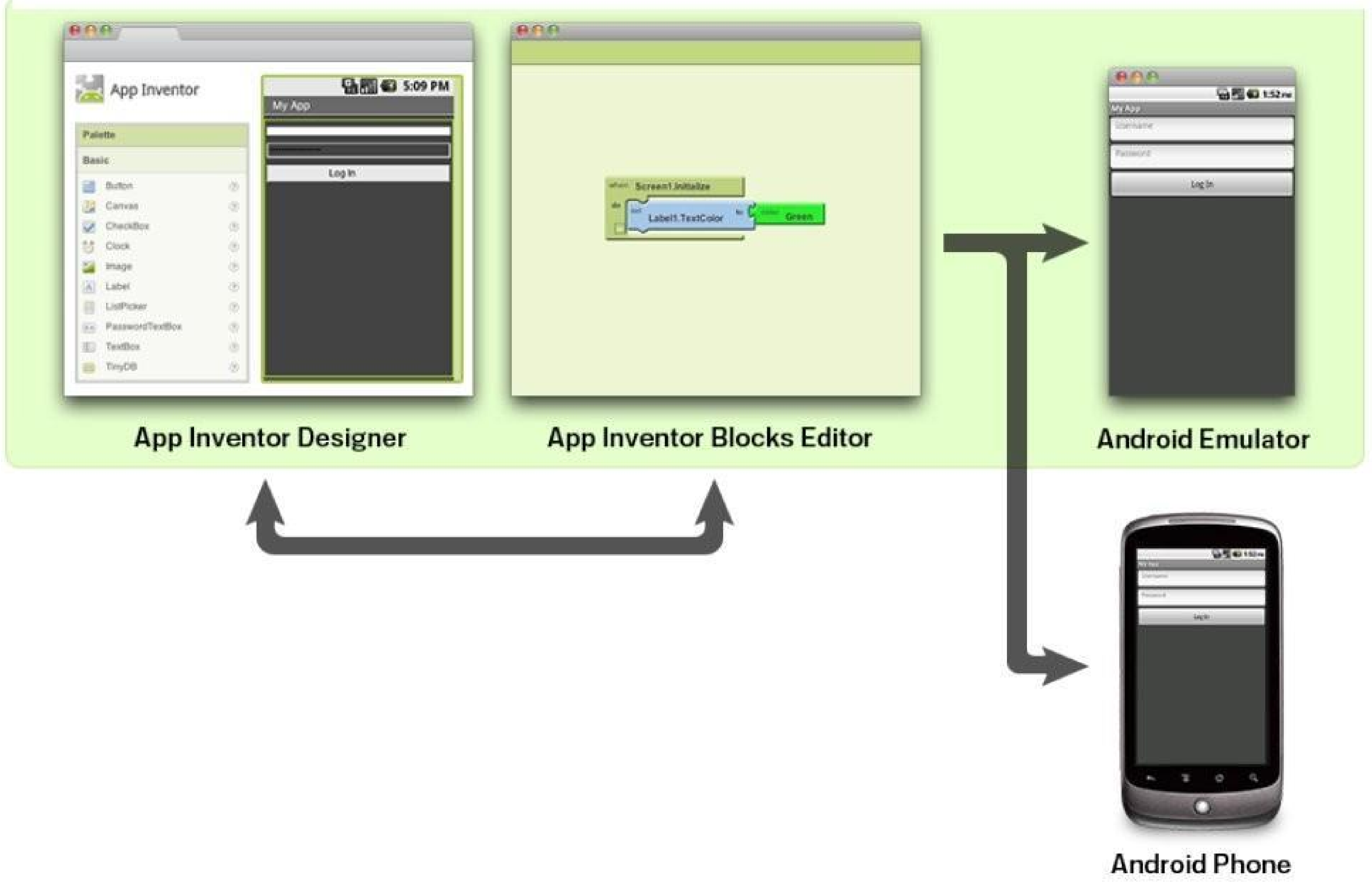
MIT App Inventor Getting Started Guide

What is App Inventor?

App Inventor lets you develop applications for Android phones using a web browser and either a connected phone or an on-screen phone emulator. The MIT 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. 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.

The App Inventor development environment is supported for Mac OS X, GNU/Linux, and Windows operating systems, and several popular Android phone models. Applications created with App Inventor can be installed on any Android phone.

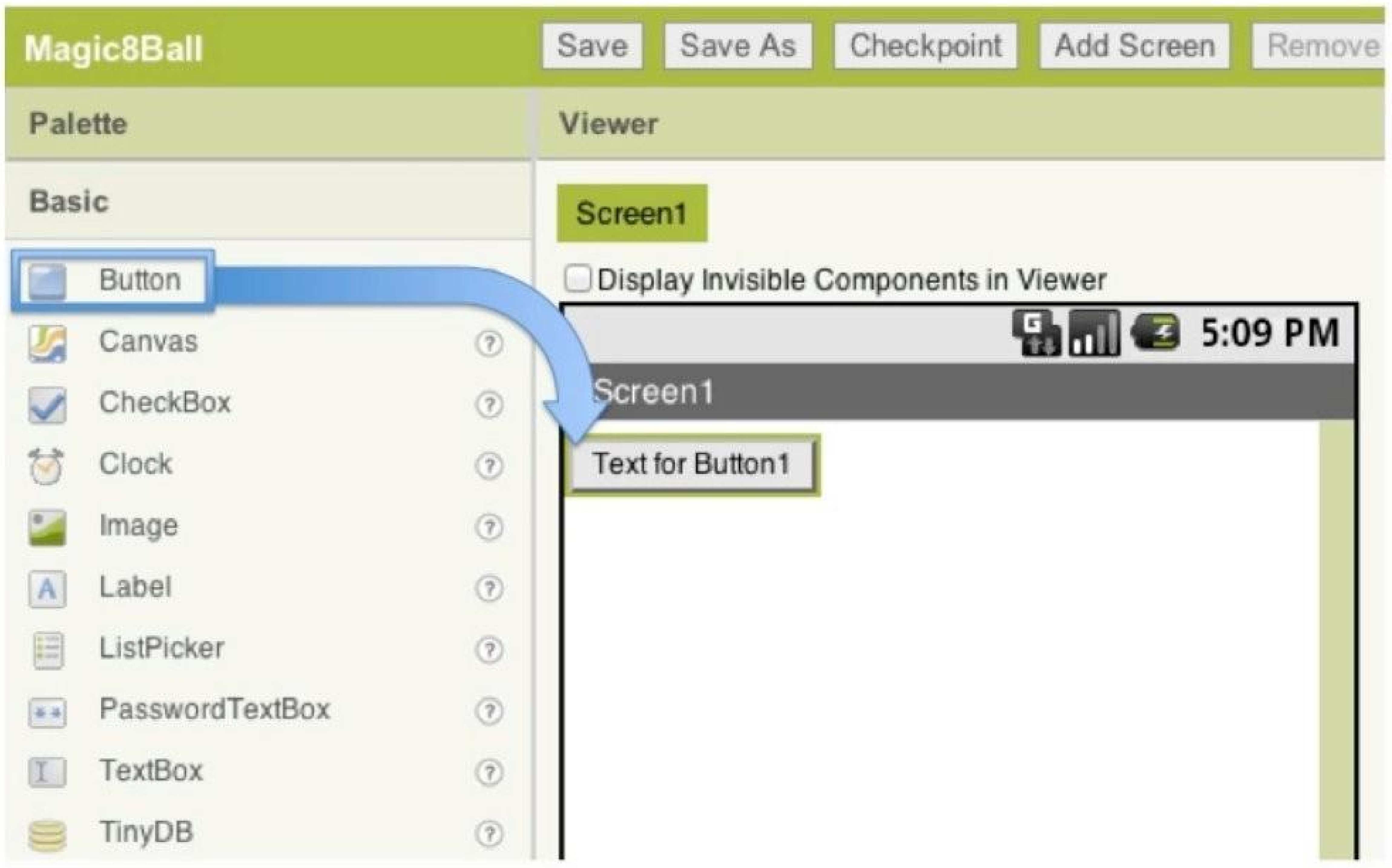
Before you can use App Inventor, you need to set up your computer and install the *App Inventor Setup* package on your computer. See: http://explore.appinventor.mit.edu/content/setup

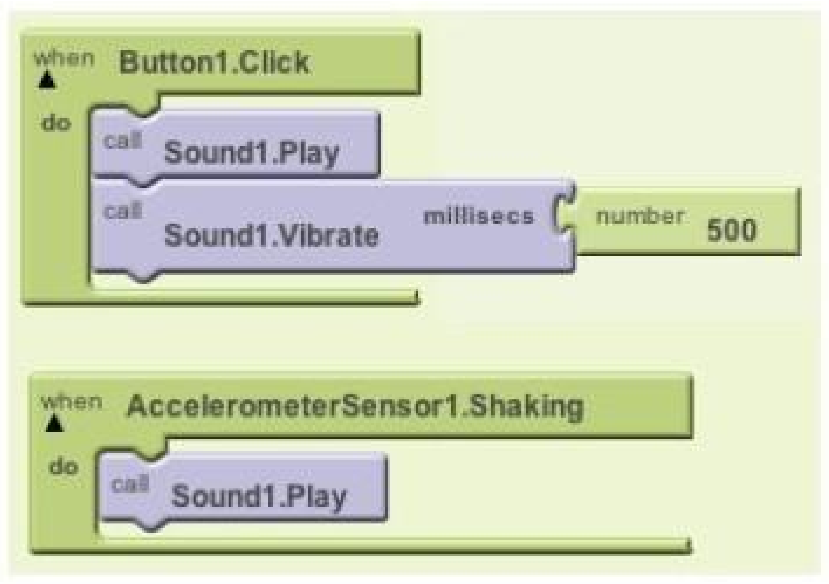
What can I do with App Inventor?

You can build many different types of apps with App Inventor. Often people begin by building games like MoleMash or games that let you draw funny pictures on your friend's faces. You can even make use of the phone's sensors to move a ball through a maze based on tilting the phone.

But app building is not limited to simple games. You can also build apps that inform and educate. You can create a quiz app to help you and your classmates study for a test. With Android's text-to-speech capabilities, you can even have the phone ask the questions aloud.

To use App Inventor, you do not need to be a professional developer. This is because instead of writing code, you visually design the way the app looks and use blocks to specify the app's behavior.




```
when Button1.Click
do  call Sound1.Play
    call Sound1.Vibrate millisecs  number 500

when AccelerometerSensor1.Shaking
do  call Sound1.Play
```

The App Inventor team has created blocks for just about everything you can do with an Android phone, as well as blocks for doing "programming-like" stuff-- blocks to store information, blocks for repeating actions, and blocks to perform actions under certain conditions. There are even blocks to talk to services like Twitter.

Simple but Powerful!

App Inventor is simple to use, but also very powerful. Apps you build can even store data created by users in a database, so you can create a make-a-quiz app in which the teachers can save questions in a quiz for their students to answer.

Because App Inventor provides access to a GPS-location sensor, you can build apps that know where you are. You can build an app to help you remember where you parked your car, an app that shows the location of your friends or colleagues at a concert or conference, or your own custom tour app of your school, workplace, or a museum.

You can write apps that use the phone features of an Android phone. You can write an app that periodically texts "missing you" to your loved ones, or an app "No Text While Driving" that responds to all texts automatically with "sorry, I'm driving and will contact you later". You can even have the app read the incoming texts aloud to you (though this might lure you into responding).

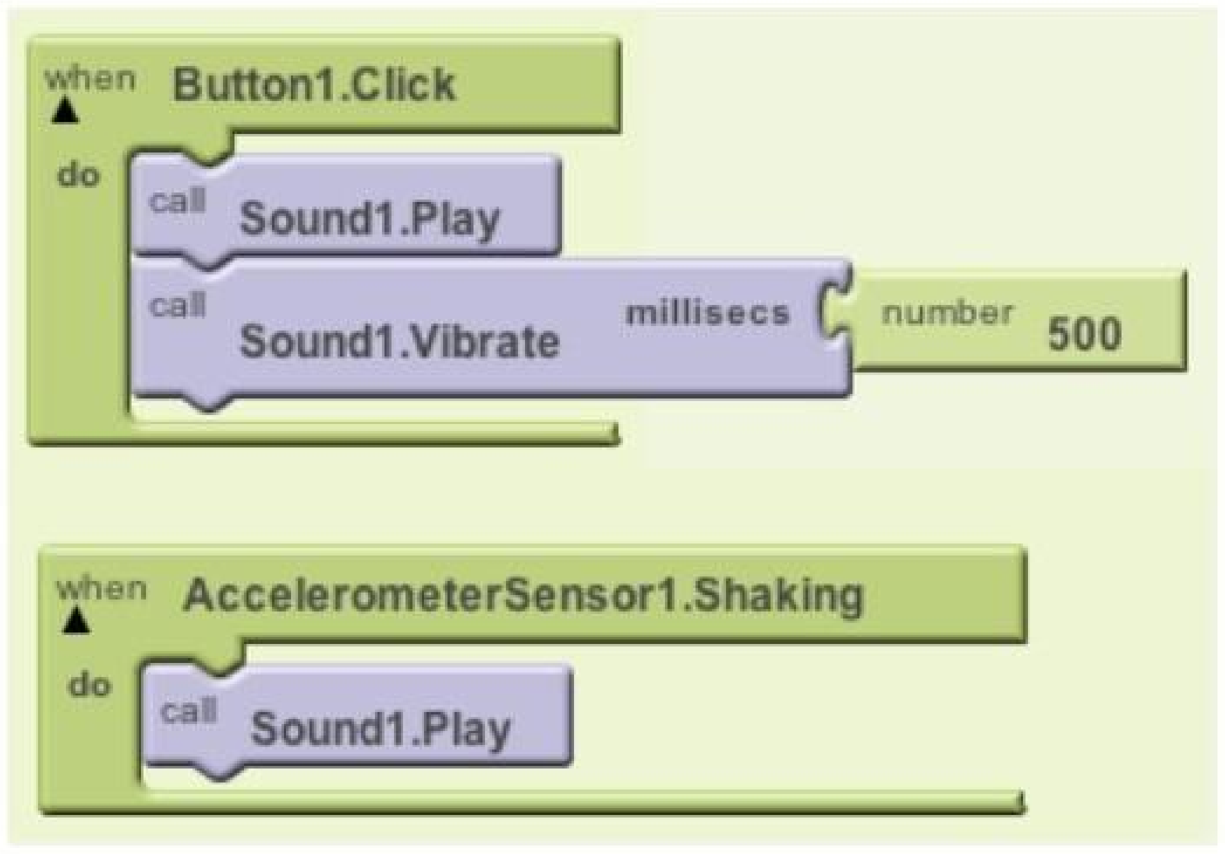
App Inventor provides a way for you to communicate with the web. If you know how to write web apps, you can use App Inventor to write Android apps that talk to your favorite web sites, such as Amazon and Twitter.

Understanding App Inventor Programming

Event Handlers

App Inventor programs describe how the phone should respond to certain events: a button has been pressed, the phone is being shaken, the user is dragging her finger over a canvas, etc. This is specified by **event handler** blocks, which used the word when. E.g., when Button1.Click and when AccelerometerSensor1.Shaking in HelloPurr.

Most event handlers are in green color and stored at the top part of each drawer. Here are the example of event handlers.

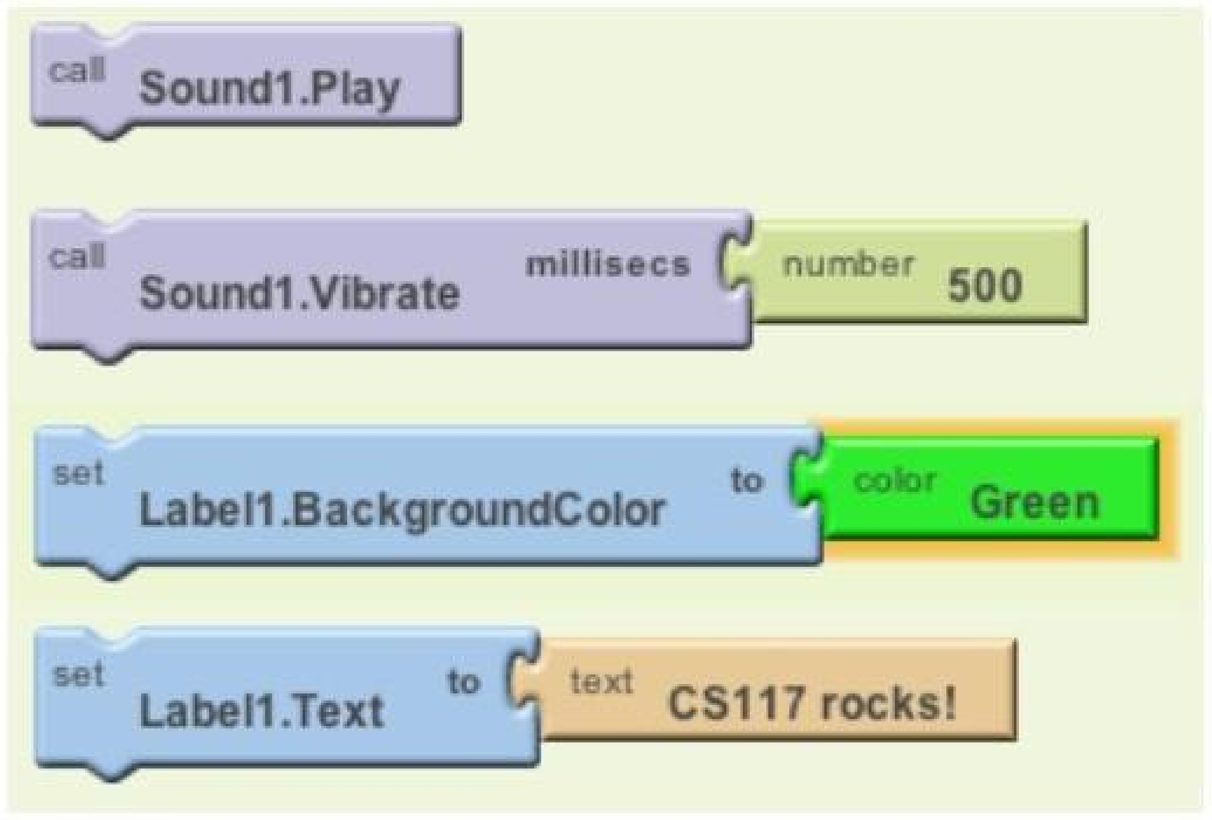


When an event occurs on a phone, the corresponding event handler is said to **fire**, which means it is executed.

Commands and Expressions

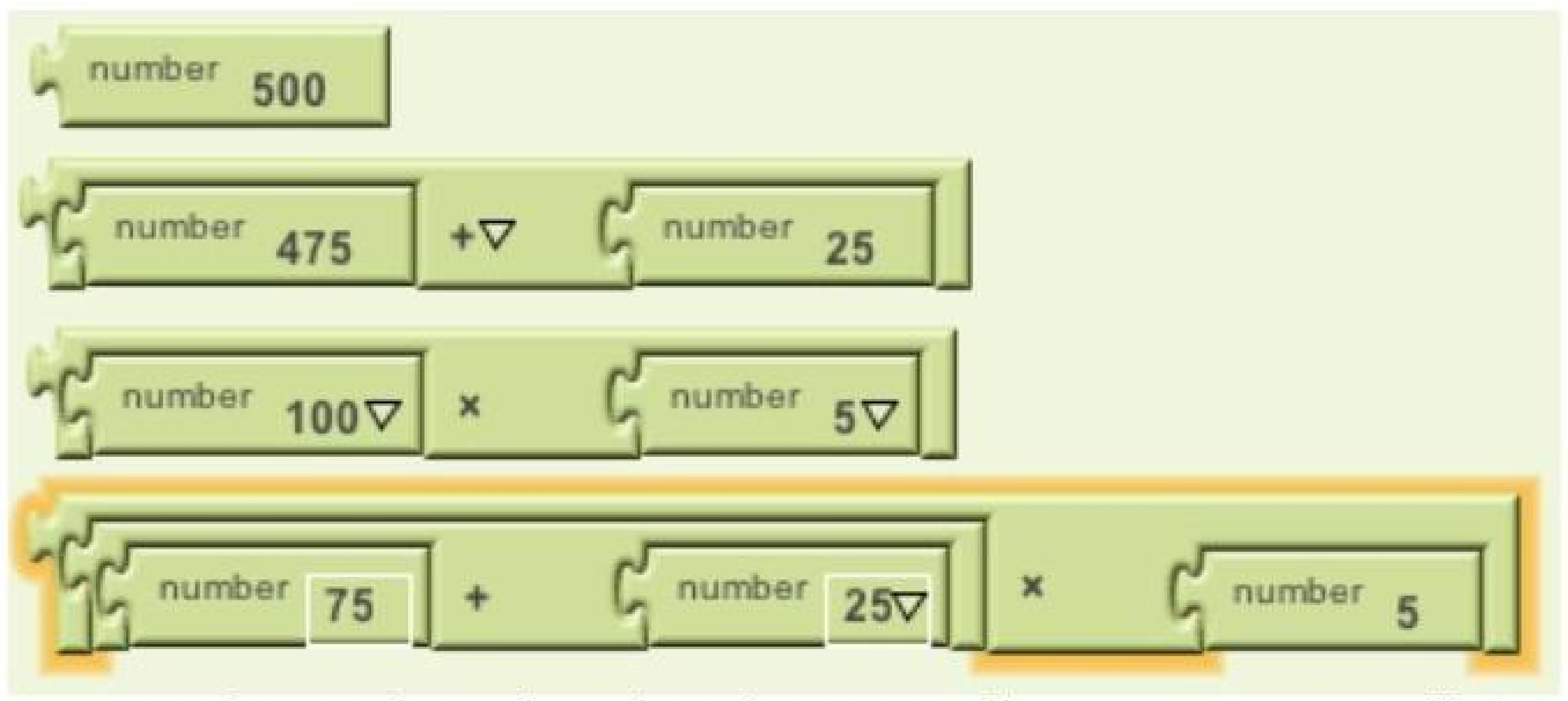
When an event handler fires, it executes a sequence of commands in its **body**. A **command** is a block that specifies an action to be performed on the phone (e.g., playing sounds). Most command blocks are in purple or blue color.

Here are some sample commands available in HelloPurr:

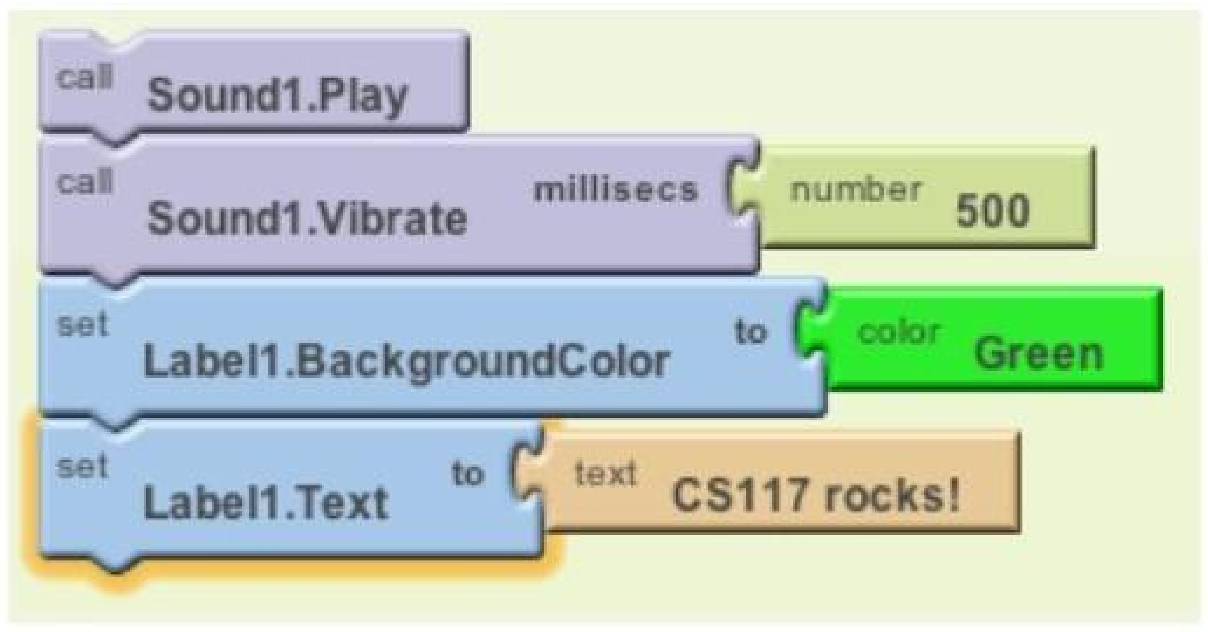


Some commands require one or more **input values** (also known as **parameters** or **arguments**) to completely specify their action. For example, call Sound1.Vibrate needs to know the number of milliseconds to vibrate, set Label1.BackgroundColor needs to know the new background color of the label, and set Label1.text needs to know the new text string for the label. The need for input values is shown by sockets on the right edge of the command.

These sockets can be filled with **expressions**, blocks that denote a value. Expression blocks have **leftward-pointing plugs** that you can imagine transmit the value to the socket. Larger expressions can be built out of simpler ones by horizontal composition. E.g., all of the following expressions denote the number 500:



Commands are shaped so that they naturally compose vertically into a **command stack**, which is just one big command built out of smaller ones. Here's a stack with four commands:


call Sound1.Play
call Sound1.Vibrate millisecs number 500
set Label1.BackgroundColor to color Green
set Label1.Text to text CS117 rocks!

When this stack of commands are placed in a body of an event handler (e.g., the ***when.Button1.Click*** event handler), the command will be executed from the top to the bottom. If the stack of command above is executed, then the phone will first play the sound, then vibrate, then change the label's color to be green, and then label will show the text "CS117 rocks!"

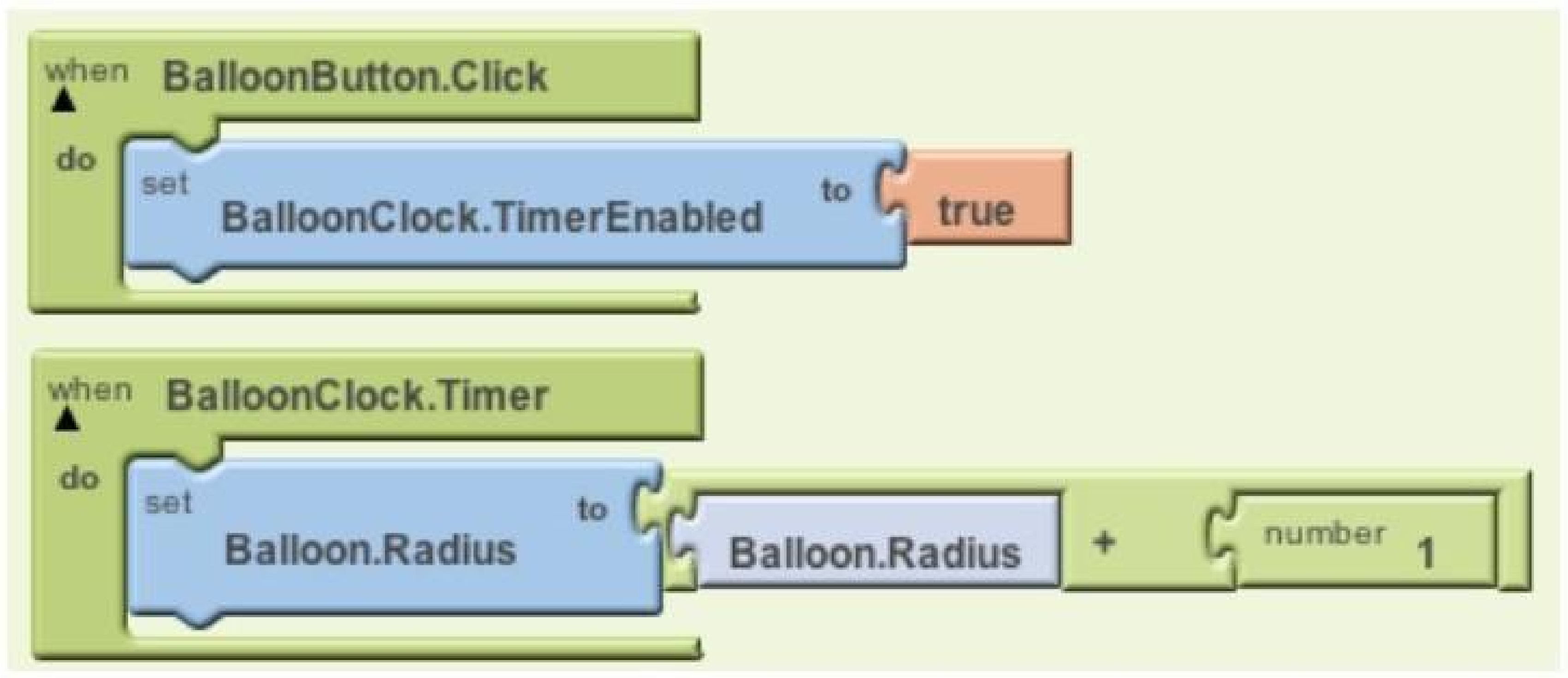
However, the execution works very fast: you would see all the actions happen at the same time.

Control Flow

When an event handler fires, you can imagine that it creates a karaoke-like **control dot** that flows through the command stack in its body. The control dot moves from the top of the stack to the bottom, and when it reaches a command, that command is **executed** -- i.e, the action of that command is performed. Thinking about control "flowing" through a program will help us understand its behavior.

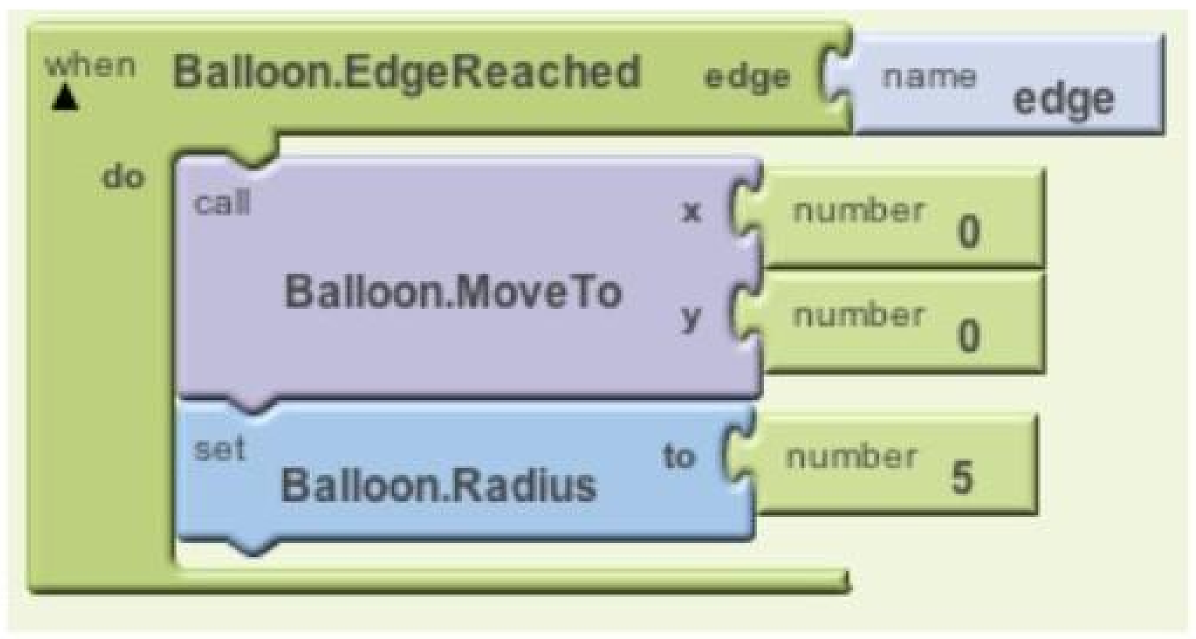
The order of the commands, or the ***control flow*** is important when you make an app. You need to make sure which action should come first.

We can get the effect of an inflating red balloon by having a red ball whose radius grows every time an associated timer fires:



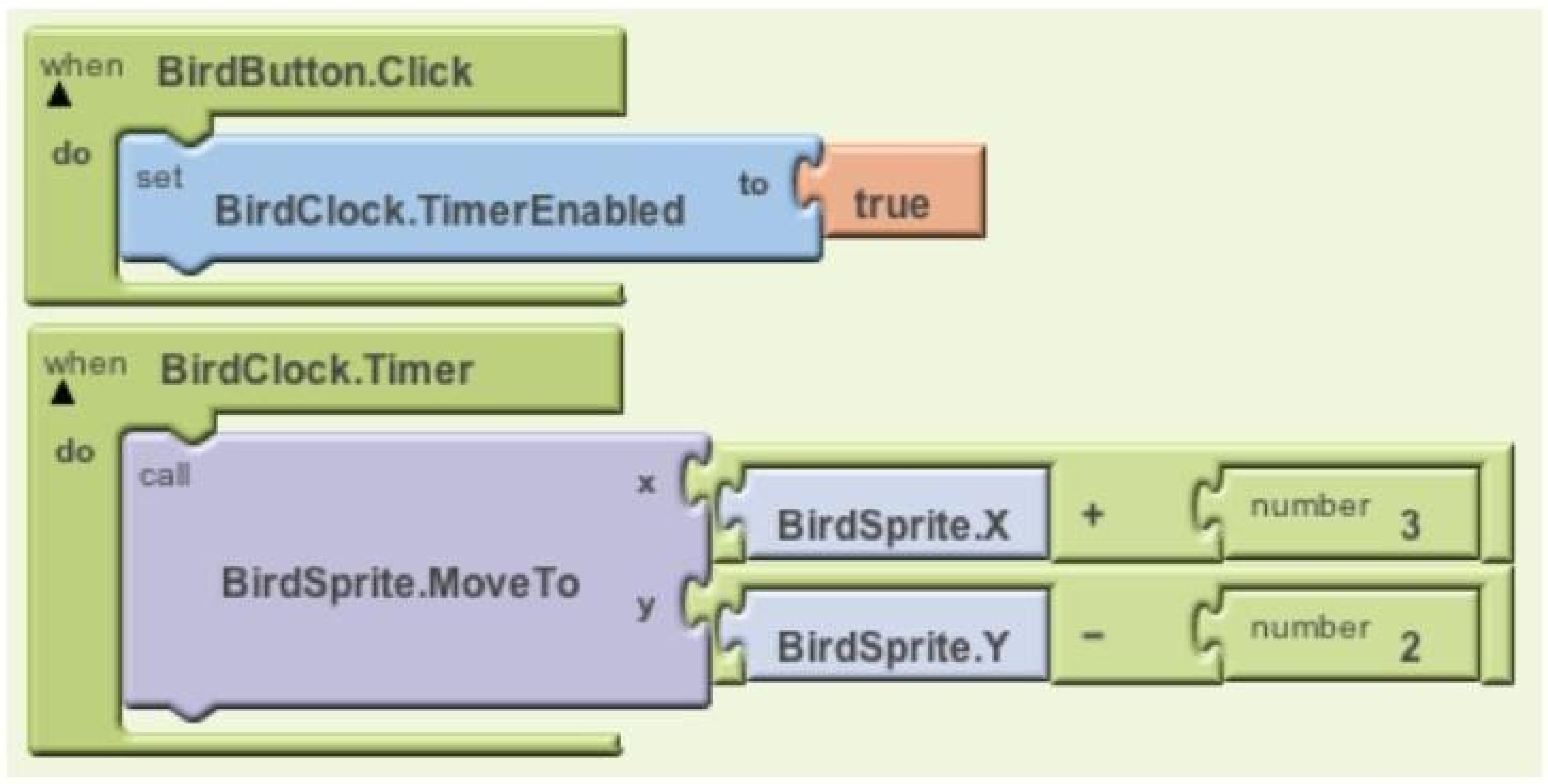
```
when BalloonButton.Click
do  set BalloonClock.TimerEnabled to true

when BalloonClock.Timer
do  set Balloon.Radius to Balloon.Radius + number 1
```

We can specify that the balloon "pops" when it hits the edge of the canvas by resetting it to its initial radius.



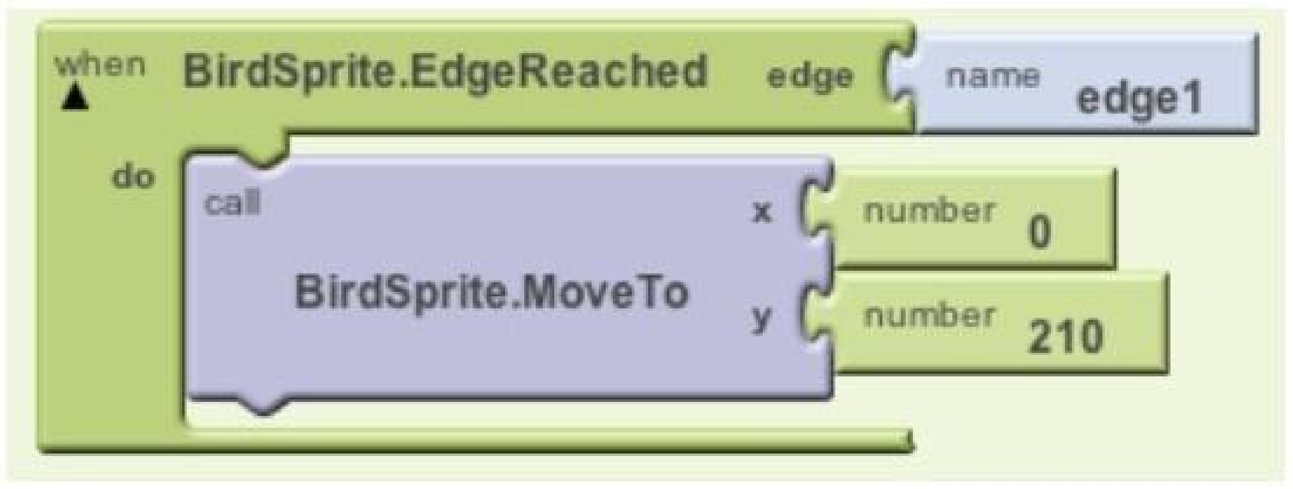
```
when Balloon.EdgeReached edge name edge
do  call Balloon.MoveTo x number 0
                        y number 0
    set Balloon.Radius to number 5
```

We can add a bird sprite that "flies" from the lower left to upper right corner of the canvas. Here are the blocks that specify the flight:

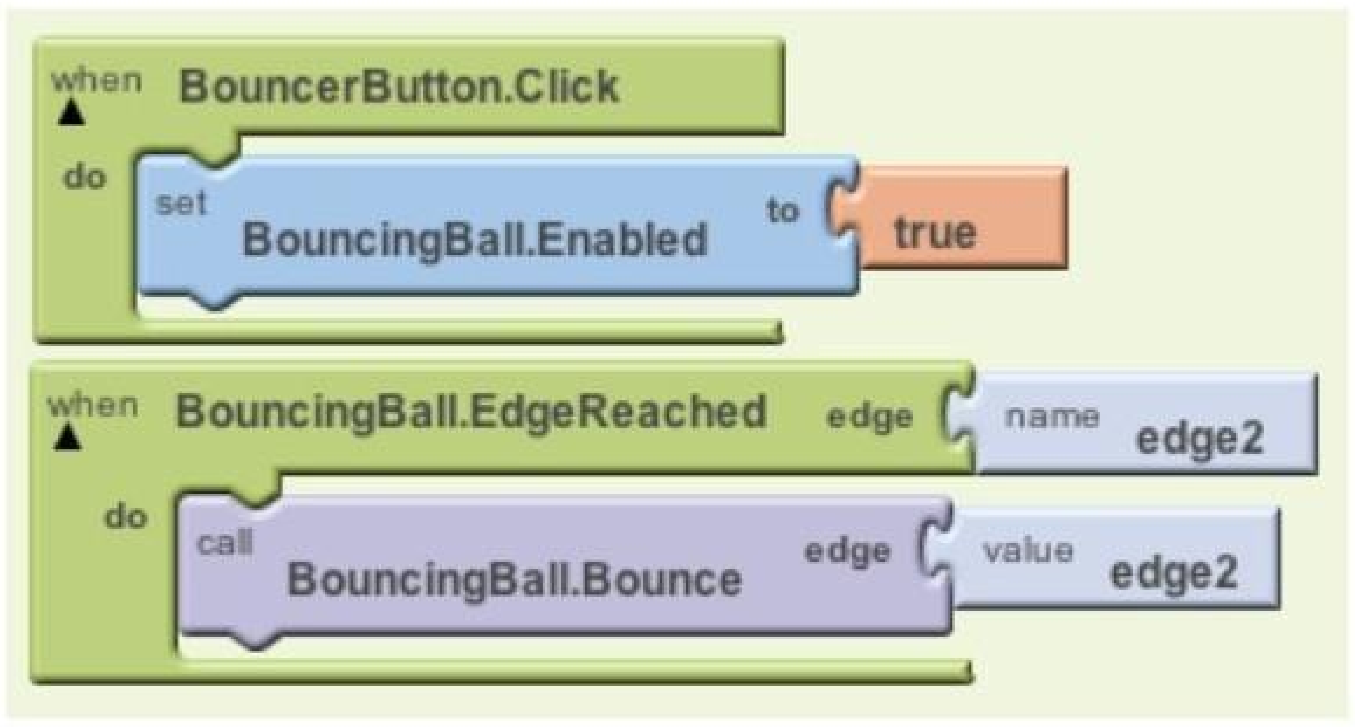


```
when BirdButton.Click
do  set BirdClock.TimerEnabled to true

when BirdClock.Timer
do  call BirdSprite.MoveTo x BirdSprite.X + number 3
                           y BirdSprite.Y - number 2
```

When the bird hits an edge, we reset it to its initial position:


when BirdSprite.EdgeReached edge name edge1
do call BirdSprite.MoveTo x number 0 y number 210

Every sprite (including balls) come equipped with their own internal timer and properties that control their speed and heading. Once we've specified the speed and heading of the yellow BouncingBall in TimerTest, here are the only blocks we need to turn it on and make it bounce:


when BouncerButton.Click
do set BouncingBall.Enabled to true
when BouncingBall.EdgeReached edge name edge2
do call BouncingBall.Bounce edge value edge2