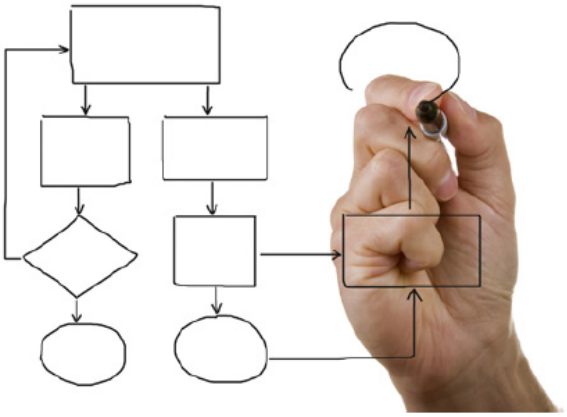
Fuente: www.appinventor.org
Traducción hecha con Google Traductor y mejorada por mi: piatticarlos@gmail.com

Programando la aplicación para tomar decisiones: Bloques condicionales

Las computadoras, incluso las pequeñas, como el teléfono en el bolsillo, son buenas para realizar miles de operaciones en tan sólo unos segundos. Aún más impresionante, es que pueden tomar decisiones especificadas por el programador basadas en la lógica y en los datos de sus slots de memoria. Esta capacidad de toma de decisiones es probablemente el ingrediente clave de lo que la gente llama inteligencia artificial y sin duda es una parte muy importante de la creación de aplicaciones inteligentes. Interesante! En este capítulo, exploraremos cómo construir la lógica de la toma de decisiones en sus aplicaciones.



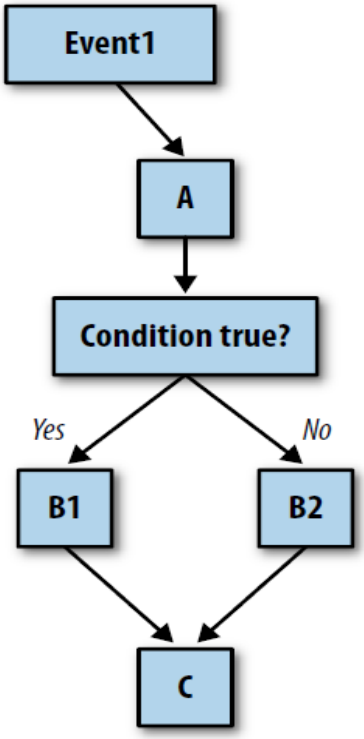


Figure 18-1. An event handler that tests for a condition and branches accordingly

Como ya comentamos en el capítulo 14, el comportamiento de una aplicación se define por un conjunto de controladores de eventos. Cada controlador de eventos ejecuta funciones específicas en respuesta a un evento en particular. La respuesta no tiene que ser una secuencia lineal de funciones, sin embargo, se puede especificar que algunas funciones se realicen sólo bajo ciertas condiciones. Una aplicación como un juego puede comprobar si la puntuación ha llegado a 100. Una aplicación *location-aware* (que reconoce su posición global) podría preguntar si el teléfono está dentro de los límites de algún edificio. Su aplicación puede hacer esas preguntas y, dependiendo de la respuesta, elige un camino determinado en el programa.

La Figura 18-1 muestra un diagrama de flujo de un controlador de eventos con una comprobación condicional.

Cuando se produce el evento, la función se lleva a cabo sin importar lo que pase. A continuación, se realiza una prueba de decisión. Si la prueba es verdadera, se lleva a cabo B1. Si es falsa, se realiza B2. En cualquiera de los casos, el resto del proceso del controlador de eventos (C) se completará.

Los diagramas de decisión, como el que se ve en la Figura 18-1, son como los árboles, es común decir que las "ramas" de la App llevan a un lado o a otro en función del resultado de la prueba. Por lo tanto, en este caso, usted diría: "Si la prueba es verdadera, la rama que contiene B1 se realiza."

Prueba de condiciones con bloques if y ifelse

App Inventor proporciona dos tipos de bloques condicionales (Figura 18-2): **if** (si) y **ifelse** (si, de otro modo) ambos se encuentran en la sección Control de los bloques integrados.

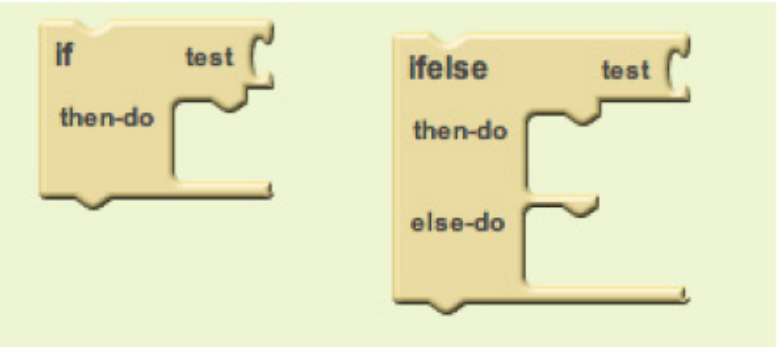


Figure 18-2. The if and ifelse conditional blocks

Usted puede conectar cualquier expresión booleana en el encastre "test" de estos bloques. Una expresión booleana es una ecuación matemática que devuelve un resultado verdadero o falso. La expresión comprueba el valor de las propiedades y variables mediante operadores relacionales y lógicos como los que se muestran en la Figura 18-3.

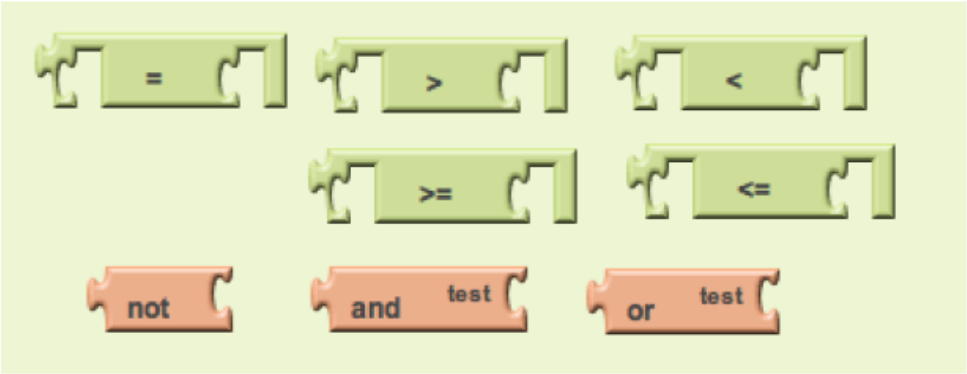


Figure 18-3. Relational and logical operator blocks used in conditional tests

Tanto en **if** como en **ifelse**, los bloques que se ponen dentro de la ranura "then-do" (a continuación-hacer) sólo se ejecutará si el test es verdadero. En un bloque if, si el test es falso, la aplicación pasa a los bloques que siguen debajo de él. Si el test **ifelse** es falso, se llevan a cabo los bloques dentro de la ranura "else-do" (de otro modo-hacer).
Por lo tanto, para un juego, es posible conectar una expresión booleana relativa a la puntuación, como se muestra en la Figura 18-4.

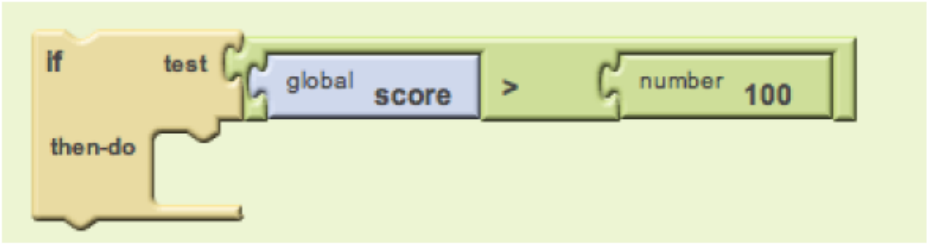


Figure 18-4. A Boolean expression used to test the score value

En este ejemplo, si la puntuación pasa de 100 se reproduce un archivo de sonido. Tenga en cuenta que si el test es falso, no hay bloques que se ejecutan. Si quieres activar una acción mediante una prueba falsa, puede utilizar un bloque **ifelse**.

Programación de una decisión: uno o el otro
Imagine una aplicación para utilizar cuando está aburrido: pulsa un botón en su teléfono y llama a un amigo al azar. En la Figura 18-5, se utiliza un bloque **random integer** (número entero aleatorio) para generar un número aleatorio y luego un bloque **ifelse** para llamar a un número de teléfono particular en base a ese número al azar.

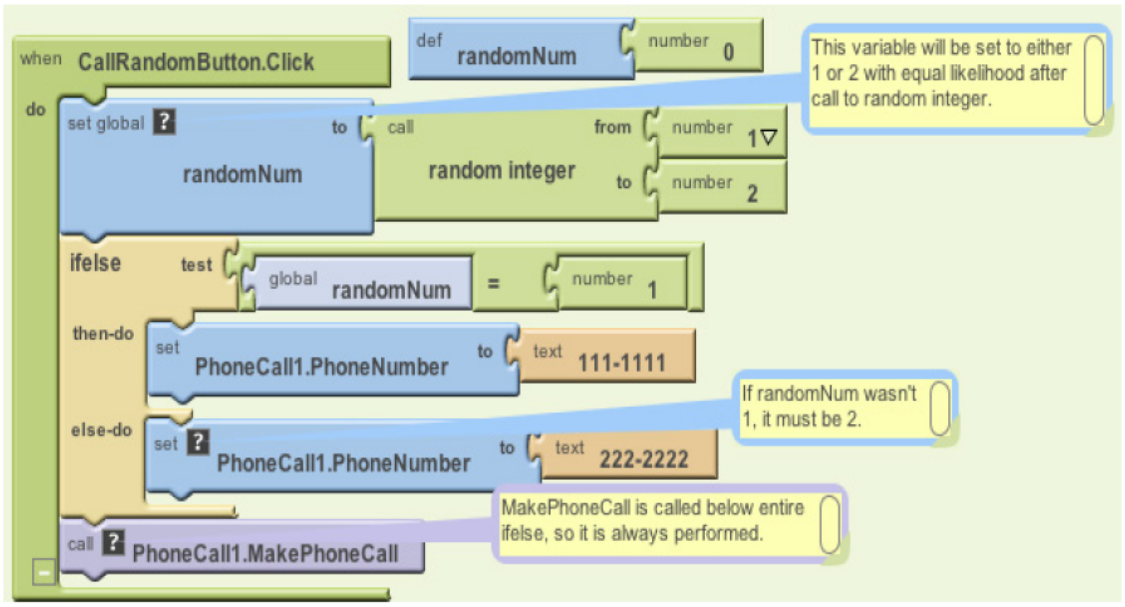


Figure 18-5. This ifelse block calls one of two numbers based on the randomly generated integer

En este ejemplo, **random integer** es invocado con los argumentos 1 y 2, lo que significa que el número devuelto al azar será 1 o 2 con la misma probabilidad. La variable RandomNum almacena el número aleatorio devuelto.

Después de ajustar RandomNum, los bloques se comparan con el número 1 en la prueba **ifelse**. Si el valor de RandomNum es 1, la aplicación toma la primera rama (then-do), y el número de teléfono se establece a 111-1111. Si el valor no es 1, el resultado es falso, por lo cual la aplicación elige la segunda rama (else-do), y el número de teléfono se establece a 222-2222. La aplicación hace una llamada telefónica de cualquier manera porque la ejecución de **MakePhoneCall** está por debajo del bloque **ifelse**.

Programar Condiciones dentro de Condiciones

Muchas situaciones de decisión no son binomio, es decir, que no tienen sólo dos resultados para elegir. Por ejemplo, es posible que desee elegir entre más de dos amigos en su programa de llamada aleatoria. Para ello, puede colocar un **ifelse** en la ranura else-dho del primer bloque **ifelse**, como se muestra en la Figura 18-6.

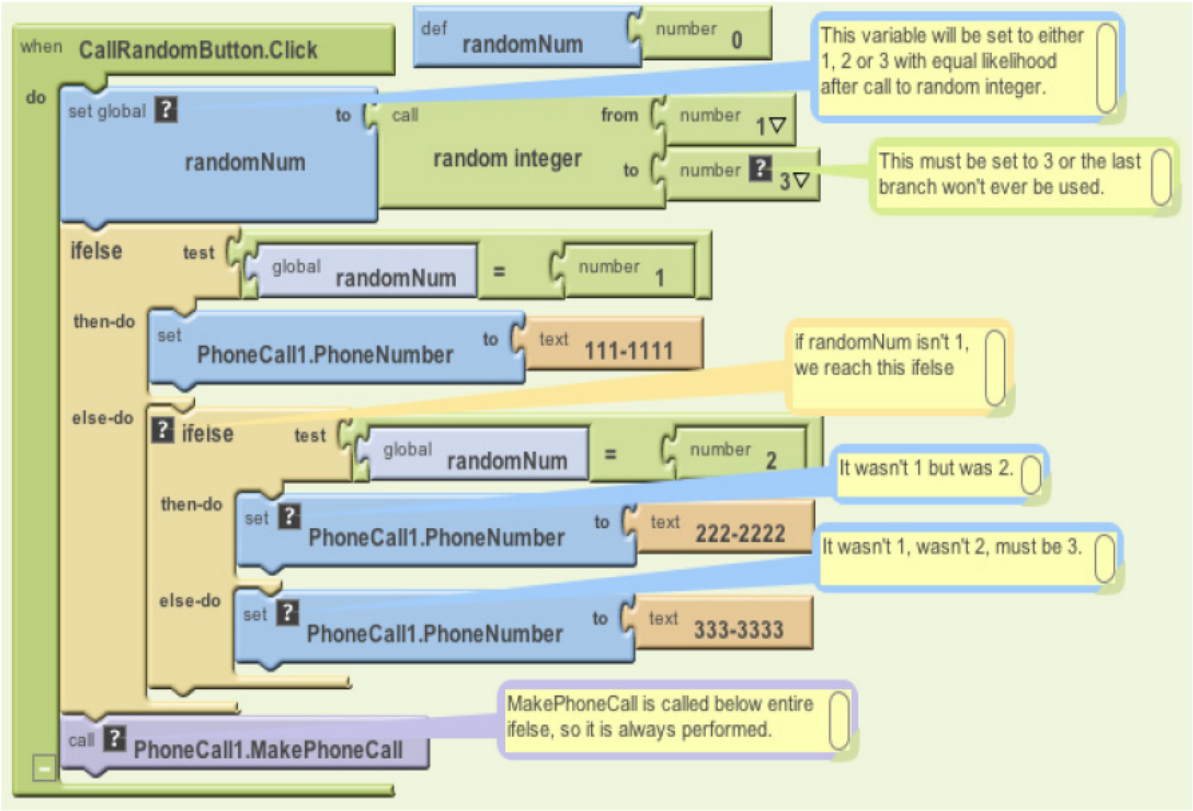


Figure 18-6. An ifelse condition is placed within the else-do of an outer condition

Con estos bloques, si el primer test es verdadero, la aplicación ejecuta la primer rama then-do y llama al número 111-1111. Si el primer test es falso, se ejecuta la otra rama else-do, a lo que inmediatamente se ejecuta la otra comprobación **ifelse**. Por lo tanto, si en la primera prueba **ifelse** (RandomNum = 1) es falso, y en la segunda prueba **ifelse** (RandomNum = 2) es verdadero, entonces el segundo then-do se ejecuta y se llama a 222-2222. Si ambas pruebas son falsas, la otra rama else-do de la parte inferior se ejecuta, y se llama al tercer número (333-3333).

Tenga en cuenta que esta modificación sólo funciona porque el parámetro *to* de la petición a **random integer** fue cambiado a 3 de manera que 1, 2, o 3 son devueltos con la misma probabilidad.

La colocación de un control dentro de otro se llama anidar. En este caso, se diría que los bloques anidados son los if-else. Se puede usar la lógica anidada para ofrecer más opciones en las peticiones al azar de su aplicación, en general, para agregar complejidad arbitraria a cualquier aplicación.

Programar condiciones complejas

Además de las comprobaciones anidadas, también puede realizar test más complejos que una simple prueba de igualdad. Por ejemplo, imagine una aplicación que vibra cuando usted

(con su teléfono) salga de un edificio o algún límite. Esta aplicación puede ser utilizada por una persona con libertad condicional para que le avise cuando se aparte demasiado de sus límites legales, o por los padres, para controlar el paradero de sus hijos. Un profesor puede utilizarlo para tomar asistencia automáticamente (si todos sus estudiantes tienen un teléfono con Android!).

Para este ejemplo, vamos a hacer esta pregunta: ¿el teléfono está dentro de los límites del Centro de Ciencias Harney de la Universidad de San Francisco? Esta aplicación requeriría un examen complejo que consta de cuatro preguntas diferentes:

- ¿La latitud del teléfono es menor que la latitud máxima del límite (37.78034)?
- ¿La longitud del teléfono es menor que la longitud máxima del límite (-122.45027)?
- ¿La latitud del teléfono es mayor que la latitud mínima del limite (37.78016)?
- ¿La longitud del teléfono es mayor que la longitud mínima del límite (-122.45059)?

Vamos a usar el componente LocationSensor para este ejemplo. Usted ya debe ser capaz de entender este tema, incluso si usted no ha visto el funcionamiento de LocationSensor, pero puede aprender más en el capítulo 23.

Usted puede construir pruebas complejas usando los operadores lógicos **AND**, **OR** y **NOT**, que se encuentran en la sección de Lógica. En este caso, usted comenzaría arrastrando un bloque **if** y un bloque **and** y luego colocando el bloque **and** dentro de la ranura "test" del bloque **if**, como se ilustra en la Figura 18-7.

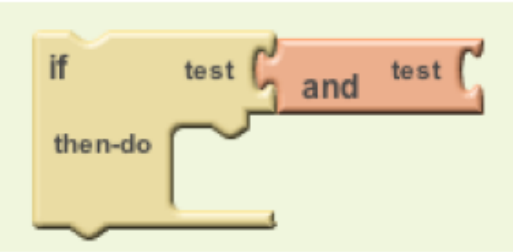


Figure 18-7. An and block is placed within the "test" slot of the if block

Entonces arrastre los bloques para la primera pregunta y coloquelos en la ranura "test", como se muestra en la Figura 18-8.

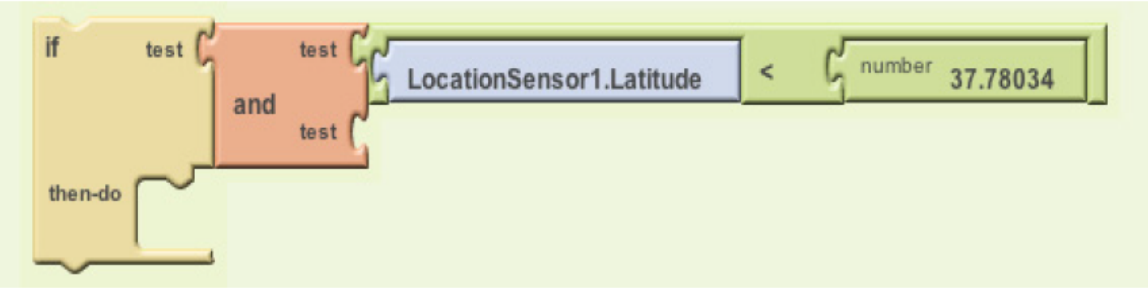


Figure 18-8. When the blocks for the first test are placed into the and block, a new test slot opens

Tenga en cuenta que a medida que encastra un bloque en la ranura “test” del bloque **and**, se abre una nueva ranura “test”. Si usted llena estos espacios con las otras pruebas y coloca **ifelse** dentro de un evento **LocationSensor.LocationChanged**, tendrá un controlador de eventos que comprueba los límites, como se muestra en la Figura 18-9.

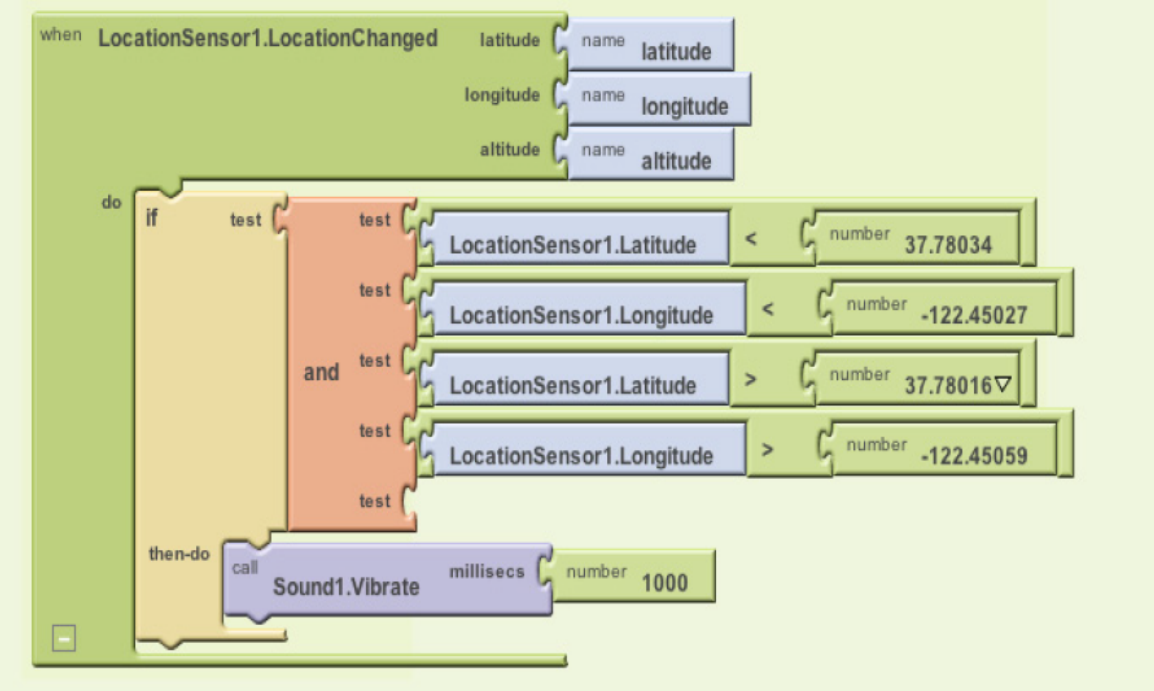


Figure 18-9. This event handler checks the boundary each time the location changes

Con estos bloques, cada vez que LocationSensor recibe una nueva lectura y su ubicación está dentro de los límites, el teléfono vibra.

Bien, hasta el momento esto va bien, pero ahora vamos a probar algo aún más complicado para darle una idea de la magnitud que tienen las aplicaciones para tomar decisiones. ¿Y si quisiera que el teléfono vibre cuando se cruzó la frontera desde el interior hacia el exterior? Antes de seguir adelante, piense en cómo puede programar tal condición.

Nuestra solución es definir una variable withinBoundary que memoriza si la lectura del sensor anterior estaba dentro del límite o no, y la compara para cada lectura que el sensor realice sucesivamente. withinBoundary es un ejemplo de una variable booleana, en lugar de almacenar un número o texto, almacena un valor verdadero o falso. En este ejemplo se iniciaría como falso, como se muestra en la Figura 18-10, lo que significa que el dispositivo no está dentro del Centro de Ciencias Harney de la USF..

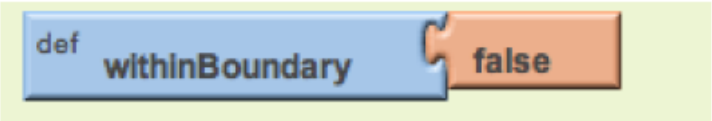


Figure 18-10. withinBoundary is initialized as false

Los bloques pueden ahora ser modificados al modo en que la variable withinBoundary se establezca en cada cambio de ubicación, y para que el teléfono sólo vibre cuando se mueve desde el interior hacia fuera de la frontera. Para poner esto en términos para que podamos usarlo en los bloques, el teléfono debe vibrar cuando (1) la variable withinBoundary es verdadera, es decir, la lectura anterior estaba dentro de la frontera, y (2) la nueva lectura de ubicación del sensor está fuera de la frontera. La figura 18-11 muestra los bloques actualizados.

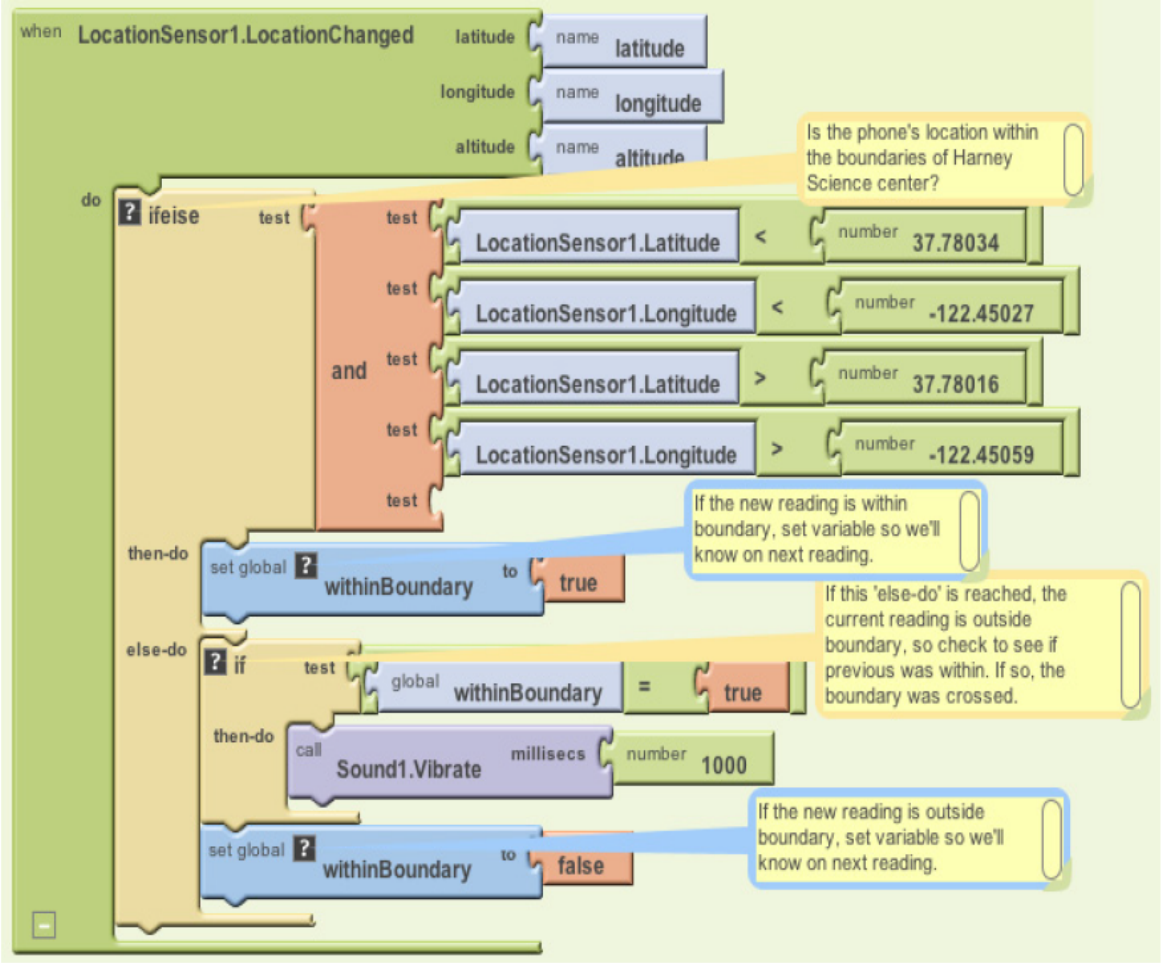


Figure 18-11. These blocks cause the phone to vibrate only when it moves from within the boundary to outside the boundary

1° nota en la imagen:
¿El teléfono está dentro de los límites del Centro de Ciencias Harney?
2° nota en la imagen:
Si la nueva lectura está dentro de los límites, establecer la variable para que lo sepamos en la próxima lectura.
3° nota en la imagen:
Si se llega a este "else-do", es porque la lectura actual está fuera de límites, por lo tanto se comprueba si la anterior estaba dentro. Si es así, la frontera fue cruzada.
4° nota en la imagen:
Si la nueva lectura está fuera de límites, establecer la variable para que lo sepamos en la siguiente lectura.

Vamos a examinar más de cerca estos bloques. Cuando el LocationSensor obtiene una lectura, en primer lugar comprueba si la nueva lectura está dentro del límite. Si lo está, LocationSensor establece la variable de withinBoundary a verdadero. Como queremos que el teléfono vibre cuando estamos fuera del límite, ninguna vibración tiene lugar en esta primera rama.
Si llegamos a “else-do”, sabemos que la nueva lectura está fuera del límite. En ese punto, tenemos que comprobar la lectura anterior: si estamos fuera de los límites, sólo queremos que

el teléfono vibre si la lectura anterior estaba dentro de los límites. withinBoundary nos muestra la lectura anterior, por lo cual podemos comprobarlo. Si es cierto, hace vibrar el teléfono.

Hay una cosa más que tenemos que hacer una vez que hemos confirmado que el teléfono se trasladó desde el interior hacia fuera de los límites ¿puede pensar lo que es? También debemos restablecer withinBoundary en false para que el teléfono no vibre de nuevo en la próxima toma de lectura.

Una última nota sobre las variables booleanas: echa un vistazo a las dos comprobaciones **if** en la figura 18-12. ¿Son equivalentes?

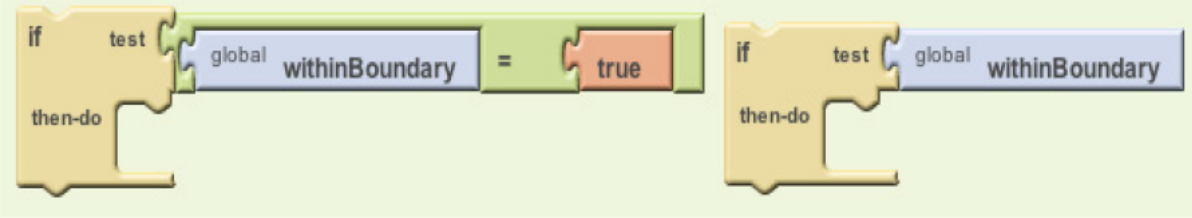


Figure 18-12. Can you tell whether these two if tests are equivalent?

La respuesta es ¡sí! La única diferencia es que la prueba de la derecha es en realidad la forma más sofisticada de hacer la pregunta. La prueba de la izquierda compara el valor de una variable booleana con “true” (verdadero). Si withinBoundary contiene “true”, comparar verdadero con verdadero, lo cual el resultado es “true” (verdadero). Si la variable contiene “false”, usted compara falso con verdadero, lo cual es “false”. Sin embargo, haciendo la comprobación con el valor de withinBoundary, como en el ensayo de la derecha, da el mismo resultado y es más fácil de codificar.

Resumen

¿Su cabeza está dando vueltas? El comportamiento anterior fue bastante complejo! Pero es el tipo de toma de decisiones que realizan las aplicaciones sofisticadas. Si usted construye comportamientos parte por parte (o rama por rama) y prueba sobre la marcha, encontrará que es posible especificar una lógica compleja (nos atrevemos a decir: inteligencia artificial). Hará que le duela la cabeza y ejercitará un poco el lado lógico de su cerebro, pero también puede ser muy divertido.