**Programming Android Apps  
Columbus State University Activ8  
Dr. Wayne Summers (wsummers@columbusstate.edu)**

**Mon: Intro**

**Hello Kitty**

**Add your own**

**Tues: Review**

**Wildlife Menu**

**PaintPot**

**Add your own**

**Wed.: Review**

**8-Ball**

**Wack-a-Mole**

**Add Your own**

**Thurs: Review**

**Pong**

**BUILD YOUR OWN**

**Class Review**

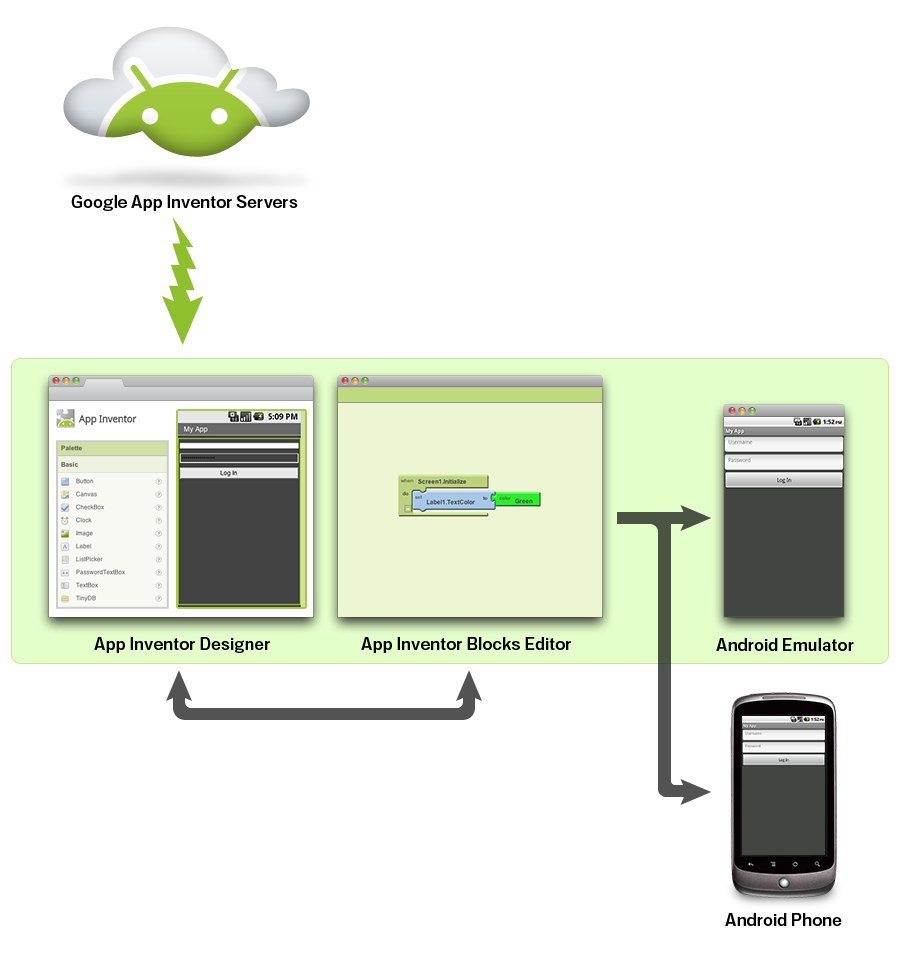
**Fri: Review**

**Final Revisions**

**Show-and-Tell**

**1. What is App Inventor (http://** **http://appinventor.mit.edu/)**

App Inventor lets you develop applications for Android phones using a web browser and either a connected phone or emulator. The App Inventor servers store your work and help you keep track of your projects.



You build apps by working with:

* The *App Inventor Designer* , where you select the components for your app.
* The *App Inventor Blocks Editor* , where you assemble program blocks that specify how the components should behave. You assemble programs visually, fitting pieces together like pieces of a puzzle.

Your app appears on the phone step-by-step as you add pieces to it, so you can test your work as you build. When you're done, you can package your app and produce a stand-alone application to install.

If you don't have an Android phone, you can build your apps using the *Android emulator*, software that runs on your computer and behaves just like the phone.

2.0 Building your first app with the emulator: Hello Purr

This page will get you started building your first app: A picture of a kitty that meows when you pet it. You can also [watch a video](http://www.youtube.com/watch?v=nC_x9iOby0g) (http://www.youtube.com/watch?v=nC\_x9iOby0g) of this app being built. When you're done building Hello Purr, you'll be ready to design and build apps on your own. Before starting, make sure that you've [set up your computer](http://beta.appinventor.mit.edu/learn/setup/index.html) .



As you build Hello Purr you'll learn how the three key tools of App Inventor work:

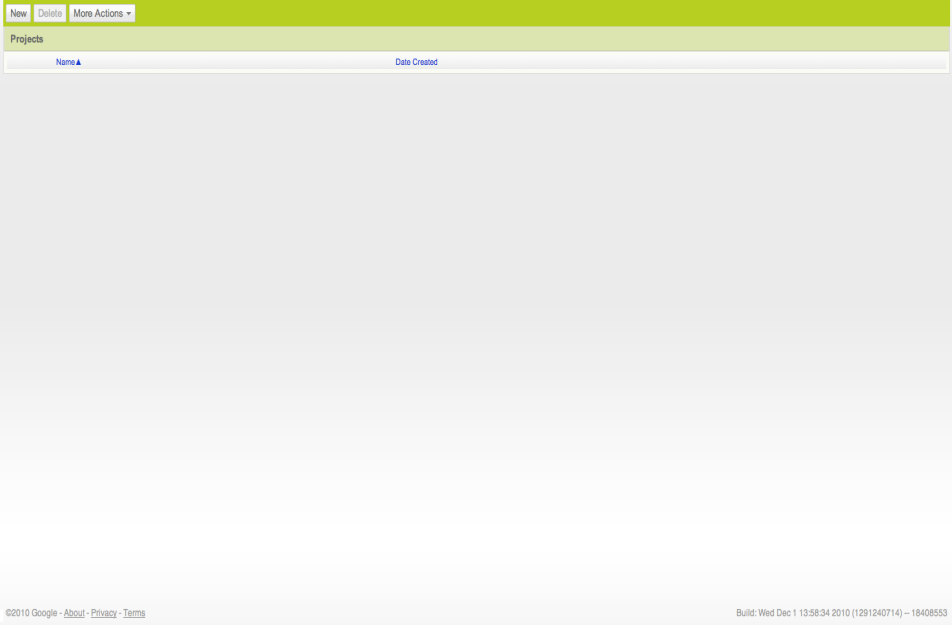
* The **Designer** , the place were you design your app. It runs in your web browser.
* The **Blocks Editor** , the place were you set the behavior of the app. It is a separate application with its own window.
* The **emulator** , a virtual mobile device that runs on your computer along side App Inventor.

To build Hello Purr you'll need a picture of the kitty and also a meow sound. Download these files to your computer:

* [kitty picture](http://beta.appinventor.mit.edu/learn/tutorials/hellopurr/HelloPurrAssets/kitty.png)
* [meow sound](http://beta.appinventor.mit.edu/learn/tutorials/hellopurr/HelloPurrAssets/meow.mp3)

## Start the Designer and create a new project

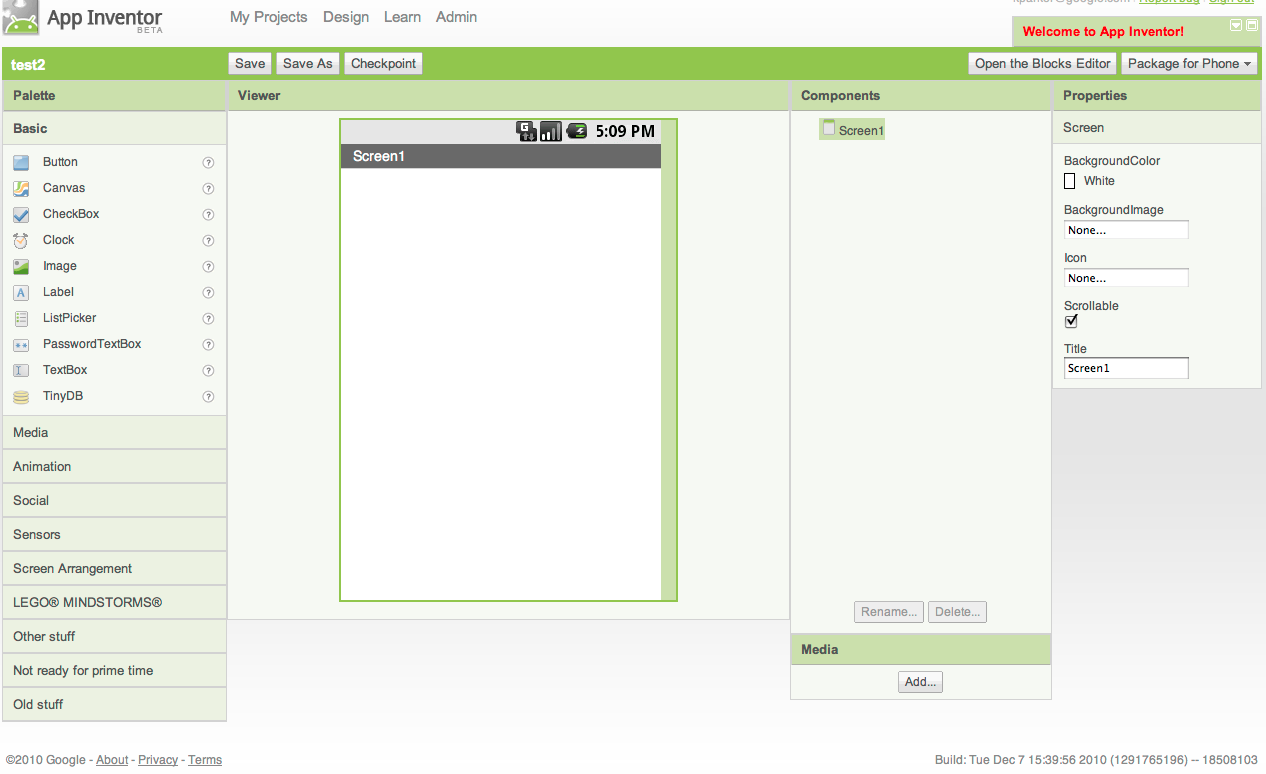
In your web browser, go to the App Inventor Web site at <http://appinventor.mit.edu>. If this is the first time you've used App Inventor, you'll see the Projects page, with no projects in it yet. It should look like this:



#### Create a new project

1. Click New on the left side, near the top of the page.
2. Enter the project name HelloPurr (one word, with no spaces) in the dialog box that appears, click OK.

The browser will open the Designer , the place where you select components for your app, and should look like this:



## Selecting components to design your app

App Inventor components are located on the left hand side of the Designer screen under the title **Palette** . Components are the basic elements you use to make apps on the Android phone. They're like the ingredients in a recipe. Some components are very simple, like a Label component, which just shows text on the screen, or a Button component that you tap to initiate an action. Other components are more elaborate: a drawing Canvas that can hold still images or animations, an accelerometer (motion) sensor that works like a Wii controller and detects when you move or shake the phone, components that make or send text messages, components that play music and video, components that get information from Web sites, and so on.

App Inventor components are located on the left hand side of the Designer screen under the title **Palette** . To use a component in your app you need to click and drag it onto the viewer in the middle of the Designer. When you add a component to the viewer it will also appear in the components list on the right hand side of the viewer.

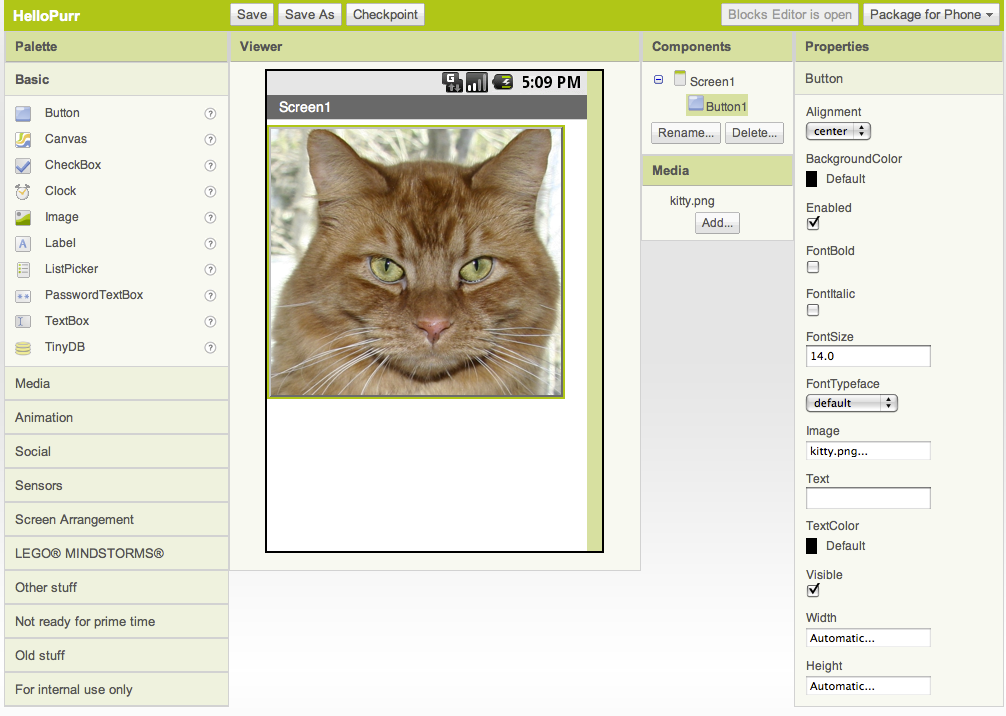
Components have properties that can be adjusted to change the way the component appears within the app. To view and change the properties of a component you must first select the desired component in your list of components.

#### Steps for selecting components and setting properties

HelloPurr will use a **Button** component that has the image property set to the kitty.png file you downloaded earlier. To do this:

1. Drag and drop the Button component to **Screen1** . The Button component is located under **Palette** .
2. Click on **Button1** listed under **Components** .
3. In the list of properties, under image, click on **none...**
4. Click add .
5. Upload the the kitty.png file.
6. Delete *Text for Button1* listed under the **Text** property.

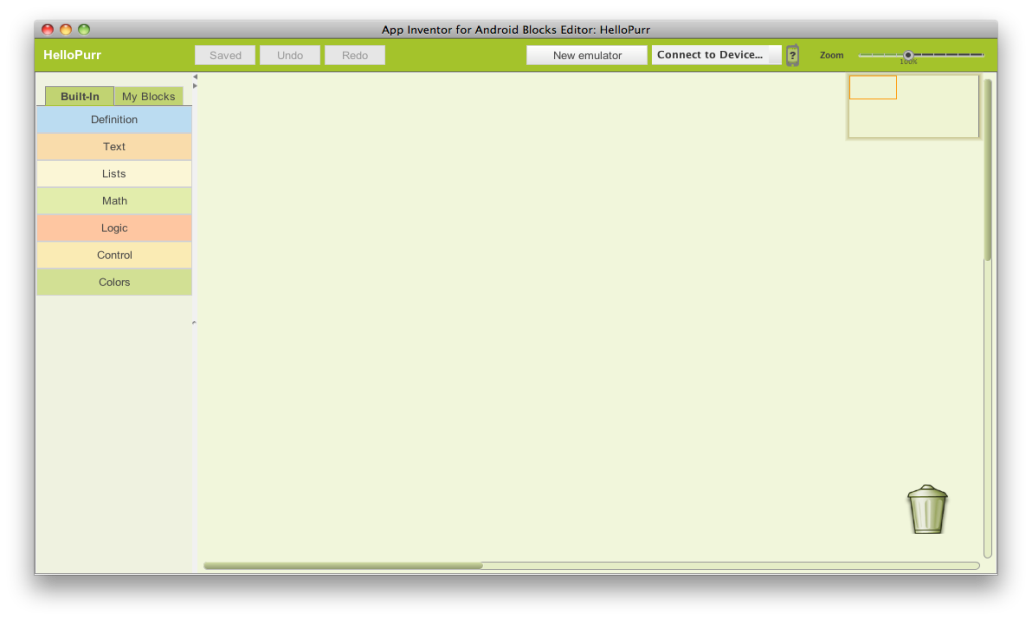
Your Designer should look like this:



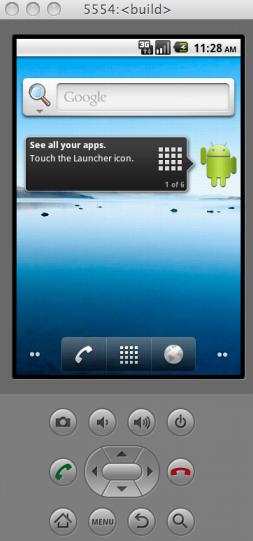
## Open the Blocks Editor and connect to the emulator

The Designer is one of three key tools you'll use in creating your apps. The second is the Blocks Editor . You'll use the Blocks Editor to assign behaviors to your components, such as what should happen when the user of your app taps a button.

The Blocks Editor runs in a separate window. When you click Open the blocks editor from the Designer window, the Blocks Editor program file should download and run. This process may take 30 seconds or longer. If the Blocks Editor never opens, it might be because your browser is not set up to run downloaded Java applications automatically. In this case, find the downloaded file named AppInventorForAndroidCodeblocks.jnlp and open it. The Blocks Editor window should look as shown below, with "drawers" for the program blocks to the left, and a large empty "canvas" space for placing blocks to assemble the program, which you'll do below.



Before continuing to build the app, you'll need to start the emulator. Click the New emulator button at top of the window. You'll get a notice saying that the emulator is starting, and asking you to be patient: starting the emulator can take a couple of minutes. The emulator will initially appear with an empty black screen. Wait until the emulator is ready, with a colored screen background as shown below. Even after the background appears, you should wait until the emulated phone has finished preparing its SD card: there will be a notice at the top of the phone screen while the card is being prepared. You might also need to use your mouse on the emulated phone screen to unlock the device by dragging the green lock button to the right.



When the emulator is finally ready, click the Connect to device... button, select the emulator from the dropdown list and click it. You'll see a yellow animated arrow move back and forth, showing that App Inventor is connecting to the emulated phone. Creating this connection can take another minute or two. When it's all done, the arrow will stop moving and turn green, and if you look at the emulated phone screen, you'll see the kitty there â€” this is the beginning of your app!

**Adding the label**

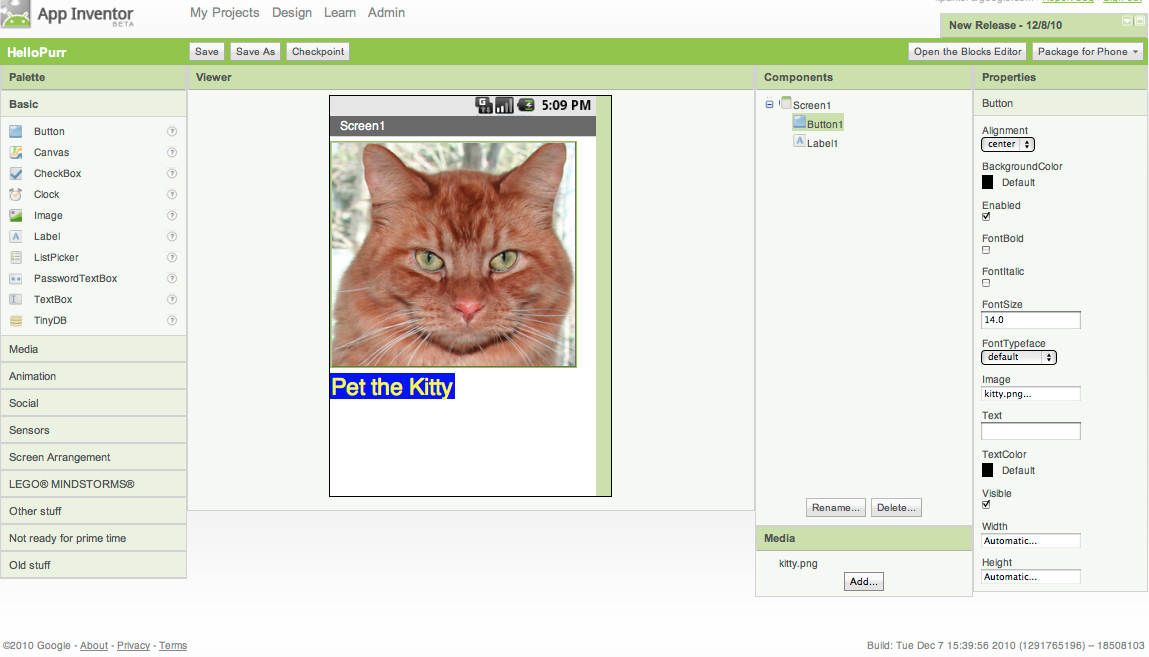
Under *Palette*

1. Drag and drop the Label component to the *Viewer* , placing it below the kitty. It will appear under your list of components as *Label1*

Under *Properties*

1. Change the Text property of Label1 to read "Pet the Kitty". You'll see the text change in the Designer and on your device.
2. Change the BackgroundColor of Label1 by clicking on the box
3. Change the TextColor of Label1
4. Change the FontSize of Label1 to 30

The Designer should now look like this:



**Adding the meow sound**

Under *Palette*

1. Click on the header marked Media to expand the Media section of the palette of components.
2. Drag out a Sound component and place it in the *Viewer* . Wherever you drop it, it will appear in the area at the bottom of the *Viewer* marked Non-visible components .

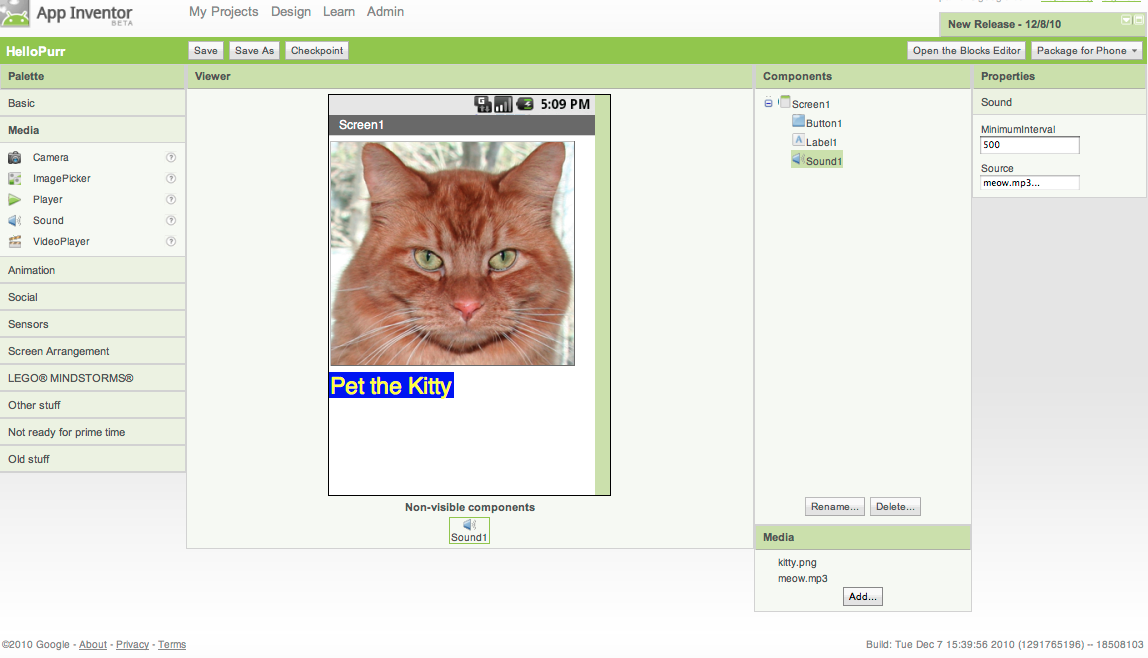
Under *Media*

1. Click Add...
2. Upload the meow.mp3 file to this project.

Under *Properties*

1. For the Sound1 component, set its Source to meow.mp3.

The Designer should now look like this:



**Making the sound play**

Using the Blocks Editor we will define how the app is going to behave. We'll tell the components what to do, and when to do it. You're going to tell the kitty button to play a sound when the user taps it. If components are ingredients in a recipe, you can think of blocks as the cooking instructions.

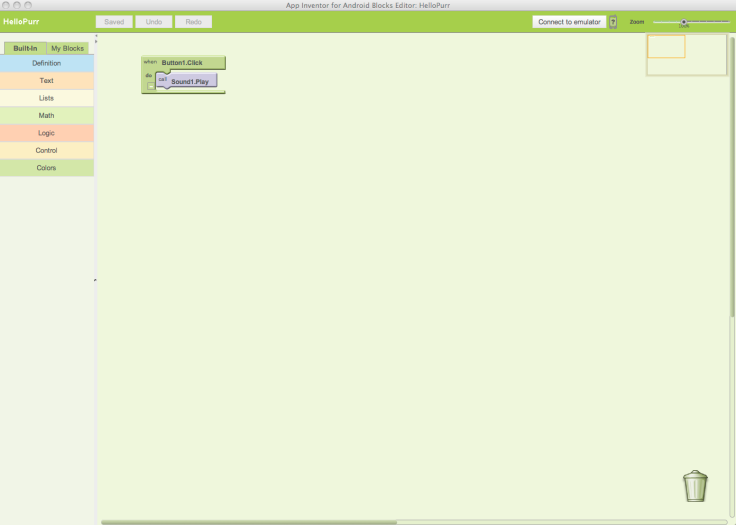
The Blocks Editor has two tabs on the upper left hand corner: Built-in and My Blocks . The buttons under each tab expand and expose blocks when clicked. The Built-in blocks are the standard set of blocks that are available for every app you build. The blocks under My Blocks contain specific blocks that are tied to the set of components that you have chosen for your app.

To make the sound play, you'll need to drag and drop the Button1.Click block and the Sound1.Play block into the editor. The blocks connect together like puzzle pieces.

**Steps for making the sound play**

1. Navigate to the *Blocks Editor* . It may be covered up by the web browser.
2. Click the My Blocks tab at the top left hand side.
3. Click Button1 .
4. Drag and drop the when Button1.Click do block onto the editor.
5. Click Sound1 .
6. Drag the call Sound1.Play block onto the editor and drop it into the when Button1.Click do block.
7. Click the kitty picture on your device. You should hear the kitty meow.

Your Blocks Editor should look like this:



**Packaging your app**

Congratulations, your first app is running! If you're using a phone, then the app is running on the phone, but it runs only while the phone is connected to App Inventor. If you unplug the USB cord, the app will vanish. You can reconnect the phone to make it come back. To get an that app runs without being connected to App Inventor, you must "package" the app to produce an *application package* (apk file). Pressing Package for Phone at the upper right of the designer page will present three options for packaging:

1. If the phone is connected, you can download and automatically install the app on your phone.
2. You can download the app to your computer as an apk file, which you can distribute and share as you like, and manually install on phones using the Android ADB program.
3. You can generate a Barcode, which you can use to install the app on your phone with the aid of a barcode scanner, like the ZXing barcode scanner (freely available in the Android Market). This barcode works only for your own phone. If you want to share the app with others, you'll need to download the apk file to your computer and share the file.

**Review**

Here are the key ideas covered so far:

* You build apps by selecting components (ingredients) and then telling them what to do and when to do it.
* You use the *Designer* to select components. Some components are visible and some aren't.
* You can add media (sounds and images) to apps by uploading them from your computer.
* You use the Blocks Editor to assemble blocks that define the components' behavior
* when ... do ... blocks define *event handlers* , that tell components what to do *when* something happens.
* call ... blocks tell components to do things.

**3.0 Building the WildLifeShow App** (http://www.appinventorblocks.com/)

|  |  |
| --- | --- |
| The concept is, we will have three buttons on the app. Upon a button click, we will display a corresponding image and fire up a specific sound. Here's a snapshot of the app we are about to make- |  |

|  |  |
| --- | --- |
| On your browser, go to [My Projects](http://beta.appinventor.mit.edu/). Click the New button to create our **WildLifeShow** app. This will make a new dialog box to open. |  |
| Enter the name of our app in the Project name box. Press OK |  |

|  |  |  |
| --- | --- | --- |
| Now drag a [VerticalArrangement](http://beta.appinventor.mit.edu/learn/reference/components/screenarrangement.html#VerticalArrangement) component on *Screen1* window where we will display our images later. This will create a vertical arrangement component on our *Screen1*. The name of the component will be automatically *VerticalArrangement1* |  | |
| All the components you drag to *Screen1* can be seen under *Components* pane. We need to change some properties of the *VerticalArrangement1*. Select *VerticalArrangement1* under *Components* pane. On *Properties* pane you will see the properties of the component you selected. Change *VerticalArrangement1'*s *Width* to *Fill parent* and *Height* to *350 pixels* | | |  | |

    .

|  |  |
| --- | --- |
| Drag an [Image](http://beta.appinventor.mit.edu/learn/reference/components/basic.html#Image) component and place it inside *VerticalArrangement1* component that we placed earlier on *Screen1*. Note that the name of *Image* component is *Image1*. |  |

|  |  |
| --- | --- |
| Get a [Label](http://beta.appinventor.mit.edu/learn/reference/components/basic.html#Label) component and place it below *VerticalArrangement1*. Take note of the name of our *Label* component which is automatically set to *Label1*. |  |

|  |  |
| --- | --- |
| We need to change some properties of the *Label* component. Select *Label1* under Components pane and change the following-   * Check the FontBold property. * In the Text property, type *Tap On A Button To Hear Them*. You can type anything you like. * Change the TextAlignment to center. * Set TextColor to Blue. * Change the Width to Fill parent. |  |

|  |  |
| --- | --- |
| Now we need a [HorizontalArrangement](http://beta.appinventor.mit.edu/learn/reference/components/screenarrangement.html#HorizontalArrangement) component where we will place our three buttons. Get a *HorizontalArrangement* component and place it below our *Label* component. Take note of the name *HorizontalArrangement1*. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Inside the *HorizontalArrangement1*, place three [Buttons](http://beta.appinventor.mit.edu/learn/reference/components/basic.html#Button) one after another. |  | | |
| Now change the properties of the buttons as follows:   * Select *Button1* and change its *Text* property to *Elephant*, *Width* to *Fill parent*, *TextAlignment* to *Center*, and have *FontBold* property checked. * Now, select *Button2*and change its *Text* property to *Bird*, *Width* to *Fill parent*, *TextAlignment* to *Center*, and have*FontBold* property checked. * Lastly, select *Button3*and change its *Text* property to *Penguin*, *Width* to *Fill parent*, *TextAlignment* to *Center*, and have*FontBold* property checked. | |  |

We also need to play sound. For us to be able to do that, drag a [Player](http://beta.appinventor.mit.edu/learn/reference/components/media.html#Player) component and place it anywhere on *Screen1*. Since *Player* is a non-visible component, this will appear underneath *Screen1* window.

|  |  |
| --- | --- |
| One more tiny thing we should do is change the *Title* of *Screen1* to *WildLifeShow. For us to do that,* select *Screen1* on *Components* pane and change the *Title* property under *Properties* pane to *WildLifeShow*  . |  |

    Let's get our picture and sound files added. For our app, we used the following image and sound files which you should download to your computer.

* [Elephant.png](http://www.appinventorblocks.com/Elephant.png?attredirects=0)
* [Bird.png](http://www.appinventorblocks.com/Bird.png?attredirects=0)
* [Penguin.png](http://www.appinventorblocks.com/Penguin.png?attredirects=0)
* [Elephant.wav](http://www.appinventorblocks.com/Elephant.wav?attredirects=0)
* [Bird.wav](http://www.appinventorblocks.com/Bird.wav?attredirects=0)
* [Penguin.wav](http://www.appinventorblocks.com/Penguin.wav?attredirects=0)

|  |  |
| --- | --- |
| Under *Media* pane, click *Add* and upload the image and sound files one by one. |  |

|  |  |
| --- | --- |
| You should see now the following under your *Media* pane.  FOUND in http://www.appinventorblocks.com/ |  |

|  |  |
| --- | --- |
| Now that we are done with the design portion of our app, we need to implement the interactions among the components we have. For instance, we want the Elephant picture to be displayed and Elephant sound file to be played once *Button1* is clicked. In order to do that, we got to open the Blocks Editor window. Click on the *Open the Blocks Editor* button. |  |

|  |  |
| --- | --- |
| On the *Blocks Editor* window, click on *My Blocks* which will display all the components we placed on our app through *Designer/Viewer* window. Locate *Button1* and click on it. Now we can see all the events and different properties of *Button1*. Select and drag the *Button1.Click* event block on the *Blocks Editor* space as shown below. |  |

|  |  |
| --- | --- |
| Now under *My Blocks*, locate and click *Image1.* This will show the events and properties of *Image1*. Get the *Image1.Picture* and place it inside the *Button1.click* block. Once you place it, you should hear a click sound which means the blocks are properly connected to each other. |  |

|  |  |
| --- | --- |
| We need to set the source of the picture to our image *Elephant.png*. To do that, click on *Built-In* tab, then locate and click *Text* block. Drag the *text* block over to the right and connect it to *Image1.Picture*block. |  |

|  |  |
| --- | --- |
| The *text* block indicates which picture to display. Click on *text* of *text* block and change it to *Elephant.png*. Careful with the casing. If you have saved the *Elephant.png* file as *elephant.png*, then type *elephant.png*. |  |

|  |  |
| --- | --- |
| Now locate the *Player1* and place the *Player1.Source* block below *Image1.Picture* block. Connect a *text* block to *Player1.Source* block and change the *text* of *text* block to *Elephant.wav*. |  |

|  |  |
| --- | --- |
| Grab and place *Player1.Start* block below *Player1.Source* block. |  |

|  |  |
| --- | --- |
| Do the same thing for *Button2* and *Button3*. For *Button2*, use *Bird* and for *Button3,* use *Penguin*. Your blocks should look like the snapshot below. |  |

|  |  |
| --- | --- |
| Now it's time to see our app in action. If your Android device/phone is connected to your computer, you can click on *Package for Phone* on *Designer/Viewer* window in your web browser and then choose *Download to Connected Phone*. This will install the app on your phone |  |

.

|  |  |
| --- | --- |
| If you don't have a phone connected, you can launch an emulator from the *Blocks Editor* window. Click on *New emulator* and it will launch an emulator for you. |  |

|  |  |
| --- | --- |
| After the emulator is launched, click on *Connect to Device...* button. You should see the emulator that you just launched. Click on that and it will load the app on the emulator. |  |

# 

# PaintPot (Part 1)

This tutorial has two parts: [Part 1](http://appinventor.mit.edu/explore/content/paintpot-part-1.html) | [Part 2](http://appinventor.mit.edu/explore/content/paintpot-part-2.html)

This tutorial introduces the **Canvas** component for creating simple two-dimensional graphics. You'll build an app that lets you draw on the phone screen in different colors.

**Historical note**: PaintPot was one of the first programs developed to demonstrate the potential of personal computers, as far back as the 1970s.

[Download Book Chapter PDF](http://cs.usfca.edu/%7Ewolber/appinventor/bookSplits/ch2PaintPot.pdf)

#### What you're building



With the *PaintPot* app, you can:

* Dip your finger into a virtual paint pot to draw in that color.
* Drag your finger along the screen to draw a line.
* Tap the screen to make dots.
* Use the button at the bottom to wipe the screen clean.
* Include an image as a drawing background.

This tutorial assumes that you have completed the *HelloPurr* tutorial. It introduces the following App Inventor concepts:

* The **Canvas** component for drawing.
* Controlling screen layout with **Arrangement** components.
* Event handlers that take arguments.
* Variables.

#### Before starting

Make sure your computer and your phone are set up to use App Inventor. Start a new project in the Designer window, and name it *"PaintPot"*. Open the Blocks Editor, click **Connect to Phone**, and make sure the phone has started the App Inventor app.

#### Screen title

To get started, go to the Properties panel on the right of the Designer and change the screen **Title** to "PaintPot". You should see this change on phone, with the new title showing in the title bar.

There are three names in App Inventor, and it's easy to confuse them:

1. The name you choose for your project as you work on it (in this case, *PaintPot*). This will also be the name of the application if you package it for the phone.
2. The name "Screen1", which is the name of the **Screen** component. You'll see it listed in the Components panel in the Designer. You can't change the name of the Screen component in the current version of App Inventor.
3. The **Title** property of the screen, which is what you'll see in the phone's title bar. Title is a property of the **Screen** component. The Title starts out being "Screen1", which is what you used in *HelloPurr*. However, you can change it, as you're doing for *PaintPot*. To reiterate, the name and the title of Screen1 are initially the same, but you can change the title if you want.

#### Set up the Components

You'll use these components to make *PaintPot*:

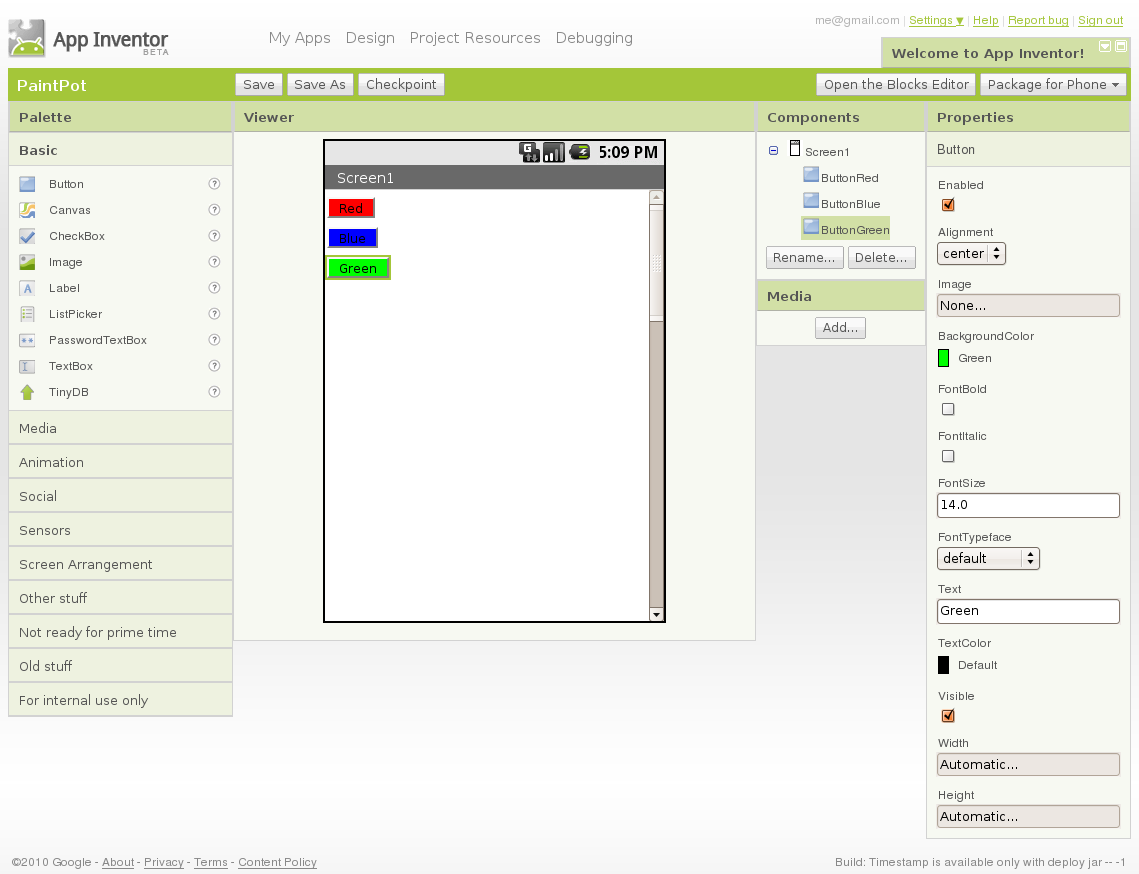
* Three **Buttons** for selecting red, blue, or green paint, and another button for wiping the drawing.
* A **Canvas**, the drawing surface. This canvas has a **BackgroundImage**, which is [this kitty](http://appinventor.mit.edu/explore/sites/all/files/helloPurr/kitty.png) from the *HelloPurr* tutorial. You can also draw on a blank canvas. That's just a canvas without a background image.
* There's also a component you don't see: you use a **HorizontalArrangement** to make the three color buttons line up.

That makes five components in all. Let's get them and build the app.

#### Color Buttons

* Drag a **Button** component onto the Viewer and change the button's **Text** attribute to "Red" and make its **BackgroundColor** red.
* Click on *Button1* in the components list in the Viewer to highlight it (it might already be highlighted) and use the **Rename...** button to change its name from "Button1" to "ButtonRed".
* Similarly, make two more buttons for blue and green, named "ButtonBlue" and "ButtonGreen", placing them vertically under the red button.

Here's how this should look in the designer, with the button names appearing in the list of project components. In this project, you're changing the names of the components rather than leaving them as the default names as you did with *HelloPurr*. Using meaningful names makes your projects more readable to yourself and others.



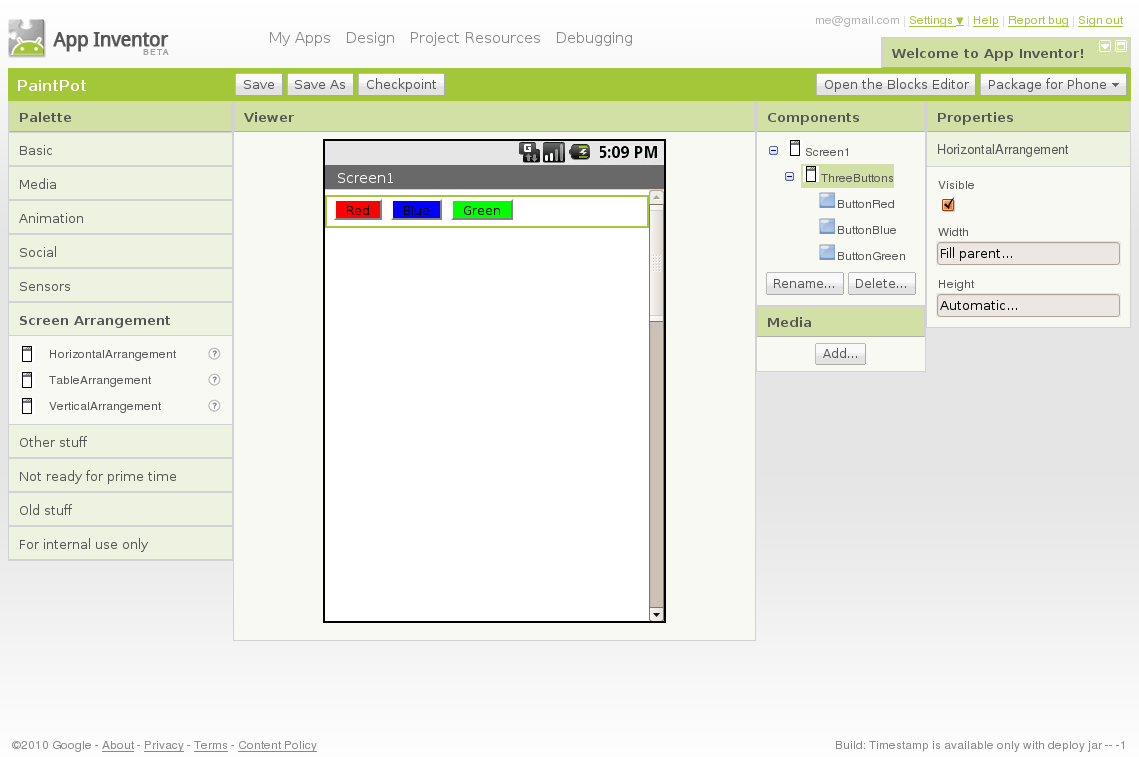
You should also see the three buttons on the phone screen.

#### Layout with Screen Arrangement

You should now have three buttons, one above the other. The next step is to make them line up horizontally. You do this using a **HorizontalArrangement** component.

1. From the Palette's Screen Arrangement category, drag out a **HorizontalArrangement** component and place it under the buttons. Change the name of this component from "HorizontalArrangement1" to "ThreeButtons".
2. In the Properties panel, change the **Width** of *ThreeButtons* to "Fill Parent..." so that it fills the entire width of the screen.
3. Move the three buttons side by side into the **HorizontalArrangement** component. **Hint**: You'll see a blue vertical line that shows where the piece you're dragging will go.

If you look in the list of project components, you'll see the three buttons indented under the *ThreeButtons* to show that they are now its subcomponents. Notice that all the components are indented under *Screen1*.



You should also see your three buttons line up in a row on the phone screen, although things might not look exactly as on the Designer. For example, the Arrangement's outline shows in the Designer but not on the phone.

In general, you use Screen Arrangement to create simple vertical or horizontal layouts. You can create more complex layouts by nesting Screen Arrangement components. There is also a **TableArrangement** component (not covered in this tutorial).

#### Canvas and wipe button

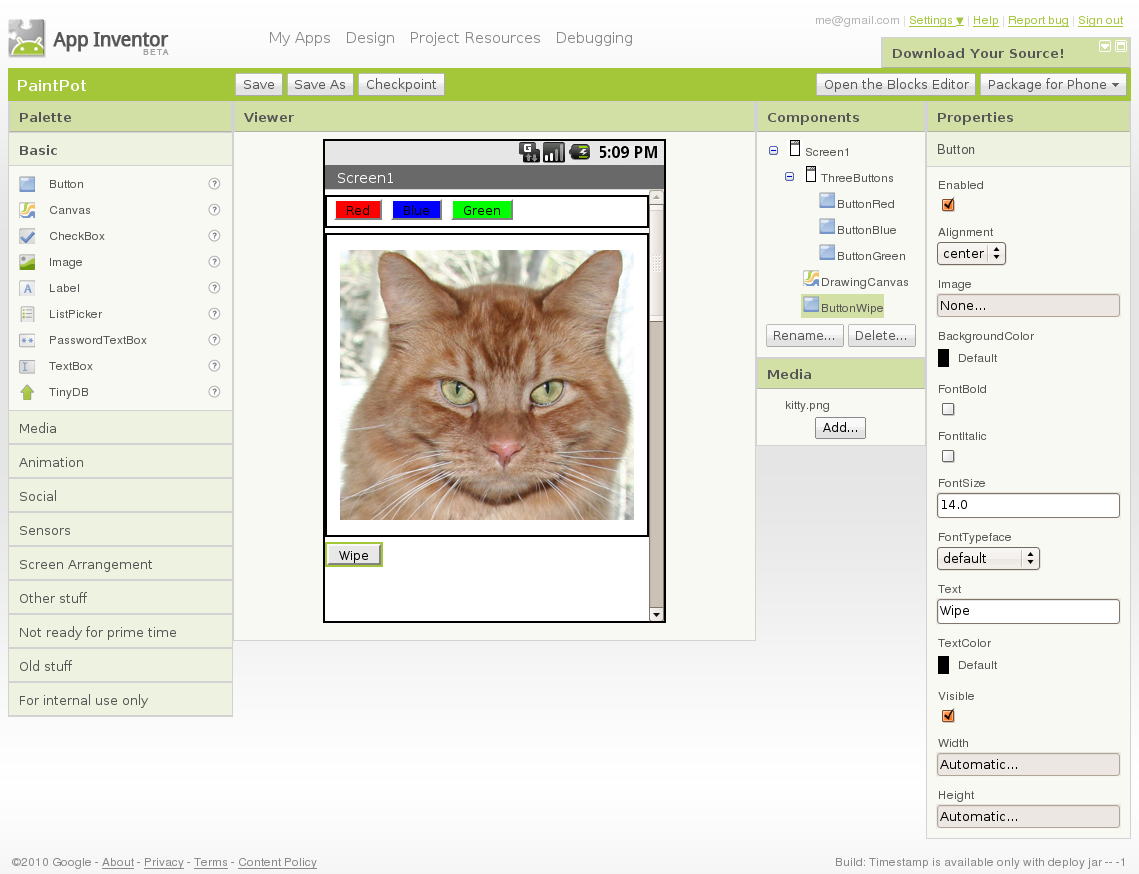
The final two components are the canvas and the wipe button.

1. From the Palette's Basic category drag a **Canvas** component onto the Viewer. Change its name to "DrawingCanvas". Set its **Width** to "Fill Parent" and set its **Height** to 300 pixels.
2. Add a Background Image to the **Canvas**. Click on the field containing "None..." next to **BackgroundImage** in the canvas's Properties panel. You can use the same kitty.png file, if you still have it on your desktop from an earlier tutorial. Or you can use another image.

You can use any image you like, but you'll get the best results if the size of the image (in pixels) is close to the size at which you'll be showing it on the phone. Also, large images will take a long time to load, and might exceed the memory capacity of the phone allocates for applications.

1. From the Palette, drag the final button onto the screen, placing it under the canvas. Change its **id** to "ButtonWipe" and change its **Text** attribute to "Wipe".

You've now completed the steps to set the appearance of your app. Here's how this should look in the Designer. Next, you'll define how the components behave.



#### Add behaviors to the components

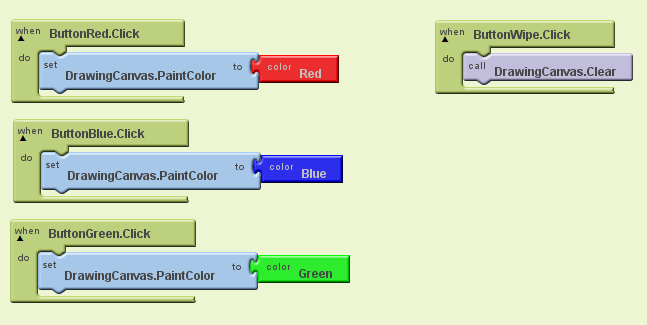
The Blocks Editor should already be open. First set up the buttons that change the paint color. Later you will add blocks to decide what happens when someone touches or drags the screen.

#### Add button event handlers

In the Blocks Editor:

1. Switch to the My Blocks column.
2. Open the drawer for *ButtonRed* and drag out the when ButtonRed.Click block.
3. Open the *DrawingCanvas* drawer. Drag out the set DrawingCanvas.PaintColor to block (you may have to scroll the list of blocks in the drawer to find it) and place it in the do section of when ButtonRed.Click .
4. Switch to the Built-In Column. Open the Colors drawer and drag out the block for the color Red and put it into set DrawingCanvas.PaintColor to .
5. Repeat steps 2-4 for the blue and green buttons.
6. The final button to set up is the Wipe button. Switch back to the My Blocks column. Make a click event handler for *ButtonWipe* by dragging when ButtonWipe.Click from the *ButtonWipe* drawer. From the *DrawingCanvas* drawer, drag call DrawingCanvas.Clear and place it in the do area of the when ButtonWipe.Click block.

The blocks for the buttons should look like this:



#### Add Touch-event Handlers

Now for the next step: drawing on the **Canvas**. You'll arrange things so that when you touch the canvas, you get a dot at the spot where you touch. If you drag your finger slowly along the canvas, it draws a line.

* In the Blocks Editor, open the drawer for the canvas and drag the when DrawingCanvas.Touched block to the workspace. As soon as you drag the block out, the three plugs on the right automatically fill in with name blocks called **x**, **y**, and **touchedSprite**.

You've already seen button click events. Clicks are simple, because there's nothing to know about the click other than that it happened. Other event handlers such as when ... Touched need information about the event. In App Inventor, this information is expressed as the value of arguments associated with the event handler. For the when ... Touched event, the first two arguments stand for the x and y coordinates of where the touch happened. We'll save **touchedSprite** for a later tutorial.

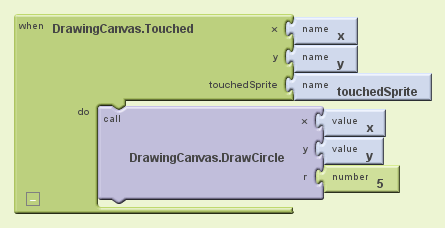
* For this touch event, make the canvas draw a small circle at the point with coordinates (x, y). Drag out a call DrawingCanvas.DrawCircle command from the canvas drawer and place it in the do section of when DrawingCanvas.Touched.

On the right side of the when DrawingCanvas.DrawCircle block are three sockets where you must specify values for the x and y coordinates where the circle should be drawn, and **r**, which is the radius of the circle. For **x** and **y**, you'll use values of the arguments that were supplied to the Touched handler:

1. Open the My Definitions drawer at the top of the column. Find the blocks for value x and value y. The blocks were automatically created when you dragged out the touch event handler block.
2. Drag out the value x and value y blocks and plug them into the corresponding sockets in the when DrawingCanvas.Touched block. Make sure to drag the value blocks, not the corresponding name blocks; they look very similar.
3. You'll also need to specify the radius of the circle to draw. Five (pixels) is a good value for this app. Click in a blank area of the screen to bring up the hover menu and select **math** (green). Select "123" from the dropdown list, to create a number block. Change the "123" to "5" and plug that in for the radius slot.

You can also just type 5 followed by return, to create a number block with a value of 5. This is an example of typeblocking: if you start typing, the Blocks Editor shows a list of blocks whose names match what you are typing; if you type a number it creates a number block.

Here's how the touch event handler should look:



Try out what you have so far on the phone. Touch a color button. Now touch the canvas, and your finger should leave a spot at each place you touch. Touching the Wipe button should clear your drawing.

#### Add Drag Events

Finally, add the drag event handler. Here's the difference between a touch and a drag:

* A touch is when you place your finger on the canvas and lift it without moving it.
* A drag is when you place your finger on the canvas and move your finger while keeping it in contact.

When you drag your finger across the screen, it appears to draw a giant, curved line where you moved your finger. What you're actually doing is drawing hundreds of tiny straight lines: each time you move your finger, even a little bit, you extend the line from your finger's immediate last position to its new position.

A drag event comes with 6 arguments. These are three pairs of x and y coordinates that show:

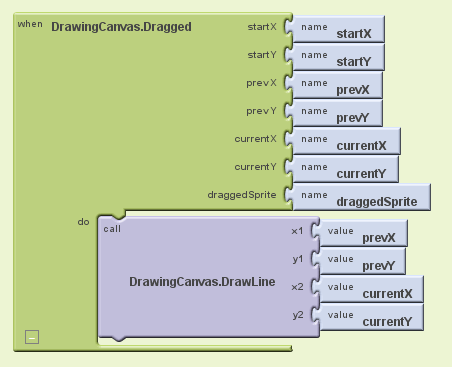
* The position of your finger back where the drag started.
* The current position of your finger.
* The immediately previous position of your finger.

There's also a sprite, which we'll ignore for this tutorial.

Now make dragging draw a line between the previous position and the current position by creating a drag handler:

1. From the *DrawingCanvas* drawer, drag the when DrawingCanvas.Dragged block to the workspace.
2. Also from the *DrawingCanvas* drawer, drag the call DrawingCanvas.DrawLine block into the do slot of the when DrawingCanvas.Dragged block.
3. Click on the My Definitions drawer. You should see the blocks for the arguments you need. Drag the corresponding value blocks to the appropriate slots in when DrawingCanvas.Dragged: **x1** and **y1** should be **prevX** and **prevY**; **x2** and **y2** should be **currentX** and **currentY**.

Here's the result:



Test your work by trying it on the phone: drag your finger around on the screen to draw lines and curves. Touch the screen to make spots. Use the Wipe button to clear the screen.

In [PaintPot Part 2](http://appinventor.mit.edu/explore/content/paintpot-part-2.html), you'll see how to use global variables to create dots of different sizes.

#### Review

Here are some of the ideas covered in this tutorial:

* You can use Screen Arrangement components to specify screen layouts other than just placing components one under the other.
* The **Canvas** component lets you draw on it. It can also sense touches and drags.
* Some event handlers are called with information about the event, such as the coordinates of where the screen was touched. This information is represented by arguments. When you select an event handler that has arguments, App Inventor creates value blocks for these and places them in the My Definitions drawer.

#### Scan the Sample App to your Phone

Scan the following barcode onto your phone to install and run the sample app.



#### Download Source Code

If you'd like to work with this sample in App Inventor, download the [source code](http://appinventor.mit.edu/explore/sites/all/files/paintPot1/PaintPot.zip) to your computer, then open App Inventor, go to the My Projects page, and choose **More Actions | Upload Source**.

## Attached File(s):

[PaintPot.apk](http://appinventor.mit.edu/explore/sites/explore.appinventor.mit.edu/files/PaintPot.apk)



[PaintPot.zip](http://appinventor.mit.edu/explore/sites/explore.appinventor.mit.edu/files/PaintPot.zip)



# Magic 8-ball

### Lesson One: Magic 8-Ball Predicts the Future



This introductory module will guide you through building a “Magic 8-Ball” app with App Inventor. When activated, your 8-ball will deliver one of its classic predictions, such as “It is decidedly so” or “Reply hazy, try again.”

### Learning Goals

After completing this app, you will be able to:

* Navigate the App Inventor environment: designer, blocks editor, emulator and/or physical phone
* Correctly use the following App Inventor components: accelerometer sensor, image, list-picker

### Materials

* A selection of images and sounds are available at the [App Inventor Media Library](http://appinventor.mit.edu/explore/media-library.html).
* Optional supporting hard copy materials such as the [App Inventor Overview Handouts](http://appinventor.mit.edu/explore/media-library.html), and the [PDF version](https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxhcHBpbnZlbnRvcmxlc3NvbnN8Z3g6MjJkNmViOGQ3MDNmZWYwZQ) of this lesson.
* Basic App Inventor tutorial - getting around [[video](http://www.youtube.com/watch?v=nC_x9iOby0g)]

### Outline

1. [Set up](http://appinventor.mit.edu/explore/content/setup-mit-app-inventor.html) computers and phones or emulators. (Suggestion: do this ahead of time)
2. Part One:[Click a Button, Hear a Sound](http://appinventor.mit.edu/explore/teach/magic-8-ball.html#m8one)
3. Part Two: [Click the Button, Get a Prediction + Hear a Sound](http://appinventor.mit.edu/explore/teach/magic-8-ball.html#m8two)
4. Part Three: [Shake the Phone, Get a Prediction + Hear a Sound](http://appinventor.mit.edu/explore/teach/magic-8-ball.html#m8three)
5. [Suggestions for further exploration:](http://appinventor.mit.edu/explore/teach/magic-8-ball.html#m8four) Text-to-Speech, Rotating image, Custom prediction lists

### Part One: Click a Button, Hear a Sound

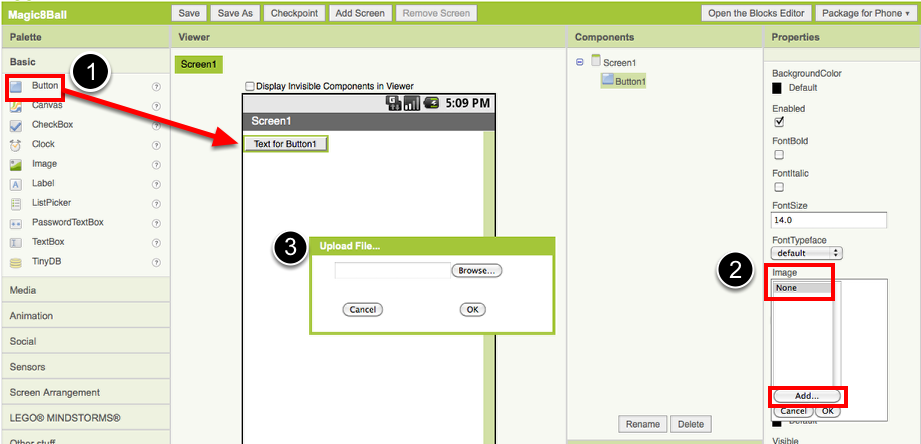
The final Magic 8-Ball App will deliver a prediction from a list that you have designed. To get started, first we'll make a button with a picture on it, and program it to play a sound when the button is clicked.

DESIGN: App Inventor Designer

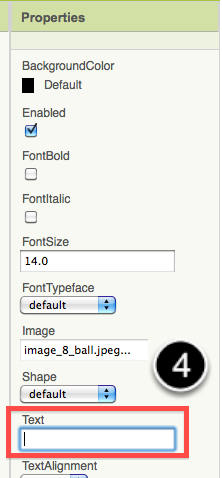
1. To open the App Inventor Designer window, go to http://appinventor.mit.edu and click "Invent". See setup instructions above if you are not sure how to sign in.
2. If you have already made an app (such as Hello Purr), you will automatically be directed to the Designer with the last project you worked on showing. Click "My Projects" in the upper left corner of the screen, which will take you to your list of projects. Click "New" and name your project something like "Magic8Ball"(note: spaces are not allowed).



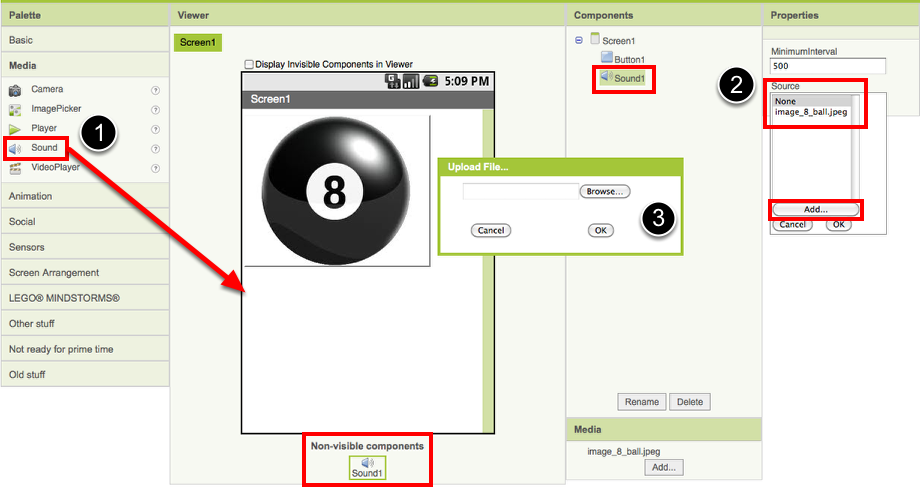
1. **Download one image and one sound file to be used in your app** from the [Media Library](http://appinventor.mit.edu/explore/media-library.html). Right click (control-click) on the link of the image or sound, then choose "Download" or "Save As". Save the media files to a convenient location that you will remember.
2. On the left column of the Designer, open the Basic palette, and drag a **Button** component over to the Viewer(#1).
3. **Set the button image to an 8-Ball image**:  
   Click on your newly added button to see its properties in the Properties pane on the right. Under "Image" click on the word "None..." and a small selection window will pop up (#2). Click the "Add" button and browse to where you saved the 8-Ball image. Select the file, then click “OK” to close the selection window. Click “OK” again on the properties pane to close the small popup window (#3).
4. Go to the text field in the Properties pane and delete the display text of your button component







1. From the Media palette, **drag over a Sound component** onto the Viewer pane (#1). Notice that since the sound will not be a visible part of the app, it appears at the bottom of the Viewer pane, as a “Non-visible component”.
2. **Set the sound component's source file:**  
   Click on your newly added sound component to see its properties in the Properties pane on the right. Under "Source" click in the small box on the word "None..." and a small selection window will pop up (#2). Click the "Add" button and browse to where you saved the sound file. Select the sound file, then click “OK” to close the selection window. Click “OK” again on the properties pane to close the small popup window (#3).



1. You have now completed the work in the Designer for Part One of this app. It's time now to go over to the Blocks Editor to program the behavior of these components.

BUILD: Blocks Editor

In the upper right corner of the Designer, click on the Blocks Editor button. Wait for a few moments while the blocks editor loads. This takes some time, and often requires you to click “accept”, “ok”, or “keep” as the java program downloads to your computer. (Be sure to look at the very top or very bottom of your browser to see if it is prompting you to accept.) If you are having trouble loading the Blocks Editor, go back to the [Setup Instructions](http://appinventor.mit.edu/explore/content/setup-mit-app-inventor.html) for help.

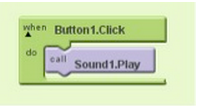
Now you are going to tell your app how to behave when the button is clicked. This is actually very simple in App Inventor, because the "code" for the program only consists of two blocks!

Once the Blocks Editor is open, there are several options running along the left side of the screen. We refer to these as "Palettes" with “Drawers.”

From the My Blocks palette, click on the Button1 drawer. Drag the when Button1.Click block into the work area (#1). From the My Blocks palette, click on the Sound1 drawer, drag the Sound1.Play block into the work area and insert it into the when Button1.Click block (#2). They will click together like magnetic puzzle pieces.



Your blocks should now look like this:



That's it! You've written the program for Part One of Magic 8-Ball. Now it's time to test that it's working right.

TEST: Phone/Emulator

You have now built an app! To test that it works, you either have to launch an emulator, or connect to a phone. Go back to the [Setup Instructions](http://appinventor.mit.edu/explore/content/setup-mit-app-inventor.html) if you do not have a phone or an emulator running.

**Emulator**: click on the picture, you will hear the sound play. **Phone**: tap the picture, you will hear the sound play.

Note: If you don't hear the sound, first be sure you have the volume turned up on your device (or computer if using emulator). Also, make sure your device has an SD card. App Inventor stores media files to the SD card. In some devices, the Play component does not work correctly. You will need to use the Player component instead of the Sound component.

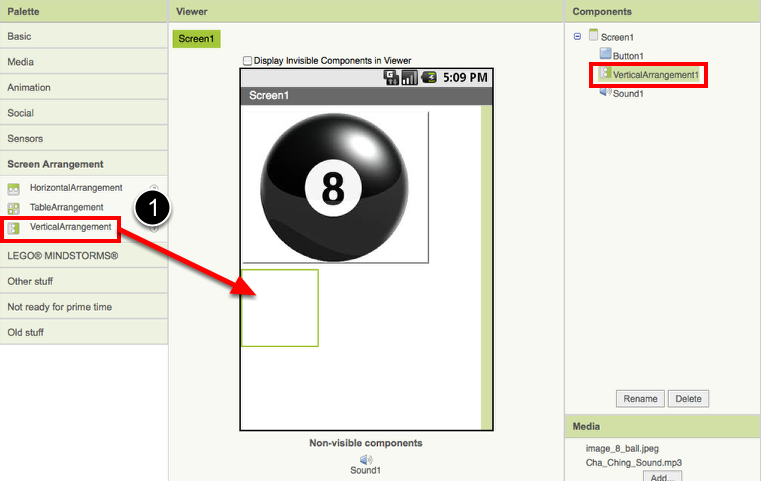
### Part Two: Output a Prediction

Now that we've gotten the button to perform an action (play a sound), we want to extend that action to include giving the user a prediction. First we'll need two labels: Label1 will display the instructions, and Label2 will display the chosen prediction. We'll use blocks to program a "list picker" to choose from a list of predictions. Each time the button is clicked, the app will change the text of Label2 to display the chosen prediction.

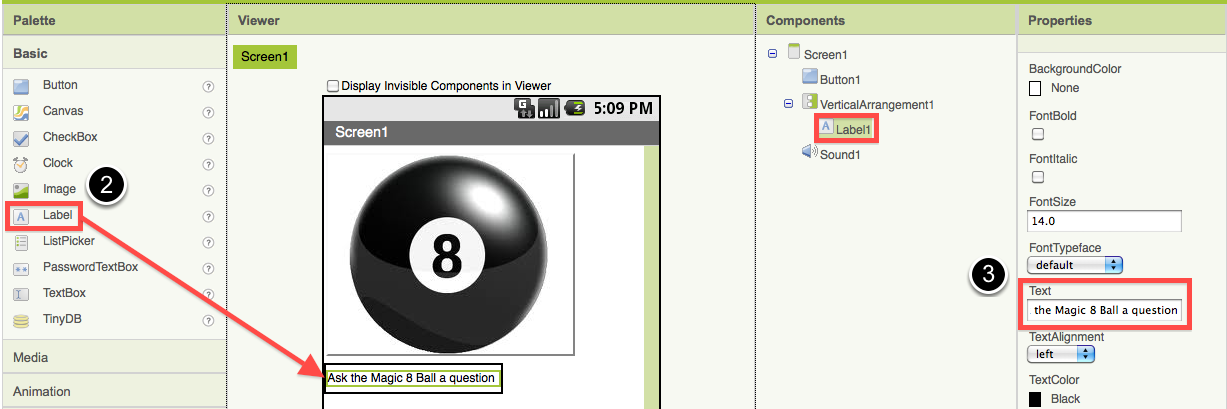
DESIGN: App Inventor

Go back to the Designer window in your browser and add some new things to your app.

1. From the Screen Arrangement palette, **drag over the Vertical Arrangement component** (#1). At first it will just look like an empty box, but when you put things in it, App Inventor will know that you want to line them up vertically (one on top of the other).



1. From the Basic palette, **drag over a Label component** (#2) and drop it inside of the vertical arrangement component. In the Properties pane, change the "Text" property of Label1 to “Ask the Magic 8-Ball a question”.(#3)



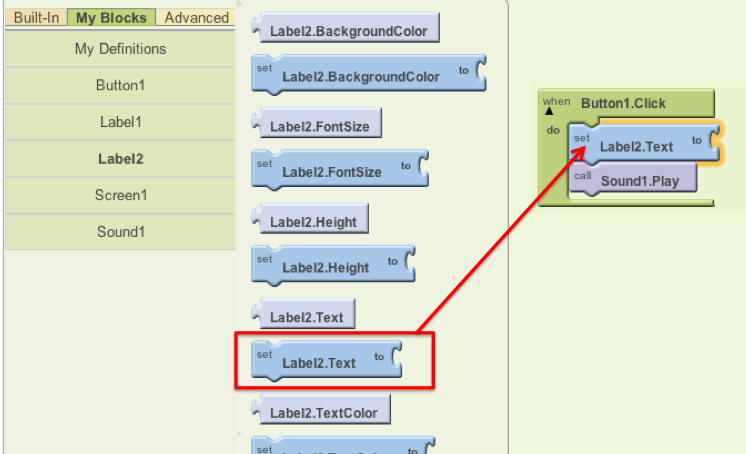
1. From the Basic palette, drag over another Label component (Label2) into the Vertical Arrangement box so that it sits right below Label1. Change the "Text" property of the Label2 to “Touch the Magic 8-Ball to receive your answer.” Now drag the 8-Ball image so that it is also inside the Vertical Arrangement component on top of the two labels. This will cause them to line up with each other in a vertical line. (Note: this can be tricky mouse work, but get them in there just right and the vertical arrangement will resize itself to fit everything.)

Now it’s time to go back into the Blocks Editor to program the components you just added to your project. (Remember, the Blocks Editor is running in a window outside of your web browser, signified by the java icon that looks like a coffee cup.)

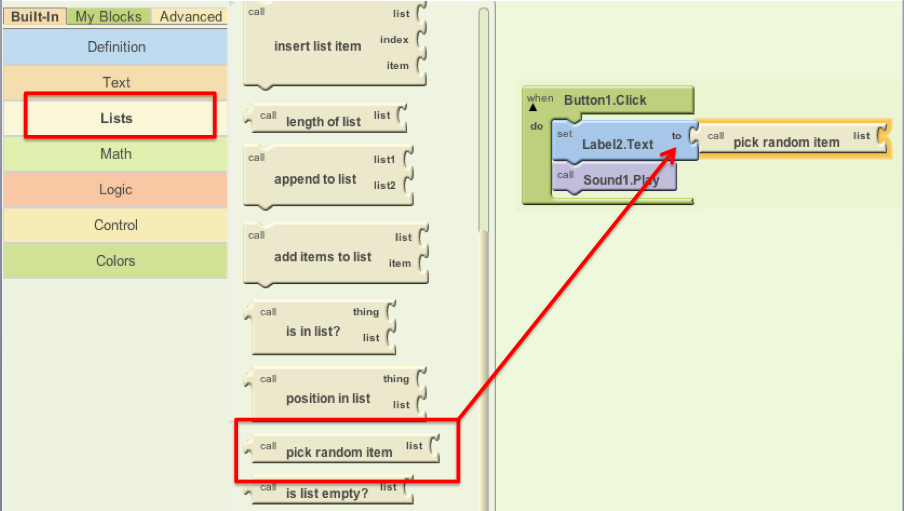
BUILD: Blocks Editor

Now for the fun part! You're going to make a list of predictions and program the button to pick one item from the list and display it inside Label2. The button will also still play the sound that you programmed in Part One. Here's how to do it...

1. From the My Blocks palette, click on Label2 drawer to see all of its associated blocks. Drag over the blue set Label2.Text and insert it just above the Sound1.Play block. Notice that the when Button1.Click block automatically gets bigger to accomodate the new block.



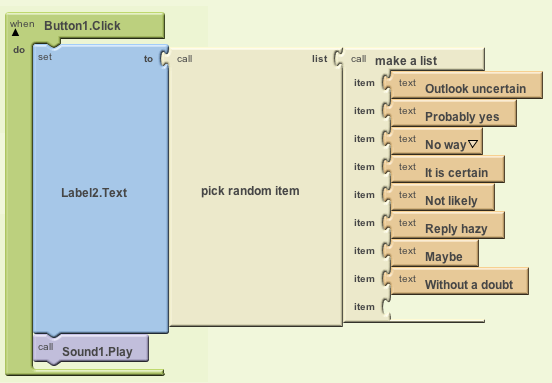
1. From the Built-In palette, click on the Lists drawer. Drag over the pick random item block and connect it underneath the set Label2.Text block (and above the Sound1.Play block).



1. From the Built-In palette, click on Lists again, then drag out the make a list block and plug it into the "list" socket on the right side of the pick random item block.
2. From the Built-In palette, click on the Text drawer, drag out a text block and connect it to the item socket of the make a list block. Click directly on the word “text” so that it gets highlighted. You can then type in new text there. Think about the sayings you want in your list of predictions for the Magic 8-Ball. Type the first prediction into this new text block.
3. Notice that when you plug in a new text block, the make a list block automatically creates a new socket. Repeat the previous step for each of the prediction choices you want programmed into your 8-Ball App. Plug each text block into the pick random item block. (Ideas for answers: <http://en.wikipedia.org/wiki/Magic_8-Ball>)

Blocks should look something like this:

(Note: it is normal for there to be a blank "item" space at the end of the make list block.)



You've got a Magic 8-Ball App! Now your app is fully functional and will predict the future with absolute certainty. Test out that this works, and then come back for some challenge tasks to make the app even more fun.

TEST: Emulator or Phone

**Emulator**: Click on the picture of the 8-Ball, you should see one of your answers displayed in the Label2.text field, followed by the sound. **Phone**: Tap on the picture of the 8-Ball, you should see one of your answers displayed in the Label2.text field, followed by the sound.

### Part Three: Shake the Phone, Get a Prediction

Even though you have a working Magic 8-Ball app, there is a way to make it even more fun. You can use the accelerometer component to make the phone respond to shaking instead of responding to a button click. This will make the app much more like a real Magic 8-Ball toy. Note: This part can only be done with an actual phone or tablet equipped with an accelerometer. If you are using an emulator, skip this part and go to Challenge 1 instead.

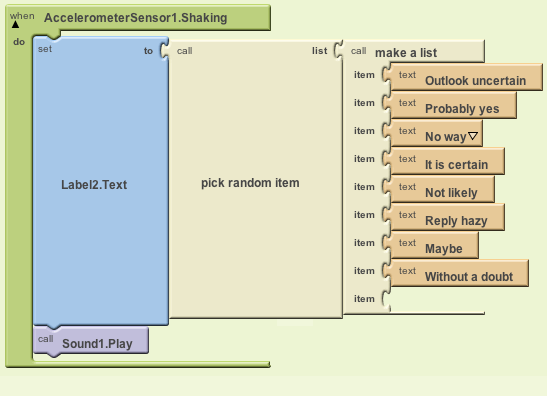
DESIGN: App Inventor

From the Sensors palette, drag over an **AccelerometerSensor** sensor component. Notice that it automatically drops down to the “Non-visible components” area of the Viewer window. This is the only new component you need, so go on over to the Blocks Editor to change your program.

BUILD: Blocks Editor

1. From the My Blocks drawer, click on AccelerometerSensor, then drag out the block for when AccelerometerSensor.Shaking.
2. Disconnect all of the blocks from inside the Button1.Click block and move them inside the AccelerometerSensor.Shaking block. **NOTE:** you can move whole sections of connected blocks by clicking on the uppermost or leftmost block and dragging it. The connected blocks will come with it.
3. Delete the Button1.Click block to keep your work area tidy.

The blocks should look something like this:



TEST: Phone/Emulator

**Phone**: When you shake the phone it should show an answer and play a sound. **Emulator**: unfortunately, you can not simulate shaking the phone when using the emulator.

Package the App to Your Phone!

Your app would disappear if you were to disconnect your phone from the Blocks Editor. This is because the app is still stored on the App Inventor server and not on your phone. Follow [these instructions](http://appinventor.mit.edu/explore/support/packaging-apps.html) to package your app to your phone or to make an ".apk" file that can be installed on any android phone. Or, if you want to make your app even cooler, try the challenges below.

### Challenge 1: Make the Magic 8-Ball Speak

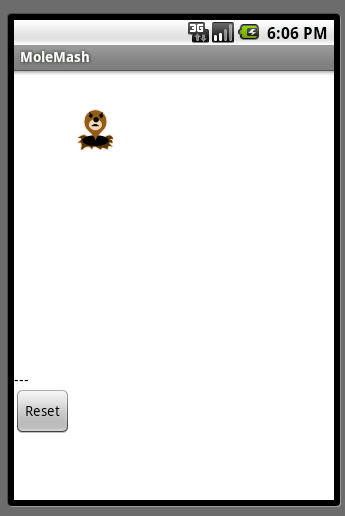
Instead of (or in addition to) making the prediction appear as text, can you make the 8-Ball speak it aloud? Hint: the text-to-speech component is under the **Other Stuff** palette in the Designer. Note: Most Android devices have the text-to-speech (TTS) capability, but if you have trouble getting the TTS component in App Inventor to work, you may need to find out how to install TTS and/or enable TTS on your device.

### Suggestions for Further Exploration

* Make the image rotate when the phone is shaken or have several images that the app rotates through while the phone is shaken. You could use this technique to make it look like the triangle piece inside the 8-ball window is surfacing. You could also make different images for different predictions and display the correct image for each prediction.
* Make a similar app but for a different purpose. The phone could be used in place of dice or yahtzee letters. It could simulate a coin toss or a random number or color generator for investigating probability.
* Ask end users to add choices to the list of predictions (See Make Quiz tutorial).
* "Crowd source" for prediction choices: allow people to send text messages and have the app add them to the list.
* Make the 8 Ball app a "server" so that anyone who sends a text to it will receive a text message prediction in return.
* Complete change the list to humorous choices (e.g. an app for teacher to use when a student has an excuse for not doing homework), or for useful purposes like randomly selecting a name from amongst people in the class.

Done with Magic 8-Ball? Return to [Curriculum (Teacher Resources)](http://appinventor.mit.edu/explore/teach.html) or [Tutorials](http://appinventor.mit.edu/explore/tutorials.html).

# 4.0 MoleMash Game



## What You're Building

In the game MoleMash, a mole pops up at random positions on a playing field, and the player scores points by hitting the mole before it jumps away. This tutorial shows how to build MoleMash as an example of a simple game that uses animation.

## Getting Started

Connect to the App Inventor web site and start a new project. Name it MoleMash, and also set the screen's Title to MoleMash. Open the Blocks Editor and connect to the phone.

Also download this [picture of a mole](http://beta.appinventor.mit.edu/learn/tutorials/molemash/MoleMashAssets/mole.png) and save it on your computer.

## Introduction

You'll design the game so that the mole moves once every half-second. If it is touched, the score increases by one, and the phone vibrates. Pressing restart resets the score to zero.

This tutorial introduces:

|  |  |
| --- | --- |
| * image sprites * timers and the Clock component * procedures | * picking random numbers between 0 and 1 * text blocks * typeblocking |

## The first components

Several components should be familiar from previous tutorials:

* A Canvas named **MyCanvas** . This is the area where the mole moves.
* A Label named **ScoreLabel** that shows the score, i.e., the number of times the player has hit the mole.
* A Button named **ResetButton**

Drag these components from the Palette onto the Viewer and assign their names. Put MyCanvas on top and **set its dimensions to 300 pixels wide by 300 pixels high**. Set the **Text of ScoreLabel to Score: ---** . Set the Text of ResetButton to Reset . Also **add a Sound component and name it Noise** . You'll use Noise to make the phone vibrate when the mole is hit, similar to the way you made the kitty purr in HelloPurr .

## Timers and the Clock component

You need to arrange for the mole to jump periodically, and you'll do this with the aid of a Clock component. The Clock component provides various operations dealing with time, like telling you what the date is. Here, you'll use the component as a timer that fires at regular internals. The firing interval is determined by the Clock 's TimerInterval property. **Drag out a Clock component; it will go into the non-visible components area. Name it MoleTimer** . Set its TimeInterval to 500 milliseconds to make the mole move every **half second**. Make sure that Enabled is checked.

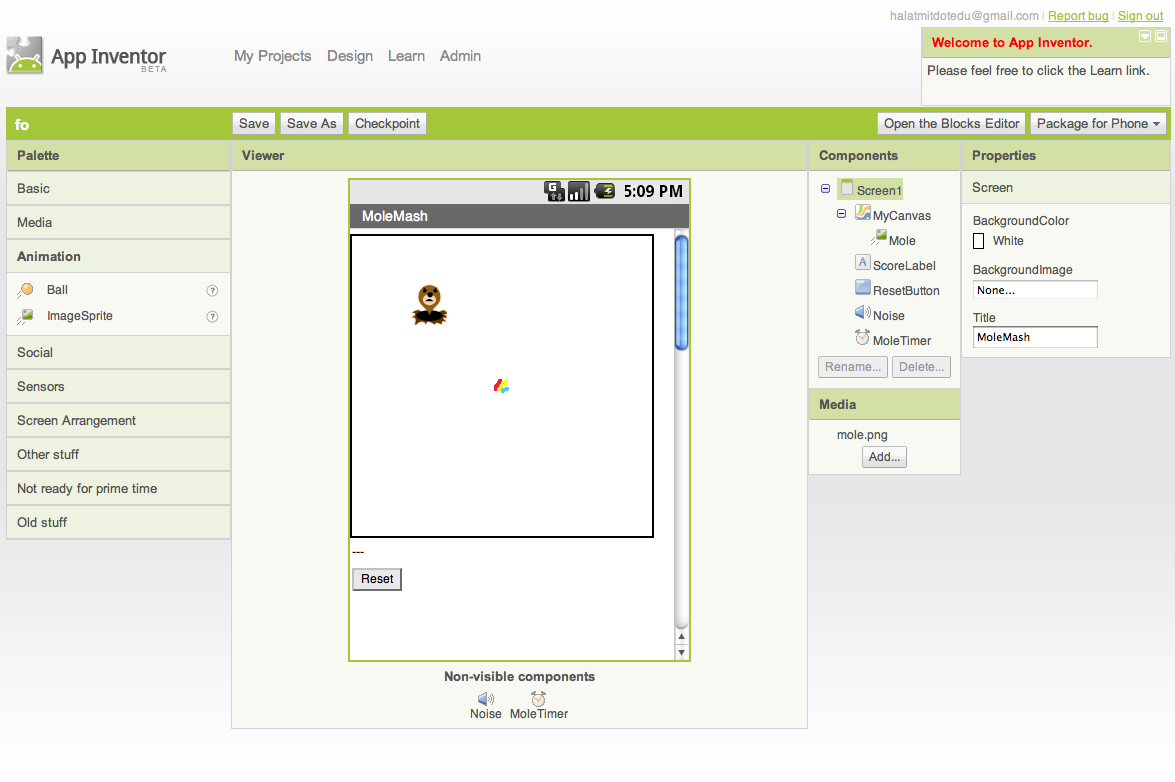
## Adding an Image Sprite

To add the moving mole we'll use a sprite .

Sprites are images that can move on the screen within a Canvas . Each sprite has a speed and a heading, and also an interval that determines how often the sprite moves at its designated speed. Sprites can also detect when they are touched. In MoleMash , the mole has a speed zero, so it won't move by itself. Instead, you'll be setting the mole's position each time the timer fires. **Drag an ImageSprite component onto the Viewer** . You'll find this component in the Animation category of the Palette . Place it within MyCanvas area. Set these properties for the Mole sprite:

* **Picture : Use mole.png, which you downloaded to your computer at the beginning of this tutorial.**
* **Enabled : checked**
* **Interval : 500**
* **The interval doesn't matter here, because the mole's speed is zero: it's not moving by itself.**
* **Heading : 0 The heading doesn't matter here either, because the speed is 0.**
* **Speed : 0.0**
* **Visible : checked**
* **Width : Automatic**
* **Height : Automatic**

You should see the X and Y properties already filled in. They were determined by where you placed the mole when you dragged it onto MyCanvas . Go ahead and drag the mole some more. You should see X and Y change. You should also see the mole on your connected phone, and the mole moving around on the phone you drag it around in the Designer . You've now specified all the components. The Designer should look like this. Notice how Mole is indented under MyCanvas in the component structure list, indicating that the sprite is a sub-component of the canvas.



## Component Behavior and Event Handlers

Now you'll specify the component behavior. This introduces some new App Inventor ideas. The first is the idea of a procedure .

A procedure is a sequence of statements that you can refer to all at once as single command. If you have a sequence that you need to use more than once in a program, you can define that as a procedure, and then you don't have to repeat the sequence each time you use it. Procedures in App Inventor can take arguments and return values. This tutorial covers only the simplest case: procedures that take no arguments and return no values.

## Define Procedures

Define two procedures:

* MoveMole moves the Mole sprite to a new random position on the canvas.
* UpdateScore shows the score, by changing the text of the ScoreLabel

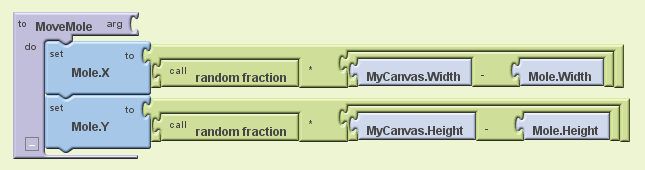
Start with MoveMole :

* In the Blocks Editor, under Built-In , open the Definition drawer. Drag out a to procedure block and change the label procedure to MoveMole .

Note: There are two similar blocks: procedure and procedureWithResult . Here you should use **procedure** .

The to MoveMole block has a slot labeled do . That's where you put the statements for the procedure. In this case there will be two statements: one to set the mole's x position and one to set its y position. In each case, you'll set the position to be a random fraction, between 0 and 1, of the difference between the size of the canvas and the size of the mole. You create that value using blocks for random -fraction and multiplication and subtraction. You can find these in the Math drawer.

* Build the MoveMole procedure. The completed definition should look like this:



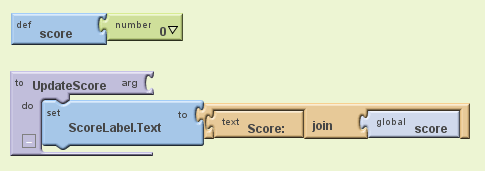
Leave the arg socket for MoveMole empty because MoveMole does not take any arguments. Observe how the blocks connect together: The first statement uses the Mole.X gets block to set mole's horizontal position. The value plugged into the block's socket is the result of multiplying:

* 1. the result of the call random-fraction block, which a value between 0 and 1
  2. the result of subtracting the mole's width from the canvas's width.

The vertical position is handled similarly.

With MoveMole done, the next step is to define a variable called score to hold the score (number of hits) and give it initial value 0. Also define a procedure UpdateScore that shows the score in ScoreLabel . The actual contents to be shown in ScoreLabel will be the text "Score: " joined to the value of the score .

* To create the "Score: " part of the label, drag out a text block from the Text drawer. Change the block to read "Score: " rather than "text".
* Use a join block to attach this to a block that gives the value of the score variable. You can find the join block in the Text drawer.
* Here's how score and UpdateScore should look:



## Add a Timer

The next step is to make the mole keep moving. Here's where you'll use MoleTimer . Clock components have an event handler called when ... Timer that triggers repeatedly at a rate determined by the TimerInterval .

Set up MoleTimer to call MoveMole each time the timer fires, by building the event handler like this:



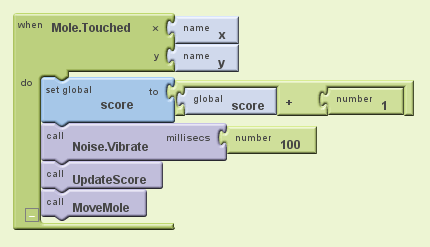
Notice how the mole starts jumping around on the phone as soon as you define the event handler. This is an example of how things in App Inventor start happening instantaneously, as soon as you define them.

## Add a Mole Touch Handler

The program should increment the score each time the mole is touched. Sprites, like canvases, respond to touch events. So create a touch event handler for Mole that:

1. Increments the score.
2. Calls UpdateScore to show the new score.
3. Makes the phone vibrate for 1/10 second (100 milliseconds).
4. Calls MoveMole so that the mole moves right away, rather than waiting for the timer.

Here's what this looks like in blocks. Go ahead and assemble the Mole.Touched blocks as shown.



Here's a tip: You can use typeblocking: typing to quickly create blocks.

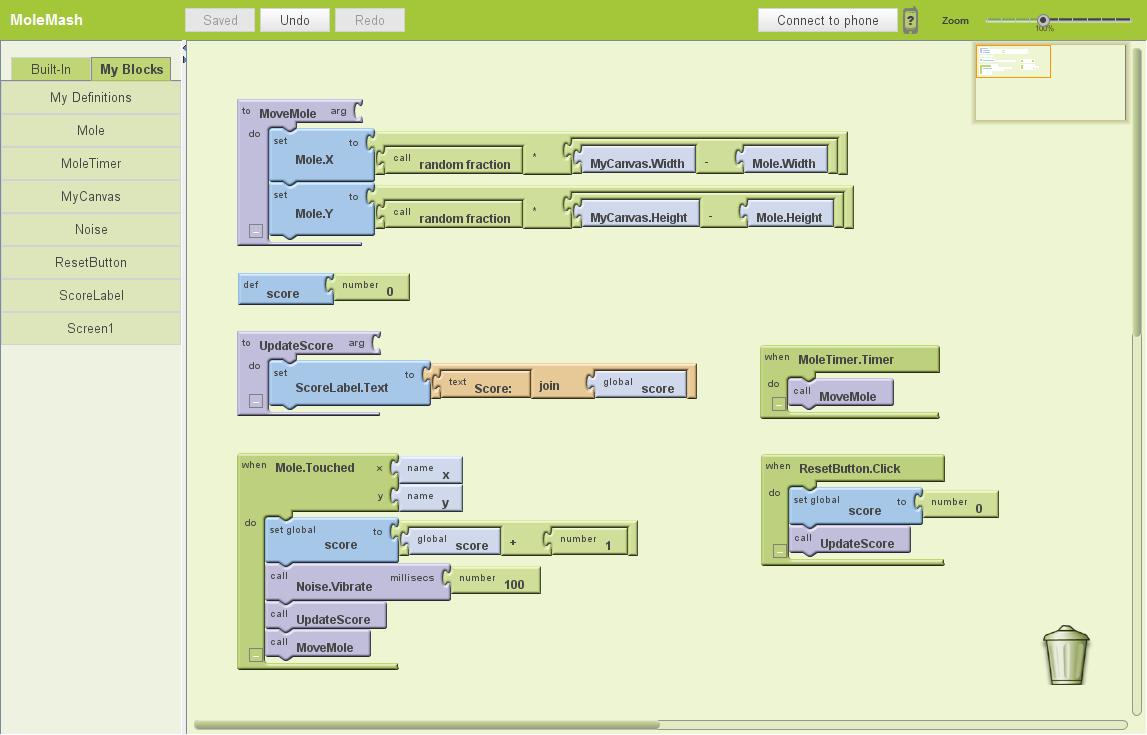
* To create a value block containing 100, just type 100 and press return.
* To create a MoveMole block, just type MoveMole and select the block you want from the list

## Reset the Score

One final detail is resetting the score. That's simply a matter of making the Reset button change the score to 0 and calling UpdateScore.

## Complete Program

Here's the complete MoleMash program:



## Variations

Once you get the game working, you might want to explore some variations. For example:

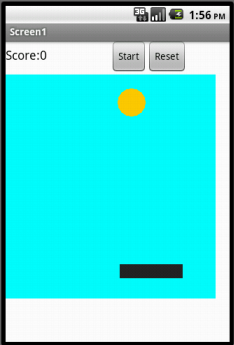
* Make the game vary the speed of the mole in response to how well the player is doing. To vary how quickly the mole moves, you'll need to change the MoleTimer's Interval property.
* Keep track of when the player hits the mole and when the player misses the mole, and show a score with both hits and misses. To do this, you'll need do define touched handlers both for Mole , same as now, and for MyCanvas . One subtle issue, if the player touches the mole, does that also count as a touch for the Canvas? The answer is yes. Both touch events will register.

## Review

Here are some of the ideas covered in this project:

* Sprites are touch-sensitive shapes that you can program to move around on a Canvas.
* The Clock component can be used as a time to make events that happen at regular intervals.
* Procedures are defined using to blocks.
* For each procedure you define, App Inventor automatically creates an associated call block and places it in the My Definitions drawer.
* calling random-fraction produces a number between 0 and 1
* text blocks specify literal text, similar to the way that number blocks specify literal numbers
* typeblocking is a way to create blocks quickly, by typing a block's name

# Pong



Pong is a simple game, which consists of a paddle (which is controlled by the user) and a ball. The ball bounces off the paddle and three walls. If the ball hits the paddle, the user gains points; if the ball hits behind the paddle, the game is over.

### Learning Goals

Completing this app will help you learn about:

* App Inventor environment: designer, blocks editor, emulator and/or physical phone
* App Inventor components: canvas, buttons, labels, animation sprites, procedures with no parameters, global variables, and conditionals

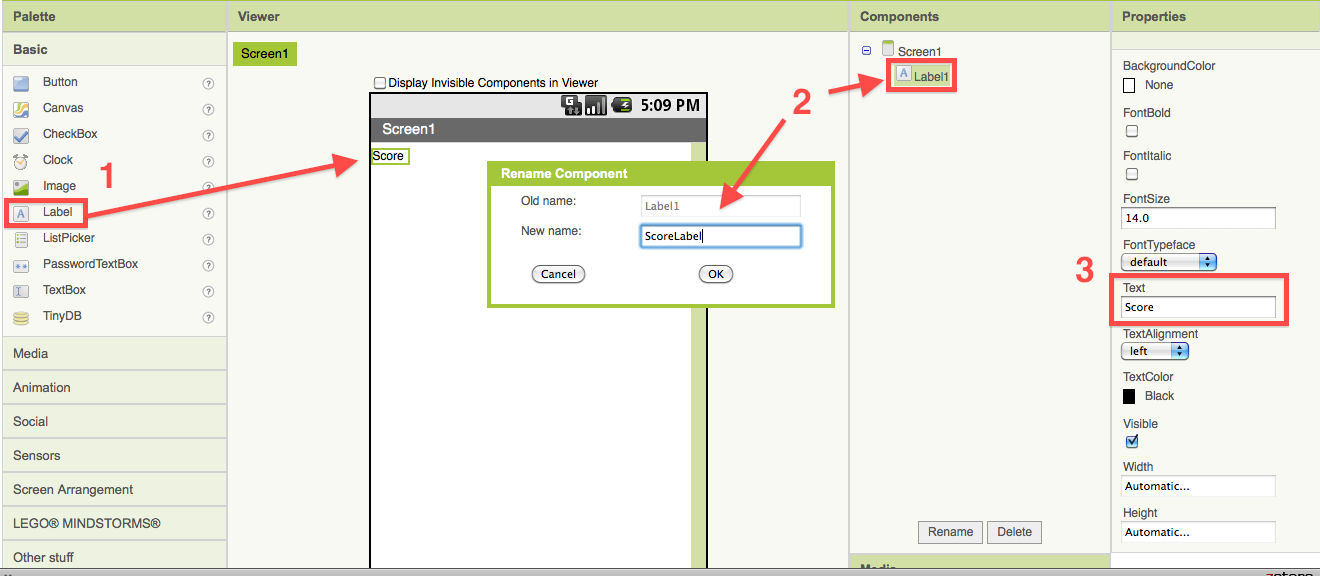
### Materials

* a selection of images and sounds in the [App Inventor Media Library](http://appinventor.mit.edu/explore/media-library.html)
* [supporting hard copy materials](http://appinventor.mit.edu/explore/resources/visual-pong-tutorial.html)
* [video snippets](http://www.valdosta.edu/~kroy/ai-tutorials.html) to assist students through this lesson (credit: K.Roy)

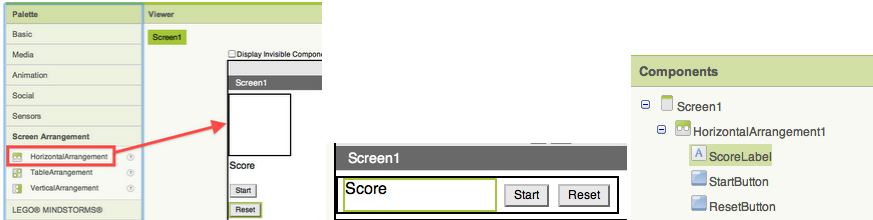
### Lesson Details

DESIGN: App Inventor Designer

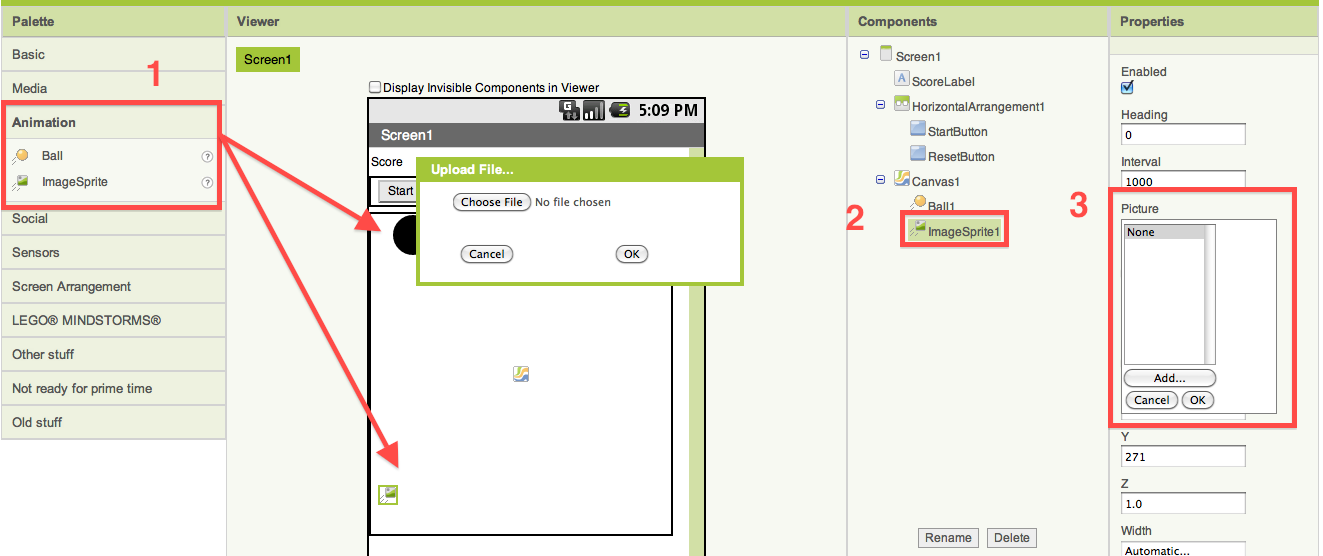
1. To open the App Inventor Designer window, go to [http://appinventor.mit.edu](http://appinventor.mit.edu/) and sign in with a Google ID (Gmail account).
2. Download and save an image of a paddle and a ding sound file from the [Media Library](http://appinventor.mit.edu/explore/media-library.html)
3. On the left column of the Designer, open the Basic palette, and first drag a Label component over to the Viewer (#1). Under the Components pane, highlight the Label1 component and change its name to "ScoreLabel" (#2). Under the Properties pane (right hand side), delete the display text of Label1 component to replace with “Score" (#3) and change its font size to 18, its width to 150 pixels, and height to be 30 pixels.



1. From the Basic palette, drag two Buttons to the Viewer under the labels. Rename the first button to "StartButton" and change its text field to "Start." Rename the second button to "ResetButton" and change its text field to "Reset."
2. Under the Screen Arrangement palette, drag a Horizontal Arrangement component to the viewer. Drag the ScoreLabel into the Horizontal Arrangement first, then drag the two buttons into the Horizontal Arrangement component next to the ScoreLabel component (you can see this under the Components pane too).



1. From the Basic palette, drag a Canvas component and set the width to 300 and the height to 390 pixels under the Properties pane. You can change the background color of the canvas to any color you like.
2. Under the the Animation pane, drag a Ball and Image Sprite components onto the Canvas component in the Viewer(#1). Highlight the Image Sprite component (#2) and change its image to paddle.gif (#3). You can download the image [here](http://dl.dropbox.com/u/90819513/Paddle.gif) if you have not downloaded it yet.



1. Highlight the Ball1 in the Components pane and go to the Properties pane to set the heading to 30, interval to 50, radius to 20, and speed to 5. You can change the color of the ball (PaintColor) to any color you like.

BUILD: Blocks Editor

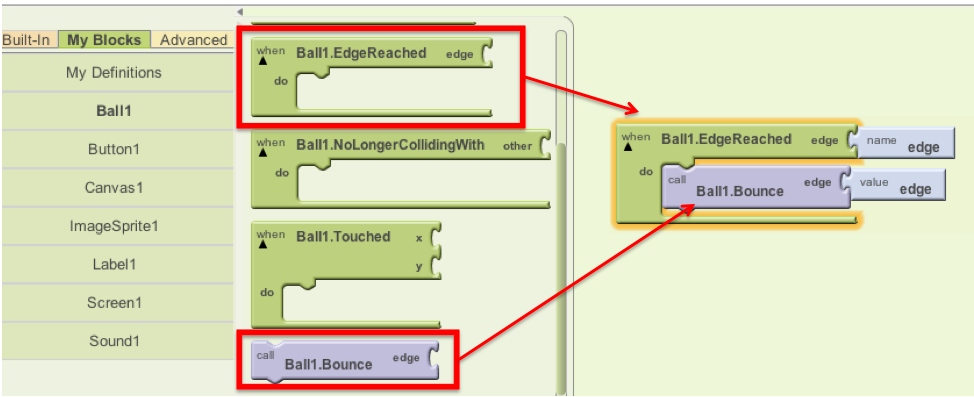
In the upper right corner of the Designer, click on the Blocks Editor button. Wait for a few moments while the blocks editor loads. This takes some time, and often requires you to click “accept”, “ok”, or “keep” as the java program downloads to your computer. If you are having trouble loading the Blocks Editor, visit [http://www.java.com](http://www.java.com/) to see if you have java installed, or to troubleshoot your installation. There is more information on launching the Blocks Editor in the [Setup Instructions](http://appinventor.mit.edu/explore/content/setup-mit-app-inventor.html).

Once the Blocks Editor is open, there are several options running along the left side of the screen.

1. From the My Blocks palette, open the Canvas1 drawer and drag the Canvas1.Touched block into the work area. This block automatically comes with three blocks: the name x, name y, and name touchedSprite blocks.
2. From the My Blocks palette, open the ImageSprite1 drawer and drag the set ImageSprite1.x block into the "do" socket of the Canvas1.Touched block.
3. From the My Blocks palette, open the My Definitions drawer and drag the value x block into the edge of the set ImageSprite1.x block. They will click together like magnetic puzzle pieces.

The value x and value y blocks were automatically added to your My Definitions drawer when you put the Canvas1.Touched block in your blocks editor work area. Whenver a touch is detected on Canvas1, these "variables" will be assigned the values of the x and y coordinates of the location of the touch.

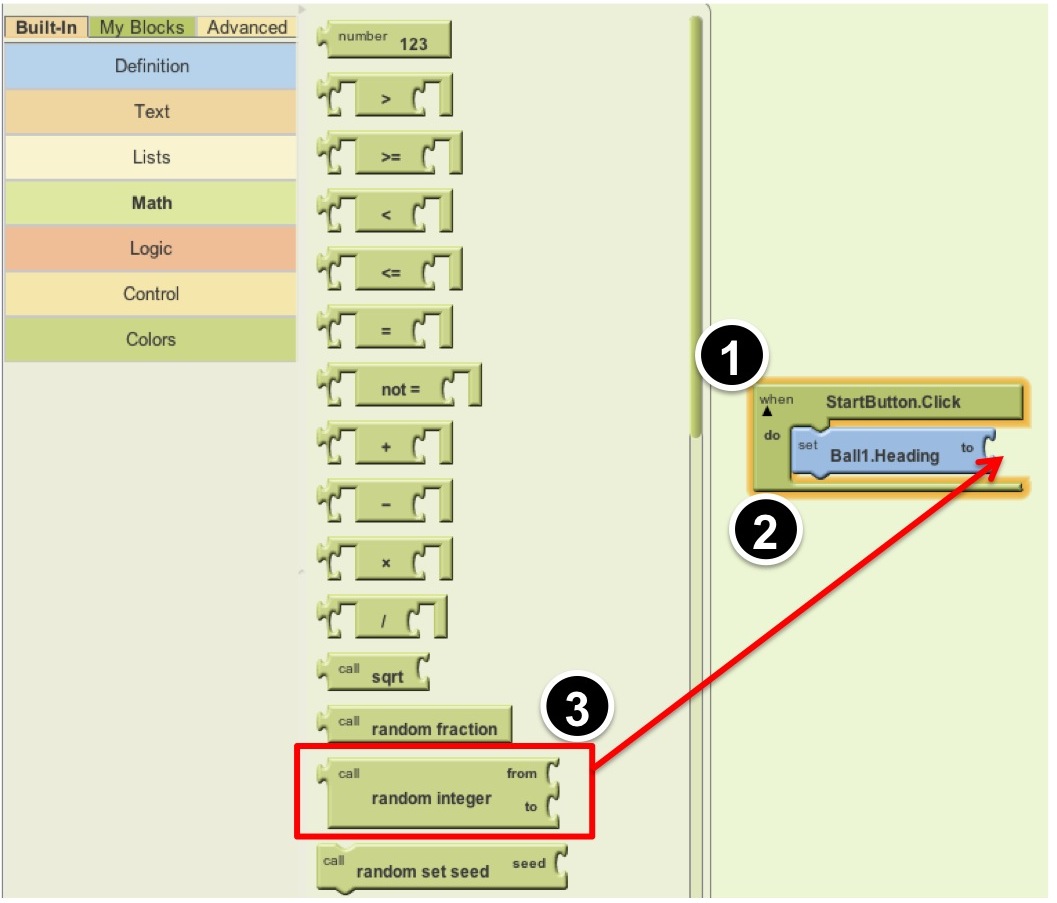
1. From the My Blocks palette, open the Ball1 drawer and drag the Ball1.EdgeReached block over the work area, as well as the Ball1.Bounce block. After this is done, open the My Definitions drawer and insert the edge block into the socket of the Ball1.Bounce block that is asking for an edge value. Bounce is a built-in function for sprites. It tells the sprite to bounce off the wall.



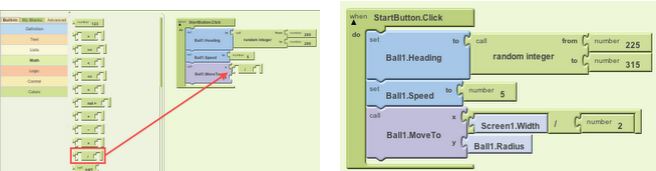
The Ball1.EdgeReached block automatically generates the edge variable under My Definitions. When the ball reaches an edge it needs a way to report that information to the main program. It does this by storing a number in the edge variable. See the image below to learn about App Inventor canvas edge values.

|  |  |
| --- | --- |
| App Inventor assigns numeric values to the **edges** of a canvas as follows: top = 1, right = 3, bottom = -1, left = -3  **Heading** values for animated objects go in a full circle like a compass, with values between 0 and 360 degrees. An object moving toward the top of the screen is said to have a heading of 90 degrees. |  |

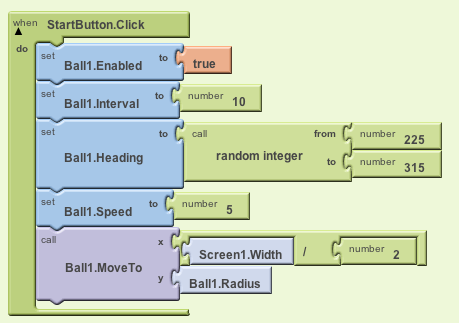
1. From the My Blocks palette, open the StartButton drawer and drag the StartButton.Click block over to the Viewer (#1). Click on the Ball1 drawer and drag out a set Ball1.Heading to block and place it at the "do" section of the StartButton.Click block (#2). From the Built-In palette's Math drawer, drag out the call random integer block and add it to the end of the Ball1.Heading block (#3).  
     
   Pull out a number 123 block from the Built-In palette's Math drawer. Plug it into the first to the "from" area and type "225" into the block in place of the "123." Do this again for the "to" area and type "315." This will make the ball take on a heading between 225 and 315 degrees when the start button is clicked (see heading diagram above).



1. From the Ball1 drawer, drag out a set Ball1.Speed to block and place it under the set Ball1.Heading to block. Put a number 123 block in the "to" socket and type in the value 5. This will move the ball 5 pixels in the direction of its heading each time its internal clock ticks. The "internal clock" of an object is called its interval. The Interval of a ball sprite is initially set at 1000 milliseconds (= 1 second) but you can change this in the properties of the sprite. Remember, at the beginning of this tutorial we changed the Interval property of Ball1 to 50ms.
2. From the Ball1 drawer, drag out a Ball1.MoveTo block and put it under the Ball1.Speed block. From the Math drawer, drag out a division ( / ) block and put that in the "x" socket in the Ball1.MoveTo block. From the My Blocks palette, open the Screen1 drawer and drag out a Screen1.Width block and drop that in the first blank area in the division block. Click on the second blank area and type "2" (a number 123 block appears with "2" in the text box). From the Ball1 drawer, drag out a Ball.Radius block and drop that into the "y" area of the Ball1.MoveTo block. This will start the ball in the middle of the screen near the top each time you hit the start button. (Do you understand why? The "MoveTo" block tells Ball1 to move to the x coordinate that is half the width of the screen, and the y coordinate that is equal to the radius of the ball itself -- which is a low number so the ball will appear very close to the top of the screen.)



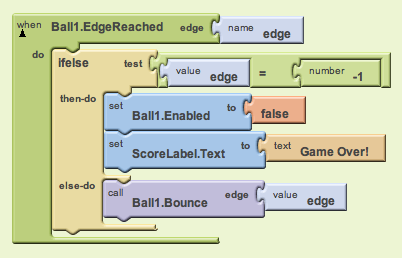
1. From the Ball1 drawer, drag out a set Ball1.Enabled to block. Click on the screen right next to the block and click on "Logic" from the small colored menu that pops up, then click on "true". This will start the ball moving.
2. From the Ball1 drawer, drag out a set Ball1.Interval to block and set it to the number 10. This will cause the ball to move every 10 milliseconds (and it will move the number of pixels specified in the "speed" property). After the interval block is created, move the set Ball1.Enabled and Ball1.Interval blocks onto the top of the Ball1.Heading block, so that they appear as the first blocks in the StartButton.Click event block.



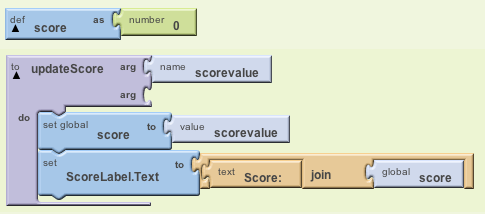
1. From the ImageSprite1 drawer, drag out a when ImageSprite1.Dragged block and drop it in the work area. From the same drawer, drag out a ImageSprite1.MoveTo and drop it in the when ImageSprite1.Dragged. Click on the My Definitions drawer, and drag out a currentX value block and drop it at the "x" slot. From the ImageSprite1 drawer, drag out a ImageSprite1.Y block and drop it at the "y" slot. This will move the paddle in x (horizontally) when you drag it, but not move it in y (vertically).
2. From the Ball1 drawer, drag out a when Ball1.CollidedWith block and drop it in an open area. Click on the word "other" in the name block and type "ImageSprite1" (you must type the name of your image sprite exactly).
3. From the Ball1 drawer, drag out a set Ball1.Heading to block and drop it inside the Ball1.CollidedWith block. Click on the Built-In palette and open the Math drawer, and drag out a subtraction block. Click in the first blank area and type "360". Click on the My Blocks palette, open the Ball1 drawer, and then drag out a Ball1.Heading block and drop it in the second blank area. This will reverse the ball when it hits the paddle. (Do you understand why? Look at the heading diagram above to see what happens when you subtract 360 from any heading. Headings must be between 0 and 360, and a negative number is treated as if it is positive.)
4. In your workspace, find the place where you put the when Ball1.EdgeReached block. We're going to add some new blocks to this event, so for now drag the Ball1.Bounce block away from Ball1.EdgeReached so that it becomes separated. Leave the Ball1.Bounce block loose in the work area for now.

Under the Built-In palette, open the Control drawer and drag out an ifelse block. From the Math drawer, drag out an equal (=) block and drop it into the "test" socket. Open the My Definitions drawer and drag out a value edge block and drop it in the first blank area. Click on the second blank area and type "-1" (for the bottom or southern edge). This will test to see if the edge that was reached is the bottom edge of the canvas.

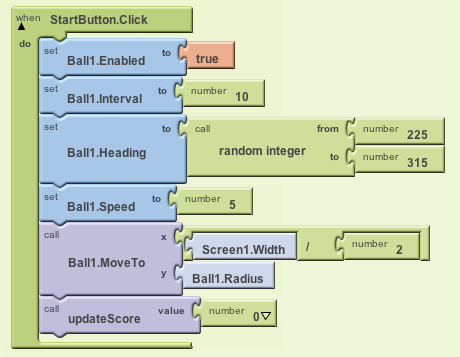
1. From the Ball1 drawer, drag out a set Ball1.Enabled to block and drop it in the "then-do" area of the ifelse block. Click to the right of that to get the popup menu, then click on "Logic" and choose "false". This will stop the ball from moving when it gets past the paddle. In addition, drag out a set scoreLabel.Text block from the ScoreLabel drawer, and attach it underneath the set Ball1.Enabled block. From the Built-In palette, open the Text drawer, and drag out a text block and drop it after the "to" (you can also click the work area to get the popup menu and choose text). Click the text and change it to "Game Over!" This text will appear on the screen in the ScoreLabel when the ball gets past the paddle.
2. Now grab the orphaned Ball1.Bounce block that has been sitting alone in your workspace and drop it into the "else-do" socket of the ifelse block. It should still have the value edge block attached to its "edge" socket. This entire ifelse block will cause the ball to bounce off of all edges except the bottom (southern) one. Steps 13 through 15 should make a set of blocks that looks like this:



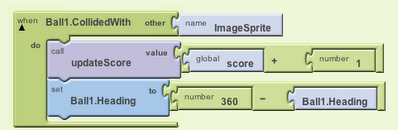
1. From the Built-In palette, open the Definition drawer, drag out a def variable as block and drop it in an open area. Click on "variable" and type "score" to change the name of the variable to score. Put a number block with the value 0 in it into the "as" socket. This creates a variable named "score" and sets its value to 0.
2. Under the Built-In palette, open the Definition drawer, and drag out a procedure block and drop it in an open area. Click on "procedure" and rename it to "updateScore". Then from the Definition drawer, drag out aname block and then click on the "name" and type "scorevalue". This creates a parameter for the procedure that is named "scorevalue". A parameter is a temporary variable that holds a value for a procedure. The value is specified when the procedure is called.  
   From the My Blocks palette, open the My Definitions and drag out a set global score to block and drop it in "do" area of the updateScore procedure. From My Definitions drag out a value scorevalue block and drop it in the "to" area. This sets the score variable to the passed value.
3. From the ScoreLabel drawer, drag out a set scoreLabel.Text block and drop it after the previous block. From the Built-In palette, open the Text drawer, and drag out a join block and drop it after the "to". Set the first blank area to the text "Score:" and set the second blank area to global score from the My Blocks palette and the My Definitions drawer. This will set the text of the score label to a string that joins together "Score:" and the actual value of the score variable.



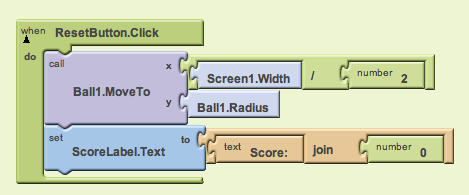
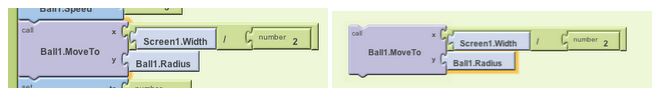
1. Under the My Blocks palette, open the My Definitions drawer, and drag out a call updateScore block and drop it at the end of the StartButton.Click next to the block and type 0 to update the score to 0.



1. Under the My Blocks palette, open the My Definitions drawer, drag out another call updateScore and drop it in the Ball1.CollidedWith block. Under the Built-In palette, open the Math drawer drag out an addition (+)block and drop it at the end of the call updateScore block. From the My Definitions drawer, drag out a global score block and drop it in the first blank area. Click in the second blank area and type 1. This will increment the score by 1.



1. Under the My Blocks palette, open the ResetButton drawer and drag out the when ResetButton.Click to the open area. Go back to the set of blocks surrounded by the StartButton.Click block that you created before and highlight the entire procedure block by clicking once on the green StartButton.Click block, Ball1.Move to, and hit control (ctrl) key and c key together to copy the block. Then click the open area (anywhere) and hit control (ctrl) and v key together to paste the block you copied. Drag the whole copied block in the ResetButton.Click.  
     
   From the ScoreLabel drawer, drag the ScoreLabel.Text to block and place it under the previous block. Type "0" and drag the "0" block to the empty socket of the ScoreLabel.Text to block. In addition, open the Text drawer under the Built-In palette and drag out a join block and drop it after the ScoreLabel.Text. Set the first blank area to the text "Score:" and set the second blank area to the number zero. (Note: Would it also work to call the procedure UpdateScore with 0 as the scorevalue parameter? Why or why not?)  
     
   Done! For a little fun, try changing the color or size of the ball or paddle to personalize your app.



Package the App to Your Phone!

Notice that your app disappears when you plug your phone out from the USB cable. It is because the app is still stored in the server and not in your phone. Follow [these instructions](http://appinventor.mit.edu/explore/curriculum/packaging-apps.html) and package your app to your phone!

Challenge 1: Changing the speed and the size of the ball!

Can you increase the speed of the ball and decreases the size of the ball when the score increases an increment of 10 ? (Hint: Under the Math drawer, the remainder block (a,b) returns the result of dividing a by b and taking the remainder. For example, remainder(11,5) = 1, remainder(-11, 5) = -1, remainder(11, -5) = 1, and remainder(-11, -5) = -1).

Challenge 2: Adding sounds!

Download the audio files from the [App Inventor Media Library](http://appinventor.mit.edu/explore/media-library.html) page, and upload them to the Designer. Make the "noink" sound play when the ball hits the edge of the wall, the "ta-da" sound when the speed of the ball increases, and the buzzer sound when the ball hits behind the paddle.

More Challenges:

* Give the player three lives so that they get three tries before "Game Over".
* If you program multiple lives, decrease the score by 1 each time the player loses a life.
* Investigate what happens if you change the range of random numbers for the start heading when the start button is clicked.
* Try making the app respond to titling of the phone instead of dragging the paddle

Improve this game - fix the "bug" that happens if you click the start button after a game is over:

Notice that when the game is over, if the player clicks "start" instead of "refresh", the message "Game Over" remains on the screen until the player scores the first point. How can this be fixed? Is it really necessary to have a start button and reset button?