variable is changed to 8.

27 Now use the ***Connect*** tab to test your app. Do all of the buttons work?



Improving the App

Currently, the user has no idea what color or dot size they are using. How could you change it so that there was always a visual representation of the color and dot size?

Also, would it be possible to have more variety in the dot size? Try experimenting with the slider component from the User Interface Palette.

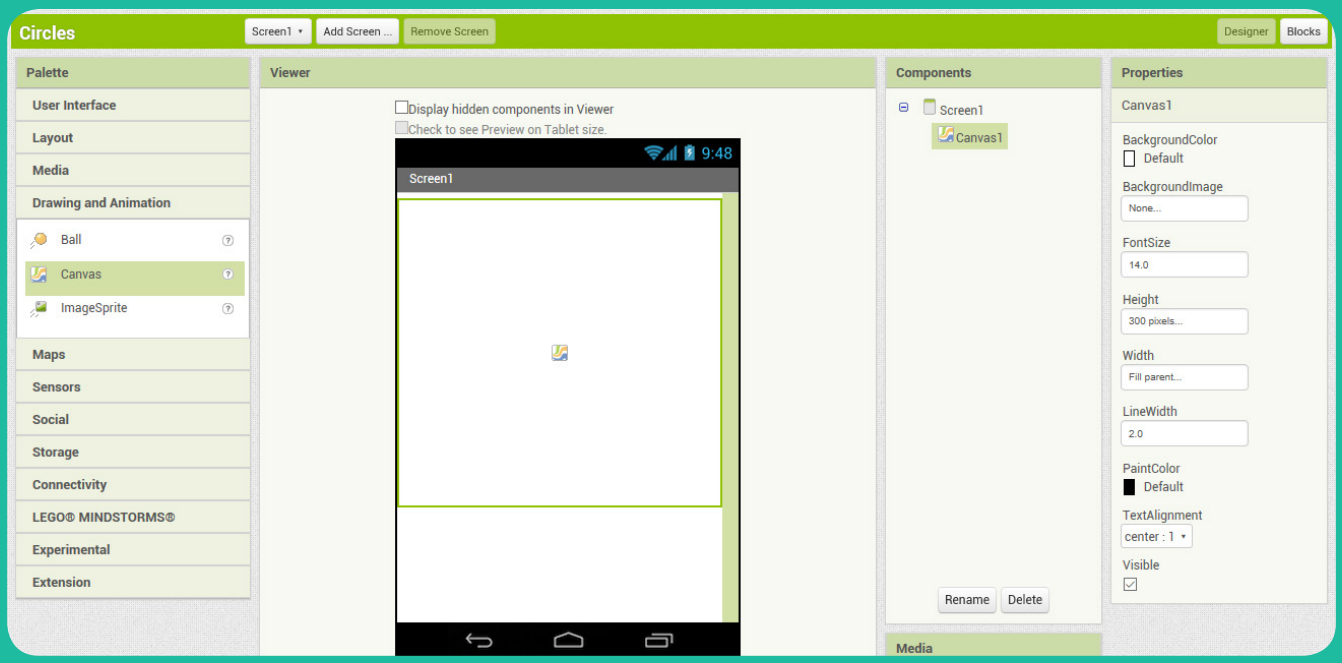
Variables

In the last activity, we used a a variable to keep track of the different sizes that we wanted to create a circle in. . A variable is kind of like a placeholder for the current value of something. As the name implies, a variable can change, meaning the places in the code where it is used can produce different results.

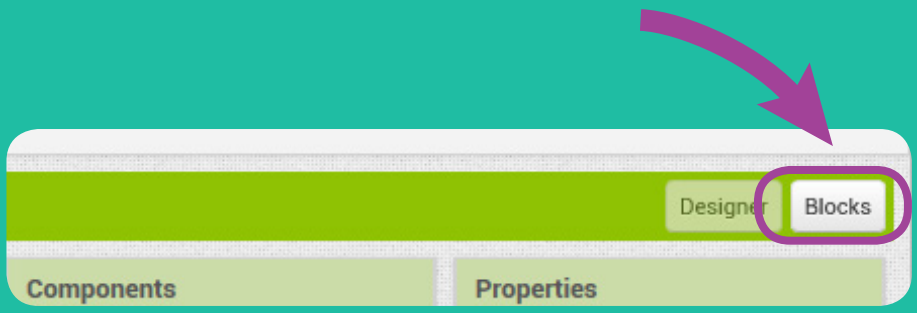
In this next example, we will once again use a variable for the radius of a circle. And we will use that variable to increase the size of the circle everytime we draw a new circle!

Activity 02-03: Circles

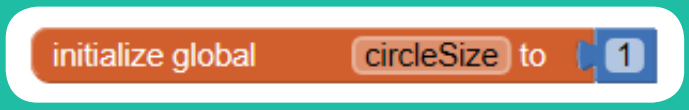
1 Begin with a new project. Give it a name like "Circles." In Designer, open the ***Drawing and Animation Palette*** and add a ***Canvas*** component to the viewer. Change the ***Width*** property to ***Fill parent*** and change the ***Height*** to "300" pixels.



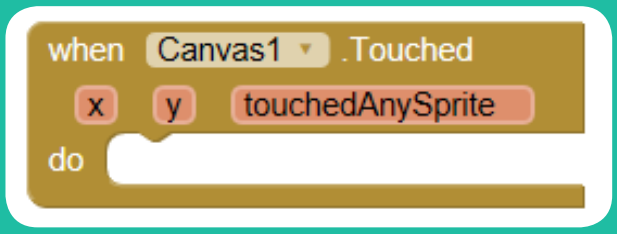
2 Switch to the Blocks Editor.



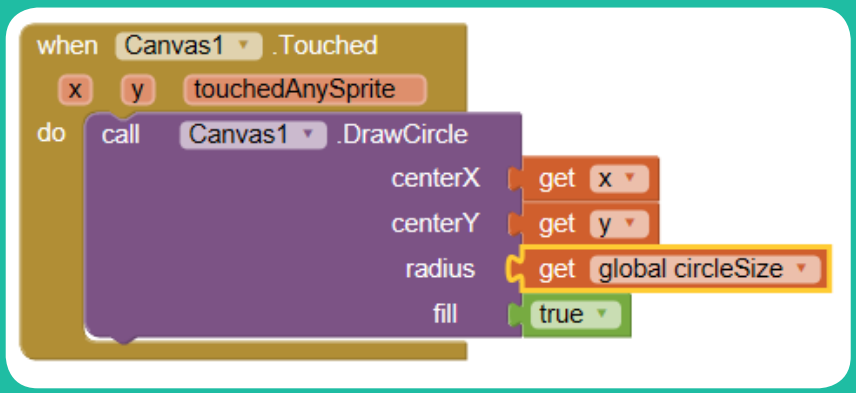
3 Variables must be "initialized" before you can use them. Select the ***Variables*** component and drag an ***initialize global name to*** block into the viewer. "Global" means that the variable can be used in all parts of the program. Some variables, particularly those used by ***Procedures***, can only be used within a specific block. Give the variable a name of "circleSize" and add a number block of "1" to the variable block.



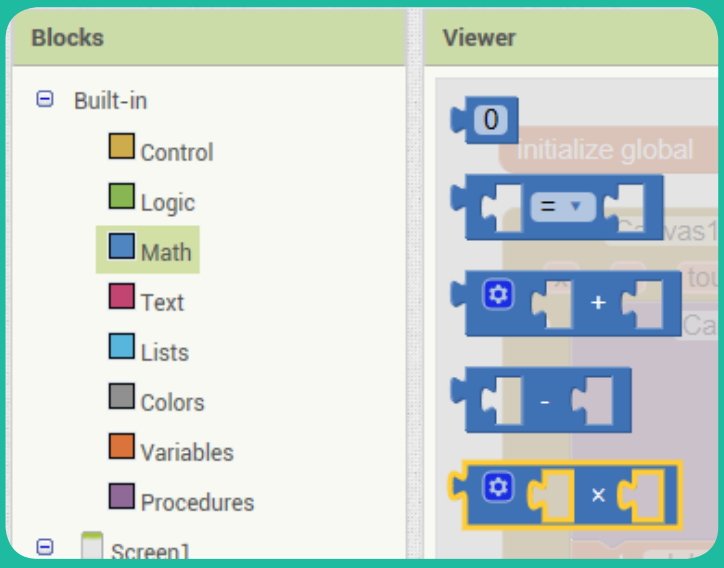
4 Select the ***Canvas1*** component and add a ***when Canvas1.Touched do*** block to the Viewer.



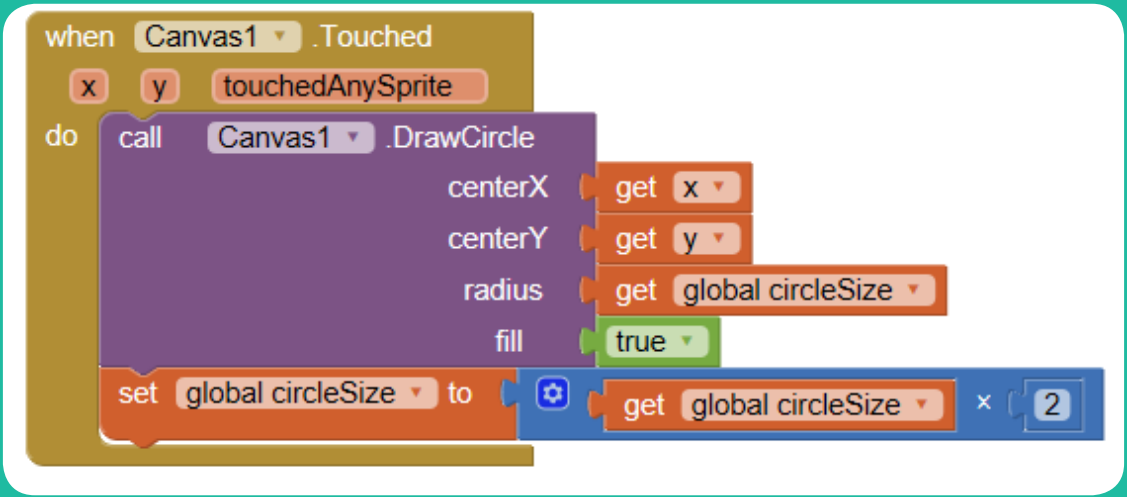
5 Select the ***Canvas1*** component again and add a ***call Canvas1.DrawCircle*** block to the event. Use the event parameters for the x and y slots and the ***circleSize*** variable for the radius.



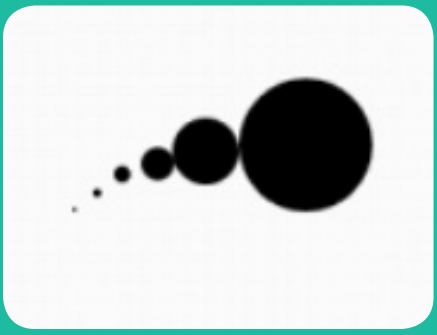
6 Right now, the app simply draws a circle with a radius of one pixel. However, we can use math to change the radius every time a circle is drawn. Add a ***set global circleSize to*** block inside the event, just after the ***DrawCircle*** procedure. Remember, blocks are executed from top to bottom and we do not want to change the variable until after the circle is drawn. From the ***Math*** component, add a multiply block as shown.



7 Each time the user touches the canvas, we want to draw a circle and then double the radius. We do this by inserting a get global circleSize block and a number block for 2.



8 Now use the ***Connect*** tab to test your app. What happens?



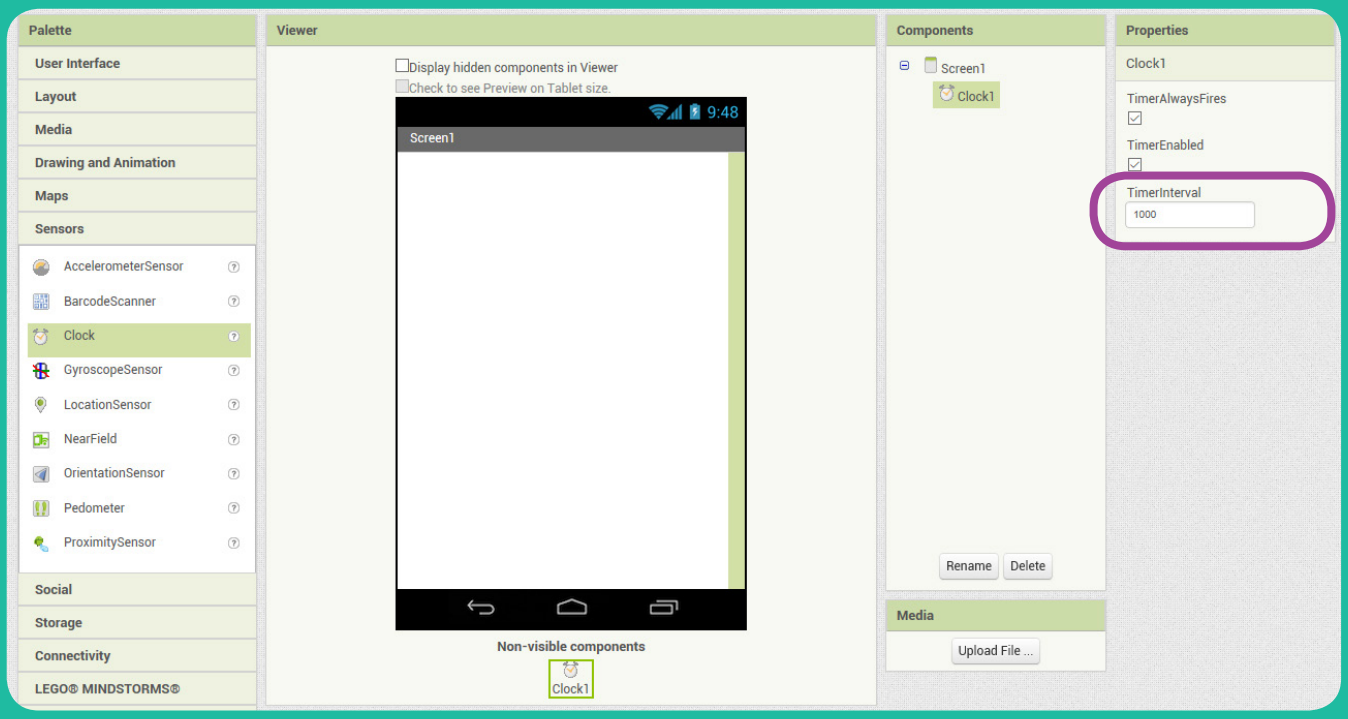
Using the Clock Component

As far as your app is concerned, everything happens at once. But sometimes you want the app to wait a bit.

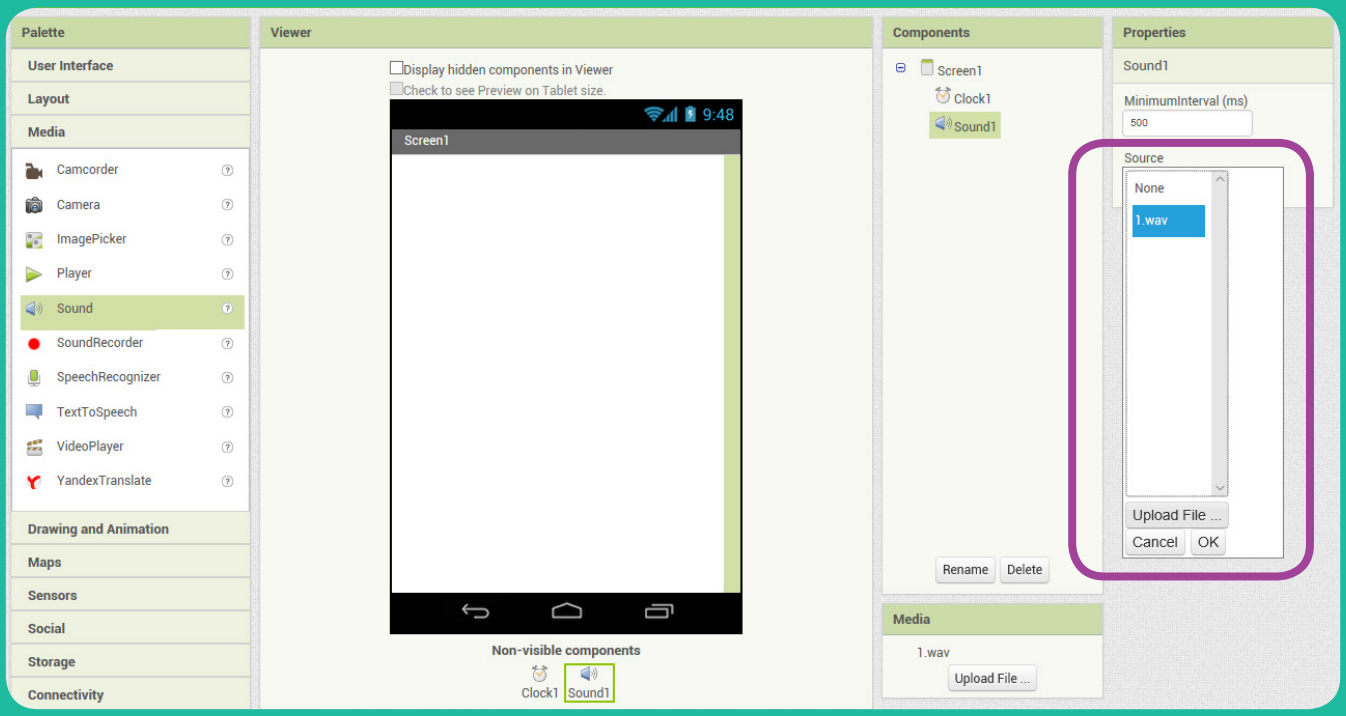
App Inventor has a Clock Component that is an event that does something every time the clock timer tells it to. Give this activity a try:

Activity 02-04: The Clock

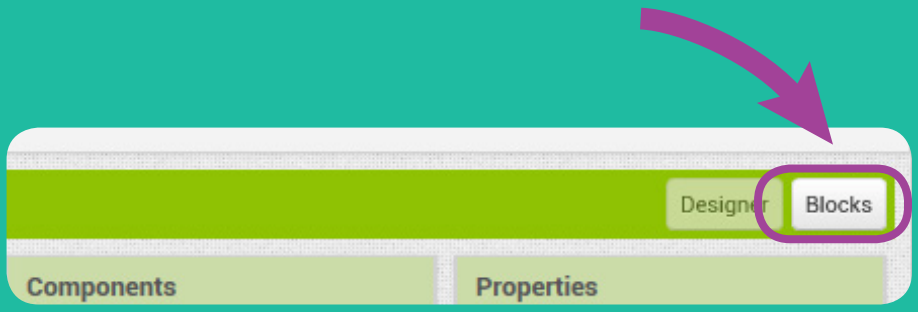
1 Begin with a new project. Give it a name like "Clock." In Designer, open the ***Sensors Palette*** and add a ***Clock*** component to the viewer. Notice that the ***TimerInterval*** property is set to "1000." That number is how many milliseconds will pass before the timer triggers an event. 1000 miliseconds is the same as one second.



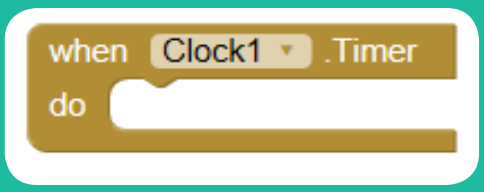
2 We want the app to play a sound every second. We will need a sound component. Drag a ***Sound*** component from the ***Media Palette*** into the ***Viewer***. The sound component will play a media file, so we need to add one by uploading it. Add any sound you like from the assets folder.



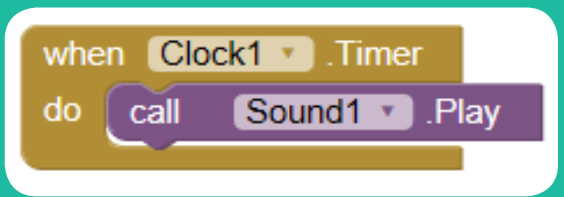
3 Switch to the Blocks Editor.



4 Select the ***Clock1*** component and drag a ***when Clock1. Timer do*** block into the viewer.



5 Select the ***Sound1*** component and drag a ***call Sound1. Play*** block into the event.



6 Now use the ***Connect*** tab to test your app. What happens when you change the ***TimerInterval***?

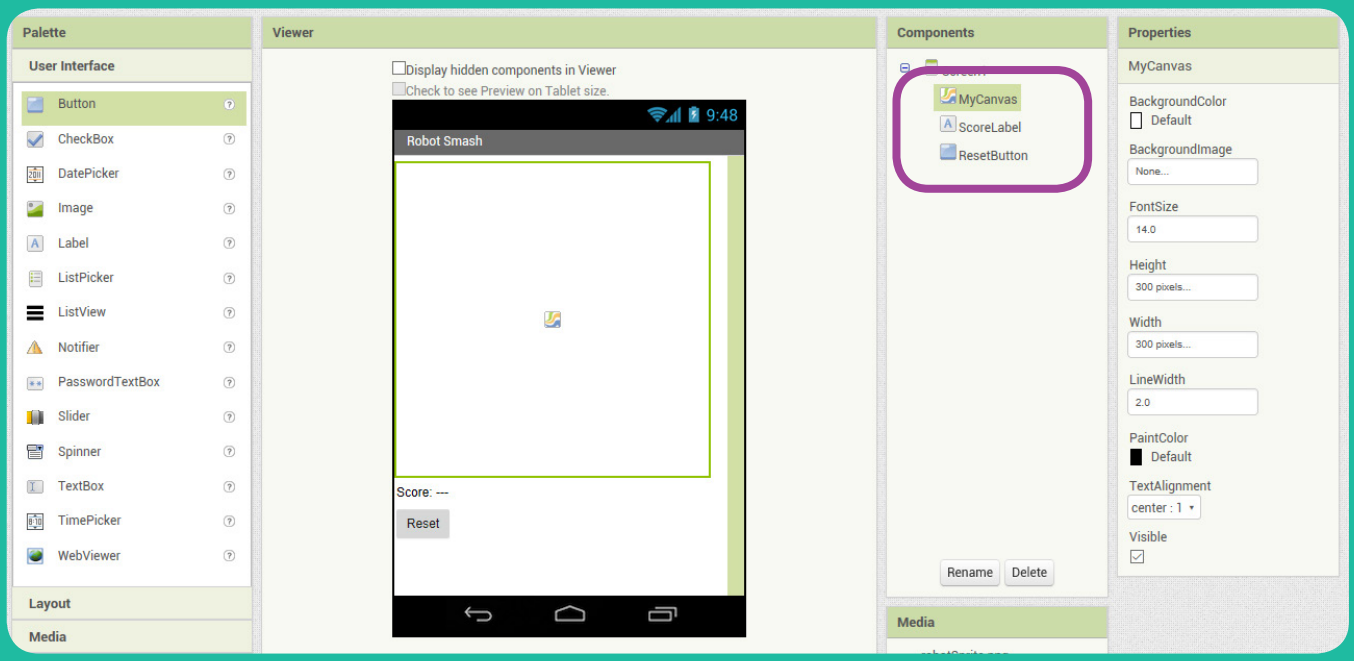
Robot Smash

The canvas is not just used for lines and shapes. It can also have "sprites," which are images that can move anywhere we need them to go in the canvas.

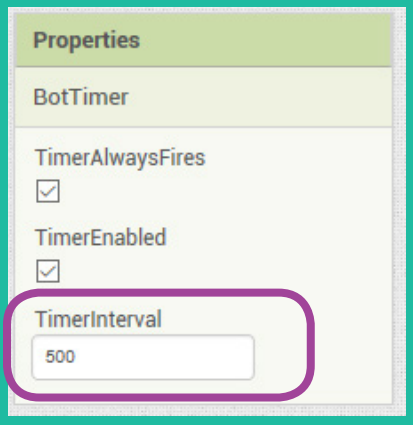
The canvas has a coordinate system based on the size of the canvas with 0,0 being in the upper left corner. This next activity puts that to use.

Activity 02-05: Robot Smash

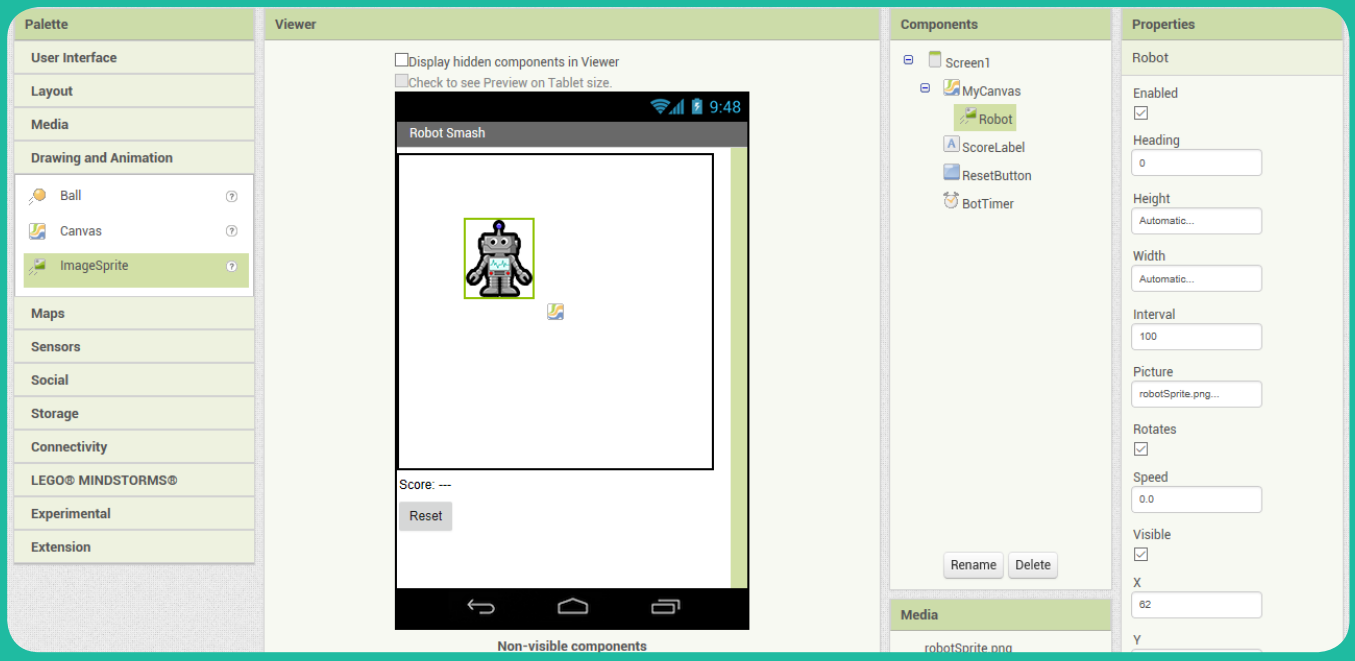
1 Begin with a new project. Give it the name, "RobotSmash" (no spaces!) Set the ***Screen1 Title*** property to "Robot Smash." From the ***Drawing and Animation Palette***, add a ***Canvas*** component to the viewer and change its ***Height*** and ***Width*** properties to "300" pixels each. Also, rename it as "MyCanvas." From the ***User Interface Palette***, add a label and a button below the canvas. Rename the label as "ScoreLabel" and change the text to "Score: ---" and rename the button as "ResetButton" and change its text to "Reset."



2 Select the ***Sensors Palette*** and drag a ***Clock*** component into the ***Viewer.*** Rename the component as "BotTimer" and change the ***TimerInterval*** property to 500 milliseconds (half a second).

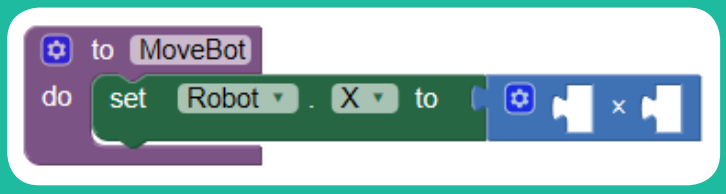


3 All that's left for this game is an actual robot to "smash." From the ***Drawing and Animation Palette***, drag an ***ImageSprite*** component into the ***Canvas*** and rename it as "Robot." Select the ***Image*** property of the sprite and upload "RobotSprite.png"



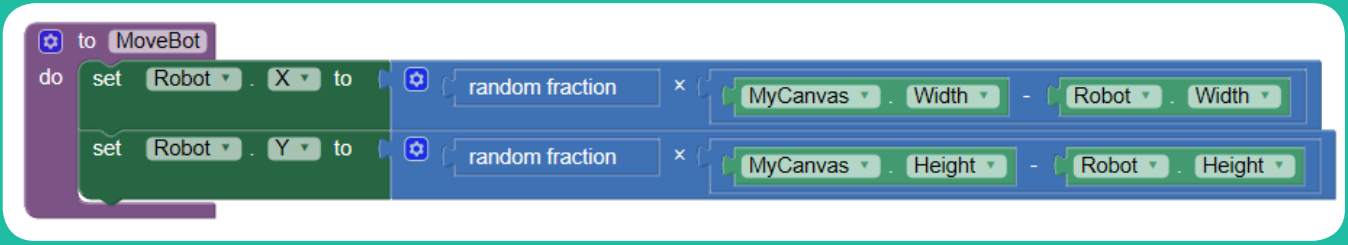
4 Now we can create our code. Switch over to the Blocks Editor. We're going to create a ***Procedure***. A procedure is like a shortcut for blocks. We can create a procedure just like we would an event by stacking blocks inside of it and then when we want to execute those blocks, we just have to call the procedure. This is especially useful when we have to do the same thing in several different places in the code.

5 Select the ***Procedures*** component and drag a ***to procedure do*** block into the viewer (be sure that you DO NOT drag the ***to procedure result*** block!) Just as with the variable block, you need to create a name for the procedure. Let's call ours, "MoveBot." When this procedure is called, it will move the Robot sprite to a new location in the canvas. Select the ***Robot*** component and drag a ***set Robot.X to*** block into the procedure. We will need to do a little math to set the new location for the sprite, so drag a multiplication block from the ***Math*** components to the slot at the end of the set block.

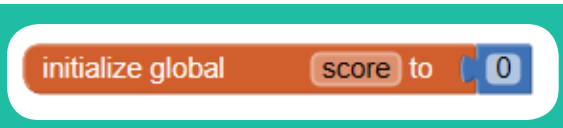


6 We want to get a number that is between 0 and the width of the ***Canvas*** component, minus the width of the sprite. From the ***Math*** components, drag out a subtraction block and add ***MyCanvas.Width*** and ***Robot.Width*** blocks to it as shown. Place this new block in one of the slots in the multiplication block.

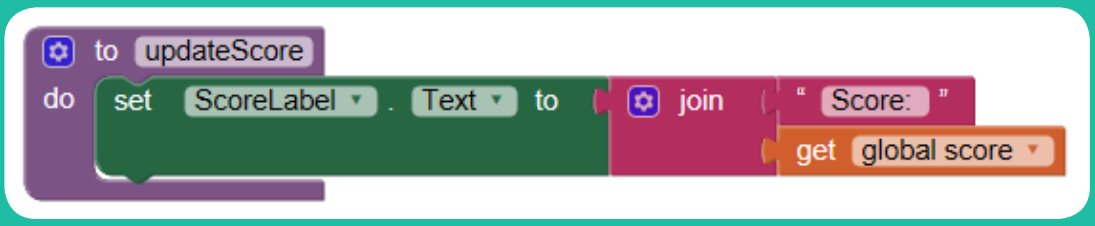
7 Select the ***Math*** components again and drag a ***random fraction*** block into the other slot in the multiplication block. Random fraction selects a random number between 0 and 1 and by multiplying by the other block, we will always get a number for the sprite that fits inside the width of the canvas. Duplicate the ***set Robot.X*** block and change it to ***set Robot.Y*** for the *height* of the Canvas as shown below.



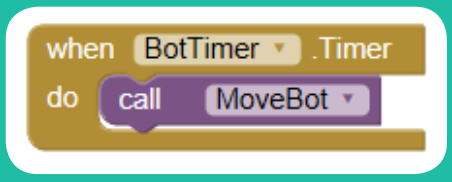
8 We also need to create a procedure for the score, but before we can do that, we need to initialize a variable to hold the score. Select ***Variables*** and drag ***initialize global name to*** into the Viewer. Change the name to "score" and add the number 0 to the slot at the end.



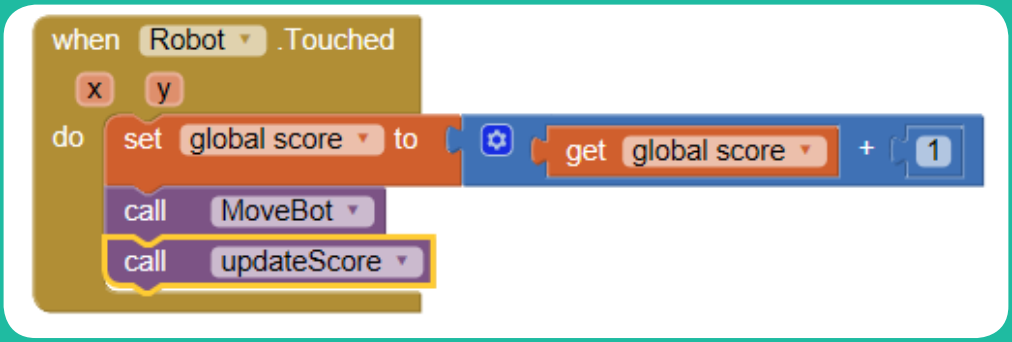
9 Select ***Procedures*** and drag another ***to procedure do*** block into the viewer. Change the name of the procedure to "updateScore." When this procedure is called, the score label gets updated with the value of the score variable. Select the ***ScoreLabel*** component and drag a ***set ScoreLabel.Text to*** block into the new procedure. We want to change the text to the word "Score: " plus the actual score. From the ***Text*** components, drag a ***join*** block and attach a text block for "Score: " and the score variable as shown.



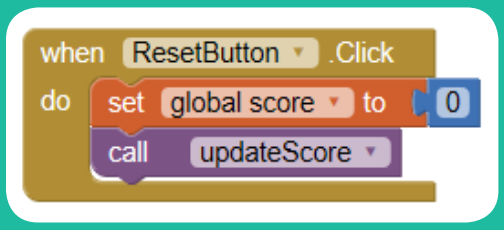
10 Every time the timer interval is triggered, we want to move the sprite. Select the ***BotTimer*** component and drag a ***when BotTimer.Timer do*** block into the viewer. To this we will add a ***call MoveBot*** procedure from the Procedures components as shown. Every interval, we will call the MoveBot procedure, which sets new x and y values for the sprite.



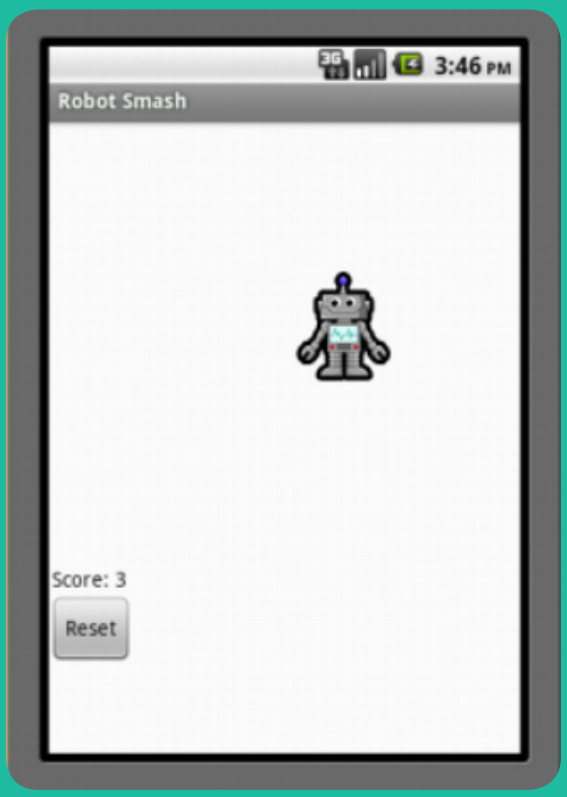
11 If the user touches the sprite, then we want to add 1 to the score, move the sprite and update the score. Select the ***Robot*** component and drag a ***when Robot.Touched do*** block into the ***Viewer.*** Add a ***set global score to*** block and place an addition block to the end so that we are adding exactly 1 to the current value of the score variable. Then place calls to both of the procedures so that the sprite gets moved and the score label gets updated.



12 Our app has a reset button in case the user wants to start over. Select the ***ResetButton*** component and drag a ***when ResetButton.Click do*** block into the viewer. Add to this block a block to set the global score variable to 0 and a call to the ***updateScore*** procedure to change the label.



13 Now use the ***Connect*** tab to test your app. The robot is hard to catch, isn't it? Try adjusting the ***TimerInterval*** to see if you can make the game easier.



Extra Steps

What if there were some way to make the ***TimerInterval*** less predictable? Since ***TimerInterval*** is a property of the ***BotTimer*** component, we could change it to a random value (between 500 and 2000 milliseconds, for instance) whenever the procedure to move the robot is called.

We could also keep track of the hits *and* misses in the game. We could set up a variable to record all of the Canvas touched events. Remember, any sprite touched events are included as a canvas touched event.

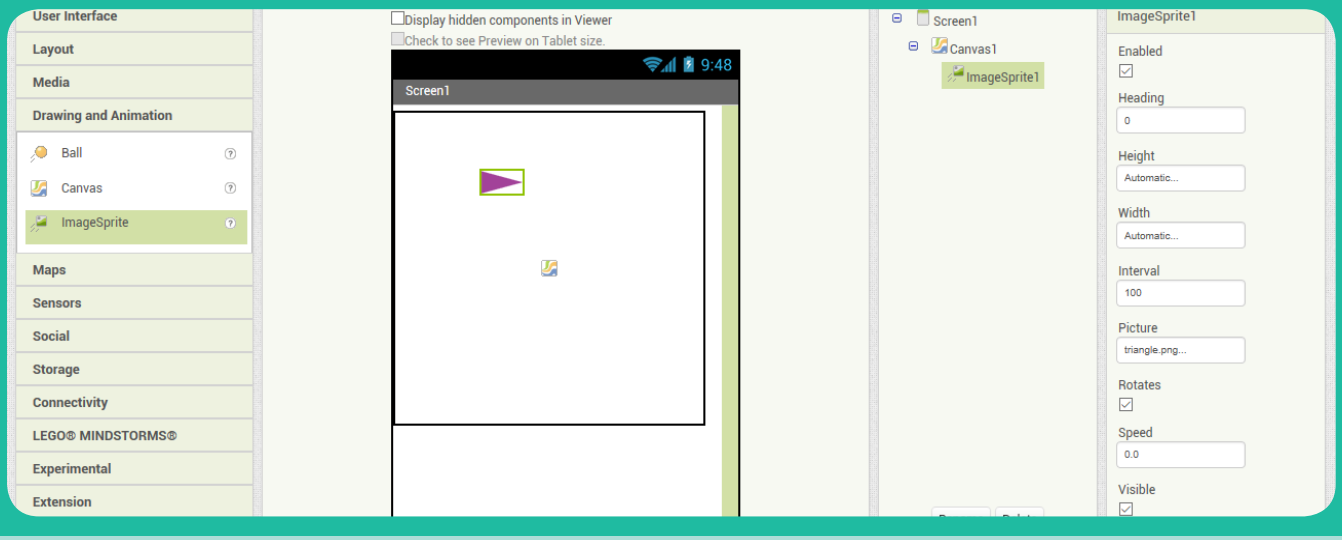
Sprite Behavior

In addition to x and y coordinates, an image sprite in the canvas component has properties for speed and heading so that it can move around on its own. The speed is determined by an interval property, much like the TimerInterval for the clock. The speed is a value that tells the sprite how many pixels it should move each interval. The heading is the direction that the sprite is moving. Look at the image on the right. A heading of 0 is directly to the right while a heading of 270 is straight down.

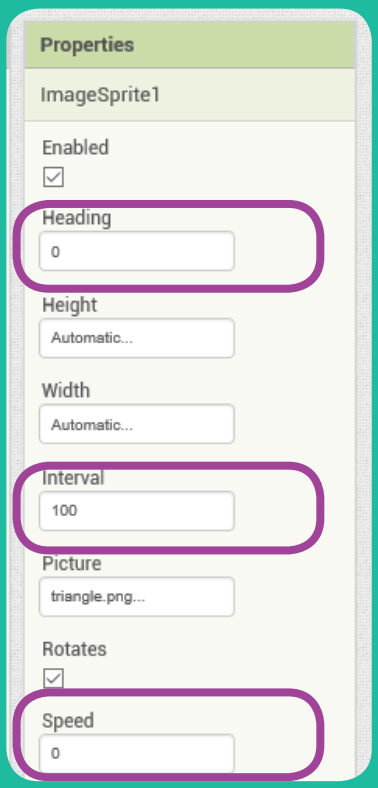
This next exercise will help you learn more about sprite behavior.

Activity 02-06: Moving Sprites

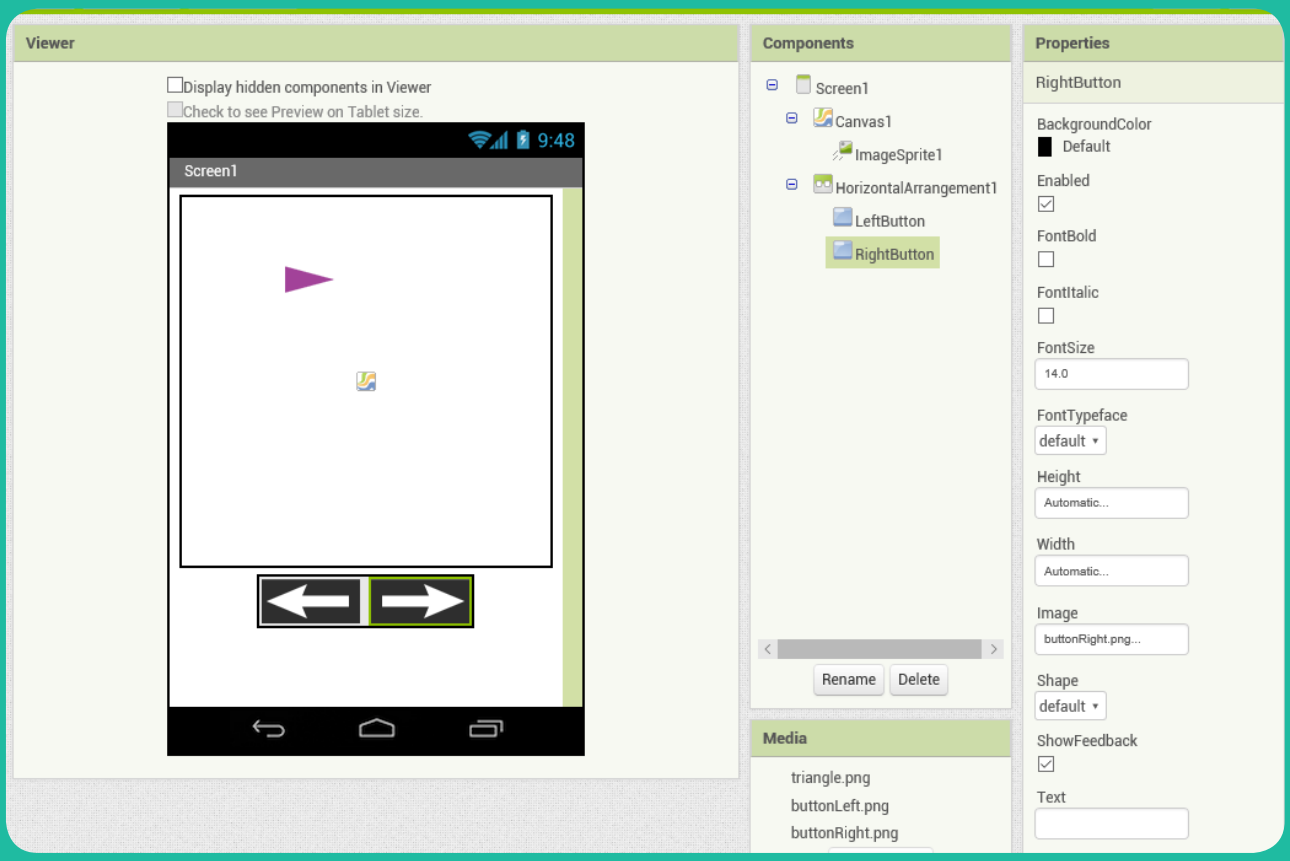
1 Begin with a new project. Give it any name, (we will use "MovingSprites") Set the ***Screen1 AlignHorizontal*** property to "center." Add a canvas from the ***Drawing and Animation Palette*** and set its width and height to 300 pixels each. With the ***Drawing and Animation Palette*** still open, drag an ***ImageSprite*** into the canvas and for the picture property, upload "triangle.png."



2 Take a look at the ImageSprite properties. By default, it has a heading of 0 and an interval of 100. But it currently won't go anywhere because the speed is 0. Change the speed to "5" and test your app.



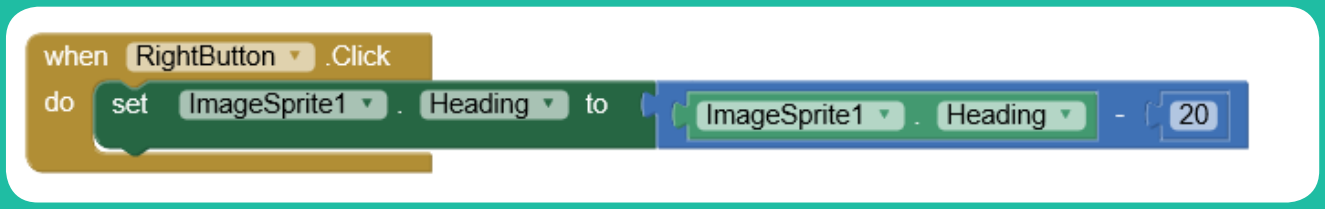
3 Let's add some more control to the sprite. From the Layout Palette, drag a HorizontalArrangment into the Viewer, under the Canvas. Open the UserInterface Palette and drag two buttons into the HorizontalArrangement. Rename the first button, "LeftButton" and the second one, "RightButton. Remove the text from the buttons and upload the images "buttonLeft.png" and "buttonRight.png" into the buttons.



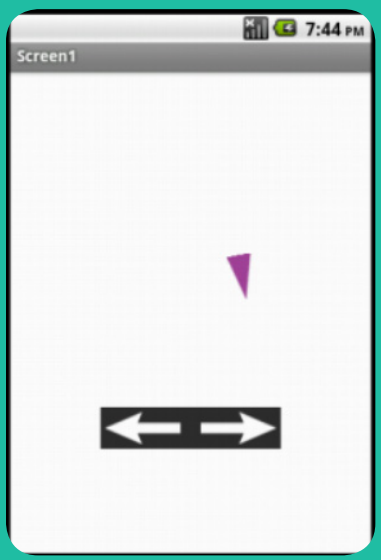
4 Switch to the Blocks Editor. Remember this chart? If you add to the heading, then the sprite rotates left. If you subtract from the heading, it rotates right. Let's apply that to our buttons.

5 Select the ***LeftButton*** component and drag a ***when LeftButton.Click*** event into the ***Viewer***. Select the **ImageSprite1** component and drag a ***set ImageSprite1.Heading to*** block into the event. Add a ***Math*** addition block and put ***ImageSprite1.Heading*** and the number 20 into the slots.

6 Duplicate this event for the RightButton, replacing the Math addition block with a Math subtraction block as shown.



7 Now use the ***Connect*** tab to test your app. Tapping the buttons turns the sprite, but you have to keep tapping to keep turning. Why is that?

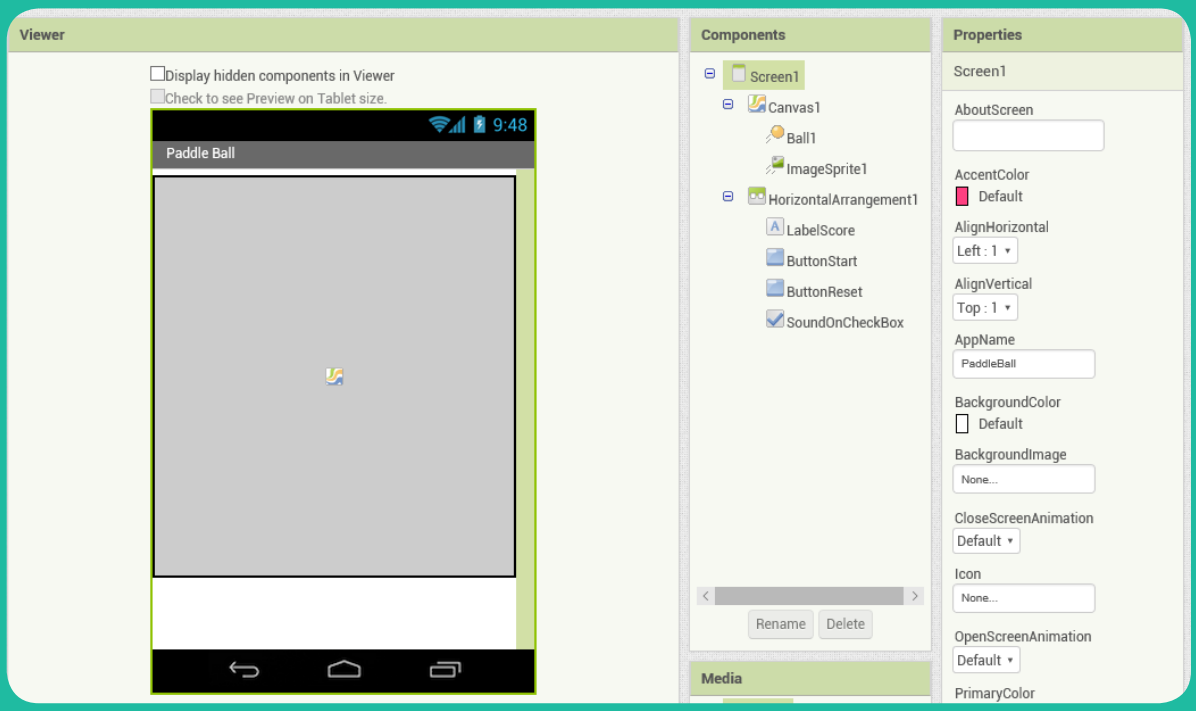


Paddle Ball

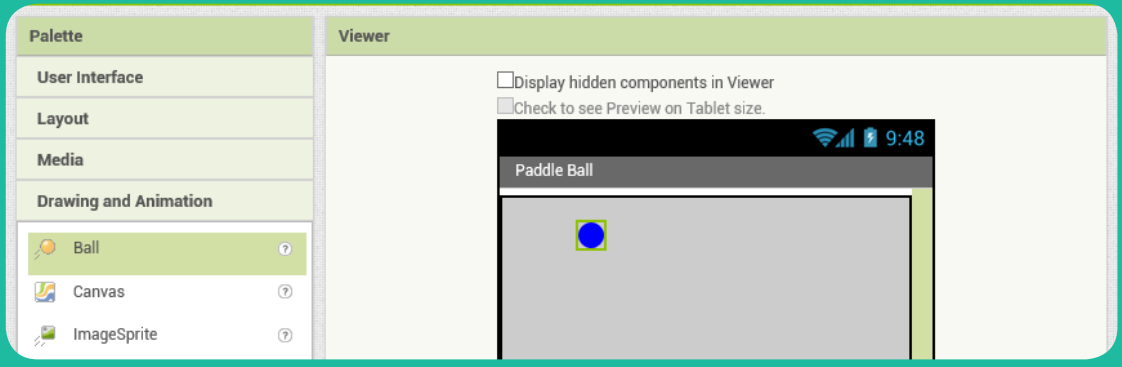
In the last activity, the sprite moved easily in the direction of its heading, but only until it hit one of the edges of the canvas. Then the sprite stopped. We can instead make the sprite bounce when it reaches an edge or collides with another object. Let's find out how.

Activity 02-07: Paddle Ball

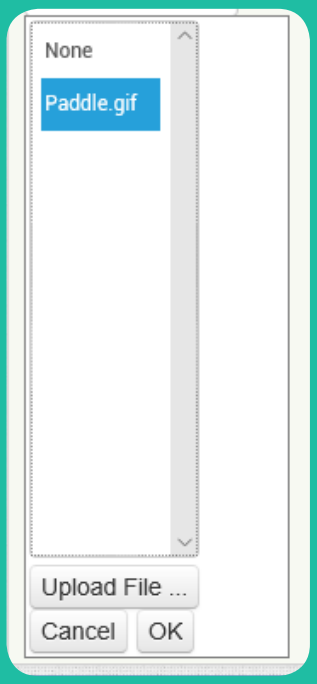
1 Begin with a new project. Name it "PaddleBall." Set the ***Screen1 Title*** property to "Paddle Ball." Add a canvas from the ***Drawing and Animation Palette*** and set its width to ***Fill Parent*** and height to 350 pixels. Change the BackgroundColor of the Canvas to "Light Gray."



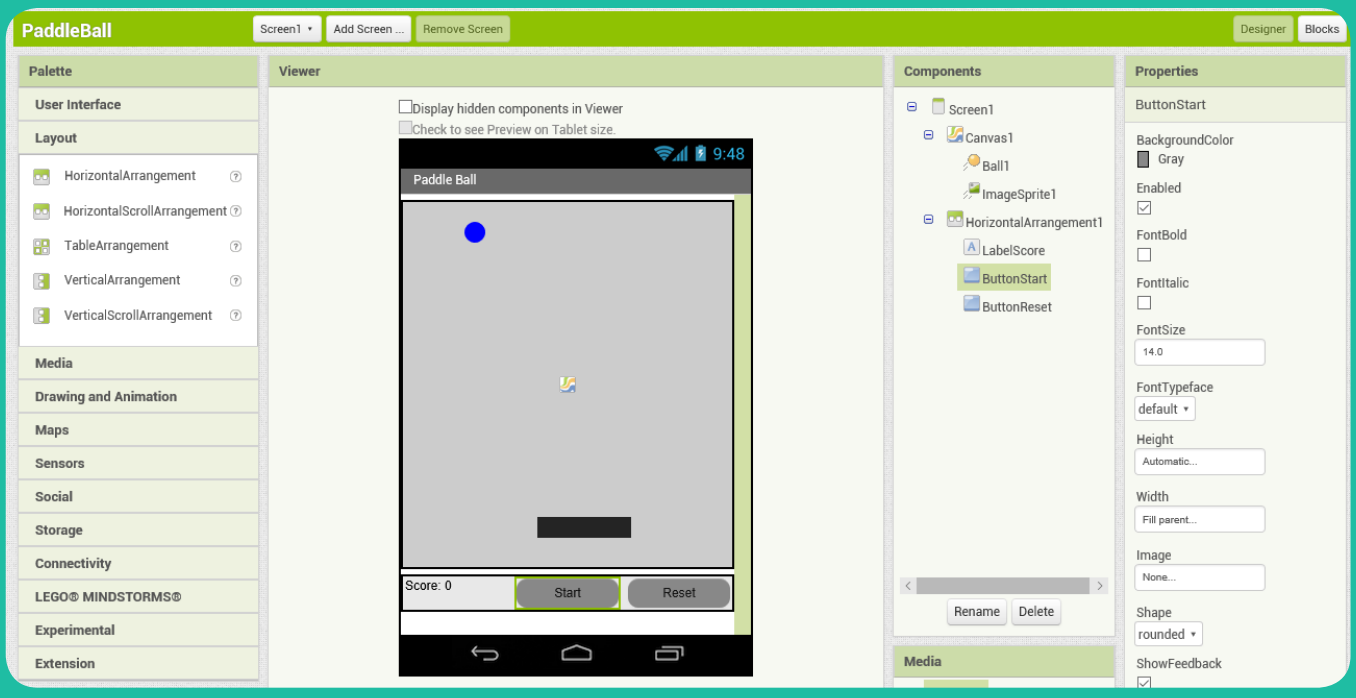
2 Open the ***Drawing and Animation Palette*** and add a ***Ball*** to the canvas. Set the ***Radius*** property of the ***Ball*** to 10. We will be changing the speed, heading and interval in code, so don't worry about that. Make sure that the ***Enabled*** property for the ball is unchecked. Feel free to change the color of the ball.



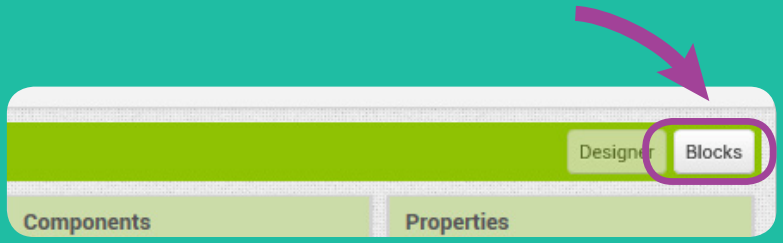
3 Now drag an ***ImageSprite*** from the ***Drawing and Animation Palette*** into the Canvas. Upload "Paddle.gif" for its picture. Change the Y property of the ImageSprite to "300."



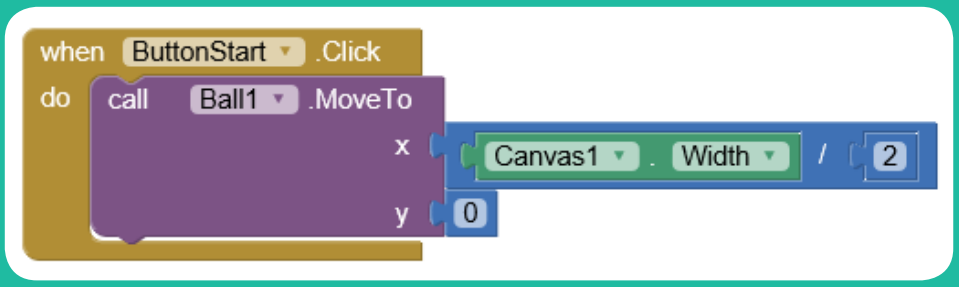
4 The final step of our design is the user interface. Select ***HorizontalArrangement*** from the ***Layout Palette*** and change its width to ***Fill Parent***. From the ***User Interface Palette***, add a label and two buttons to the ***HorizontalArrangement***. Rename the label to ***LabelScore*** and change its text to "Score: 0." Rename the first button as ***ButtonStart*** and the second as ***ButtonReset***. Set the width of both buttons to ***Fill Parent***. Finally, change the text of ***ButtonStart*** to "Start" and ***ButtonReset*** to "Reset."



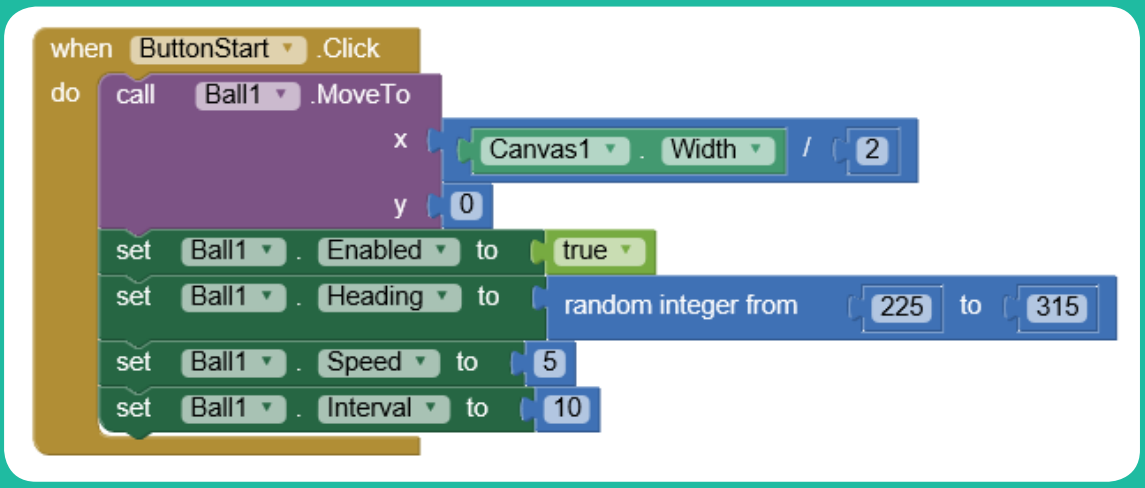
5 Switch to the Blocks Editor.



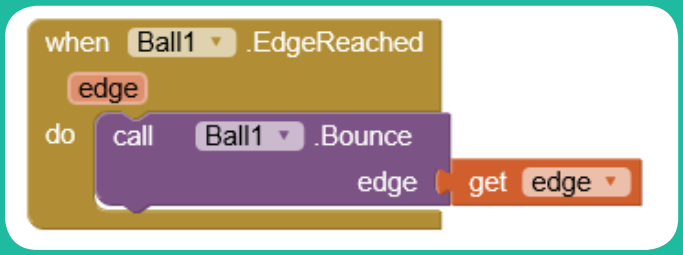
6 Select the ***ButtonStart*** component and drag a ***when ButtonStart.Click do*** event into the ***Viewer***. Every-time the game is started, we want to put the ball at the top middle of the canvas, set the heading, speed and interval of the ball and enable the ball so that it starts moving. Select the ***Ball1*** component and drag ***call Ball1. MoveTo*** into the event. Set the y value to "0" (top of the canvas) and the x value to Canvas1.Width divided by "2" (middle of the canvas).



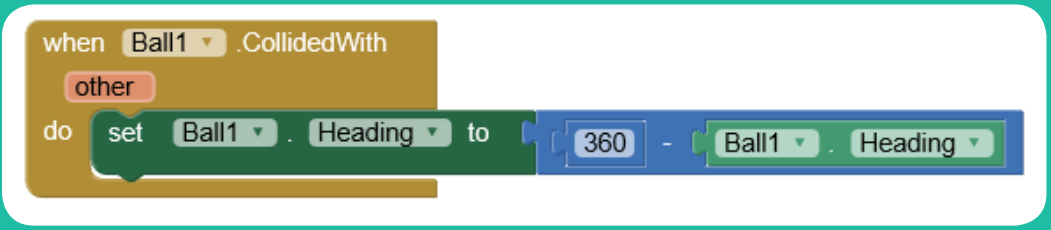
7 Select the ***Ball1*** component and drag ***set Ball1.Speed*** into the event. Set this property to "5." Duplicate this block to set the Interval, Heading and Enabled properties as seen below. Set the Interval to "10." The Heading should be a bit unpredictable so drag a ***random integer from 1 to 100*** block from ***Math*** into the slot for that. Remember that 270 is straight down, so any number from 225 to 315 should send the ball down, but always in a random direction. Finally, take a ***true*** block from ***Logic*** so that the ball is enabled and able to move.



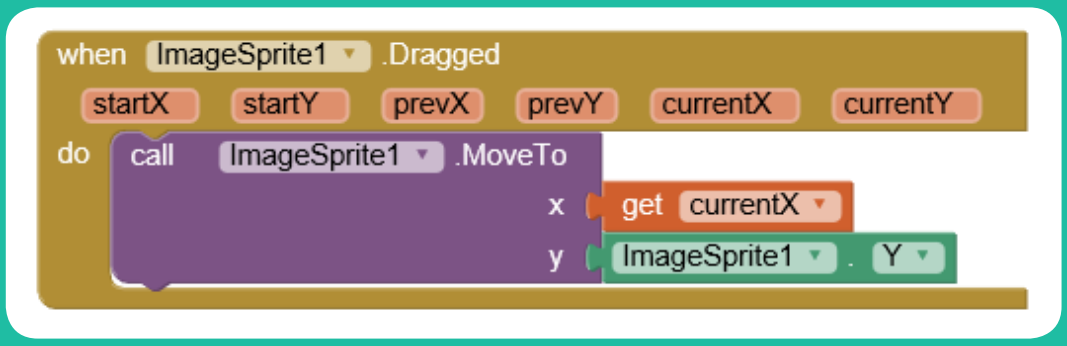
8 We want the ball to bounce off of the edges of the can-vas. This is easy to do. Select the ***Ball1*** component and drag a ***when Ball1.EdgeReached do*** block into the ***Viewer***. Drag a ***call Ball1.Bounce*** block into the event and drag the ***edge*** parameter of the event into the edge slot as shown.



9 We also want the ball to bounce when it touches the paddle. Unfortunately, ***Bounce*** only works with edges, so we have to use something else. Select the ***Ball1*** component and drag a ***when Ball1.CollidedWith do*** block into the viewer. Since the paddle is the only other sprite in the canvas, we don't have to specify what it is colliding with. Drag a ***set Ball1.Heading to*** block into the event. To make an object bounce off a horizontal surface like the paddle, we get a new heading by subtracting the current heading from 360 (the number of degrees in a circle). Drag a ***Math*** subtraction block into the slot on the set heading block and insert the number "360" and ***Ball1.Heading*** as shown.

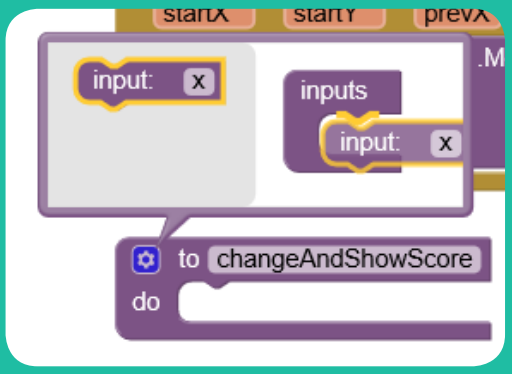


10 Getting the ball to bounce off the paddle is challenging because the paddle doesn't move. We can let the user move the paddle by using the dragged event. Select the ***ImageSprite1*** component and drag ***when ImageSprite1.Dragged*** do into the viewer. When the sprite is dragged, we want to move it, so drag a ***call ImageSprite1.MoveTo*** block into the event. When the sprite is dragged, we want to place it at whatever the current x is, so drag the ***currentX*** parameter into the x slot. We don't want to change the y, but we still need to fill the slot, so drag an ***ImageSprite1.Y*** block into the y slot.

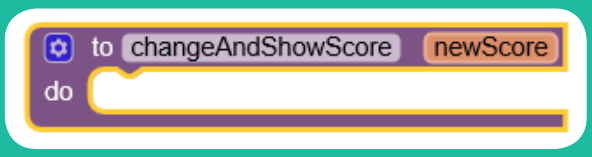


11 If we try the app now (go ahead, we'll wait) we will see that the paddle moves and when the Start button is pressed, the ball moves and bounces off the edges and the paddle. But it still isn't much of a game. We should have a score and some way to lose. Let's set a variable for the score by dragging an ***initialize global name to*** block from ***Variables*** and changing the name to "score" and the value to the number "0."

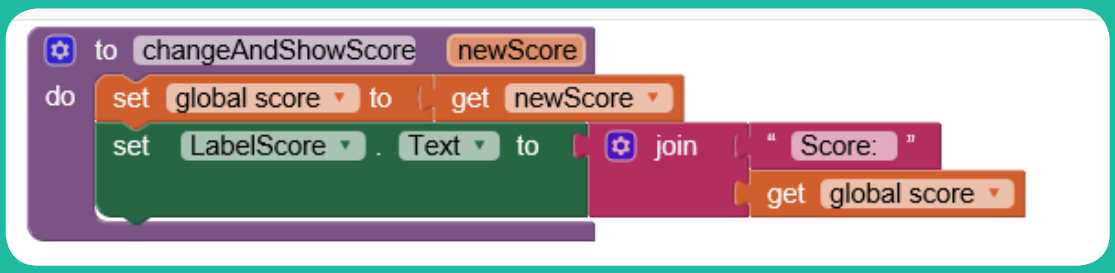
12 Since we'll be updating the score a lot, let's create a procedure. Drag a ***to procedure do*** block into the viewer and change the name of the procedure to "changeAndShowScore." This procedure needs to be able to use an input to process the score. Click on the blue modify icon and drag an ***input*** block into the procedure block as shown.



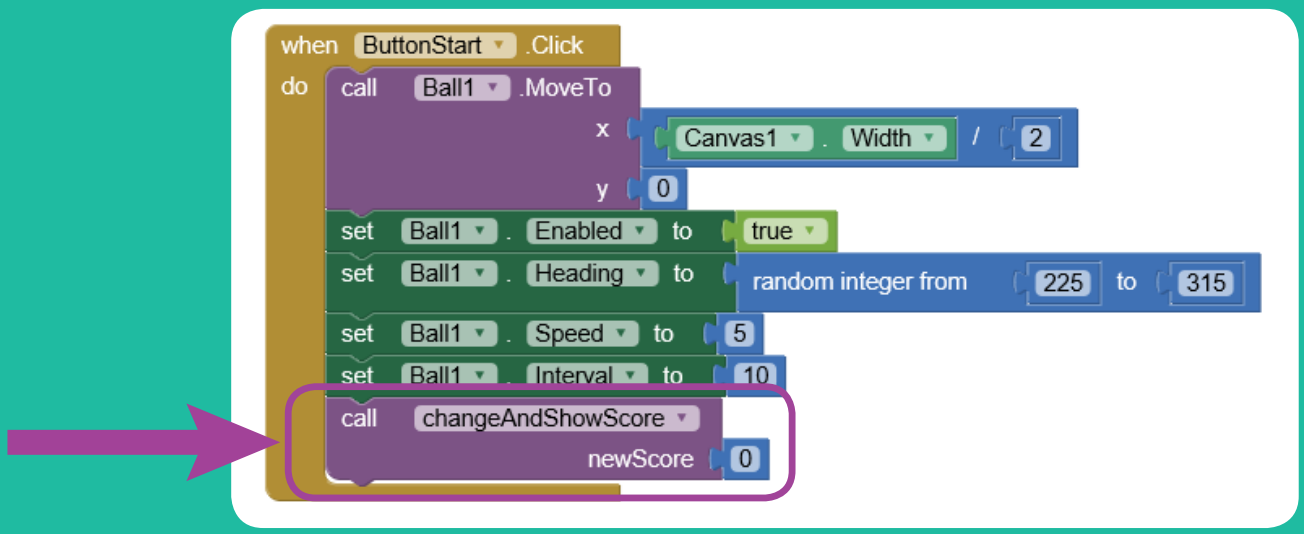
13 Change the name of the input from "x" to something more descriptive: "newScore." That way, we know what we intend to do with the input.



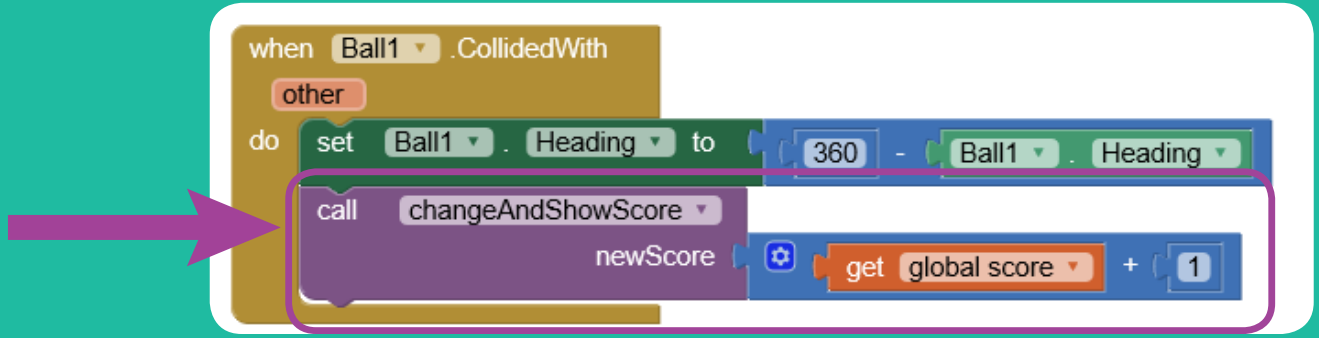
14 When the procedure is called, we want to update the score variable and update LabelScore to show it. From ***Variables***, drag a ***set global score to*** block into the procedure and drag the ***newScore*** parameter into its slot. Select the ***LabelScore*** component and drag a ***set LabelScore.Text to*** block into the procedure. We want the label to show the word "Score: " plus the actual score, so drag a ***join*** block from ***Text*** and add a text block and ***get global score*** block as shown.



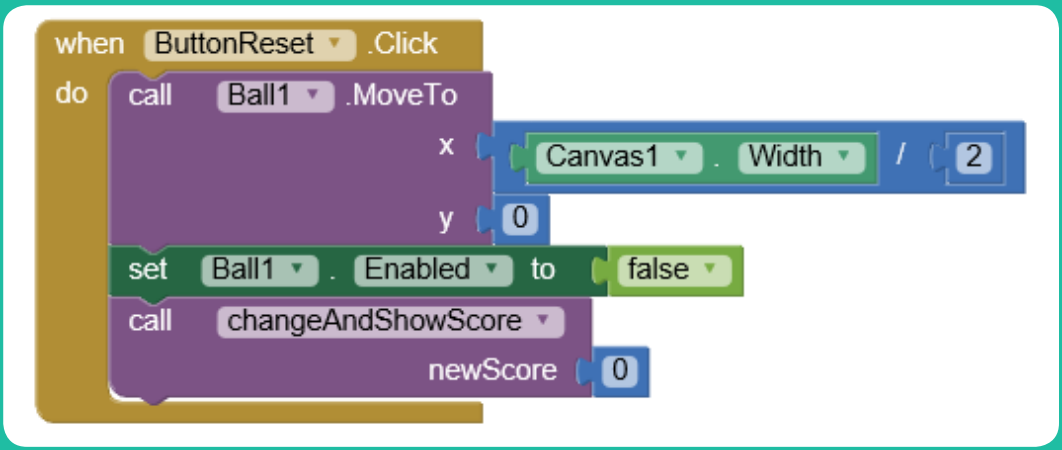
15 Now that we have a procedure for the score, let's use it. Add a ***call changeAndShowScore*** block to the ***Button-Start.Click*** event and give the newScore input a value of "0" as shown.



16 We should increase the score every time the paddle hits the ball, so add a ***call changeAndShowScore*** block to the ***when Ball1.CollidedWith*** event and add a ***Math*** addition block to increase the ***score*** variable by "1" as shown.

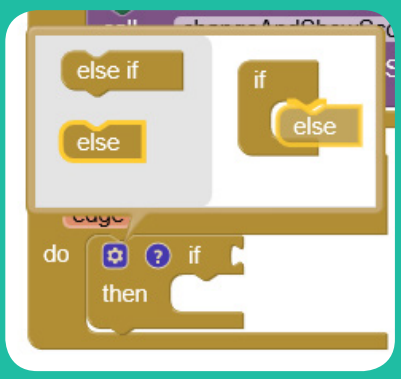


17 Let's also set up the ***Reset*** button. Duplicate the ***ButtonStart*** event and change it to ***ButtonReset***. Delete the blocks for ***Ball1.Heading***, ***Speed*** and ***Interval*** by dragging them over to the garbage can. Change the ***Ball1.Enabled*** setting to ***false*** and leave the other blocks as shown.

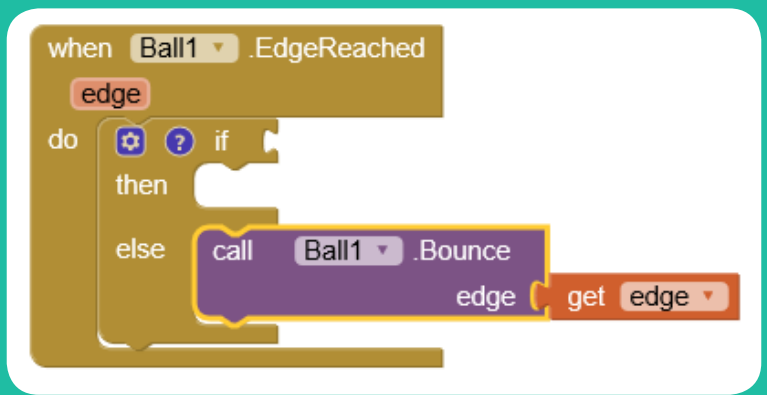


18 Our game is nearly complete. However, the ball still bounces off the bottom edge when we want it to end the game. Looking at the sprite direction chart on page 50, we can see that each edge has a specific value. The bottom edge is "-1"

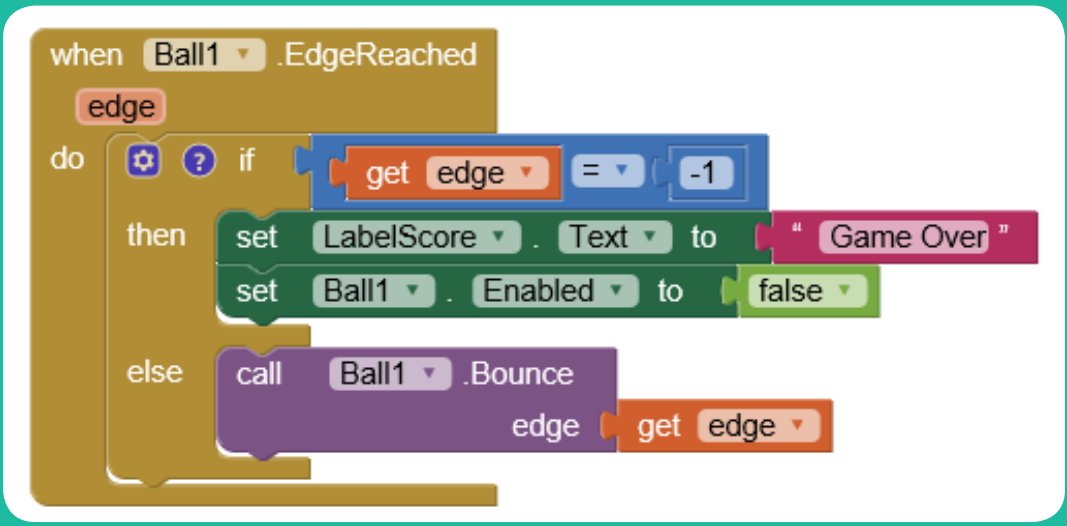
19 On our ***when Ball1.EdgeReached event***, we need to check if the ball is touching the edge equal to "-1." For that, we need a conditional. From the ***Contol*** components, drag an ***if then*** block into the event. Click on the blue modifier icon to add an "else" to the conditional - if the edge isn't -1, then we still want to bounce the ball.



20 Drag our ***call Ball1.Bounce*** block into the ***else*** slot.



21 For our condition, drag an "=" block from ***Math***. Put our ***edge*** parameter in one slot and the number "-1" in the other slot. If the ball reaches the bottom edge, we want to display a Game Over message and stop the ball. From the ***LabelScore*** component, drag a ***set LabelScore.Text to*** block into the then slot and add a text block that says "Game Over." From the ***Ball1*** component, drag a ***set Ball1.Enabled to*** block beneath the ***LabelScore*** block and set that to ***false***.



22 Now use the ***Connect*** tab to test your app. Keeping the ball from reaching the bottom shouldn't be too difficult. What can you do to make it more challenging?

What's Next?

There's still a lot more that can be done with App Inventor. Next, we will take a look at lists and what they can do for you.

3 Lists and Information

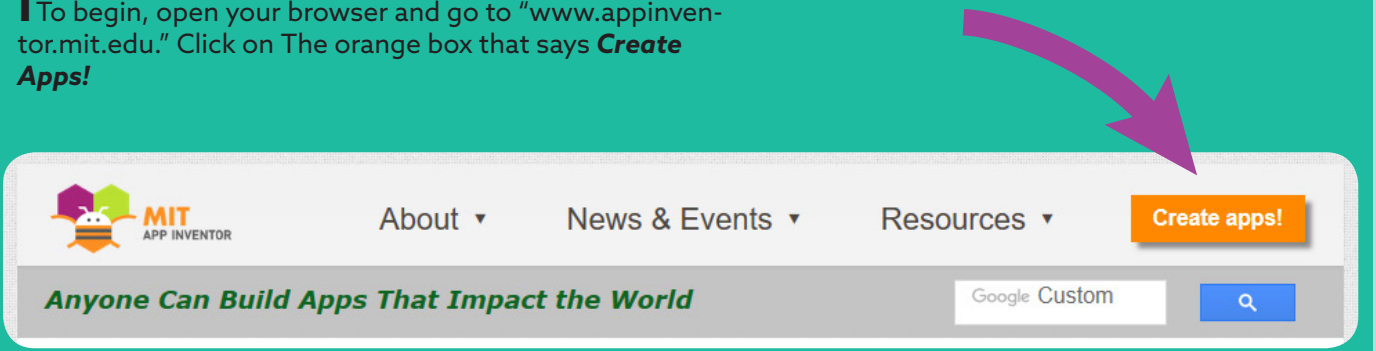
Often, it isn't enough to simply have variables to keep track of information. App Inventor executes events in a certain order and the information that we use also needs to be in order.

A Slideshow

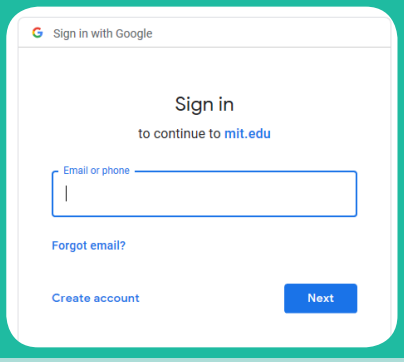
Imagine that you have a set of pictures and they have names like "1," 2" and so on. If you were writing an app to show those pictures in order, how would you tell the app which picture you wanted next?

Activity 03-01: Slideshow 1

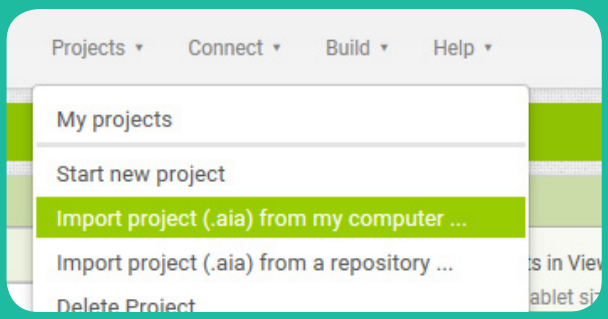
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



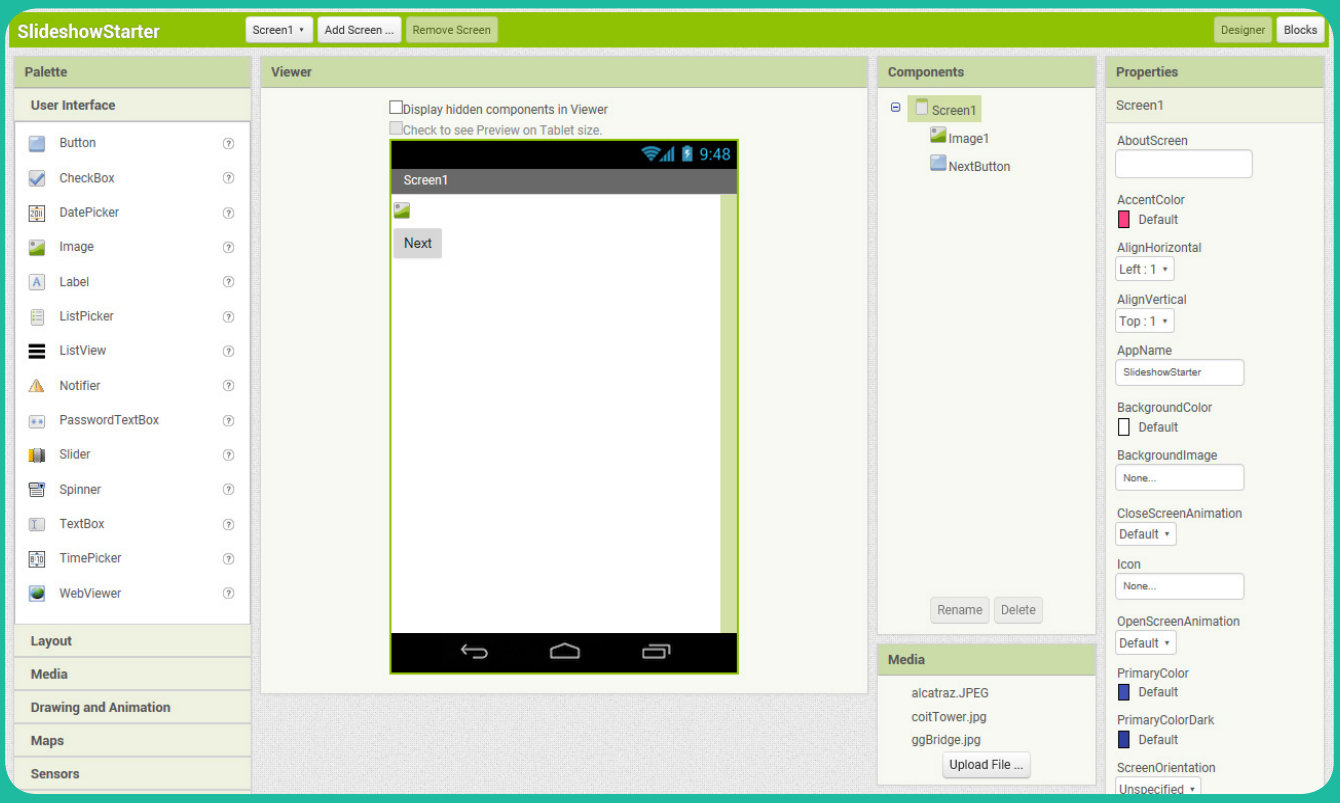
2 Remember, if you've been away for a while, you are going to need to sign in so that App Inventor can keep track of what you create.



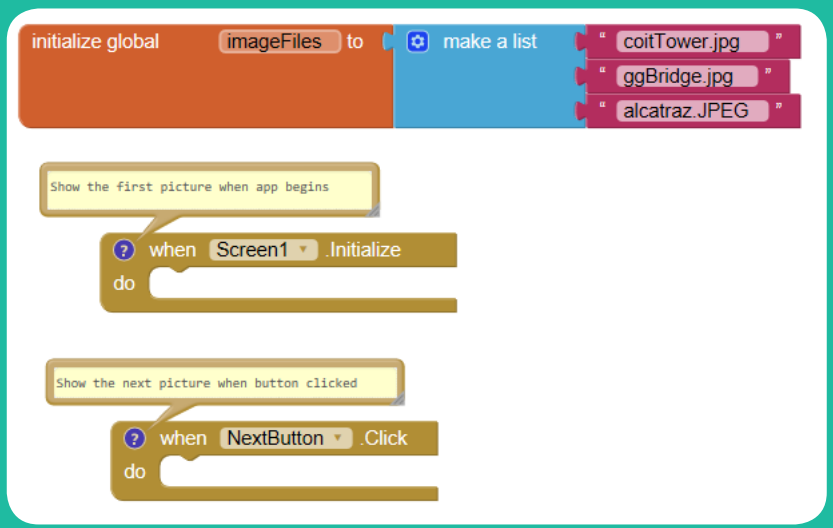
3 For this project, we are going to import a file from the Assets folder. Click on the ***Projects*** tab and select ***Import project (.aia) from my computer***. Select ***Browse*** and from the Assets folder, ***SlideShowStarter.aia*** and click OK.



4 The project should look like this. There is an image component and a button in the screen. Switch to the Blocks Editor.

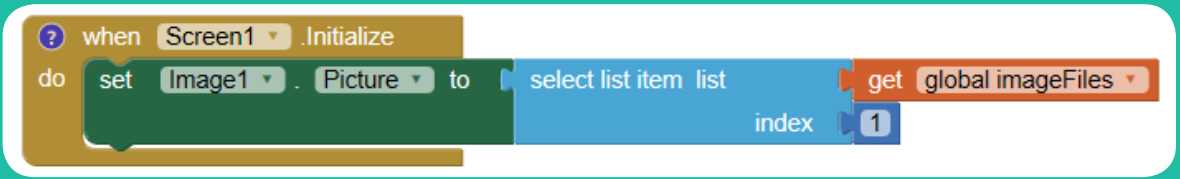


5 As you can see, part of the blocks have already been set up. So how do we show the pictures?



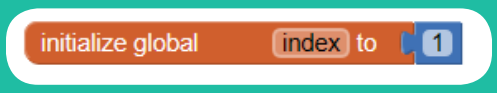
6 A variable called ***imageFiles*** has been initialized. Unlike previous variables, the variable is a ***List*** that contains the text of three different image file names. The file names are known as ***items*** in the list. Each item in a list has an ***index***, telling us where it is in the list. The first item has an index of 1, the second an index of 2 and so on.

7 The first event is triggered as soon as the app is started. It is supposed to show the first picture. So we need to ***set image1.Picture to*** property to the file name that is first in the list. The set picture property is part of the ***Image1*** component. For the file name, we need to get a ***List*** block that lets us ***select list item***. The list is our variable, ***imageFiles*** and the first picture is index 1.

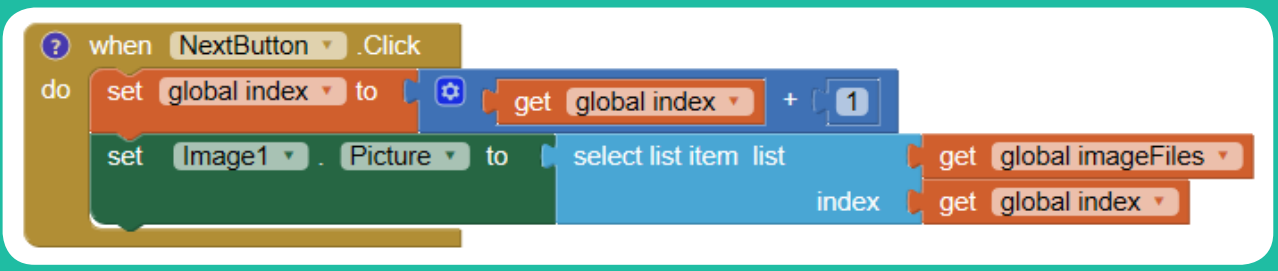


8 Connect your app to your emulator or device to test it.

9 The ***set Image1.Picture*** blocks are going to be an important part of the next event. But first, we need to be able to increment the index each time the ***NextButton*** is clicked. First, we will need to initialize a variable to keep track of our current index.



10 When the ***NextButton*** is clicked, we can increase the value of index by adding one to it. Drag a ***set global index to*** block into the ***NextButton*** event and use a ***Math*** addition block to add 1 to the current index variable. Duplicate the ***set image1.Picture to*** block from the first event and add it to the second event. Replace the number 1 with ***get global index***.



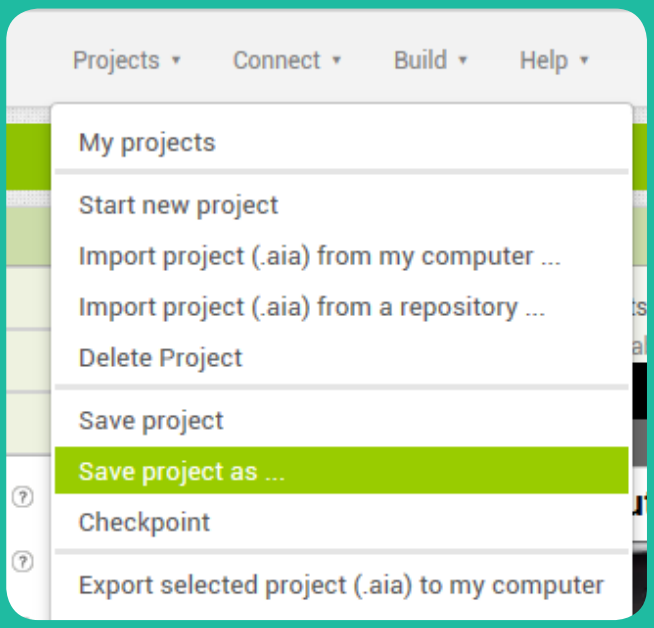
11 Now what happens when you test the app? If you press the Next button too many times, the app stops working because the index is more than the number of items in the list!

An Index Too Far

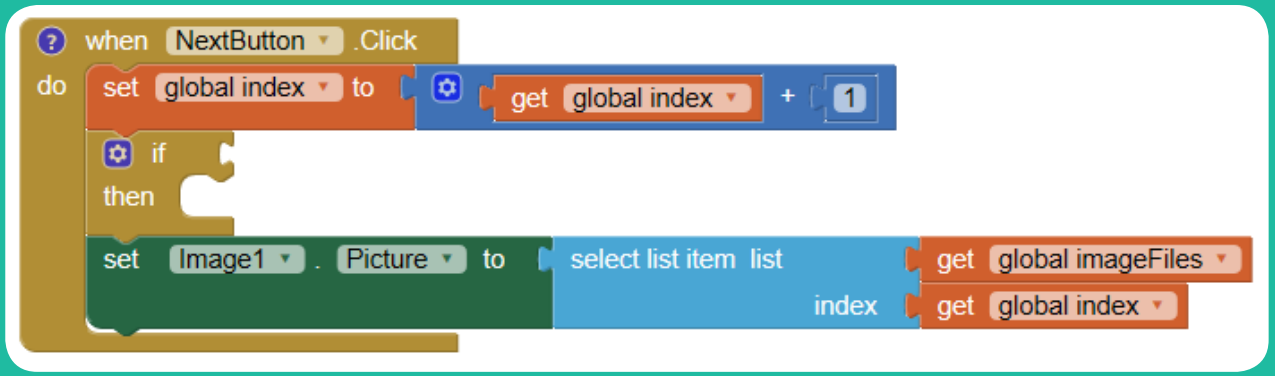
In the previous activity, the app has the names of the slides stored in something called a "list." It seems to work well until you run out of pictures to show. What do you do then? We know how many pictures are in our slide show, could we use a conditional (if/then) to stop when it reaches the end? What if the number of items inside the list changes? Is it possible to always get the total number of items in the list?

Activity 03-02: Slideshow 2

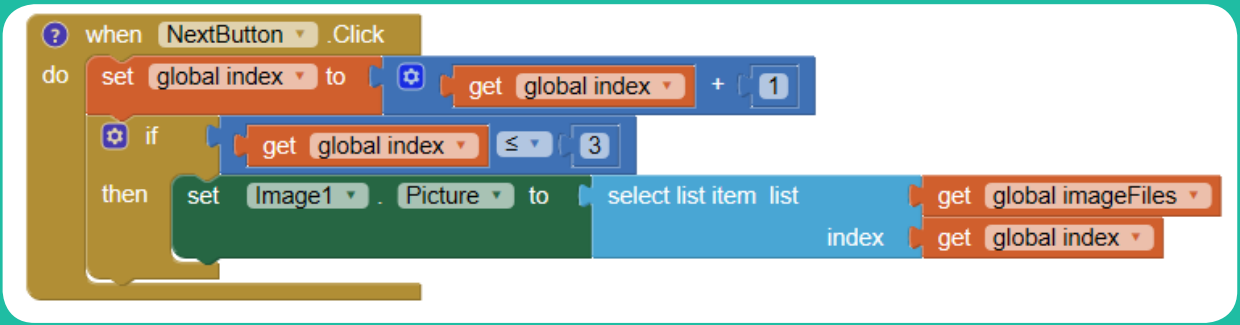
1 Let's go back to the app we were working on. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



2 In the Blocks Editor, let's look at the NextButton event. We only want to update the picture if the index is below a certain number. Drag an ***if then*** block from the ***Control*** components and insert it after the ***set global index*** block.

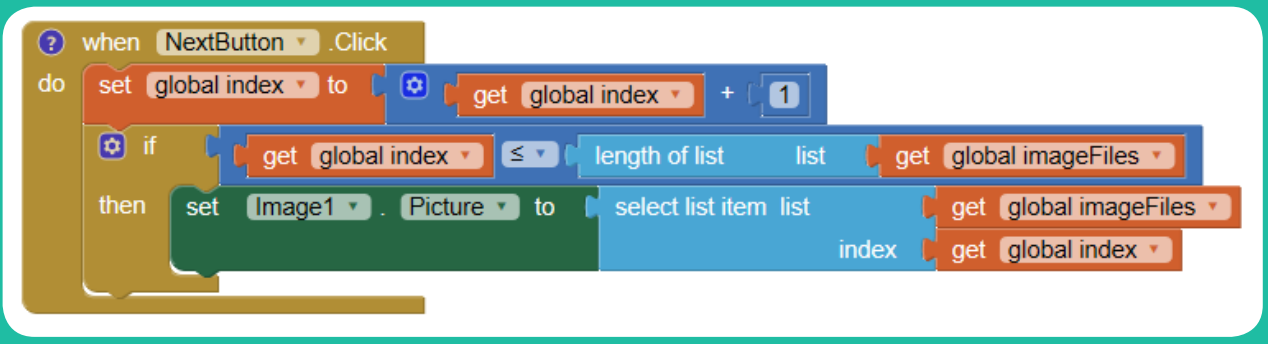


3 If the ***index*** number is less than or equal to the number of items in the list (which is 3) then we can update the image. Otherwise, we do nothing. Add the equals block from ***Math*** and change the operator from "=" to "≤" Insert a g***et global index*** block in the first slot and the number "3" in the second slot. Drag the ***set Image1.Picture*** block into the then slot.



4 Connect your app to your emulator or device to test it. Now the app doesn't do anything if the index gets too big. But what happens if we add more items to the list?

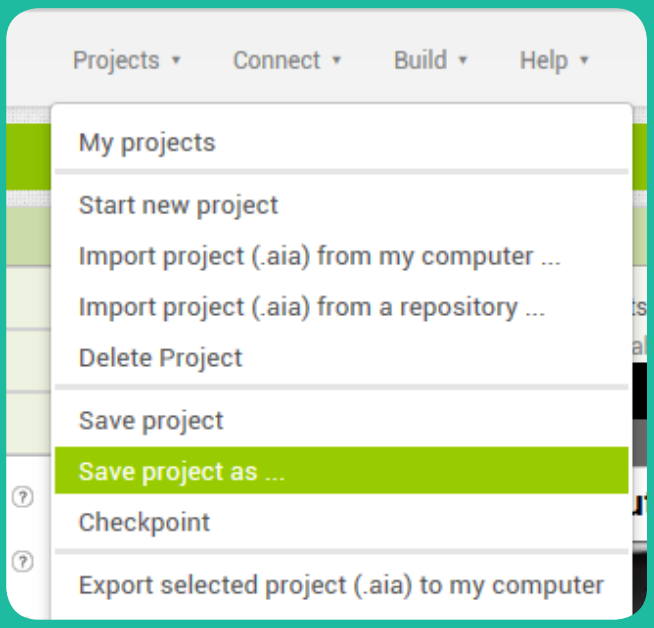
5 Fortunately, we have a block that we can use to tell us how many items are in a list. Select the ***Lists*** component and drag a ***length of list*** block into the second slot in the condtition. The only list we have is ***imageFiles***, so put that in the empty slot at the end of the block.



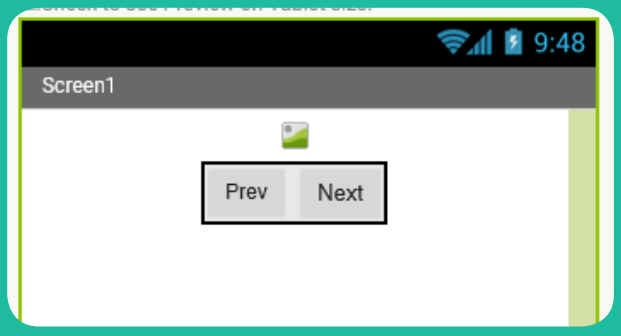
6 Now our app can handle a list of any size. It would be nice if we could sort through it backwards as well as forwards. Think about how you would do that before looking at the solution on the next page.

Activity 03-03: Slideshow 3

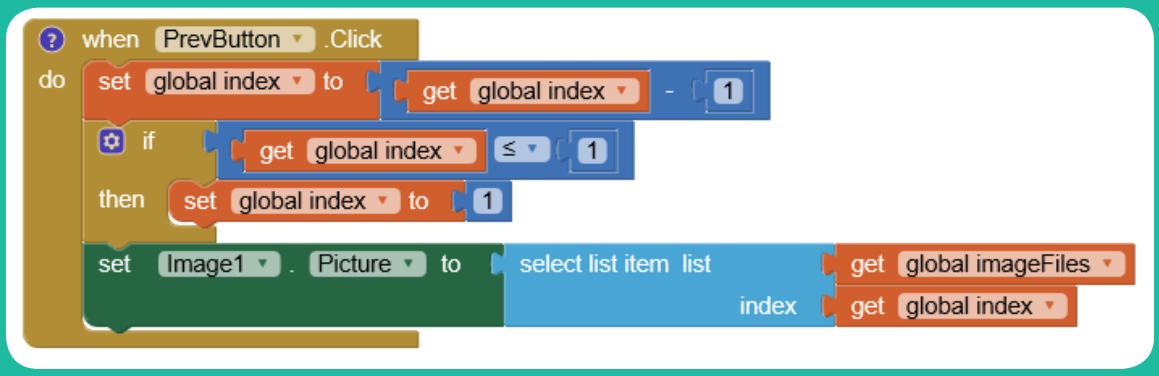
1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



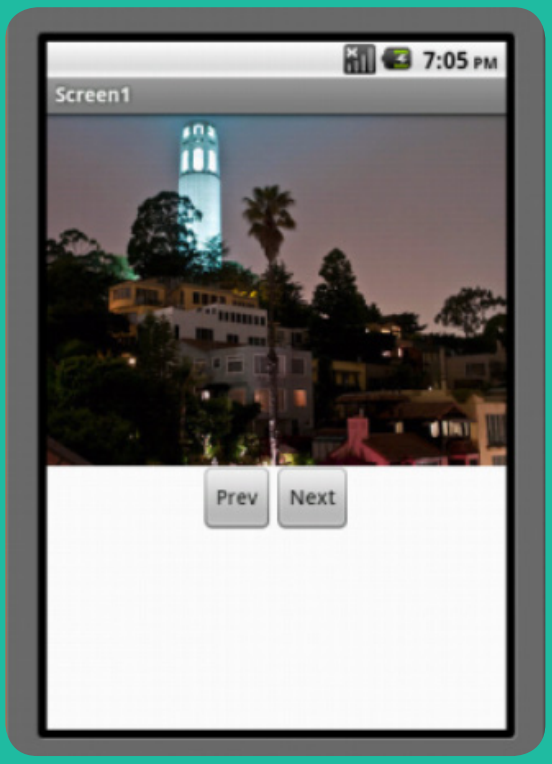
2 We need a new button for our "Previous" button, so go back to the ***Designer*** and add it.



3 Switch back to the Blocks Editor. Duplicate the ***Next-Button*** event and change it to ***PrevButton***. Instead of adding 1 to the index variable, we are subtracting 1. We need to change our condition for this button. No matter how long the list is, the first index is always 1, so we need to make sure that the index never gets below 1. Move the ***set image1.Picture*** block outside of the conditional and put a ***set global index to1*** block in the then slot as shown.



4 Connect your app to your emulator or device to test it. The two buttons let you go forward or backward through the list. If you wanted to, you could have the index "wrap around by going to the first or last index. Think about how you would do that and give it a try.

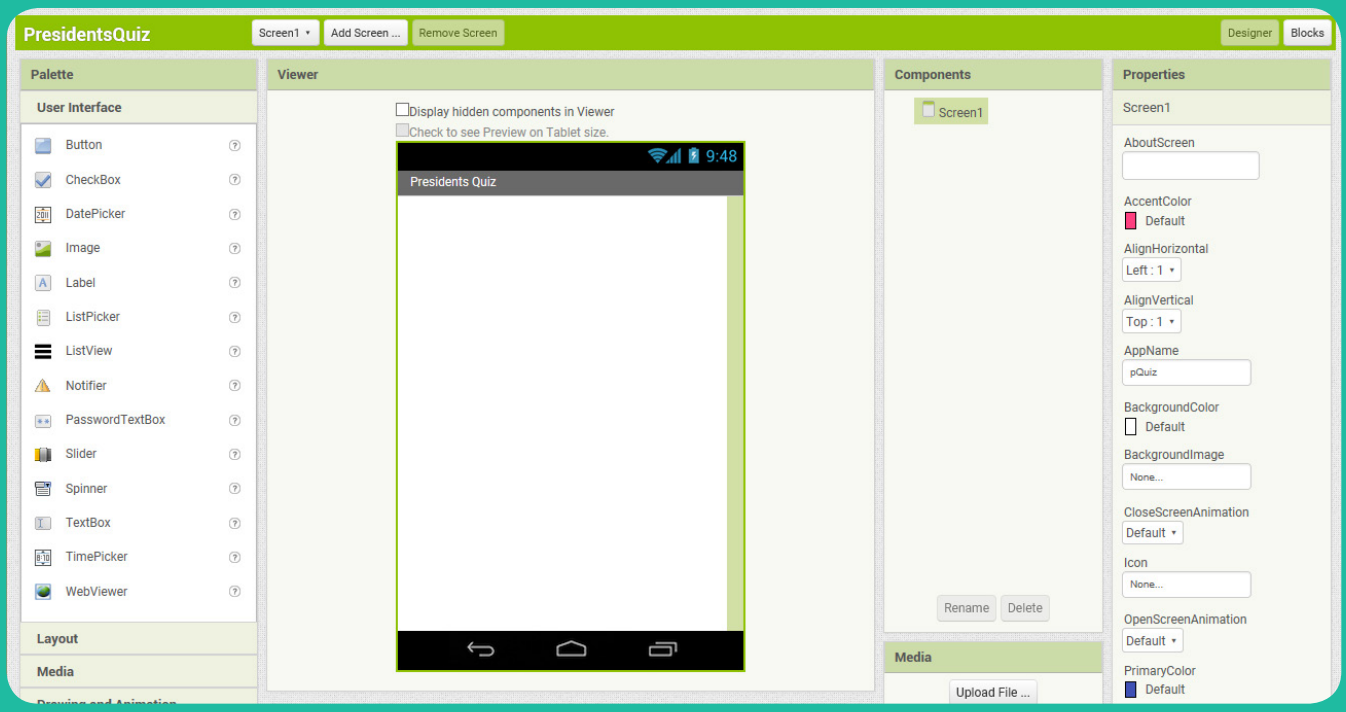


Presidents Quiz

Your app can use more than one list. In fact, you can have a list for pictures, a list for questions related to each picture, and a list for the answers for each question. As long as the related picture, question and answer all have the same index, the app will always be able to keep up.

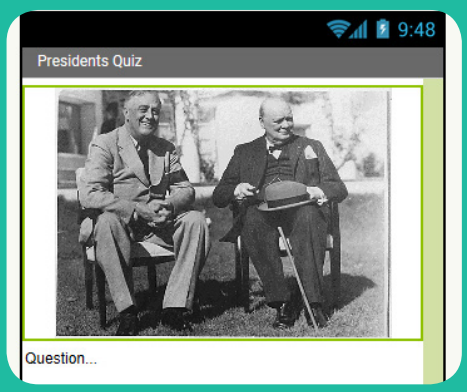
Activity 03-04: Presidents Quiz

1 Begin with a new project. Name it "PresidentsQuiz." Set the ***Screen1 Title*** property to "Presidents Quiz."

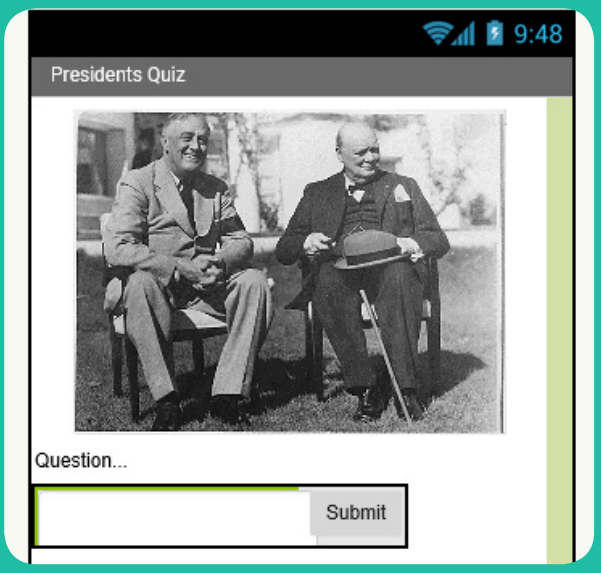


2 This app uses four images, so click ***Upload File*** in the ***Media*** section (below the ***Components*** section) and browse and upload the following four files listed at right from the Assets folder. Uploading them adds them to the project so you will always have them available.

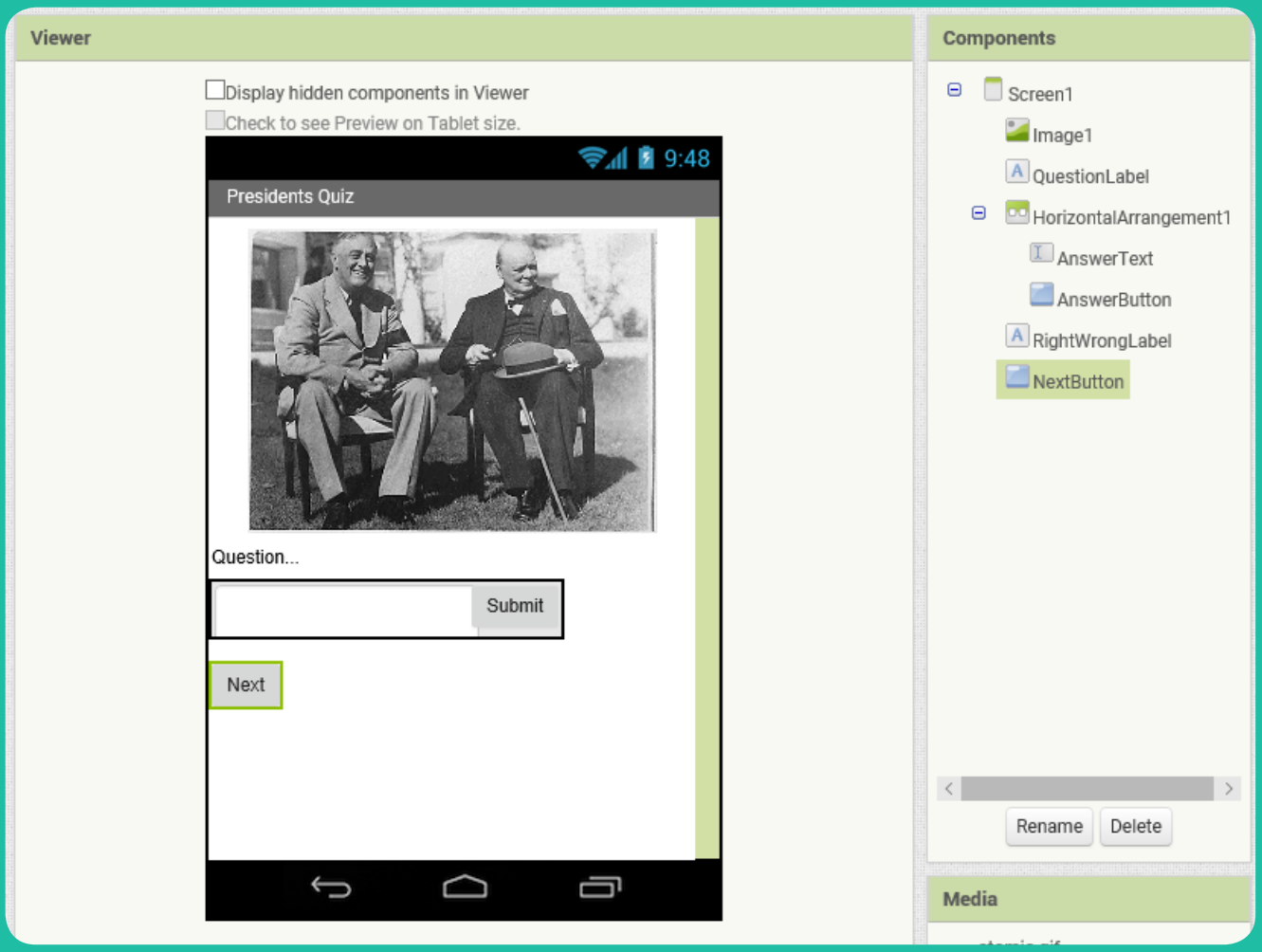
3 Select the ***User Interface Palette*** and add an ***Image*** component and a ***Label*** component. Rename ***Label1*** as "QuestionLabel." Set the width of ***Image1*** to ***Fill Parent*** and the height as 200 pixels. Set the ***Picture*** property of ***Image1*** to "roosChurch.gif" Change the text property of ***QuestionLabel*** to "Question..."



4 Click on the ***Layout Palette*** and add a ***HorizontalArrangement*** component to the ***Viewer***. From the ***User Interface Palette*** add a ***TextBox*** component and a ***Button*** component to the ***HorizontalArrangement***. Rename the ***TextBox*** as ***AnswerText*** and the ***Button*** as ***AnswerButton***. In the ***AnswerText*** properties, set ***Hint*** to "Enter an answer." Change the ***AnswerButton Text*** to "Submit."



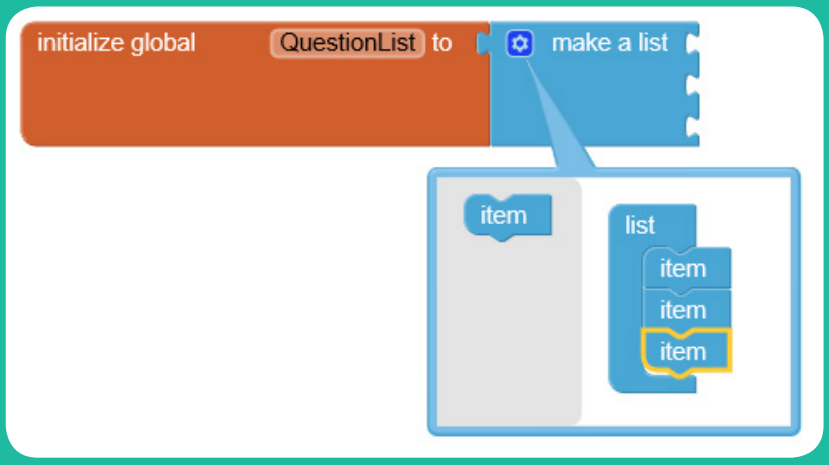
5 From the ***User Interface Palette*** add another ***Label*** component and a ***Button*** component. Rename the ***Label*** as ***RightWrongLabel*** and the ***Button*** as ***NextButton***. In the ***RightWrongLabel*** properties, set ***Text*** to " " (blank) Change the ***NextButton Text*** to "Next."



6 Switch to the Blocks Editor. When the app starts, the first question will appear along with the corresponding image. When the user clicks on the ***Next*** button, the next question and picture is revealed and so on. If the user clicks on the ***Next*** button after all of the questions have been seen, it will return to the first question. After answering a question and clicking on ***Submit***, the app will respond by saying the response is right or wrong.

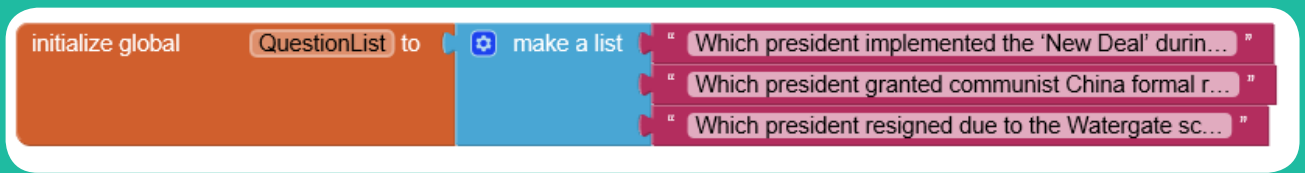
7 Just as with our slideshow app, we need to create a variable to represent our list. Select the ***Variables*** component and drag an ***initialize global name to*** block into the ***Viewer*** and change "name" to "QuestionList."

8 Click on the ***Lists*** component and drag a ***make a list*** block to the end of our variable. We need 3 questions, so click on the blue modifier icon and drag an additional item slot into the list as shown.



9 Drag a ***text*** block into each of the slots for the ***List***. Use the questions below:

1. Which president implemented the 'New Deal' during the Great Depression?
2. Which president granted communist China formal recognition in 1979?
3. Which president resigned due to the Watergate scandal?

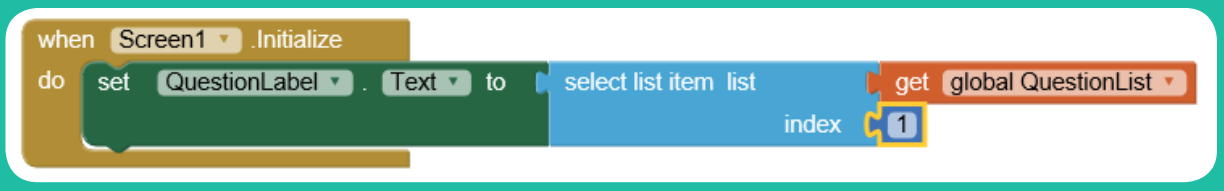


10 We also need a list for the answers. Duplicate the ***QuestionList*** block and change the name of the variable to "AnswerList." Change the text of the items to the President's names as listed below:

1. Roosevelt
2. Carter
3. Nixon

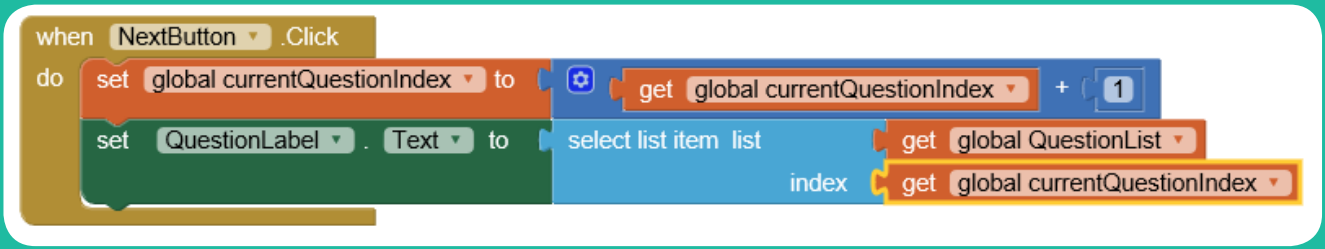
11 Also like our slideshow app, we need a variable to keep track of our list index. Add another ***iniitialze global to*** block from ***Variables*** and change "name" to "currentQuestionIndex." Add the number "1" to the slot, because we want to start at the beginning.

12 When the app first starts, we want to display the first question in the list, no matter what that question is. Select the ***Screen1*** component and drag a ***when Screen1.Initialize*** block into the ***Viewer***. Select the ***QuestionLabel*** component and drag a ***set QuestionLabel.text to*** block into the ***Screen1.Initialize*** event. Drag a ***select list item*** block from ***Lists*** into the slot and a ***get global QuestionList*** block for the list slot and the number 1 for the index slot.



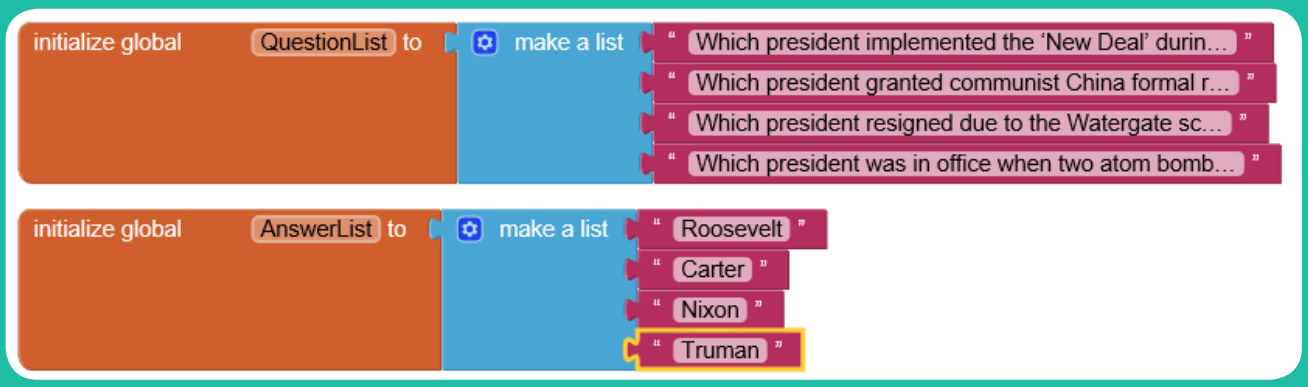
13 Connect your app to your emulator or device to test it. You should see the first question displayed. What about the rest of the questions?

14 The ***Next*** button is supposed to advance to the next question. Select the ***NextButton*** component and drag a ***when NextButton.Click*** block into the viewer. This next part should be familiar from the slideshow app. Add blocks to increase the number of the ***currentQuestionIndex*** variable and duplicate the ***set QuestionLabel.Text*** block from ***Screen1.Initialize***, replacing "1" with ***get global currentQuestionIndex***.



15 Of course, the app will fail if you press the ***Next*** button too many times. We need to add a condition to check that the Index is not greater than the total number of items in the list. Add an ***if then*** block from ***Control*** and add a > check from ***Math*** for our condition. The first slot will be for our ***currentQuestionIndex*** and the second slot is for the ***length of QuestionList***. If true, then ***set currentQuestionIndex*** to "1."

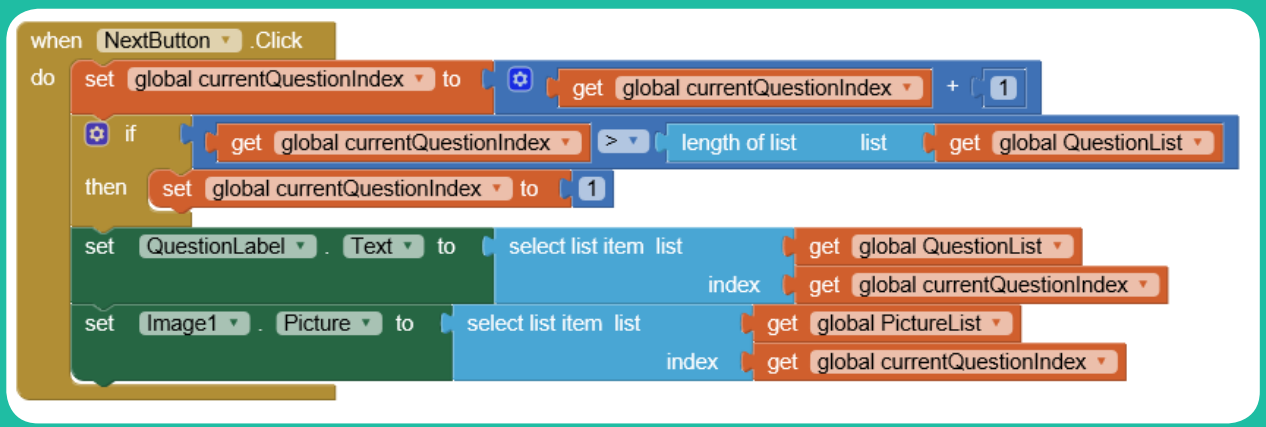
16 Now our app can handle a list of any length. Let's prove that by adding an additional question and answer to our current lists. Select the blue modify icon to add an additional item to each of the question and answer lists. For the fourth question, use "Which president was in office when two atom bombs were dropped on Japan?" and the fourth answer is "Truman."



17 At the moment, the image isn't changing. We can create a list for that, too. Duplicate one of the lists and change the name to PictureList. The names of the picture files must be EXACTLY the same as they are in the media section, including the file extention. Put the image files in this order:

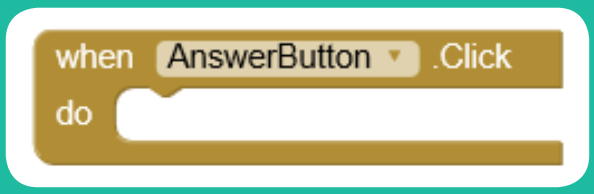
1. roosChurch.gif
2. carterChina.gif
3. nixon.gif
4. atomic.gif

18 We want to update the picture with the question, so select the ***Image1*** component and drag a ***set Image1. Picture to*** block into the ***NextButton*** event. Duplicate the ***select list item*** block from ***QuestionLabel*** and change the list to ***PictureList***.

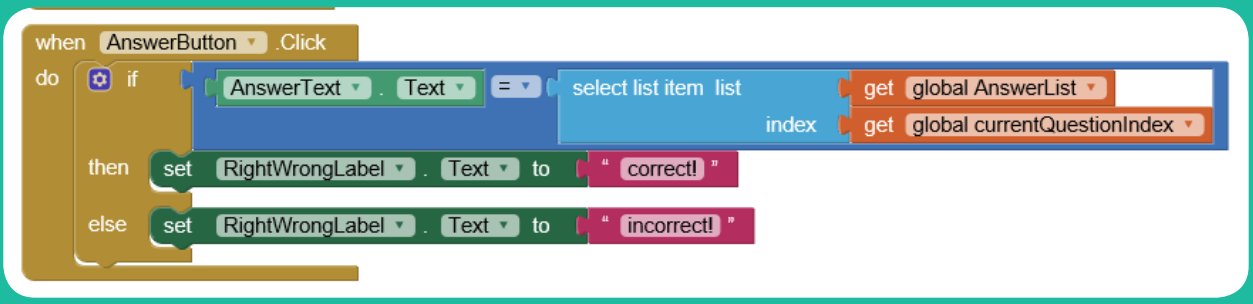


19 Connect your app to your emulator or device to test it. Now the questions and pictures should be changing each time you press the ***Next*** button. Now for the answers.

20 The user must answer the question by writing their answer in the text box and clicking the ***AnswerButton*** (Submit). Select the ***AnswerButton*** component and drag a ***when AnswerButton.Click*** event into the Viewer. If the text matches the item in the AnswerList, we will show a "correct" message. Otherwise, we will show an "incorrect" message.

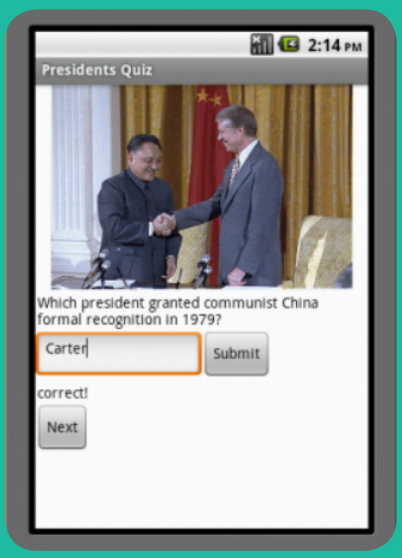


21 Select the ***Control*** components and drag an ***if then*** block into the ***AnswerButton*** event. Click on the blue modifier icon to add an ***else*** parameter to the ***if then*** block. Drag an equals block from Math for the condition. For the first slot, get an ***AnswerText.Text*** block from the ***AnswerText*** component. For the second slot, duplicate a ***select list item*** block from the ***NextButton*** event and change the list parameter to ***AnswerList***. Take a ***set RightWrongLabel.Text*** block from the ***RightWrongLabel*** component and add it to the ***then*** slot. Add a ***text*** block and insert the word, "correct." Duplicate this block for the else slot and change the text to "incorrect."



22 If we tried the app now, we would notice that the previous answer and the RightWrongLabel response are unchanged when we go to a new question. That's because we need to clear those out when the user taps on the Next button. Add a ***set AnswerText.Text to*** block to the top of the ***NextButton*** event and put an empty text block at the end. This will "reset" the TextBox to it's default state where only the hint text is seen. Let's do the same thing with the ***RightWrongLabel.Text***.

23 Connect your app to your emulator or device to test it. Turn the page for some suggestions for improving the quiz.



Adding to the Quiz

You can easily add more pictures and questions to the quiz, or even make your own quiz. Remember, every picture needs a corresponding question and answer. If you forget one, then the rest of the quiz won't make any sense. Also, the quiz is very strict about your answers. What can be done to make the responses to the quiz answers a little more flexible?

Finally, the quiz just goes back to the beginning when you finish the last question. Maybe it would be useful to keep track of the right and wrong answers and give the user a total score when they reach the end. What else could you do?

4 Timing and Sound

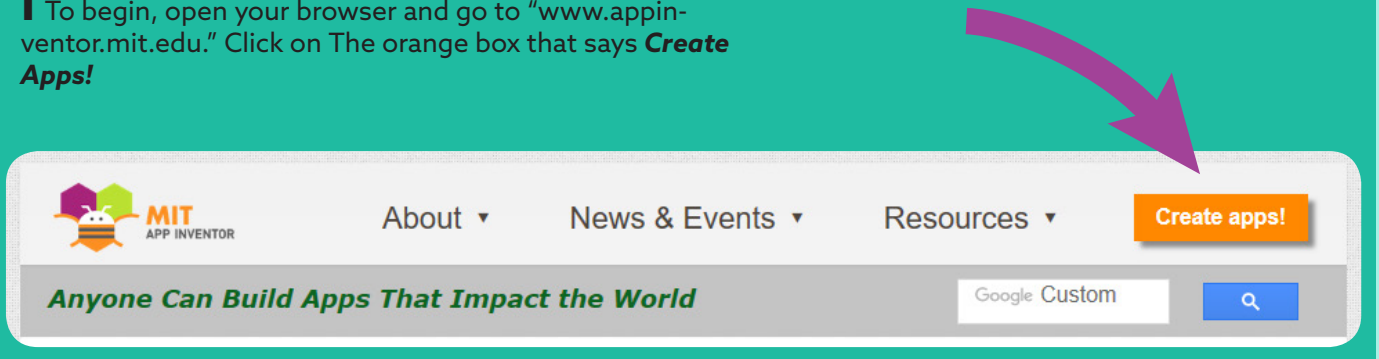
As you probably noticed, the Clock component serves an important function by keeping track of when things are supposed to happen. This is especially important when playing musical notes!

Xylophone

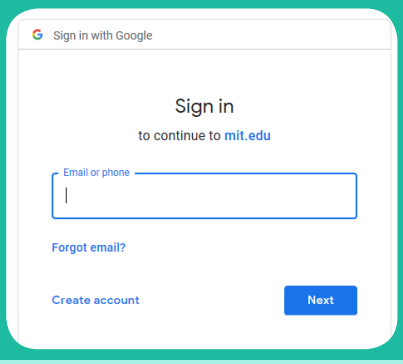
This next project will let you build a simple xylophone that plays a range of 8 notes and even saves and plays back your songs with the correct timing.

Activity 04-01: Xylophone 1

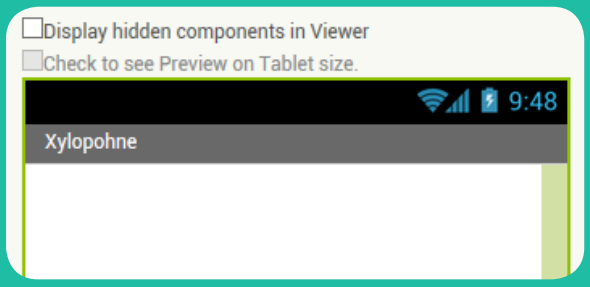
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



2 Remember, if you've been away for a while, you are going to need to sign in so that App Inventor can keep track of what you create.

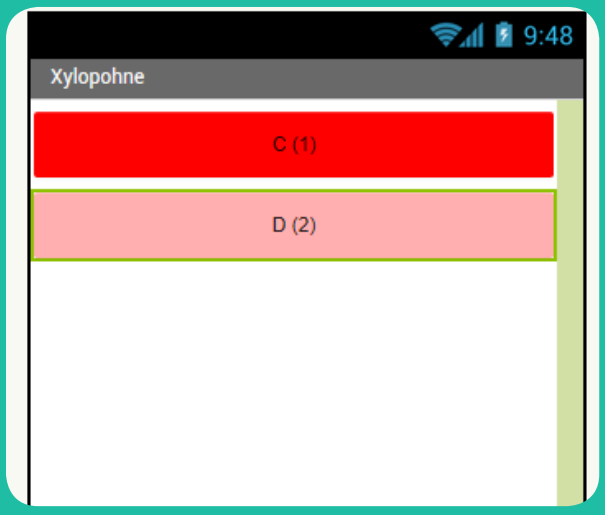


3 Create a new project. Name it "Xylophone." Set the ***Screen1 Title*** property to "Xylophone."

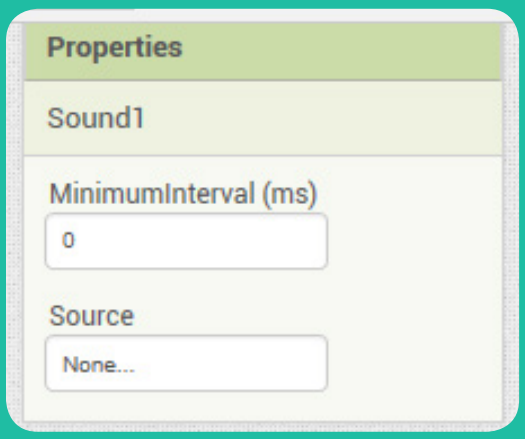


4 Drag a button from the ***User Interface Palette*** into the viewer. This will be the first of 8 buttons that represent the parts of our xylophone. Change the background color of this button to "red." Change the text to "C (1)." Set the width to ***Fill parent*** and the height to 40 pixels.

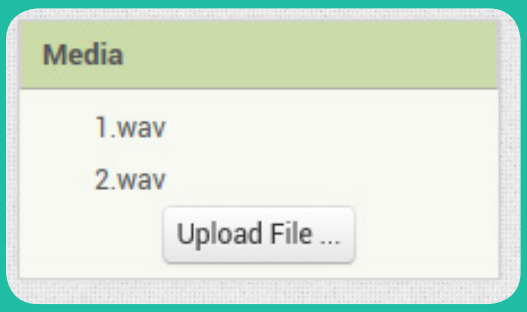
5 Repeat the steps to create a second button, below the first one with the same height and width properties. For this button, use the background color of "pink" and change the text to "D (2)." (We will create the remaining six buttons later)



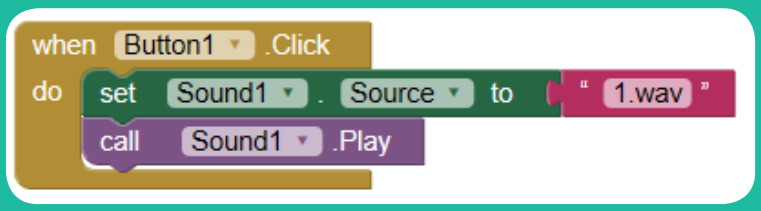
6 Our xylophone can't produce music without sound. Drag a ***Sound*** component from the ***Media Palette*** into the ***Viewer***. Change the ***MinimumInterval Property*** from 500 to 0 so we can play our sounds as often as we want. We will be setting the source property with the blocks, so leave that as it is.



7 Now let's upload the sound files for our first two buttons. Click on ***Upload File*** and select 1.wav from the Assets folder. Do the same for 2.wav. It is important that the names of these files are kept exactly as they are. We will upload the remaining six sounds later.

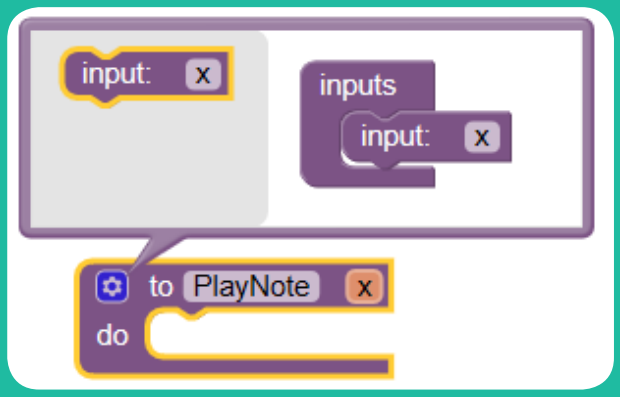


8 Now let's connect the buttons to the sounds. Switch to the Blocks Editor. Click on the ***Button1*** component and drag the ***when Button1.Click*** event into the viewer. From the ***Sound1*** component, drag a ***set Sound1.Source to*** block into the event. Add a text block to this block and enter the name of the first sound file, "1.wav" exactly as written. Add a ***call Sound1.Play*** block from the ***Sound1*** component as shown.

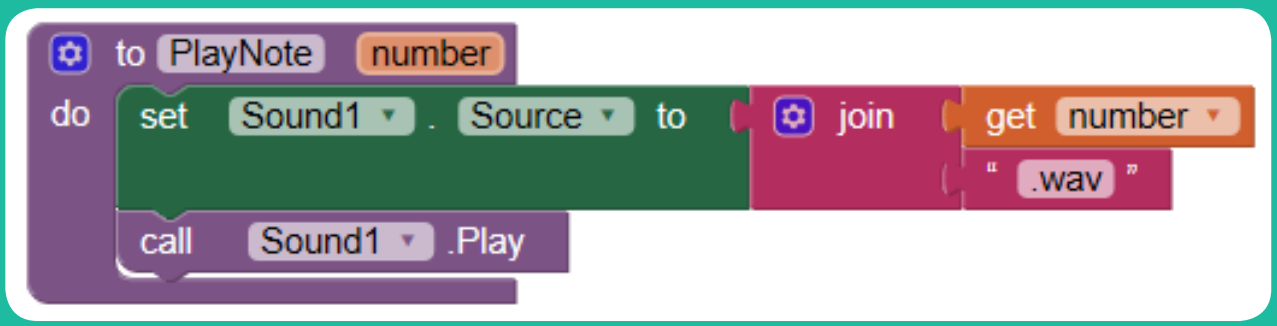


9 We could do the same for Button2 by duplicating the event, but that would be a bit repetitive when we got around to our eighth note. Instead, we want to create a procedure to take the note as a parameter and set the ***Sound1.Source*** property there.

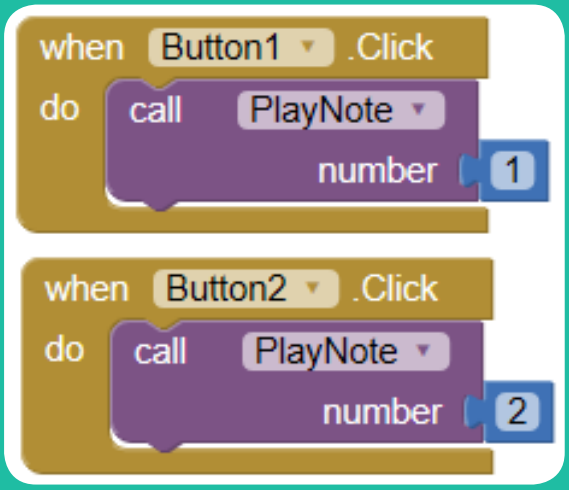
10 Select ***Procedures*** and drag the ***to Procedure do*** block into the ***Viewer***. Change the name of the procedure to "PlayNote." Click on the blue modify icon in the corner to add an input to the procudure. This means that the procedure will expect an additional parameter when you call it.



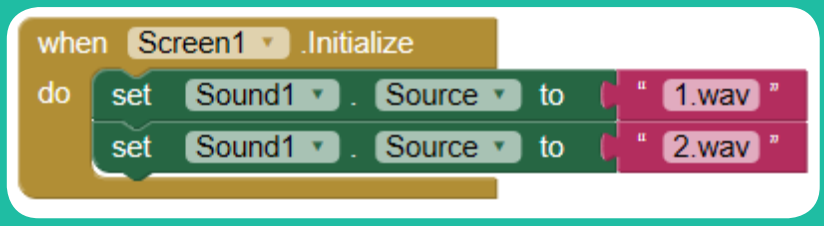
11 Change the input from "x" to "number." When the procedure is called, we want to know the number corresponding to the button that was pressed. Drag the set Sound1.Source block from Button1 into the procedure. Insert a join block from text in the source block - we are going to get the name of the sound file by joining the number parameter (***get number***) with ".wav" text. For instance, if "number" was "1," we would set the source to "1.wav" and so on. Drag the ***call Sound1.Play*** block from ***Button1*** into this procedure.



12 Select Procedures and drag a call PlayNote block into the Button1 event. It needs a number parameter, so drag a number block from Math and change it to "1." Duplicate the event for Button2 as shown:



13 If you tried to test the app right now, you may not hear anything. That's because Android still needs to load the sounds before it can play them. To make the app load the sounds when it is started, we need to click on the ***Screen1*** component and add a ***when Screen1.Initialize*** block to the Viewer. Add a ***set Sound1.Source to*** block to the event for each of the sound files as shown. Now when you start the app, the sounds are loaded into memory even before you play a single note.

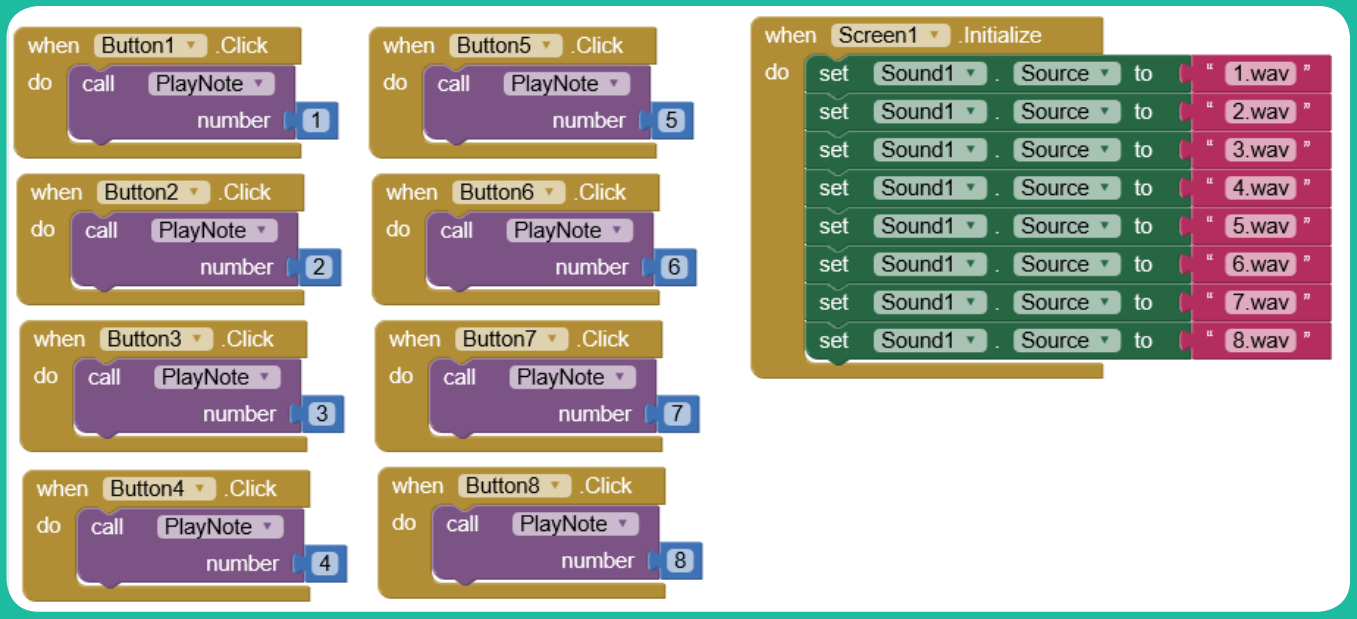


14 Connect your app to your emulator or device to test it. Do you hear the sounds?

15 Now that everything is working, Let's add the rest of the notes. Upload the sound files 3.wav through 8.wav to the project. Add the remaining six buttons as follows:

- Button3 ("E (3)", orange color)
- Button4 ("F (4)", yellow color)
- Button5 ("G (5)", green color)
- Button6 ("A (6)", cyan color)
- Button7 ("B (7)", blue color - change the text color to white)
- Button8 ("C (8)", magenta color)

16 Switch to the Blocks Editor and duplicate the ***Button1*** event and change it to the remaining 6 buttons. Also, be sure to load all of the sounds in the ***Screen1.Initialize*** event.



17 Connect your app to your emulator or device to test it. Do you hear the sounds? Next, we will set up your app to record and play back your sounds.

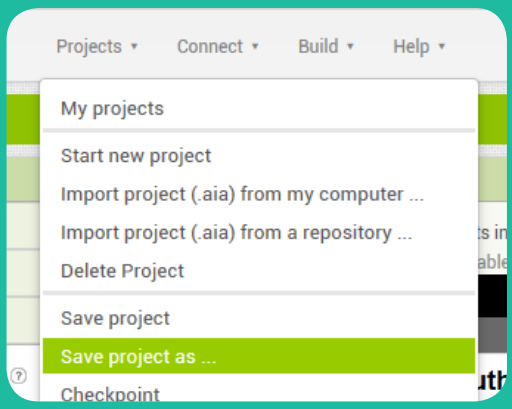
Adding to a List

Did you notice that instead of having a different sound component for each note, we used the same one and just had it play a different file? Is it possible to keep track of which notes have been played?

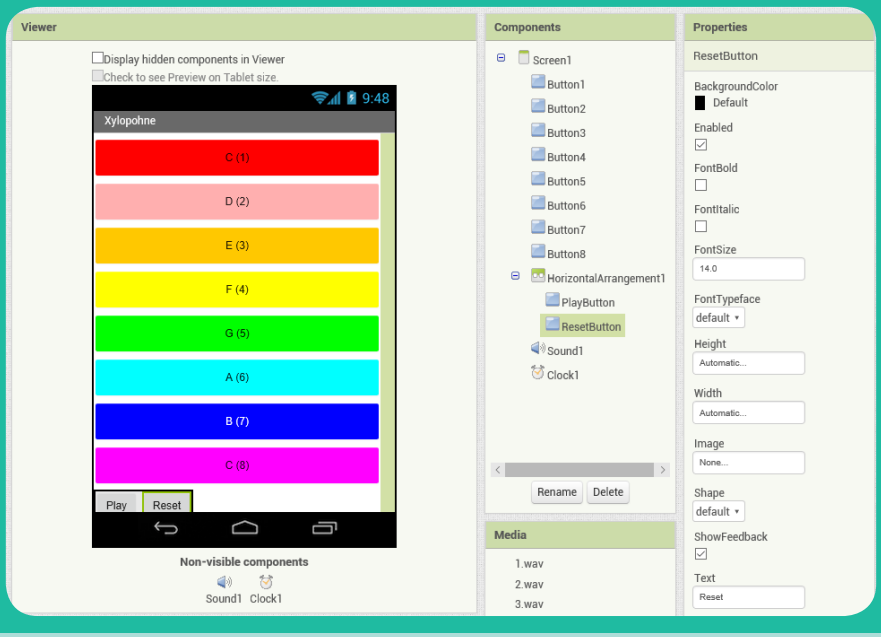
Each time the user presses a button, we can save that information in a list. Then we can play it back. We'll also need a way to "reset" the list when we want to erase everything and play a new song.

Activity 04-02: Xylophone 2

1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



2 Make sure that you're in the Designer section. Add a ***Clock*** component from the ***Sensors Palette***. Uncheck the clock's ***TimerEnabled Property***. We don't want the clock to do anything just yet. We also want buttons to play our song and to reset the recording. Add a ***HorizontalArrangement*** component from the ***Layout Palette*** and add two buttons to it from the ***User Interface Palette***. Rename the first button as "PlayButton" and change its text to "Play." Rename the second button as ResetButton and change its text to "Reset."



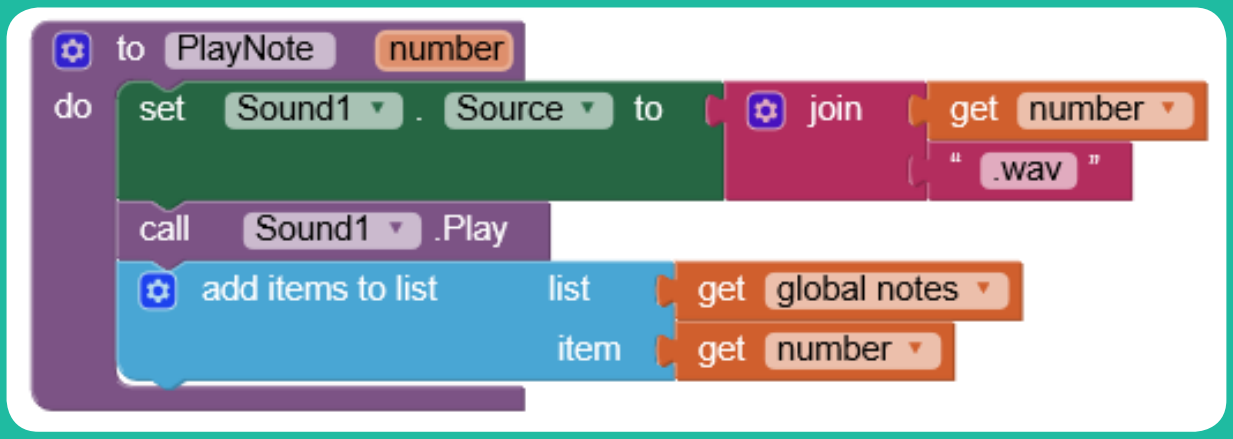
3 Knowing what notes to play is only part of a song. You also need to know how long each note must be before the next one starts. To record our music, we need to keep track of both the notes and the times.

4 Switch to the Blocks Editor. Drag an ***initialize global name to*** block from ***Variables*** into the Viewer. Change the name of the variable to "notes." The best way to keep track of a lot of notes and the order they are played is in a list. Drag a ***create empty list*** block into the Viewer and attach it to the variable.

5 Duplicate the block and initialize a new variable as an empty list and call it "times."

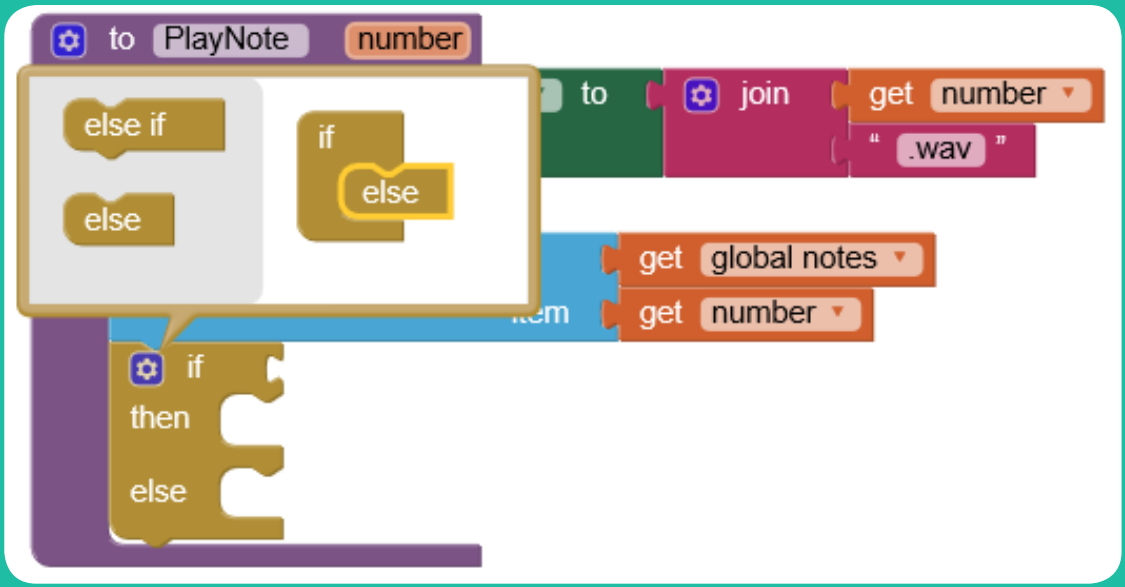
6 To record the times, we need to be able to get the duration of the interval between the time the first note is played and the next note is played. We will save the data for the previous note in a variable called "tempTime." Duplicate the initialize variable block again, rename it as "tempTime," and replace the empty list block with the number "0."

7 Everytime our ***PlayNote*** procedure is called, we should be keeping track of what note is being played and the duration (the time between one note and the next). Select an ***add items to list*** block from ***Lists*** and add it to the ***PlayNote*** procedure. Each time a note is played, we can add the number of that note to the ***notes*** list. We could have added the entire ***Sound1.Source***, but only the number is changing, so this is more efficient.

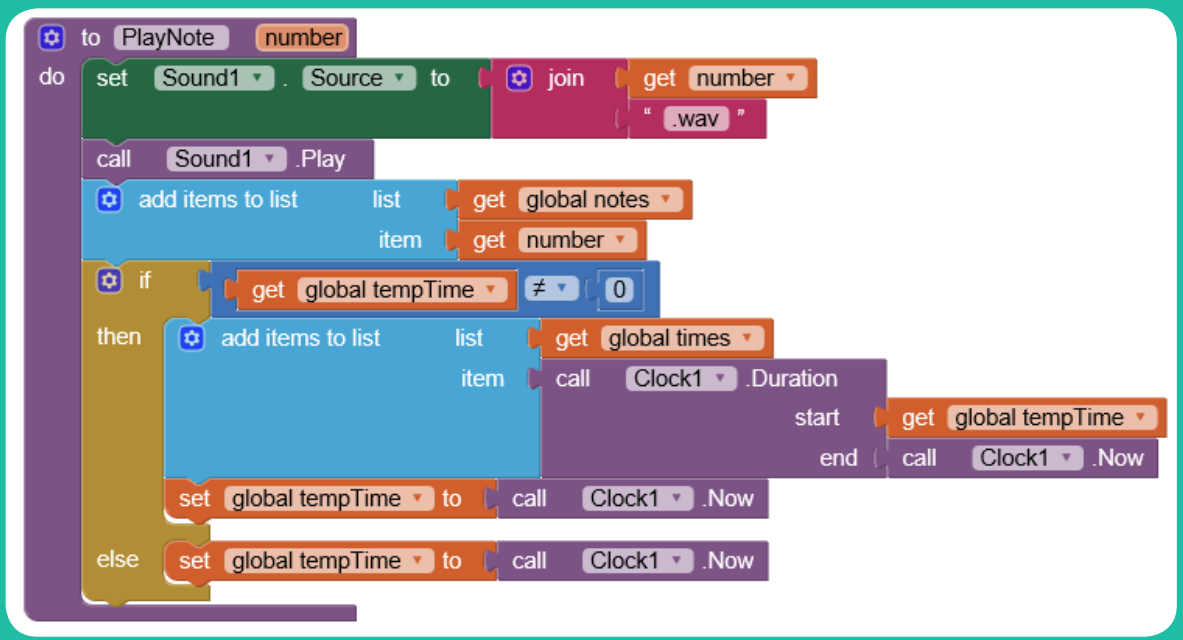


8 The duration is a bit trickier, since we won't know how much time has passed until an additional note is played. To get the duration, we need both the current time (***call Clock1.Now***) and the value of the previous time stored in the ***tempTime*** variable. Remember, our ***tempTime*** variable initially starts at "0." So if ***tempTime*** is anything other than "0," we can use both the variable and the current time to get a duration which we can add to our list. If ***tempTime*** actually is "0," then we can set ***tempTime*** to our current time without saving any duration.

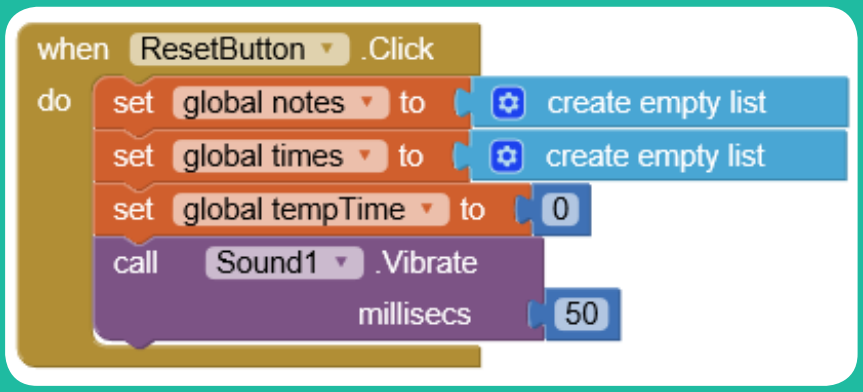
9 The first thing we need is to find out if the ***tempTime*** variable is equal to "0." Drag an ***if then*** block from ***Control*** and place it in the ***PlayNote*** procedure. Click on the blue modify icon to add ***else*** to the block. We want to do one thing if the condition is true and another if it isn't.



10 Drag the ***equal*** block from ***Math*** into the block for our condition. Change the condition to "not equal" (≠) and insert ***get global tempTime*** and the number "0" to complete the condition. If ***tempTime*** is something other than "0," we can add an interval to our times list. Duplicate the ***add items to list*** block from earlier in the procedure and insert it in the ***then*** slot. Change the variable to ***times*** and add a ***call Clock1.Duration*** block as shown. The difference in milliseconds between ***tempTime*** and ***call Clock1.Now*** is added to the list. Finally, we set the value of ***tempTime*** to ***call Clock1.Now***. If ***tempTime*** does equal "0," we just ***set tempTime*** to ***call Clock1.Now*** as shown.



11 When the user presses the Reset button, we want to clear all of the variables for notes and times. Drag a ***when ResetButton.Click*** block into the viewer and add ***set global notes to create empty list*** as shown. Duplicate that block to also ***set global times to create empty list***. Finally, ***set global tempTime*** to "0." If you are using a device instead of the emulator, you can insert ***call Sound1.Vibrate*** to vibrate the device for 50 milliseconds to confirm that the button was pressed.

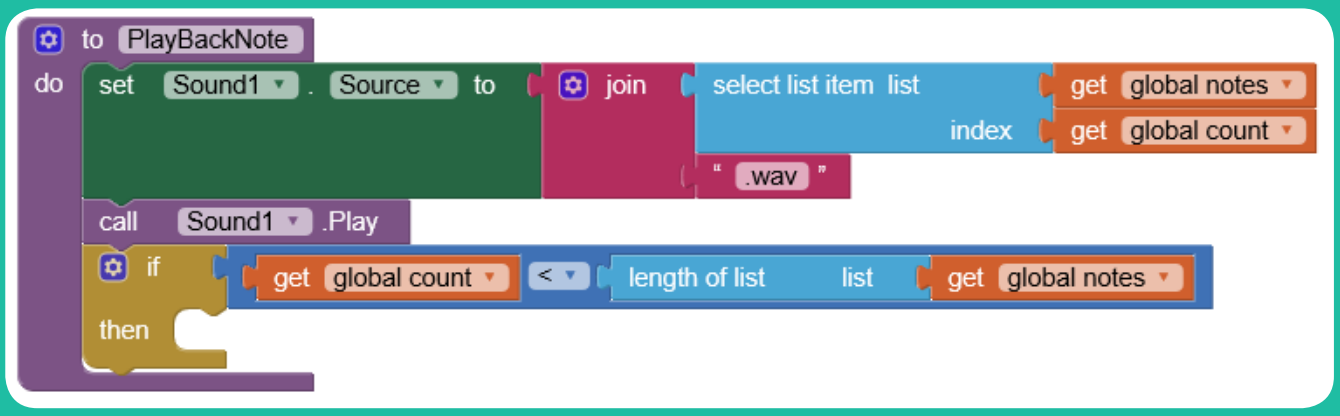


12 When we have all of the notes saved in a list, we will want to play them back starting with the first item in the list. So we will need another variable to keep track of the list index while we're playing everything back. Select ***Variables*** and drag an ***initialize global*** block into the viewer. Change the name to "count" and set it to the number "0."

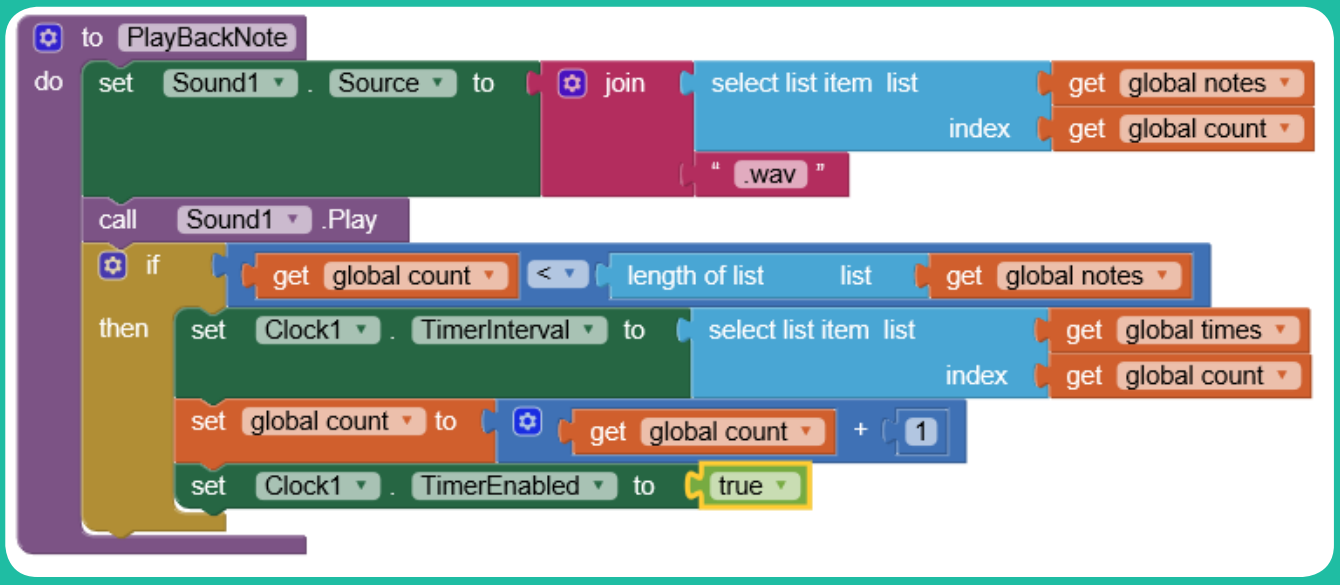
13 Just like we used a procedure to play each note, we are going to create a procedure to play back our stored music. Drag a ***to procedure do*** block from ***Procedures*** into the ***Viewer*** and name it "PlayBackNote." When the procedure is called, we want to play a sound, just like the ***PlayNote*** procedure. Duplicate the ***set Sound1.Source*** and ***call Sound1.Play*** blocks from the ***PlayNote*** procedure and insert them into the ***PlayBackNote*** procedure. The number is coming from our list, so replace the ***get number*** block with ***select item from list*** and select ***notes*** for the list and ***count*** for the index.

14 Next, we need to find out if there are any more items in the list and if so, play those, too. Remember, the app is going to want to do everything at once, so we need to use the intervals that we saved when saving the notes to have the app pause before playing the next note.

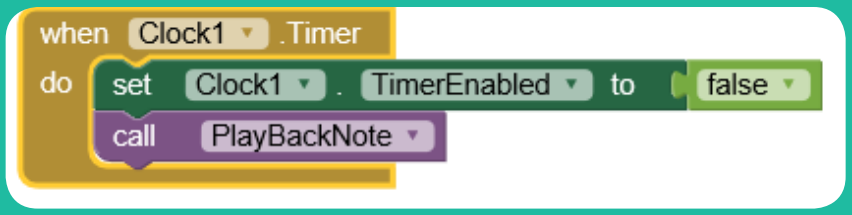
15 To check if there are any items left in the list, we need a condition. Drag an ***if then*** block from ***Control*** and insert it into the ***PlayBackNote*** procedure. For our condition, we are checking if ***count*** is less than the ***length of list*** of ***notes***. Add the "=" block from ***Math***, set it to < and insert the variables as shown.



16 If the condition is true, we will ***set Clock1.TimerInterval to*** the matching interval stored in the ***times*** list as shown. Then we will increase the ***count*** variable and ***set Clock1.TimerEnabled to true*** so that after the interval has passed, we can call this procedure again and play the next note.



17 We need to tell the app what to do when the timer is triggered. Select the ***Clock1*** component and drag a ***when Clock1.Timer do*** event into the viewer. Inside the event, add a block to ***set Clock1.TimerEnabled to false*** and then a block to call the ***PlayBackNote*** procedure. Thanks to the condition in the procedure, this will keep going until there are no more notes to play!



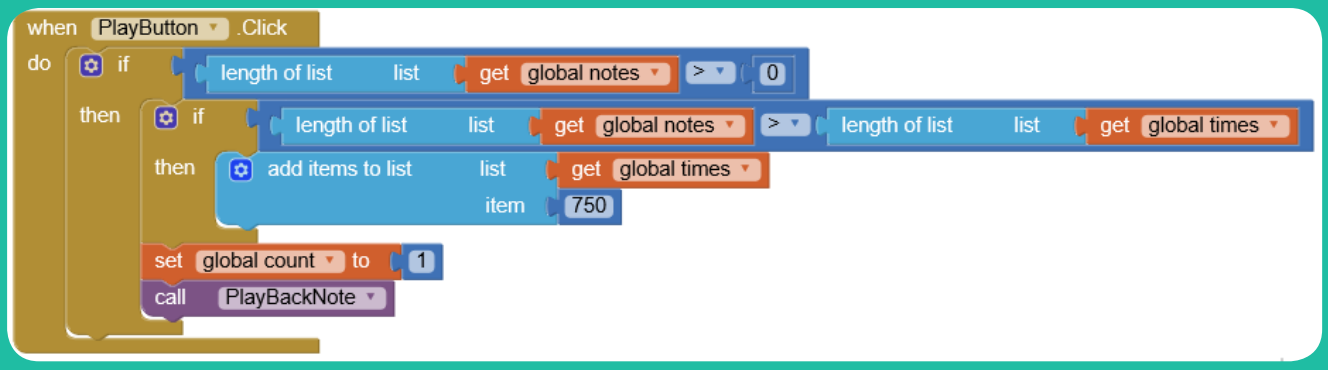
18 The app won't play any notes until we actually have blocks for ***when PlayButton.Clicked*** event! Drag the event into the Viewer. Before we can play any notes, we need to be sure that there are notes in the list to play. Add an ***if then*** from ***Control*** to the event and insert ***length of list get global notes > 0*** for our condition. If the condition is true, then we will ***set global count to 1*** (to start at the beginning of the list) and ***call PlayBackNote*** to start playing our notes.

19 Connect your app to your emulator or device to test it. Can you hit a few notes and hear them played back? What happens if, instead of hitting Reset, you record a few more notes and hit Pay again? The time that you listened to the playback is now part of the interval! Let's fix that!

20 In the ***PlayBackNote*** procedure, there is a condition to keep playing notes as long as there are items in the list. When the list is empty, we want to reset the ***tempTime*** variable to "0." Click on the blue modifier icon in the ***if then*** block and add an ***else*** to the condition. Then we can insert ***set global tempTime to 0***.

21 There's one more thing that can create problems for us. When we first start saving notes, we don't start saving the intervals until the second note is played. This isn't a big deal if we only play a song once, since the last interval technically lasts forever. But if we start recording notes again without resetting the app, the difference in length between the two lists will keep growing and start giving us errors. How do we fix that?

22 When we press the ***PlayButton***, we can check to see if the two lists are equal in length and if they are not, we can add an interval to the times list to make them equal in length again. After the condition to check the length of the notes list, insert another ***if then*** and set the condition to see if the length of ***times*** is less than the length of ***notes***. If so, add an item to the times list. We're using 750 milliseconds (three quarters of a second) which seems to work well.



23 Connect your app to your emulator or device to test it. Wouldn't it be nice to have the app have some songs stored in it to play back? We'll do that next!

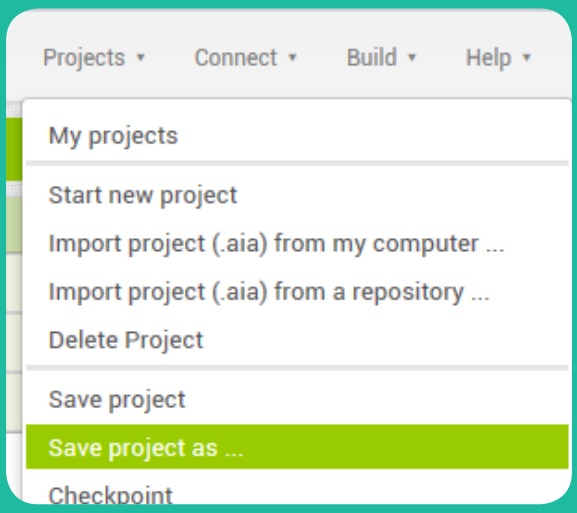
Saved Songs

The xylophone app works pretty well at the moment, but unless you're pretty good with music, it's not going to sound all that impressive to anyone else. Fortunately, we now have everything we need to add our own music library to the app.

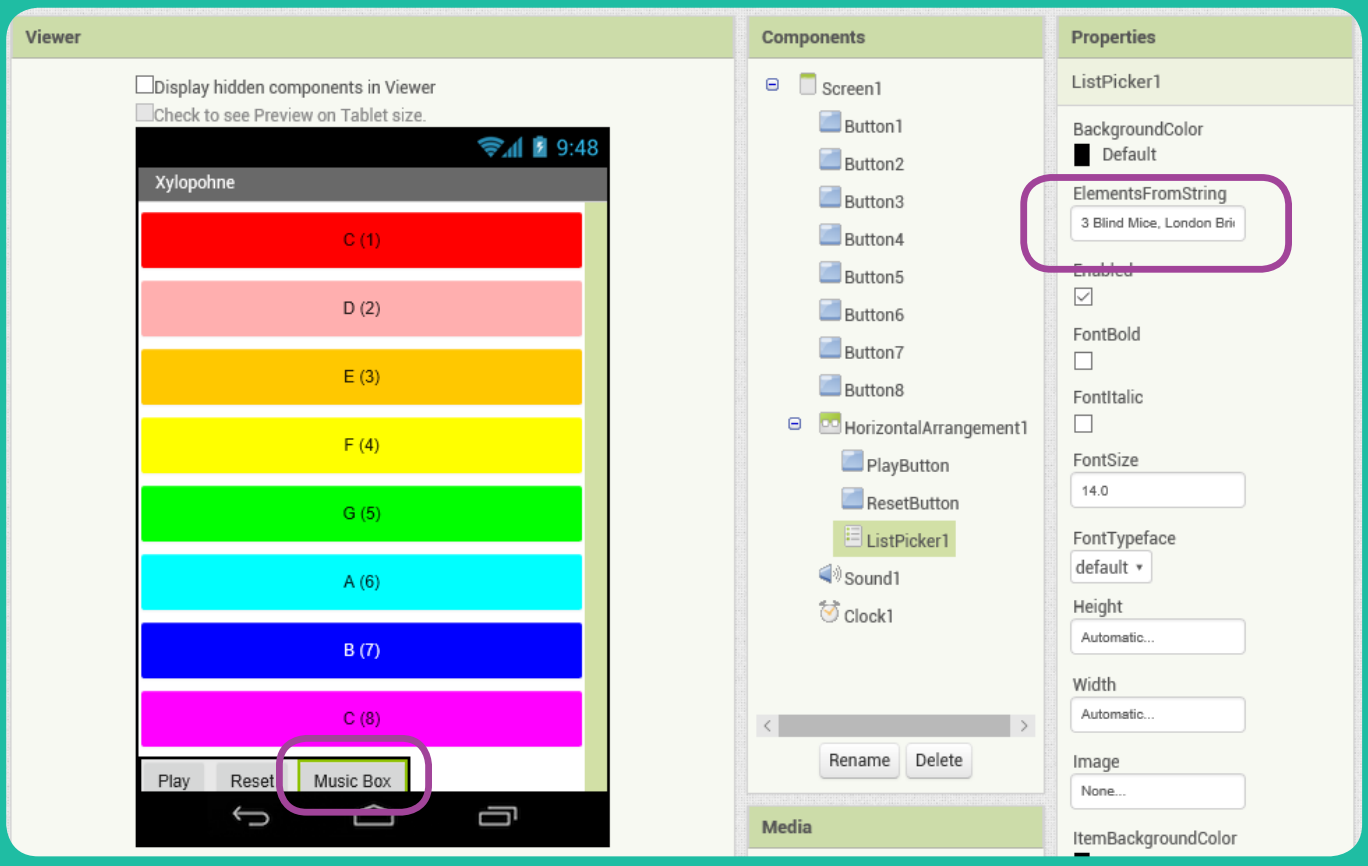
Get ready for the music box!

Activity 04-03: Music Box

1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



2 Make sure that you're in the Designer section. Add a ***ListPicker*** component from the ***User Interface Palette*** into the ***HorizontalArrangement***. The ***ListPicker*** looks like a button, but it's actually a menu that gives you many options. Change the ***Title*** property of ***ListPicker*** to "Music Box." At the top of the ***ListPicker*** properties is ***ElementsFromString***. This is where we will put our menu entries. This can be set in blocks, or we can set it here. For this activity, we will start with two options. Each choice in the ***ListPicker Elements*** property is separated by a comma. Our two elements are "3 Blind Mice" and "London Bridge." Put those two titles in the ***ListPicker Elements*** property, separated by a comma as shown.



3 When we were saving our songs for playback, we were just saving the numbers that represented each note. The list of notes for our two songs, "3 Blind Mice" and "London Bridge" are below:

- **3 Blind Mice:** 32132154435443588767855588876785558876785554321
- **London Bridge:** 56543452343455654345253 1

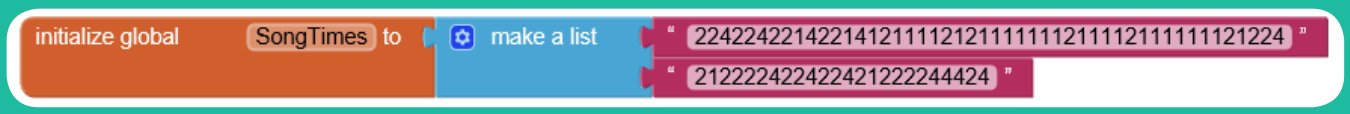
That may not seem like much until you realize that you would have to create a list and insert additional item slots in the list for each of those numbers. And add number blocks to each of those slots! There must be a better way!

4 Switch to the Blocks Editor and initialize a variable and give it the name, "SongList." Our list has two songs, so add a ***make a list*** block. For each of our songs, add a text block with the list of numbers from the previous page.

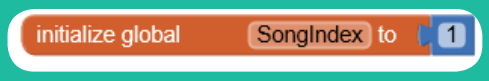
5 You may be wondering how we can make a list from text like this. There are only 8 notes, so each character in the text can represent a single note. We can use the text properties to identify the notes in the text, one character at a time.

6 Before we do that, we also need the intervals for our timing. Duplicate the SongList block and change the variable name to SongTimes. Use the information below for the song times text.

- **3 Blind Mice:** 22422422142214121111212111111121111211111121224
- **London Bridge:** 21222242242242122224442 4



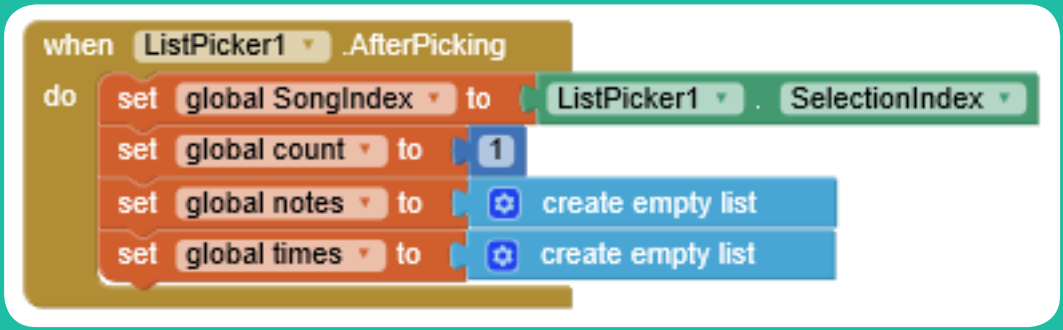
7 We need one more variable to keep track of which song we're playing. We will call this "SongIndex."



8 When the user taps on the ***ListPicker*** component, it will show them a list containing the elements that we put into the ***ListPicker Element*** properties. When a user picks one of the items in the list, that information is stored as ***ListPicker.SelectionIndex***. That index can be used just like any other list index.

9 At the moment, it's difficult to tell which item is which song. Right-click on the ***SongList*** block and click on ***Add Comment***. This adds a blue question mark to the block. Click on it and add the titles of your songs. Now you will always know which song is which.

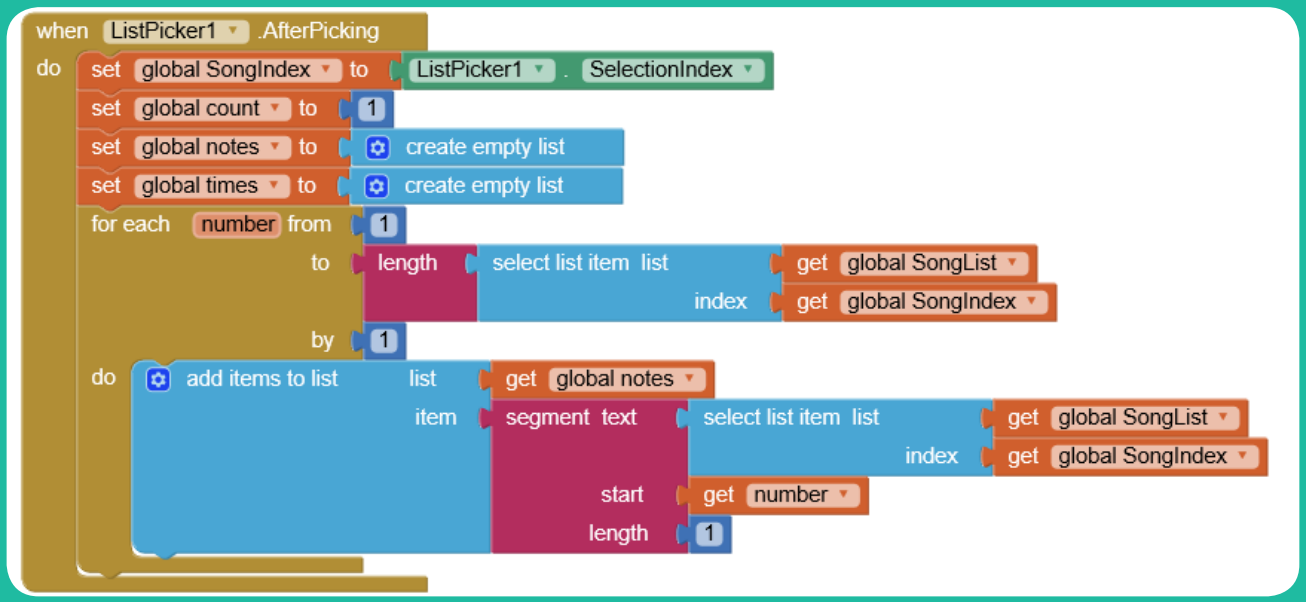
10 Click on the ListPicker1 component and drag a ***when ListPicker1.AfterPicking do*** block into the Viewer. Add a ***set global SongIndex to*** block into the event and set it to ***ListPicker1.SelectionIndex***. We also want to reset the ***notes*** and ***times*** lists as well as the ***count*** variable.



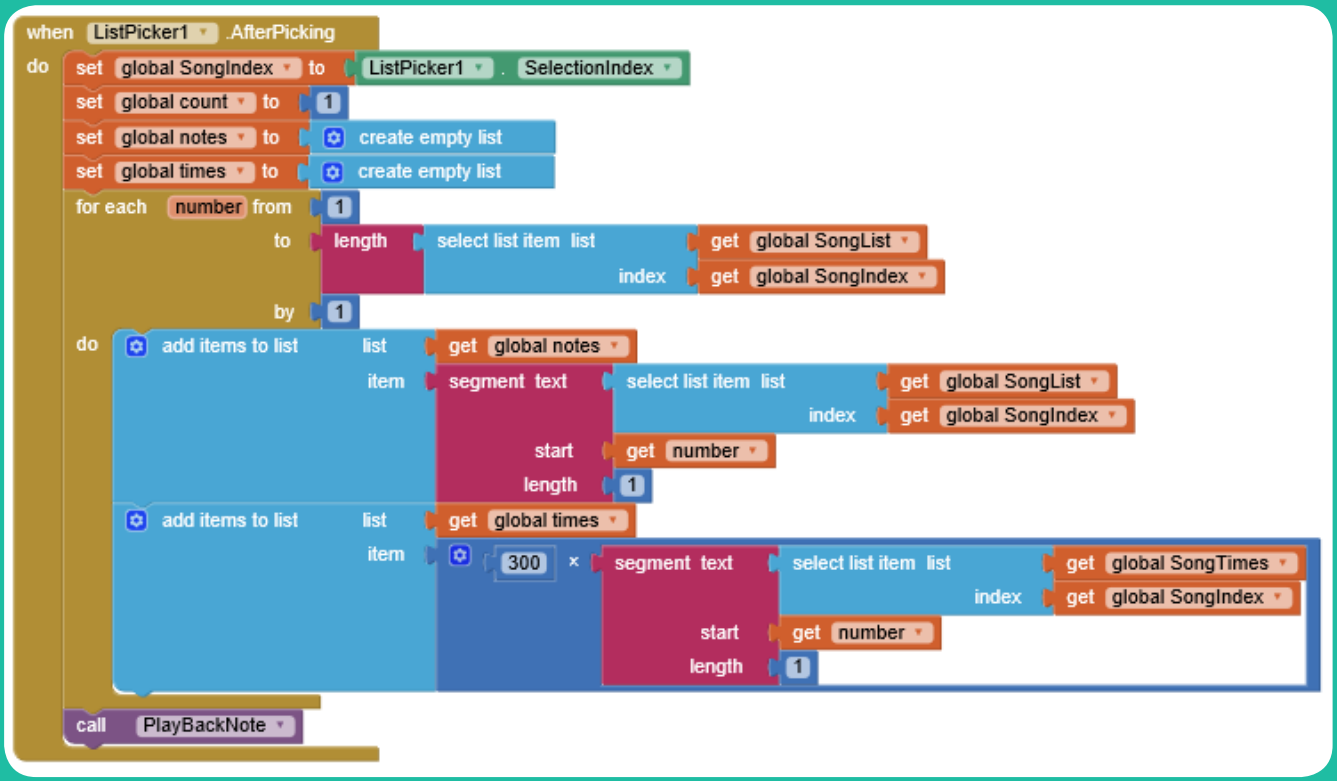
11 The next step is to take the notes and times from the selected text and put them in the appropriate lists. In ***Control***, select the ***for each number from to by*** block and put it in the ***ListPicker1*** event. What this block does is that it starts at one number and performs the same steps over and over again until it reaches the other number. By default, it starts at "1," and performs the step (number) by adding "1" to the start number until "5" is reached.

12 We want this block to do steps for each character in our text, so we need to replace "5" with ***length*** (from ***Text***) and ***select item from list*** and use the ***SongList*** and ***SongIndex*** variables. So whatever song we have picked, we will get the total number of characters in that text and perform an action once for each character.

13 So, starting at "1," we are going to add a single character from our ***SongList*** text to the ***notes*** list. Drag an ***add items to list*** block into the ***do*** slot of our this block Use ***global notes*** for the list. For the item slot, drag a ***segment text*** block from ***Text***. The first slot is to tell it which text we're get our segment from. Duplicate the ***select list item*** block from above and insert that. Then, we need to specify which character we want. For this, we will use the ***number*** parameter. As each step is executed, the value of ***number*** gets higher and so we will always be getting the next character in the text. The last slot is for the number of characters we want to grab, which is "1."



14 For times, duplicate the block for ***notes*** and insert it beneath. Change the ***global notes*** variable to ***global times*** and change ***global SongList*** to ***global SongTimes***. The since the intervals are in milliseconds, these times are going to be far too short. Grab a multiplication block from ***Math*** and insert it so that each number from ***SongTimes*** is multiplied by "300" to get a more reasonable interval. Finally, add call ***PlayBackNote*** block so that the app can play the song just like any other song you might have made.



15 Connect your app to your emulator or device to test it. Want some more songs? Add some these to your list. Don't forget to add the song titles to the ***ListPicker ElementsFromString*** property!

1. **3 Blind Mice:**
   Notes: 32132154435443588767855588767855558876785554321
   Times: 22422422142214121111212111111121111211111121224
2. **London Bridge:**
   Notes: 565434523434556543452531
   Times: 21222242242242122224424
3. **Jingle Bells:**
   Notes: 33333351234444433333223253333333512344444333355421
   Times: 22422422218222122211222244224224222142221222112228
4. **Yankee Doodle**
   Notes: 32132154435443588767855588767855558876785554321
   Times: 11111111111221111111111121
5. **Mulberry Bush:**
   Notes: 32132154435443588767855588767855558876785554321
   Times: 11121111211112111121111212111122

6. **Muffin Man:**
   Notes: 1144564432554311114456444551 14
   Times: 12221222122212221222122212222 4
7. **Baby Bumble Bee:**
   Notes: 14654221145566565321465422114
   Times: 21111222242222111122111122224
8. **It Ain't Gonna Rain:**
   Notes: 14444444114443457775553311111 1234
   Times: 22112222222112242112112221121 1224
9. **Mary Had a Little Lamb:**
   Notes: 32123332223553212333322321
   Times: 21222242242242122222222228
10. **Pop Goes the Weasel:**
    Notes: 11223531111223111122353624 31
    Times: 21211112121212212121116322122
11. **Reveille:**
    Notes: 14641646416464146414641646416464114666668646464646666864646 4114
    Times: 22112221122211224222112221122211224222224222222422222242222224
12. **Ring Around the Rosie:**
    Notes: 553653355365353533551
    Times: 222242222224444422444
13. **Row Row Row Your Boat:**
    Notes: 11123323458885553311154321
    Times: 44424424282222222222242428
14. **This Old Man:**
    Notes: 535535654323434511111234552243 21
    Times: 22422422222221122112111142222224
15. **Twinkle Twinkle Little Star:**
    Notes: 11556654433221554433255443321 1556654433221
    Times: 22222262222226222222622222262222226222222 6

Bonus Steps

It would be nice to have the buttons change appearance when their note is being played. How would you do that?

Also, it is possible to break the app by pressing Play or Reset while the app is already playing something. Perhaps you could fix that by "hiding" the buttons during play back and bringing them back after the app is finished playing. Maybe even have a message that says "playing..." until playback is finished.

If you do set the app up that way, you might want to add a button to stop playback by setting the count variable to the length of the notes variable.

5 Making a Space Shooter Game

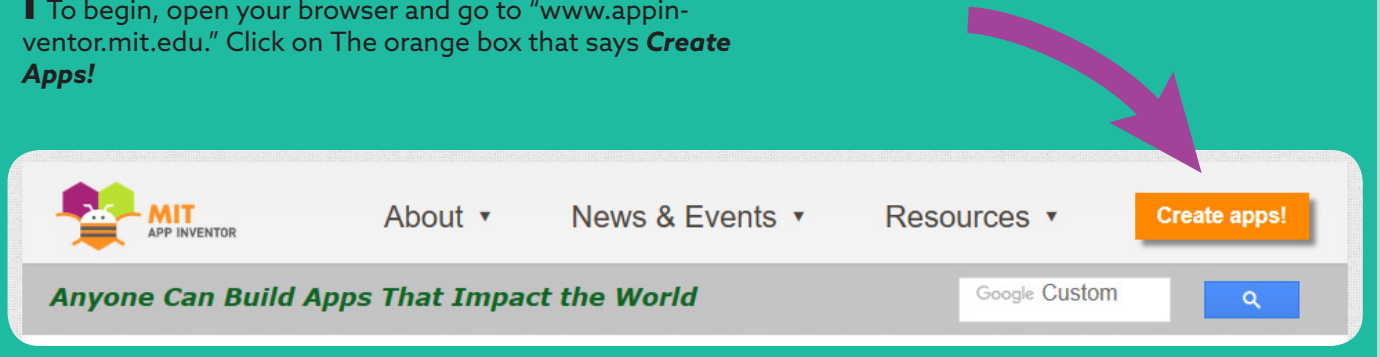
At this point, you should have all of the basic skills to make any kind of game you want with App Inventor. Such as this simple shooter, "Alien Attack."

Alien Attack

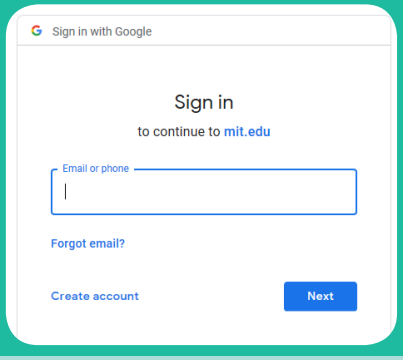
In this game, we have menacing, green aliens coming in to attack. It is up to the trusty ninja to drive them away. The attacking aliens show up in random locations on the screen. The player drags the ninja left and right to aim and the ninja shots when the user lifts their finger.

Activity 05-01: Alien Attack 1

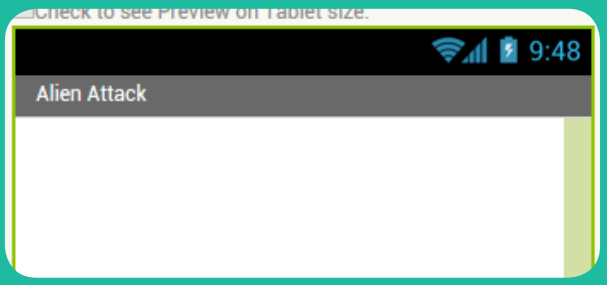
1 To begin, open your browser and go to "www.appinventor.mit.edu." Click on The orange box that says ***Create Apps!***



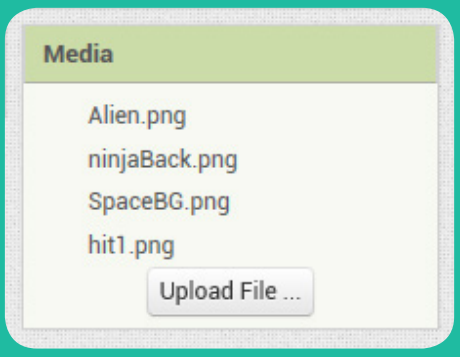
2 Remember, if you've been away for a while, you are going to need to sign in so that App Inventor can keep track of what you create.



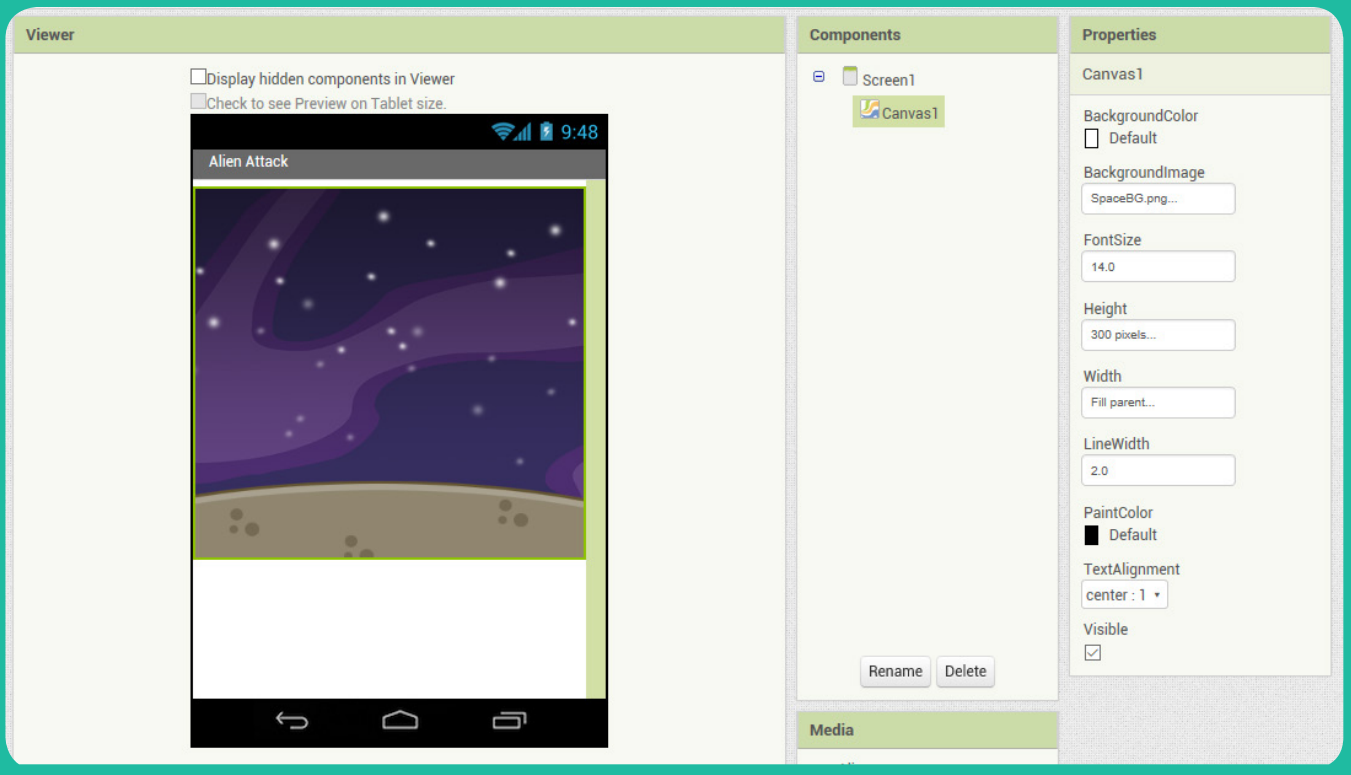
3 Create a new project. Name it "AlienAttack" (no spaces). Set the ***Screen1 Title*** property to "Alien Attack."



4 In the ***Media*** section,upload the following assets: *Alien.jpg, ninjaBack.png, SpaceBG.png, hit1.png*. We will be uploading more files later.

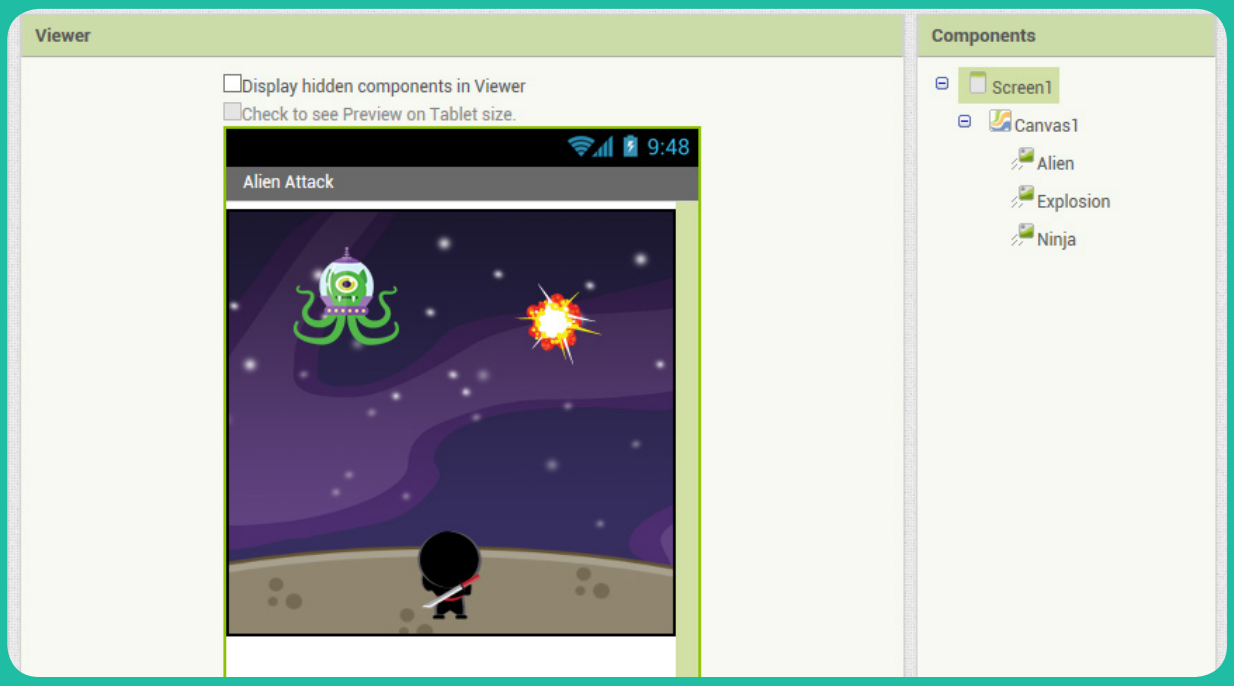


5 From the ***Drawing and Animation Palette***, drag a ***Canvas*** component into the viewer. In ***Properties***, change the ***Width*** to ***Fill Parent***, the ***Height*** to 300 pixels and the ***BackgroundImage*** to *SpaceBG.png*.

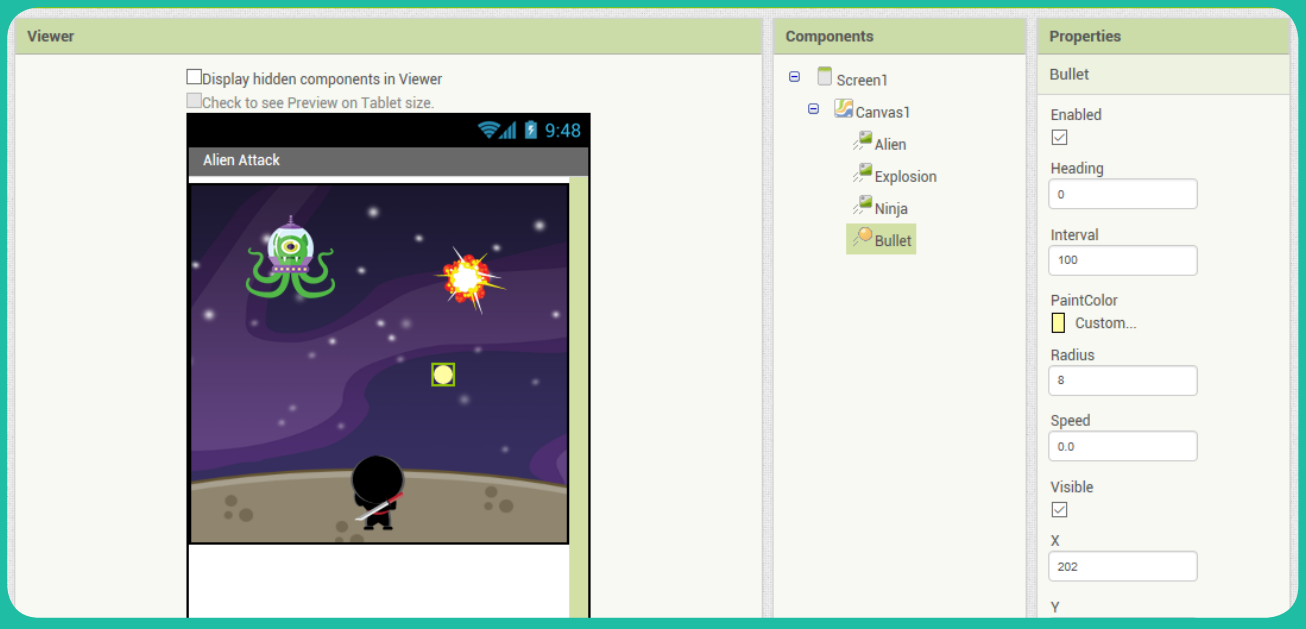


6 While the ***Drawing and Animation Palette*** is still open, drag three image sprites into the canvas. Rename the components and set their ***Picture*** properties as shown below:

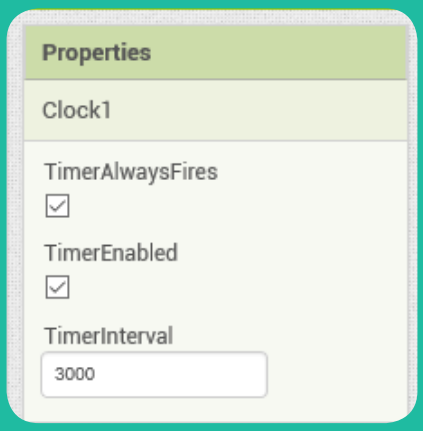
Component:	Rename to:	Picture:	Action:
ImageSprite1	Ninja	ninjaBack.png	Set y to 225
ImageSprite2	Alien	Alien.png	
ImageSprite3	Explosion	hit1.png	



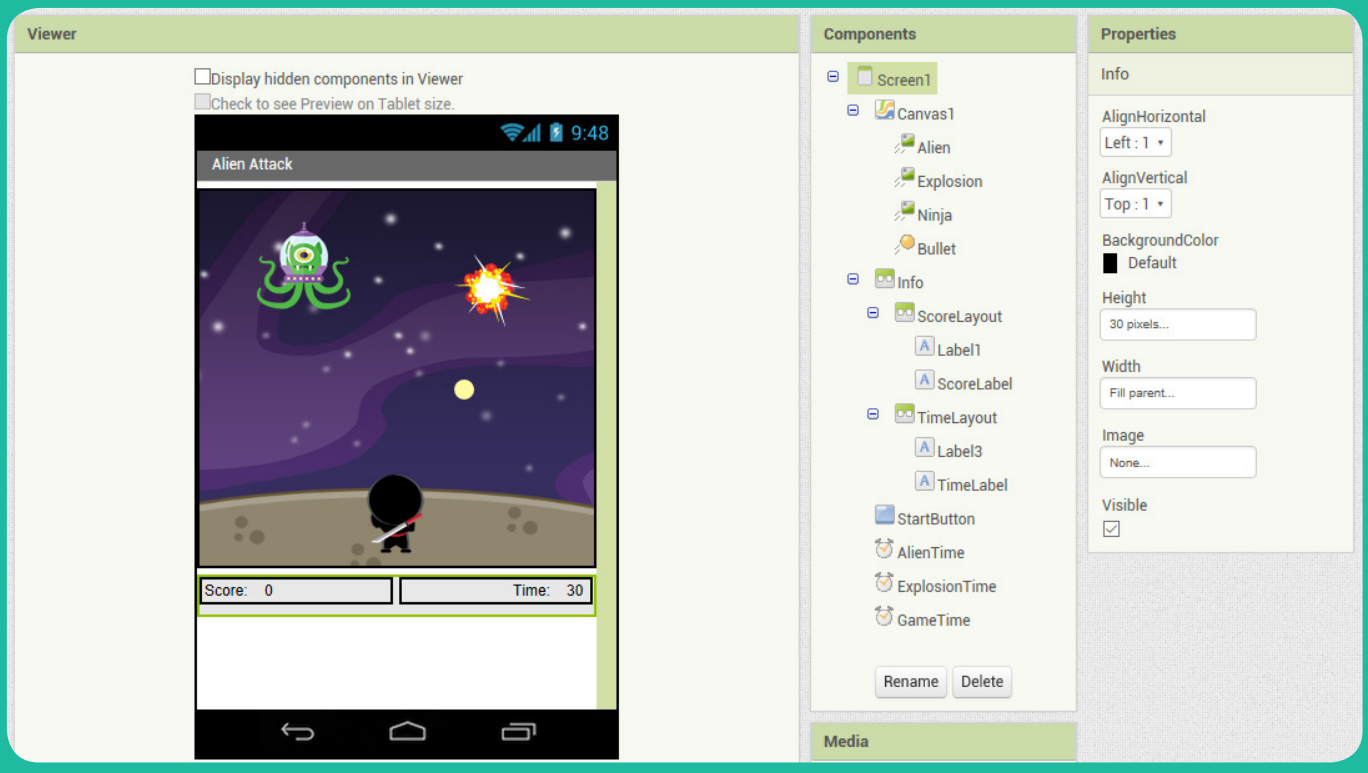
7 Add a ***Ball*** sprite from the ***Drawing and Animation Palette***. Rename it as "Bullet" and give it a ***Radius*** of 8. Change the ***PaintColor*** to something that shows up well against the background.



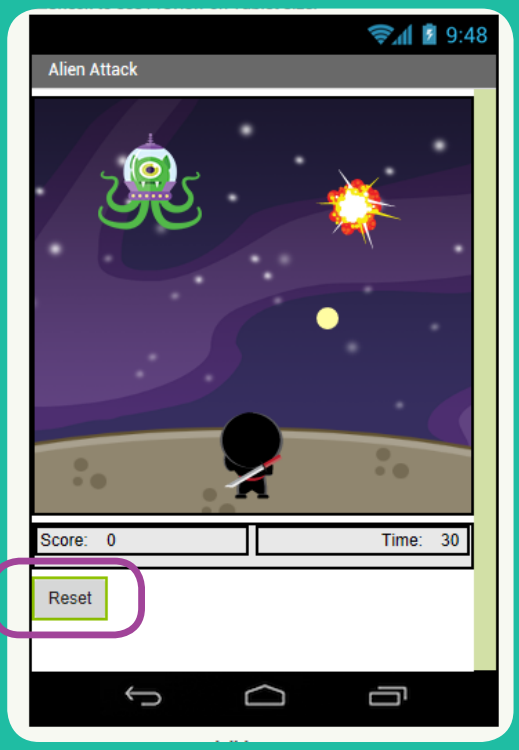
8 Add a ***Clock*** component from the ***Sensors Palette*** and rename it to "AlienTime." Set the ***Timer Interval*** property to 3000. Add a second ***Clock*** and rename it to "ExplosionTime" and set the ***TimerInterval*** to 500. Add a third ***Clock*** component and rename it to "GameTime." Keep the ***TimerInterval*** at 1000.



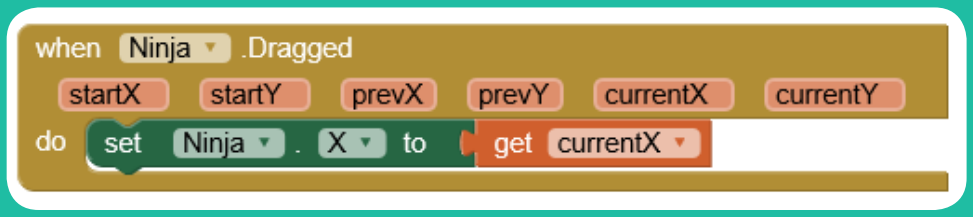
9 We need a place for our game information. Add a ***HorizontalArrangement*** component from ***Layout*** and rename it as "Info." Change the ***Width*** to ***Fill Parent*** and the ***Height*** to 30 pixels. Add another ***HorizontalArrangement*** and rename it as ***ScoreLayout***. Set its ***Width*** to Fill Parent. Insert two labels from the ***User Interface Palette*** into ***ScoreLayout***. Change the text in the first label to "Score: " and rename the second label to "ScoreLabel" and change its text to "0." Add a third ***HorizontalArrangement*** and rename it "TimeLayout." Change its ***Width*** to ***Fill Parent*** and ***AlignHorizontal*** to ***Right***. Add two labels to the ***TimeLayout*** component, changing the text of the first one to "Time: ". Rename the second one as "TimeLabel" and change its text to "30." Finally, drag the ***ScoreLayout*** and ***TimeLayout*** components into ***Info*** as shown below:



10 Add a Button component from the User Interface Palette and rename it as "StartButton." Change the text to "Reset."

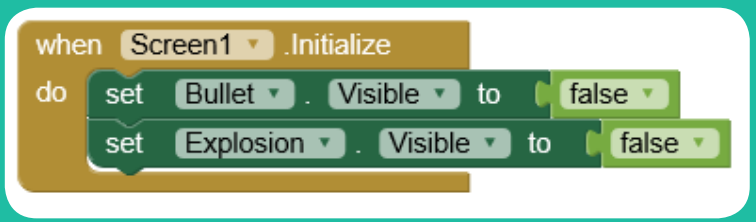


11 Now that all of our components are in place, switch to the Blocks Editor to add our code. Select the ***Ninja*** component. We want to move it from side to side, so drag a ***when Ninja.Dragged*** event into the viewer. Insert a ***set Ninja.X to*** block and add the ***currentX*** parameter to that. Since we're only changing the x, the sprite can only move side to side.

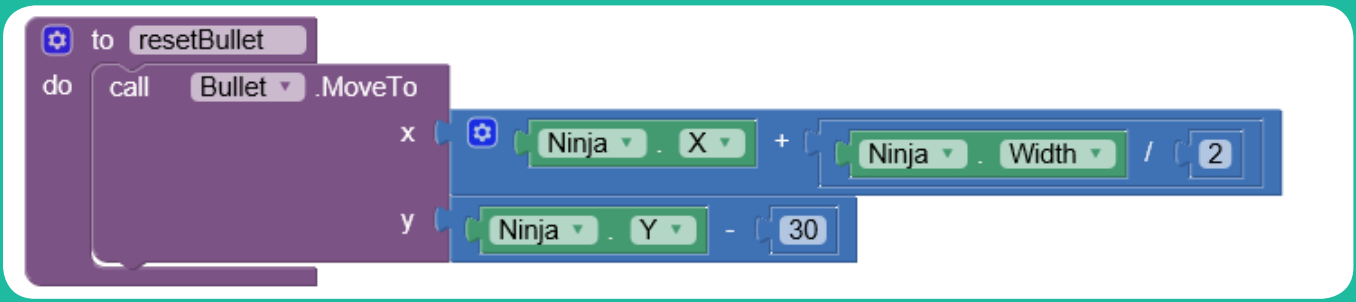


12 Connect your app to your emulator or device to test it. Can you move the ninja? Let's set up the other components.

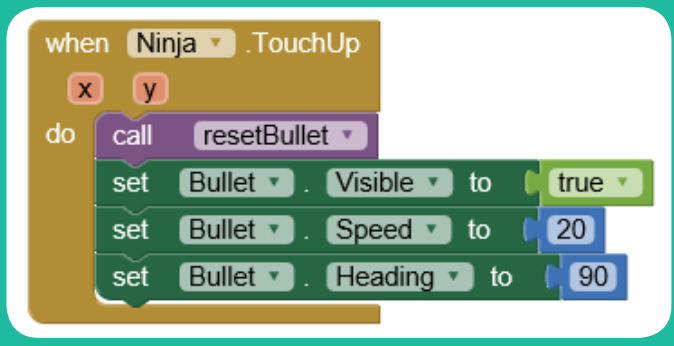
13 When the game starts, we don't want to see the ***Bullet*** or ***Explosion*** sprites. Click on the ***Screen1*** component and drag ***when Screen1.Initialized*** into the Viewer. Drag ***set Bullet.Visible*** and ***set Explosion.Visible*** blocks into the event and set them as ***false***.



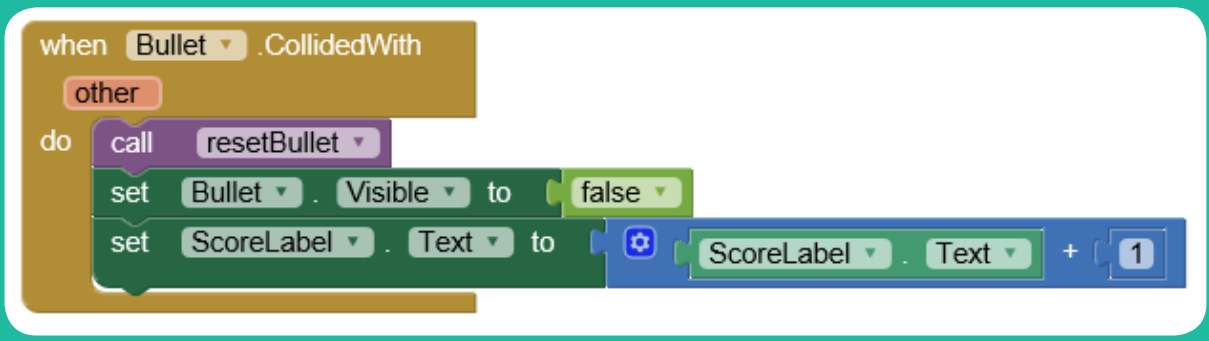
14 The Bullet is going to be moved around a lot during the game, so we need a Procedure to reset its position. Drag a ***to procedure do*** block into the viewer and name it "resetBullet." Inside the ***Procedure***, add a ***call Bullet.MoveTo*** block. We want the ***Bullet*** to always be at the middle of the ***Ninja***, so add a ***Math*** addition block to set the Bullet X at ***Ninja.X*** + half of ***Ninja.Width*** (place Ninja.Width inside a Math divide by 2 block). The bullet should be a bit above the Ninja, so set the Bullet y at ***Ninja.Y*** - 30.



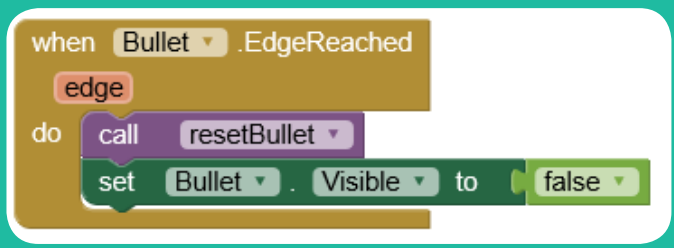
15 To fire the bullet, the player must release the ***Ninja***. Select the ***Ninja*** and drag a ***when Ninja.TouchUp do*** block into the viewer. When this event happens, we will move the bullet to the proper location by adding a ***call resetBullet*** block. Add a ***set Bullet.Visible to true*** block so we can see the Bullet and give it a ***Speed*** of 20 and a ***Heading*** of 90 (straight up).



16 We want to see if the ***Bullet*** hits the ***Alien***. Select the ***Bullet*** component and drag a ***when Bullet.CollidedWith*** block into the viewer. Currently, the only other sprite the bullet can touch is the ***Alien***, so if it hits anything, it will be the Alien. When this happens, we want to move and hide the ***Bullet*** by calling ***resetBullet*** and adding ***set Bullet.Visible to false***. Also, we will score a point. Add a ***set ScoreLabel.Text*** block and set it to ***ScoreLabel.Text*** + 1.

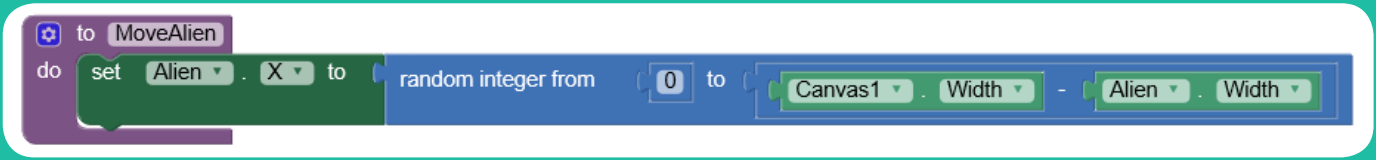


17 If the ***Bullet*** misses the ***Alien***, then it will keep going until it reaches the edge of the ***Canvas***. When this happens, we want to reset the bullet and hide it again. Select the ***Bullet*** and drag a when ***Bullet.EdgeReached*** block into the viewer. Duplicate the ***resetBullet*** and ***Bullet.Visible*** blocks from the ***Bullet.ColliedWith*** event and insert them into this event.

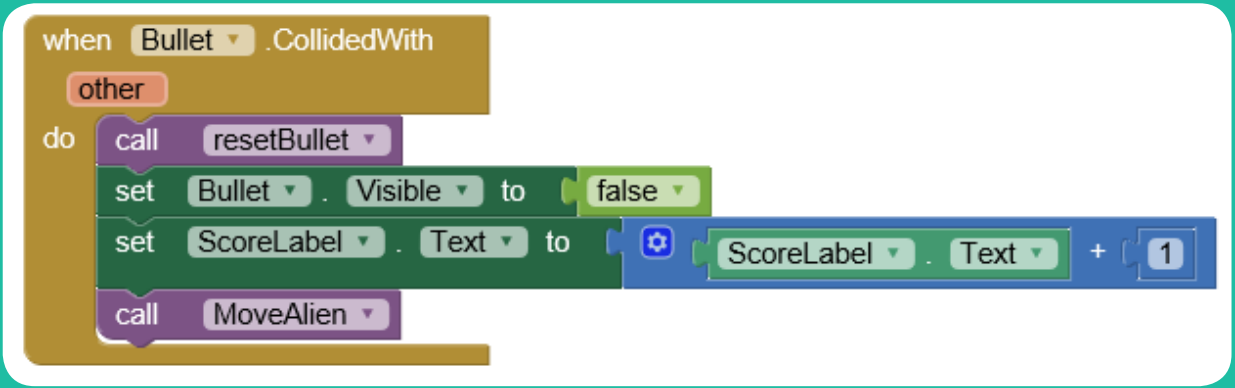


18 Connect your app to your emulator or device to test it. Now the Ninja can shoot at the alien. But not only is it pretty easy to hit the Alien, not a lot happens when it gets hit.

19 Let's move the ***Alien*** around. Add a ***Procedure*** to the Viewer and call it "MoveAlien." This procedure will move the Alien to a new horizontal position. Drag a ***set Alien.X to*** block into the ***Procedure*** and add a ***random integer from "0" to "100"*** block from ***Math*** into the slot. Discard the "100" number block and replace it with a ***Math*** subtraction block. Insert ***Canvas1.Width*** and ***Alien.Width*** blocks as shown. Now the Alien can be placed anywhere within the width of the Canvas without going past the edge.

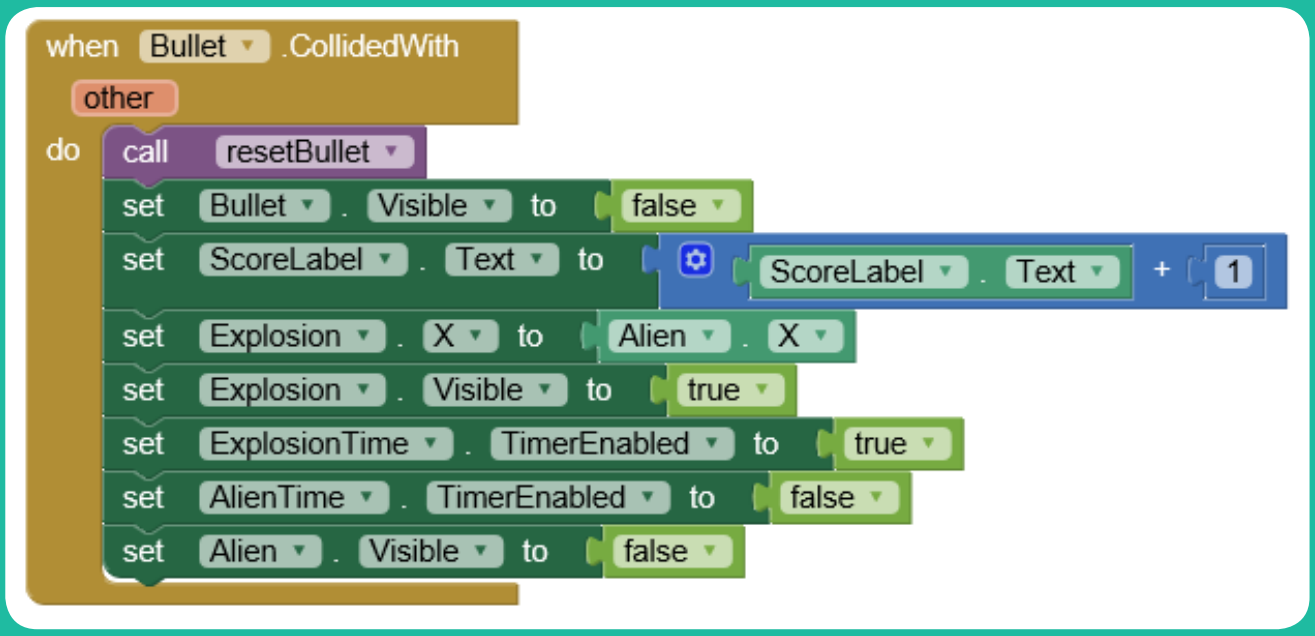


20 Add a ***call MoveAlien*** block to the end of the ***Bullet.CollidedWith*** event. Now the Alien is moved everytime it's hit.

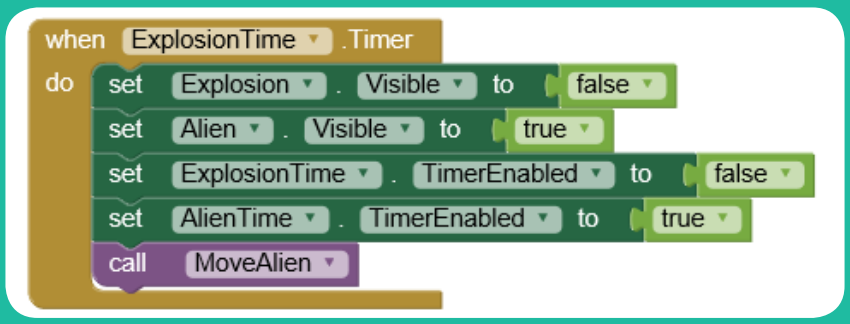


21 But the ***Alien*** shouldn't just sit around waiting to be hit. Click on the ***AlienTime*** component and add a ***call MoveAlien*** block here as well. Remember, the ***TimerInterval*** for this component is set at 3000 milliseconds, so the ***Alien*** will move every 3 seconds.

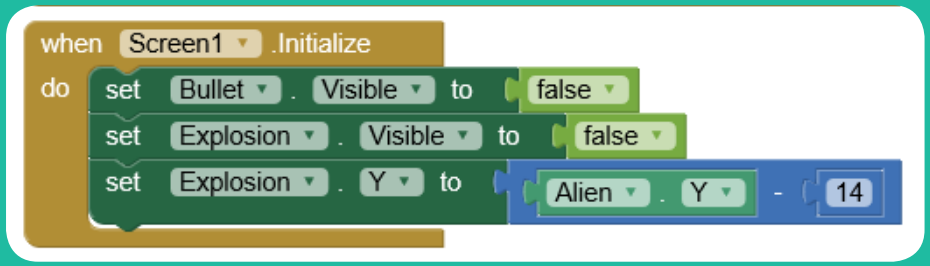
22 We still need something to happen when the ***Alien*** is hit. Let's hide the ***Alien*** and replace it with the ***Explosion*** sprite. In the ***Bullet.CollidedWith*** event, add a block to ***set Explosion.X to Alien.X***. Then ***set Explosion.Visible to true***. We need the Explosion to be seen for just a little while, so also add a ***set ExplosionTime.TimerEnabled to true*** block. Also, set ***AlienTime.TimerEnabled to false*** and ***set Alien.Visible to false***. Let's not move the ***Alien*** yet, so remove the ***call MoveAlien*** block.



23 The ***ExplosionTime TimerInterval*** property is set to 500 milliseconds (half a second). When that time is up, then the Timer event is triggered. Drag a ***when ExplosionTime.Timer do*** block into the viewer. Now we will do the opposite of what we did when the ***Bullet*** first hit the ***Alien***. set ***Explosion.Visible*** and ***ExplosionTime.TimerEnabled*** to ***false*** and set ***Alien.Visible*** and ***AlienTime.TimerEnabled*** to ***true***. Now we will move the Alien with the ***call MoveAlien*** block.

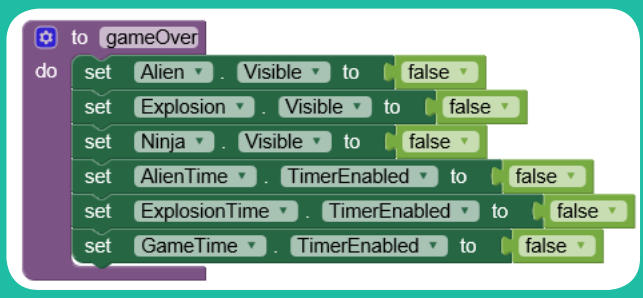


24 Just in case the ***Explosion*** sprite isn't showing up in the right place, add a ***set Explosion.Y to Alien.Y - 14*** block to the ***Screen1.Initialize*** event.

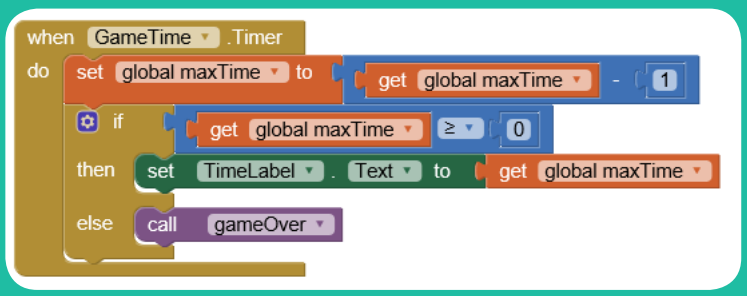


25 The game needs to end some time. Let's have the game end after 30 seconds (you can change the length of the game later). Add an ***initialize global*** variable block and give it the name of "maxTime." Set it to the number "30."

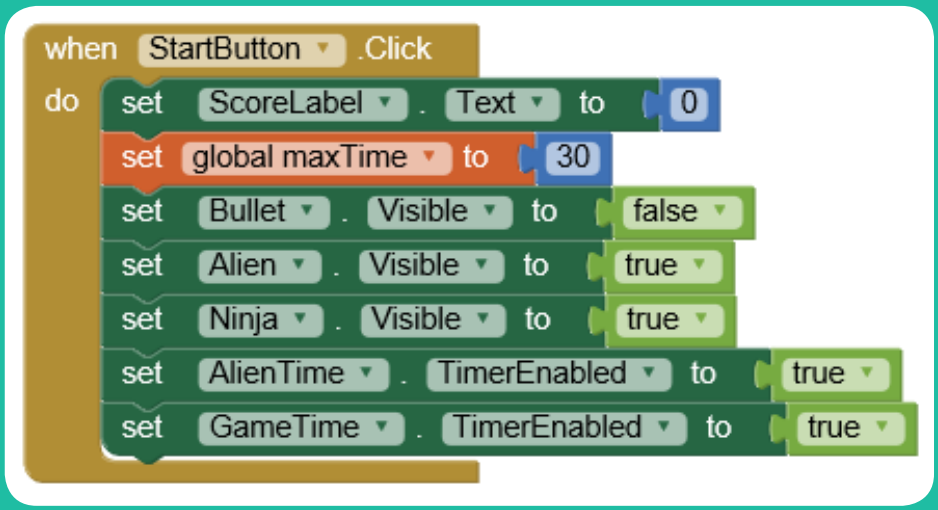
26 What do we want to do when the game is over? We should hide all of the sprites and stop all of the timers. Add a new ***Procedure do*** block and name it "gameOver." Add blocks to set the ***Alien, Explosion*** and ***Ninja*** sprite ***Visible*** property to ***false***. Also, set the ***AlienTime, ExplosionTime*** and ***GameTime TimerEnabled*** properties to ***false***.



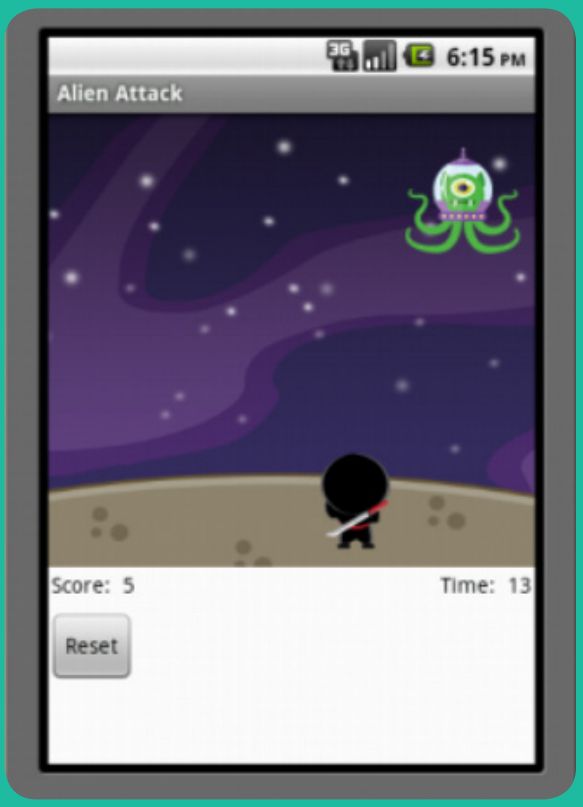
27 The ***GameTime TimerInterval*** is set to 1000 milliseconds (one second), which is perfect for our needs. Add a ***when GameTime.Timer do*** block to the Viewer. Every second, we subtract one from maxTime. Add a ***set global maxTime to*** block and set it to ***global maxTIme - 1***. We need to check if there is any time left, so add an ***if then*** block from ***Control***. Click on the blue modify icon to add an ***else*** to the block. For the condition, check if ***global maxTime*** is greater than or equal to (≥) "0." If it is, then we ***set TimeLabel.Text to global maxTime***. Otherwise, we call the ***gameOver Procedure***.



28 After the game is over (or if the user decides to start over), there needs to be a way to start things up again. Select the ***StartButton*** component and drag a ***when StartButton.Clicked*** event into the Viewer. Set ***ScoreLabel.Text*** to "0" and ***set global maxTime*** to "30." Set ***Bullet.Visible to false*** and ***Alien*** and ***Ninja Visible*** properties to true. Finally, set the ***AlienTime*** and ***GameTime TimerEnabled*** properties to ***true***.



29 Connect your app to your emulator or device to test it. What's your high score?



So Far, So Good

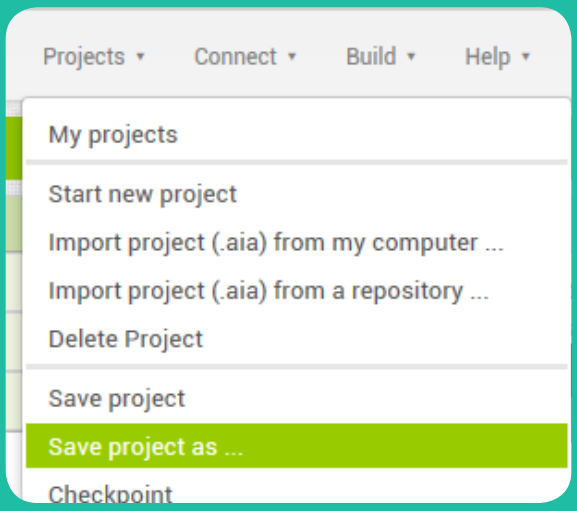
Now that the game is working, we can add some things to make it look more fun. In the world of Computer Games, such effects that do not have a direct effect on the actual game mechanics are called "bells and whistles." These get added after we know that the game works (and is hopefully fun already) to make things look even better.

Frame Animation

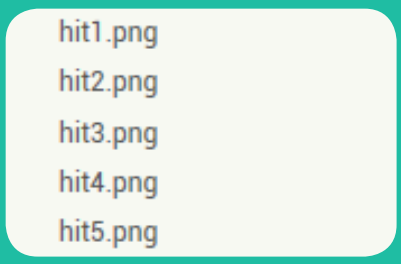
Typical animation in App Inventor is usually moving something from one place to another. But you probably noticed that we replaced the alien with the explosion sprite to show the alien getting hit. We can use that to make the explosion really stand out, by substituting the images being displayed one at a time.

Activity 05-02: Alien Attack 2

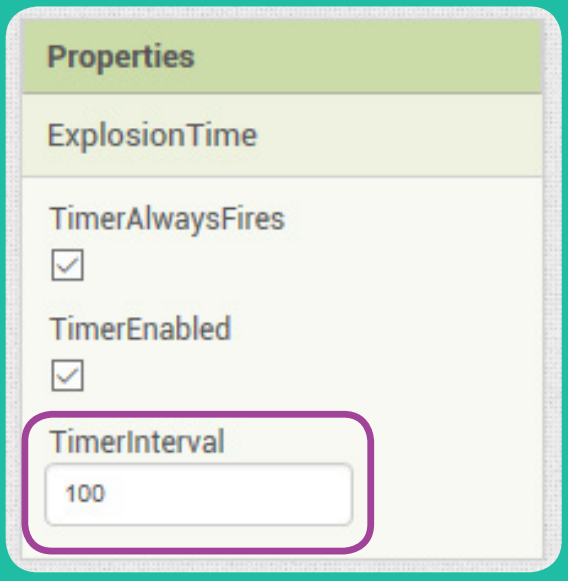
1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



2 Switch to Designer. In the ***Media*** section, make sure that you have uploaded all 5 of the "hit" png files.

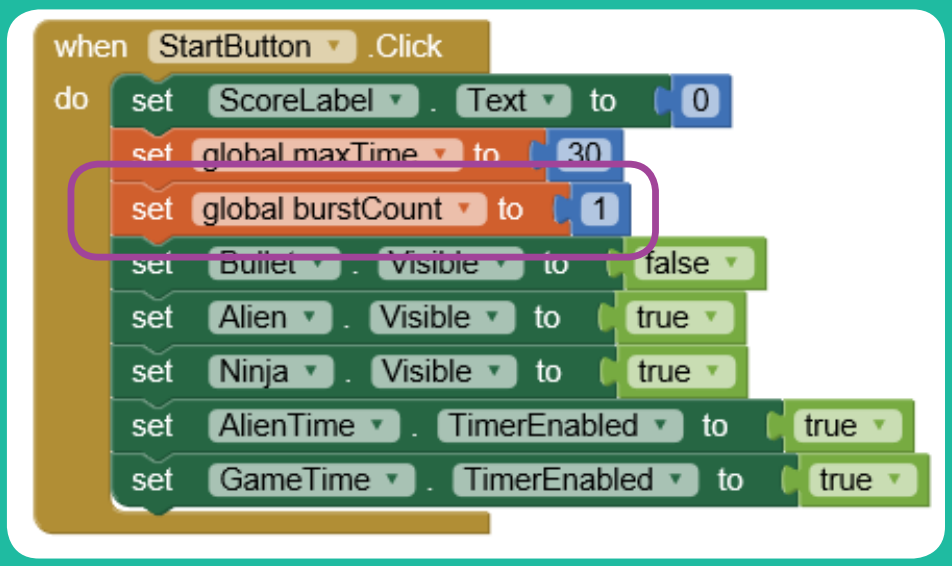


3 Select the ***ExplosionTime*** component and change the ***TimerInterval*** property to "100." We will show each image for 1 tenth of a second, so that when the animation is complete, half a second will have passed, just like before.

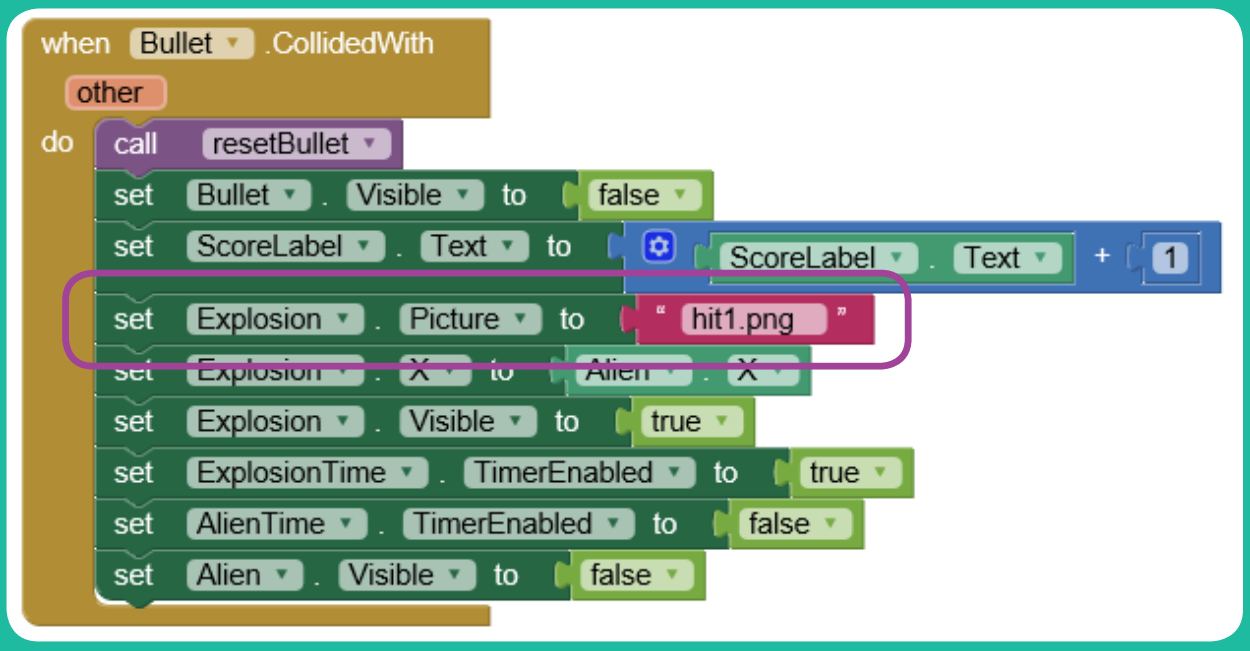


4 Switch to the Blocks Editor. We will need some way to keep track of which picture we are using in the animation. Add an ***initialize global*** block from ***Variables*** and name it "burstCount." Set it to the number "1."

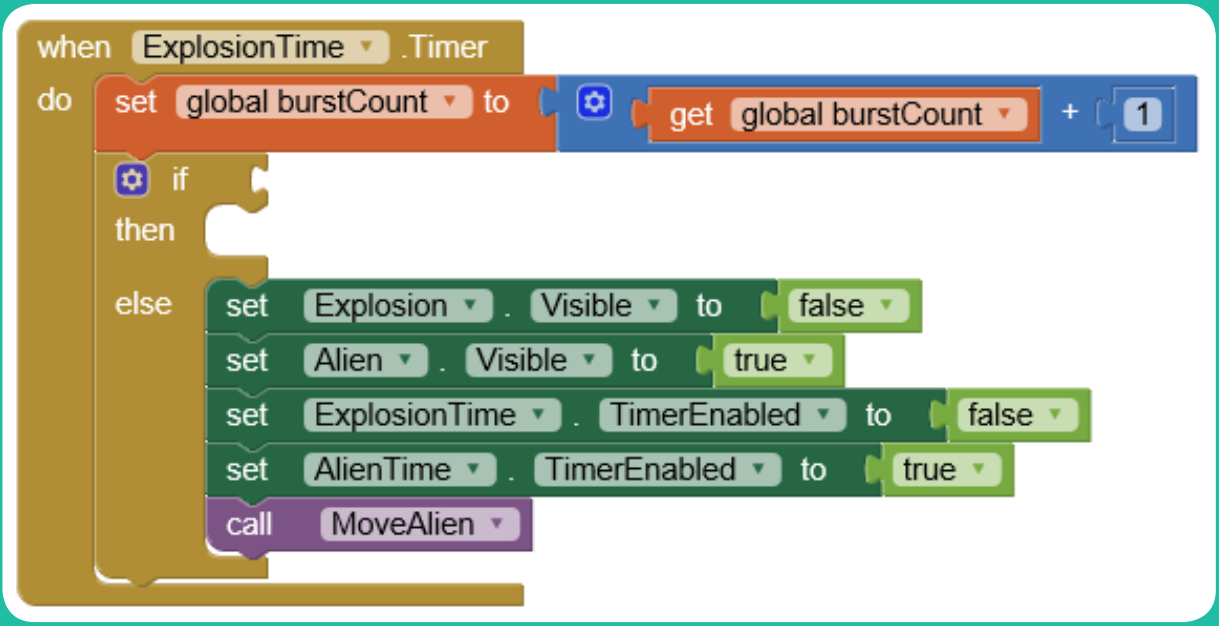
5 Select the ***StartButton.Click*** event and add a ***set global burstCount*** to "1" block.



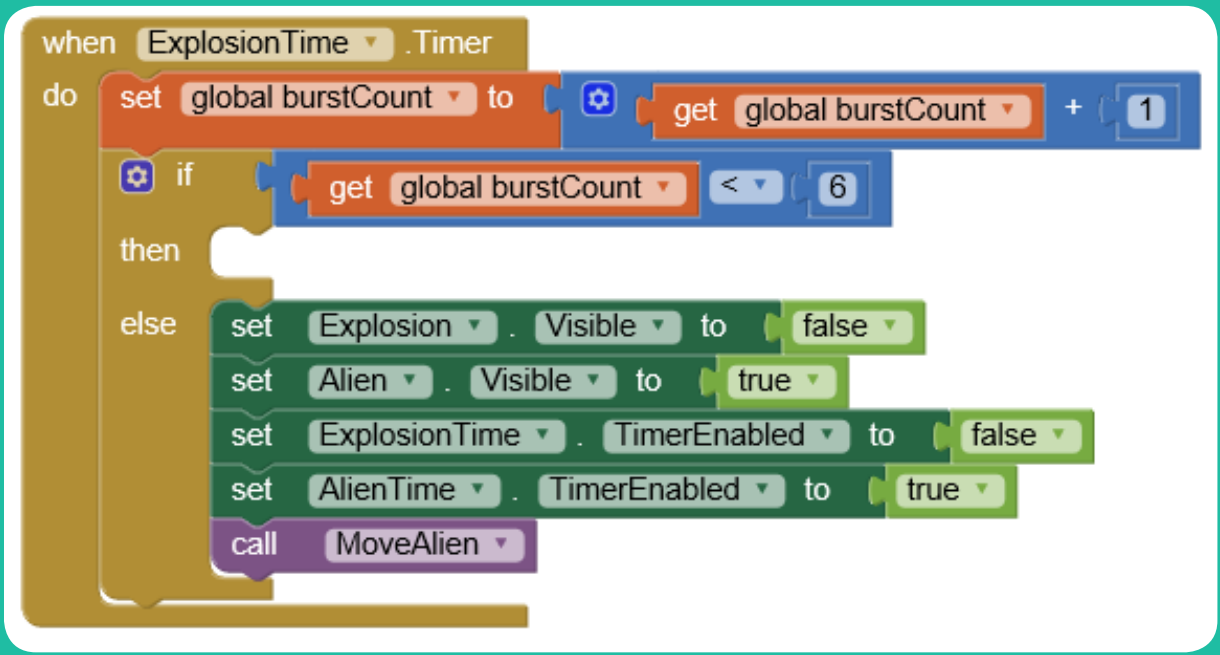
6 Select the ***when Bullet.CollidedWith*** event and add ***set Explosion.Picture to hit1.png***. So that no matter what happens, the first part of the explosion animation is the first thing we see.



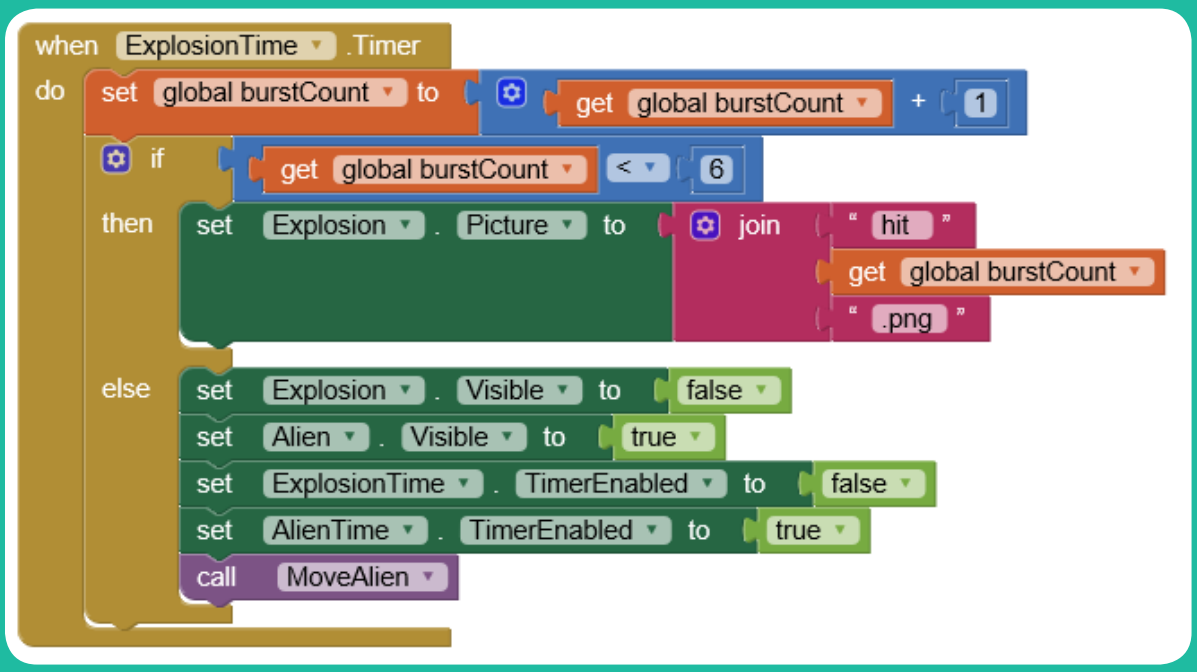
7 When the ***ExplosionTime.Timer*** is triggered is when we will make our animation happen. Add a ***set global burstCount to global burstCount + 1*** block at the top of the event. Insert an ***if then*** block after that and click on the blue modification icon to add an ***else*** to the condition. Take the five blocks at the end of the event and move them into the ***else*** slot.



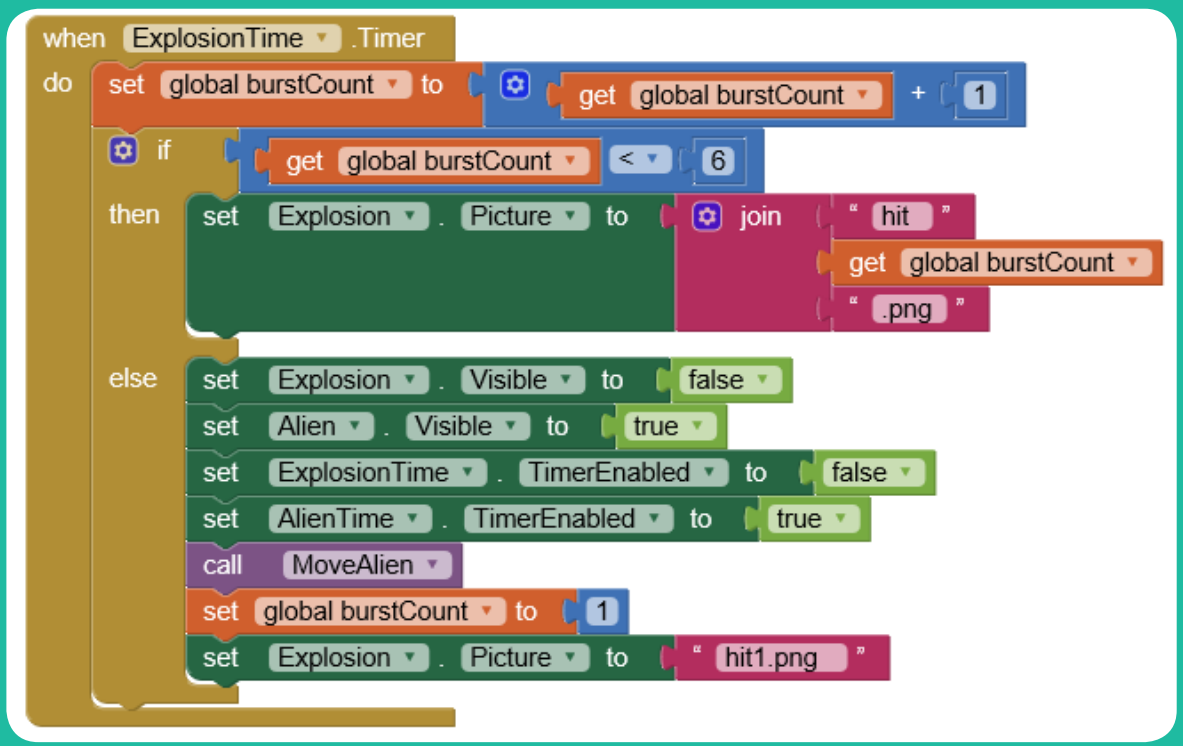
8 For our condition, we want to keep track of the ***burstCount*** variable. As long as it is less than 6, we will simply change the ***Explosion.Picture*** property. Add an equals block from ***Math*** and change it to "<" and insert ***get global burstCount*** and "6" as shown.



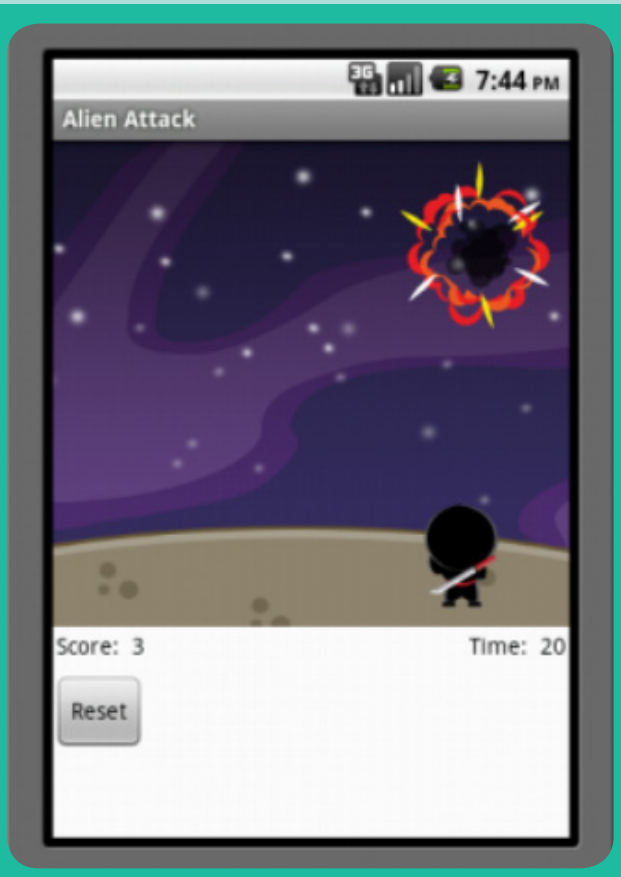
9 Add ***set Explosion.Picture to*** block into the then slot. Our images are the word "hit" plus a number, plus ".png." Add a ***join*** block from ***Text*** and click on the blue modification icon to add a third slot to the block. Insert text blocks for "hit" and ".png" as shown and insert ***get global burstCount*** for the number. Every tenth of a second, we will replace the picture with a new one until all five have been seen.



10 After the animation is finished, we hide the explosion and move the ***Alien*** and disable or enable the timers as before. There is one thing left to do. We need to reset the ***burstCount*** variable and ***Explosion.Picture*** as shown.



11 Connect your app to your emulator or device to test it. Looks better, doesn't it?



A Little Too Quiet

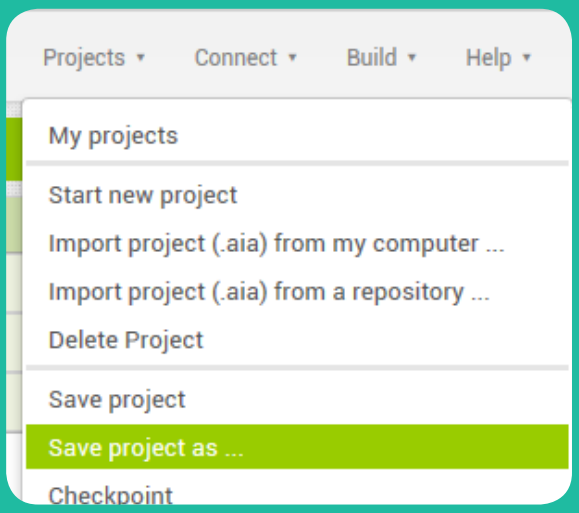
Now we have a good looking explosion effect, we should do something about how the game sounds

Sound Effects

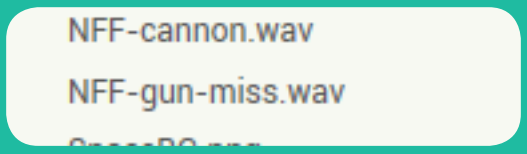
In this next activity, we will add a sound for when the Ninja is shooting and a different sound for when the Alien explodes.

Activity 05-03: Alien Attack 3

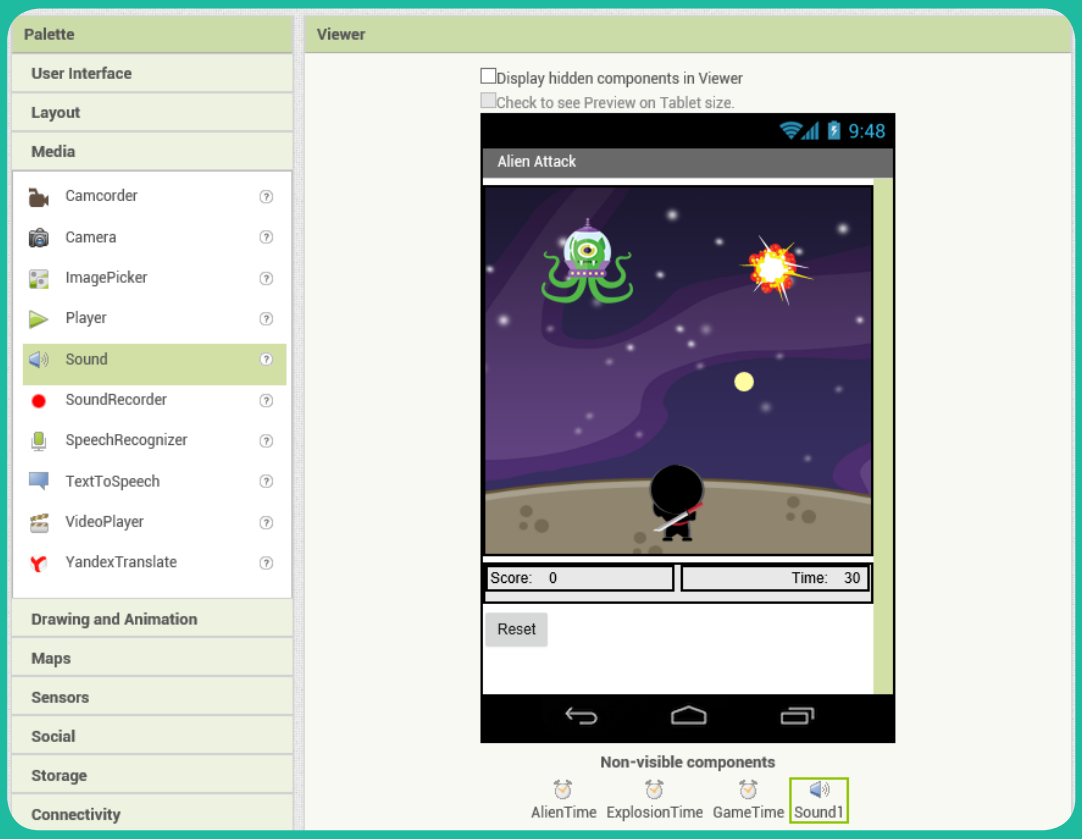
1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



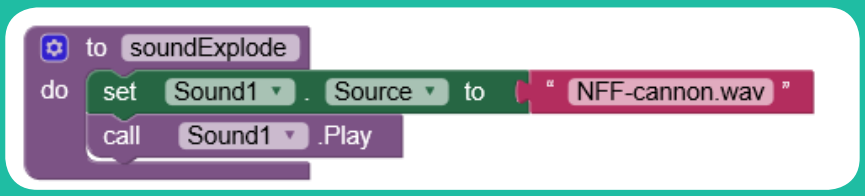
2 Switch to Designer. In the ***Media*** section, make sure that you have uploaded the following files from Assets: *NFF-cannon.wav* and *NFF-gun-miss.wav*.



3 From the ***Media Palette***, drag a ***Sound*** component into the ***Viewer***.

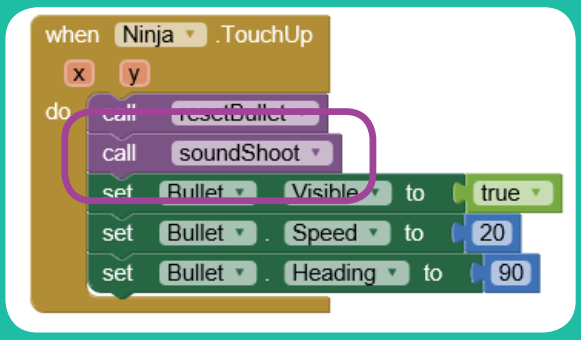


4 Switch to the Blocks Editor. We will need to create a procedure for our sound effects. From ***Procedures***, drag a ***to procedure do*** block into the Viewer. name it, "soundExplode." Drag a ***set Sound1.Source to*** block into the ***Procedure*** and insert a ***Text*** block with the name of "NFF-cannon.wav." Add a ***call Sound1.Play*** block.

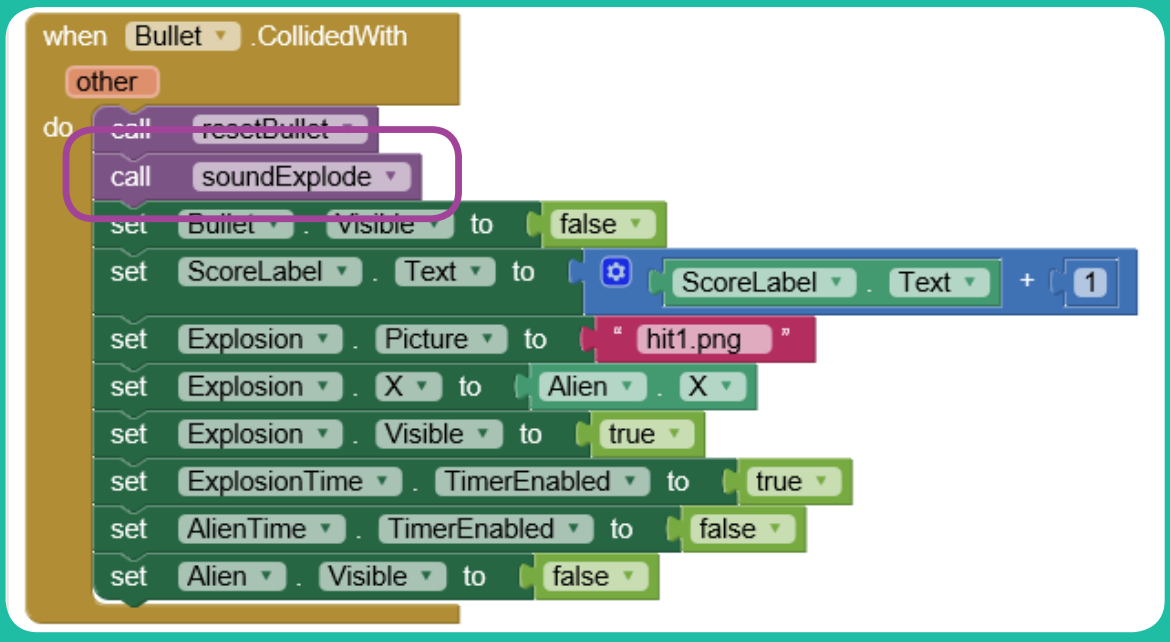


5 Duplicate the ***soundExplode*** block and give the duplicate the name of "soundShoot." Change the text for the audio file to "NFF-gun-miss.wav."

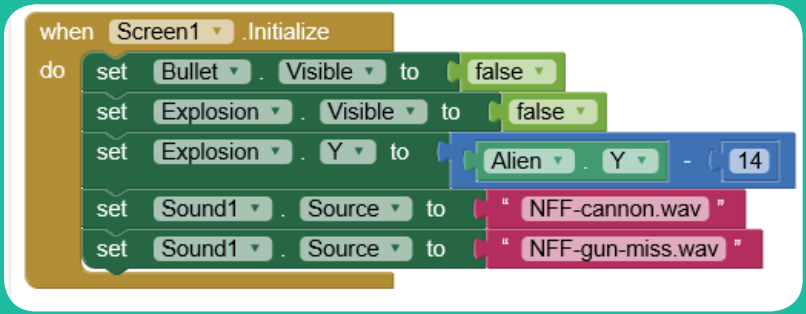
6 In the ***when Ninja.TouchUp*** event, insert a ***call soundShoot*** block.



7 In the ***when Bullet.CollidedWith*** event, insert a ***call soundExplode*** block.



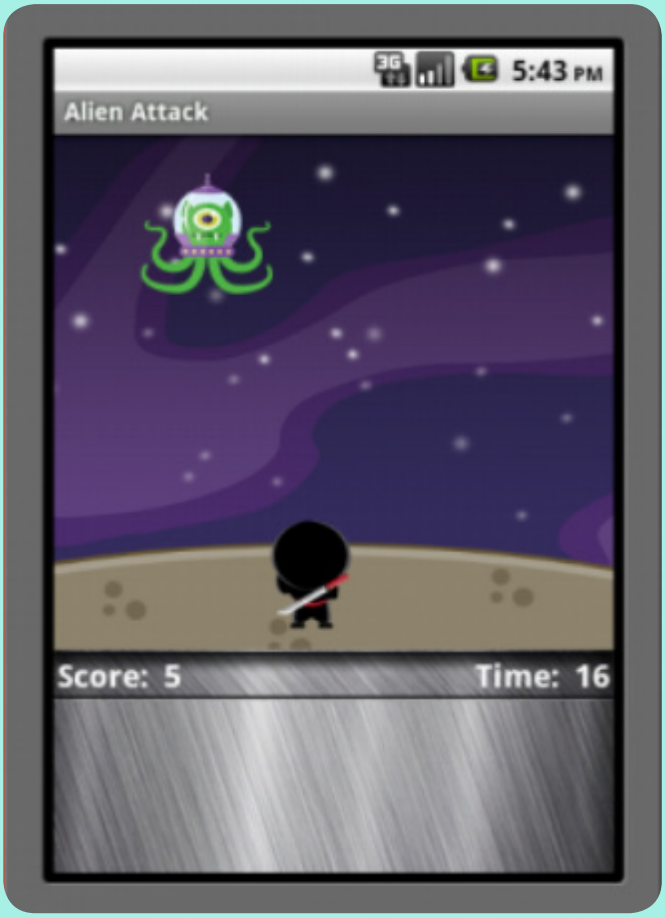
8 To pre-load the sound files when the app starts, insert ***set Sound1.Source to*** blocks for both sound files in the ***when Screen1.Initialize*** event.



9 Connect your app to your emulator or device to test it. Make sure the sound is turned up!

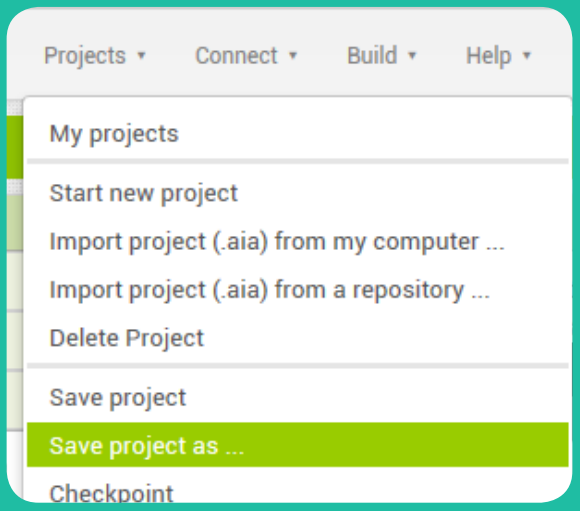
Finishing Touches

The game is looking pretty good, but it needs just a little more polish. We shouldn't have the game start until the user is ready to play. Additionally, what do we do when the game ends?

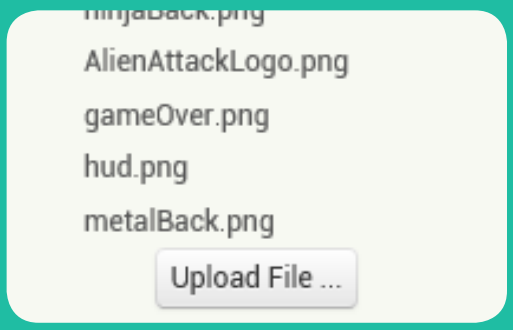


Activity 05-04: Alien Attack 4

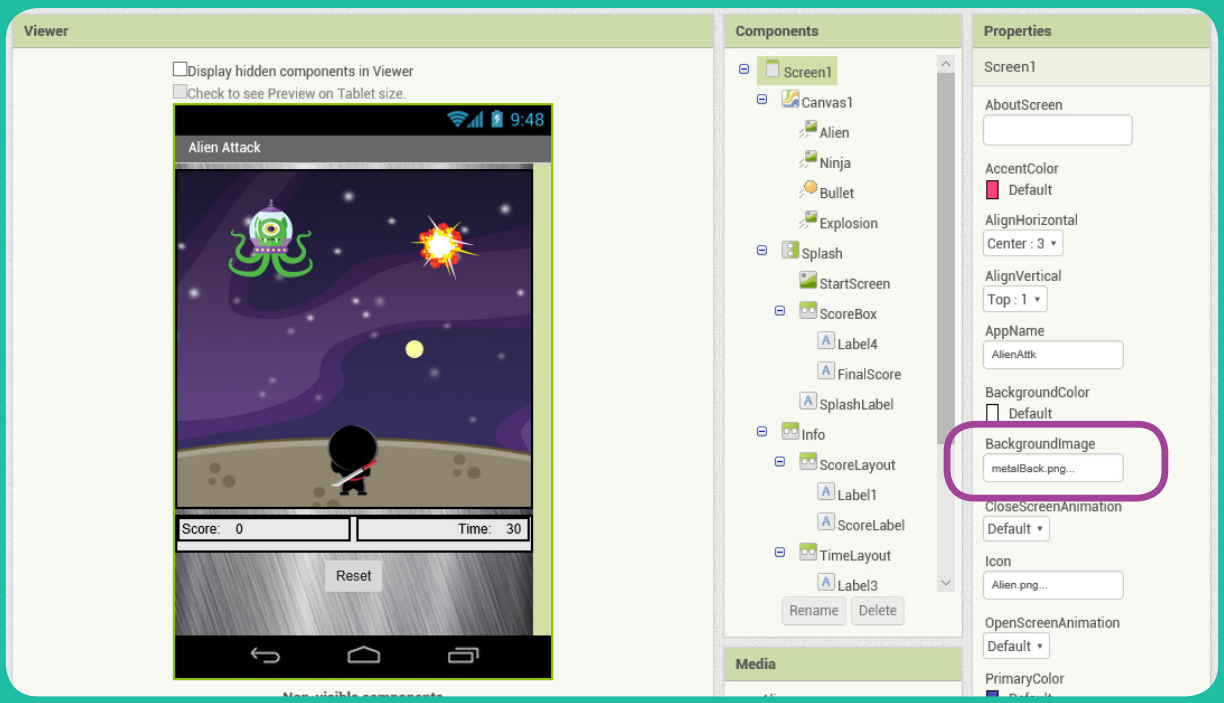
1 We are still working on the app we started with. Use ***Save project as*** in the ***Projects*** tab to make a copy of your progress so far.



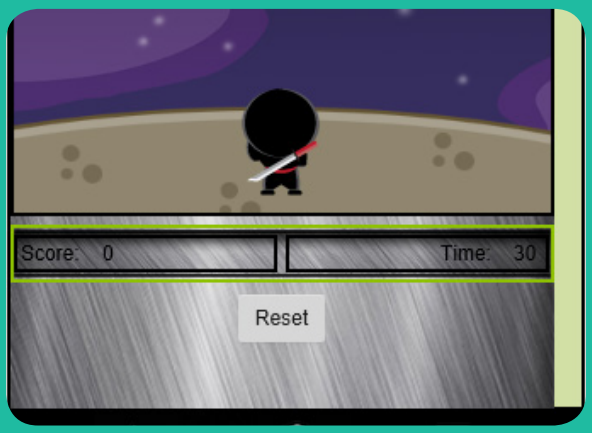
2 Switch to Designer. In the ***Media*** section, make sure that you have uploaded the following files from Assets: *AlienAttackLogo.png, gameOver.png, hud.png* and *metalBack.png*.

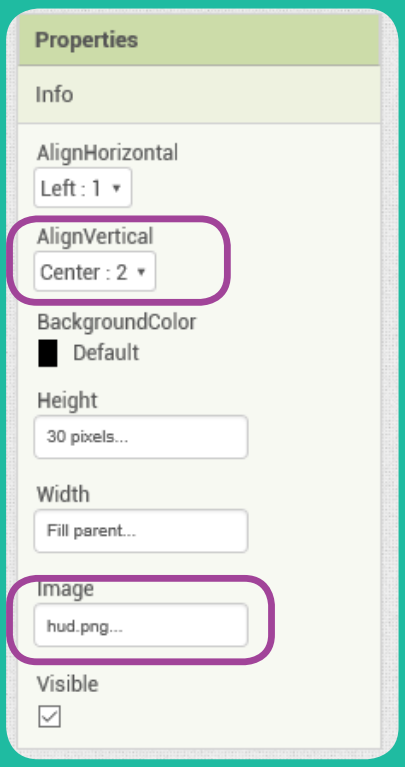


3 By now, you're probably tired of seeing the plain, white background. Select the ***Screen1*** component and change the ***BackgroundImage*** property to "metalBack.png."

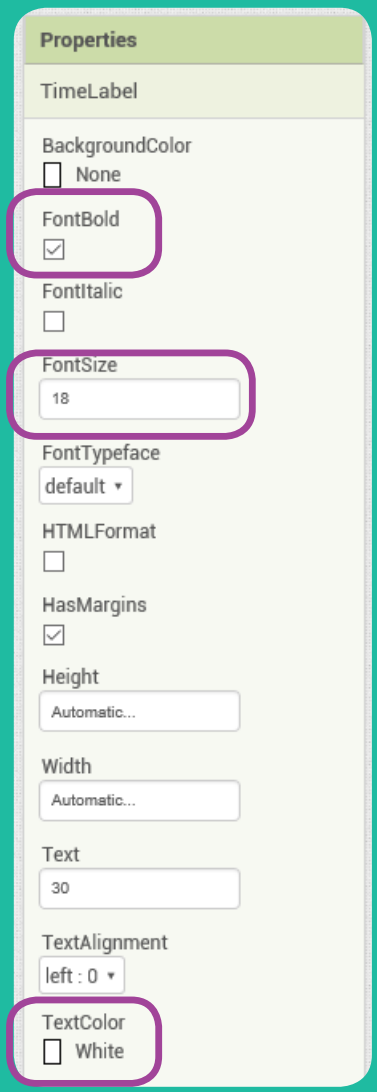


4 Now the Info component looks a little odd. Select it and change its ***Image*** property to "hud.png." Also, change the ***AlignVertical*** property to ***Center***.

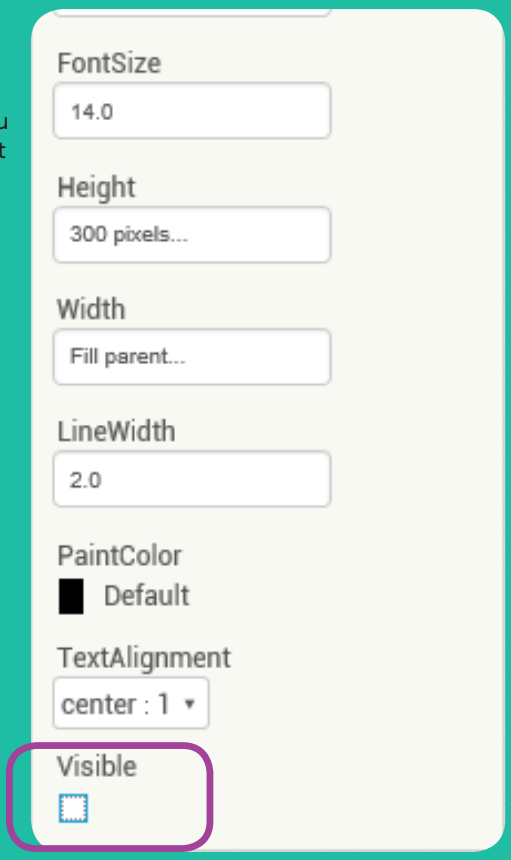




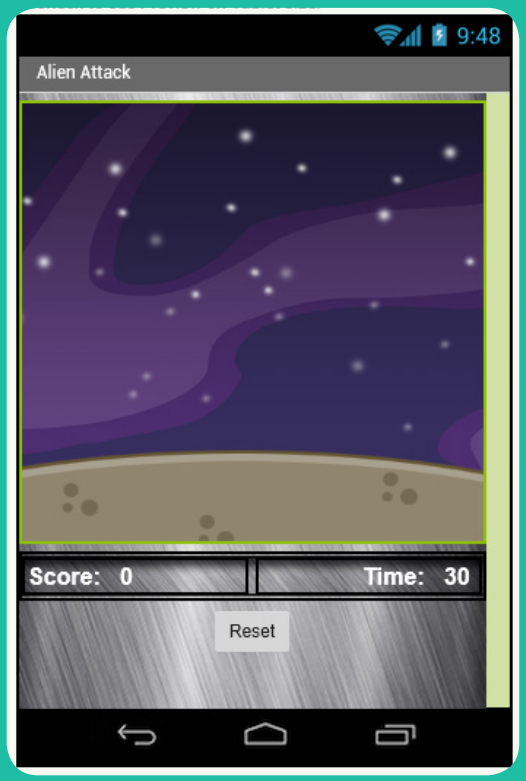
5 Now the two layouts are a bit hard to see. Select ***ScoreLayout*** and ***TimeLayout*** and change the ***BackgroundColor*** to "None." Then select each of the labels and make sure the ***FontBold*** property is checked, the ***FontSize*** is "18" and the ***TextColor*** is ***White***.

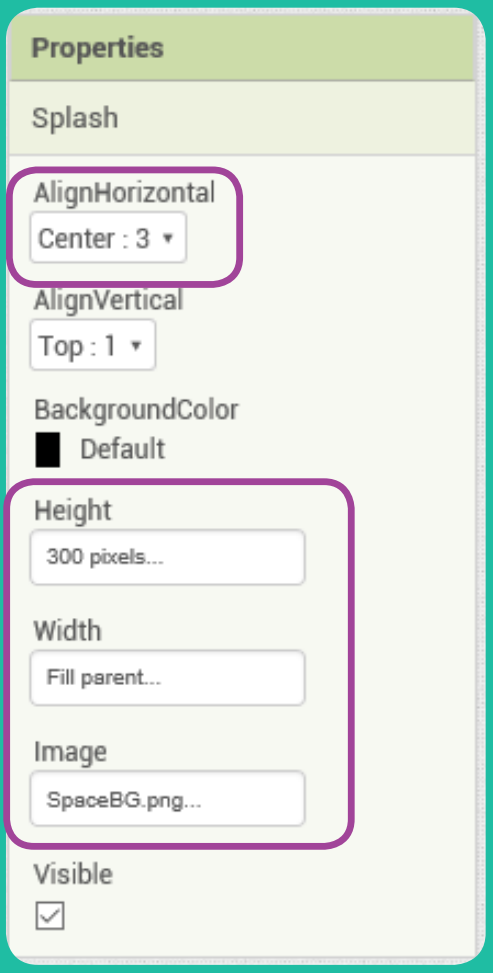


6 Next, we want a screen to show until the user starts the game. This is known as a "splash screen." Before we can set that up, we need to make a little room in our ***Viewer***. Select the ***Canvas1*** component and uncheck the ***Visible*** property. Now you can't see the ***Canvas*** and everything has moved up as if it wasn't there. Don't worry, it's not gone, we have just hidden it.

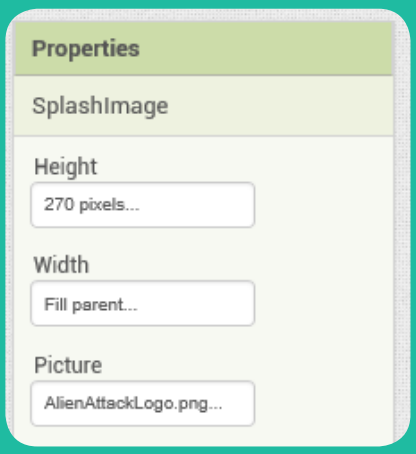


7 Now for our splash screen. Select the ***Layout Palette*** and drag a ***VerticalArrangement*** component to the top of the ***Viewer***. Rename it as "Splash." and change the ***Width*** to ***Fill Parent*** and the ***Height*** to 300 pixels. Set the ***AlignHorizontal*** property to ***Center*** and Select "SpaceBG.png" for the ***Image*** property.



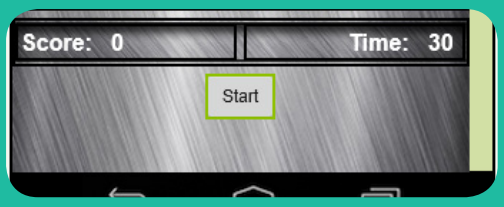


8 A good splash screen has an attractive image to get users excited about your game. Select the ***User Interface Palette*** and add an ***Image*** component to the ***Splash*** component. Rename it as "SplashImage" and set the ***Height*** property to 270 pixels, the ***Width*** to ***Fill parent*** and the ***Picture*** to "AlienAttackLogo.png"



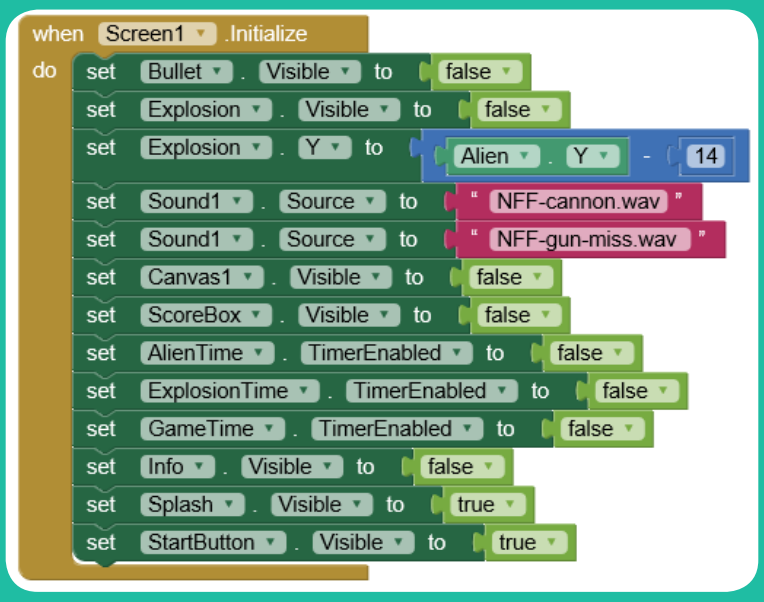
9 When this screen appears, we want to have a message to tell the user to press the start button to begin. Drag a ***Label*** component from the ***User Interface Palette*** and place it below the ***SplashImage***. Change the name to "SplashLabel" and the ***Text*** property to "Press START to continue." Change the ***TextColor*** to ***White***.

10 Select the ***StartButton*** component and change the ***Text*** property to "Start."

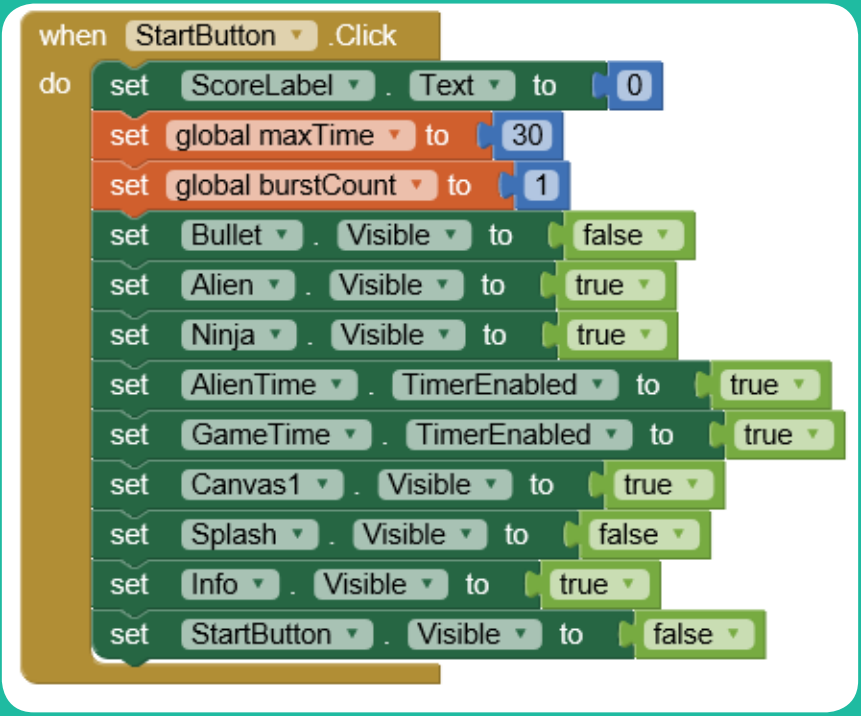


11 There's one last thing left to do. We need a component to show the final score after the game has ended. Select the ***Layout Palette*** and drag a ***HorizontalArrangement*** inbetween ***SplashImage*** and ***SplashLabel***. Rename it as "ScoreBox." (Hide the ***SplashImage*** component if you're having trouble seeing things - just make sure to unhide it after you're done. Add two labels from the ***User Interface Palette*** to ***ScoreBox***. For each label, check the ***FontBold*** property, change the ***FontSize*** to "24" and change the ***TextColor*** to ***White***. Change the ***Text*** of the first label to "Your Score: " and change the ***Text*** of the second label to "0." ***Rename*** the second label as "FinalScore."

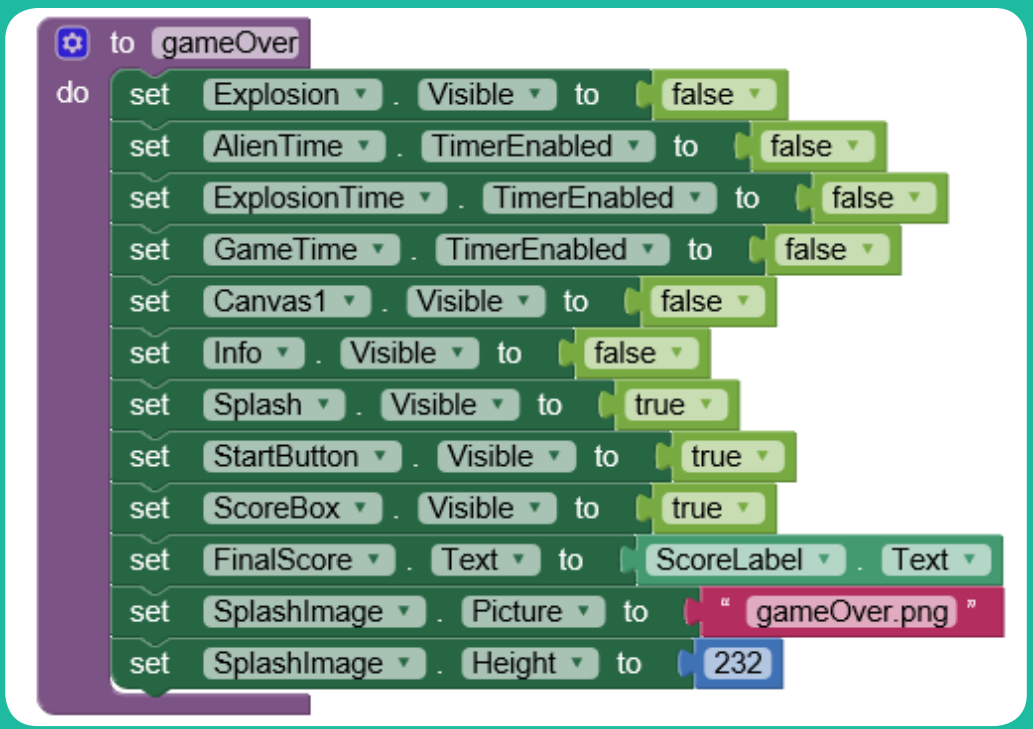
12 Now we just have to make sure the right parts are showing at the right times. Switch to the Blocks Editor and select the ***when Screen1.Inititalize*** event. When the app is opened, we don't want to see the ***Canvas*** or ***Info*** components and we want all of the timers disabled. Add blocks to the event to set ***Canvas1, Info*** and ***ScoreBox*** visiblity to ***false***. Set ***AlienTime, ExplosionTime*** and ***GameTime TimerEnabled*** to ***false*** as well. Set ***Splash*** and ***StartButton Visible*** to ***true***.



13 Now, the game doesn't start until the start button is pressed. When that happens, we'll hide the ***Splash*** screen and ***Start*** button and show the ***Canvas*** and ***Info*** components. Add blocks to the ***when StartButton.Click*** event to set ***Canvas1*** and ***Info Visible*** to ***true*** and ***Splash*** and ***StartButton Visible*** to ***false***.



14 When the game ends, we will hide the Canvas and Info components and show the Splash and StartButton components again. Add blocks to the ***gameOver Procedure*** to set ***Canvas1*** and ***Info Visible*** to ***false***. Set ***Splash, StartButton*** and ***ScoreBox Visible*** to ***true***. Add a block to ***set FinalScore.Text to ScoreLabel.Text*** and ***set SplashImage.Picture to "gameOver.png."*** To make room for the ***ScoreBox***, add a ***set SplashImage.Height to 232*** block.



15 Connect your app to your emulator or device to test it. What do you think of your bells and whistles?

Improvements

As it stands now, the game forgets any previous scores when a new game is started. How can you set up a system to save the high score? (Hint: the TinyDB component in STORAGE can save data for the app to recover the next time the game is played.)

Can the alien movement be more interesting? Maybe have it move down from the top and side to side? What if you wanted to make the game easier or harder? What properties would you change?

What's Next?

There are quite a few features in App Inventor that we haven't touched on. The emulator cannot simulate the phone and text capabilities of an average Android device. It also doesn't have GPS. But there are components in App Inventor that take advantage of those systems in mobile devices. If you want to learn more, you can find a few activities that you can experiment with here.

http://www.appinventor.org/