

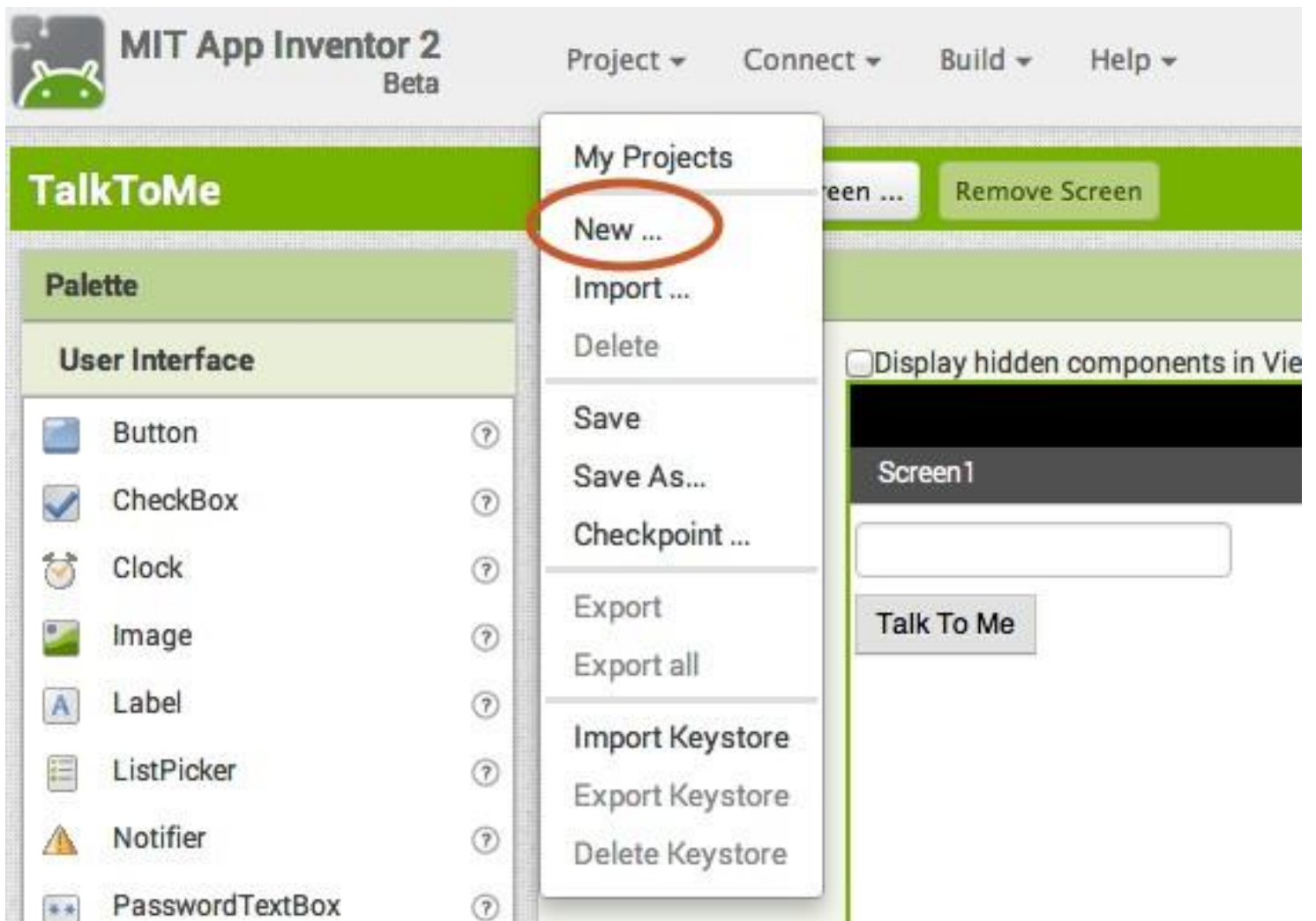


BallBounce: A simple game app

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

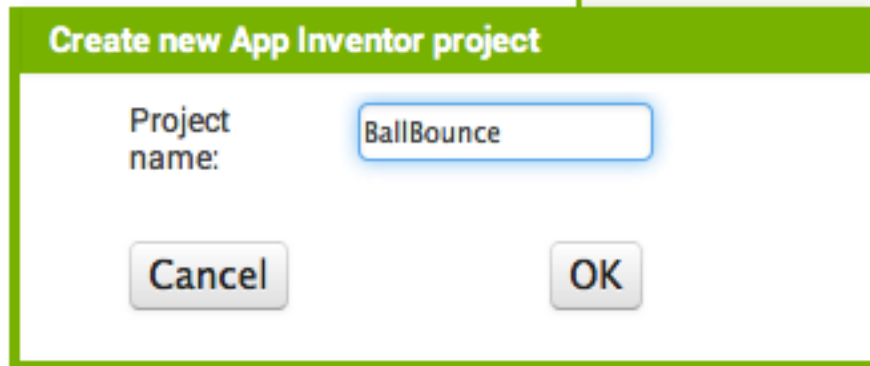
If you have another project open, go to My Projects menu and choose New Project.





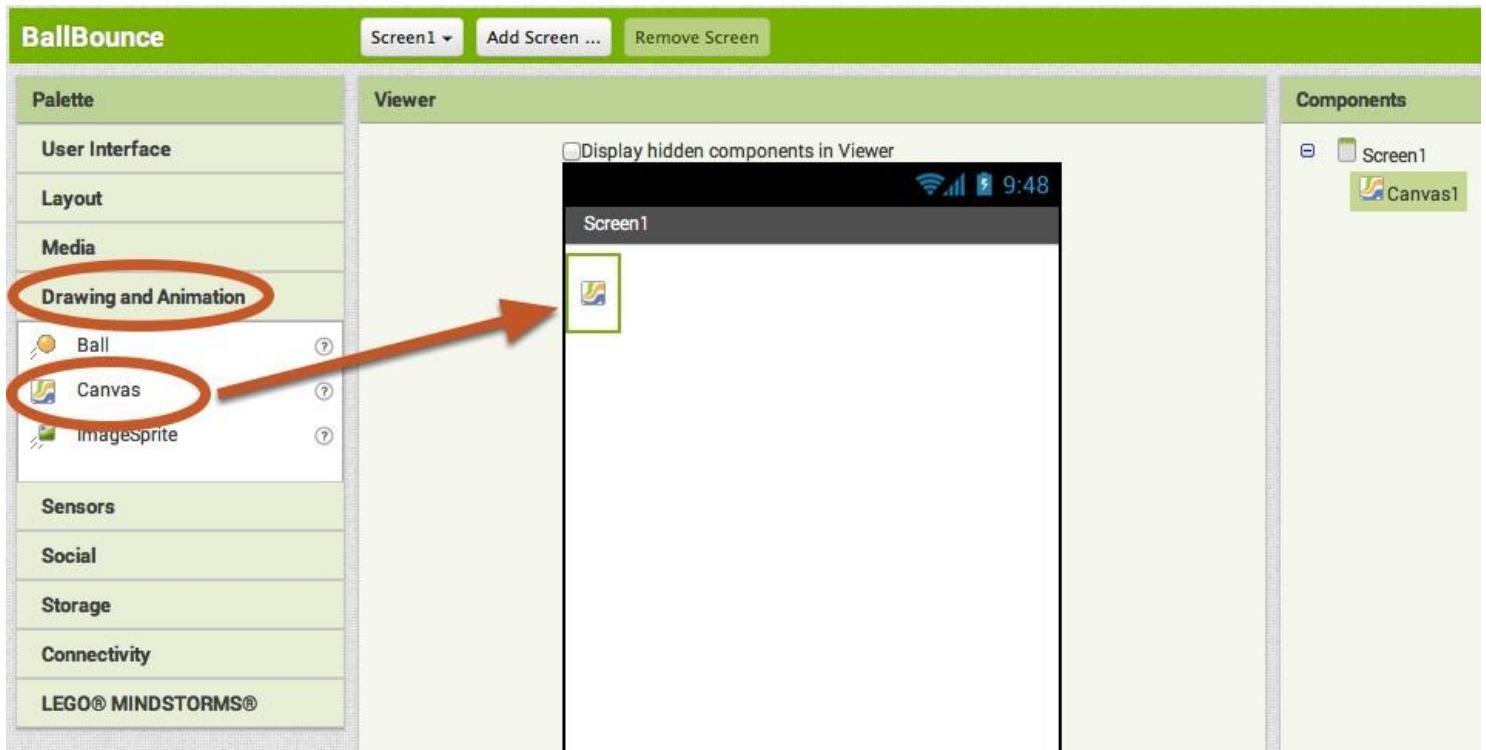
Name the Project

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



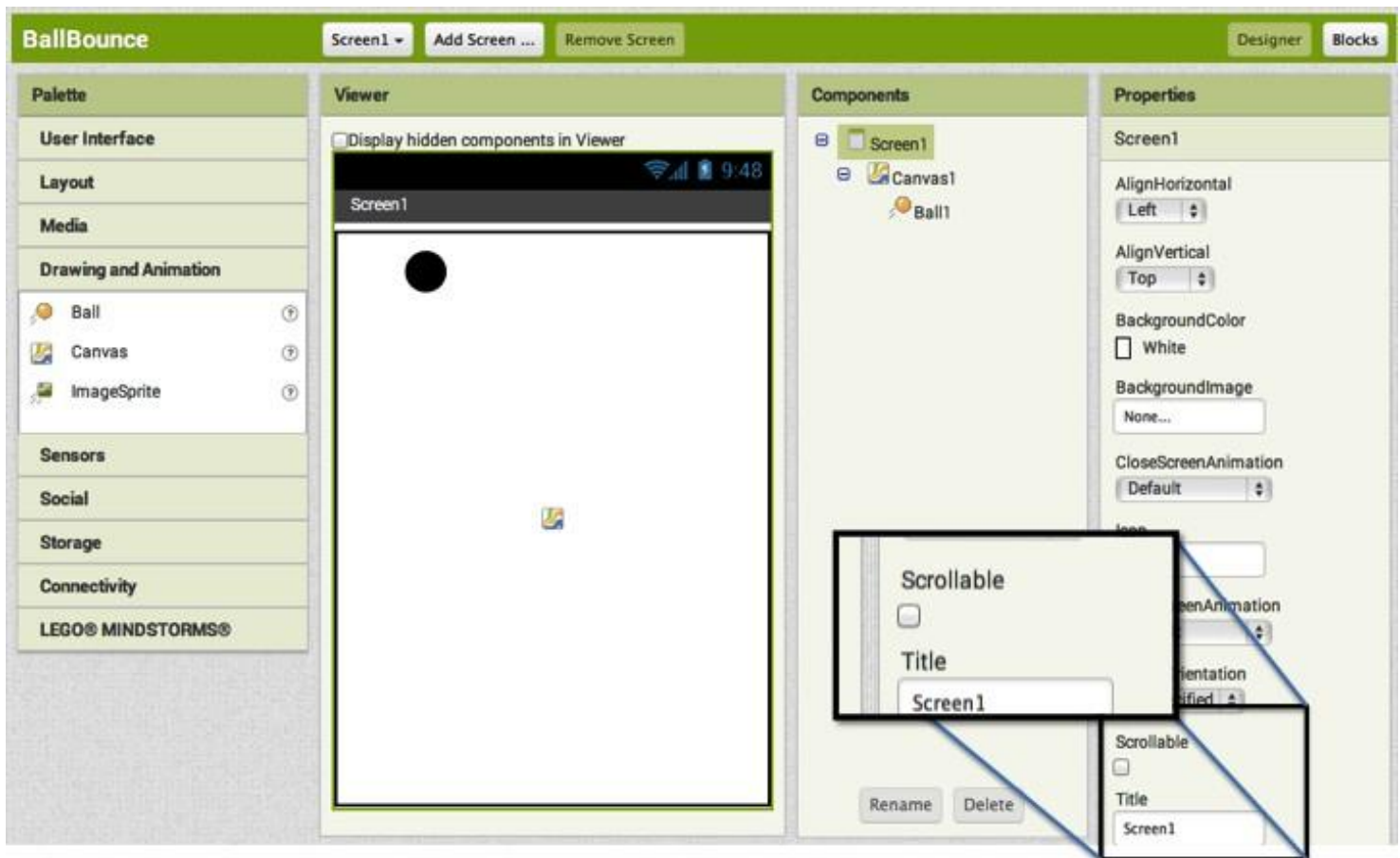
Add a Canvas

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 up the whole screen.



The screenshot shows the MIT App Inventor interface for an app named "BallBounce". The interface is divided into four main panels: Palette, Viewer, Components, and Properties. The Palette panel on the left contains categories like User Interface, Layout, Media, Drawing and Animation, Sensors, Social, Storage, Connectivity, and LEGO MINDSTORMS. The Viewer panel in the center shows a preview of the app screen with a black ball and a small icon. The Components panel on the right shows a tree view with "Screen1" containing "Canvas1" and "Ball1". The Properties panel on the far right shows the settings for "Screen1", including "AlignHorizontal" (Left), "AlignVertical" (Top), "BackgroundColor" (White), and "CloseScreenAnimation" (Default). A callout box highlights the "Scrollable" property, which is currently unchecked. Another callout box highlights the "Title" property, which is set to "Screen1".



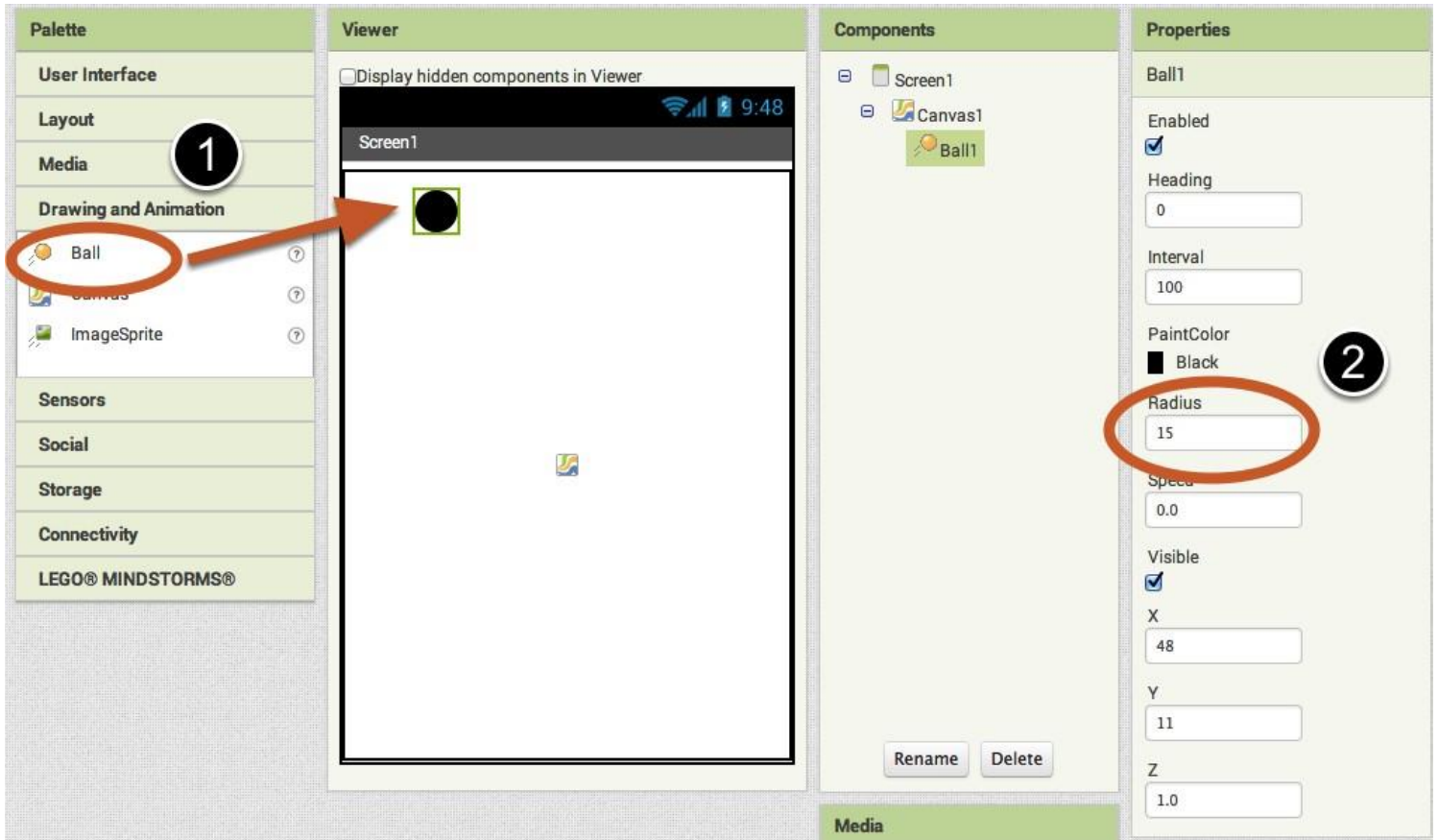
Change the Height and Width of the Canvas to Fill Parent

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**.

The screenshot shows the MIT App Inventor interface for a project named "BallBounce". The interface is divided into several panes: a Palette on the left, a Viewer in the center, a Components pane on the right, and a Properties pane on the far right. The Components pane shows "Screen1" selected, with "Canvas1" highlighted below it, marked with a circled "1". The Properties pane shows the properties for "Canvas1", with the "Height" and "Width" properties highlighted, marked with a circled "2". A dialog box is open over the "Height" property, showing three radio button options: "Automatic" (selected), "Fill parent", and "pixels". An arrow points to the "Fill parent" option. The "Width" property in the background Properties pane also shows "Fill parent..." as its value.

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 displays 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 **Viewer** panel's Canvas. In the **Components** panel, the hierarchy shows Screen1 containing Canvas1, which contains Ball1. In the **Properties** panel, the **Radius** property for Ball1 is set to 15 and is circled in red with a circled '2' next to it. Other visible properties 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. The top bar is green and contains the project name 'BallBounce', a dropdown menu for 'Screen1', and buttons for 'Add Screen ...' and 'Remove Screen'. Below this is the 'Blocks' panel on the left, which is divided into 'Built-in' and 'Screen1' sections. The 'Built-in' section has categories like Control, Logic, Math, Text, Lists, Colors, Variables, and Procedures. The 'Screen1' section has a 'Ball1' component selected and circled in red. The 'Viewer' panel on the right shows the workspace with several event handler blocks for 'Ball1':

- when Ball1.CollidedWith**: has an 'other' block in the 'do' field.
- when Ball1.Dragged**: has input fields for 'startX', 'startY', 'prevX', 'prevY', 'currentX', and 'currentY' in the 'do' field.
- when Ball1.EdgeReached**: has an 'edge' block in the 'do' field.
- when Ball1.Flung**: has input fields for 'x', 'y', 'speed', 'heading', 'xvel', and 'yvel' in the 'do' field.
- when Ball1.No longer Colliding With**: is partially visible at the bottom.

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*.

The image shows a single event handler block for 'when Ball1.Flung'. The block is yellow and has a 'do' field with input fields for 'x', 'y', 'speed', 'heading', 'xvel', and 'yvel'.



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' panel, which is organized into categories: Built-in (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), Screen1, Canvas1, and Any component. The 'Ball1' component is selected and circled in orange. At the bottom of the 'Blocks' panel are 'Rename' and 'Delete' buttons. On the right is the 'Viewer' panel, which displays a list of blocks for the selected 'Ball1' component. The blocks are: 'set Ball1 . Enabled to', 'Ball1 . Heading', 'set Ball1 . Heading to' (circled in orange), 'Ball1 . Interval', 'set Ball1 . Interval to', 'Ball1 . PaintColor', 'set Ball1 . PaintColor to', 'Ball1 . Radius', 'set Ball1 . Radius to', 'Ball1 . Speed', 'set Ball1 . Speed to' (circled in orange), and 'Ball1 . Visible'. A red arrow points downwards on the right side of the Viewer panel, and a text box says 'Scroll down to the green "setter" blocks for the Ball's Heading and Speed'.

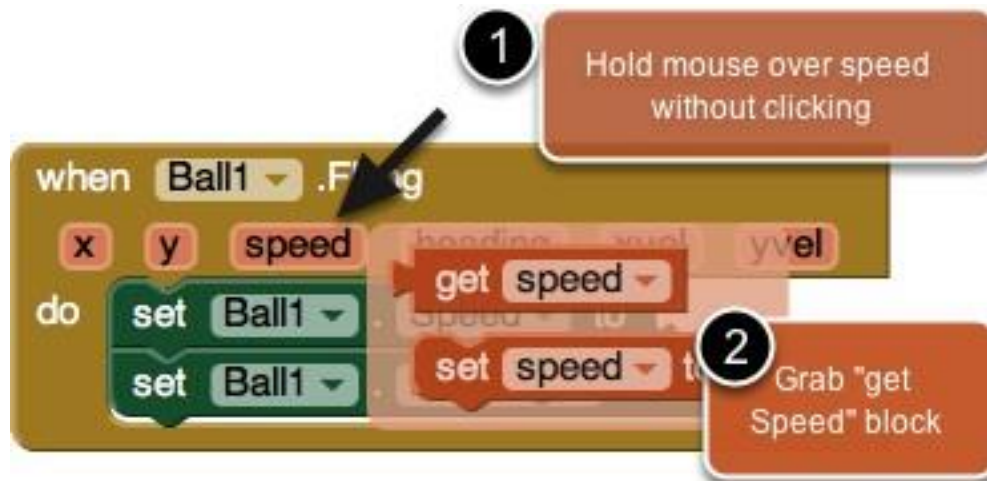


Plug the set Ball1.Speed and set Ball1.Heading into the Flung event handler

```
when Ball1 .Flung
  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 Flung gesture's speed

Mouse over the "speed" parameter of the **when Ball1.Flung** 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

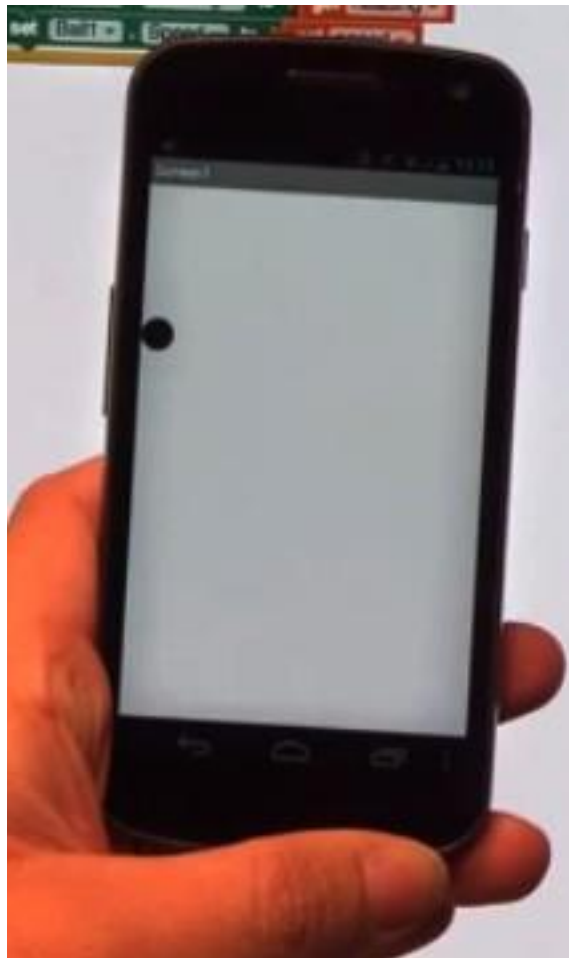
Do the same for the Ball's heading. Mouse over the **heading** parameter and you'll see the **get heading** block appear. Grab that block, and click it into the **set Ball1.Heading** block.

```
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' palette, and on the right is the 'Viewer' showing a code editor with several event blocks for 'Ball1'. The 'Ball1' component is selected in the 'Blocks' palette, and the 'when Ball1.EdgeReached do' block is highlighted in the Viewer. An orange arrow points from the highlighted block in the Viewer to the right.

Blocks

- Built-in
 - Control
 - Logic
 - Math
 - Text
 - Lists
 - Colors
 - Variables
 - Procedures
- Screen1
- Ball1
- Any component

Viewer

```
when Ball1 .CollidedWith  
  other  
do  
  
when Ball1 .Dragged  
  startX startY prevX prevY currentX currentY  
do  
  
when Ball1 .EdgeReached  
  edge  
do  
  
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 is the 'Blocks' palette with categories like Control, Logic, Math, Text, Lists, Colors, Variables, and Procedures. Under 'Screen1', 'Canvas1', and 'Ball1' are listed. On the right is the 'Viewer' showing a script for 'Ball1'. The script includes several event blocks: 'when Ball1.TouchDown', 'when Ball1.TouchUp', 'when Ball1.Touched', and 'when Ball1.EdgeReached'. A red circle highlights a 'call Ball1.Bounce' block in the 'do' area of the 'when Ball1.Touched' block. An orange arrow points from this block to the 'edge' parameter in the '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.

This close-up shows the 'when Ball1.EdgeReached' block. The 'edge' parameter is highlighted. A 'get edge' block is being dragged from the 'edge' parameter slot into the 'edge' parameter slot of a 'call Ball1.Bounce' block within the 'do' area of the event 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!

