



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

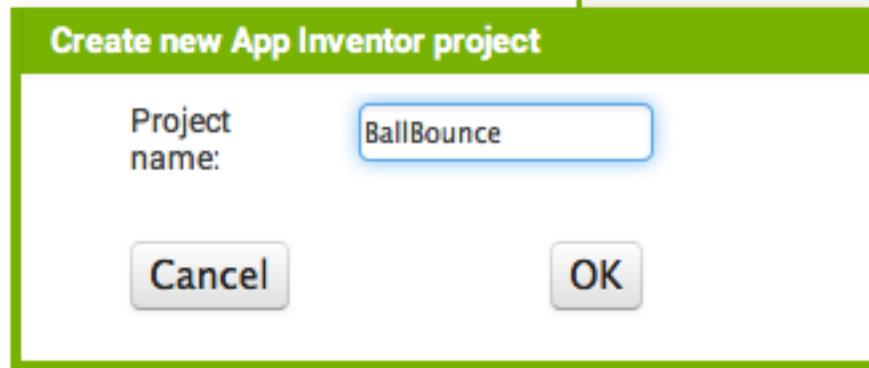
If you have another project open, go to My Projects menu and choose New Project.

The screenshot shows the MIT App Inventor 2 Beta web interface. At the top, there is a navigation bar with the MIT App Inventor logo and the text 'MIT App Inventor 2 Beta'. To the right of the logo are four dropdown menus: 'Project', 'Connect', 'Build', and 'Help'. Below the navigation bar is a green header area with the text 'TalkToMe' on the left and 'Screen ...' and 'Remove Screen' buttons on the right. On the left side, there is a 'Palette' section with a 'User Interface' sub-section. This section contains a list of UI components: Button, CheckBox, Clock, Image, Label, ListPicker, Notifier, and PasswordTextBox. Each component has a small icon and a question mark icon to its right. In the center, a 'My Projects' dropdown menu is open, and the 'New ...' option is circled in red. The menu also includes options like 'Import ...', 'Delete', 'Save', 'Save As...', 'Checkpoint ...', 'Export', 'Export all', 'Import Keystore', 'Export Keystore', and 'Delete Keystore'. On the right side, there is a 'Display hidden components in View' checkbox and a 'Screen1' label above a text input field and a 'Talk To Me' button.



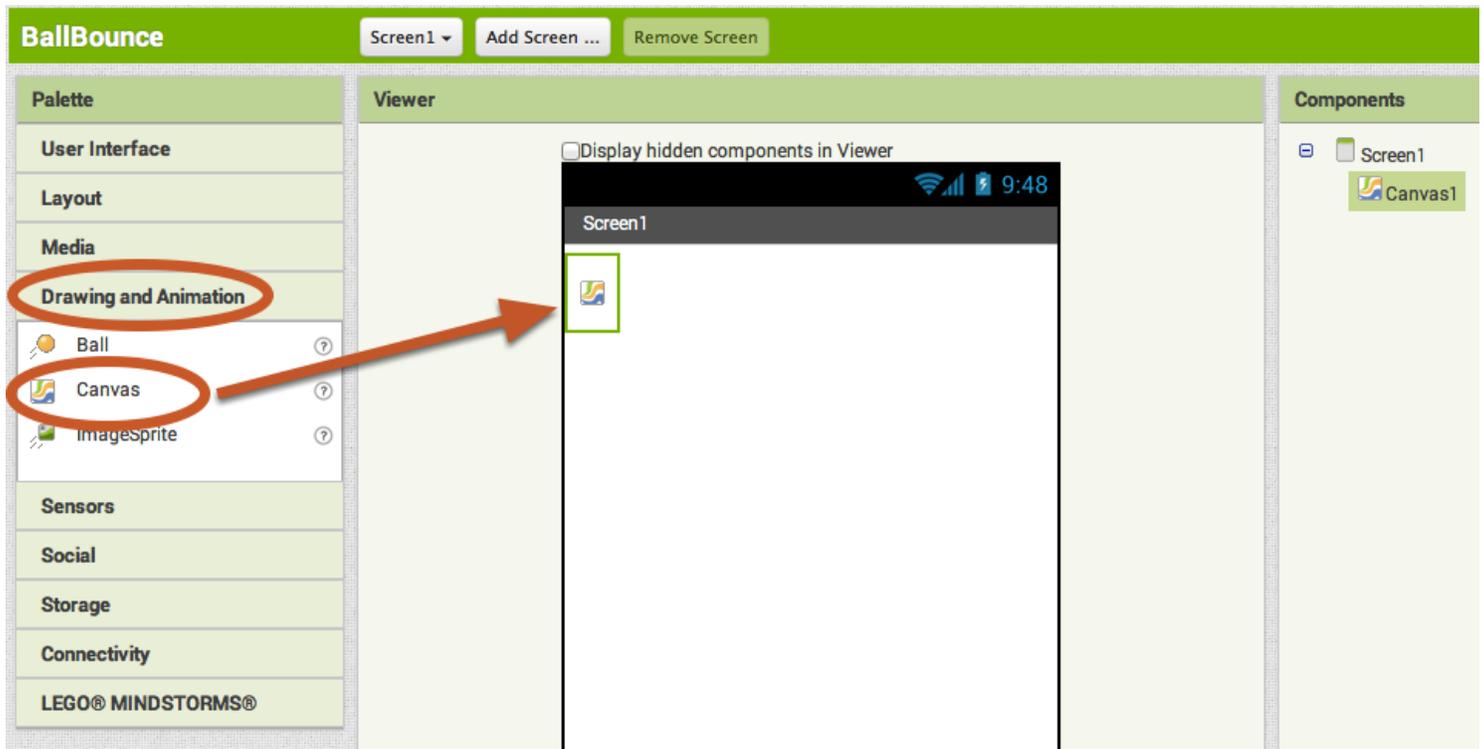
Name the Project

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



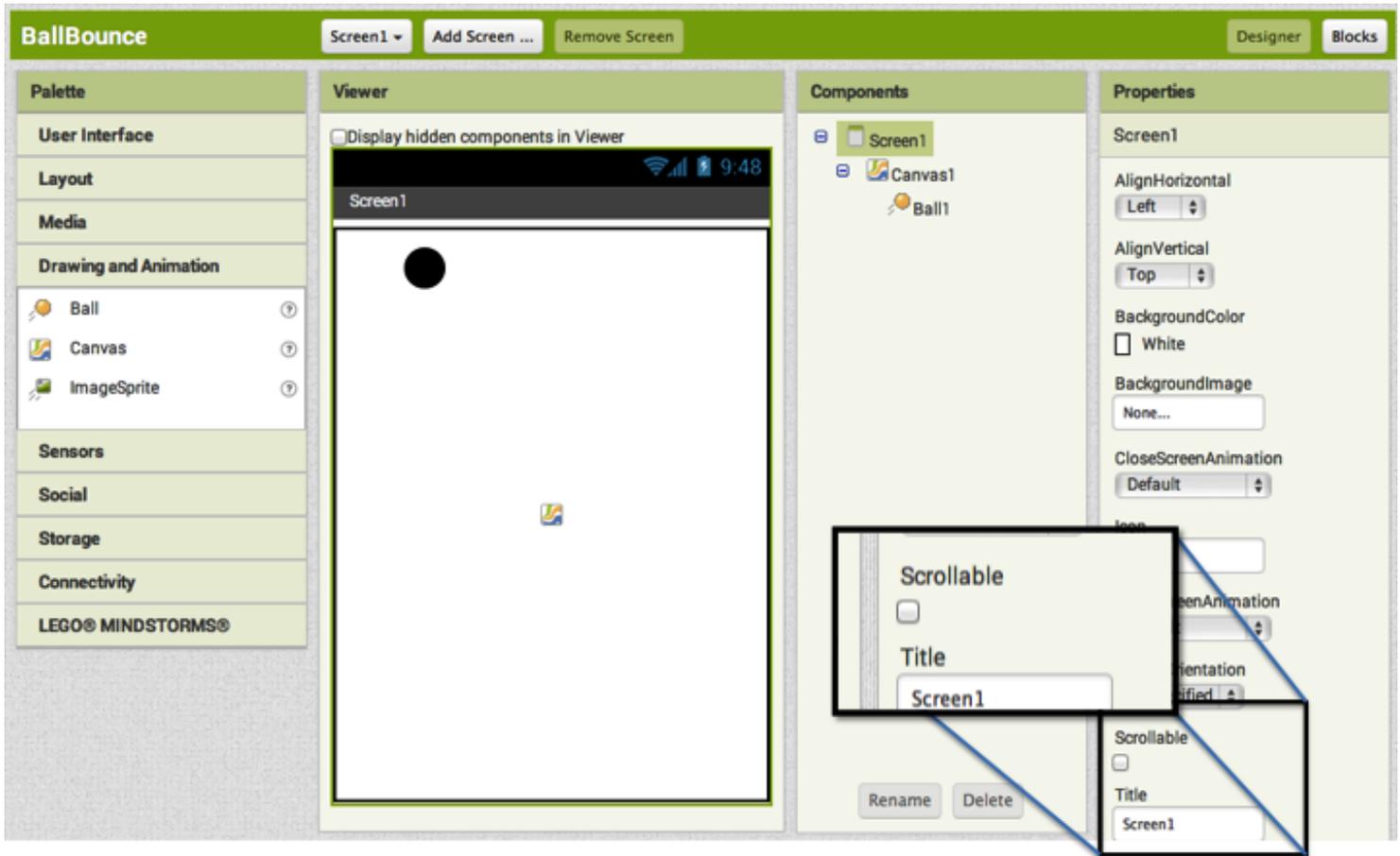
Add a Canvas

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



Set the Screen so that it does not scroll

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



The screenshot shows the MIT App Inventor interface for an app named "BallBounce". The interface is divided into four main panels: Palette, Viewer, Components, and Properties. The Palette on the left lists various components like Ball, Canvas, and ImageSprite. The Viewer in the center shows a mobile device screen with a black ball and a small icon. The Components panel on the right shows a hierarchy: Screen1 containing Canvas1, which contains Ball1. The Properties panel on the far right shows settings for Screen1, including AlignHorizontal (Left), AlignVertical (Top), BackgroundColor (White), and CloseScreenAnimation (Default). Two callout boxes are overlaid on the Properties panel, highlighting the "Scrollable" checkbox, which is currently unchecked. The top callout box shows "Scrollable" with an unchecked checkbox and "Title" set to "Screen1". The bottom callout box shows "Scrollable" with an unchecked checkbox and "Title" set to "Screen1".



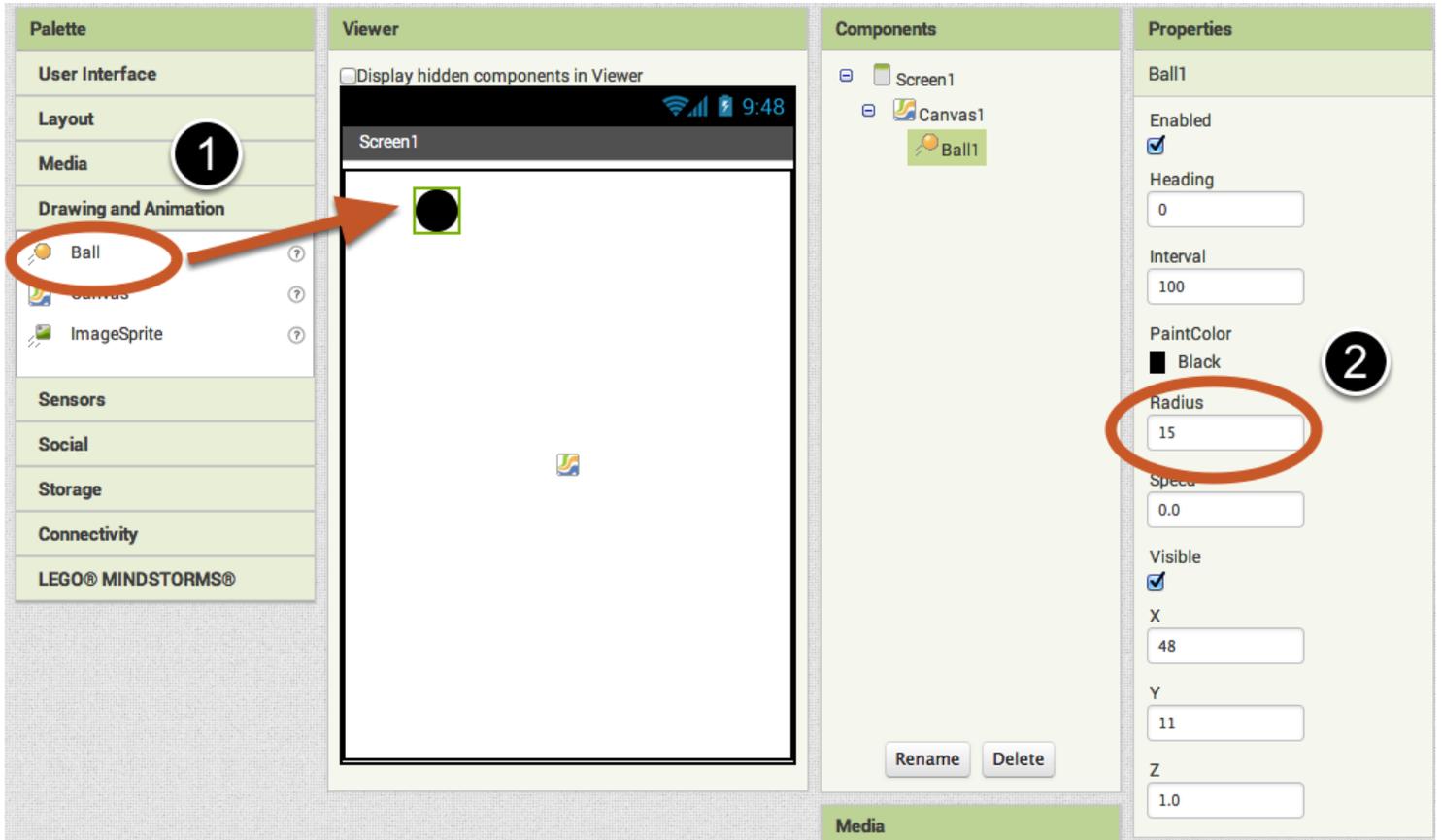
Change the Height and Width of the Canvas to Fill Parent

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: Palette, Viewer, Components, and Properties. The Components pane shows "Screen1" and "Canvas1". The Properties pane shows the properties for "Canvas1". A dialog box is open over the Properties pane, showing the "Height" property set to "Automatic" and the "Width" property set to "Fill parent...". A black arrow points to the "Fill parent" radio button in the Height section. Circled numbers 1 and 2 are placed over the Components and Properties panes respectively.

Add a Ball

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



The screenshot shows the MIT App Inventor interface with four main panels: Palette, Viewer, Components, and Properties. In the **Palette** panel, the **Drawing and Animation** category is selected, and the **Ball** component is circled with a red circle and a circled '1'. An orange arrow points from the Ball component to the **Viewer** panel, where a black ball is placed on the Canvas. In the **Components** panel, the **Ball1** component is visible under the **Canvas1** component. In the **Properties** panel, the **Radius** property is circled with a red circle and a circled '2', and its value is set to 15. Other properties like **PaintColor** (Black), **Speed** (0.0), **Visible** (checked), **X** (48), **Y** (11), and **Z** (1.0) are also visible.

Open the Blocks Editor.



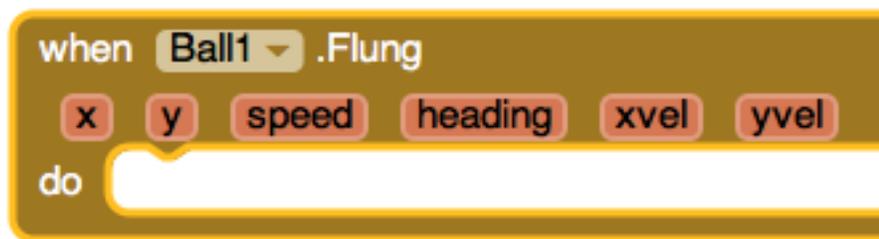


Open the Ball1 Drawer to view the Ball's blocks.

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 a 'Blocks' panel on the left and a 'Viewer' panel on the right. The 'Blocks' panel has a tree view with categories like 'Built-in', 'Screen1', and 'Any component'. Under 'Any component', 'Ball1' is selected and circled in red. The 'Viewer' panel shows a workspace with several event handler blocks for 'Ball1': 'when Ball1 .CollidedWith', 'when Ball1 .Dragged', 'when Ball1 .EdgeReached', and 'when Ball1 .Flung'. Each block has a 'do' field for code.

Drag out the Flung Event Handler

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





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

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

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

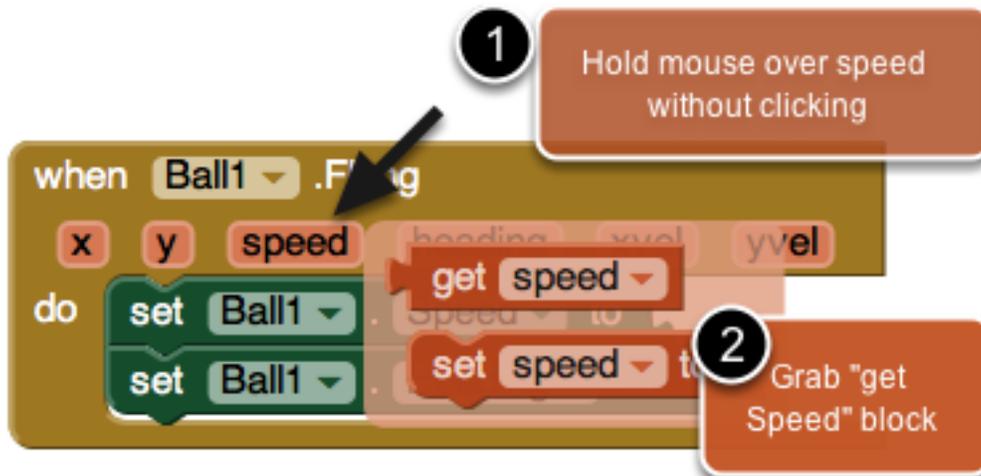


Plug the set Ball1.Speed and set Ball1.Heading into the Flung event handler

```
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.





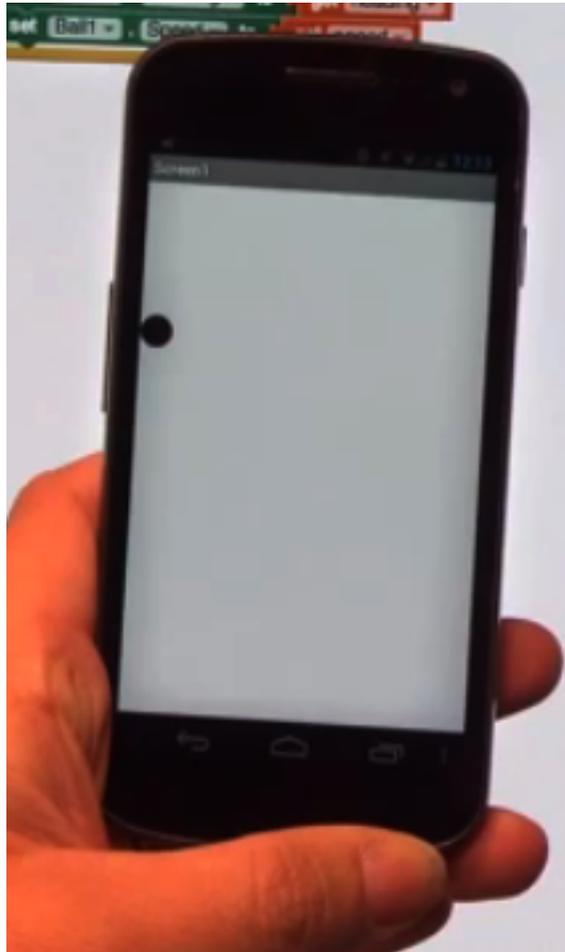
Set the Ball's heading to be the same as the Fling gesture's heading

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' drawer, and on the right is the 'Viewer' workspace.

Blocks Drawer:

- Built-in
 - Control
 - Logic
 - Math
 - Text
 - Lists
 - Colors
 - Variables
 - Procedures
- Screen1
- Component1
 - Ball1** (highlighted with a red circle)
- Any component

Buttons: Rename, Delete

Viewer Workspace:

The workspace contains several event blocks for 'Ball1':

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



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

The screenshot shows the MIT App Inventor interface. On the left is the 'Blocks' panel with a tree view showing 'Screen1' > 'Canvas1' > 'Ball1'. On the right is the 'Viewer' panel showing a code editor. Several event blocks for 'Ball1' are visible: 'when Ball1.TouchDown', 'when Ball1.TouchUp', 'when Ball1.Touched', and 'when Ball1.EdgeReached'. A purple 'call Ball1.Bounce' block is highlighted with an orange oval, and an orange arrow points from it to the 'edge' parameter of 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.

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



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

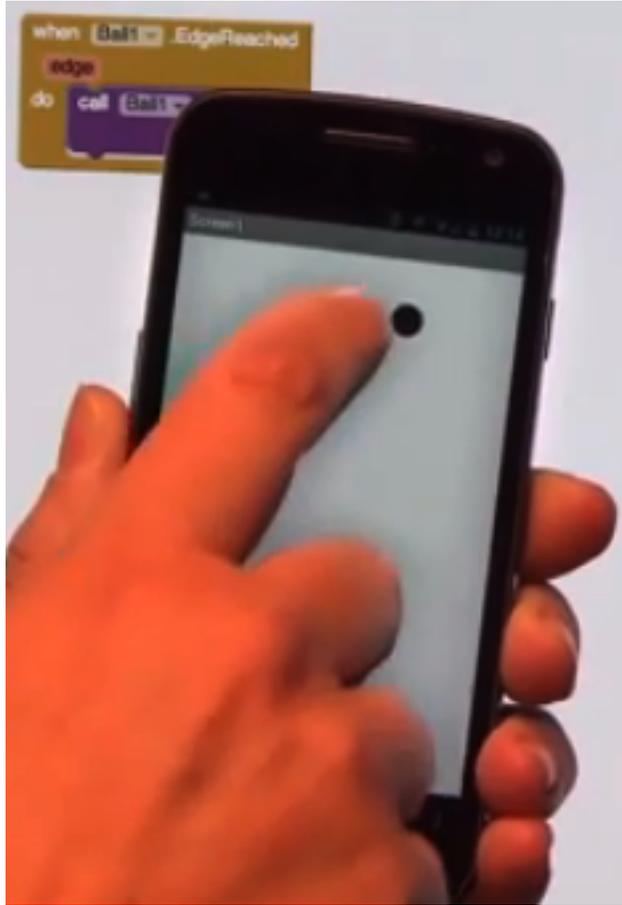
```
when Ball1 .Flung
  x y speed heading xvel yvel
do
  set Ball1 . Speed to get speed
  set Ball1 . Heading to get heading
```

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



Test it out!

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



There are many ways to extend this app.

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

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

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

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