SAMPLE CHAPTER

Hello App Inventor!

Android programming for kids and the rest of us

Paula Beer and Carl Simmons

MANNING

Hello App Inventor!

Android programming for kids and the rest of us

by Paula Beer and Carl Simmons

Chapter 9

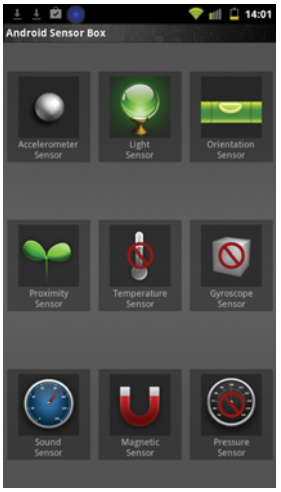
Brief contents

CHAPTER 9

Position sensors

In chapter 2, you shook your phone and made a sheep disappear using the phone's accelerometer sensor. Most phones have sensors that can detect much smaller motions than a strong shake. You can see this when you tilt the phone to one side and the display rotates, or if you've played a driving game where you steer left and right by gently tilting the phone. Phones can tell which way up they're facing—for instance, some people set their phone's ringer to be silent automatically if the phone is face down on a desk. Your phone can also tell which direction is north by using its own built-in compass.

All the apps in this chapter rely on you using an actual phone, because the emulator doesn't have any position sensors. Most phones will work with the apps in this chapter because the sensors you're using are common. You can double-check exactly which sensors your phone has by downloading a free app from the Google Play Store such as the excellent Android Sensor Box from iMobLife (shown at right). You might find that your phone has some sensors that App Inventor can't access yet (like a proximity or light sensor), but these features may be added to App Inventor as the sensors become common in more phones.



By the end of the chapter, you'll have made apps that are controlled not by clicking buttons, but by altering the phone's position in space. You'll create an amazing magic trick and a simple motion-controlled game.

We'll start by looking at how the phone knows which direction you're facing by looking at the compass sensor. Or, as you can call it if you want to impress your friends, the *magnetometer*.

Compass app

PURPOSE OF THIS APP

This app is a simple compass. When you move the phone, the compass rotates so that N always lines up with the Earth's magnetic North Pole (as long as there are no strong magnets near the phone). That means the compass direction that is at the top of the phone is the direction you're currently facing. There's also a heading label at the top of the screen that tells you your current direction in degrees from north. So, for example, if you face south, the letter S will be at the top of your phone and the heading will be 180 degrees.

APP RATING

2

ASSETS YOU'LL NEED

Compass.png.

Compass			
Screen1 properties	**Title**: Compass **AlignHorizontal**: Center **ScreenOrientation**: Portrait **Scrollable**: No (unselected)		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
Label	**HeadingLabel**	Displays the direction in which the top of the phone is pointing (in degrees)	**FontBold**: Yes (selected) **FontSize**: 28 **Text**: "Heading"
Canvas	**Canvas**	Holds the compass face	**Width** and **Height**: Fill Parent

Components	What do I rename it?	What does it do?	What properties do I set?
`ImageSprite`	`CompassSprite`	Displays the compass face	`BackgroundImage`: compass.png `Rotates`: Yes (selected) `Width` and `Height`: Automatic
`OrientationSensor` Palette group: Sensors	`OrientationSensor1`	Tells you the azimuth—the phone's current heading based on the Earth's magnetic field	`Enabled`: Yes (selected)

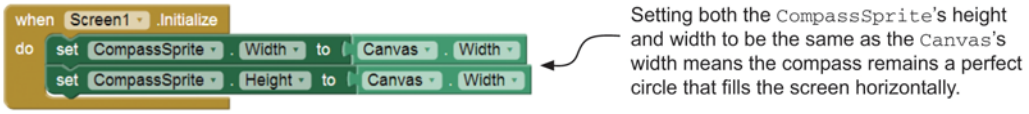
1. Setting up the screen

The screen layout for this app is simple: a heading label plus a compass face. One new idea we're including for this app is that the compass face automatically resizes to any screen size; so whether you're using a tiny 2-inch smartphone screen or a 10-inch tablet, the compass should fill the screen. You do this by setting up a basic screen layout and then programming some blocks to resize the compass face depending on the size of the screen. Here's the basic layout.

The non-visible `OrientationSensor` tells you the phone's position in three-dimensional space.

The canvas's height and width are set to Fill Parent so that when the app starts, the canvas automatically resizes to the width and height of your phone's screen. This would make the

compass a strange-looking ellipse. You want **CompassSprite** to have the same height and width as the canvas width—then it will fill the screen horizontally. Here are the blocks to resize the compass:

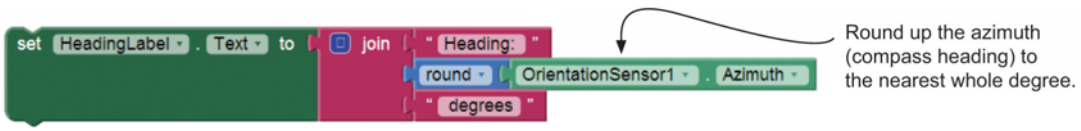


Notice that you're using the **Screen1.Initialize** event to trigger this change so it will happen as soon as the app runs. The user probably won't even notice that the compass is being resized.

2. Coding the blocks: where are you heading?

You need to know what direction the phone is pointing; then you can update your heading label and rotate the compass face to match that direction. The orientation sensor you added in Design view gives you lots of information about the position of the phone (see the "Defying gravity" Learning Point, later in the chapter, for details). You'll use the **Azimuth** property of the sensor to tell you which compass direction the top of the phone is pointing toward. The azimuth gives you a reading from 0 to 360 degrees. For example, the azimuth is 0 degrees when the top of the device is pointing north, 90 degrees when it's pointing east, 180 degrees when it's pointing south, 270 degrees when it's pointing west, and so on.

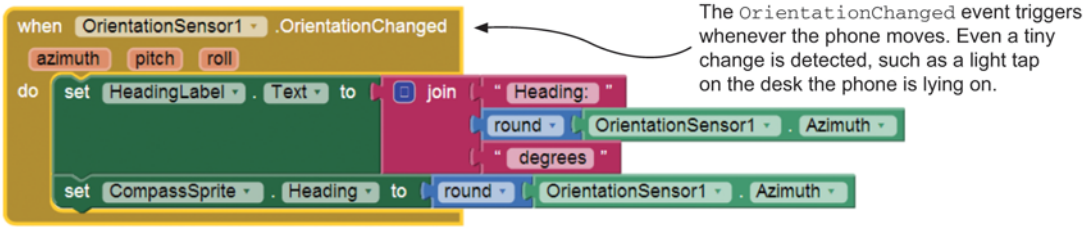
The azimuth reading gives you lots of decimal points of accuracy. That's great if you have a highly accurate device and need to steer a ship around the globe, but most of us only need a rough navigation guide—say, to the nearest degree. So you'll round off the **Azimuth** value using a Math block called **round**. Here's the block that outputs the heading at the top of the screen:



That takes care of the Heading label at the top of the screen. Now you need to rotate the compass to the same heading—and this is surprisingly easy! You've set **CompassSprite**'s **Rotates** property to **yes**, so if you set **CompassSprite**'s **Heading** property to match **Azimuth**, the sprite will rotate to point in the right direction. Here's the block:

The reason this works is complicated and is probably easiest to understand from a diagram—see the "Compass and sprite headings" Learning Point.

The final piece of the puzzle is how and when you should trigger the previous two blocks. You know from chapter 7 that you could set up a clock that triggers, say, every tenth of a second to run these blocks and update the screen. But the orientation sensor has one more trick up its sleeve: it can trigger an event whenever the position of the phone changes. The event is called `OrientationSensor.OrientationChanged`. If you drag out that event block and insert the two blocks you just saw, you have the finished program:

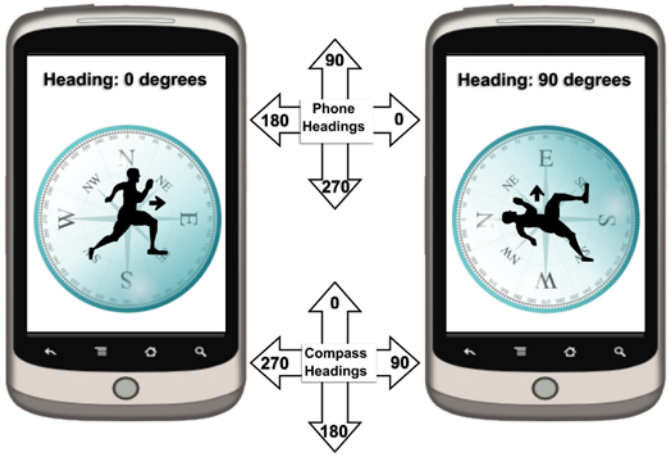


Learning Point: Compass and sprite headings

Compass headings start at 0 degrees (north) pointing up and increase as you move around the compass clockwise. You found out in chapter 8 that App Inventor sprite headings start at 0 degrees pointing to the right side of the screen and increase as you travel counterclockwise. If that's so, how does the Compass app work without converting from one system to another?

We hope this diagram will explain. We have superimposed a running man onto the compass so you can see the direction the compass sprite would travel if we gave it some speed (remember, in reality it stays in one place and rotates around its center):

When the phone is pointed north, the azimuth heading is 0 degrees. A moving sprite with a heading of 0 moves right. App Inventor assumes sprites are drawn facing right, so the sprite won't rotate.



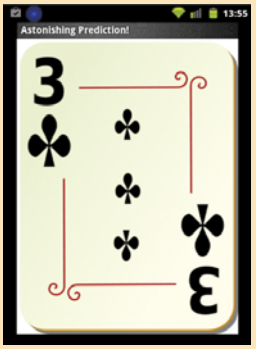
When the phone is pointed east, the azimuth heading is 90 degrees. A moving sprite with a heading of 90 moves up. App Inventor rotates the sprite so it looks like it's facing up on the screen, and your compass now has east at the top.

Before we move on to look at some of the other orientation sensors in your phone, you're going to use the magnetometer (or compass) once more to perform an astonishing magic trick.

Astonishing Prediction! app

PURPOSE OF THIS APP

This app turns you into a master illusionist. You deal four playing cards face up on the desk and place your phone face down. You ask a member of the audience to point to any one of the cards. Announcing that you've already predicted their choice, you flip over your phone to reveal a matching card on its screen. The audience can examine the phone as much as they like, and then you can repeat the trick with another audience member.



APP RATING

3

ASSETS YOU'LL NEED

Images of four playing cards (2 to 5 of clubs).

Astonishing Prediction			
Screen1 properties	`Title`: Astonishing Prediction! `AlignHorizontal`: Center `ScreenOrientation`: Portrait `Scrollable`: No (unselected)		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
`Button`	`PredictionButton`	Lets you switch the `OrientationSensor` on/off. Also displays an image of the chosen card via its `Image` property.	`Text`: Blank `Width` and `Height`: Fill Parent
`OrientationSensor` Palette group: Sensors	`OrientationSensor1`	Tells you the azimuth—the phone's current heading based on the Earth's magnetic field	`Enabled`: No (unselected)

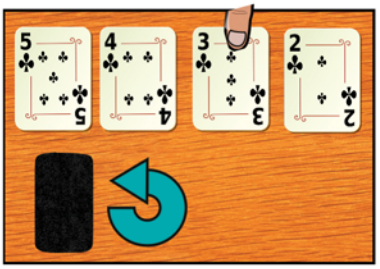
Understanding the secret of the app

Have you guessed how such a simple app knows which card has been chosen? Cunningly, you'll use the phone's compass to work out which direction the phone is facing when it's flipped over. If it's pointing roughly northeast, you'll display the 2 of clubs; southeast, and it's the 3 of clubs; southwest, the 4 of clubs; and northwest, the 5 of clubs.

How do you stop the image from changing whenever the phone is being waved around? You'll use a *long click* of a button (which fills the screen) to activate the orientation sensor—a long click means you leave your finger on the button for a couple of seconds. A short click (regular click) disables the orientation sensor. So any time you show the screen to your audience, a quick touch of the screen freezes the image until you long-click the button again. The following sequence should make the trick clear from start to finish:

1. Before you start, locate a memorable item in the room that is directly north of where you'll perform the trick. Memorize the positions of the cards shown above.

2. Run the app. Keeping the phone screen private, long-click the grey button to activate the orientation sensor. Flip the phone face down on the desk.



3. Invite an audience member to point to a card. While asking them, "Are you sure that was a free choice?" casually spin the phone (keeping it face down) to point in whichever compass direction you memorized for that card in step 1.

4. Carefully flip the phone, keeping it pointing in the correct direction. At the same time …

5. … short-click the button (which now displays the selected card). This disables the orientation sensor and "locks" the card so it doesn't change.

6. After the applause dies down, you can restart the trick by secretly long-clicking the card and flipping the phone face down.

Here are some performance top tips:

- Make up a story to explain your powers. Perhaps you have mind control and have already selected the card that you'll insert into the audience member's brain!
- Always point the phone either NE, NW, SE, or SW. Pointing directly N, S, E, or W will mean you're never sure whether you have it in the right direction. A little less than north, and the card will be the 5 of clubs, whereas a little more than north and it will be the 2 of clubs.
- If you're struggling to remember which compass direction you need to point the phone, you could lay out the cards in their compass positions. So, place the 2 of clubs in the northeast position on the table, the 3 of clubs southeast, and so on, in a circle with the phone face down in the center—but don't be surprised if your audience guesses the secret.
- If you've set your phone to go into sleep mode when it's not being used, you'll find that it keeps sleeping because you aren't touching the screen when the phone is face down on the desk. Flipping over a blank screen isn't impressive, so adjust or disable the screen-timeout duration in your phone's system settings.

1. Setting up the screen

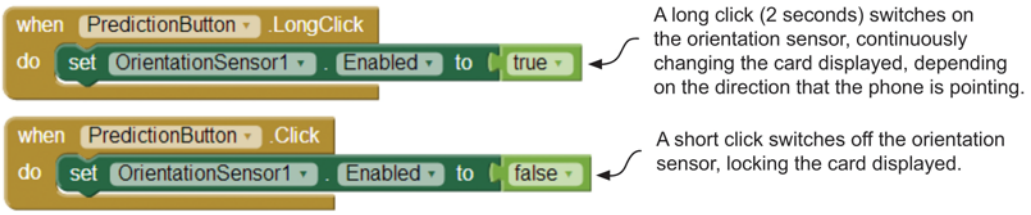
You use a straightforward layout here: just a blank button that fills the screen, an orientation sensor, and four images of the four cards. You could substitute different images if you wanted to. For example, you could take pictures of four objects to put in the app, and then place those objects on the table for the spectator to choose from. Another idea might be to use pictures of four friends you know will be watching the trick, and have one of them selected as the prediction. Use your imagination to create a magical, memorable illusion.

Note that you start the app with the orientation sensor disabled because you use the `OrientationChanged` event to trigger the card change.

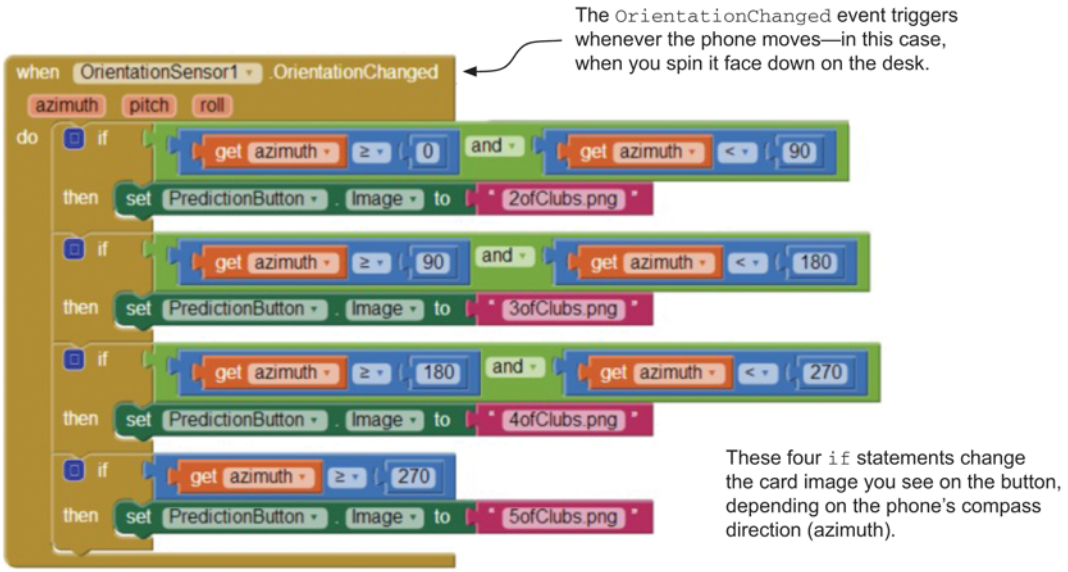
You don't want that to happen until the phone is face down and the screen is out of sight.

2. Coding the blocks: abracadabra!

Let's deal with switching the orientation sensor on and off first. Remember that a long click of the button activates the sensor and a short click deactivates it, like so:



Whenever the phone moves, you want to update the picture of the card that is displayed:



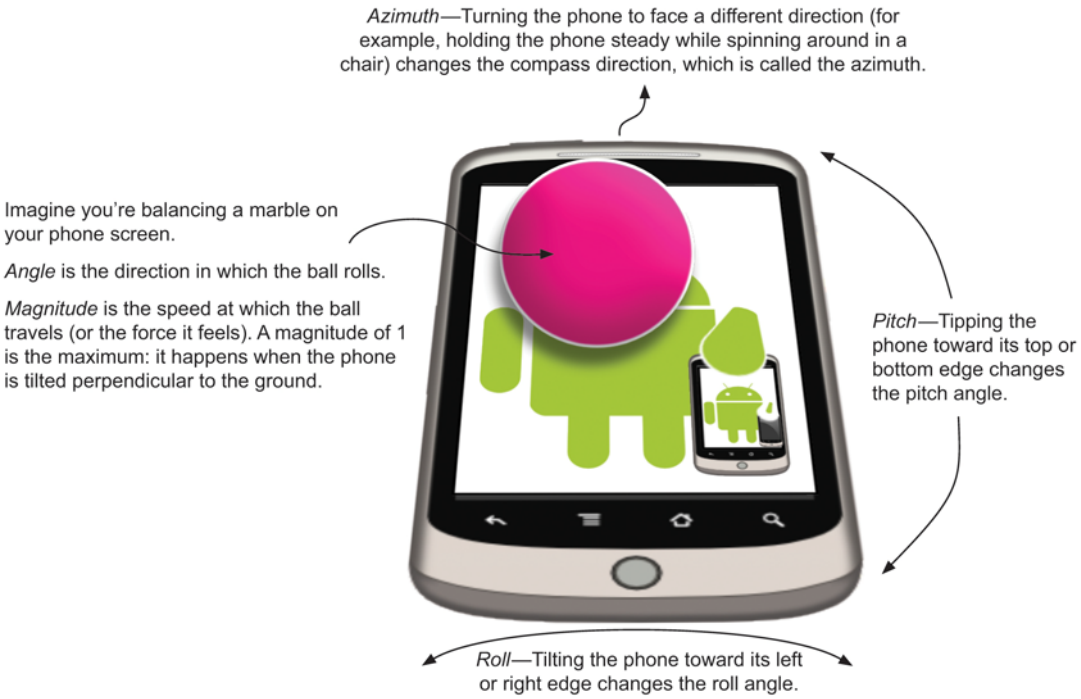
To avoid errors and unpredictable behavior, it's important that you make sure all values from 0 to 360 can be true for only one of the **`if`** statements. For example, if the phone is pointing 90 degrees from north, it will only display the 3 of clubs—because the previous **`if`** statement covers from greater than or equal to zero up to *less than 90*.

Testing it

If you try the app on your phone with the screen face up, you should quickly get the hang of switching the orientation sensor on/off, and you'll be able to see how the card changes as you rotate the phone. Practice the spin, click, and flip move—then go astonish your friends!

Learning Point: Defying gravity

You've come to grips with the magnetometer part of the orientation sensor (azimuth). The sensor also senses the phone's position in space. It can do this because there's a force (gravity) acting on the phone that you can use to work out which way is up, down, or somewhere in between. The phone uses three accelerometers to detect gravity in three dimensions. These three readings are combined to give you two basic measures of movement: *pitch* and *roll* (see the following diagram). The orientation sensor can also combine pitch and roll into *angle* and *magnitude*, which provides the direction and speed of an imaginary ball balanced on the phone's screen. Here's a summary of all five readings that the orientation sensor can provide:



Hungry Spider app

PURPOSE OF THIS APP

This app uses the Creepy Spider app from chapter 8 and turns it into a simple fly-catching game. You control the spider by tilting the phone and try to gobble up all the flies as quickly as possible.

APP RATING

3

ASSETS YOU'LL NEED

Images: Fly.gif and Spider_web.png. A working version of the Creepy Spider app.

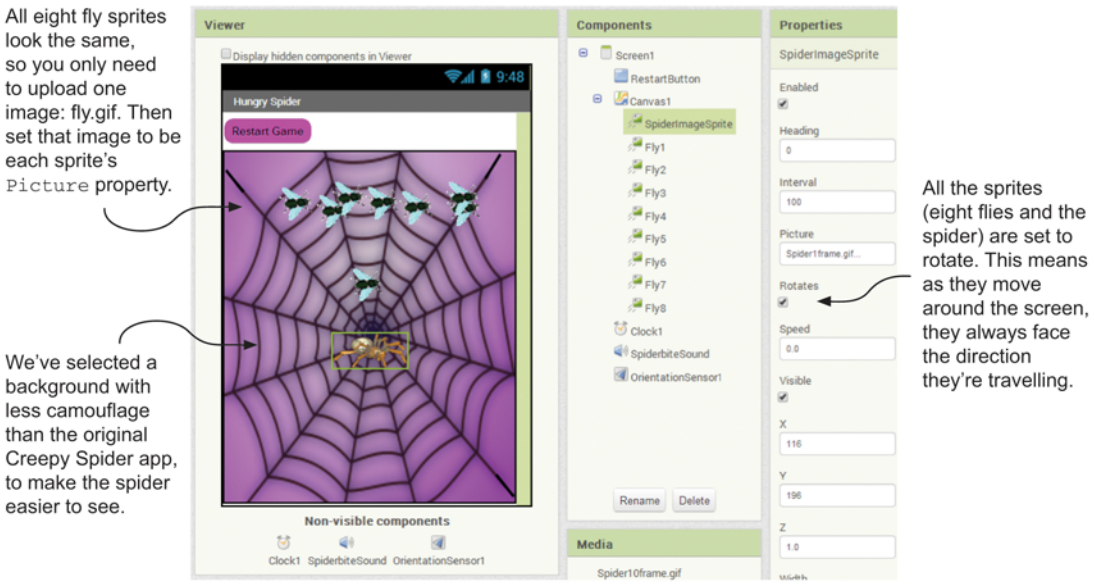
Open your Creepy Spider app, and save it as Hungry Spider. Make the following changes and additions to your project.

Hungry Spider			
Screen1 properties	**Title**: Hungry Spider **ScreenOrientation**: Portrait **Scrollable**: No (unselected)		
Components	**What do I rename it?**	**What does it do?**	**What properties do I set?**
Button	**RestartButton**	Causes any eaten flies to reappear so the game can begin again.	**Text**: "Restart Game" **BackgroundColor**: Magenta **FontBold**: Yes (selected) **Shape**: Rounded
OrientationSensor Palette group: Sensors	**OrientationSensor1**	Detects how the phone is tilted—the spider always heads toward the ground. The bigger the angle of tilt, the faster the spider crawls.	**Enabled**: Yes (selected)
Canvas	**Canvas**	The play area that displays a spiderweb background.	**Width** and **Height**: Fill Parent **BackgroundImage**: SpiderWeb.png
8 **ImageSprites**	**Fly1**, **Fly2** … up to **Fly8**	Display eight juicy flies, which move around the web.	For all eight sprites: **Picture**: fly.gif **Rotates**: Yes (selected) **Width** and **Height**: Automatic

1. Setting up the screen

We've made some changes to the original Creepy Spider screen. The leaves that provided camouflage for the spider have been replaced by a more neutral background and a spiderweb. This helps the player see the spider and makes it look like the flies are trapped in a web.

All the sprites will move—the flies will move randomly, bouncing off the screen edges, and the spider will always head toward the point of the screen that is closest to the ground (using the orientation sensor to detect gravity). The game will look much better if all the sprites face the same direction they're moving, so for all sprites you'll set the `Rotates` property to `true` by selecting the `Rotates` box. Here's the layout:

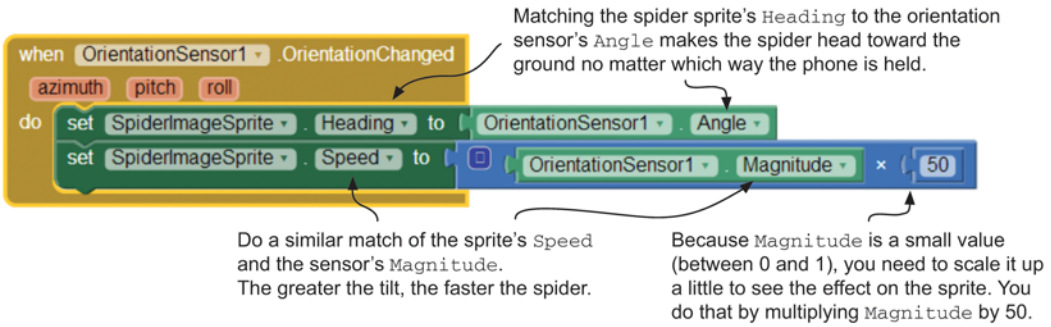


2. Coding the blocks: moving the spider

First you'll get the spider moving around the web in the direction that you tip the phone—later you'll introduce the flies. The orientation sensor provides two useful readings to help you here:

- *Angle*—Tells you which direction a ball would roll if you balanced it on the phone's screen.
- *Magnitude*—Tells you how far the phone has been tipped. A value of 0 means the phone is lying flat (parallel to the ground), and a value of 1 means the phone is tipped onto one of its edges.

You know from previous apps that **`Sprite`**s have **`Heading`** and **`Speed`** properties. If you link a sprite's **`Heading`** to the orientation **`Angle`**, the sprite will move toward the ground (just like a ball balanced on the screen). If you link the sprite's **`Speed`** to the orientation **`Magnitude`**, the sprite will move more quickly as the phone is tipped further toward one of its edges. Here are the blocks to do just that with your hungry spider:



Just like in the Compass app, you're using the **`OrientationChanged`** event to trigger the spider's movement so that whenever the phone moves, the spider reacts right away.

Try the app right now. You should see an animated spider whose movement changes as you tilt the phone. Try getting it to walk slowly in a circle and then dash around the edges of the screen—you'll soon get the hang of it. Although you'll see the flies on the screen, the spider can't interact with them yet. It's time to add that part …

3. Freeing the flies

After all that running around, the spider must be hungry—let's animate some juicy flies for it to chomp. This is a two-step process:

1. Animate the flies so they move randomly around the web.
2. Detect when the spider collides with a fly. When this happens, play a spider-bite sound and make the fly disappear.

To animate the flies, you first need to set a random heading and speed for each of them. Doing this eight times (once for each fly) means a lot of blocks, but there is a handy shortcut you can take to reduce the blocks using a list. You know that lists can contain all kinds of useful items like text, numbers, and colors. Lists can also contain components from the Palette—and once you've filled a list with components, you can set the properties for all those components at once using a loop. Let's break that idea into simple steps.

Start by defining a variable containing an empty list. Call it **`FlyList`**, because it's going to contain all eight fly sprites.

When the app starts, you'll fill the list with your eight fly sprites using the **`Screen1.Initialize`** event. You'll find the **`Fly1`** to **`Fly8`** blocks at the bottom of each **`Fly`** sprite's list of blocks.

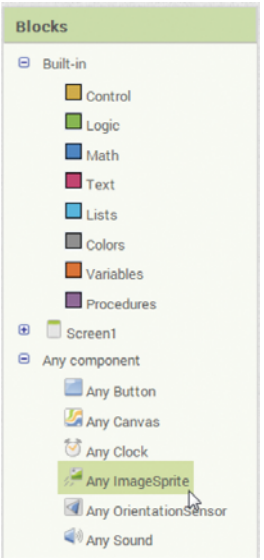
Normally when you make a list you fill it when you first define it, but a quirk of App Inventor means you have to do this two-step process for any list that contains components.

You have a list of sprite components—now what? Well, if you wrote down what you wanted to do in English, you would probably say something like, "For each sprite in the list, set its heading to a random number between 1 and 360 and set its speed to a random number between 2 and 5." That seems straightforward. You know there is a **`for each`** loop that works through a list, so you could do something like this:

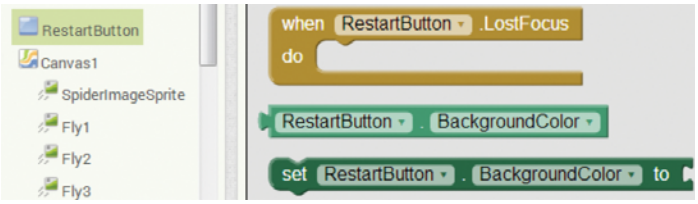
But there's a big problem with this block. Can you see what it is? Which flies get changed by this loop?

The answer is, only **`Fly1`** is changed. What you really want is to use the **`item`** variable of the loop to change which fly you're dealing with each time you go around the loop. That way, you set **`Fly1`**'s **`Heading`** and **`Speed`**, then **`Fly2`**'s, **`Fly3`**'s and so on.

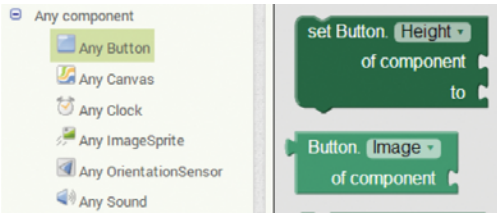
Fortunately, App Inventor has a built-in set of Any Component blocks you can use to program all the components in a list. It's at the bottom of the Blocks list. App Inventor automatically creates an Any Component for each of the types of objects you've added on the Design screen, so adding your first button to an app adds an Any Button category at the bottom of your blocks.



Open the Any Button blocks now, and look at the sort of blocks you have access to—how are they different from when you click a specific button's blocks, such as `RestartButton`?

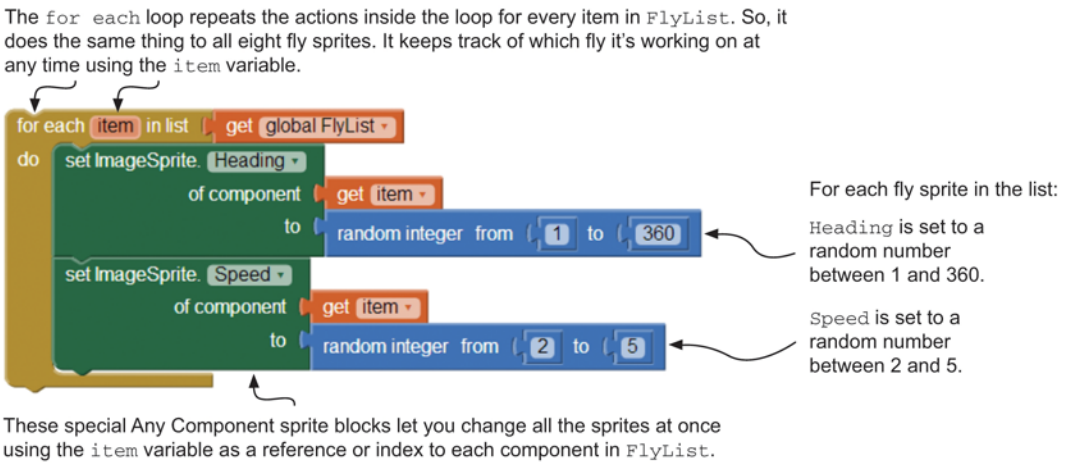


Specific component blocks like `RestartButton` include event blocks like `Button.Click` and blocks that let you find out and set specific properties of the component, such as its background color.



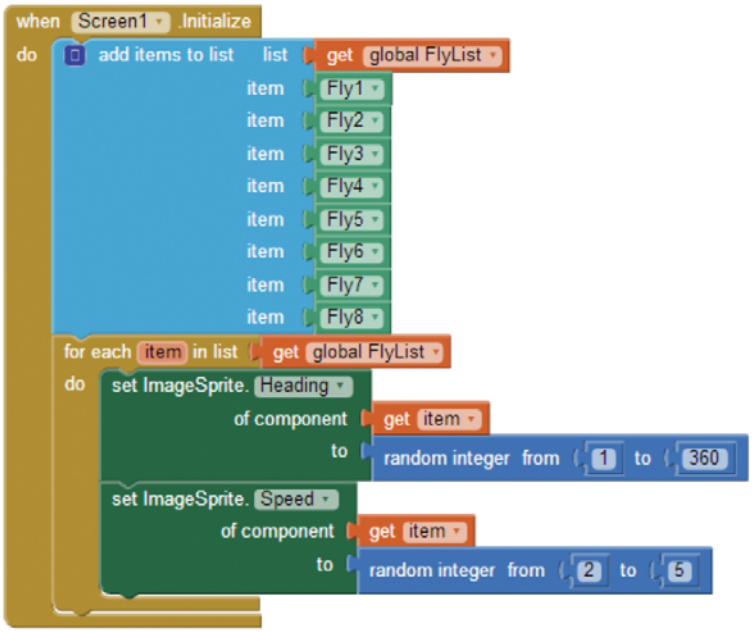
Looking at the Any Component blocks, you'll notice that there are no event blocks. That means it isn't possible to write an app that detects, for example, when any button is clicked. But you'll see that there are blocks for finding out and setting component properties. So you *can*, for example, set all your app buttons to be yellow or have a bold font. You'll also notice that all the blocks in Any Component have an extra component socket, and this is what you'll use in your `for each` loop.

Here's the new and improved `for each` block:



This might seem a bit complicated—why not forget the `for each` loop and write a separate couple of blocks for each fly? That would certainly work, but 2 blocks per fly equals 16 blocks, and you've just done the same thing with 3. Also, now you have the power to manipulate all the flies at once—you can have them flying in formation at the click of a button, or change their image so they all become butterflies at the end of the game. Any Component blocks are powerful!

You want all the flies' `Heading` and `Speed` properties to be set at the beginning of the app, so insert the `for each` loop into the `Screen1.Initialize` event block that you created earlier:



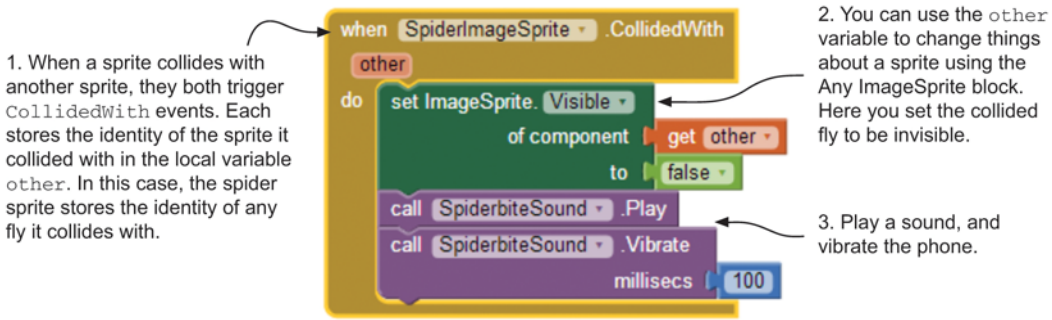
Now if you run the app you'll see that all the flies set off moving in different directions and at different speeds.

When flies reach the screen edge, though, they slide along it until they all end up stuck in a corner. To solve this, you can bounce a fly that reaches an edge so it heads back toward the center of the screen (just like you did with the rat in chapter 7). Unfortunately, detecting when a sprite reaches the edge of the screen is an event—and you can't use Any Components for events—so in this case you need eight sets of blocks:

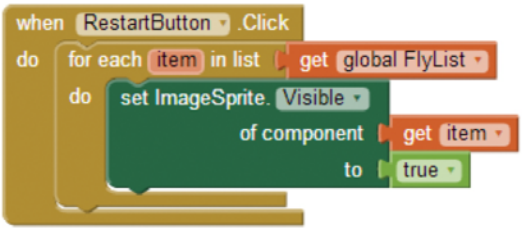
You can speed up the process of making these blocks by creating the first set of blocks, copying and pasting it seven times, and then changing the fly sprite referred to in the `EdgeReached` event and `Bounce` procedure of each block.

4. Feeding the spider

You have a controllable spider and moving flies. Now you need to say what should happen when they run into each other. When flies collide with other flies, they'll pass over each other. But if the spider collides with a fly, you want to play a spider-bite sound and make the fly disappear. You can do this using the `SpiderImageSprite`'s `CollidedWith` event, just like you used in Cheeky Hamster in chapter 8. The difference here is that you're going to change the fly sprite that the spider collides with by setting its `Visible` property to `false`. Here are the blocks and explanation:



5. Restarting the game



Once the flies are eaten, the game is considerably less interesting. You need more flies, or you can make the invisible flies visible again. When the button at the top of the screen is clicked, you'll set each fly's `Visible` property back to `true` using a `for each` loop similar to the one you used to set the flies moving.

Taking it further

There are lots of ways you could improve this basic game or adapt the principles to make an entirely new game. Adapt the Hungry Spider app using one or more of these ideas:

1. Add a score counter that increases when a fly is eaten.
2. Add a countdown timer so the spider has to eat the flies before the time runs out.
3. Reverse the spider's direction so it always heads up the screen, away from the ground.
4. Make the game harder by making the spider slower, or make the spider and flies smaller. You could add levels so the game starts easy and gets harder whenever the game restarts.
5. *Extra challenge*—Can you make the spider turn left and right rather than running in a circle so that it ends up upside down? You'll need to create a new set of spider sprites by flipping the current sprites in a program like Paint. Then you'll need to detect the roll of the phone—if it's to the left, use the left sprites, and if it's to the right, use the original (right-facing) sprites.

What did you learn?

In this chapter, you learned the following:

- That smartphones contain sensors that tell you about their physical position in the world
- That the orientation sensor can tell you
  - The compass direction the phone is pointing (using the azimuth)
  - The position of the phone in relation to the ground (using angle, magnitude, roll, and pitch)
- That when the phone moves, the orientation sensor triggers an event called `OrientationChanged`

- That buttons can tell the difference between a short click and a long click
- How to resize objects like sprites so they fill the phone's screen
- How to rotate sprites so they turn to face the direction they're travelling in
- How to set the properties of lots of objects at the same time using a `for each` loop and Any Component blocks

Test your knowledge

1 Which orientation sensor (azimuth, angle, magnitude, roll, or pitch) would be best to use for each of these apps?

a A driving game, where tilting the phone left and right steers the car

b A game where you steer a ball through a maze of obstacles

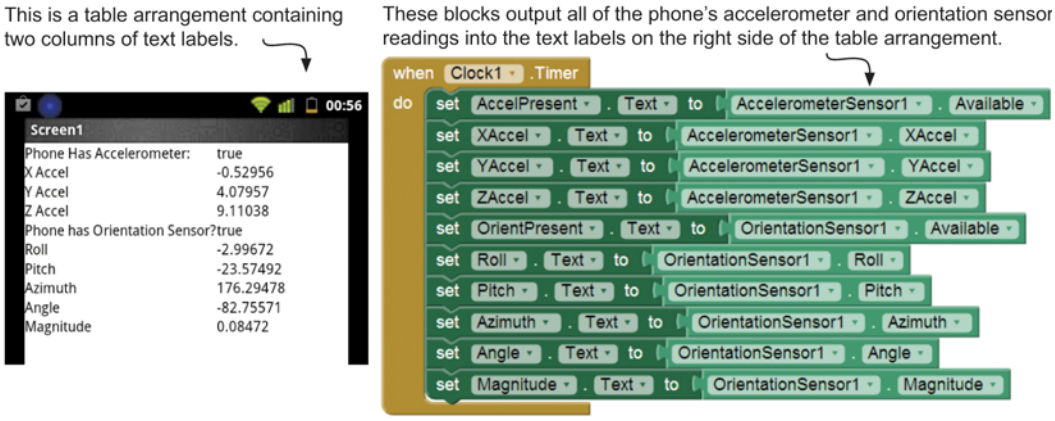
c A "spin the phone" app that counts how many times the phone can be spun around on a table top

d Is each of the following statements True or False? You can use Any Component blocks to

- Set the size of all buttons on the screen
- Play a sound whenever any text box contains the word *beep*
- Change the color of all labels to green
- Display an alert notification whenever a button is clicked for a long time

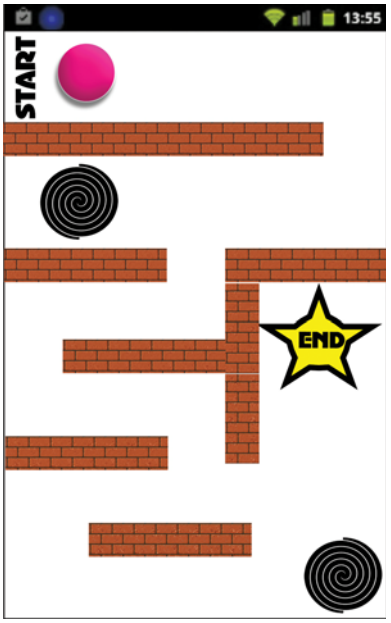
Try it out

1 In addition to the orientation sensor, phones also have a (related) accelerometer sensor. Create a Sensor Test app that outputs all the phone's sensor readings so you can see how they change as the phone is moved around, like so:



2. Use the phone sensors to create a Smartphone Burglar Alarm app that works like this:
   a. The user types a password and clicks an Activate button to activate the alarm, after which they put down the phone.
   b. After 10 seconds, the phone activates the orientation sensor.
   c. If an `OrientationChanged` event happens, the phone gives the user 10 seconds to type their password and click a Deactivate button.
   d. If the password isn't entered correctly, the phone sounds an alarm until the password is correctly entered.
   e. *Extra challenge*—Have the phone send a text message like, "Help! I am being stolen!" to another smartphone.
   f. *Extra-extra challenge*—Have the text message include some location information about where the phone is at the moment.

3. Create a Maze Game like the one shown. The player steers a ball from a START position through a maze of walls and holes to an END star. The ball, walls, and holes are all separate sprites. The rules of the game are as follows:



   a. The ball starts at a specific x, y position (hold this in a variable).
   b. The ball rolls toward the ground (just like the spider in Hungry Spider).
   c. If the ball collides with a wall, it bounces off (or, to make the game really hard, you could have it return to the START position).
   d. If the ball collides with a hole, it returns to the START position.
   e. If the ball collides with the END star, the game displays a "Congratulations!" message.

   You could extend the game by adding scores and levels.

MOBILE/ANDROID/KIDS

Hello App Inventor!

Android programming for kids and the rest of us

Paula Beer and Carl Simmons

Have you ever wondered how apps are made? Do you have a great idea for an app that you want to make reality? This book can teach you how to create apps for any Android device, even if you have never programmed before. With App Inventor, if you can imagine it, you can create it. Using this free, friendly tool, you can decide what you want your app to do and then click together colorful jigsaw-puzzle blocks to make it happen. App Inventor turns your project into an Android app that you can test on your computer, run on your phone, share with your friends, and even sell in the Google Play store.

Hello App Inventor! introduces young readers to the world of mobile programming. It assumes no previous experience. Featuring more than 30 invent-it-yourself projects, this book starts with basic apps and gradually builds the skills you need to bring your own ideas to life. We've provided the graphics and sounds to get you started right away. And a special Learning Points feature connects the example you're following to important computing concepts you'll use in any programming language.

WHAT'S INSIDE

- Covers MIT App Inventor 2
- How to create animated characters, games, experiments, magic tricks, and a Zombie Alarm clock
- Use advanced phone features like:
  - Movement sensors
  - Touch screen interaction
  - GPS
  - Camera
  - Text
  - Web connectivity

App Inventor is developed and maintained by **MIT**

"A great way to introduce kids to mobile programming."

—Ron Sher, Totango

"A must-have for both novice and experienced alike."

—Phanindra V Mankale, BanyanLeaf Technologies

"Fun, interesting, addictive. Enjoy!"

—Andrei Bautu, Romanian Naval Academy

"Stands out as a successful way to make complex technologies accessible to younger readers."

—Jacqueline Wilson, Avon Grove Charter School

Paula Beer and *Carl Simmons* are professional educators and authors who spend most of their time training new teachers and introducing children to programming.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/HelloAppInventor

ISBN-13: 978-1617291432
ISBN-10: 1617291439
53999
9 781617 291432

MANNING US $39.99 | Can $41.99