Fuente: www.appinventor.org
Traducción hecha con Google Traductor y mejorada por mi: piatticarlos@gmail.com

Creación de aplicaciones animadas

En este capítulo se analizan los métodos para la creación de aplicaciones con animaciones simples (objetos que se mueven). Usted aprenderá los fundamentos de crear juegos bidimensionales con App Inventor, a entender los ImageSprites (duendes con imágenes)
y el manejo de eventos tales como dos objetos que chocan.

Cuando usted ve un objeto que se mueve suavemente a lo largo de la pantalla del ordenador, lo que realmente está viendo es una rápida sucesión de imágenes con una pequeña diferencia de posición cada una. Es una ilusión no muy diferente a la de "flipbooks" (folioscopio), en la que se ve una imagen en movimiento por pasar rápidamente a través de las páginas (y es también algo parecido a como se hacen las películas animadas más sofisticadas!).

Con App Inventor, podrá definir la animación mediante la colocación de objetos dentro de un componente Canvas (lienzo) y mover aquellos objetos por el lienzo a través del tiempo. En este capítulo, usted aprenderá cómo funciona el sistema de coordenadas para el Canvas, cómo usar el evento ***Clock.Timer*** *para activar un movimiento, la forma de controlar la velocidad de objetos, y la respuesta a eventos tales como dos objetos chocandose.*

Agregar un componente Canvas a su aplicación

Puede arrastrar un componente *Canvas* a su aplicación desde la paleta básica. Después de arrastrar hacia fuera, especifique el ancho y alto del *Canvas*. A menudo, usted desea que abarque todo el ancho de la pantalla del dispositivo, para ello, cuando se especifica el ancho seleccione "Fill Parent", como se muestra en la Figura 17-1.

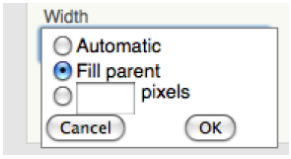

Figure 17-1. Setting the Canvas's Width to span the screen

También puede hacer lo mismo con la altura, pero generalmente la establece con un número (por ejemplo, 300 píxeles) así puede dejar espacio para otros componentes por encima y por debajo.

El sistema de coordenadas del Canvas

Un dibujo sobre un *Canvas (lienzo)* es realmente una tabla de píxeles, donde un píxel es el punto de color más pequeño que puede modificarse en el teléfono (u otro dispositivo). Cada píxel tiene una ubicación (o celda de la tabla) en el *Canvas*, que se define por un sistema de coordenadas x-y, como se ilustra En la Figura 17-2. En este sistema de coordenadas, “x” define

una ubicación en el plano horizontal (de izquierda a derecha), y “y” define una localización en el plano vertical (de arriba hacia abajo).

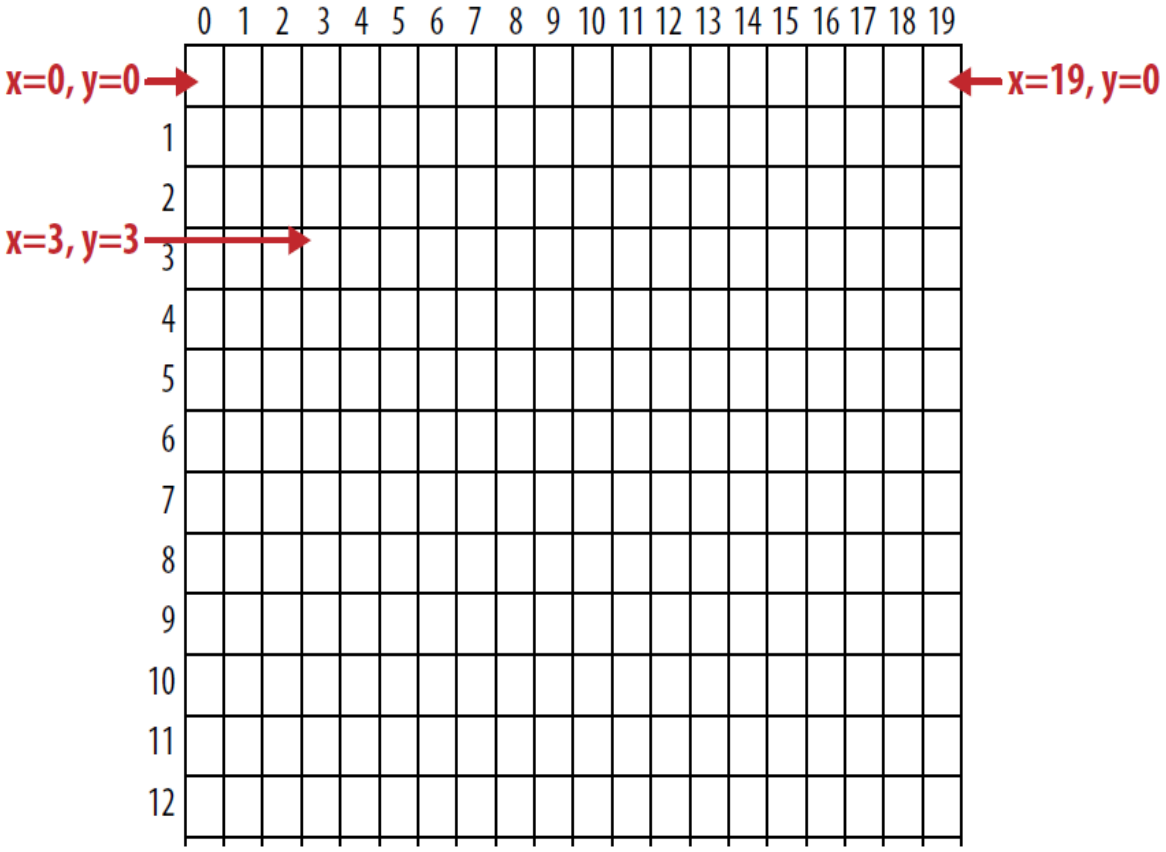

Figure 17-2. The Canvas coordinate system

Puede parecer un poco contradictorio, pero la celda superior izquierda comienza en 0 para las dos coordenadas, por lo que esta posición se representa como (x = 0, y = 0). (Esto es diferente al índice que se utiliza en App Inventor para las listas, que se inician con el valor 1, aparentemente más normal) A medida que se desplaza hacia la derecha, la coordenada “x” va creciendo, mientras que si se desplaza hacia abajo, la coordenada “y” es la que aumenta. La celda inmediatamente a la derecha de la esquina superior izquierda es (x = 1, y = 0). La esquina superior derecha tiene una coordenada “x” igual al ancho del Canvas menos 1. La mayoría de las pantallas de los teléfonos tienen una ancho cercano a 300, pero para el Canvas del ejemplo que se muestra aquí, el ancho es de 20, por lo que la esquina superior derecha es la coordenada (x = 19, y = 0).

Puede cambiar la apariencia del *Canvas* de dos maneras: (1) pintando sobre él, o (2) mediante la colocación y movimiento de objetos dentro de él. Este capítulo se centrará principalmente en la segunda forma, pero primero vamos a hablar de cómo "pintar" y cómo crear una animación pintando (este es también el tema de la aplicación PaintPot en el capítulo 2).

Cada celda del *Canvas* tiene un píxel que define el color que debe aparecer allí.

El componente Canvas proporciona los bloques **Canvas.DrawLine** y **Canvas.DrawCircle** para pintar píxeles en él. En primer lugar, establezca la propiedad Canvas.PaintColor al color que desee y luego enlazar a uno de los bloques de dibujo para dibujar en ese color. Con **DrawCircle**, puede pintar círculos de cualquier radio, pero si establece el radio en 1, como se muestra en la Figura 17-3, podrá pintar el trazo de un píxel individual.

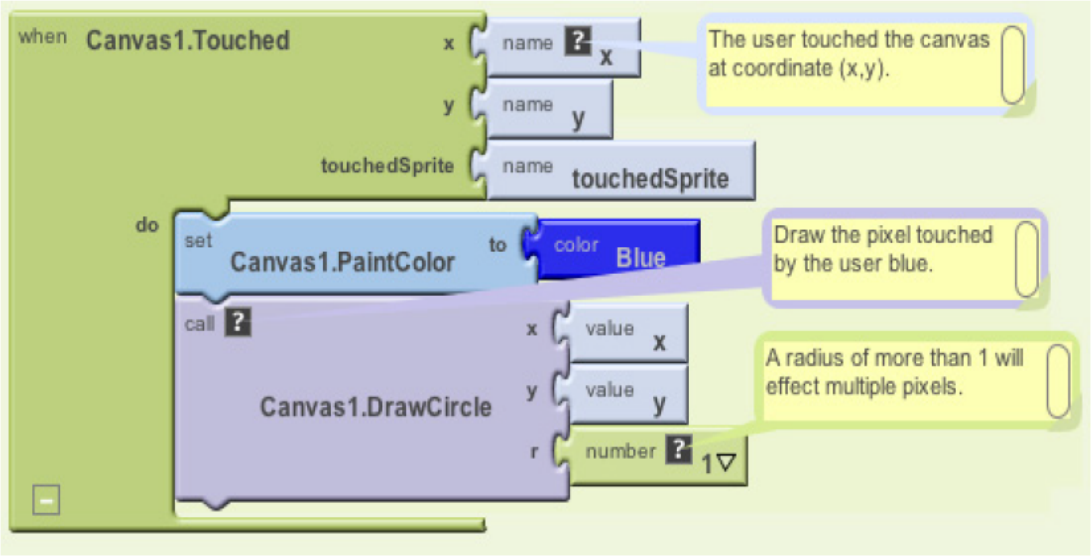

Figure 17-3. DrawCircle with radius 1 paints an individual pixel with each touch

App Inventor ofrece una paleta de 14 colores básicos que se pueden aplicar a los píxeles de pintura (o fondos de componente). Puede acceder a una gama más amplia de colores mediante el uso del Esquema de numeración de color que se detalla en la documentación de App Inventor en *http://beta.appinventor.mit.edu/learn/reference/blocks/colors.html http://beta.appinventor.mit.edu/learn/reference/blocks/colorchart.html*

La segunda manera de modificar la apariencia de un *Canvas* es colocar los componentes Ball e ImageSprite en el. Un *sprite* es un objeto gráfico colocado dentro de una escena más grande, en este caso, un Canvas. Tanto Ball como ImageSprite son *sprites*; solo son diferentes en apariencia. Un Ball es un círculo cuyo aspecto sólo se puede modificar cambiando su color o radio, mientras que un ImageSprite puede tomar cualquier aspecto según se define en un archivo de imagen que usted cargará. ImageSprite y Ball solamente se pueden agregar a un *Canvas*, no se puede arrastrar individualmente en la interfaz de usuario.

Animación de objetos con eventos de temporizador

Una forma de realizar la animación en App Inventor es cambiar un objeto en respuesta a un temporizador de eventos. Por lo general, podrás mover sprites a diferentes lugares del *Canvas* al establecer intervalos de tiempo. El uso de eventos de temporizador es el método más común de definir los intervalos de tiempo. Más tarde, también vamos a hablar de un método alternativo

de programar una animación utilizando los componentes Ball e ImageSprite y las propiedades Speed y Heading.

Los clics en botones y otros eventos iniciados por el usuario son fáciles de entender: el usuario hace algo, y la aplicación responde mediante la realización de algunas operaciones. Los eventos de temporizador son diferentes: no se activan por el usuario final, sino por el paso del tiempo. Usted tiene que conceptualizar la activación de los eventos mediante el reloj del teléfono en lugar de que lo realice el usuario

Para definir un evento de temporizador, primero arrastre un componente Clock a su aplicación en el Diseñador de componentes. El componente Clock tiene asociada una propiedad TimerInterval con el. El intervalo se define en términos de milisegundos (1/1000 de segundo). Si
se establece el TimerInterval a 500, significa que un evento de temporizador se activará cada medio segundo. Cuanto menor sea el TimerInterval, hará que el objeto se mueva más rápido.

Después de añadir un Clock (reloj) y establecer un TimerInterval en el Diseñador, puede arrastrar un evento **Clock.Timer** en el Editor de bloques. Usted puede poner todos los bloques que le gusten en este evento, y van a realizarse en cada intervalo de tiempo.

Crear Movimiento
Para mostrar un *sprite* moviéndose a través del tiempo, va a utilizar la función **MoveTo** que se encuentra en ambos componentes ImageSprite y Ball. Por ejemplo, para mover un Ball horizontalmente a través de la pantalla, tendrá que utilizar los bloques de la Figura 17-4.

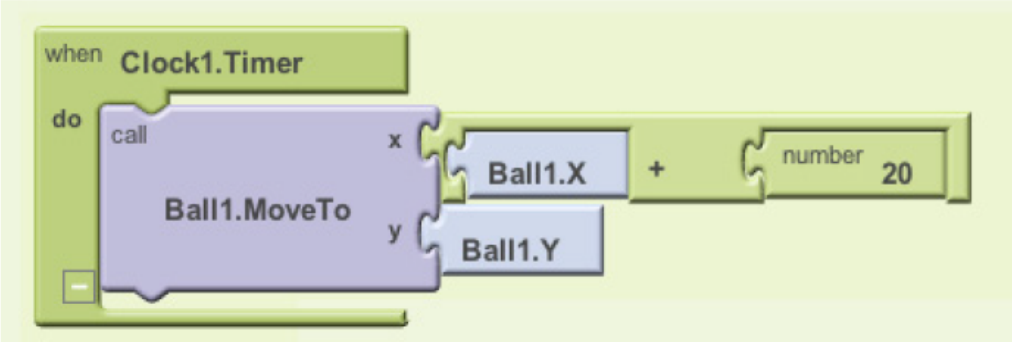

Figure 17-4. Moving the ball horizontally across the screen

MoveTo mueve un objeto a una posición absoluta en el *Canvas*, no una cantidad relativa. Por lo tanto, para mover un objeto una cierta cantidad, se establecen los argumentos **MoveTo** a la ubicación actual del objeto más el desplazamiento. Como nos estamos moviendo horizontalmente, el argumento “x” se establece en la ubicación “x” actual (**Ball1.X**) más 20 del desplazamiento, mientras que el argumento “y” se mantendría en su configuración actual (**Ball1.Y**).

Si usted quisiera mover el *Ball* en diagonal, deberá añadir un desplazamiento tanto a la coordenada “x” como a la “y”, tal cual se muestra en la Figura 17-5.

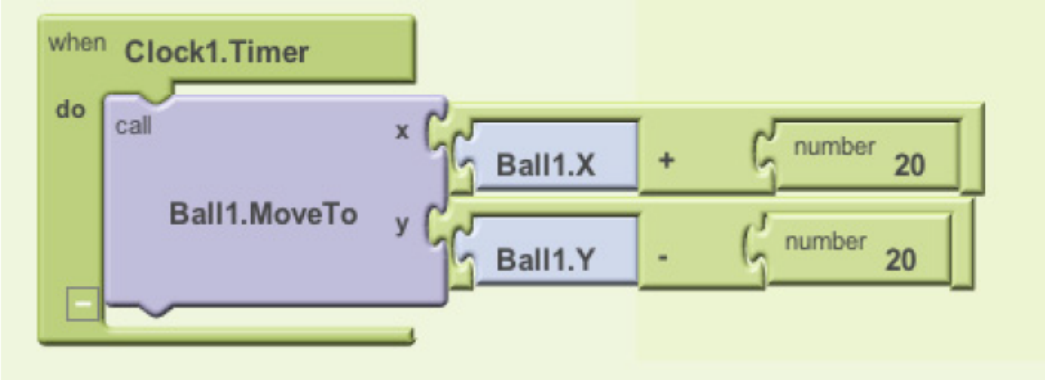

Figure 17-5. Offsetting both the x and y coordinates to move the ball diagonally

Velocidad

¿A qué velocidad se mueve la pelota en el ejemplo anterior? La velocidad depende tanto de la propiedad de Clock TimerInterval y los parámetros que especifique en la función MoveTo. Si el intervalo se establece en 1.000 milisegundos, lo que significa que se dispara un evento cada segundo. Como en el ejemplo horizontal que se muestra en la Figura 17-4, la Ball se mueve 20 píxeles por segundo.

Pero un TimerInterval de 1.000 milisegundos no proporciona una animación muy lenta; el *Ball* sólo se moverá cada segundo, y esto se verá entrecortado. Para obtener un movimiento más suave, se necesita un intervalo más pequeño. Si el TimerInterval se fija en 100
milisegundos, la pelota se movería 20 píxeles cada décima de segundo, o 200 píxeles por segundo, una tasa que aparecerá mucho más suave ante cualquier persona que utilice su aplicación. Hay otra manera de cambiar la velocidad en lugar de cambiar el intervalo del temporizador, ¿puede pensar cual es? (Pista: La velocidad y la distancia son una función de la frecuencia con la que se mueve el *Ball* cada vez) También puede alterar la velocidad, manteniendo un intervalo de temporizador de 1.000 milisegundos, y en lugar, cambiar el funcionamiento de **MoveTo** para que la *Ball* sólo se mueva 2 píxeles cada intervalo de tiempo (2 pixels/100ms siguen siendo 20 píxeles/segundo).

Funciones de animación de Alto-Nivel

La capacidad de mover un objeto a través de la pantalla es útil para cosas como anuncios animados que se deslizan dentro y fuera, pero para construir juegos y otras aplicaciones de animación, se necesita funcionalidad más compleja. Afortunadamente, App Inventor ofrece algunos bloques de Alto-Nivel para hacer frente a eventos de animación tales como un objeto que alcanza el borde de la pantalla o dos objetos que chocan.

En este contexto, “bloques de alto-nivel” significa que App Inventor se ocupa de los detalles bajo-nivel para determinar eventos como cuando chocan dos sprites.. Usted puede comprobar estos hechos por sí mismo utilizando los eventos **Clock.Timer** y verificar X, Y, Height y Width de los *sprites*. Sin embargo, dicha programación requeriría alguna lógica bastante compleja. Debido a que estos eventos son comunes a muchos juegos y otras aplicaciones, App Inventor se las pone a su disposición.

Llegar al borde

Consideremos de nuevo la animación en la que el objeto se mueve en diagonal desde la parte superior izquierda a la esquina inferior derecha del lienzo. Tal como lo programado, el objeto se movería en diagonal y luego se detiene al llegar al borde derecho o inferior del Canvas (el sistema no va a mover un objeto más allá de los límites del lienzo).

Si en cambio quiere que el objeto vuelva a aparecer en la esquina superior izquierda después de que alcance la esquina de abajo a la derecha, se podría definir una respuesta al evento **Ball.EdgeReached** como se muestra en la Figura 17-6.

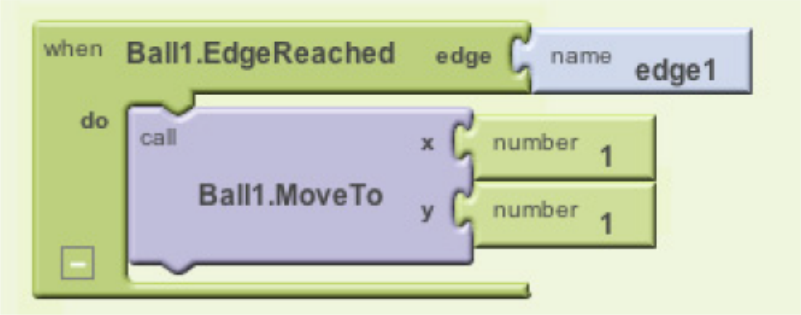

Figure 17-6. Making the ball reappear at the top-left corner when it reaches an edge

EdgeReached (un evento que sólo es aplicable para los *sprites* y *balls*) se activa cuando la *ball* golpea cualquiera de los bordes del *Canvas*. Este controlador de eventos, combinado con el movimiento diagonal especificado con el evento de temporizador descrito anteriormente, causará que *ball* se mueva diagonalmente desde la parte superior izquierda a la inferior derecha, y cuando llegue al borde saltará nuevamente hacia a la parte superior izquierda, esto lo hará una y otra vez (o hasta que usted le indique lo contrario).

Tenga en cuenta que con el evento **EdgeReached** hay un argumento edge1. El argumento especifica que *Ball* alcanzará el borde, usando el siguiente esquema de numeración direccional:

- North (norte)= 1
- Northeast (noreste)= 2
- East (este)= 3

- Southeast (sureste)= 4
- South (sur)= –1
- Southwest (suroeste)= –2
- West (oeste)= –3
- Northwest (noroeste)= –4

CollidingWith (Chocando con) y NoLongerCollidingWith (Ha dejado de chocar con)
Los juegos de tiros, deportes, y otras aplicaciones de animación a menudo dependen de la actividad que ocurre cuando chocan dos o más objetos (por ejemplo, una bala que pega en un blanco).

Consideremos un juego, por ejemplo, en el que un objeto cambia de colores y genera un sonido de explosión cuando golpea a otro objeto. La figura 17-7 muestra los bloques para tal controlador de eventos.

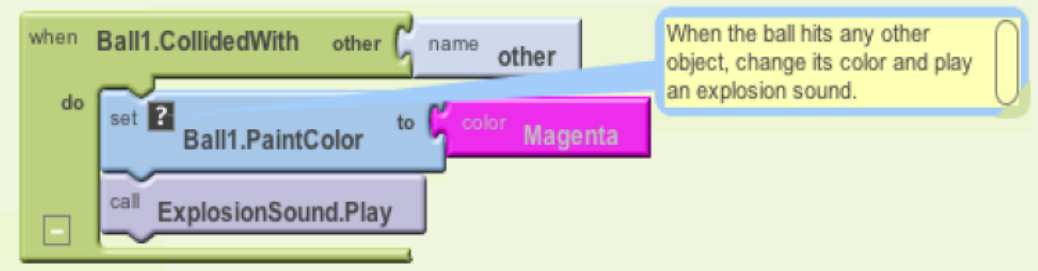

Figure 17-7. Making the ball change color and play an explosion sound when it hits another object

NoLongerCollidingWith proporciona el evento opuesto a **CollidedWith**. Se activa sólo cuando dos objetos se han separado, después de haberse chocado. Por lo tanto, en su juego, puede incluir bloques, como se muestran en la Figura 17-8..

Figure 17-8. Changing the color back and stopping the explosion noise when the objects separate

Tenga en cuenta que ambos, **CollidedWith** y **NoLongerCollidingWith** tienen un argumento, other. El argumento other especifica el objeto particular con que chocó (o se separó de él). Esto permite realizar operaciones únicamente cuando el objeto (por ejemplo, *Ball1*) interactúa

con otro objeto particular, tal como se muestra en la Figura 17-9.

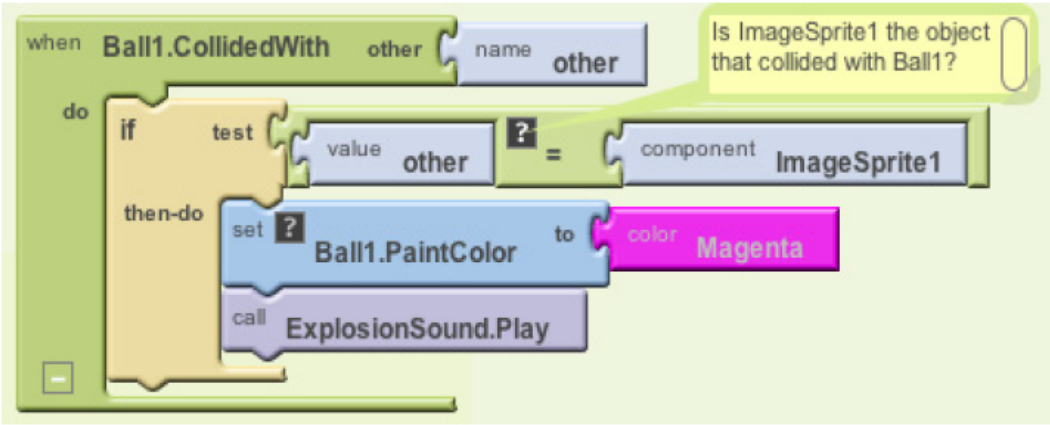

Figure 17-9. Only perform the response if Ball1 hit ImageSprite1

Figura 17-9. Sólo elabora la respuesta si Ball1 toca a ImageSprite1

El bloque **component ImageSprite1** no lo hemos discutido aún. Cuando se necesitan comparar los componentes (para saber cuáles han colisionado), como en este ejemplo, tiene que haber alguna forma de referirse a un componente específico. Por esta razón, cada componente tiene un bloque especial que se refiere a sí mismo. Así, en la sección para ImageSprite1, usted encontrará el bloque **component ImageSprite1**.

Animación interactiva

En los comportamientos animados que hemos discutido hasta ahora, el usuario no está involucrado. Por supuesto, lo principal de los juegos es que son interactivos con el usuario. A menudo, el usuario controla la velocidad o la dirección de un objeto con botones u otro objeto de interfaz de usuario.

Como ejemplo, vamos a actualizar la animación diagonal, permitiendo al usuario iniciar y detener el movimiento diagonal. Usted puede hacer esto mediante la programación de un controlador de evento **Button.Click** para desactivar y volver a activar el evento de temporizador del componente reloj.

De manera predeterminada, se testea la propiedad timerEnabled del componente Clock . Se puede desactivar de forma dinámica estableciendo en *false* en el controlador de eventos. El controlador de eventos de la figura 17-10, por ejemplo, con el primer click podría detener la actividad del temporizador del Clock.

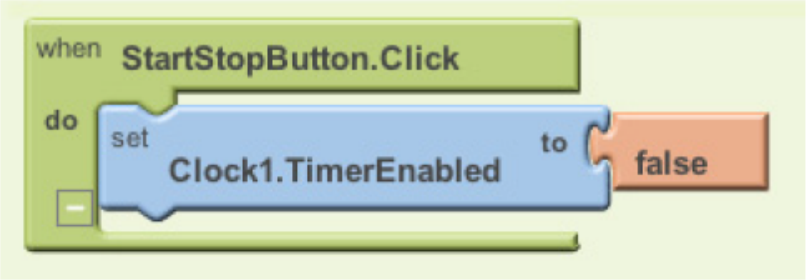

Figure 17-10. Stopping the timer the first time the button is clicked

Después de que la propiedad Clock1.TimerEnabled se establece en *false*, el evento **Clock1.Timer** ya no se activará, y la bola se detendrá.

Por supuesto, deteniendo el movimiento en la primera posición no es muy interesante. En su lugar, podría "cambiar" el movimiento de *Ball* mediante la incorporación de un **ifelse** en el controlador de eventos que activa o desactiva el temporizador, como se muestra en la figura 17-11.

Este controlador de eventos detiene el temporizador con el primer clic, y restablece el botón para que diga "Start" en lugar de "Stop". La segunda vez que el usuario hace clic en el botón, el TimerEnabled es falso, así que la "otra" parte es ejecutada. En este caso, el temporizador está activado, el cual pone al objeto de nuevo en movimiento, y el texto del botón se cambia nuevamente a "Stop". Para obtener más información sobre los bloques **Ifelse**, consulte el Capítulo 18, y para ejemplos de animaciones interactivas que utilizan el sensor de orientación, véanse los capítulos 5 y 23.

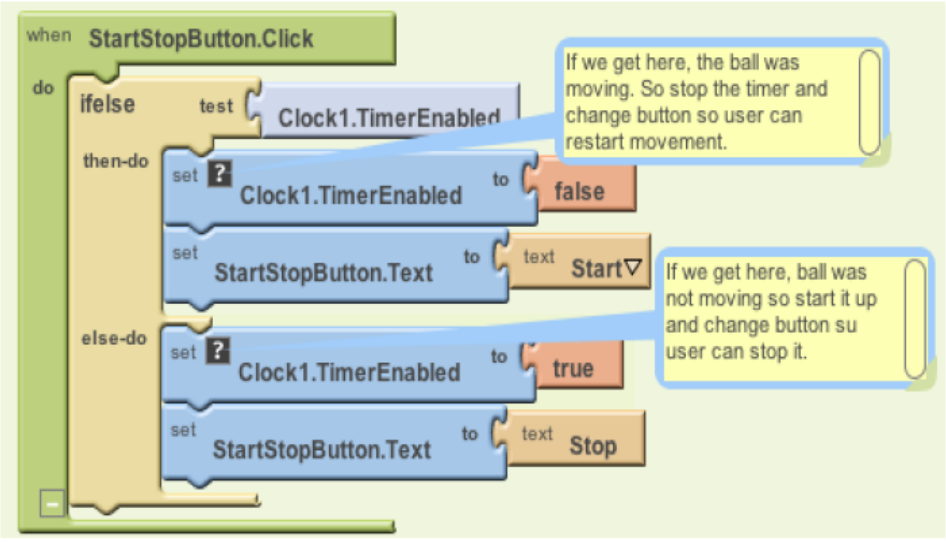

Figure 17-11. Adding an ifelse so that clicking the button starts and stops the movement of the ball

Designar la animación de un Sprite sin el temporizador de reloj

Las muestras de animación descritos hasta ahora utilizan un componente Clock y especifican que un objeto debe moverse cada vez que se activa el evento de temporizador. El esquema de evento **Clock.Timer** es el método más común de realizar una animación, otro modo de aplicarlo sería para cambiar el color de un objeto mientras transcurre el tiempo, introducir un texto de a poco (para que parezca que la app está escribiendo), o pronunciar palabras a un ritmo determinado.

Para el movimiento de un objeto, App Inventor proporciona una alternativa que no requiere el uso de un componente de reloj. Como te habrás dado cuenta, los componentes ImageSprite y Ball tienen propiedades Heading (Rumbo), Speed (Velocidad), e Interval (Intérvalo). En lugar de definir un controlador de evento **Clock.Timer**, puede establecer estas propiedades en el Diseñador de componentes o en el Editor de Bloques para controlar cómo se mueve un sprite.

Para ilustrar esto, vamos a reconsiderar el ejemplo del Ball con movimiento diagonal. La propiedad Heading (Rumbo) de un *sprite* o un*Ball* tiene un alcance de 360 grados, como se ve en la figura 17-12.

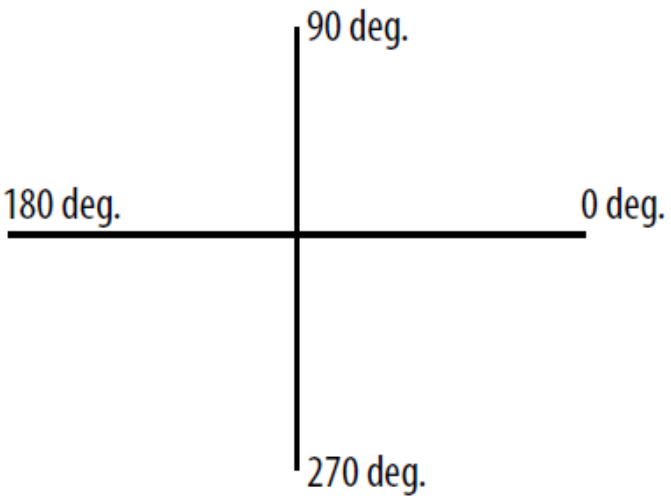

Figure 17-12. The Heading property has a range of 360 degrees

Si la propiedad Heading se establece en 0, el objeto *Ball* se moverá de izquierda a derecha. Si se establece en 90, se moverá de abajo hacia arriba. Si se establece en 180, se moverá de derecha a izquierda. Si lo establece en 270, se moverá de arriba a abajo.

Por supuesto, se puede establecer cualquier número entre 0 y 360. Para mover una pelota en diagonal desde la parte superior izquierda a la inferior derecha, se hace con Head a 315. También es necesario establecer la propiedad Speed (velocidad) en un valor distinto a 0.La propiedad Speed funciona de la misma manera que mover objetos con **MoveTo**: especifica el número de píxeles que el objeto se mueve por intervalo de tiempo, donde el intervalo se define

con la propiedad Interval del objeto.

Para probar estas propiedades, puede crear una aplicación de prueba con un *Canvas*, un *Ball* y hacer clic en "Conect to Phone" (conectar al teléfono) para ver su aplicación. A continuación, modifique las propiedades de *Ball* como Head (rumbo), Speed (velocidad), e Interval (intervalo) para ver cómo se mueve.

Si desea que el programa lo mueva desde la parte superior izquierda a la inferior derecha y luego , a la inversa, tendría que inicializar la propiedad Head a 315 en el Diseñador de componentes. A continuación, debería agregar el controlador de eventos **Ball1.EdgeReached**, para cambiar la dirección de la bola cuando llega a cualquiera de los bordes como se muestra en la figura 17-13

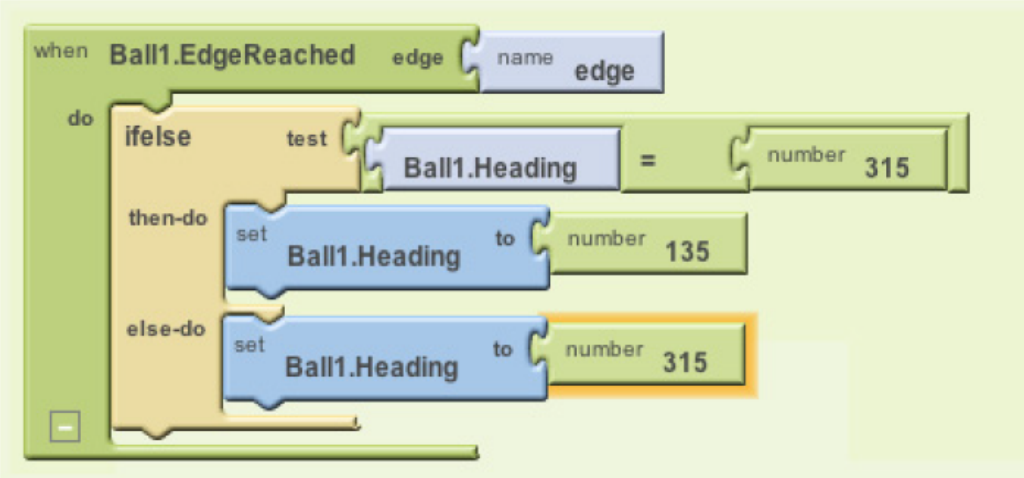

Figure 17-13. Changing the ball's direction when it reaches either edge

Resumen

La animación es: un objeto en movimiento, o dicho de otra manera: que se transforma con el tiempo y App Inventor proporciona algunas funcionalidades y componentes de alto-nivel para facilitar su tarea. Mediante la programación del componente Clock (reloj) y eventos Timer (temporizador), puede especificar cualquier tipo de animación, incluyendo el movimiento de un objeto (la actividad fundamental en casi cualquier tipo de juego).

El componente *Canvas* (lienzo) le permite definir una subzona de la pantalla del dispositivo en el que los objetos pueden moverse e interactuar. En un *Canvas* usted puede poner sólo dos tipos de componentes, *ImageSprites* y *Balls.* Estos componentes proporcionan la funcionalidad de alto-nivel para el manejo de eventos tales como colisiones y llegar al borde del *Canvas*. También tienen propiedades (dirección, velocidad e intervalo) que ofrecen un método alternativo
de movimiento.