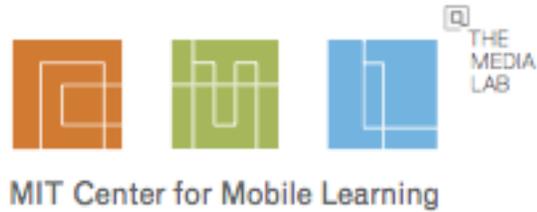


App Inventor Beginner Tutorials



1 Four Simple Tutorials for Getting Started with App Inventor

1.1	TalkToMe: Your first App Inventor app	4
1.2	TalkToMe Part 2: Shaking and User Input	23
1.3	BallBounce: A simple game app	33
1.4	DigitalDoodle: Drawing App	47

Four Simple Tutorials for Getting Started with App Inventor



TalkToMe: Your first App Inventor app

This step-by-step picture tutorial will guide you through making a talking app.

To get started, go to App Inventor on the web.

Go directly to ai2.appinventor.mit.edu, or click the orange "Create" button from the App Inventor website.

MIT App Inventor Home Blog Support **Create**

Follow Us: [f](#) [t](#) [y](#) [m](#) Google™ Custom Search

Your ideas.
Your designs.
Your apps.
Invent Now

Get Started
Follow these simple steps to build your first app.
Get Started

Create
Design and program your own apps using MIT App Inventor.
Create

Tutorials
Step-by-step guides show you how to build all kinds of apps.
Tutorials



MIT App Inventor
appinventor.mit.edu

Log in to App Inventor with a gmail (or google) user name and password.

Use an existing gmail account or school-based google account to log in to ai2.appinventor.mit.edu
To set up a brand new gmail account, go to accounts.google.com/SignUp



One account. All of Google.

Sign in with your Google Account

appinventorskilz@gmail.com

•••••••

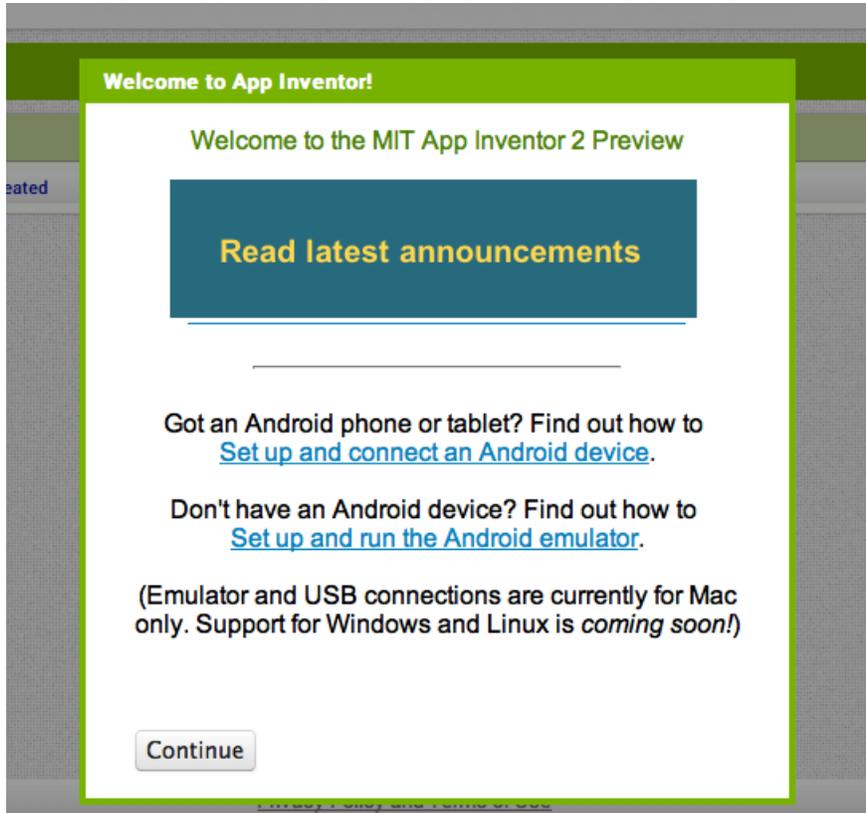
Sign in

Stay signed in [Need help?](#)

[Create an account](#)



Click "Continue" to dismiss the splash screen.





Start a new project.

MIT App Inventor 2
ai2.appinventor.mit.edu

MIT App Inventor 2
Beta

Project ▾ Connect ▾ Build ▾ Help ▾ My Projects Guide Report an Issue appinventorskilz@gmail.com ▾

New Project ... Delete Project

Projects

Name	Date Created	Date Modified ▾
------	--------------	-----------------

Welcome to App Inventor!

You don't have any projects yet. To learn how to use App Inventor, click the "Guide" link at the upper right of the window; or to start your first project, click the "New" button at the upper left of the window.

Happy Inventing!

[Privacy Policy and Terms of Use](#)

Name the project "TalkToMe" (no spaces!)

Type in the project name (underscores are allowed, spaces are not) and click OK.

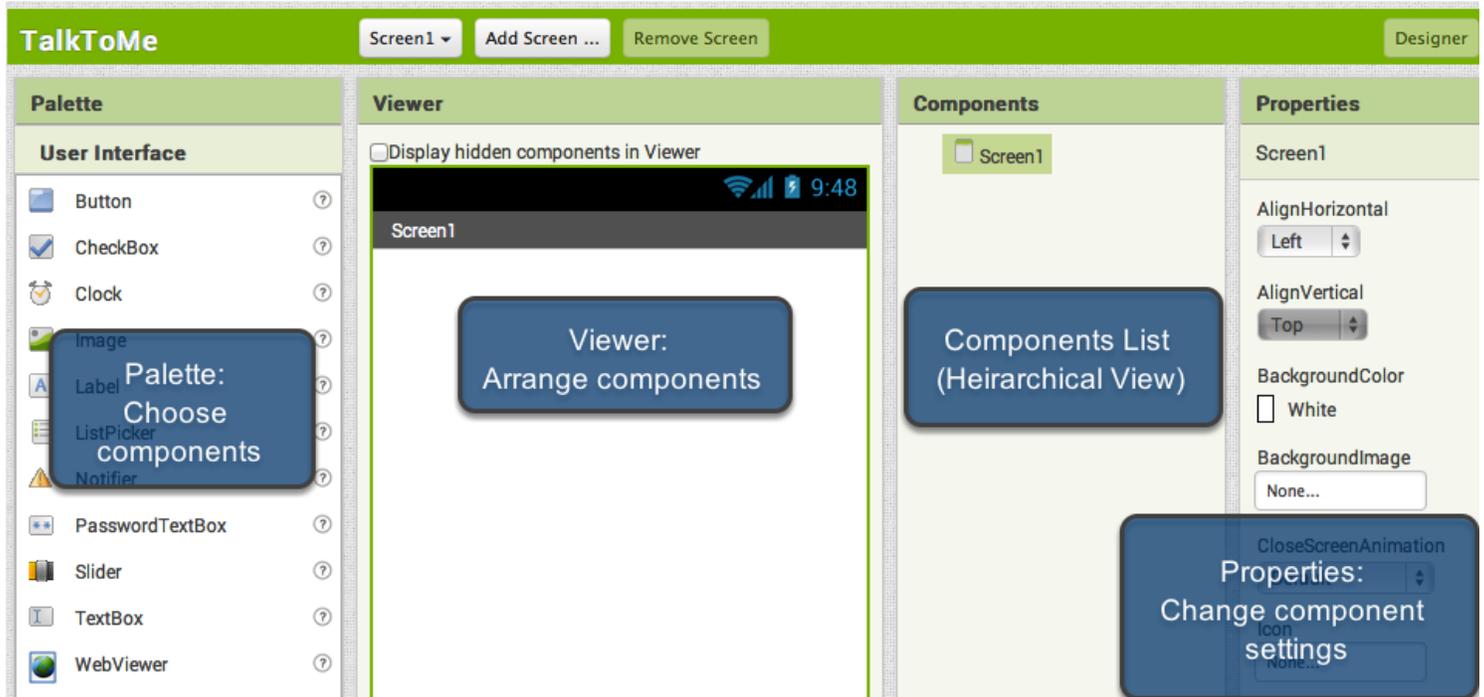
Create new App Inventor project

Project name:

Cancel OK

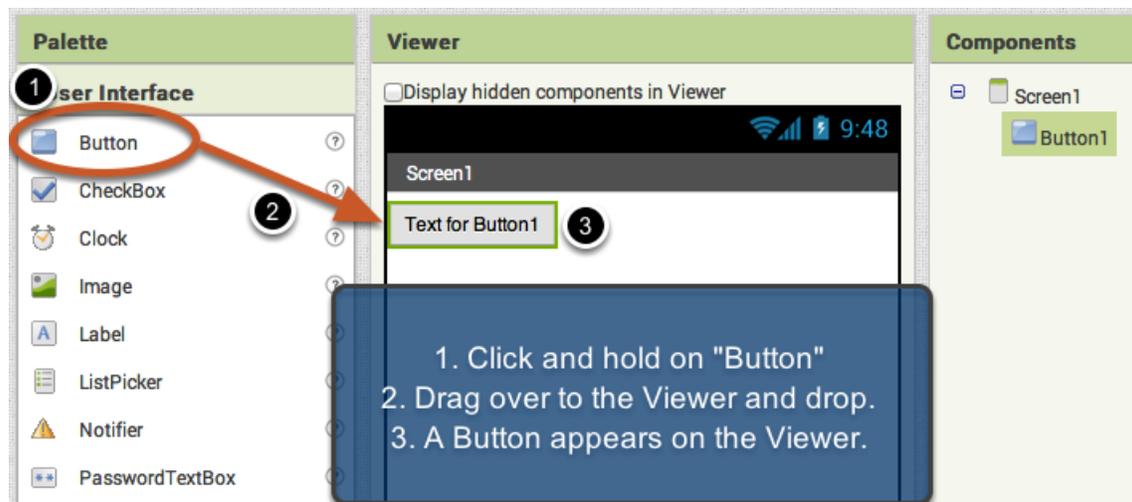
You are now in the Designer, where you lay out the "user interface" of your app.

The Design Window, or simply "Designer" is where you lay out the look and feel of your app, and specify what functionalities it should have. You choose things for the user interface things like Buttons, Images, and Text boxes, and functionalities like Text-to-Speech, Sensors, and GPS.



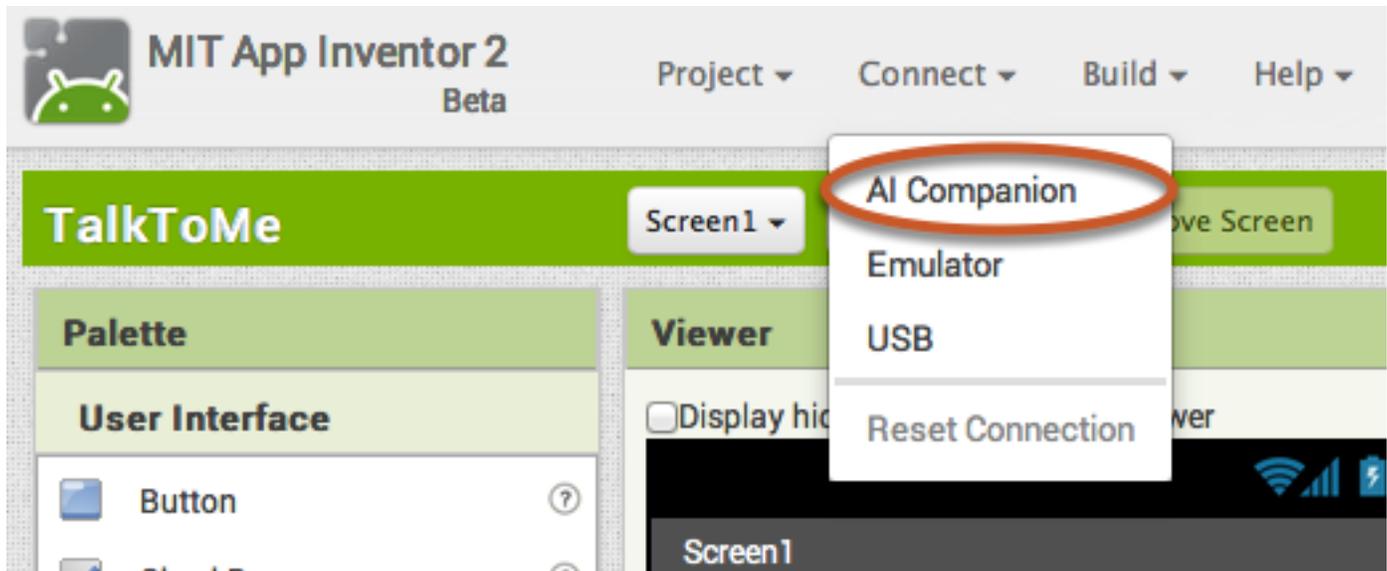
Add a Button

Our project needs a button. **Click and hold** on the word "Button" in the palette. **Drag** your mouse over to the Viewer. **Drop** the button and a new button will appear on the Viewer.



Connect App Inventor to your phone for live testing

One of the neatest things about App Inventor is that you can see and test your app while you're building it, on a connected device. If you have an **Android phone or tablet**, follow the steps below. *If you do not have a device*, then follow the instructions for [setting up the on-screen emulator](#) (opens a new page) and then come back to this tutorial once you've gotten the emulator connected to App Inventor.

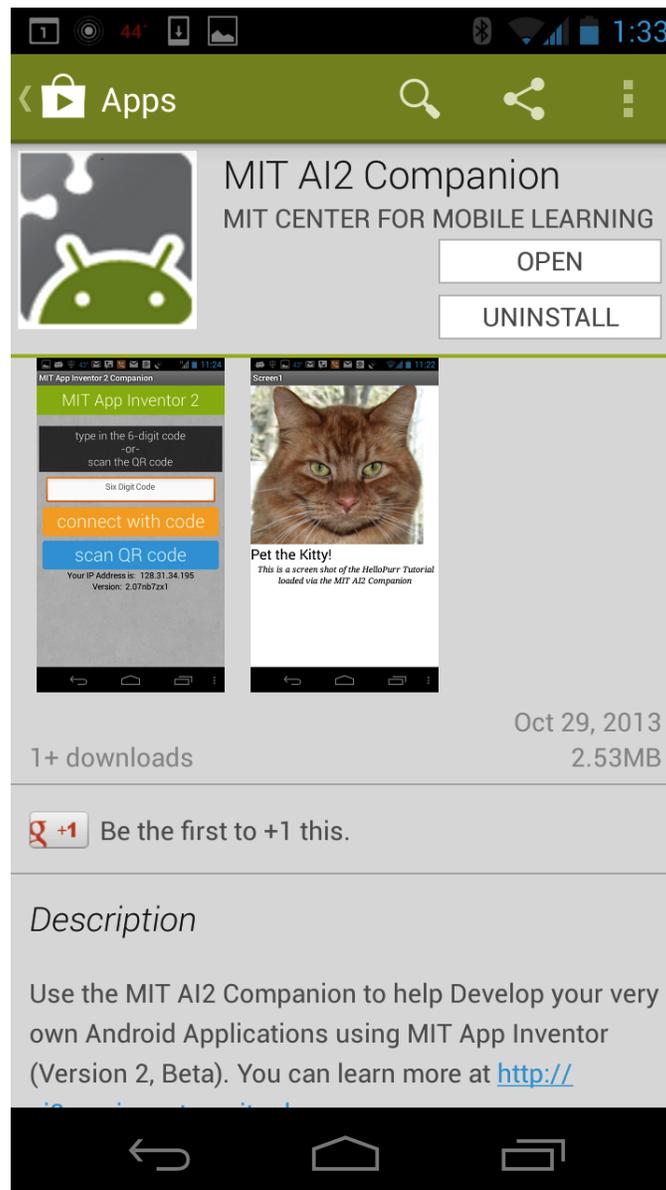




MIT App Inventor
appinventor.mit.edu

Get the MIT AI2 Companion from the Play Store and install it on your phone or tablet.

The preferred method for getting the AI2 Companion App is to **download the app from the Play Store by searching for "MIT AI2 Companion"**.





To download the AI2 Companion App to your device directly (SKIP THIS STEP IF YOU already got the app from Play Store)

If for some reason you can not connect to the Google Play store, you can download the AI2 Companion as described here.

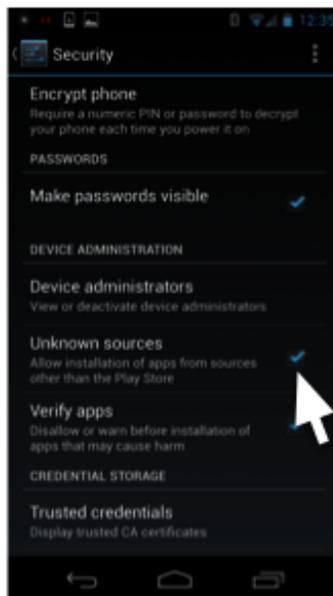
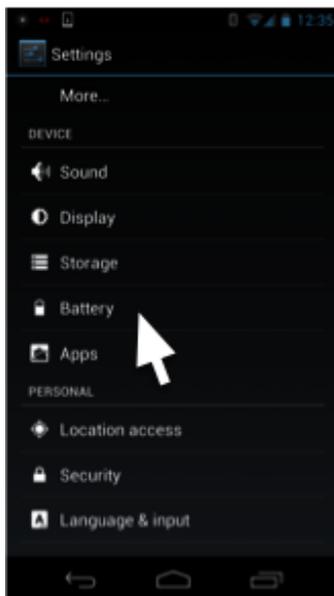
First, you will need to go into your phone's settings (#1), choose "Security", then scroll down to allow "Unknown Sources", which allows apps that are not from the Play Store to be installed on the phone.

Second, do one of the following:

A) **Scan the QR code above (#2)**

or

B) Click the "Need help finding..." link and you'll be taken to the download page. From there you can download the MITAI2Companion.apk file to your computer and then move it over to your device to install it.



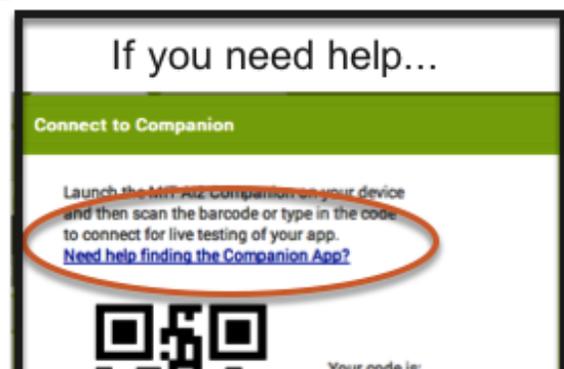
SKIP THIS STEP if you already got the AI2 Companion from the Play Store

1

1. Open your phone's settings and click "Security".
2. CHECK the box for "Unknown sources"

2

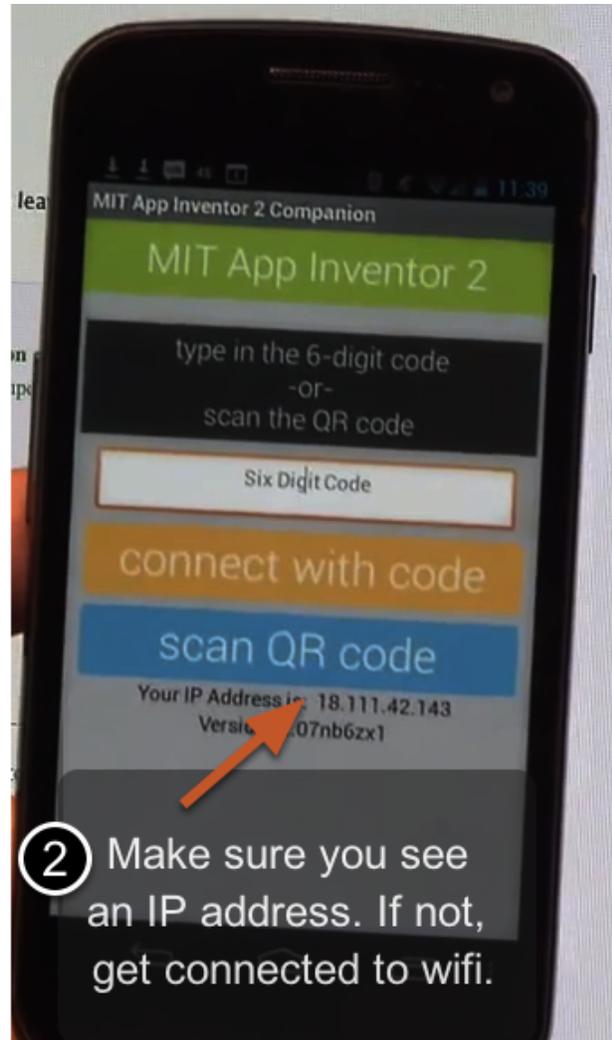
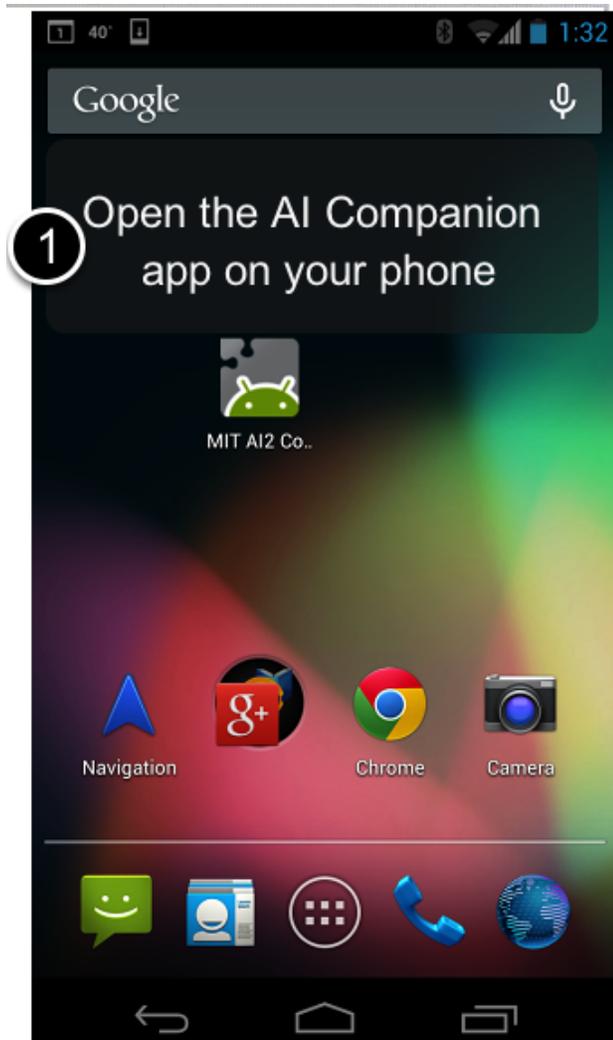
Scan to download MIT AI2 Companion directly to phone





Start the AI Companion on your device

On your phone or tablet, click the icon for the MIT AI Companion to start the app. NOTE: Your **phone and computer must both be on the same wireless network**. Make sure your phone's wifi is on and that you are connected to the local wireless network. If you can not connect over wifi, go to the Setup Instructions on the App Inventor Website to find out how to connect with a USB cable.





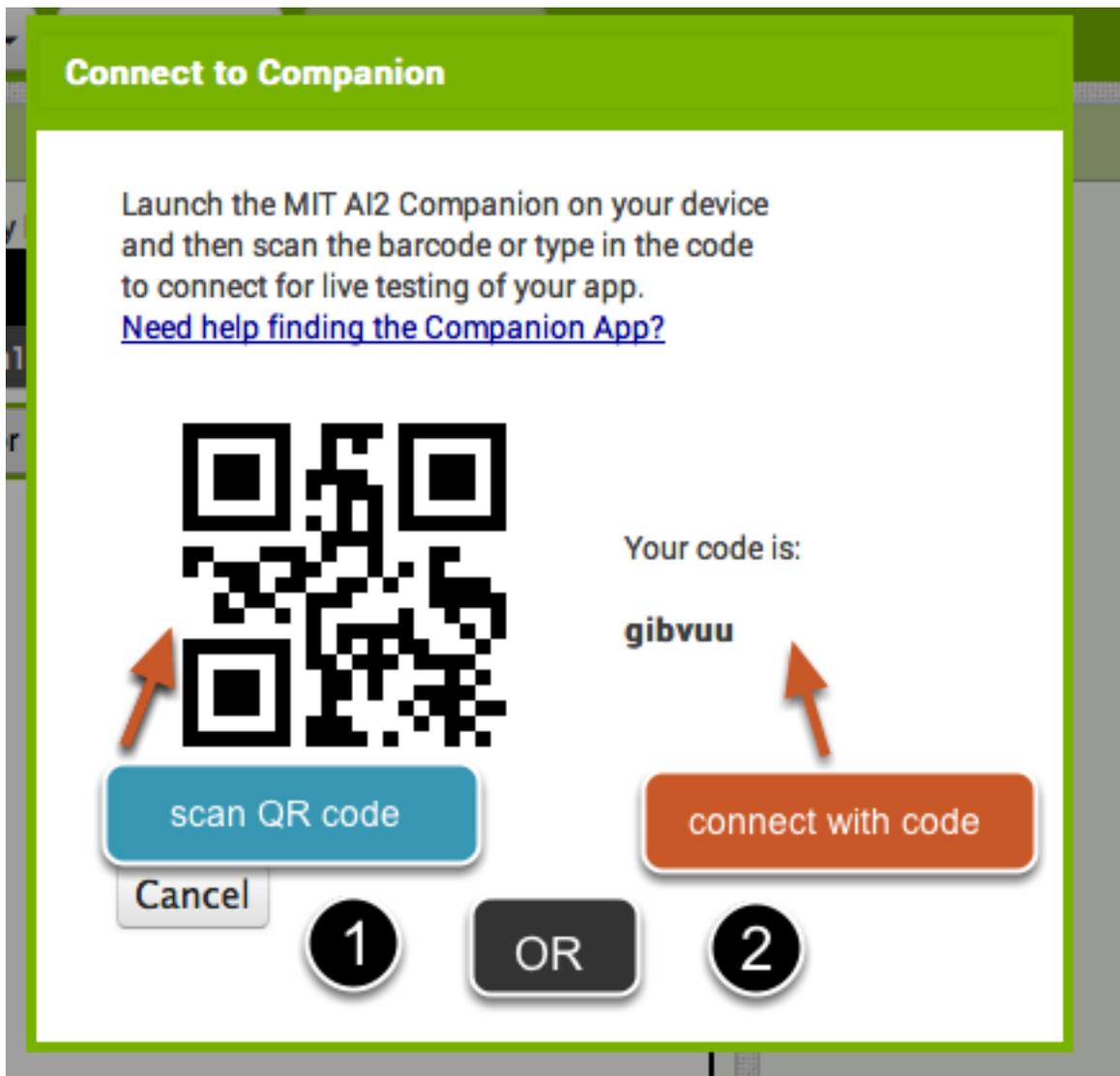
Get the Connection Code from App Inventor and scan or type it into your Companion app

On the Connect menu, choose "AI Companion". You can connect by:

1 - Scanning the QR code by clicking "Scan QR code" (#1).

or

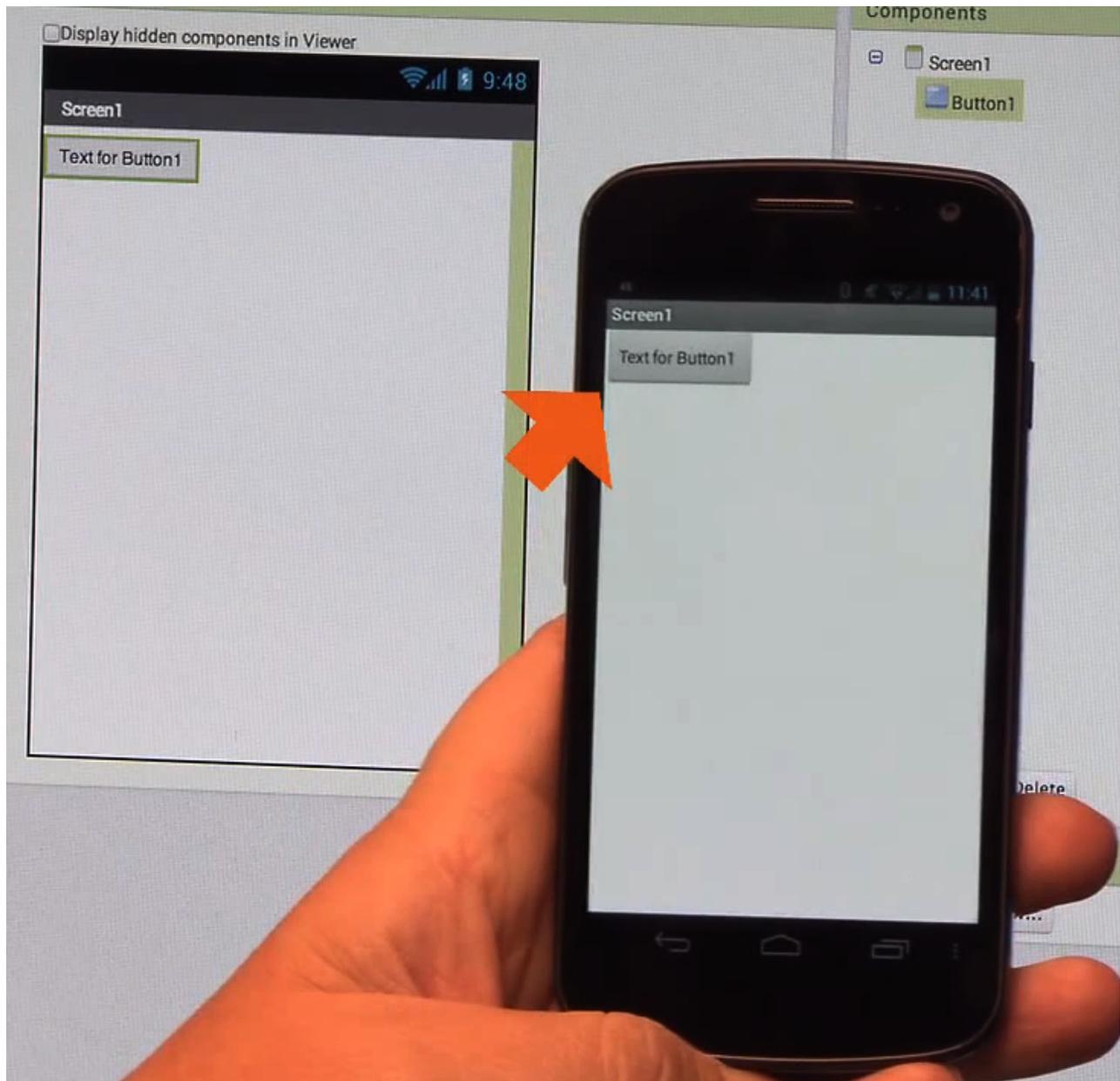
2 - Typing the code into the text window and click "Connect with code" (#2).





See your app on the connected device

You will know that your connection is successful when you see your app on the connected device. So far our app only has a button, so that is what you will see. As you add more to the project, you will see your app change on your phone.





Change the Text on the Button

On the properties pane, change the text for the Button. Select the text "Text for Button 1", delete it and type in "Talk To Me". Notice that the text on your app's button changes right away.

The screenshot displays the MIT App Inventor interface with three main panels: Viewer, Components, and Properties. In the Viewer panel, a mobile app preview shows a button with the text "Talk To Me". The Components panel shows a hierarchy with "Screen1" containing "Button1". The Properties panel is open for "Button1", showing various settings. A context menu is overlaid on the button in the Viewer, with the "Text" property highlighted. This context menu includes options for Shape (default), ShowFeedback (checked), Text (Talk To Me), TextAlignment (center), and TextColor (Default). Below the context menu, the main Properties panel also shows the "Text" property set to "Talk T".



Add a Text-to-Speech component to your app

Go to the Media drawer and drag out a TextToSpeech component. Drop it onto the Viewer. Notice that it drops down under "Non-visible components" because it is not something that will show up on the app's user interface. It's more like a tool that is available to the app.

The screenshot displays the MIT App Inventor interface with four main panels: Palette, Viewer, Components, and Properties.

- Palette:** The 'Media' category is circled in red. The 'TextToSpeech' component is also circled in red, with an orange arrow pointing from it to the Viewer.
- Viewer:** Shows a mobile app preview with a button labeled 'Talk To Me'. Below the preview, a grey box contains the text: 'Drop here. Component will automatically show up in Non-visible components area below'. Below this box, the 'Non-visible components' area contains a 'TextToSpeech1' component.
- Components:** Shows a tree view with 'Screen1' containing 'Button1' and 'TextToSpeech1'. The 'TextToSpeech1' component is highlighted with a green box.
- Properties:** Shows the properties for the selected 'TextToSpeech1' component, including 'TextToS', 'Country', and 'Language'.



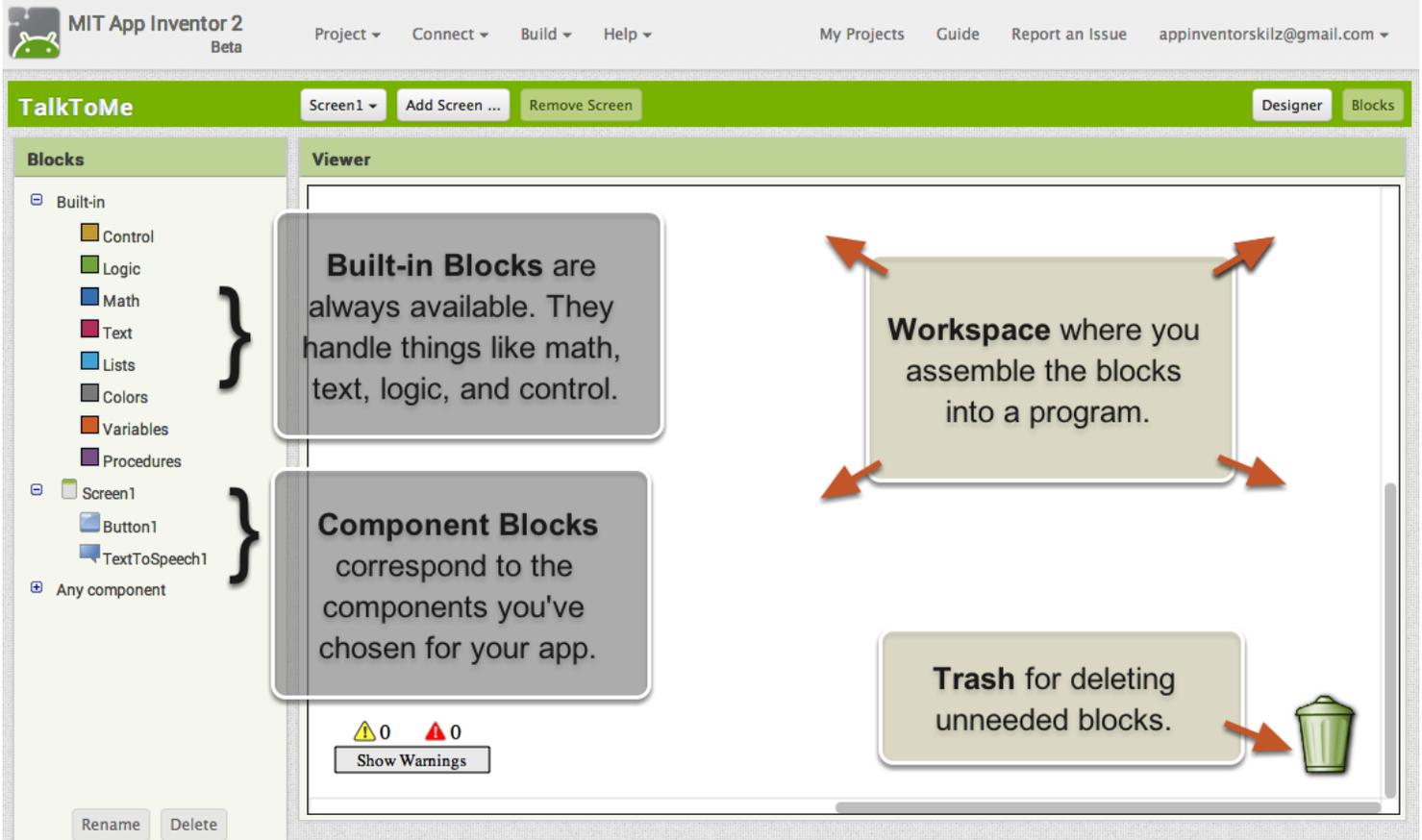
Switch over to the Blocks Editor

It's time to tell your app what to do! Click "Blocks" to move over to the Blocks Editor. Think of the Designer and Blocks buttons like tabs -- you use them to move back and forth between the two areas of App Inventor.

The screenshot shows the MIT App Inventor web interface. At the top, there is a navigation bar with links for "My Projects", "Guide", "Report an Issue", and a user profile "appinventorskilz@gmail.com". Below this is a green header bar with two tabs: "Designer" and "Blocks". The "Blocks" tab is highlighted with a red circle. The main workspace is divided into three panels: a preview area on the left showing a mobile app interface with a status bar at 9:48; a "Components" panel in the middle showing a hierarchy with "Screen1" containing "Button1"; and a "Properties" panel on the right for "Button1" with settings for BackgroundColor (Default), Enabled (checked), FontBold, and FontItalic.

The Blocks Editor

The Blocks Editor is where you program the behavior of your app. There are Built-in blocks that handle things like math, logic, and text. Below that are the blocks that go with each of the components in your app. *In order to get the blocks for a certain component to show up in the Blocks Editor, you first have to add that component to your app through the Designer.*



MIT App Inventor 2 Beta

Project ▾ Connect ▾ Build ▾ Help ▾

My Projects Guide Report an Issue appinventorskilz@gmail.com ▾

TalkToMe Screen1 ▾ Add Screen ... Remove Screen Designer Blocks

Blocks

- Built-in
 - Control
 - Logic
 - Math
 - Text
 - Lists
 - Colors
 - Variables
 - Procedures
- Screen1
 - Button1
 - TextToSpeech1
- Any component

Viewer

Built-in Blocks are always available. They handle things like math, text, logic, and control.

Component Blocks correspond to the components you've chosen for your app.

Workspace where you assemble the blocks into a program.

Trash for deleting unneeded blocks.

0 0
Show Warnings

Rename Delete



Make a button click event

Click on the Button1 drawer. Click and hold the **when Button1.Click do** block. Drag it over to the workspace and drop it there. This is the block that will handle what happens when the button on your app is clicked. It is called an "Event Handler".

The screenshot displays the MIT App Inventor 2 Beta interface. The top navigation bar includes the MIT App Inventor logo, the text "MIT App Inventor 2 Beta", and menu items for "Project", "Connect", "Build", and "Help". On the right side of the navigation bar are links for "My Projects", "Guide", and "Rep". Below the navigation bar is a header for the current project, "TalkToMe", with buttons for "Screen1", "Add Screen ...", and "Remove Screen".

The interface is divided into two main sections: "Blocks" on the left and "Viewer" on the right. In the "Blocks" section, under the "Built-in" category, the "Control" sub-category is expanded, and the "Button1" block is circled in red, with a circled number "1" next to it. In the "Viewer" section, a list of event handler blocks for "Button1" is shown. The top block, "when Button1.Click do", is circled in red, with a circled number "2" next to it. An orange arrow points from this block to a new "when Button1.Click do" block that is being dragged into the workspace from the right, with a circled number "3" next to it. Below the event handler blocks, there are several property blocks for "Button1", including "BackgroundColor", "Enabled", and "set Button1.BackgroundColor to", and "set Button1.Enabled to".

Program the TextToSpeech action

Click on the TextToSpeech drawer. Click and hold the **call TextToSpeech1.Speak** block. Drag it over to the workspace and drop it there. This is the block that will make the phone speak. Because it is inside the Button.Click, it will run when the button on your app is clicked.



The screenshot shows the MIT App Inventor 2 Beta interface. The top bar includes the MIT App Inventor logo, the text "MIT App Inventor 2 Beta", and navigation menus for "Project", "Connect", "Build", and "Help". On the right, there are links for "My Projects", "Guide", and "Report an Issue".

The main workspace is titled "TalkToMe" and contains a "Screen1" dropdown, "Add Screen ..." and "Remove Screen" buttons. The "Blocks" panel on the left is expanded to show "Screen1", which contains "Button1" and "TextToSpeech1". The "TextToSpeech1" block is circled with a red circle and labeled with a "1".

The "Viewer" panel on the right shows the workspace. It contains several blocks: a "when TextToSpeech1 .AfterSpeaking" block, a "when TextToSpeech1 .BeforeSpeaking" block, a "call TextToSpeech1 .Speak message" block (circled with a red circle and labeled with a "2"), a "TextToSpeech1 . Country" block, a "set TextToSpeech1 . Country to" block, and a "TextToSpeech1 . Language" block. A "when Button1 .Click" block is also present, containing the "call TextToSpeech1 .Speak message" block (circled with a red circle and labeled with a "3"). An orange arrow points from the "call TextToSpeech1 .Speak message" block in the "Viewer" panel to the "call TextToSpeech1 .Speak message" block in the "when Button1 .Click" block.



Fill in the message socket on TextToSpeech.Speak Block

Almost done! Now you just need to tell the TextToSpeech.Speak block what to say. To do that, click on the Text drawer, drag out a **text** block and plug it into the socket labeled "message".

The screenshot shows the MIT App Inventor interface for an app named "TalkToMe". The interface is divided into a "Blocks" panel on the left and a "Viewer" panel on the right. In the "Blocks" panel, the "Text" drawer is circled in orange. In the "Viewer" panel, a script is visible: "when Button1 Click" followed by "do call TextToSpeech1 .Speak message". The "message" socket and a "Text" block are also circled in orange, with an arrow pointing from the "Text" block in the drawer to the "message" socket.

Specify what the app should say when the button is clicked

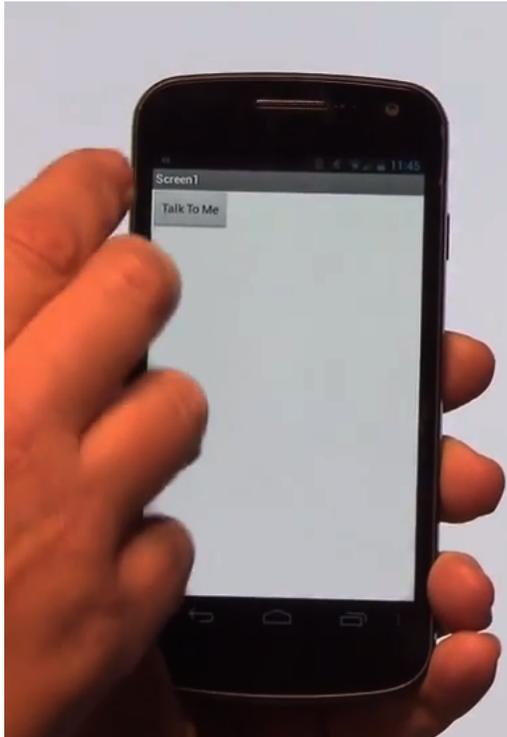
Click on the text block and type in "Congratulations! You've made your first app." (Feel free to use any phrase you like, this is just a suggestion.)

The screenshot shows a close-up of the MIT App Inventor script editor. The script starts with "when Button1 Click" followed by "do call TextToSpeech1 .Speak". The "message" socket of the "TextToSpeech1 .Speak" block is filled with the text "Congratulations! You've made your first app.".



Now test it out!

Go to your connected device and click the button. Make sure your volume is up! You should hear the phone speak the phrase out loud. (This works even with the emulator.)



Great job!

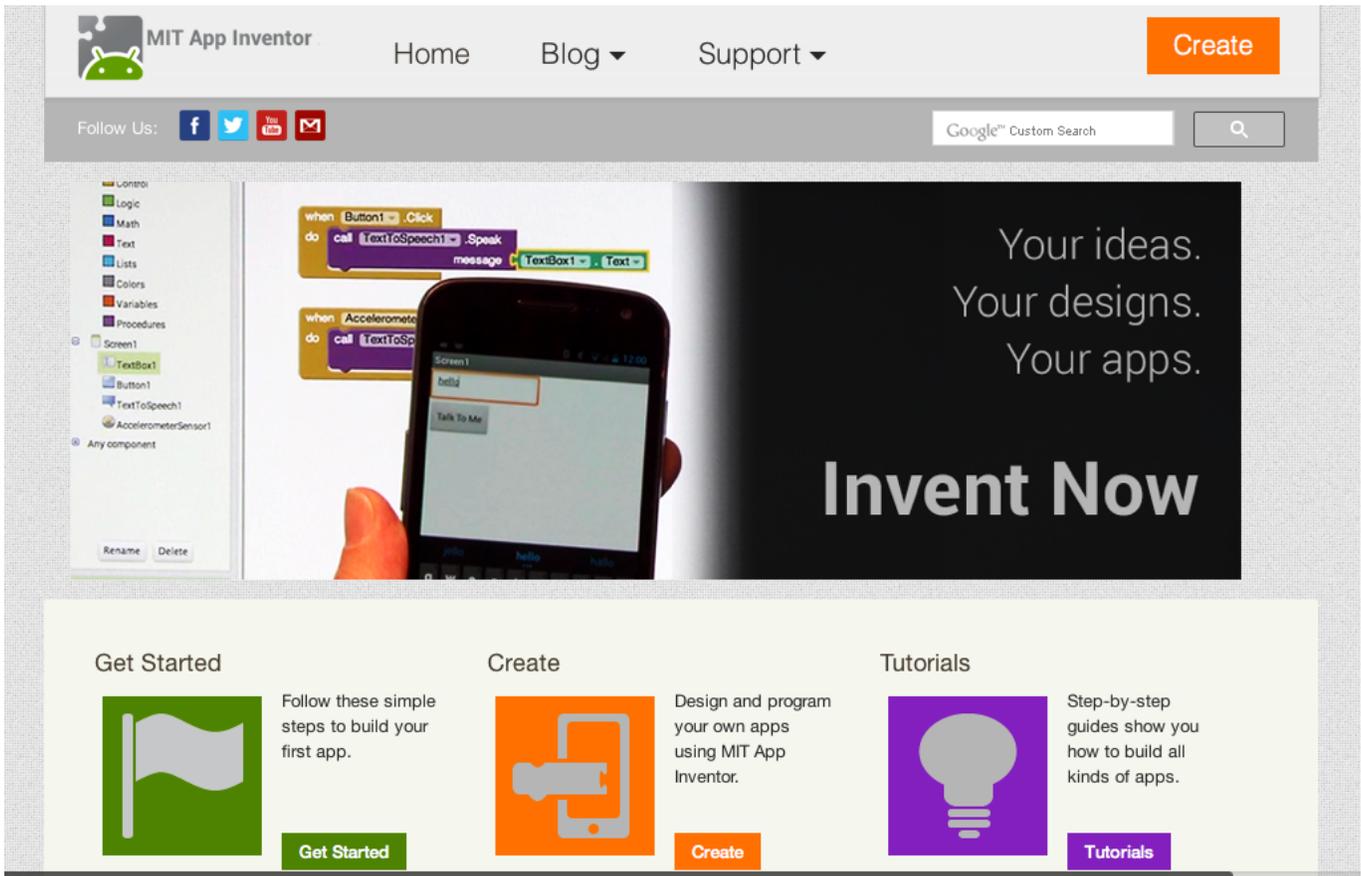
Now move on to TalkToMe Part 2 to make the app respond to shaking and to let users put in whatever phrase they want.

TalkToMe Part 2: Shaking and User Input

This tutorial shows you how to extend the basic TalkToMe app so that it responds to shaking, and so that the user can make the phone say any phrase s/he types in.

Go to App Inventor on the web and log in.

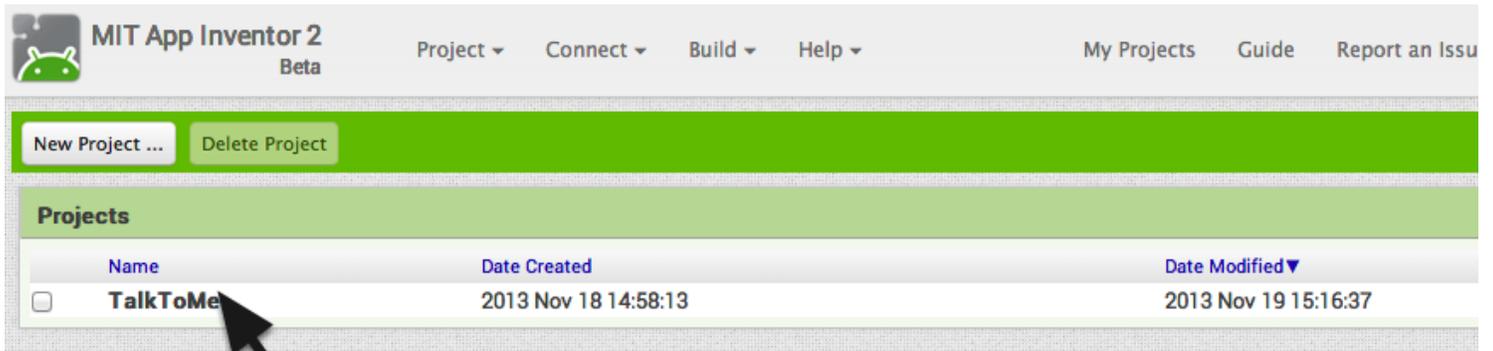
Go to appinventor.mit.edu and click "Create" or log in directly at ai2.appinventor.mit.edu.



The screenshot shows the MIT App Inventor website homepage. At the top, there is a navigation bar with the MIT App Inventor logo, links for Home, Blog, and Support, and a prominent orange "Create" button. Below the navigation bar, there are social media icons for Facebook, Twitter, YouTube, and Email, along with a Google Custom Search box. The main content area features a large banner with a smartphone displaying a "Talk To Me" app interface. The banner includes the text "Your ideas. Your designs. Your apps." and "Invent Now". Below the banner, there are three main sections: "Get Started" with a flag icon and a "Get Started" button, "Create" with a smartphone icon and a "Create" button, and "Tutorials" with a lightbulb icon and a "Tutorials" button. Each section provides a brief description of the resource.

Open the "TalkToMe" project that you worked on in the last tutorial.

App Inventor will always open the last project you worked on, so you may automatically be taken into your TalkToMe app.



Go to the Designer Tab

Your project may open in the Designer. If it does not, click "Designer" in the upper right.





Add an Accelerometer Sensor

In the **Sensors** drawer, drag out an AccelerometerSensor component and drop it onto the Viewer. (It's a non-visible component, so it drops to the bottom of the screen.) NOTE: emulator users should skip this part and proceed to the next section of this tutorial called "Say Anything". (The emulator can not respond to shaking!)

The screenshot displays the MIT App Inventor interface. On the left is the **Palette** with various categories. The **Sensors** category is expanded, and the **AccelerometerSensor** component is circled in orange. A red arrow points from this component to a circular callout with the number '2' in the **Viewer** area. The **Viewer** shows a mobile emulator with a status bar at the top displaying '9:48' and a button labeled 'Talk To Me'. Below the emulator, the **Non-visible components** section is visible, containing 'TextToSpeech1' and 'AccelerometerSensor1'. The 'AccelerometerSensor1' component is highlighted with a green box and a circular callout with the number '3'. The **Component** palette on the right is partially visible, showing icons for 'Screen', 'Image', 'Text', 'TextToSpeech', and 'Media'.



Go to the Blocks Editor

Click "Blocks" to program the new Accelerometer Sensor that you just added.



Program the Accelerometer Shaking event

Click the AccelerometerSensor1 drawer to see its blocks. Drag out the **when AccelerometerSensor1.Shaking do** block and drop it on the workspace.

The screenshot shows the MIT App Inventor interface with two main panels: 'Blocks' on the left and 'Viewer' on the right.

Blocks Panel: A list of built-in components is shown. Under the 'AccelerometerSensor1' category, the 'AccelerometerSensor1' block is highlighted with a red circle. Other categories include Control, Logic, Math, Text, Lists, Colors, Variables, and Procedures.

Viewer Panel: A workspace containing several blocks. The 'when AccelerometerSensor1.Shaking do' block is highlighted with a red circle and an arrow pointing to the right. Other blocks include 'when AccelerometerSensor1.AccelerationChanged do', 'AccelerometerSensor1.Available', 'AccelerometerSensor1.Enabled', 'set AccelerometerSensor1.Enabled to', 'AccelerometerSensor1.MinimumInterval', and 'set AccelerometerSensor1.MinimumInterval to'.



What do we want the app to do when the accelerometer detects shaking?

Copy and paste the blocks that are currently inside the when Button1.Click event handler. You can select the purple block, then hit the key combination on your computer to copy and then to paste. You'll have a second set of blocks to put inside the when Accelerometer.Shaking block.

(Alternatively, you can drag out a new *call TextToSpeech1.Speak* block from the TextToSpeech drawer, and a new pink *text* block from the Text drawer.)

when Button1 .Click
do
call TextToSpeech1 .Speak
message " Congratulations! You've made your first app. "

Tip: You can copy and paste blocks! Just use the same key combination that you use for copy and pasting text.

when AccelerometerSensor1 .Shaking
do
call TextToSpeech1 .Speak
message " Congratulations! You've made your first app. "

Change the phrase that is spoken when the phone is shaking.

Type in something funny for when the phone responds to shaking.

when AccelerometerSensor1 .Shaking
do
call TextToSpeech1 .Speak
message " Stop Shaking Me! "



Test it out!

You can now shake your phone and it should respond by saying "Stop shaking me!" (or whatever phrase you put in.)



Say Anything

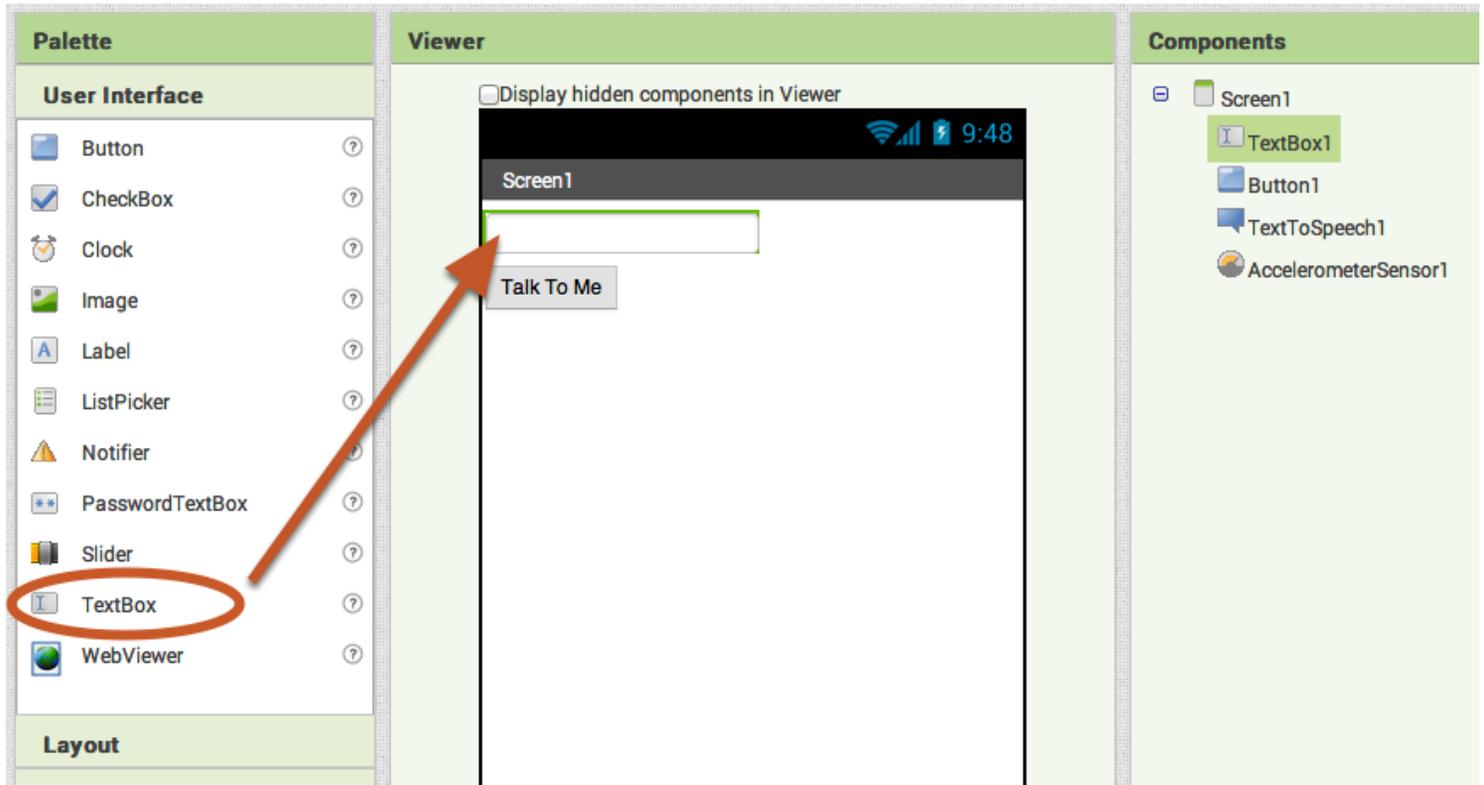
Is your phone talking to you? Cool! Now let's program the button click so that it causes the phone to speak whatever phrase the user put into the text box. Go back to the Designer.





Add a Text Box to your user interface.

From the User Interface drawer, drag out a TextBox and put it above the Button that is already on the screen.



Back to the Blocks Editor!





Get the text that is typed into the TextBox.

Get the text property of the TextBox1. The green blocks in the TextBox1 drawer are the "getters" and "setters" for the TextBox1 component. You want your app to speak out loud whatever is currently in the TextBox1 Text property (i.e. whatever is typed into the text box). Drag out the **TextBox1.Text** getter block.

The screenshot shows the MIT App Inventor interface. On the left is the 'Blocks' palette, and on the right is the 'Viewer' showing a sequence of blocks for a TextBox1 component. The 'TextBox1.Text' block is highlighted with a red circle, and an orange arrow points from it towards the right side of the interface.

Blocks

- Built-in
 - Control
 - Logic
 - Math
 - Text
 - Lists
 - Colors
 - Variables
 - Procedures
- Screen1
 - TextBox1
 - Button1
 - TextToSpeech1
 - AccelerometerSensor1
- Any component

Viewer

- set TextBox1 . Height to
- TextBox1 . Hint
- set TextBox1 . Hint to
- TextBox1 . MultiLine
- set TextBox1 . MultiLine to
- TextBox1 . NumbersOnly
- set TextBox1 . NumbersOnly to
- TextBox1 . Text**
- set TextBox1 . Text to
- TextBox1 . TextColor



Set the Button Click event to speak the text that is in the Text Box.

Pull out the "congratulations..." text box and plug in the TextBox1.Text block. You can throw the pink text block away by dragging it to the trash in the lower right corner of the workspace.

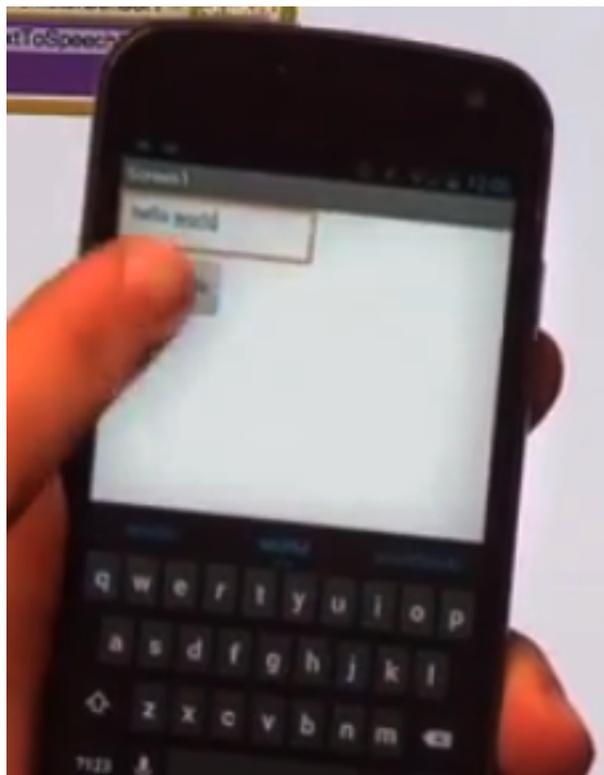


“ Congratulations! You've made your

Test your app!

Now your app has two behaviors: When the button is clicked, it will speak out loud whatever words are currently in the Text Box on the screen. (if nothing is there, it will say nothing.)

The app will also say "Stop Shaking Me" when the phone is shaken.



Congrats! You've built a real app!

Give some thought to what else this app could do. Here are some ideas for extensions:

- Random phrase generator
- Mad Libs - player chooses noun, verb, adjective, adverb, person and it picks one from a list that you program.
- Magic 8 Ball App
- Name picker - useful for teachers to call on a student

You could also play around with Speech-To-Text. Have fun!

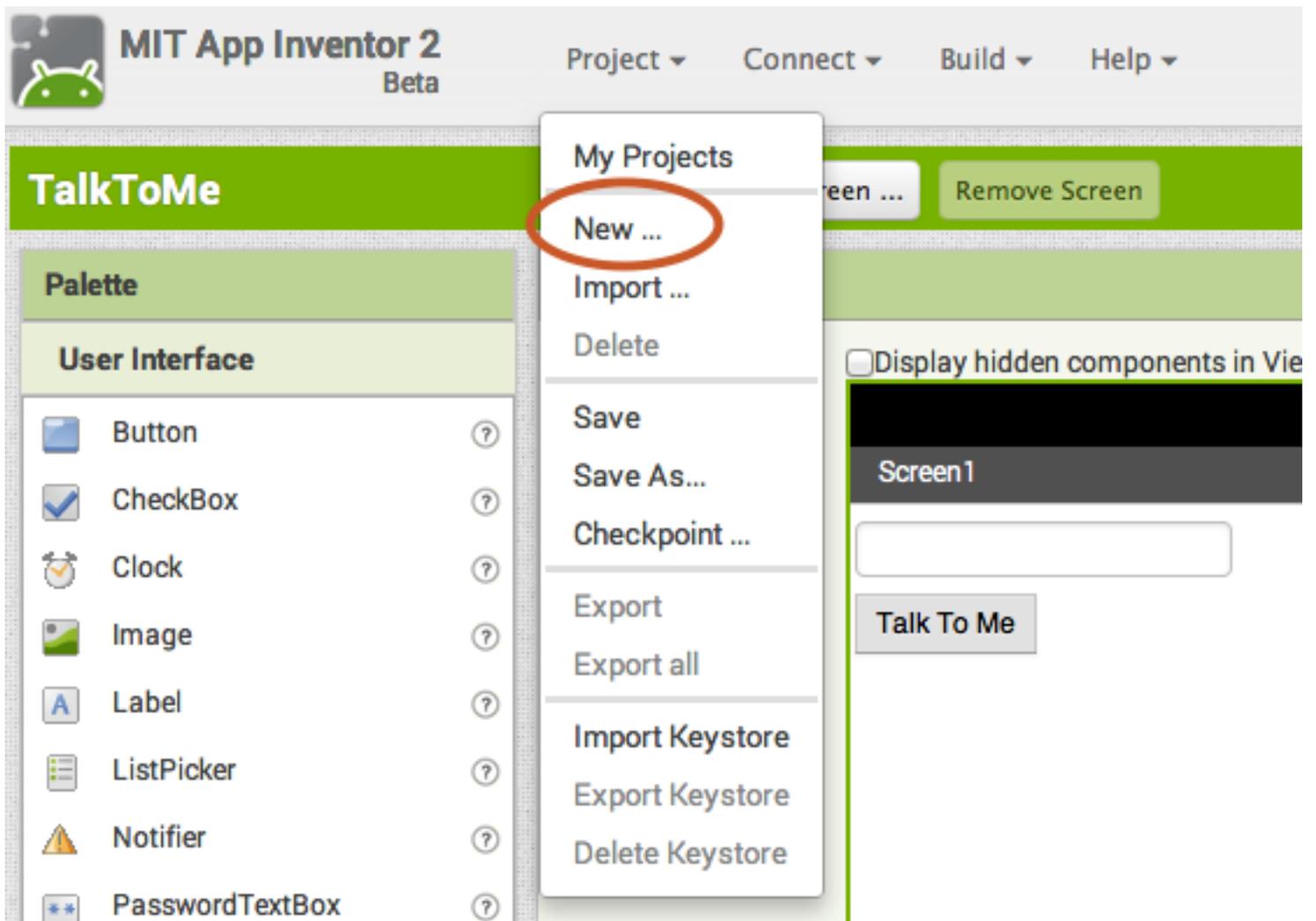


BallBounce: A simple game app

In this tutorial, you will learn about animation in App Inventor by making a Ball (a sprite) bounce around on the screen (on a Canvas).

Start a New Project

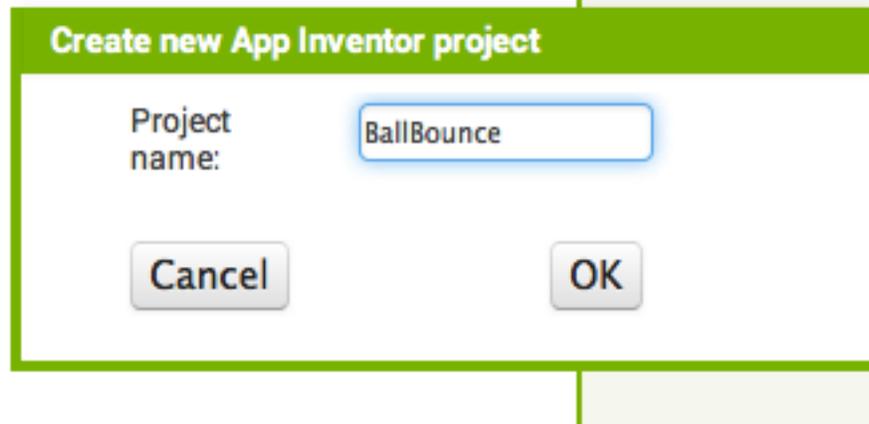
If you have another project open, go to My Projects menu and choose New Project.





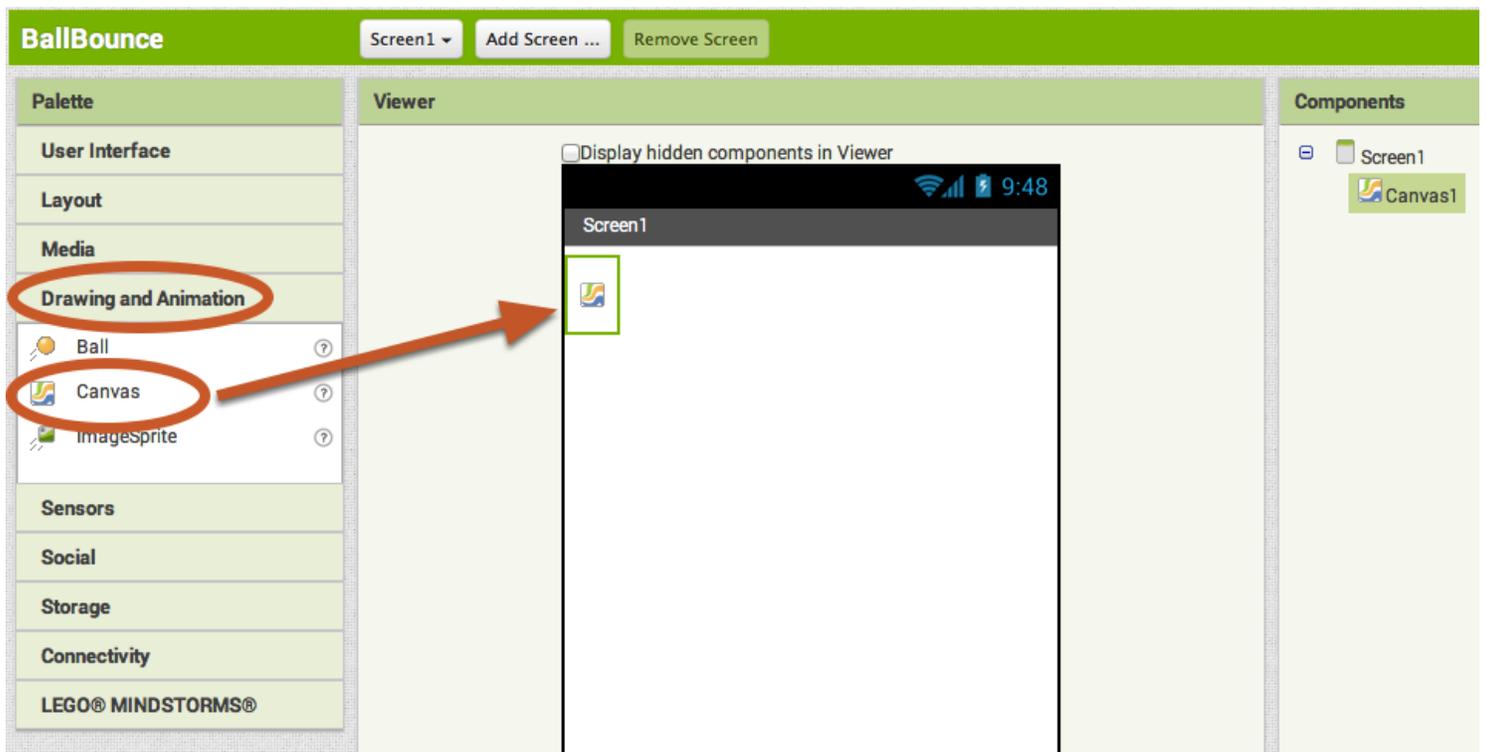
Name the Project

Call it something like "BallBounce". Remember, no spaces. But underscores are OK.



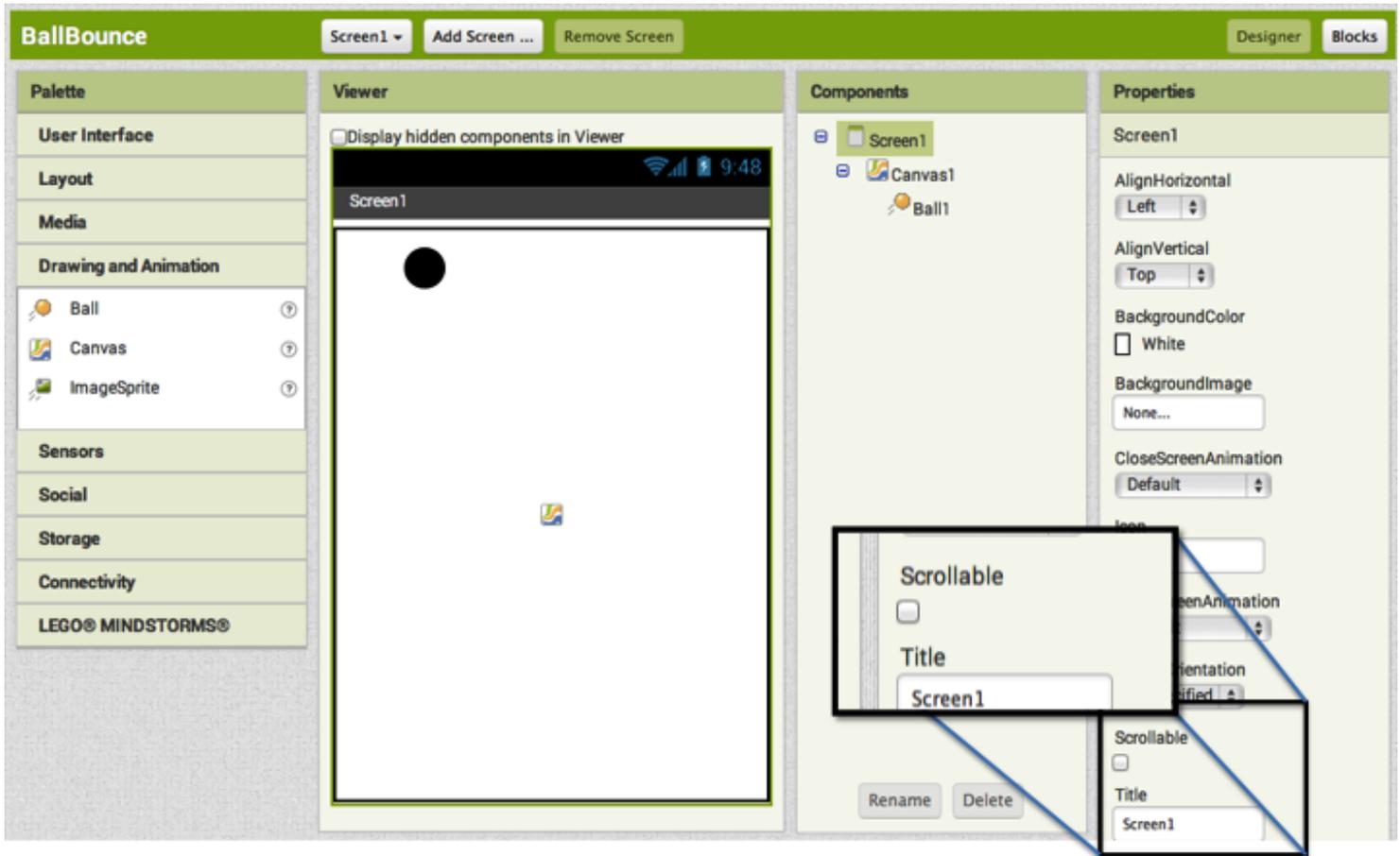
Add a Canvas

From the **Drawing and Animation drawer**, drag out a **Canvas component** and drop it onto the viewer.



Set the Screen so that it does not scroll

The default setting for App Inventor is that the screen of your app will be "scrollable", which means that the user interface can go beyond the limit of the screen and the user can scroll down by swiping their finger (like scrolling on a web page). When you are using a Canvas, you have to **turn off the "Scrollable" setting** (UNCHECK THE BOX) so that the screen does not scroll. This will allow you to make the Canvas to fill the whole screen.



The screenshot shows the MIT App Inventor Designer interface for an app named "BallBounce". The interface is divided into four main panels: Palette, Viewer, Components, and Properties.

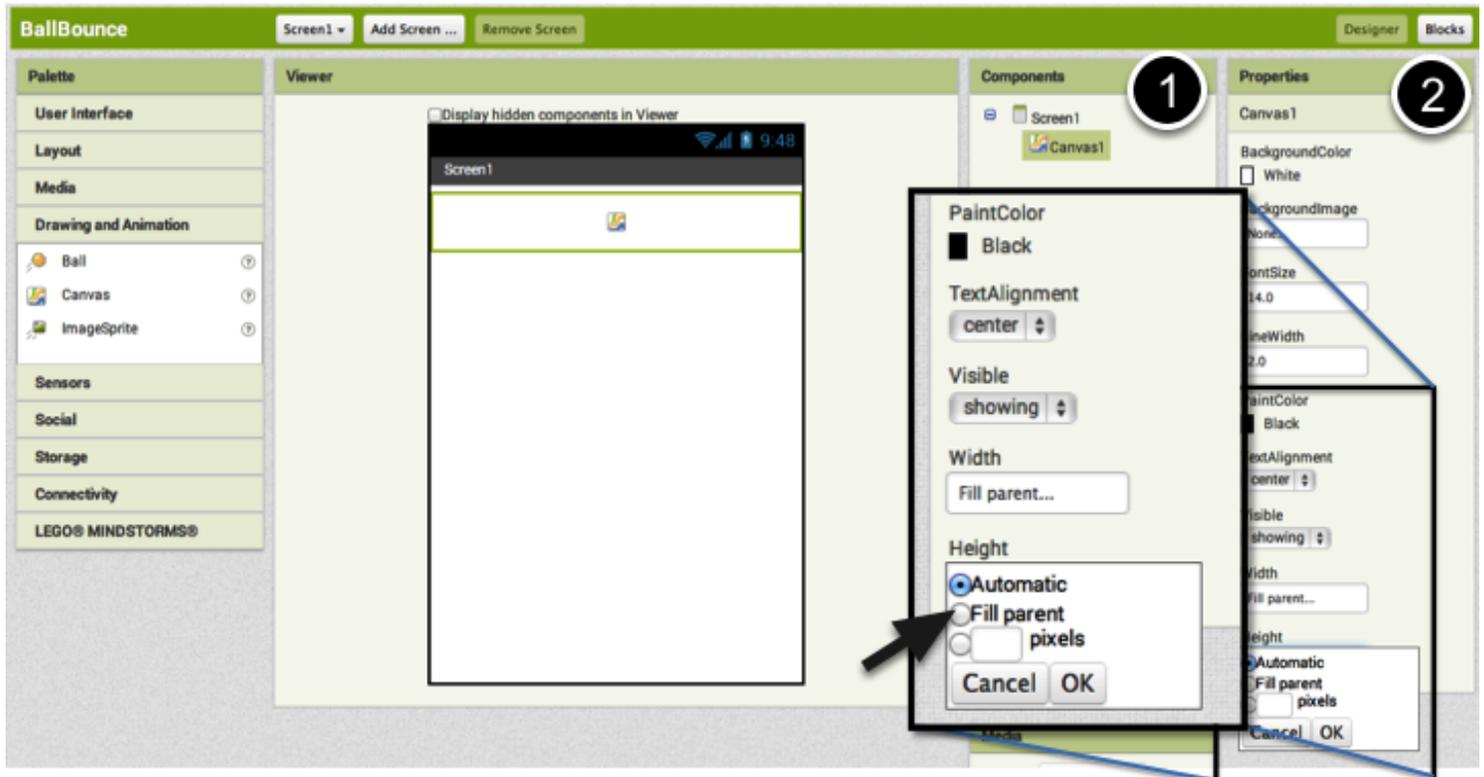
- Palette:** Contains categories like User Interface, Layout, Media, Drawing and Animation, Sensors, Social, Storage, Connectivity, and LEGO® MINDSTORMS®. Under "Drawing and Animation", there are items for Ball, Canvas, and ImageSprite.
- Viewer:** Displays a mobile device screen with a black circle (Ball) and a small icon (Canvas) on a white background. The status bar at the top shows signal strength, Wi-Fi, battery, and the time 9:48.
- Components:** Shows a tree view with "Screen1" containing "Canvas1" and "Ball1".
- Properties:** Shows the properties for "Screen1". The "Scrollable" property is highlighted with a red box and is currently unchecked. The "Title" property is set to "Screen1".

Two callout boxes are present: one pointing to the "Scrollable" checkbox in the Properties panel, and another pointing to the "Scrollable" checkbox in the Components panel, both indicating that this setting should be turned off.



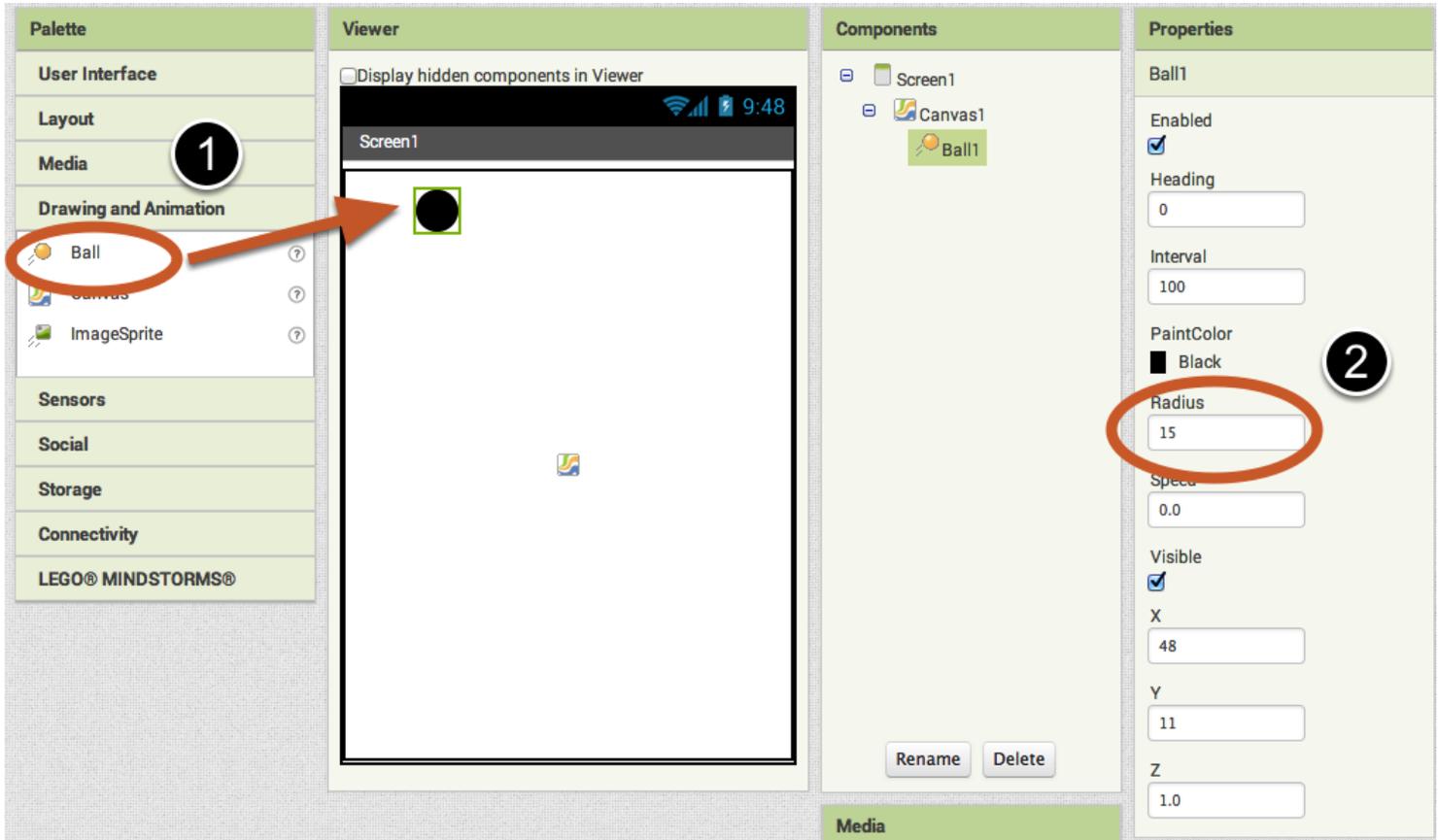
Change the Height and Width of the Canvas to Fill Parent

Make sure the Canvas component is selected (#1) so that its properties show up in the Properties Pane (#2). Down at the bottom, **set the Height property to "Fill Parent"**. Do the **same with the Width property**.



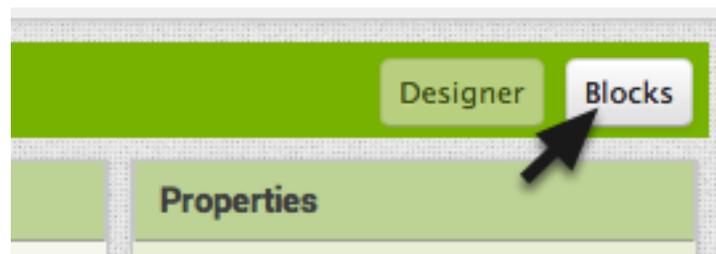
Add a Ball

Now that we have a Canvas in place, we can add a Ball Sprite. This can also be found in the **Drawing and Animation drawer**. Drag out a **Ball component** and drop it onto the Canvas (#1). If you'd like the ball to show up better, you can change its **Radius property** in the Properties pane (#2).



The screenshot shows the MIT App Inventor interface with four main panels: Palette, Viewer, Components, and Properties. In the **Palette** panel, the **Drawing and Animation** category is selected, and the **Ball** component is circled in red with a circled '1' next to it. An orange arrow points from the Ball component to a black ball on the **Canvas1** in the **Viewer** panel. In the **Components** panel, the **Ball1** component is visible. In the **Properties** panel, the **Radius** property is set to 15 and is circled in red with a circled '2' next to it. Other properties shown include Enabled (checked), Heading (0), Interval (100), PaintColor (Black), Speed (0.0), Visible (checked), X (48), Y (11), and Z (1.0).

Open the Blocks Editor.





Open the Ball1 Drawer to view the Ball's blocks.

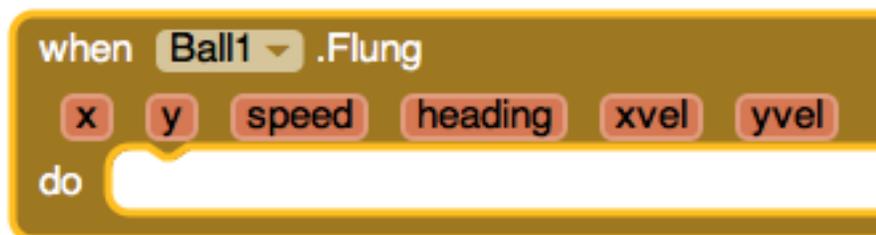
The screenshot shows the MIT App Inventor interface for a project named "BallBounce". The interface is divided into two main sections: "Blocks" on the left and "Viewer" on the right. The "Blocks" section shows a tree view of components, with "Ball1" highlighted in a red circle. The "Viewer" section shows a workspace with several event handler blocks for the "Ball1" component. The blocks are:

- when Ball1 .CollidedWith
- when Ball1 .Dragged
- when Ball1 .EdgeReached
- when Ball1 .Flung
- when Ball1 .NoLongerCollidingWith

Each block has a "do" field for user-defined code. The "when Ball1 .Flung" block has input fields for "x", "y", "speed", "heading", "xvel", and "yvel".

Drag out the Flung Event Handler

Choose the block **when Ball1.Flung** and drag-and-drop it onto the workspace. Flung refers to the user making a "Fling gesture" with his/her finger to "fling" the ball. Flung is a gesture like what a golf club does, not like how you launch Angry Birds! In App Inventor, the event handler for that type of gesture is called *when Flung*.





Set the Ball's Heading and Speed. First get the setter blocks.

Open the Ball drawer and scroll down in the list of blocks to get the **set Ball1.Heading** and **set Ball1.Speed** blocks

The screenshot shows the MIT App Inventor interface. On the left is the 'Blocks' palette, and on the right is the 'Viewer' showing a sequence of blocks for a component named 'Ball1'. The 'Ball1' component is selected in the 'Blocks' palette. In the 'Viewer', the following blocks are visible: 'set Ball1.Enabled to', 'Ball1.Heading', 'set Ball1.Heading to', 'Ball1.Interval', 'set Ball1.Interval to', 'Ball1.PaintColor', 'set Ball1.PaintColor to', 'Ball1.Radius', 'set Ball1.Radius to', 'Ball1.Speed', 'set Ball1.Speed to', and 'Ball1.Visible'. The 'set Ball1.Heading to' and 'set Ball1.Speed to' blocks are circled in orange. A text box on the right says 'Scroll down to the green "setter" blocks for the Ball's Heading and Speed' with a large orange arrow pointing downwards.

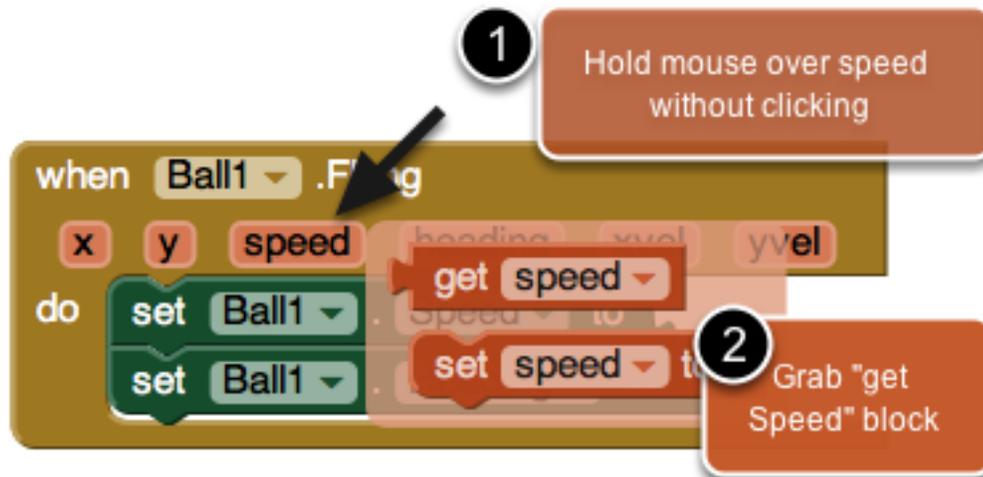


Plug the set Ball1.Speed and set Ball1.Heading into the Fling event handler

```
when Ball1 .Fling
  x y speed heading xvel yvel
do
  set Ball1 . Speed to
  set Ball1 . Heading to
```

Set the Ball's speed to be the same as the Fling gesture's speed

Mouse over the "speed" parameter of the **when Ball1.Fling** event handler. The get and set blocks for the speed of the fling will pop up. Grab the **get speed** block and plug that into the **set Ball1.Speed** block.





Set the Ball's heading to be the same as the Fling gesture's heading

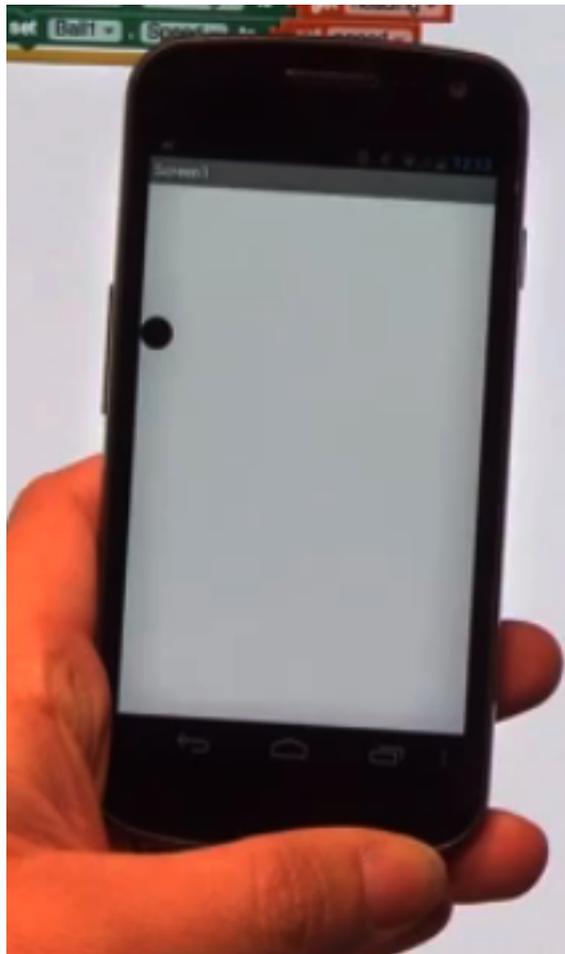
Do the same for the Ball's heading. Mouse over the **heading** parameter and you'll see the **get heading** block appear. Grab that block, and click it into the **set Ball1.Heading** block.

```
when Ball1 .Fling
  x y speed heading xvel yvel
do
  set Ball1 . Speed to get speed
  set Ball1 . Heading to get heading
```



Test it out

A good habit while building apps is to test while you build. App Inventor lets you do this easily because you can have a live connection between your phone (or emulator) and the App Inventor development environment. If you don't have a phone (or emulator) connected, go to the connection instructions and then come back to this tutorial. (Connection instructions are in Tutorial #1 or on the website under "Getting Started".)



Why does the Ball get stuck on the side of the screen?!

After flinging your ball across the screen, you probably noticed that it got stuck on the side. This is because the ball's heading has not changed even though it hit the side of the canvas. To make the ball "bounce" off the edge of the screen, we can program in a new event handler called "When Edge Reached".



Add an Edge Reached Event

Go into the Ball1 drawer and pull out a **when Ball1.EdgeReached** do event.

The screenshot shows the MIT App Inventor interface. On the left is the 'Blocks' panel, which is organized into categories: Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1, and Any component. The 'Ball1' component is selected and circled in red. On the right is the 'Viewer' panel, which displays a sequence of event blocks for 'Ball1':

- when Ball1 .CollidedWith other do
- when Ball1 .Dragged startX startY prevX prevY currentX currentY do
- when Ball1 .EdgeReached edge do (highlighted with a red circle and an arrow pointing right)
- when Ball1 .Moving x y speed heading xvel yvel do
- when Ball1 .NoLongerCollidingWith other do



Go back into the Ball1 drawer and pull out a Ball.Bounce block.

The screenshot shows the MIT App Inventor interface. On the left, the 'Blocks' panel is visible, showing a tree view with 'Screen1' and 'Canvas1' containing 'Ball1'. On the right, the 'Viewer' panel shows a code editor with several event blocks for 'Ball1'. A purple 'call Ball1.Bounce' block is circled in orange, and an orange arrow points from it to the 'edge' parameter of a 'when Ball1.EdgeReached' block.

Add the edge value for the Ball.Bounce block

The **Ball.Bounce** method needs an edge argument. Notice that the **Ball1.EdgeReached** event has an "edge" as a parameter. We can take the **get edge** block from that argument and plug it into the call **Ball1.Bounce** method. Grab the **get edge** block by mousing over (hover your mouse pointer over) the "edge" parameter on the when **Ball1.EdgeReached** block.

The close-up screenshot shows a 'when Ball1.EdgeReached' block with an 'edge' parameter. A 'get edge' block is being dragged from the 'edge' parameter to the 'call Ball1.Bounce' block. A 'set edge to' block is also visible below the 'call' block.



Your final blocks should look like this. Now test it out!

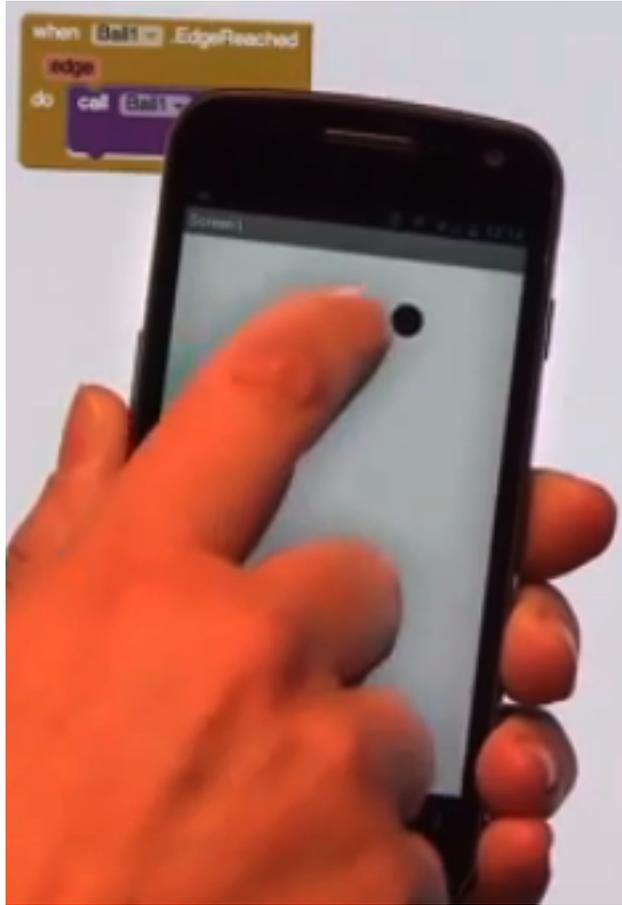
```
when Ball1 .Flung
  x y speed heading xvel yvel
do
  set Ball1 . Speed to get speed
  set Ball1 . Heading to get heading
```

```
when Ball1 .EdgeReached
  edge
do
  call Ball1 .Bounce
  edge get edge
```



Test it out!

Now, when you fling the ball, it should bounce off the edges of the canvas. Great job!



There are many ways to extend this app.

Here are some ideas... but the possibilities are endless!

- Change the color of the ball based on how fast it is moving or which edge it reaches.
- Scale the speed of the ball so that it slows down and stops after it gets flung.
- Give the ball obstacles or targets to hit
- Introduce a paddle for intercepting the ball, like a Pong game

Visit the [App Inventor website](#) to find tutorials that help you extend this app, particularly the [Mini Golf tutorial](#).

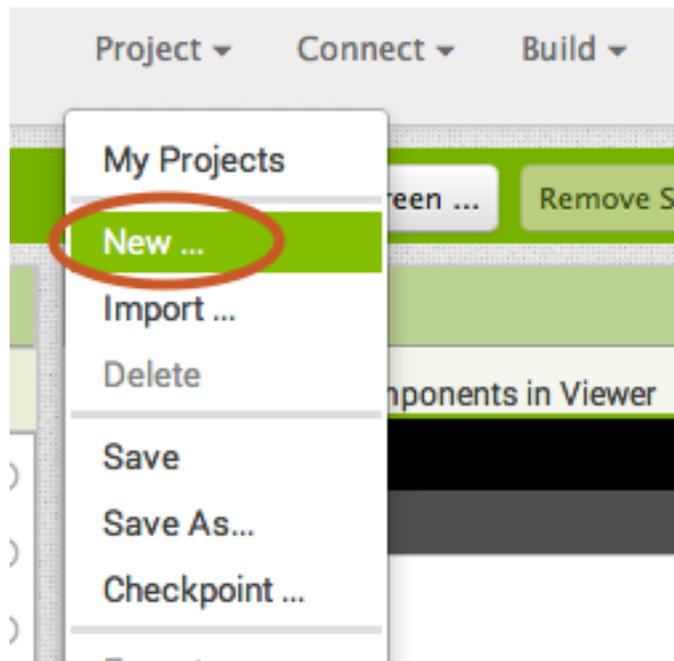
Have fun with these extensions, or others that you think up!

DigitalDoodle: Drawing App

This tutorial will show you how to draw a line on the screen as the user drags a finger around.

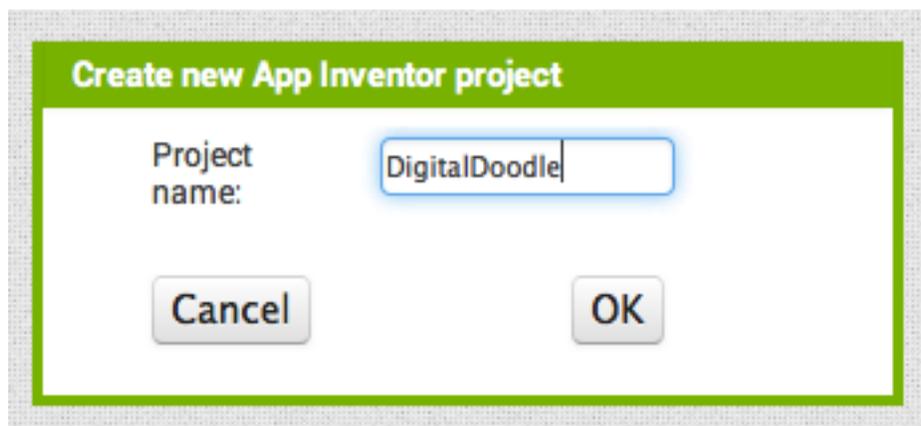
Start a New Project

From the My Projects page, click New Project. If you have another project open, go to My Projects menu and choose New Project.



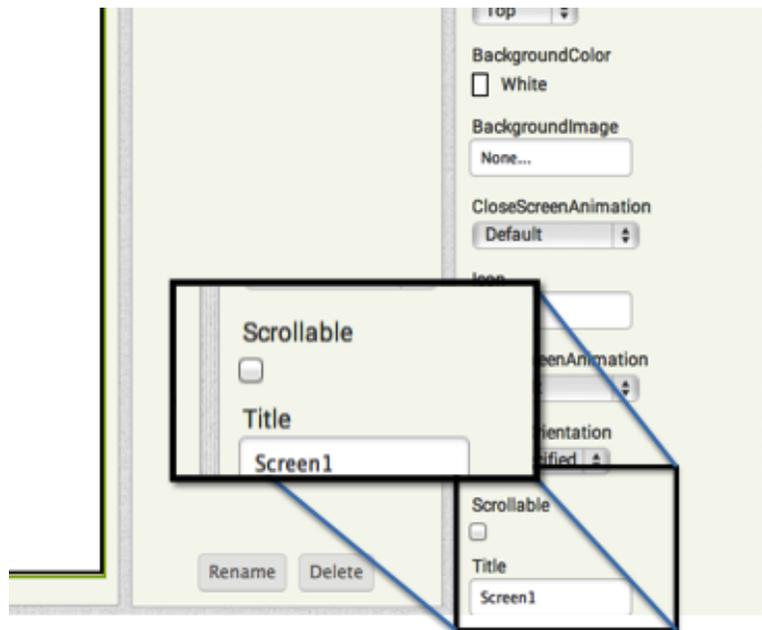
Name the Project

Call this project DigitalDoodle, or create your own name for this drawing app.



Set the Screen so that it does not scroll

The default setting for App Inventor is that the screen of your app will be "scrollable", which means that the user interface can go beyond the limit of the screen and the user can scroll down by swiping their finger (like scrolling on a web page). When you are using a Canvas, you have to **turn off the "Scrollable" setting** (UNCHECK THE BOX) so that the screen does not scroll. This will allow you to make the Canvas to fill up the whole screen.





Add a Canvas

From the Drawing and Animation drawer, drag out a Canvas component.

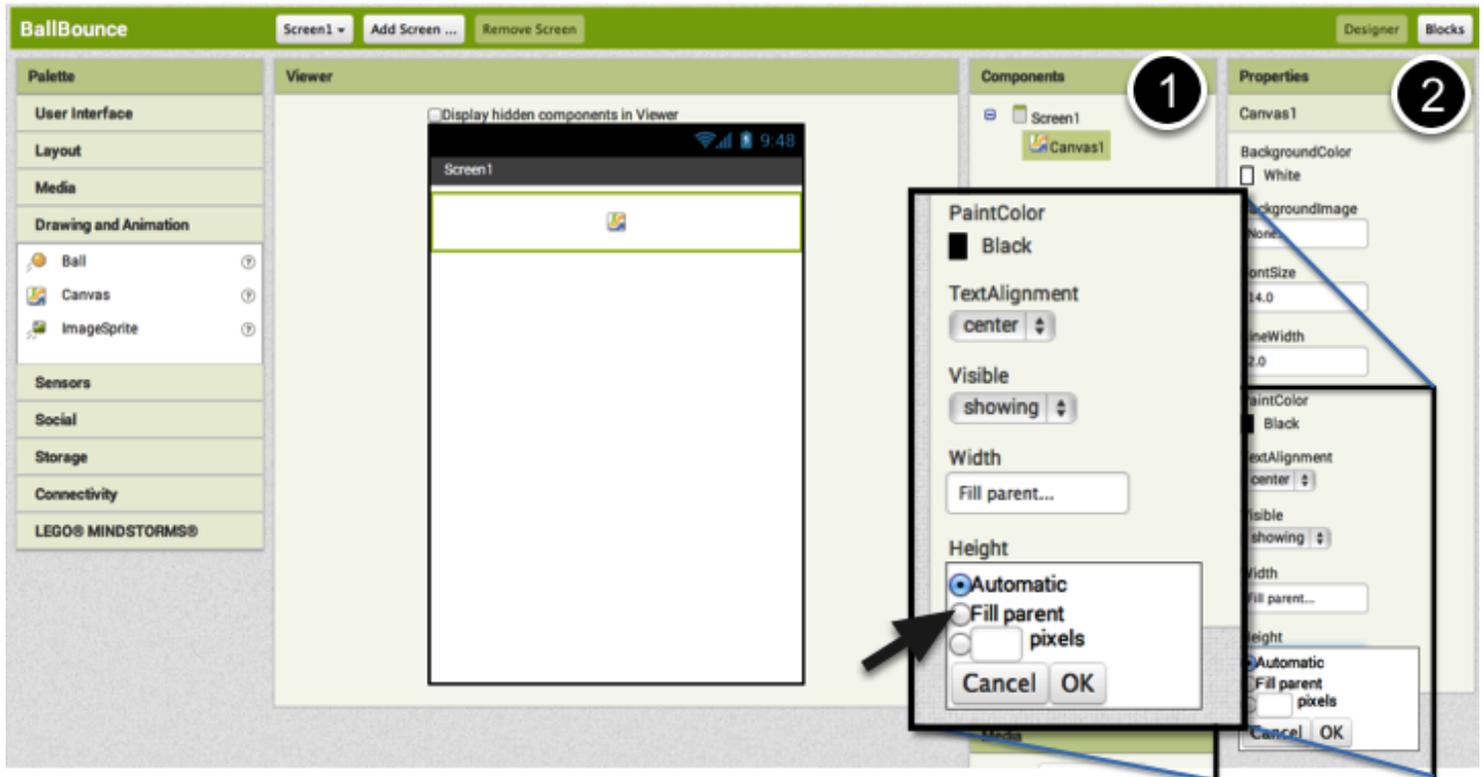
The screenshot displays the MIT App Inventor interface for a project named "DigitalDoodle". At the top, there are controls for "Screen1", "Add Screen ...", and "Remove Screen". The interface is divided into three main sections: "Palette", "Viewer", and "Components".

- Palette:** A vertical list of component categories. The "Drawing and Animation" category is expanded, showing "Ball", "Canvas", and "ImageSprite". The "Canvas" component is circled in red, and a red arrow points from it to the Viewer.
- Viewer:** A central area showing a mobile device screen. The screen has a status bar at the top with the time "9:48" and icons for Wi-Fi, signal strength, and battery. Below the status bar, the text "Screen1" is visible. A small "Canvas" component icon is placed on the screen, highlighted with a green border.
- Components:** A vertical list on the right side showing the components currently on the screen. It includes "Screen1" and "Canvas1".



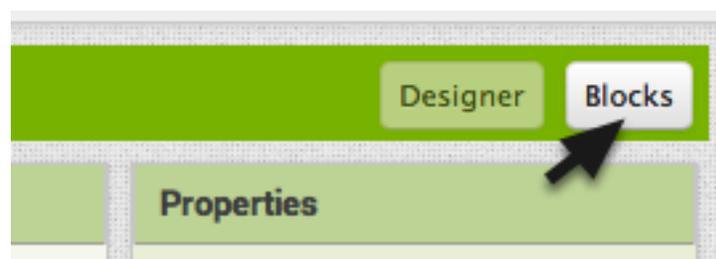
Change the Height and Width of the Canvas to Fill Parent

Make sure the Canvas component is selected (#1) so that its properties show up in the Properties Pane (#2). Down at the bottom, **set the Height property to "Fill Parent"**. Do the **same with the Width property**.



That's all for the Designer! Go over to the Blocks.

Believe it or not, for now this app only needs a Canvas. Go into the Blocks Editor to program the app.





Get a Canvas.Dragged event block

In the Canvas1 drawer, pull out the **when Canvas1.Dragged** event.

The screenshot shows the MIT App Inventor interface for a project named "DigitalDoodle". At the top, there are buttons for "Screen1", "Add Screen ...", and "Remove Screen". The interface is divided into two main sections: "Blocks" on the left and "Viewer" on the right.

In the "Blocks" section, under the "Built-in" category, the "Canvas1" component is selected and circled in red. The "Viewer" section displays three event blocks for the "Canvas1" component:

- The top block is "when Canvas1 .Dragged", which is circled in red. It includes parameters: `startX`, `startY`, `prevX`, `prevY`, `currentX`, `currentY`, and `draggedSprite`.
- The middle block is "when Canvas1 .Flung", with parameters: `x`, `y`, `speed`, `heading`, `xvel`, `yvel`, and `flungSprite`.
- The bottom block is "when Canvas1 .TouchDown", with parameters: `x` and `y`.



Get a Canvas.DrawLine call block

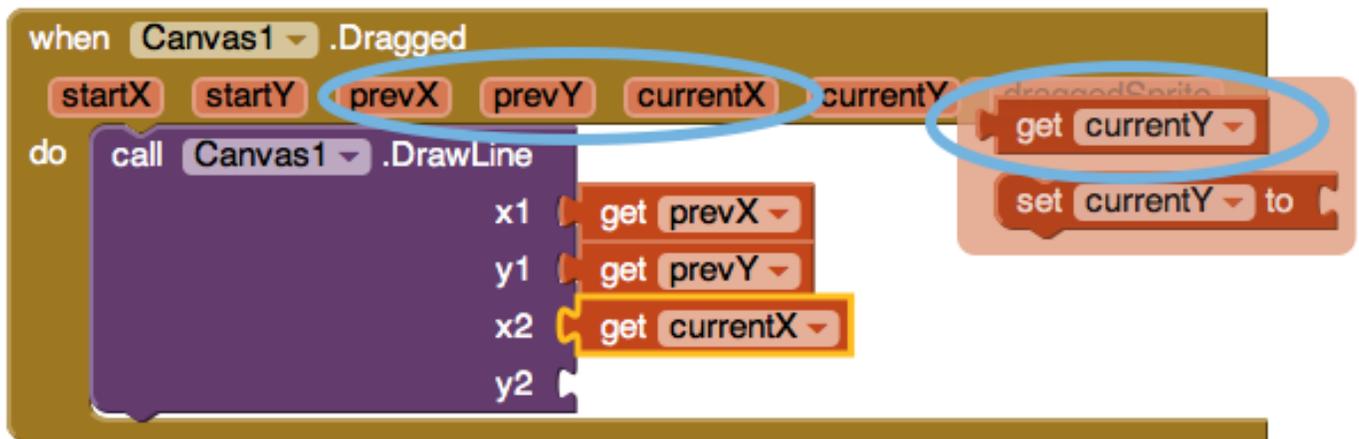
In the Canvas1 drawer, pull out the **when Canvas1.DrawLine** method block..

The screenshot shows the MIT App Inventor interface. On the left is the 'Blocks' palette, and on the right is the 'Viewer' showing a code block. In the 'Blocks' palette, the 'Canvas1' drawer is expanded and highlighted with an orange circle. In the 'Viewer', a 'when Canvas1.Touched' block is shown with 'x', 'y', and 'touchedSprite' inputs. Below it are 'call Canvas1.Clear', 'call Canvas1.DrawCircle', 'call Canvas1.DrawLine', and 'call Canvas1.DrawPoint' blocks. The 'call Canvas1.DrawLine' block is highlighted with an orange circle.



Use the get and set blocks from the Dragged block to fill in the values for the Draw Line block.

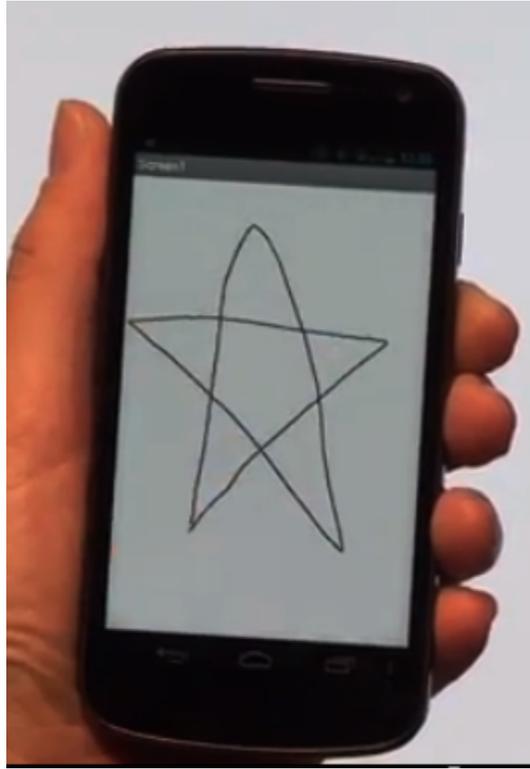
The Canvas Dragged event will happen over and over again very rapidly while the user drags a finger on the screen. Each time that Dragged event block is called, it will draw a small line between the previous location (prevX, prevY) of the finger to the new location (currentX, currentY). Mouse over the parameters of the Canvas1.Dragged block to pull out the get blocks that you need. (Mouse over them, don't click on them.)





Test it out!

Go to your connected device and drag your finger around the screen. Do you see a line?



Great work! Now extend this app

Here are some ideas for extending this app. You can probably think of many more!

- Change the color of the ink (and let the user pick from a selection of colors). See [Paint Pot tutorial](#).
- Change the background to a photograph or picture.
- Let the user draw dots as well as lines (hint: Use DrawCircle block).
- Add a button that turns on the camera and lets the user take a picture and then doodle on it.