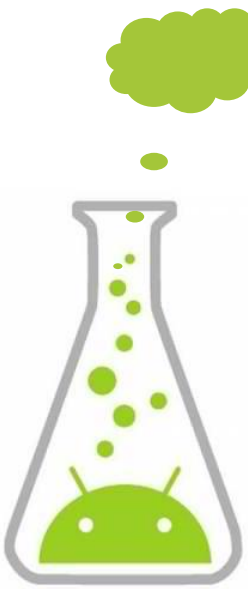
App Inventor 2 – Tutorial



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Introducción a App Inventor 2

Objetivo

Que el alumno se interiorice con las características generales de App Inventor, tanto en lo referente al entorno de diseño y al entorno de programación por bloques, como a los softwares necesarios para su uso.

Consideraciones Previas

- Requisitos del Sistema Informático
- Requisitos del Dispositivo Android
- Información para la descarga de software
- Creación de una cuenta

Dentro de App Inventor 2

- Ingreso y creación un proyecto.
- Características Generales del Entorno de Diseño
- Características Generales del Editor de Bloques

Sobre los bloques de programación

- Visión general de los distintos tipos de bloques, separados por categoría

Programación en App Inventor 2

Objetivo

Que el alumno sea capaz de comenzar nuevos proyectos para la construcción de **Apps**, se interiorice en el uso y configuración de los componentes, aprenda a utilizar y combinar distintos bloques de programación, así como también conceptos de programación básicos propios de **App Inventor**.

Ejemplo de construcción de una App

- Crear y guardar un nuevo proyecto
- Concepto de componente
- Concepto de Propiedad de un componente
- Bloques de programación generales y propios de los componentes
- Orientación a eventos y área de tareas
- Getters y Setters para Propiedades
- Unión de Bloques
- Advertencias en App Inventor
- Asignar un Nuevo Valor a un Bloque
- Uso de los Bloques de Control IF
- Concepto de bloque Mutador
- Duplicación de Bloques
- Inicialización y uso de Variables
- Typeblocking
- Getters y Setters para Variables
- Errores en App Inventor
- Secuencia de tareas
- Bloque Comparador como Condición de Control
- Funcionamiento de la App Propuesta

Probando Nuestras Apps

Objetivo

En este módulo estudiaremos las distintas posibilidades con las que contamos dentro de **App Inventor** para probar el funcionamiento de nuestras **Apps**, a fin de realizar lo que en programación se conoce como **depuración**. Aprenderemos también sobre el uso de múltiples pantallas dentro de una misma **App**, así como también conceptos de programación referente a bloques de texto y matemáticos.

- Concepto de Depuración
- Modos de Prueba o Depuración
- Software Necesario
- Uso del programa aiStarter
- Uso del Emulador
- Conexión a través de USB
- Modo de Depuración USB
- Problemas Comunes

- Actualización de la MIT AI2 Companion App
- Restablecer la conexión (Reset Connetion)
- Conexión a través de Wifi
- Uso del Código QR para Establecer la Conexión
- Uso del Código de 6 Caracteres para Establecer la Conexión

Proyecto 02

- Uso de Etiquetas (Label) y Casillas de Texto (TextBox)
- Propiedad Hint para TextBoxes (sugerencia)
- Uso de las Operaciones Matemáticas Básicas
- Alineación de los Bloques Dentro de la Ventana
- Agregar una nueva ventana a Nuestra App
- Uso de los Bloques is_empty y length
- Cambio de Pantalla y Cierre de la App

Uso de los Bloques While y For

Objetivo

En este módulo aprenderemos conceptos de programación tales como: **Listas**, uso de los bloques de control **while** y **for**, uso de **variables globales** y **locales**, bloque matemático **random**, bloque de texto **join**, etc. Aprenderemos también de qué manera se **empaquetan** las **App** para poder instalarlas en nuestros dispositivos **Android**.

Generador de Números Aleatorios

- Listas en **App Inventor**
- Uso del Bloque While (mientras)
- Desigualdad como Condición de Control
- Acceso rápido a Getters y Setters para variables
- Uso del Bloque Random Integer
- Funcionamiento del Bloque While Paso por Paso
- Uso de las Listas
- Uso del Bloque Join
- Mostrando la Lista en Pantalla
- Vaciar una Lista

Generador de Números Aleatorios 2

- Crear una Copia de un Proyecto Existente
- Eliminar Bloques de un Programa
- Variables Locales vs Globales
- Uso del Bloque For (Para)
- Funcionamiento del Bloque For Paso por Paso
- Bloque For vs While

Empaquetado de una App

- Instalar una App por Medio de Código QR
- Guardar la App como Archivo en la PC

Dibujando con App Inventor

Objetivo

En este módulo introduciremos al componente **Canvas** (lienzo), el cual utilizaremos para dibujar. Aprenderemos como se organizan los componentes en pantalla con el uso de **Arrangements**, así como también el uso de **Argumentos** dentro de ciertos bloques de control de eventos.

PaintPot: Vamos a Dibujar!

Diseño y Configuración

- Títulos y Nombres en App Inventor
- Subiendo Archivos a App Inventor
- Renombrando los Componentes
- Uso de Arrangements
- Sobre el Componente Canvas
- Colocando una Imagen dentro del Canvas

Conceptos para la Programación

- Uso del Componente Canvas
- Concepto de Argumento
- Concepto de Bloque de Comando

Programación de la App

- Cambiando el Color de la Pintura
- Dibujando un Círculo
- Dibujando una Línea
- Borrando lo Dibujado

Construcción de Apps Multimedia

Objetivo

En este módulo vamos a aprender sobre el uso de los compontes del tipo **media** (multi-media) y del tipo **storage** (almacenamiento), así como también algunos del tipo **User Interface** que nos serán de particular utilidad para con los dos primeros tipos.

HelloPurr: Acariciar al gatito para que haga miau!

- Colocando una Imagen Dentro de un Botón
- Uso del Componente Sound (Sonido)

Grabadora de Voz

- Sobre el Componente TinyDB (Tiny Data Base)
- Sobre el Componente Spinner
- Sobre el Componente Slider (deslizador)
- Uso del Componente SoundRecorder
- Uso del Componente Player
- Uso del Componente Slider
- Uso del Componente TinyDB
- Uso del Componente Spinner
- Almacenando Datos dentro de la TinyDB

Sensores en App Inventor

Objetivo

En este módulo vamos a aprender sobre el uso de los componentes del tipo "**Sensor**". Aprenderemos también sobre uso del componente **Notifier**, uso de **ImageSprites** para

desplazar imágenes dentro de un **Canvas**, así como otros conceptos propios del editor de bloques.

Obteniendo Coordenadas y Direcciones

- Sobre el Componente LocationSensor
- Sobre el Componente Notifier
- Uso del Componente Notifier
- Colapsar y Expandir los Bloques
- LocationSensor – Obteniendo una Dirección Física
- LocationSensor – Obteniendo Coordenadas Geográficas

Haciendo uso del AccelerometerSensor

Diseño y Configuración

- Sobre el Componente ImageSprites
- Sobre el Componente AccelerometerSensor

Conceptos para la Programación

- Uso del Componente AccelerometerSensor
- Uso del Componente ImageSprite
- Uso de Bloques del Tipo Any component
- Funciones Alternativas del Componente Player

Programación de la App

- Animando los ImageSprites
- Obteniendo un Resultado
- Hacer que los ImageSprites Reboten
- Interacción entres los ImageSprites

Uso de Procedimientos e IF Multiples

Objetivo

En este módulo aprenderemos sobre el uso de los bloque **de procedimiento (Procedures)** así como también el modo de expandir nuestros **bloques if** para llevar a cabo selecciones múltiples, lo que nos permitirá hacer uso del concepto de máquina de estados.

Programemos una Calculadora!!

Diseño y Configuración

- Uso de TableArrangements
- Determinando el Tamaño de los Componentes

Conceptos para la Programación

- Concepto de Máquina de Estados
- Concepto de Procedimiento
- Bloques de Procedimiento "Do"
- Bloques de Procedimiento "Result"

Programación de la App

- Definiendo el estado inicial
- Definiendo los Procedimientos
- Asignando un Parámetro a un Procedimiento
- If para Selección Múltiple – Procedimiento Numero
- Programación para las Teclas de Operación
- Programación del Procedimiento de Calculo
- Programación para el Botón Igual
- Programación para el BotonBorrar

Casos particulares

- Programación para el Botón Cero
- Programación para Botón Punto

Introducción a App Inventor 2

A lo largo del presente curso vamos a aprender el uso de esta entretenida **utilidad web** desarrollada por **Google** y brindada por el **MIT** (Instituto Tecnológico de Massachusetts) para el desarrollo de aplicaciones para teléfonos celulares (así como otros dispositivos) con sistema operativo **Android**.

Objetivo

El objetivo del presente módulo es que el alumno se interiorice con las características generales de App Inventor, tanto en lo referente al entorno de diseño y al entorno de programación por bloques, como a los softwares necesarios para su uso.

Consideraciones Previas

Antes de comenzar, necesitamos contar con los siguientes elementos:

- **Para Ingresar a la Página Web de App Inventor 2:**

1) Computadora y Sistema Operativo

   Una **computadora** con conexión a **internet** y puerto **USB**. Se puede optar por alguna de las siguientes **3** posibilidades:

   a. **PC** con **Windows**, ya sea Windows **XP**, Windows **Vista** o Windows **7** u **8**.
   b. **PC** con **GNU/Linux**, ya sea **Ubuntu 8**, **Debian 5** o versiones superiores de estos.
   c. **Macintosh** con procesador **Intel** y **Mac OS X 10.5** o superior.

Nota: De aquí en adelante se supondrá el uso de **PC** con **Microsoft Windows**.

2) Navegador

   Se puede optar entre los siguientes **3** softwares:

   a. **Mozilla Firefox 3.6** o superior.
   b. **Google Chrome 4.0** o superior.
   c. **Apple Safari 5.0** o superior.

   Nota: De momento, **no** es posible utilizar Microsoft **Internet Explorer**.

3) Una Cuenta de **Gmail**.

   Explicaremos como crear la misma en pasos posteriores.

- Para probar las Apps:

  Se puede optar por las siguientes **3** posibilidades:

  a. Utilizando el **emulador** (el cual veremos más adelante) directamente en la **PC**.
  b. Utilizando un **dispositivo Android** conectado a través de **USB** a la computadora.
  c. Utilizando un **dispositivo Android** conectado a la misma **red** que la computadora a través de **Wifi**.

Para los dos últimos casos, el teléfono o tablet debe contar con sistema operativo **Android 2.3** (**Gingerbread**) o **superior** y tener instalada la aplicación **MIT AI2 Companion App** que se puede descargar desde el siguiente link:

https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3

Además, deberemos instalar en la **PC** el programa **aiStarter**. Este programa es necesario tanto para **enlazar** como para **emular** el dispositivo **Android**. El mismo se puede descargar desde aquí:

http://appinv.us/aisetup_windows

Este programa requiere del uso de **Java**, que es provisto por **Oracle** gratuitamente en esta dirección:

https://www.java.com/es/download/

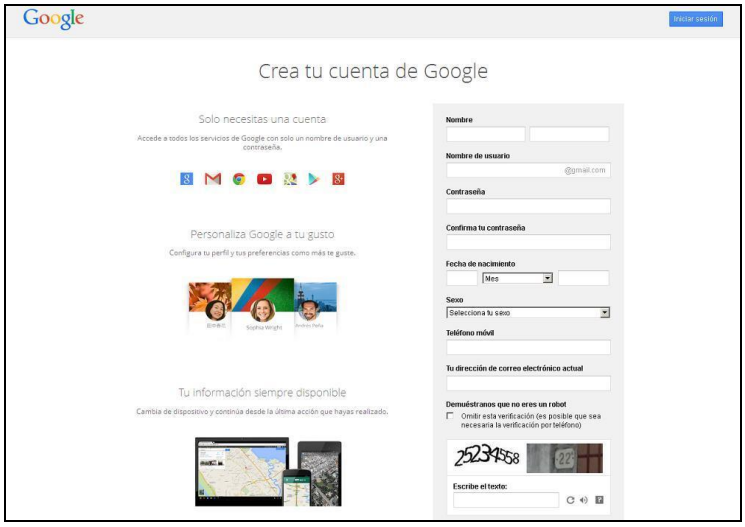
Creación de una cuenta

Para poder comenzar a trabajar con **App Inventor**, necesitaremos disponer de una cuenta de **Google**, la cual utilizaremos para "**loggearnos**" en dicho sitio web. La misma la podremos crear a través de la página de **Gmail**:

https://mail.google.com/

En la **esquina superior derecha**, encontraremos el botón **Crear una cuenta**:

Le damos **clic** y nos aparecerá la siguiente página:



Rellenamos los campos que aparecen en la columna gris con **nuestra información**:

- Nombre y Apellido.
- Un nombre de usuario.
- Una **contraseña**: Puede ser cualquier cadena caracteres con un **mínimo** de **ocho**.
- Fecha de nacimiento.
- Sexo.
- Número de teléfono (**opcional**).
- Una dirección de correo actualmente **en uso**.

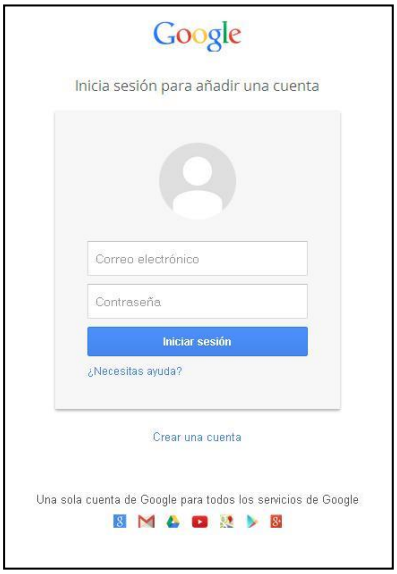
Ingresamos el código de verificación o **capcha** de la imagen, tildamos la casilla de verificación al lado de "Acepto las Condiciones del servicio y la Política de privacidad de Google", y damos clic en **Siguiente paso**:

Listo, ya tenemos nuestra cuenta de **Google**. Se nos propondrá agregar más información a nuestro perfil de usuario para usarlo en **Google+** (la **red social** de **Google**), pero esto no será necesario para utilizar el servicio de **App Inventor**.

Dentro de App Inventor 2

Ya estamos listos para dirigirnos a la página de **App Inventor**, cuya dirección es:

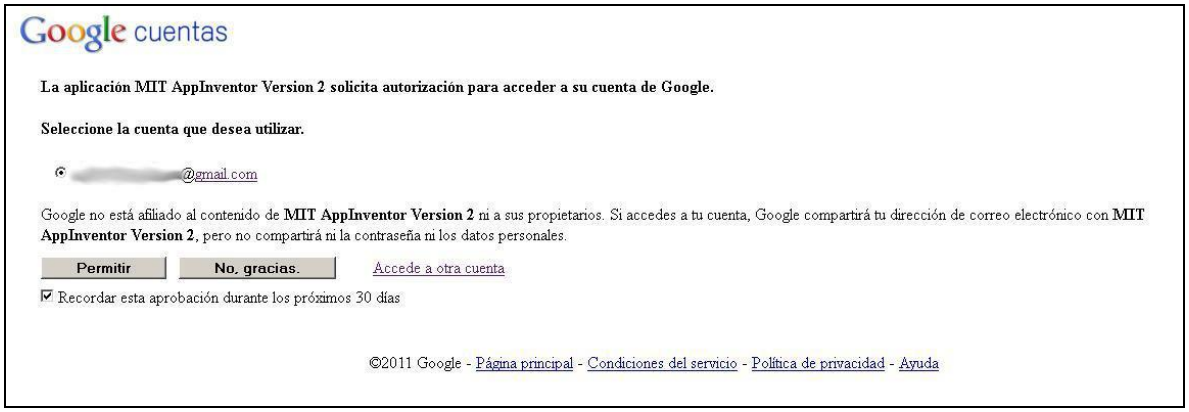
http://appinventor.mit.edu/explore/



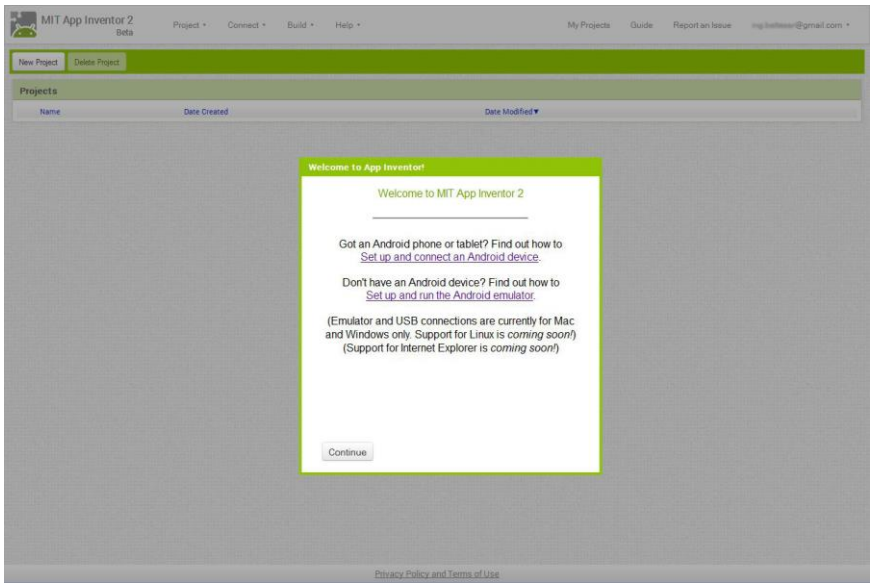
Para comenzar, damos clic en el botón "**Create**" (arriba a la derecha) y nos abrirá una página de **loggeo** con un cuadro como el de la derecha.

Aquí debemos ingresar la dirección de correo **Gmail** que acabamos de crear ("**nombre de usuario**"@gmail.com) y la contraseña de la misma.

Damos clic en **Iniciar Sesión** y se nos abrirá la siguiente página:



En la misma verificamos que la cuenta que usaremos para "loggearnos" es la **misma** que **ingresamos** en el paso anterior, damos clic en **Permitir** y ya estaremos listos para empezar a utilizar los servicios de **App Inventor 2**. Al hacer esto se nos presenta la siguiente página:

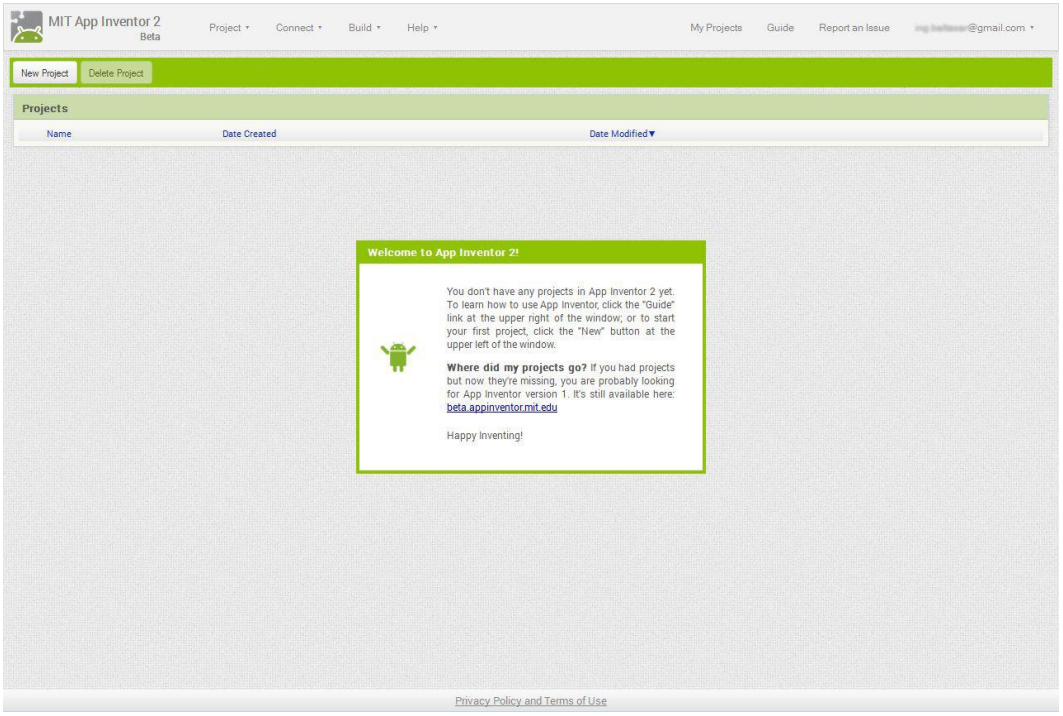


En ella se nos presenta un cuadro de **bienvenida**, el cual nos informa las últimas novedades de **App Inventor**. Hacemos clic en **Continue** y listo, ya habremos ingresado dentro del entorno de desarrollo.

Es posible que antes de esto nos aparezca un recuadro proponiéndonos contestar una encuesta para mejorar el servicio. De estar interesados en completar la misma mas tarde cliqueamos en **Take Survey Later**, de otro modo daremos clic en **Never Take Survey** y ya no nos aparecerá más este cartel.

Nota: Esta encuesta se debe a que el sitio de **App Inventor 2** está siendo constantemente modificado y mejorado, por lo que **algunos contenidos** del presente curso **pueden diferir** de lo que se puede encontrar en la página al momento de su estudio.

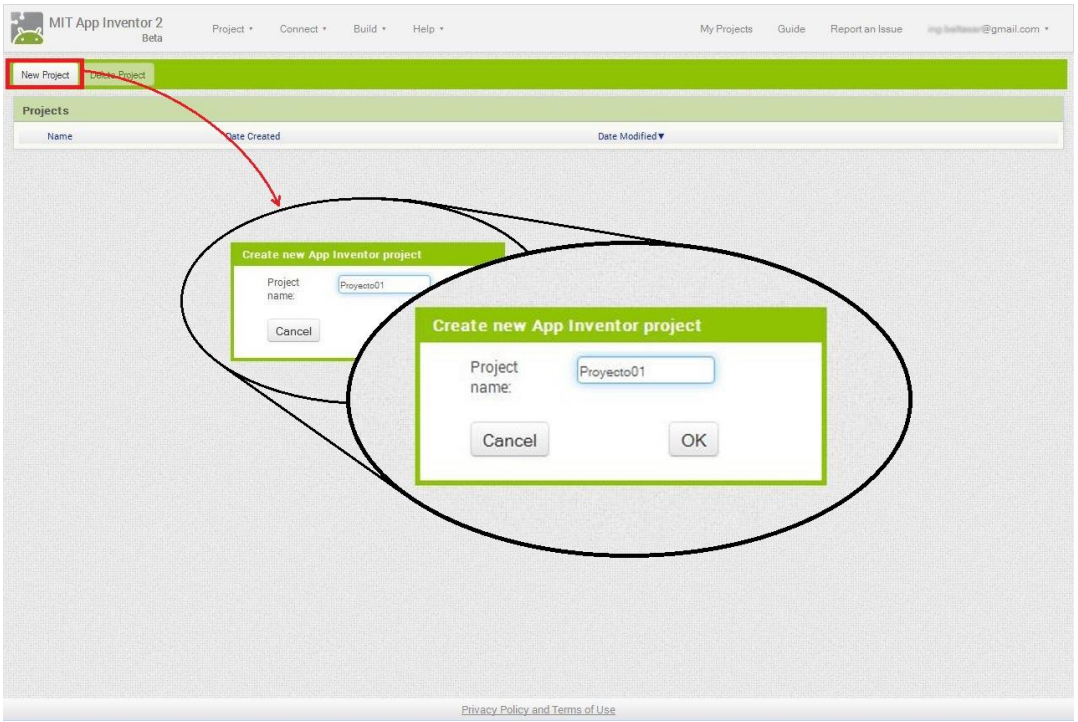
Una vez hecho esto, nos aparecerá otro cartel indicándonos que no disponemos de **ningún proyecto** dentro de **App Inventor 2,** y que de haber trabajado en el **App Inventor** anterior (**App Inventor 1**, por así decir), nuestros proyectos se encuentran a salvo todavía en esa otra versión.



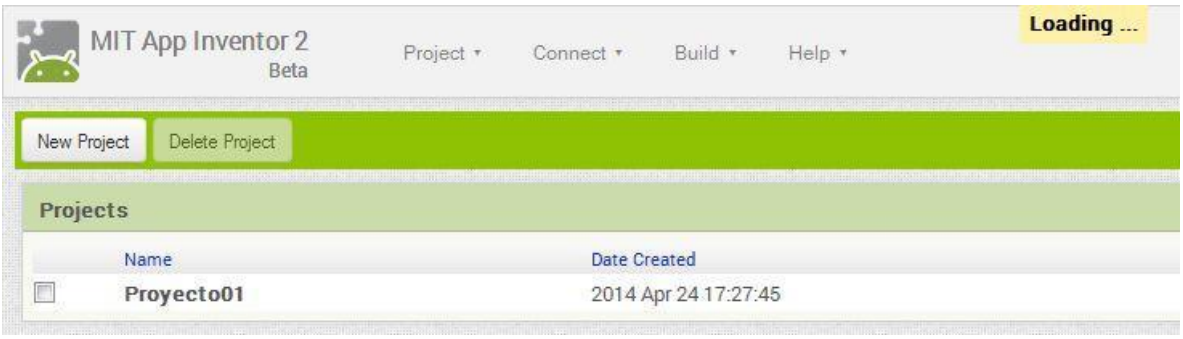
Simplemente **ignoramos** este **cartel** y nos disponemos a comenzar. Para esto, bastará cliquear en cualquier parte de la página, por lo que vamos a **cliquear** directamente en el botón **New Project**, que se encuentra en la **esquina superior derecha**, debajo del logo del **MIT App Inventor 2**.

Creación de un nuevo proyecto

Una vez que cliqueamos en **New Project**, nos parecerá automáticamente un cuadro de diálogo en el cual vamos a indicar el **nombre** que llevará nuestro proyecto, supongamos por caso **Proyecto01**:



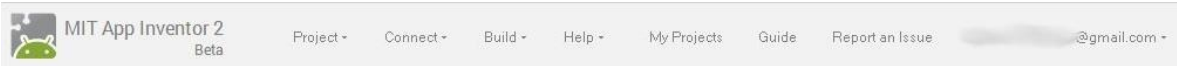
Al presionar la tecla enter, veremos por unos segundos el nombre de proyecto recién creado, dentro del espacio en cual nos van a aparecer **todos** los **proyectos** que vayamos creando, en la forma de un **listado**:



Debido a que, **sólo** contamos con este **proyecto** (por ser el primero), este se abrirá **automáticamente**, con lo cual estaremos frente a una **pantalla** como la que sigue a continuación, a la cual hemos agregado algunas **descripciones generales**:

Características Generales del Entorno de Diseño

Cómo podemos apreciar, en la parte superior de la página se encuentra una **barra de menús** de color gris:



Dichos **menús** son:

- **Project** (Proyecto)
- **Connect** (Conectar)
- **Build** (Construir)
- **Help** (Ayuda).

En dicha **barra** también aparecen algunos **links** a las funciones más usadas. Estos son:

- **My Projects:** nos devuelve a nuestra **lista** de proyectos.
- **Guide:** Aquí encontraremos **información** de soporte o **ayuda**.
- **Report an Issue:** Sirve para **informar errores** en el funcionamiento de la página.

Por último, nuestro **nombre de usuario**, donde al cliquear podremos **cerrar** la sesión (**Sing out**).

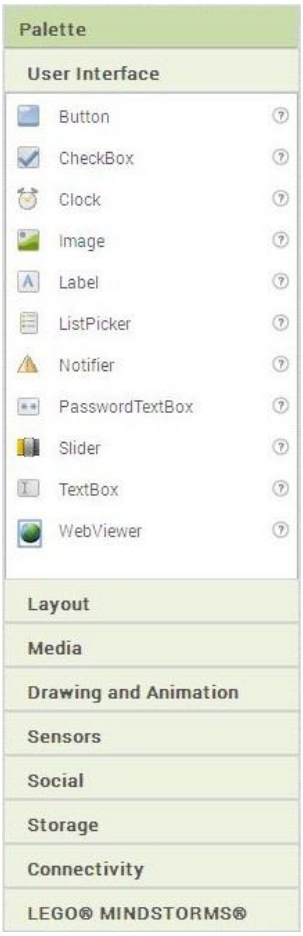
Todo esto lo veremos con más detalle a medida que lo vayamos usemos en módulos posteriores.

Seguido a esto, encontramos una **barra de menús** color verde con el **nombre del proyecto** y algunos **botones**:

Con el primer botón (**Screen1**) podremos seleccionar en cuál de las **pantallas** de la **App** queremos trabajar en ese momento, para los casos en los que contemos con más de una **pantalla** o **Screen**.

El siguiente botón (**Add Screen**), sirve para agregar una **nueva pantalla** al proyecto.

Luego tenemos el botón **Remove Screen**, el cual se encuentra **sombreado**, ya que en este caso, contamos con **una sola** pantalla (nuestras aplicaciones requieren al menos de una pantalla para funcionar).



Los últimos dos botones, los cuales aparecen en el extremo derecho, son **Designer** y **Blocks**, los cuales sirven para cambiar del **entorno de diseño visual** de la **App,** al **entorno de programación por bloques**, respectivamente. De momentos nos encontramos en el entorno de diseño, por lo cual el botón **Designer** se encuentra sombreado.

A continuación y del lado izquierdo encontraremos a la columna **Palette** (paleta) la cual contiene todos los **Componentes** que podremos agregar a nuestra aplicación.

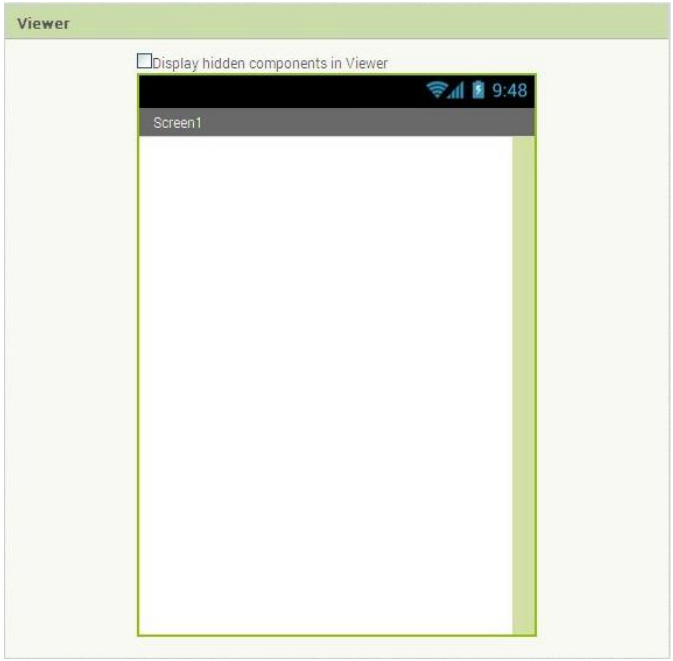
La misma se encuentran dividida en diferentes submenús, los cuales contienen los diferentes tipos de componentes que podremos agregar, algunos de los cuales, como por ejemplo **Clock,** no serán visibles al usuario. Veremos que quiere decir esto más adelante.

Entre los submenús de los cuales disponemos, haremos uso de los siguientes:

- **User Interface:** Aquí encontraremos todos los componentes de interfaz de usuario, que son los que nos van a permitir interactuar con la aplicación cuando la estemos usando.
- **Layout:** Para determinar la disposición de los componentes visibles.
- **Media:** Aquí se encuentran los componentes multimedia.
- **Drawing and Animation:** Para dibujo y animación.
- **Sensors:** Con estos podremos leer los sensores internos del dispositivo (GPS, acelerómetro, brújula, etc.).
- **Social:** Componentes para el uso de herramientas sociales.
- **Storage**: Para el almacenamiento de información.
- **Connectivity:** Componentes para el uso de las interfaces de conectividad del dispositivo (Bluetooth, web, etc.)

Los componentes **LEGO MINDSTORMS** son sólo útiles en el caso de contar con el juego de robótica para niños del mismo nombre, por lo que **no los vamos a utilizar**.

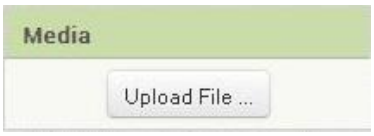
Al lado de dicha columna y en el centro del entorno, encontramos la ventana **Viewer** o **Visor**.



Aquí es donde vamos a **colocar** los distintos **componentes** antes mencionados, de modo que nuestra aplicación vaya adquiriendo el **aspecto** y la **forma** deseados. De este modo, tendremos una **vista preliminar** de cómo se verán las distintas pantallas de dicha **App** una vez terminada.

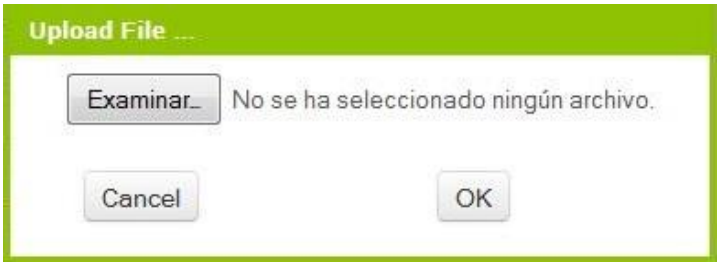
A continuación encontraremos la columna **Components** (Componentes), que es donde aparecerá la **lista** de todos los componentes que vayamos agregando, ya sean **pantallas** o **módulos** (sean estos **visibles** o **no** al usuario), o cualquier otro componente.

Debajo de dicha columna se encuentra el cuadro **Media** (Multimedia).



A través de este cuadro, podremos subir **archivos multimedia** (como por ejemplo **imágenes**, **sonidos**, **videos**, etc.) los que luego tendremos disponibles para utilizar dentro de los componentes que hayamos agregado.

Al cliquear en al botón **Upload File**, nos aparecerá un cuadro de dialogo, el cual nos permitirá por medio de su botón **examinar,** buscar dentro de la **PC** los archivos que deseamos integrar a nuestra **App**.

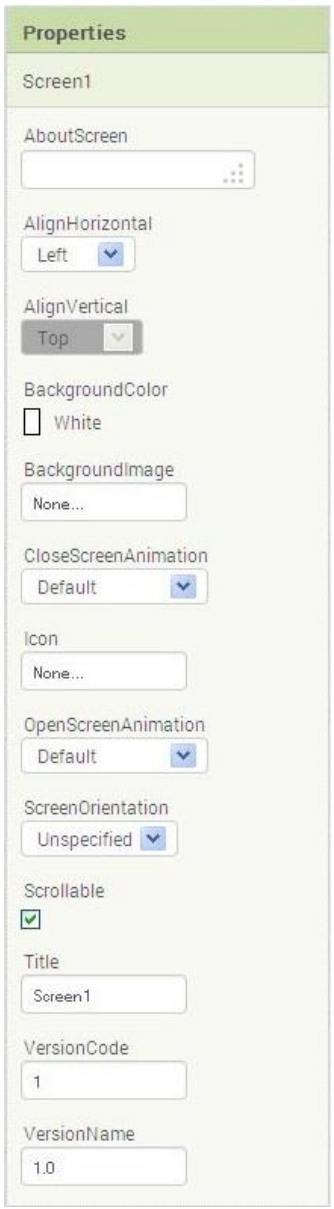


Al cliquear en dicho botón se nos abrirá una ventana de exploración de archivos, donde podremos seleccionar el archivo de deseado. Una vez seleccionado, daremos clic en aceptar, para retornar al cuadro de diálogo, donde nos aparecerá el nombre del archivo que vamos a subir:

Luego, cliquearemos en OK y el archivo comenzará a subir a la página de **App Inventor**.

Continuando, veremos que a la derecha de **Components** encontramos la columna **Properties** (Propiedades), la cual contiene todas las propiedades del elemento que hayamos seleccionado dentro de **Components**.

En este caso podemos apreciar las propiedades del componente **Screen1**. Entre las más importantes podemos mencionar:



- **BackgroundColor:** Color de fondo. Aquí podremos seleccionar el color de fondo que deseamos para la pantalla actual.

- **BackgroundImage:** Imagen de fondo. En caso de querer colocar una imagen de fondo para la pantalla, podremos seleccionarla desde aquí.

- **Icon**: Podremos cambiar aquí el **ícono** que usará nuestra aplicación, el cual debemos subir previamente como cualquier otro archivo multimedia. Este icono es el que se verá en el dispositivo Android una vez que la App haya sido instalada en él.

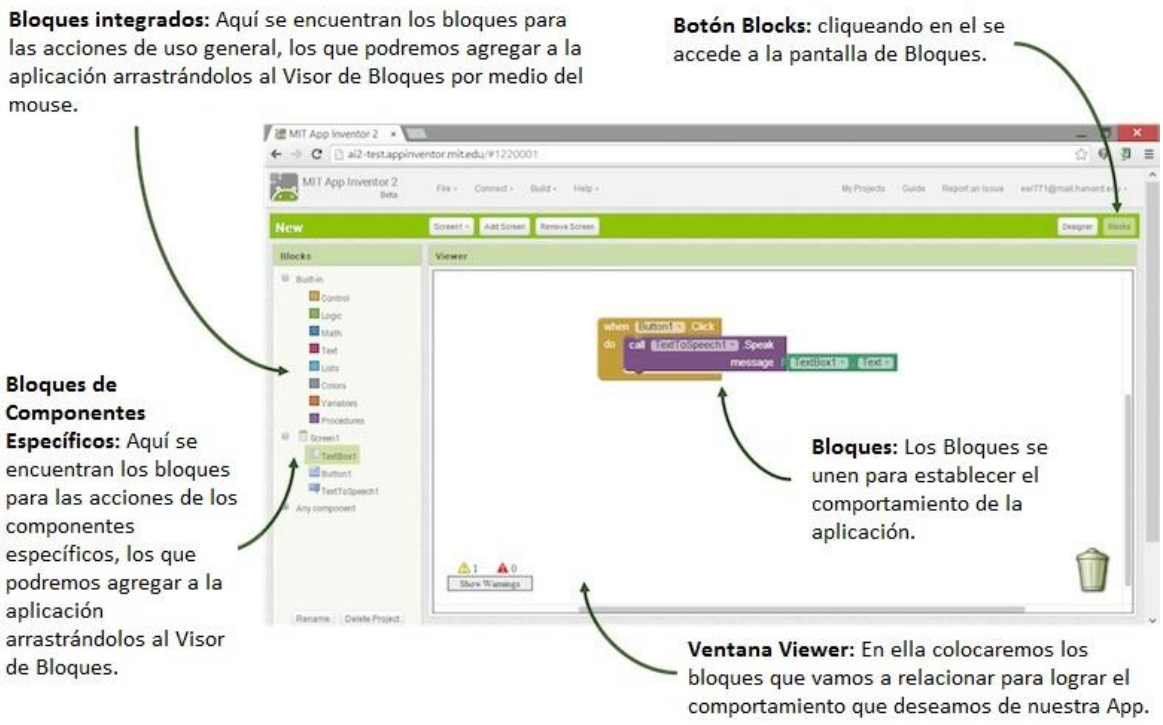
- **Scrollable**: Tildando o destildando esta casilla determinaremos si el usuario podrá desplazar o no la pantalla al deslizar el dedo sobre ella.

- **Title**: Aquí podremos cambiar el título que deseamos para la pantalla. Este es el título que verá el usuario en el dispositivo.

Más adelantes veremos estas y otras cuestiones con mayor detalle.

Características Generales del Editor de Bloques

Si ahora cliqueamos en el botón **Blocks**, veremos la siguiente pantalla, a la cual hemos agregado algunas descripciones rápidas:



Como se puede apreciar aquí, seguimos disponiendo tanto de la **barra de menú gris**, como de la **barra verde** antes mencionadas, pero en este caso vamos a contar con columnas y ventanas **diferentes**.

A la derecha del entrono de programación encontramos la columna **Blocks** (Bloques). En ella encontraremos distintos tipos de bloques, agrupados por sus **características** comunes dentro de distintos menús.

Debajo del la columna **Blocks**, encontraremos nuevamente al cuadro **Media**, el cual sirve para incluir archivos multimedia, y se utiliza de la misma que en el entorno de diseño.

A la derecha de **Blocks** encontraremos una nueva **ventana Viewer**, la que cumplirá la función de contener los **bloques** con los que vamos a **programar** la aplicación.

Conviene recalcar aquí, que la programación que llevemos a cabo dentro de la **ventana Viewer**, lo será para una **pantalla** o **Screen** determinada. Esto quiere decir que tendremos que seleccionar a cuál de las pantallas corresponderán los **bloques** que iremos agregando en dicha ventana.

Para cambiar de pantalla, cliqueamos sobre del botón **Screen'N'** (donde **'N'** es el número de la pantalla actualmente seleccionada), que encontraremos en la **barra de menús verde**. Se nos desplegará un menú donde podremos llevar a cabo dicha selección.

Sobre los bloques de programación

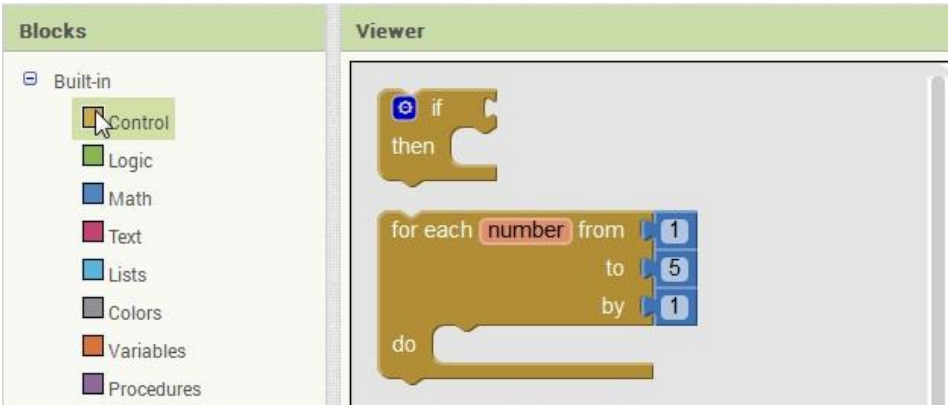
Veremos que dentro de la columna **Blocks** hay una serie de menús, conocidos como **drawers** (cajones). En primer lugar encontraremos el menú **Built-in** (o de bloques integrados).

En segundo lugar, encontraremos el menú **Screen1**. En el encontraremos los **bloques específicos** para trabajar con los distintos **componentes** que hayamos agregando dentro del **entorno de diseño**. Por último, encontraremos el menú **Any Components**, que contiene bloques que nos permitirán trabajar con varios **componentes** del mismo tipo a la vez.

Bloques de Control (Control)

Los bloques de control son aquellos que nos permitirán determinar el **comportamiento** de la **App** a partir del cambio de ciertos **estados** o **condiciones**.

La **App** "percibirá" dichos cambios a través distintos elementos, como puede ser el cambio del valor de una **variable**, el cambio de estado de un **sensor** (**AccelerometerSensor**, **LocationSensor**), una **acción del usuario** (un toque en la pantalla), **sucesos de tiempo** (componente **Clock**), etc.



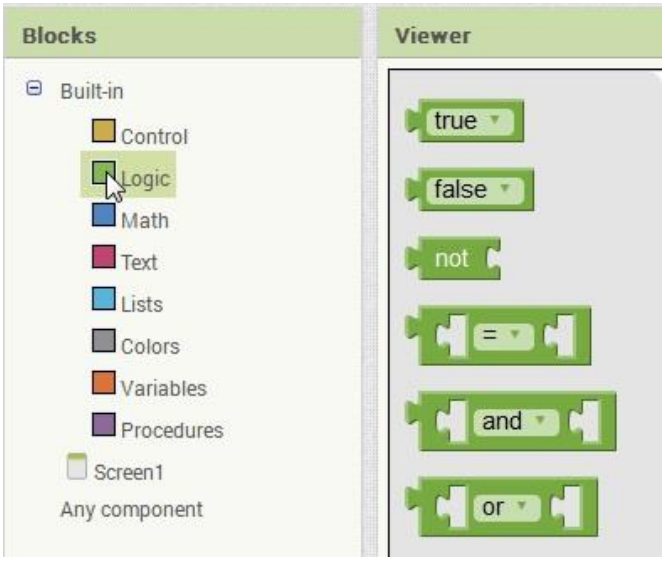
Bloques lógicos (Logic)

Con este tipo de bloques podremos realizar lo que se conocen como **operaciones lógicas**, o sea, operaciones entre distintos **valores de verdad** (**verdadero** o **falso**).

Con ellos podremos crear relaciones entre distintos elementos con el fin de detectar las **condiciones** deseadas para llevar a cabo una determinada **acción**. Usaremos dichas relaciones dentro de los **bloques de control**, de manera que el **comportamiento del programa** sea más complejo y acertado a nuestras expectativas.

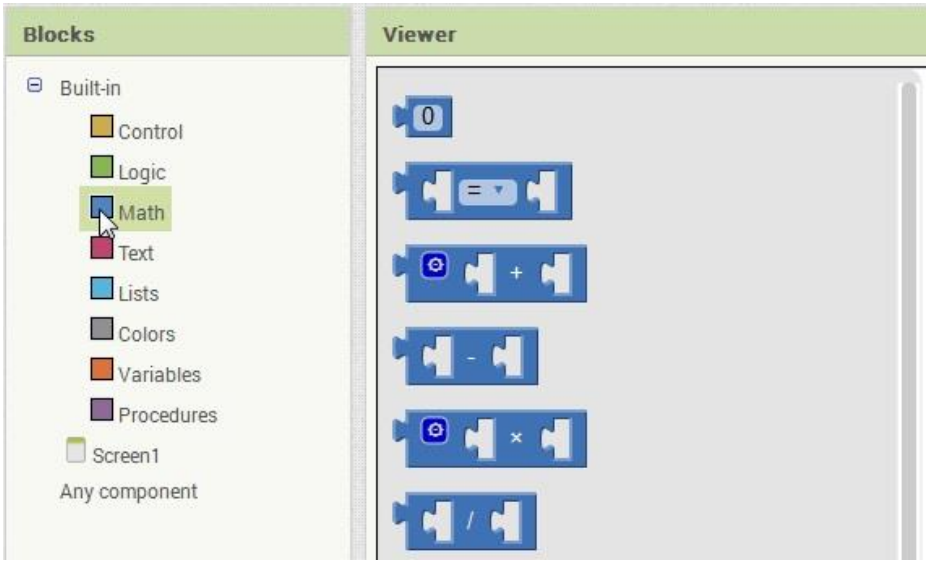
Podemos pensar en su funcionamiento, de la misma forma que cuando realizamos operaciones de conjuntos, en donde un determinado elemento pueden estar dentro de un conjunto **y a la vez** dentro de otro (**and**), en **uno u otro** de ellos (**or**), o ser igual (**=**), mayor (**>**), menor (**<**) o distinto (**≠**) que otro elemento.

También tenemos bloques que nos permiten asignar valores lógicos **verdaderos** (**true**), valores lógicos **falsos** (**false**), o invertir el valor de verdad encontrado (**not**), o sea, que se convierta en **falso** un valor **verdadero** y viceversa (que se convierta en **verdadero** un valor **falso**).



Bloques matemáticos (Math)

Entre estos bloques encontraremos los necesarios para realizar todo tipo de operaciones aritméticas, como ser **suma** y **resta**, **multiplicación** y **división**, **raíz cuadrada** y **potenciación**, etc.



Vamos a encontrar también bloques con funciones especiales, como aquellos del tipo **random**, que nos van a permitir generar números aleatorios, o los bloques **sin**, **cos** y **tan** para resolver funciones trigonométricas.

Bloques de texto (Text)

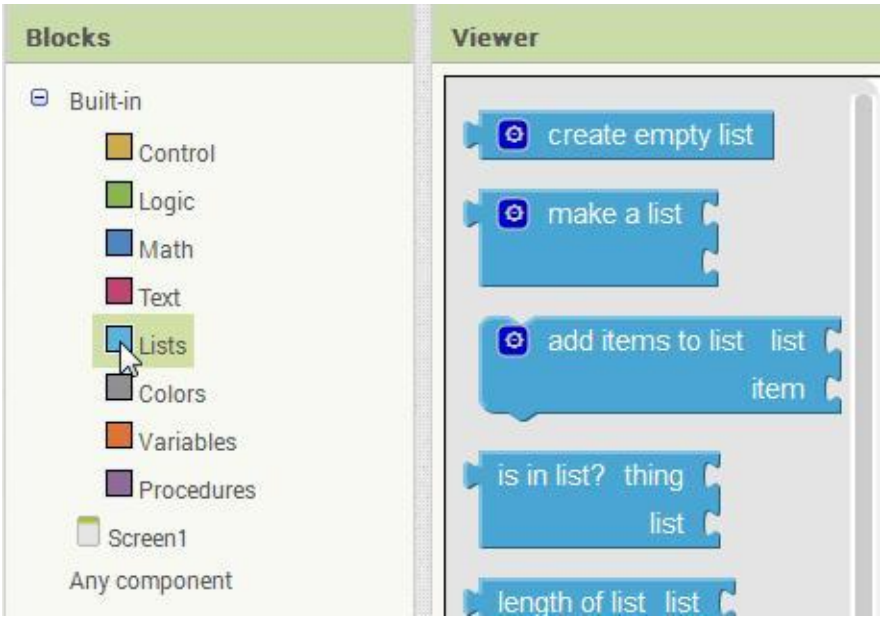
Los bloques de texto nos resultaran muy útiles para realizar operaciones sobre lo que se conoce como **cadenas de caracteres** (o **strings**).

Podemos pensar en una cadena de caracteres como en una **oración**, la cual está formada por cualquier tipo de carácter (**letras**, **espacios**, **números**, **comas**, **puntos**, etc.) en **cualquier orden**, y cuyo largo puede ser tanto de **uno** como de **muchos** de los mismos.

En la lista de la imagen podemos ver que el primero de ellos tiene la forma "_". Dicho bloque nos permite ingresar cualquier palabra o conjunto de caracteres que queramos utilizar dentro de la App cuando esta se encuentre corriendo dentro de un dispositivo.

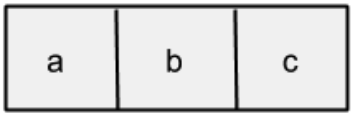
Bloques para el Manejo de Datos por listas (Lists)

Estos bloques nos van a permitir crear y manipular diferentes conjuntos de **datos**, **valores** o **elementos**, agrupándolos dentro de los que conoceremos como listas (lists). Las listas son un tipo de estructura de datos muy utilizada en todo tipo de lenguajes de programación, no sólo App Inventor.



Como ejemplo, podemos pensar en un juego que puede contener una lista de los puntajes más altos o en la web de Facebook que mantiene una lista de sus amigos.

Para determinar la posición de un elemento en una lista se utilizan lo que conoceremos como índices (index). En App Inventor, el primer elemento de una lista tendrá siempre el índice 1. Por ejemplo, en la siguiente figura, la letra 'a' tendrá el índice '1', la letra 'b' el '2', mientras que la letra 'c', el '3'.

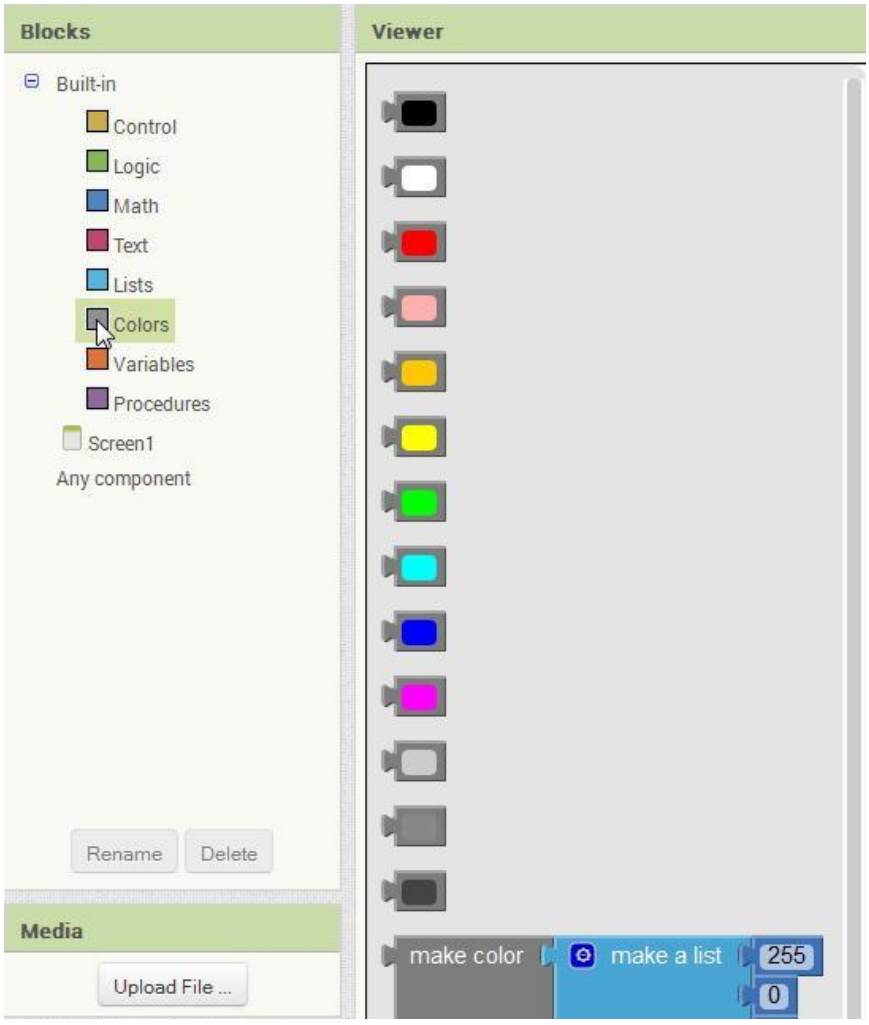


Esto quiere decir, que podremos hacer **referencia** a un **elemento** determinado dentro de una determinada lista, sabiendo el **nombre** de la misma y el **número** de índice del elemento buscado. Profundizaremos al respecto más adelante.

Bloques de color (Colors)

Estos bloques nos permitirán hacer cambios en la selección del **color** de determinados elementos dentro de nuestra aplicación.

Veremos más adelante que muchas de las propiedades (las que aparecen en la columna **Properties**), pueden modificarse directamente dentro del editor de bloques. Un ejemplo de esto sería el **color de fondo** de la pantalla actual, el cual podremos cambiar utilizando los bloques a continuación mostrados:

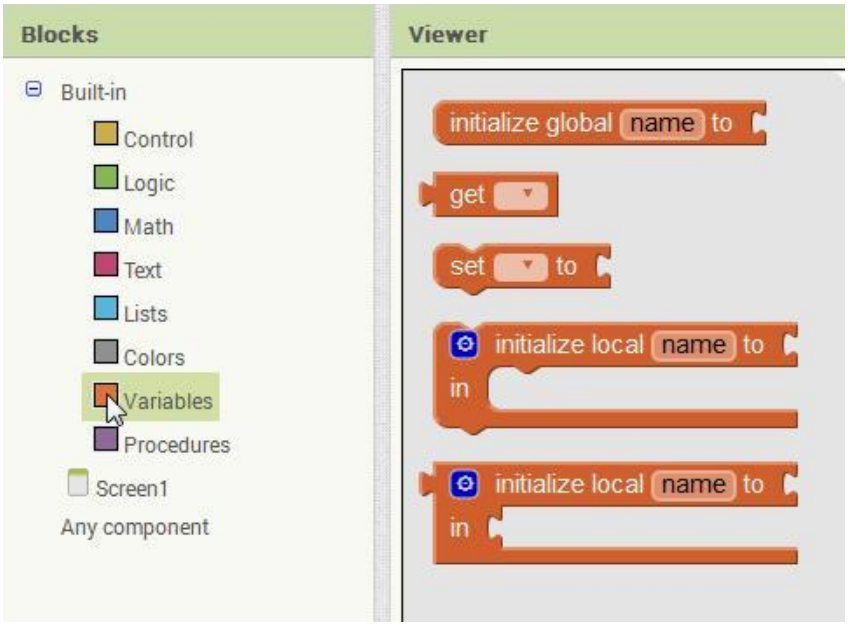


Podremos crear también colores personalizados, por medio del bloque **make_color**, al cual le "pasaremos" **tres valores numéricos** de entre **0** y **255**, indicando la cantidad de color **rojo**, **verde** y **azul** que deseamos mezclar.

Bloques para el uso de variables (Variables)

Encontraremos aquí los bloques que nos permitirán **crear** (**initialize**), **llamar** (**get**) y **modificar** (**set**) lo que en programación se conocen como **variables**. Como dijimos anteriormente, podemos pensar en las **variables** como "cajitas" en las cuales vamos a **guardar** la información que vamos a necesitar posteriormente para realizar una **determinada acción** dentro de un **bloque** o **secuencia de bloques**.

Crearemos nuestras **variables** con el uso de los bloques del tipo **initialize**, que quiere decir **inicializar**. En programación se dice que se **inicializa** una **variable**, cuando se la **crea**, se define su **tipo** y se le **asigna un valor**.



Podemos dividir a las variables en dos grandes grupos; **variables globales** (**global**) y **variables locales** (**local**).

Las **variables globales** son "visibles" dentro de **todo el programa**; esto quiere decir que las podemos **leer** o **modificar** dentro de **cualquier bloque** y en **cualquier momento**, y conservan su valor **hasta la próxima** modificación.

Las **variable locales**, por el contrario, serán sólo visibles para **leerlas** o **modificarlas** dentro del **bloque** en el que se las **creo**, y conservan su valor mientras el programa no salga de dicho bloque. Una vez fuera de este, la variable **desaparece** junto con su **valor**.

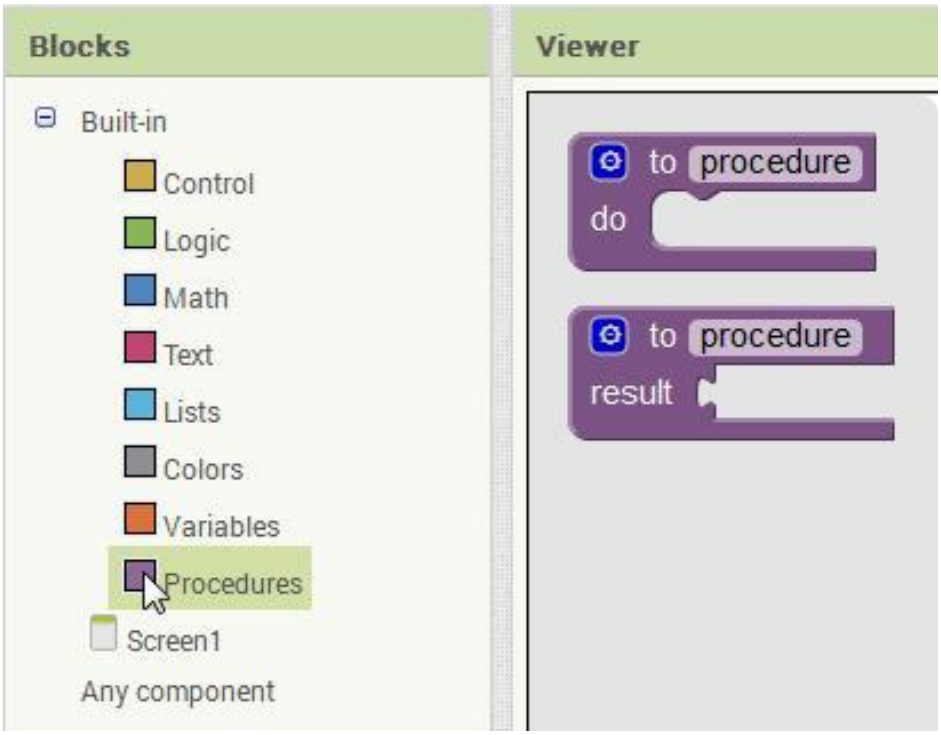
Si el bloque mencionado vuelve a ser ejecutado, la **variable local** vuelve a ser **inicializada**, para ser utilizada dentro de este hasta el momento en que su ejecución termine.

Bloques de procedimiento (Procedures)

Un procedimiento (**Procedure**) es una **secuencia de bloques**, o pieza de código, que se agrupa dentro de un mismo **nombre**. Este nombre lo determinaremos al momento de crear el procedimiento.

Esto permite que en lugar de tener que colocar la misma **secuencia de bloques** una y otra vez, podamos crear un **procedimiento**, el cual contenga todos los bloques tan **sólo una vez**, para luego **llamar** a dicha secuencia por medio de su **nombre**, todas y cada una de las veces que necesitemos que la misma sea ejecutada.

En informática, a un procedimiento también se lo conoce con los nombres de **función** o **método** (**method**).



Bloques Especiales

Cómo podremos apreciar en la siguiente imagen, existen bloques especiales para determinados componentes, en este caso, la pantalla en la que estamos trabajando.

Estos bloques nos dan acceso a funciones que de otro no podríamos utilizar, como por ejemplo, el ejecutar una secuencia de bloques especifica al momento del inicio de la pantalla (**when_Screen1_initialize_do**), o qué sucederá cuando el usuario presione la tecla volver (**when_Screen1_BackPressed_do**).

Por otro lado, también contaremos con bloques de programación específicos para los componentes que hayamos agregado dentro del entorno de diseño (**designer**).

Estos bloques aparecerán agrupados por componente y a continuación del componente de pantalla a la que han sido agregados (en este caso, **Screen1**).

Por último, encontraremos el grupo **Any component** (Cualquier componente), que contiene **bloques** que nos permitirán trabajar con varios **componentes** del mismo tipo a la vez, y los iremos mencionando a medida que los vayamos usando.

Una vez que hayamos configurado la **PC**, y que estemos familiarizados con las funcionalidades generales del **entorno de diseño** y del **editor de programación por bloques**, estaremos listos para dar los primeros pasos en la construcción de nuestras **Apps**.

Programación en App Inventor 2

Objetivo

El objetivo del presente módulo es que el alumno sea capaz de comenzar nuevos proyectos para la construcción de **Apps**, se interiorice en el uso y configuración de los componentes, aprenda a utilizar y combinar distintos bloques de programación, así como también conceptos básicos de programación propios de **App Inventor**.

Ejemplo de construcción de una App

Crear y Guardar un Nuevo Proyecto

Comencemos por crear un nuevo proyecto. Cliqueando sobre la barra gris en **Project** -> **Start new project**, e ingresando el nombre del proyecto, por caso "Proyecto01" (no se admiten espacios como parte del nombre).

Todos los cambios que realicemos a nuestro proyecto se guardaran automáticamente dentro del entorno (siempre que haya conexión), por lo que no debemos preocuparnos por "salvarlo" constantemente.

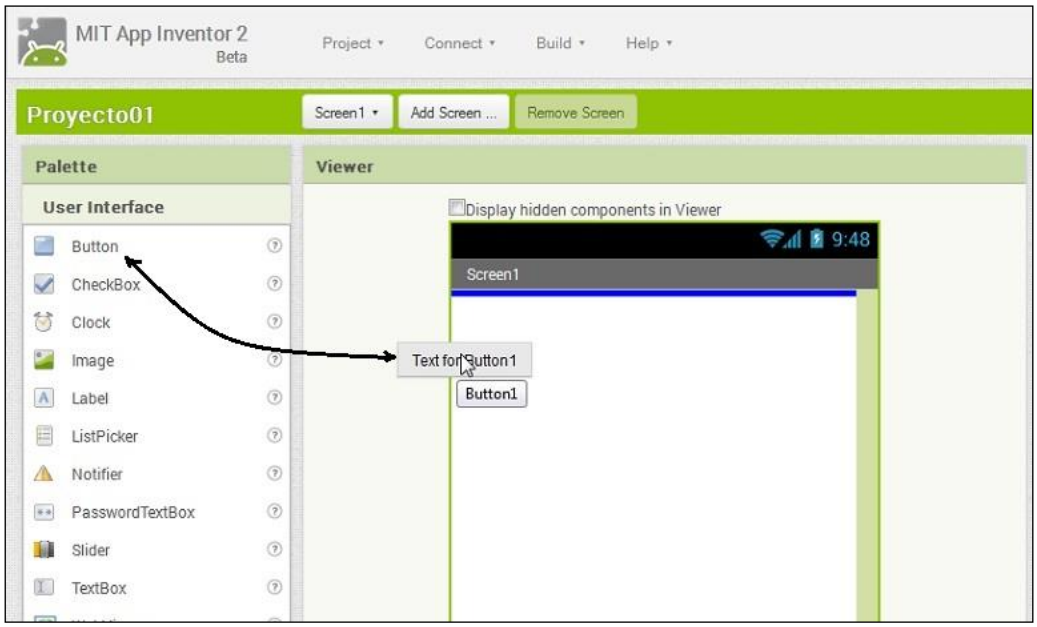
De cualquier forma, si deseamos salvar el proyecto manualmente antes de salir, bastará con cliquear sobre la barra gris, en **Project** -> **Save project**.

Concepto de componentes

Como comentamos anteriormente, los **componentes** que vamos agregar a nuestra **App** se encuentran en el lado **izquierdo** del **entorno de diseño**, dentro de la columna **Palette**. Dichos componentes son los **elementos básicos** que necesitamos para construir nuestras **Apps**.

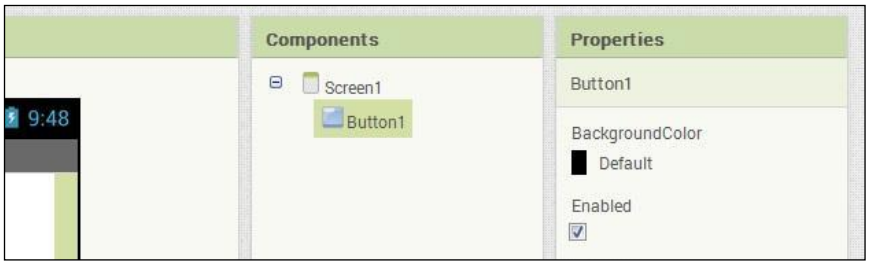
Estos son como los **ingredientes** de una receta. Algunos componentes son muy **simples**, como el componente **Label** (**etiqueta** o **leyenda**), el cual sólo muestra un texto en pantalla, o el componente **Button**, que al tocarlo ejecuta una serie de acciones.

Comencemos colocando uno de estos dentro de nuestra **App**. Para esto, cliquearemos sobre la opción **Button**, y mientras mantenemos cliqueado, desplazaremos el cursor sobre la pantalla **Screen1**, que está dentro de la ventana **Viewer**:

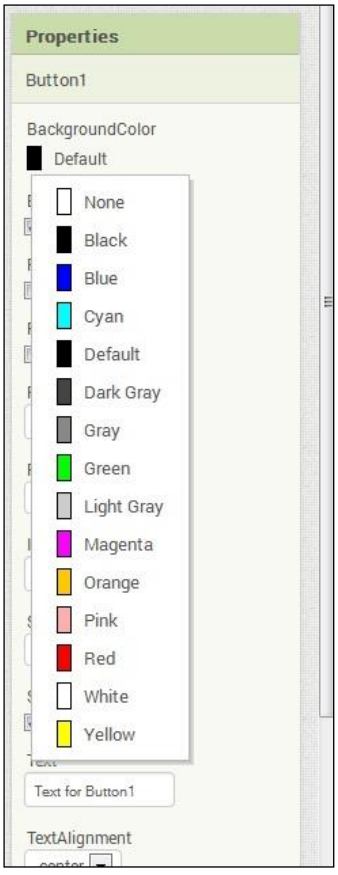


Concepto de Propiedad de un componente

Una vez colocado dicho botón, veremos como en el cuadro **Components** aparece automáticamente el componente **Button1**, y en el cuadro **Properties**, las propiedades de dicho componente.



En dicho cuadro de propiedades, podremos cambiar (entre otras cosas), el **color de fondo** de dicho botón, haciendo clic en la etiqueta que aparece debajo de la opción **BackgroundColor**, que en el presente caso es **Default** (el color por defecto).



En la imagen a la derecha podemos apreciar un lista, en la cual podremos seleccionar el color de fondo que deseemos dar a nuestro botón. Dicho color será el que tenga el botón dentro de la App, cuando la iniciemos en un dispositivo **Android** o dentro del **emulador**; por esto decimos que ese será el **valor inicial** de la propiedad.

Existe otra forma de cambiar esta **propiedad** (la propiedad color), así como también las otras propiedades que presenta el botón **Button1**.

Esto nos va a resultar bastante útil para dar "vida" a nuestras aplicaciones, y es posible gracias al **Editor de Bloques** (**Blocks Editor**). A este accederemos, como ya lo hemos mencionado, cliqueando en el botón **Blocks** que se encuentra en la barra verde, a la derecha del botón **Designer**.

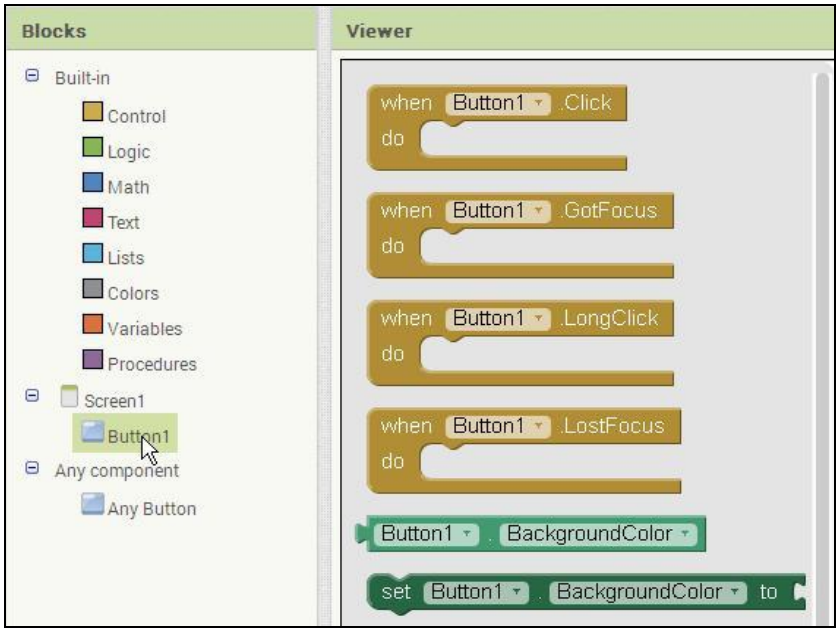
Bloques de programación generales y propios de los componentes

Una vez dentro del editor, veremos que en la columna **Blocks** aparecen nuevos elementos, tanto debajo de **Screen1** como de **Any component.**

Dichos elementos están relacionados (en este caso) con el **botón Button1**, el cual agregáramos en el **entorno de diseño**.

De este modo, cualquier otro componente que agreguemos a la pantalla **Screen1** aparecerá aquí, de manera que tengamos disponibles los **bloques de programación** y las funcionalidades que nos permitirán operar con ellos.

Cómo ejemplo, podemos ver que al cliquear en **Buttón1**, nos aparecerá lo siguiente:



Como podemos apreciar, el color de los bloques varía según la función. Los bloques de color mostaza son lo que llamaremos **bloques de control de eventos**, **controladores de eventos**, o simplemente "**eventos**", y su color se corresponde con el de los bloques de **control** de uso general. Podremos distinguirlos además, porque este tipo de bloques comienzan con la palabra **when…** (**cuando**…).

Los **bloques de control de eventos** especifican cómo debe responder el dispositivo a ciertos eventos: un botón ha sido **pulsado**, el dispositivo está siendo **sacudido**, el usuario está **deslizando** su dedo sobre la pantalla, etc.

Orientación a eventos y área de tareas

El espacio dentro de estos bloques de **control de eventos**, el cual aparece marcado como "**do**", es lo que llamaremos **área** o **espacio de tareas**. Aquí colocaremos los **bloques** que deseamos se **ejecuten** cuando suceda el **evento esperado**.

Debido a esta característica particular, donde para realizar una acción primero tiene que suceder algo, decimos que en **App Inventor** el lenguaje de programación está **orientado a eventos**.

Continuando con la programación, seleccionamos en este caso el bloque **when_Button1.Click_do**, cliqueando sobre él para luego arrastrarlo y soltarlo dentro de la ventana **Viewer**.

Getters y Setters para Propiedades

Centremos ahora nuestra atención en los bloques **Button1.BackgroundColor** y **set_Button1.BackgroundColor_to**. Obsérvese que si bien estos tienen nombres similares, los mismos tienen funcionalidades distintas.

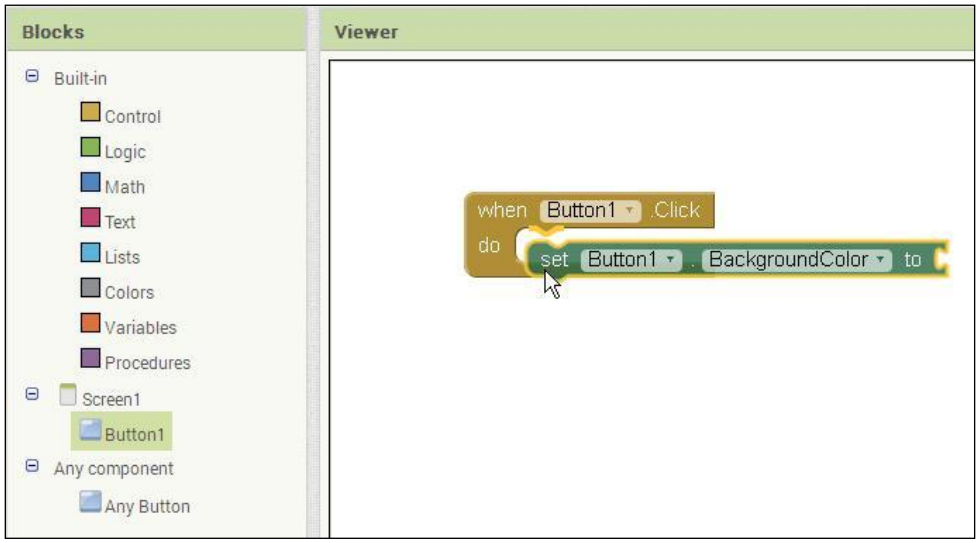
Al primero de estos se lo conoce como **getter** (de obtener), y de volverá el valor actual de la propiedad **BackgroundColor** (que en un primer momento será igual a **default**), mientras que al segundo se lo conoce como **setter** (de establecer), y nos permitirá definir ("setear") un nuevo valor de color para dicha propiedad.

Unión de Bloques

Tomemos ahora el bloque **set_Button1.BackgroundColor_to** y coloquémoslo dentro de la **sección de tareas** (**do**) del bloque **when_Button1.Click**.

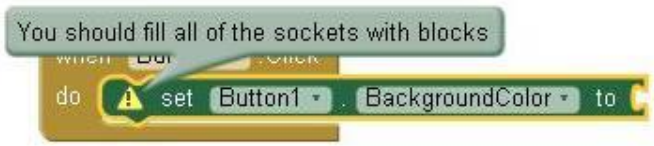
Los bloques se conectan entre sí como las piezas de un rompecabezas. Al aproximar este bloque al conector correspondiente, aparece lo que podríamos llamar un "halo" amarillo.

Este nos indica cual será la posición en la que terminará el bloque **set_Button1.BackgroundColor_to** una vez que soltemos el clic de mouse.

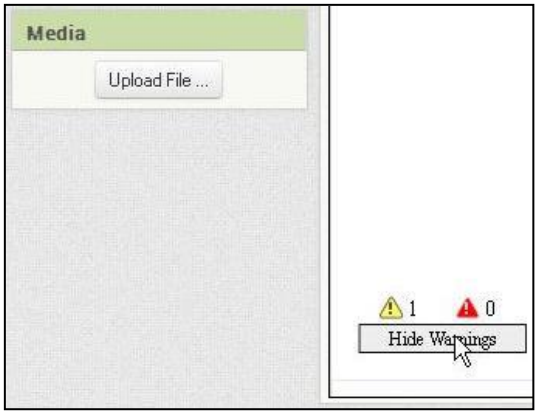


Advertencias en App Inventor

Una vez colocado, puede suceder que en dicho bloque aparezca un **signo de advertencia** (⚠). Al cliquear sobre este, veremos un mensaje que nos informa que debemos completar todas las **entradas** o **encastres** (**sockets**) vacíos de dicho bloque:

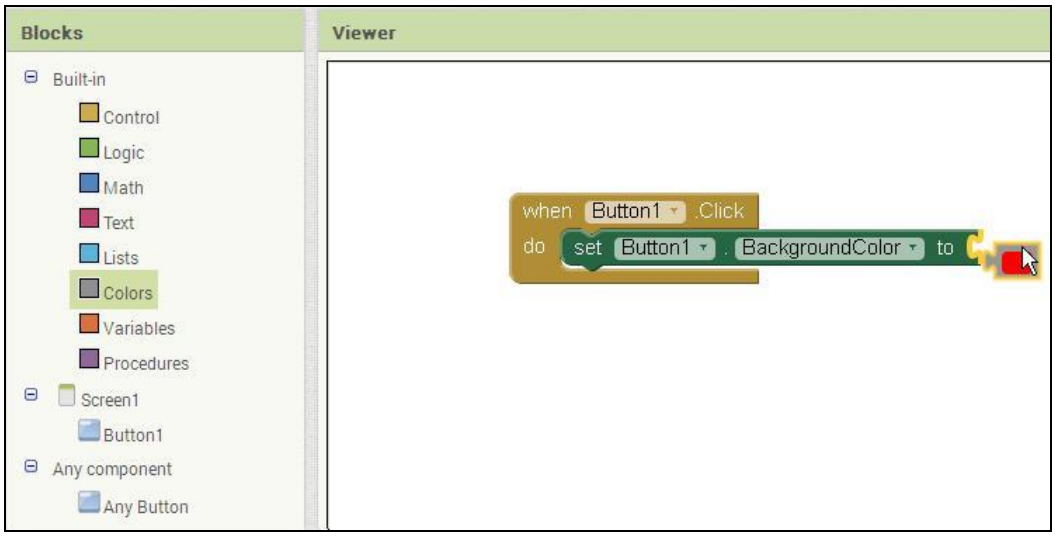


El mismo **desaparecerá** una vez que completemos el bloque, pero también es posible "esconderlo" cliqueando sobre la leyenda "**hide warnings**", que se encuentra en la **parte inferior izquierda** de la ventana **Viewer**.

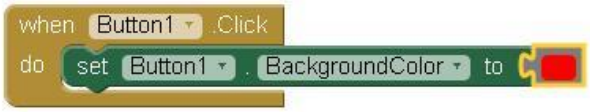


Asignar un Nuevo Valor a un Bloque

Continuando con la programación, seleccionaremos ahora un nuevo color para el fondo del botón **Button1**. Para ello, cliquearemos en **Colors** y luego sobre cualquiera de los colores, por ejemplo el **rojo**, y lo arrastraremos hasta alcanzar el **encastre** o **entrada** (**to**) que se encuentra a la derecha del bloque **set_Button1.BackgroundColor**.



El resultado final será el siguiente:



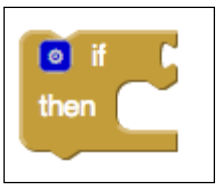
El funcionamiento es muy sencillo. Cuando la App se esté ejecutando y el usuario **toque el botón** en la pantalla, este cambiara del color por **defecto** (**default)** al color **rojo** y permanecerá así hasta que cerremos la aplicación.

Esto es lo que podemos llamar **modificación dinámica de propiedades**, ya que cambiamos el aspecto de los componentes de la App dentro del tiempo de ejecución de la misma.

Uso de los Bloques de Control IF

Ahora vamos a **modificar** un poco la aplicación para hacer que su **comportamiento** sea algo **más complejo**. Para esto, vamos a utilizar un tipo de bloques de control conocidos como **if_then** (si…entonces…) e **if_then_else** (si…entonces…sino…).

A continuación podemos ver un bloque **if_then**. Lo que hace este bloque es comprobar la condición colocada en el encastre **if**; **si** la condición es **verdadera**, realiza las acciones que se encuentran dentro del espacio titulado como **then**; de lo contrario, dichas acciones son **ignoradas**.



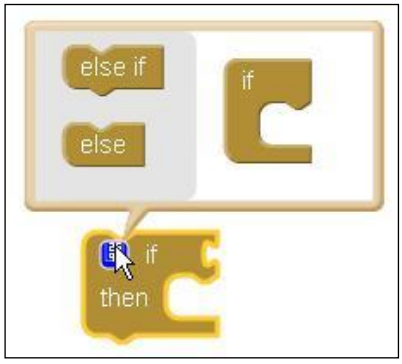
Concepto de bloque Mutador

Como se puede ver en la figura, este bloque tiene un **cuadrito azul** en su esquina **superior izquierda** con un simbolito adentro ( ). Esto es debido a que el mismo pertenece a un tipo particular de bloques dentro de **App Inventor** conocidos como **Mutators** o **Mutadores**.

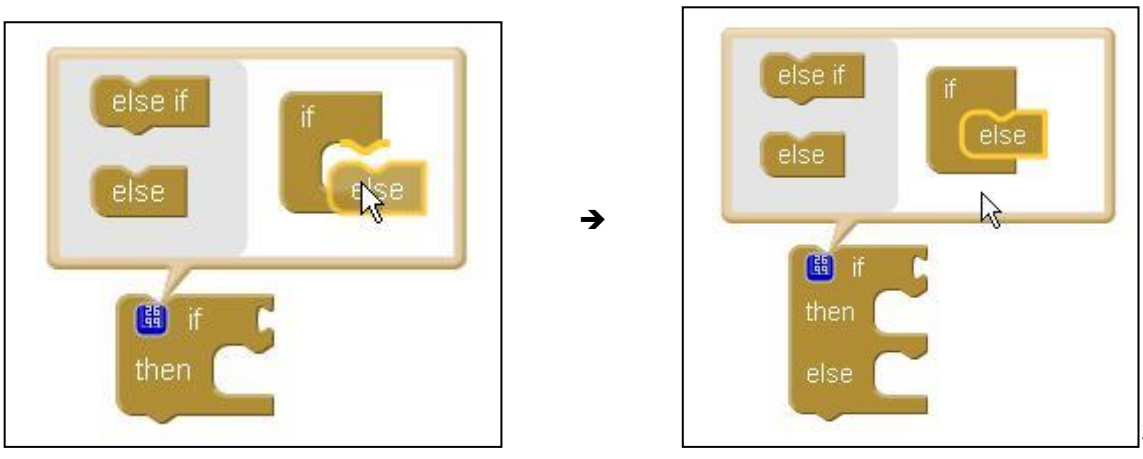
Si hacemos clic en dicho cuadro, podremos ver que nos aparecerá una burbuja dividida en dos. Dentro de esta burbuja podremos ver una serie de "pseudo-bloques" que representan ciertas características del **bloque mutador**.

Hemos dado en llamar aquí a estos bloques como **pseudo-bloques**, para diferenciarlos de los bloques de programación propiamente dichos, ya que cumplen una función diferente.

En la parte de la derecha del globo que aparece, vamos a encontrar un pseudo-bloque que representa las **entradas** o **encastres** que ya tenemos disponibles actualmente; en este caso, el encastre **if**. En la parte de la izquierda se encuentran dos pseudo-bloques que representan **nuevas entradas** y/o **funcionalidades** que vamos a poder agregar al bloque con el que estamos trabajando.

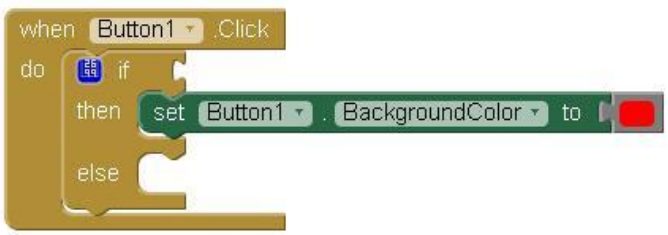


De este modo, si por ejemplo agregáramos el **pseudo-bloque else**, nos quedaría un bloque del tipo **if_then_else:**



El nuevo bloque funcionará de la siguiente forma: Se comprueba la condición informada en el encastre **if**. **S**i el resultado **es verdadero**, se ejecutan los bloques dentro del **espacio de tareas then**; en caso contrario, se llevarán a cabo las acciones dentro del **espacio** titulado como **else**.

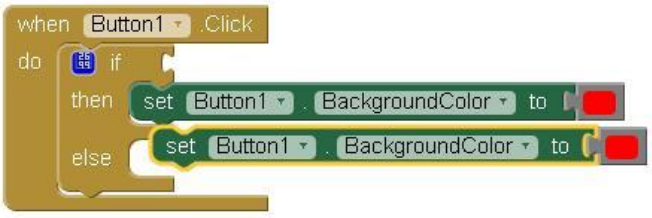
Continuemos ahora con el ejemplo anterior. Agregaremos un bloque **if_then_else** de la siguiente forma:



Duplicación de Bloques

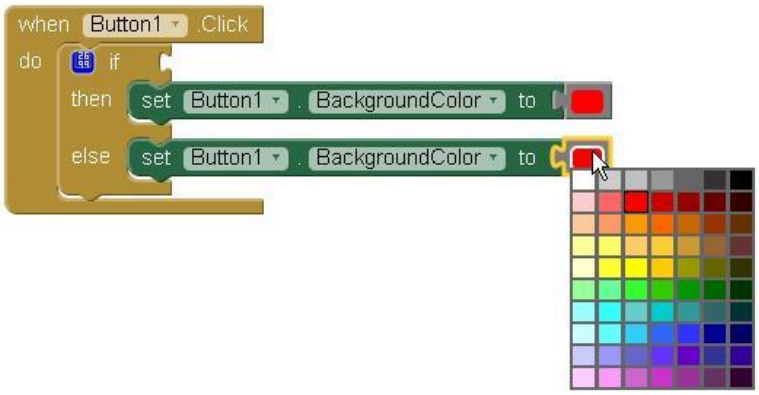
Luego, colocaremos un nuevo bloque **set_Button1.BackgroundColor**, pero con un color distinto. Podríamos buscar los bloques pertinentes **uno a uno** dentro de la columna **Blocks** otra vez, pero existe una manera más directa.

Lo que haremos será cliquear sobre el bloque **set_Button1.BackgroundColor**, presionar "**Ctrl + C**" y luego "**Ctrl + V**", de manera que nos aparecerá un nuevo bloque **set_Button1.BackgroundColor**, con el bloque de color ya adosado.



También podemos hacer clic derecho sobre el conjunto de bloques que se desea duplicar y seleccionar la opción **Duplicate** en el menú desplegable que nos aparece.

Una vez duplicado, bastará con que reacomodemos este nuevo conjunto de bloques **set_Button1.BackgroundColor** y que seleccionemos un nuevo color, cliqueando sobre el bloque de color que queremos cambiar:



Elegimos por caso el color verde y listo; Ya tendremos nuestro nuevo bloque **set_Button1.BackgroundColor** con su color correspondiente.

Inicialización y uso de Variables

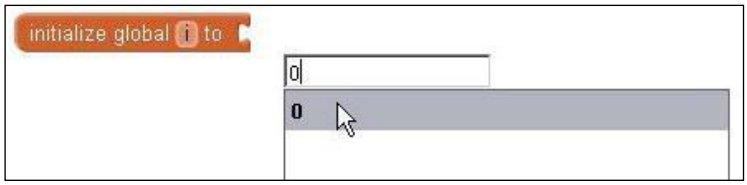
Ahora vamos a inicializar una variable de nombre '**i**', la cual vamos a utilizar para "seguir" los cambios en el **color de fondo** de nuestro botón. Tomaremos para esto un bloque **initialize_global_name_to** que está dentro de **Blocks**, en el submenú **variables**.
La expresión "**global**" nos indica que está será una variable del tipo global, lo que significa que dicha variable podrá **usarse**, **leerse** o **modificarse** dentro de **cualquier** secuencia de bloques que pertenezca a esa **pantalla** (o **Screen**), no así a las **pantallas restantes**.

Podremos cambiar el nombre de la variable cliqueando dentro del cuadrito **name**. Le colocaremos por nombre '**i**' y le daremos un valor igual a '**0**'. Esto lo lograremos colocando un **bloque numérico**, el cual podremos encontrar en el submenú **Math**.

El hacer esto (el darle un **nombre**, un **tipo** y un valor a una **variable**), es lo que en programación se conoce como **inicialización de una variable**.

Typeblocking

Existe otra manera obtener el **bloque numérico** para completar la inicialización de la **variable 'i'**. Lo que haremos para esto será cliquear en un espacio vacío del visor **Viewer**, y escribir el número '**0**' directamente con el teclado. Al hacer esto aparecerá un menú, mostrando todos los posibles bloques que incluyen '**0**' en su nombre.



Escribir en el área en blanco de la pantalla es lo que se llama typeblocking, y nos resultará muy útil como **atajo** para obtener **cualquier tipo** de bloque, no sólo los numéricos.

Haremos clic en el primero de ellos, que es el número '0' en sí, y automáticamente aparecerá un **bloque numérico** con un valor igual a **cero**. Enchufamos este al bloque **initialize_global_i_to** y listo.

Getters y Setters para Variables

Como se mencionó anteriormente, vamos a utilizar la variable '**i**' para seguir los cambios de color asignándole un valor determinado para cada caso. En el caso del color **rojo** vamos a asignarle el valor '**1**', y para el **verde**, el '**0**'.

Como sucede para el caso de las propiedades, tenemos también disponibles los bloques conocidos como **getters** y **setters** para el caso de las variables.

Los **getters** se utilizan para obtener el valor actual de una variable, y son aquellos en las que comienzan con la leyenda **get_** (obtener…), mientras que los **setters** se utilizan para asignar un nuevo valor de una variable, y comienzan con la leyenda **set_** (establecer…). Ambos tipos los podemos encontrar dentro del submenú **variables**.

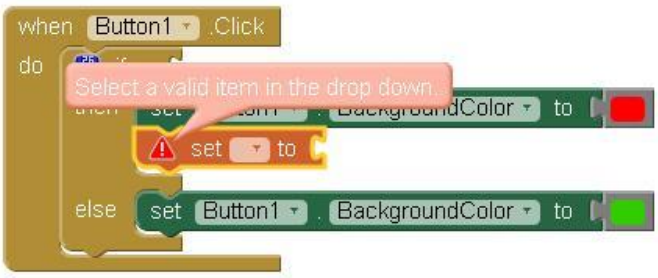
En nuestro caso, vamos a seleccionar un bloque **set_to**, el que luego modificaremos a fin de que haga referencia a la **variable 'i'**. Colocaremos dicho bloque a continuación del primer bloque **set_Button1.BackgroundColor** (el que tiene el color rojo).

Errores en App Inventor

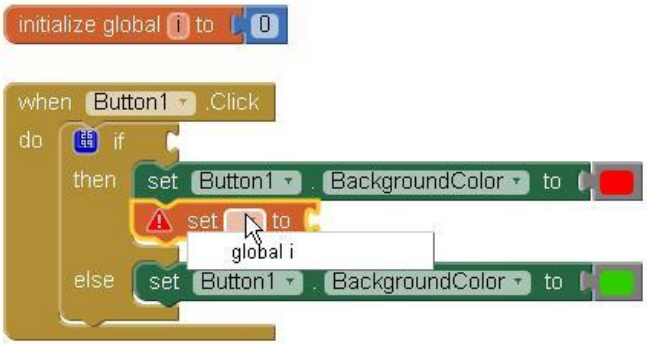
Luego de esto, veremos que sobre el bloque **set_to** aparece un **signo de error** (⚠). Este nos indica que existe algún tipo de error en la utilización de dicho bloque.

Dicho error será contabilizado junto a las **advertencias** en la esquina inferior izquierda de la ventana **Viewer** (sobre "**hide warnings**"). Estos no pueden ser ignorados, como sí sucede con los **warnings** (advertencias).

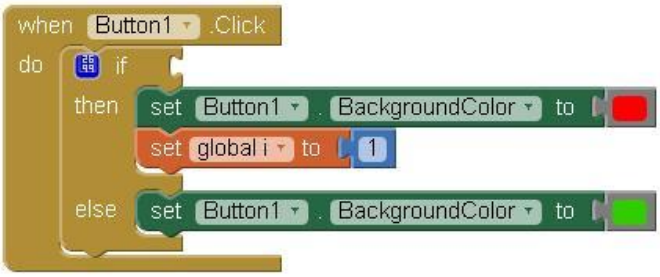
Al hacer clic sobre dicho símbolo de error, nos aparecerá un **globo** con información sobre lo sucedido:



En el presente caso, este nos indica que debemos seleccionar un **ítem valido** dentro del **menú desplegable**. El menú al que se refiere es el que nos va a aparecer al cliquear en la casilla que está al lado de **set**, la que tiene una pequeña flechita hacia abajo.

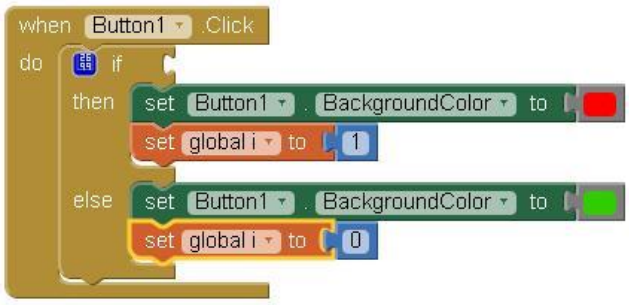


Al hacer esto nos, aparecerá una **lista** de las **variables** que tenemos en uso. Ya que la **única** que hemos inicializado es la **variable global 'i'**, será esta la **única opción** que se nos presentará. La seleccionamos y le asignamos el valor '**1**'.



Necesitamos colocar ahora otro bloque **set_global_i_to** (establecer_global_i_como…), debajo del segundo bloque **set_Button1.BackgroundColor**. Nuevamente, podremos

utilizar aquí la combinación "**Ctrl + C**" y "**Ctrl + V**" para obtener una copia del mismo, a la que luego la asignaremos un valor igual a '0'.



Secuencia de tareas

Como pueden ver, en ambos espacios de tarea, tanto **then** como **else**, hemos colocado los bloques siguiendo un orden específico. Conoceremos a estos conjuntos de bloques como **secuencia de bloques** o **secuencia de tareas**, debido a que la **App** ejecutará los mismos uno a uno y de arriba hacia abajo, o sea **secuencialmente**.

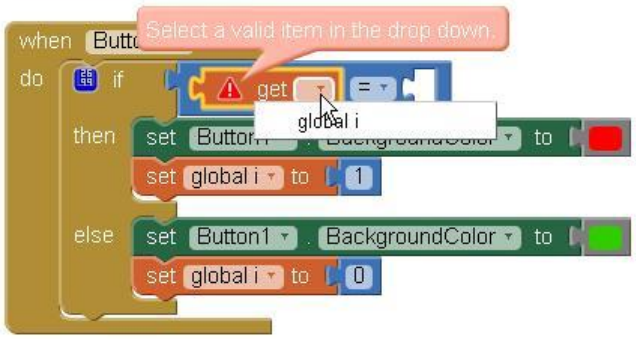
En este caso, no hubiese habido ninguna diferencia en el resultado final si hubiésemos puesto los bloques **set_global_i_to** antes de los **set_Button1.BackgroundColor**, debido a que no se afectan entre sí, pero es importante tenerlo en cuenta para casos distintos a este.

Bloque Comparador como Condición de Control

A continuación vamos a establecer la condición de control para el bloque **if**. Lo que haremos será comparar el valor de la **variable 'i'** con el valor '0'. Colocaremos el bloque "**igual**" del submenú **Math** en el encastre vacío de nuestro **if**. Si la igualdad se cumple, este bloque devuelve un valor **verdadero** (**true**), sino, devuelve un valor **falso** (**false**).

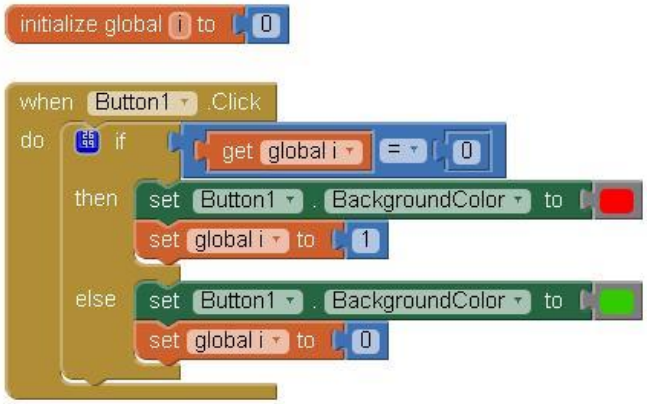
Dentro de esta comprobación de igualdad vamos a necesitar obtener el valor de la **variable 'i'**, por lo que necesitaremos utilizar un **setter**, o sea un bloque del tipo **get_** (obtener_).

El mismo se encuentra dentro del submenú **variables**. Lo tomaremos y colocaremos en el primer espació vacío del bloque "**igual**". Veremos que nos aparece nuevamente aquí un signo de error que nos indica que falta definir cuál es la variable que vamos a utilizar:

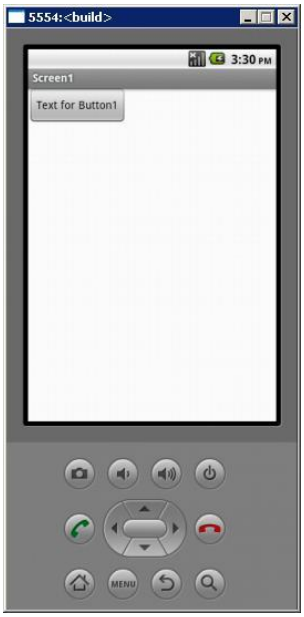


De nuevo aquí bastará con cliquear en la casilla al lado de **get** para que podamos seleccionar la variable **global 'i'**. Por último, colocaremos un **bloque numérico** con el valor '0' dentro del espacio vacío restante.

Todo el conjunto nos quedará de la siguiente forma:



Funcionamiento de la App Propuesta

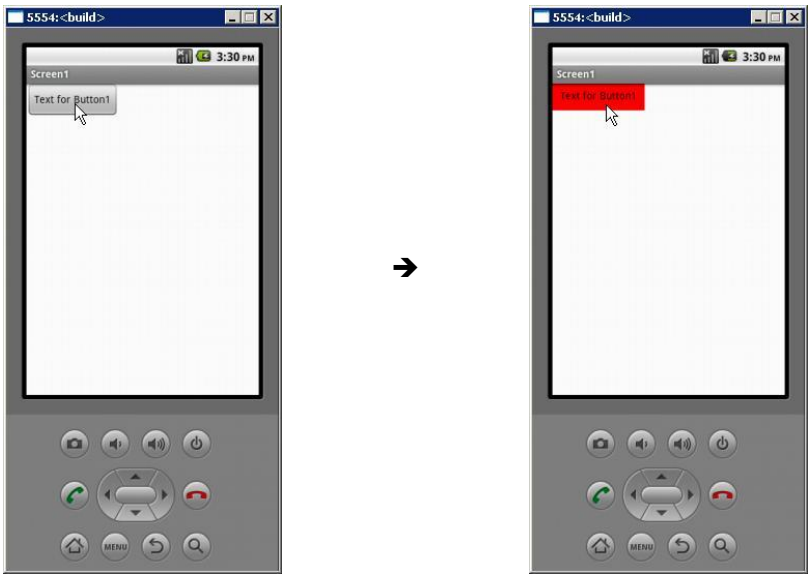


Lo que hará la **App** al iniciar, será crear la variable **global 'i'** y darle el valor **cero**, mostrando la pantalla **Screen1**, en la cual aparecerá el botón **Button1** en color gris (el color por **default**). Mientras que no toquemos este botón, no sucederá nada.

Al tocar dicho **botón**, se "llamará" al bloque **when_Button1.Click_do**, que ejecutará la sentencia **if**, la cual encontrará que el valor de la **variable 'i'** es **igual** a **cero**, por lo que realizará las tareas dentro del espacio **then**; esto es, cambiar el color de **Button1** a **Rojo** y establecer en **uno** el valor de la **variable 'i'**.

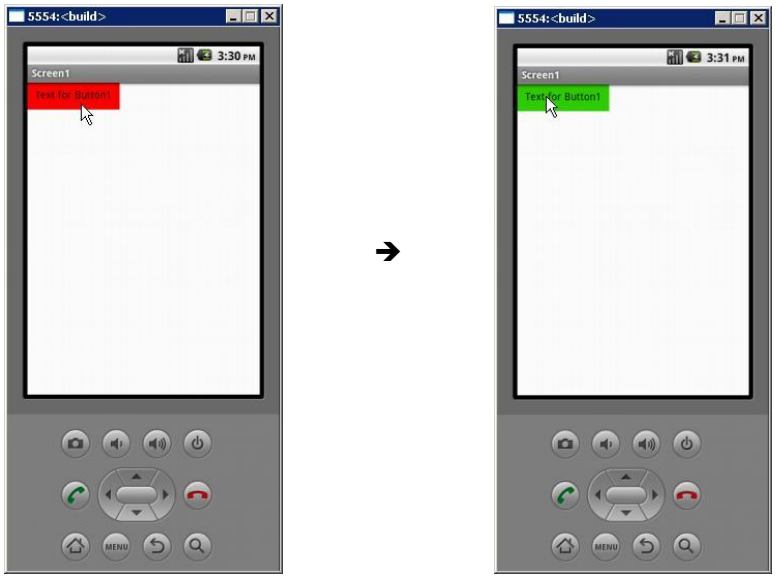
De este modo, las sentencias dentro del espacio **else** serán ignoradas, ya que la condición esperada por el bloque **if** resulta cumplirse, por lo que dicho bloque termina su ejecución en este punto.

Adicionalmente, al no tener otros bloques que ejecutar debajo del **if**, también el bloque **when_Button1.Click_do** terminará su ejecución en este punto, y el programa queda a la espera de un nuevo **evento**.



Nótese aquí, que el conjunto **initialize_global_i_to=0** se ejecuta **sólo** al iniciar la aplicación, por lo que luego de esta primera ejecución del bloque **when_Button1.Click_do**, la variable **'i'** tendrá un valor igual a **uno**.

Si ahora volvemos a pulsar sobre **Button1**, el bloque **when_Button1.Click_do** será llamado nuevamente, pero esta vez al ejecutarse la sentencia **if**, esta encontrará que el valor de la **variable 'i'** es **igual** a **uno**, por lo que realizará las tareas dentro de su espacio **else**; esto es, cambiar el color de **Button1** a **verde** y establecer el valor de la **variable 'i'** como **cero**.



Ya que la **variable 'i'** valdrá nuevamente **cero**, la próxima vez que pulsemos el botón y se ejecute el bloque **when_Button1.Click_do**, el bloque **if** dará como resultado la ejecución de las sentencias incluidas en el espacio **then**, el botón se tornará **rojo** y la **variable 'i'** tomará el valor **uno**, y así sucesivamente hasta que salgamos de la **App**.

Las imágenes de las pantallas con las que hemos explicado el funcionamiento de la esta **App**, fueron generadas utilizando el programa **emulador**, el cual se instalará en nuestra **PC** junto con el programa **aiStarter**. En el próximo módulo vamos a aprender cómo utilizar el mismo.

Probando Nuestras Apps

Objetivo

En este módulo estudiaremos las distintas posibilidades con las que contamos dentro de **App Inventor** para probar el funcionamiento de nuestras **Apps**, a fin de realizar lo que en programación se conoce como **depuración**. Aprenderemos también sobre el uso de múltiples pantallas dentro de una misma **App**, así como también conceptos de programación referente a bloques de texto y matemáticos.

Concepto de Depuración

Decimos que estamos **depurado** un programa o aplicación, cuando estamos **realizando** el proceso de **pruebas**, al mismo tiempo que realizamos **cambios sobre el programa** para modificar su comportamiento o **solucionar algún error**.

Dichas pruebas se pueden llevar a cabo dentro de un dispositivo **Android físico**, o bien realizarse dentro de un software que **emula** un dispositivo **Android virtual**. A dicho software se lo conoce como **emulador**.

Modos de Prueba o Depuración

En **App Inventor** contamos con **3 modos** de prueba diferenciados por el tipo de **dispositivo** y de **conexión**. Estos **modos de prueba** son:

- Por **conexión USB** de un dispositivo **Android** a la **PC**.
- Por **conexión Wifi** de un dispositivo **Android** a la misma red de la **PC**.
- Por **emulación** de un dispositivo **Android** dentro de la misma **PC**.

Nótese que todos los casos involucran el uso de una **PC**. Esto es debido a que la **depuración** se realiza como parte del proceso de construcción de la **App**, por lo tanto es necesario disponer de toda la **información** que la constituye hasta el momento.

Dicha información se encuentra dentro de **App Inventor**, y es por esta razón que la **depuración** implica algún tipo de **comunicación** o **enlace** entre el **dispositivo de pruebas** y el **entorno de desarrollo**, por intermedio de la **PC**.

Es por medio de esta comunicación que, todos los cambios que realicemos dentro de **App Inventor** sobre nuestra **App**, se verán **automáticamente reflejados** dentro del dispositivo de pruebas.

Software Necesario

Como mencionamos en el **módulo 1**, para los casos en los que se vaya a utilizar un dispositivo **Android físico**, será necesario tener instalada en el mismo la "**MIT AI2 Companion App**", que se puede instalar desde **Google Play** a través de este link:

https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3

Adicionalmente, cuando deseemos realizar las pruebas por medio de **USB** o del **emulador**, necesitaremos tener instalado en la **PC** el software "**aiStarter**". El archivo de instalación es el "**AppInventor_Setup_Installer_v_2_2**", el cual podemos descargar de la siguiente dirección:

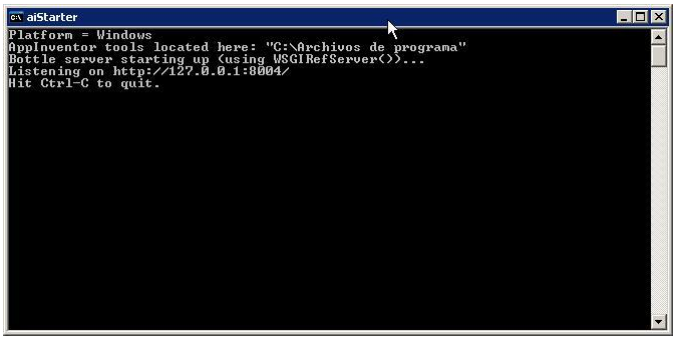
http://appinv.us/aisetup_windows

Nota: Si por alguna razón ya tuvieran instalada una versión anterior, será necesario desinstalar dicha versión y reiniciar la máquina, antes de instalar la nueva versión.

El programa **aiStarter** se usa, tanto para **emular** las **Apps** dentro nuestra **PC**, como para **enlazar** un **dispositivo Android** por conexión **USB**. Esta es la función que cumple dicho programa, es el "puente" que une **App Inventor** con nuestro dispositivo **Android** o **emulador**, permitiendo de este modo llevar a cabo la **depuración**.

Uso del programa aiStarter

Para comenzar, iniciemos el programa **aiStarter**. Al hacerlo se nos abrirá una ventana de línea de comandos (**CMD** o **Terminal**) como la que sigue:



El programa quedará a la espera de las instrucciones que le ha de enviar **App Inventor** desde nuestro **navegador**. A continuación veremos cómo.

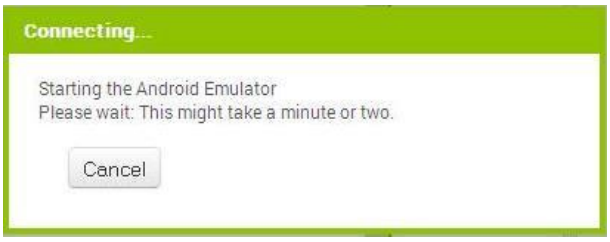
Nota: Una vez que **aiStarter** se encuentra iniciado, **App inventor** es capaz de detectarlo por sí mismo. En caso contrario, nos consultará por la ubicación de este; la ruta por defecto es "**C:\Archivos de Programa\Appinventor\commands-for-Appinventor**".

Uso del Emulador

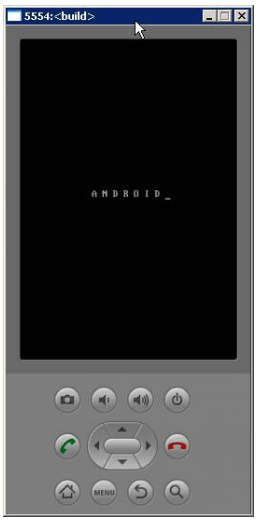
Como mencionamos en el módulo anterior, las imágenes de la **App** que mostramos en el mismo, fueron generadas utilizando el **programa emulador**, el cual se instala en nuestra **PC** junto con el programa **aiStarter**.

Para acceder al **emulador**, ingresaremos a **App Inventor** y, con el programa **aiStarter** en ejecución, abrimos el proyecto de la **App** que deseamos probar. Una vez dentro, cliqueamos sobre el menú **Connect** de la barra gris, y seleccionamos la opción **Emulator**.

Al hacer esto veremos un cartel de que nos informa que el **Emulador de Android** se está iniciando y que dicho proceso **tomará unos minutos**:



Cuando esto suceda, el programa **aiStarter** abrirá dos nuevas ventanas en las que se ejecutará el **emulador**. Si su **PC** cuenta con un **firewall**, es posible que este bloque la conexión, por lo que deberemos dar permiso para que **aiStarter** se conecte normalmente. De las dos ventanas que se abren, la que nos interesa es la siguiente:



En ella se representa una interfaz que imita todas las **características básicas** de un teléfono celular con **sistema operativo Android**:

La **pantalla** de este dispositivo virtual **responde al cursor del mouse**, de la **misma forma** en que lo haría la pantalla de un teléfono **a nuestro dedo**. Esto quiere decir que el hacer **clic** sobre ella es equivalente a realizar un **toque** con un "dedo virtual". A su vez, mantener el **clic** es equivalente a mantener el **contacto** del dedo sobre la pantalla.

Luego de iniciar el **emulador**, se pasará a iniciar automáticamente dentro del mismo la **MIT AI2 Companion App**, al tiempo que dentro de la página de **App Inventor 2** se nos informa de dicho proceso:

Una vez que la página de **App Inventor** constata que todo se está ejecutando correctamente, que el **emulador** se encuentra **enlazado** y que se **transfirió** toda la **información** del proyecto a la **MIT AI2 Companion App**, podremos **ver** y **probar** nuestro desarrollo, sin necesidad de que el mismo esté terminado, lo que nos permite **corregir** lo que sea necesario.

Conexión a través de USB

En el caso de utilizar una **conexión USB** para hacer las pruebas directamente en un teléfono celular u otro dispositivo **Android**, será necesario **instalar los drivers** de dicho **dispositivo** en la **computadora**, para lo cual deberán referirse al manual de usuario de sus dispositivos.

Modo de Depuración USB

Una vez instalados los drivers del **dispositivo**, hay que asegurarse de que el mismo se encuentra en **modo de depuración USB**. Para habilitar dicho modo, siga estos pasos:

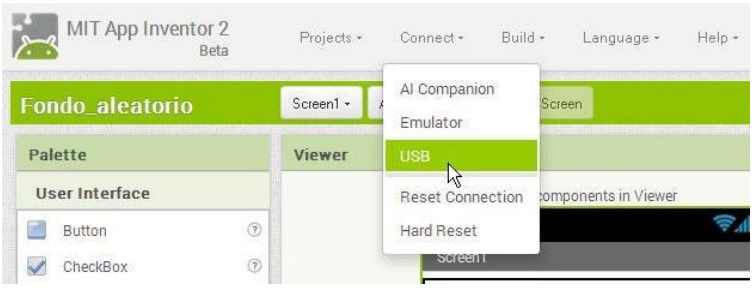
1. Dentro del dispositivo, vamos al **menú principal**.
2. Entramos en **Ajustes**.
3. Dentro de **Ajustes** buscamos el menú **Aplicaciones** y lo seleccionamos.
4. Deslizamos el dedo hacia arriba hasta alcanzar el final de la lista. Allí encontraremos la opción **desarrollo**.
5. Dentro de desarrollo tildamos la casilla al lado de "**Depuración USB**"

La próxima vez que conectemos nuestro dispositivo **Android** a la **PC**, nos aparecerá un mensaje dentro del mismo informándonos que se encuentra conectado y en modo de depuración.

Existe una página para probar la conexión de nuestro dispositivo. La misma es:

http://appinventor.mit.edu/test/

Una vez que nuestro dispositivo se encuentra correctamente conectado, ingresaremos a **App Inventor** y, con el programa **aiStarter** en ejecución, abrimos el proyecto de la **App** que deseamos probar. Una vez dentro, cliqueamos sobre el menú **Connect** de la barra gris, y seleccionamos la opción **USB**:



Al hacer esto veremos un cartel de que nos informa que se está estableciendo la **conexión USB**:

Una vez establecida dicha conexión, la **MIT AI2 Companion App** se iniciará automáticamente dentro del dispositivo, al tiempo que dentro de la página de **App Inventor 2** se nos informa de dicho proceso de manera similar a como sucedió con el **emulador**:

Una vez que la página de **App Inventor** constata que todo se está ejecutando correctamente, que el **dispositivo** se encuentra **enlazado** y que se transfirió toda la información del proyecto a la **MIT AI2 Companion App**, podremos probar la **App** y cambiar lo que sea necesario.

Problemas Comunes

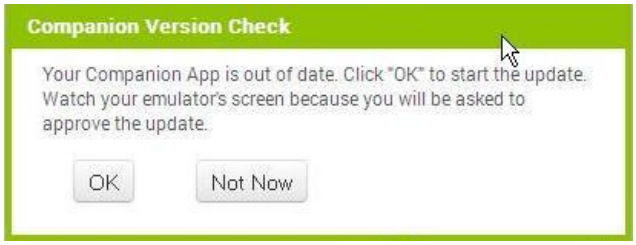
Pueden suceder algunos problemas al momento de realizar las pruebas. Por ejemplo, que dentro de la página de **App Inventor** nos salga un cartel como el que sigue, el cual nos informa que no se ha podido realizar el **enlace**:

Daremos clic en "Keep Trying" para volverlo a intentar. Es posible que acompañado a esto, nos suja el siguiente mensaje, informándonos que la **MIT AI2 Companion App** no responde:

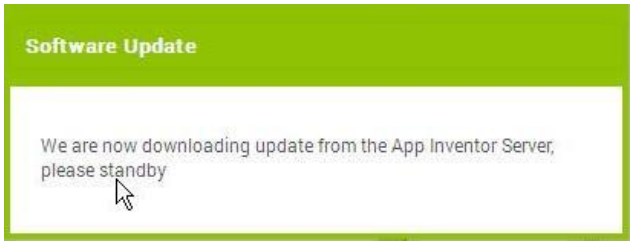
Daremos clic en "Wait" para darle tiempo a la **MIT AI2 Companion App** a que salga de este estado y se comunique correctamente con la página de **App Inventor**.

Actualización de la MIT AI2 Companion App

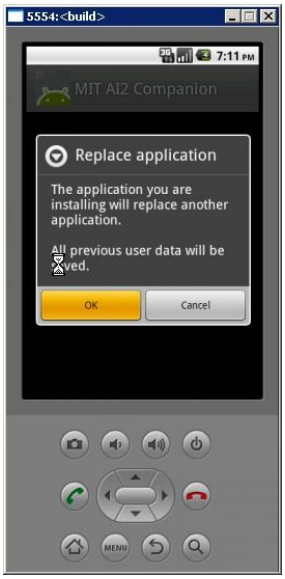
También, puede suceder que **App Inventor** les informe que la **MIT AI2 Companion App** se encuentra desactualizada:



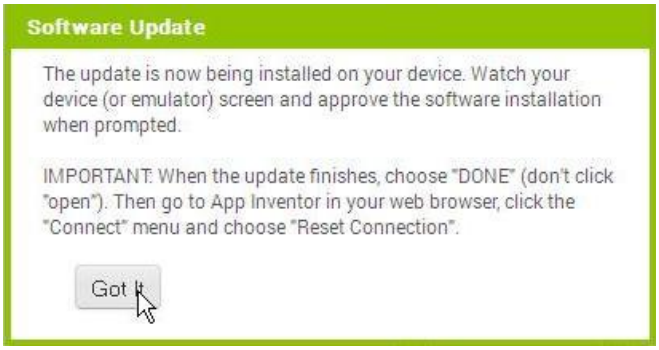
En muchos casos, no hará falta que **actualicemos** dicha aplicación para que la ejecución del proyecto que deseamos probar **funcione correctamente**, por lo que bastará con hacer clic en "**Not Now**". En caso contrario, haremos clic en "**OK**" para que se inicie la actualización. Nos aparece la el siguiente mensaje informándonos que se está descargando la nueva versión:



Cuando esto suceda, veremos que dentro del **dispositivo Android** o **emulador** (según sea el caso), se nos informará que la **MIT AI2 Companion App** será reemplazada por otra versión. Daremos clic en **OK** y luego en **Install** para proceder:



A su vez, dentro de **App Inventor** se nos informará que la nueva versión de la **MIT AI2 Companion App** está siendo instalada dentro del nuestro **dispositivo** (o **emulador**), por lo que deberemos revisarlo con el fin de permitir dicha instalación, que es lo que hicimos en el paso anterior:



IMPORTANTE: Cuando la actualización se termine, seleccionar la opción "**DONE**" o "**LISTO**" (no cliquear en "**open**" o "**abrir**"). Luego, haga clic en "**Connect**" y elija "**Reset Connection**".

Restablecer la conexión (Reset Connetion)

Dentro de la ventana anterior, daremos clic en "**Got It**" y procedemos a realizar los dos pasos mencionados. Primero, seleccionaremos "**Done**" dentro del **dispositivo** o del **emulador**:

Luego, dentro de **App Inventor**, cliqueamos sobre el menú **Connect** y seleccionamos **Reset Connection**. Esto restablecerá la conexión con el programa **aiStarter**, por lo que en el caso del emulador, este se cerrará:

Para volver establecer la **conexión** o abrir la ventana del **emulador**, seleccionaremos la opción que corresponda dentro del mismo menú.

Una vez que completamos la instalación correctamente y restablecemos la conexión, no debería volver a aparecernos el mensaje de actualización, al menos no hasta que salga una nueva versión de la **MIT AI2 Companion App**.

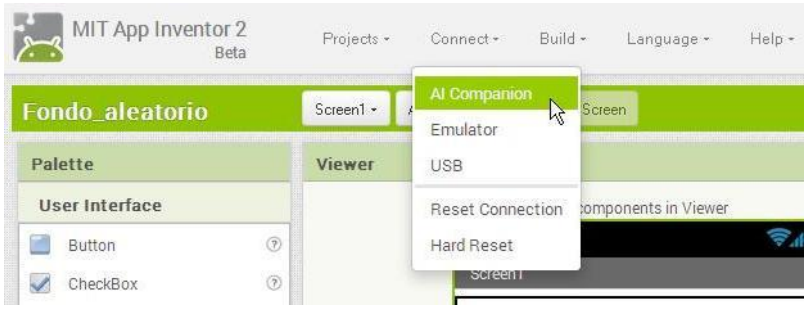
Conexión a través de Wifi

Para realizar las pruebas a través de **Wifi**, sólo nos hará falta contar con la **MIT AI2 Companion App**, lo que significa que no tendremos necesidad de iniciar el programa **aiStarter**.

Tanto la **PC** como el dispositivo **Android** tienen que estar conectados **a la misma red Wifi**, o en el caso de la **PC**, conectada a la misma red por medio de un **cable Ethernet**

al **cable modem** o **router** genera la señal de **Wifi**, el cual vamos a utilizar para conectar nuestro **dispositivo Android**.

Una vez dentro de **App Inventor**, abrimos el proyecto de la **App** que deseamos probar, cliqueamos sobre el menú **Connect**, y seleccionamos la opción **AI companion**:



Nos aparecerá un mensaje que nos dice que debemos iniciar la **MIT AI2 Companion App**, un código **QR** (un cuadrado con varios pixeles) y un **código de 6 caracteres**:

Uso del Código QR para Establecer la Conexión

Los códigos **QR** sirven para **almacenar** y **transferir información** como lo hacen los **códigos de barras**. Para poder "leerlo", necesitamos tener instalada en nuestro dispositivo alguna **aplicación de escaneo** de códigos **QR**, como por ejemplo **Googles** de **Google**, **QR code reader** o cualquier otra de las que podemos encontrar en **Google Play**.

Una vez instalada cualquiera de estas aplicaciones de escaneo, abrimos la **MIT AI2 Companion App** y seleccionamos la opción "**scan QR code**". Se nos abrirá automáticamente dicha **App** de lectura, la cual usaremos para **escanear la imagen del código QR** que nos aparece dentro de **App Inventor**:

Uso del Código de 6 Caracteres para Establecer la Conexión

La otra opción de la cual disponemos, es la de ingresar **manualmente** el código de **6 caracteres** que nos parece a la derecha del código **QR**. Para esto, tocamos sobre la casilla que dice "**Six Digit Code**", y seleccionamos la opción "**connect with code**" debajo de la misma:

Una vez establecida la conexión, podremos realizar la prueba de nuestra App de la **misma forma** que lo haríamos vía **USB**.

A continuación mostraremos unos ejemplos aplicados a un nuevo proyecto, de manera que puedan aprender nuevos conceptos y probar los distintos modos de depuración aquí presentados.

Proyecto 02

Como indica el título, crearemos un nuevo project "Proyecto02". A la primera pantalla (que titulamos como **Operaciones Matemáticas**) colocamos los componentes que se muestran en la imagen que se puede ver más abajo.

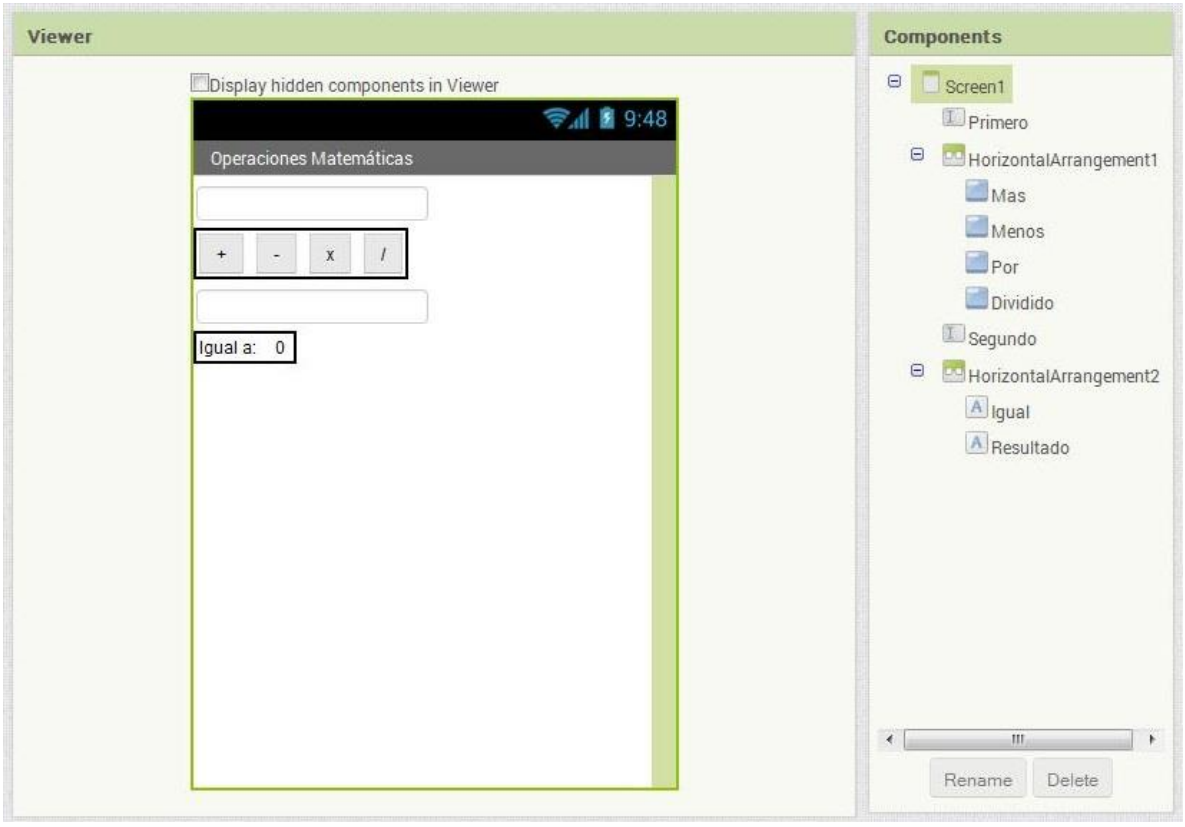
Uso de Etiquetas (Label) y Casillas de Texto (TextBox)

Entre los componentes que van a integrar esta nueva App, podemos encontrar algunos del tipo Label (etiqueta) y otros del tipo TextBox (casilla de texto), los cuales encontraremos dentro del menú User Interface.

El componente Label nos permitirá mostrar un texto en pantalla con sólo ingresarlo dentro del **Designer** en su propiedad Text, o modificarlo dentro del **Editor de Bloques**.

El componente TextBox es similar al Label, con la diferencia que el usuario será capaz de **ingresar el texto** que desee, por medio de un teclado que aparecerá al tocar la pantalla sobre dicho componente.

En la presente pantalla hemos colocado dos TextBoxes renombrándolos como "**primero**" y "**segundo**", en referencia a los números con los que vamos a operar.



Propiedad Hint para TextBoxes (sugerencia)

Entre las propiedades de los **TextBoxes** encontraremos una llamada "**Hint**" (sugerencia). Su función es la de permitir el ingreso de un texto breve, que indique cual es el uso que se le dará a dicho **TextBox** dentro de la **App**.

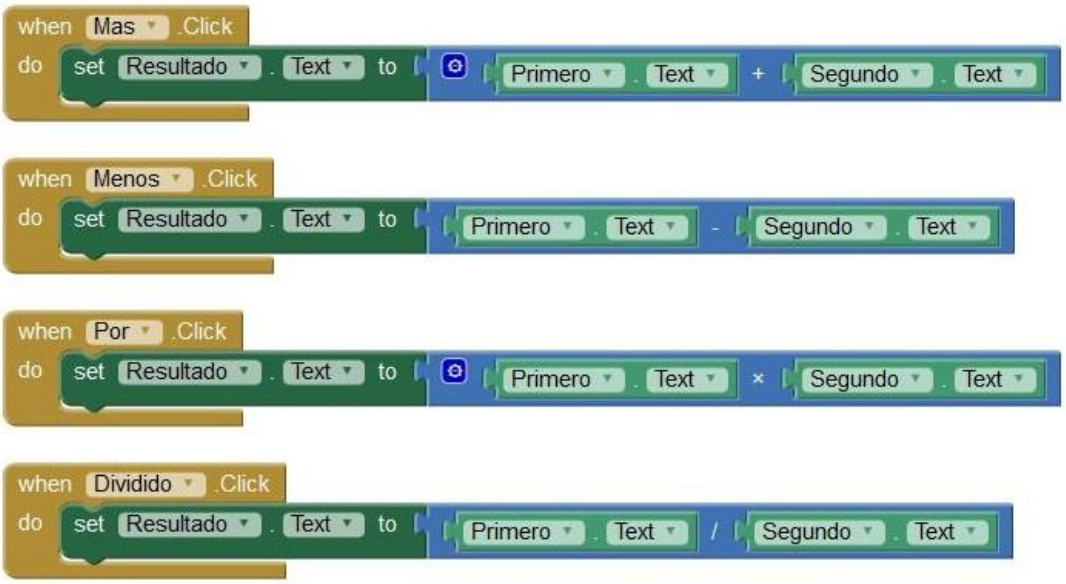
Al iniciarse la **App**, y mientras que no se ingrese un nuevo texto, veremos que dentro de estas "**TextBoxes**" se lee el texto de **hint** en color gris claro, de modo que el usuario tenga una idea de cuál es la función que cumple dicha casilla. Una vez ingresado el nuevo texto, este reemplazará a la sugerencia, pudiéndose leer el mismo en **color negro**.

Dentro de dichas propiedades **hint** colocamos respectivamente los siguientes textos: "Ingrese primer operando" e "Ingrese segundo operando".

Continuando con el diseño, hemos colocado en medio de estas casillas de texto una serie de **botones,** destinados a realizar la operación indicada en cada caso, alineados dentro de un **HorizontalArragements**. Más abajo y dentro de otro **HorizontalArragements**, colocamos dos **Labels** donde se mostrará el resultado de la operación elegida.

Uso de las Operaciones Matemáticas Básicas

A continuación se muestra una imagen en la que hemos utilizado las cuatro operaciones matemáticas básicas dentro de sendos bloques **when_"operación".Click**. Dichos bloques matemáticos se encuentran dentro de menú **Math**.

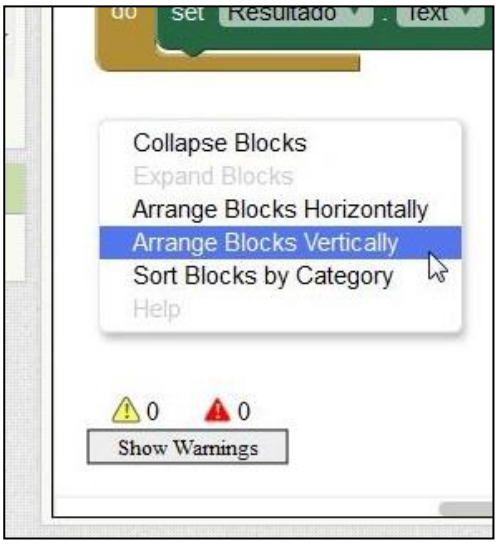


Como se puede apreciar en la imagen, tanto el bloque "**suma**" como el bloque "**producto**" son mutadores, ya que cuentan con el símbolo en su parte superior izquierda (no así los bloques "**resta**" y "**división**").

Por esta razón, cuando necesitemos sumar o multiplicar más de dos números, podremos hacer uso de esta función especial.

Alineación de los Bloques Dentro de la Ventana

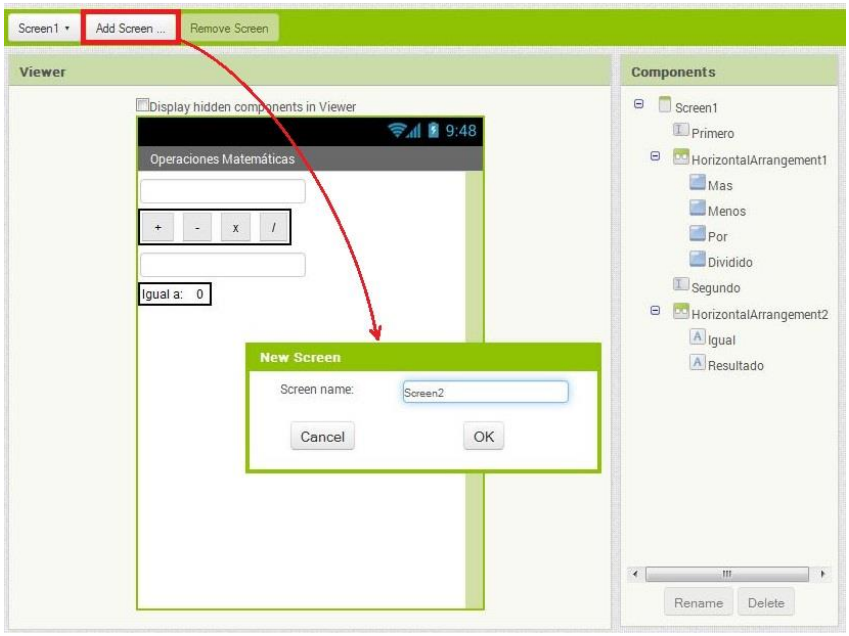
Como podemos ver, los bloques antes presentados se encuentran perfectamente alineados. Esto se debe a que hemos utilizado una función particular de la ventana **Viewer**, a la cual se puede acceder haciendo clic derecho en cualquier parte del espacio en blanco de dicha ventana. Encontraremos también otras opciones útiles dentro del menú que se despliega:



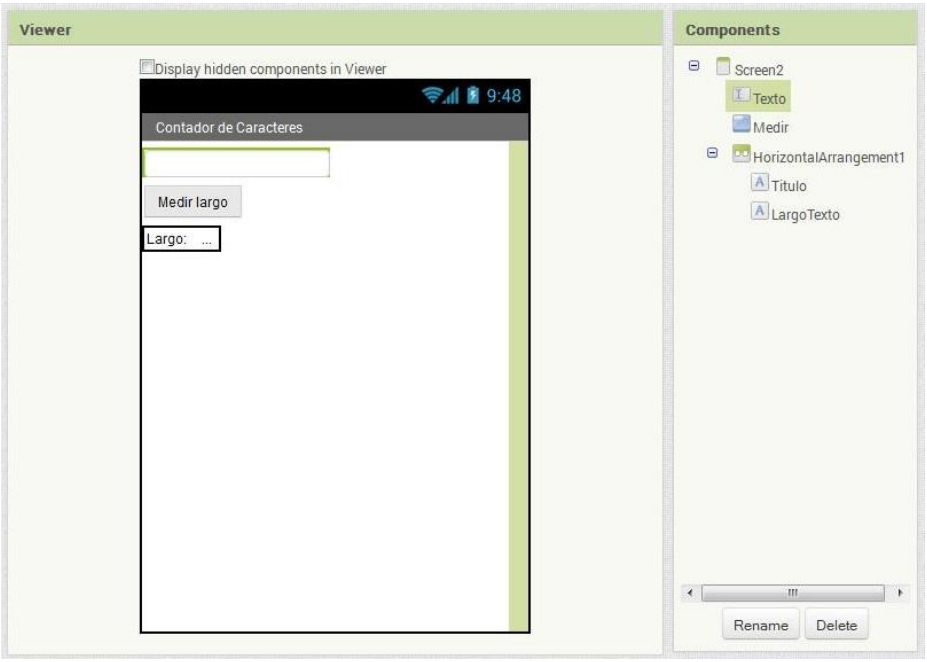
De cualquier, forma la disposición de estos **conjuntos de bloques** dentro de dicha ventana no modifica en nada al comportamiento de la **App**, distinto a lo que sucede con las secuencias de bloques dentro de un **espacio de tareas**, en la cual el orden **sí** puede afectar el resultado ya que, como dijimos anteriormente, la ejecución de los mismos es **secuencial**.

Agregar una nueva ventana a Nuestra App

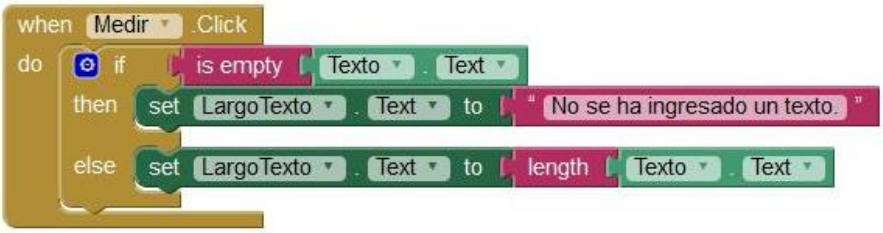
Ahora vamos a colocar una nueva pantalla a nuestra **App**. Lo haremos cliqueando sobre el botón **Add Screen** dentro del entorno de diseño:



La podremos nombrar como queramos. En nuestro caso lo dejaremos como esta y daremos clic en OK. Esta nueva pantalla la vamos a usar para hacer un **contador de caracteres**. Lo que hará será contar la cantidad de caracteres de un texto que se ingresará en un **TextBox**. Aquí el diseño:



Y esta es la programación:



Uso de los Bloques is_empty y length

Hemos utilizado para la programación de esta segunda pantalla los bloques **is_empty** y **length**.

El bloque **is_empty** nos devuelve un valor igual a **true** (verdadero) si es que el texto que recibe está (o es igual a) "vacío", mientras que devolverá un valor igual a **false** (falso) en caso contrario.

El bloque **length** que nos devuelve el largo en caracteres de un texto recibido. Si dicho texto es igual a "vacío", el valor devuelto será igual a "cero".

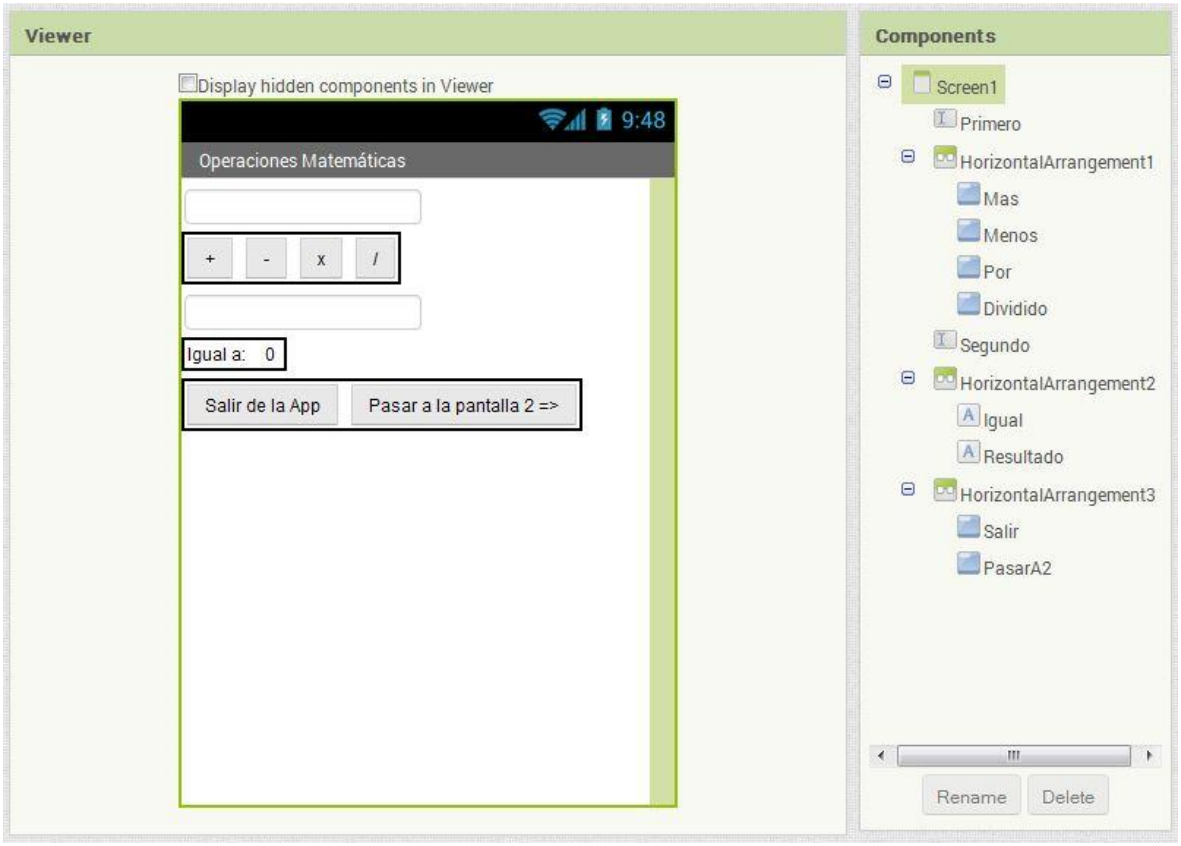
Cambio de Pantalla y Cierre de la App

Esta aquí está todo muy bien, pero si dejamos la **App** de esta forma, no podremos acceder a esta segunda pantalla, a menos que agreguemos algunos elementos destinados para tal fin. Colocaremos además otros elementos que permitirán el cierre de la **App**.

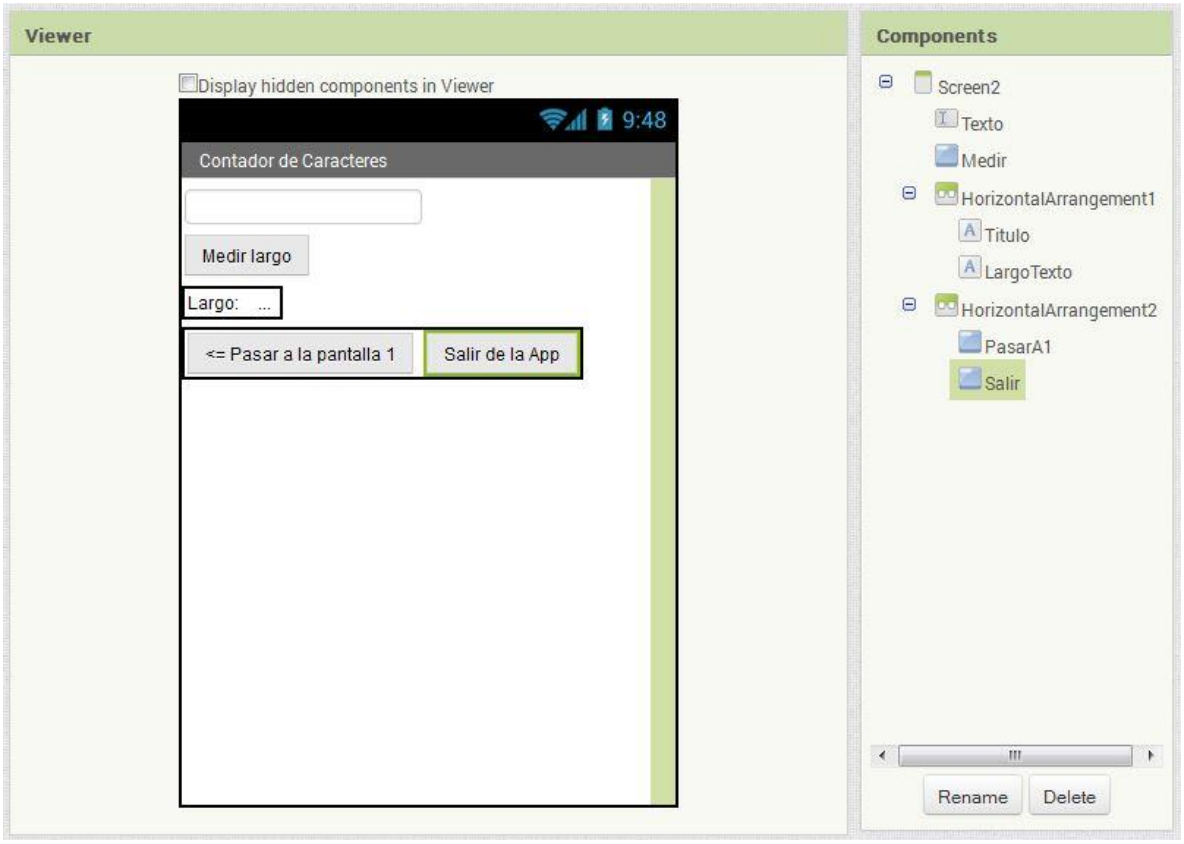
Los bloques que vamos a necesitar son el bloque **open_another_screen** (abrir otra pantalla) y el bloque **close_application** (cerrar aplicación), ambos los cuales se encuentran dentro del menú **Control**.

Para hacer uso de los bloques mencionados, vamos a necesitar algún **evento** que los dispare. Por esta razón, vamos a colocar un par de botones más, tanto a la primera pantalla como a la segunda. La intención es usarlos para pasar de una a otra pantalla o salir de la aplicación, según sea el caso.

En la siguiente imagen podemos ver el cambio propuesto para la **Screen1**:

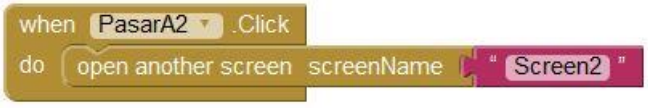


Y a continuación las modificaciones para el **Screen2**:

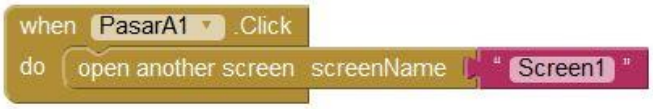


Nótese aquí que el nombre de los botones de dos pantalla distintas puede ser idéntico, debido a que, como dijimos anteriormente, la programación para cada pantalla se realiza de forma separada.

Aquí están los conjuntos de bloques que hemos agregado a la programación de la **Screen1**:



Y aquí los que hemos agregado a la programación de la **Screen2**:



Uso de los Bloques While y For

Objetivo

En este módulo aprenderemos conceptos de programación tales como: **Listas**, uso de los bloques de control **while** y **for**, uso de **variables globales** y **locales**, bloque matemático **random**, bloque de texto **join**, etc. Aprenderemos también de qué manera se **empaquetan** las **App** para poder instalarlas en nuestros dispositivos **Android**.

Generador de Números Aleatorios

Vamos a crear una pequeña **App** cuyo propósito será generar una lista de **números aleatorios**. Con esto en mente y ya dentro del **entorno de diseño**, vamos a cambiar el título de la pantalla de **Screen1** por el de "Números Aleatorios".

Dentro de la ventana **Viewer**, colocaremos un componente **Button1** al que renombraremos como **Generador**, para luego cambiar su texto a "Generar Números".

Debajo de dicho botón, colocaremos un componente **Label**, a cuya propiedad de texto definiremos como '**0**'. Esto lo haremos a los efectos de distinguir entre el **estado inicial** de dicha **etiqueta**, y el listado de números a mostrar. Dichos números los definiremos entre **uno** y **diez**, dentro de **editor de bloques**. Renombraremos además dicha etiqueta como "Números":

Nótese aquí, que si bien, no es posible utilizar caracteres con **tilde** como parte del nombre de un **componente** (o, de por ejemplo, una variable dentro del **editor de bloques**), si es posible hacerlo cuando se trata del **título** de una pantalla o el **texto** de un **botón**.

Programación

Ya dentro del entorno de programación por bloques, crearemos dos **variables**, una a la cual llamaremos "**Numero**" e inicializaremos con un valor igual a '**0**', y la otra a la cual vamos a dar el nombre de "**Lista**". Esta última variable será justamente eso, una **lista en la cual iremos colocando los números** generados aleatoriamente.

Listas en App Inventor

Las variables del tipo **list** (lista) son un caso especial de **variable**, parecida a un **vector**, en la cual cada uno de los elementos de dicha lista tiene un índice (**index**) o **número asociado**, el cual indica su **posición** dentro de la misma.

Este **índice** no forma parte de la lista como carácter **visible**, sino que es **interno al programa**, por lo que no debemos preocuparnos por confundir a dicho **index** con los números que vamos a **guardar como información** dentro de la misma.

Para definir a una variable como del tipo **list**, necesitamos terminar la inicialización de la misma colocando el bloque **create_empty_list** (crear una lista vacía), el cual encontraremos dentro del submenú **Lists** (listas), al principio del mismo:

Continuando con la programación, colocaremos luego un bloque de control por evento **when_Generador.Click_do**, el cual encontraremos como parte del grupo de bloques definidos para dicho componente del tipo **Button**, dentro del menú **Generador**, que se encuentra debajo del menú **Screen1** de la columna **Blocks.**

Uso del Bloque While (mientras)

Dentro del espacio de tareas del bloque **when_Generador.Click** colocaremos otro bloque de control, en este caso uno del tipo **while_test…do** (**mientras** se cumpla la **condición**… **hacer**…) el cual se encuentra dentro del submenú **Control**.

La función de dicho bloque "**while**" será la de realizar una serie de **tareas** de forma repetitiva, **hasta el momento** en el que una **condición dada** deje de cumplirse.

Dicha función, así como también la que cumple el bloque **for**, resulta de gran utilidad en **programación**, por lo que podremos encontrar ejemplos de dichas funciones en **otros lenguajes de programación**, donde se los conoce como **bucles** o **ciclos**.

Desigualdad como Condición de Control

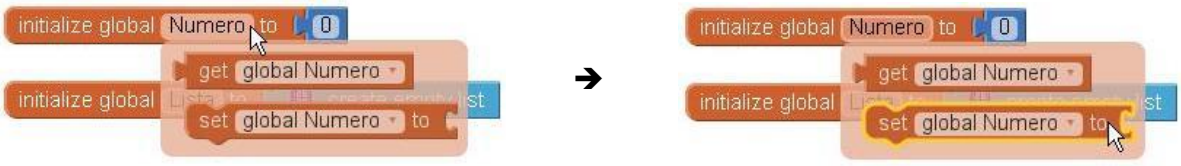
La condición que vamos establecer para nuestro bloque **while**, será la comparación numérica entre el valor de la **variable Numero** y un valor igual a '**cero**', pero en este caso, haremos uso del bloque '**desigual**', el cual obtendremos de modificar el bloque '**igual**' que se encuentra dentro del submenú **Math**.

Para esto, cliqueamos sobre el signo '=' de dicho bloque, lo que nos desplegará un menú en el cual seleccionamos el signo **desigual**, como podemos ver en la siguiente imagen:

Acceso rápido a Getters y Setters para variables

Ahora, vamos a colocar dentro del **while** los bloques que necesitamos, tanto para generar los números aleatorios como para ir guardándolos dentro de la **Lista**. Para esto, colocaremos primeramente un bloque **set_global_Numero_to**.

A dicho bloque podremos acceder rápidamente si posicionamos el cursor del mouse (sin cliquear) dentro del cuadro que contiene el nombre de la **variable** en el bloque **initialize_global_Numero_to**:



Se nos desplegará un menú en el cual seleccionaremos el bloque **set_global_Numero**, al que luego le asignaremos un bloque del tipo **random**.

Uso del Bloque Random Integer

Como dijimos, vamos a asignar a **set_global_Numero_to** un bloque del tipo **random**, en nuestro caso el bloque **random_integer_from_1_to_100** (entero aleatorio de 1 a 100),

el cual encontraremos dentro del menú **Math**. Este será el encargado de **generar** los números aleatorios para la lista.

Como deseamos que el rango de números generados vaya del **1** al **10**, modificaremos el valor del bloque numérico que viene preasignado de '**100**' a '**10**'. El de valor '**1**' lo dejamos tal como está.

Haremos esta pequeña modificación para acotar el rango de valores a generar, aunque podríamos dejarlo como está sin mayores problemas, o incluso elegir un rango de valores más amplio.

Colocaremos este conjunto de bloques dentro del área de tareas del bloque **while**, de manera que nos quede de la siguiente forma:

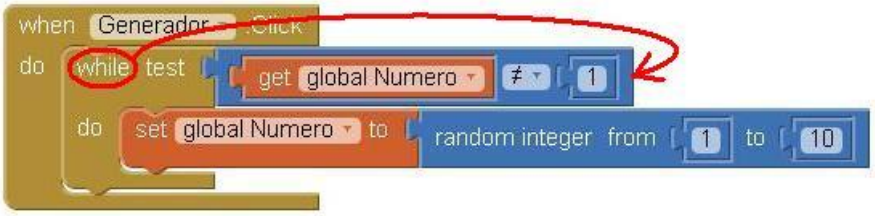
Funcionamiento del Bloque While Paso por Paso

A continuación, vamos a explicar paso a paso el comportamiento del bloque **while** dentro del bloque **when_Generador.Click** para que podamos entender lo hecho hasta el momento:

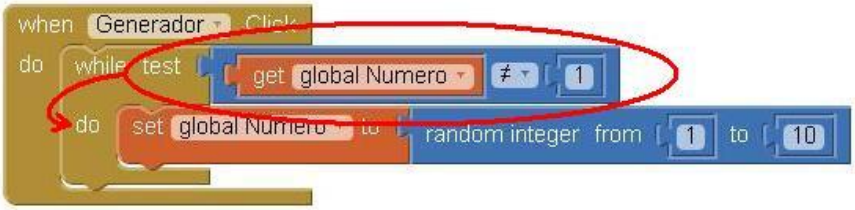
1. El programa espera al momento en que se presione el botón **Generador**.
2. Una vez que dicho botón es presionado, se procede a ejecutar lo que esté dentro del espacio de tareas (**do**) del bloque **when_Generador.Click**, en este caso, el **while**.

3. El bloque **while** comprobará (**test**) la condición asignada, o sea, "mirará" si es que el valor de "**Numero**" resulta o no distinto de '**1**'. De cumplirse dicha condi-

ción, se procederá a ejecutar lo que sea que se encuentre dentro del espacio de tareas (**do**) de dicho **while**.



4. Ya que el valor de la **variable Numero** es inicialmente '**0**', se cumple la **desigualdad** entre esta y el valor '**1**', por lo cual, todo lo que se encuentra dentro del **while** será ejecutado.

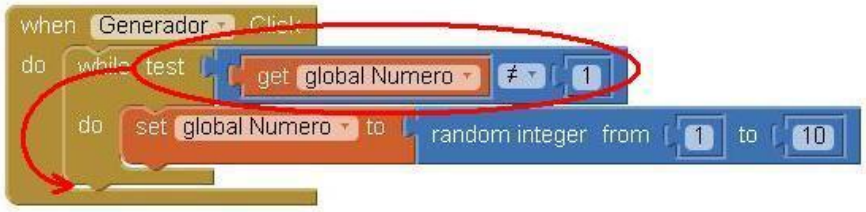


5. Al ejecutarse, el bloque **random_integer_from_1_to_10** generará al azar un valor cualquiera entre '**uno**' y '**diez**', y se lo asignará a la **variable Numero**.

6. Al terminar la ejecución de los bloques que se encuentran dentro de su **área de tareas**, nuestro bloque **while** volverá a probar la condición de control que le fuera asignada, o sea, la **desigualdad** entre "**Numero**" y '**1**'.

7. De cumplirse dicha condición, se ejecutará nuevamente la asignación del bloque **random_integer_from_1_to_10** a la **variable Numero**, para luego volver a probar la condición de control establecida para el **while**.

8. Esto seguirá sucediendo así, hasta el momento en que el valor que se le asigne a **Numero** sea igual a '1', momento en el cual cesará la ejecución del **while**, y se procederá a ejecutar lo que se encuentre debajo de él.



9. A su vez, al no existir ninguna otra secuencia de tareas debajo de dicho **while**, la ejecución del bloque **when_Generador.Click** también habrá culminado, por lo que se "saldrá" de este a la espera de que el usuario vuelva a presionar el botón **Generar**.

Importante: Nótese que si la condición establecida para el **while** resultase **siempre verdadera**, este nunca cesaría su ejecución, lo que probablemente **provoque fallos** en la **App** y/o **resulte indeseable**.

En otros entornos de programación esto puede resultar útil (se llega incluso a establecer dicha condición de control directamente como **true**), pero debido a que la programación en **App Inventor** está orientada a eventos, no resulta conveniente hacer esto.

Uso de las Listas

A partir de lo explicado hasta aquí, los números generados aleatoriamente se sobrescribirán a la **variable Número** en cada ejecución del **while**, por lo que todo esto no nos servirá de mucho si es que no podemos saber cuáles fueron dichos números.

Debido a esto, necesitamos una forma de almacenar dichos números para luego mostrarlos. Para esto vamos a utilizar la **variable Lista** que creáramos previamente.

A ella vamos a asignar **uno a uno** los **números** a medida que se vayan generando. Dicha asignación se lleva a cabo con el uso del bloque **add_items_to_list**, el cual encontramos como parte del conjunto de bloques del tipo **Lists**, dentro de la columna **Blocks**.

Este último requiere que le informemos dos **parámetros**: la lista a utilizar y el **elemento** a agregar.

Para definir la lista a utilizar, necesitaremos utilizar el bloque get_global_Lista, el cual podremos obtener de manera similar a la ya explicada para el bloque set_global_Numero_to. Podemos ver esto en la imagen a continuación:

El elemento a agregar puede ser cualquier valor **numérico** o **lógico**, **palabra**, **carácter** o **cadena de caracteres** que necesitemos guardar, por lo que podremos asignar directamente como ítem el valor de nuestra variable Número.

Lo que sucederá al mostrar nuestra lista en pantalla será que los números aparecerán separados tan sólo por un espacio. Lo que nos interesaría lograr es que los mismos aparezcan separados a su vez por una coma.

Uso del Bloque Join

Para lograr esto, y sabiendo que los elementos que podemos guardar en la lista incluyen a cualquiera de los caracteres del teclado, lo que haremos será una **unión** entre el **número generado** y el **carácter ','**.

Usaremos para esto el bloque **join**, que sirve para unir dos o más cadenas de textos dadas (ya que es del tipo **mutador** y se le pueden agregar más entradas). Una vez ejecutado, devuelve una cadena de caracteres formada por las cadenas a él asignadas.

Podremos encontrar dicho bloque dentro del menú **Text**, dentro de la columna **Blocks**. Lo hemos utilizado aquí de la manera antes mencionada:

Nota: Si bien el bloque **join** está pensado para unir dos o más textos, no existe problema alguno en asignarle una variable numérica, ya que **App Inventor** realizará automáticamente la conversión necesaria.

Continuando con la programación de nuestra **App**, agregaremos este último conjunto de bloques dentro de nuestro **while**, a continuación de lo que colocáramos anteriormente, de manera que el mismo nos quedará de la siguiente forma:

Mostrando la Lista en Pantalla

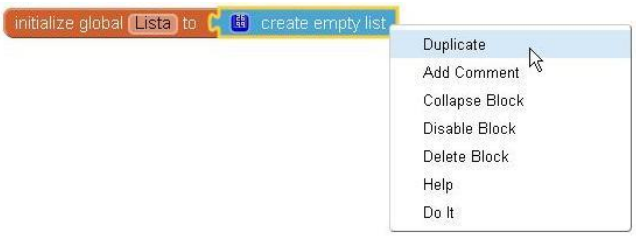
Vamos a agregar ahora los bloques que necesitamos para mostrar nuestra **Lista** en pantalla. Esto lo haremos a continuación del **while**, de manera que la misma ya se encuentre cargada con todos los números que se generaron.

Lo que haremos será asignarle dicha lista a la propiedad **Text** de la **Label** "**Números**", utilizando el bloque **set_Numeros.Text**:

Vaciar una Lista

Luego de esto, necesitaremos "vaciar" nuestra **Lista** para poder reutilizarla la próxima vez que se presione el botón **Generar**, de modo que podamos obtener un nuevo conjunto de números en pantalla. Lograremos esto asignándole una nueva copia del bloque **create_empty_list** a la **variable Lista**.

Podemos duplicar dicho bloque haciendo clic derecho sobre el anteriormente utilizando y seleccionando la opción **duplicate**:



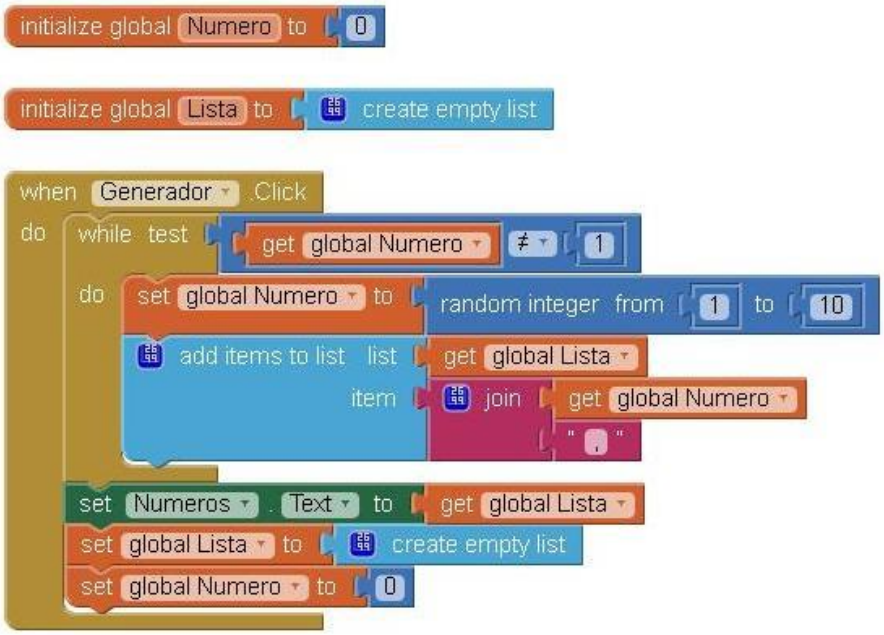
Luego, procederemos a asignarle este duplicado a la **variable Lista** por medio del bloque **set_global_Lista_to**:

Por último, necesitamos restablecer la **variable Numero** a un valor distinto de **uno**, de manera que la próxima vez que el bloque **when_Generador.Click** sea ejecutado, y nuestro **while** evalúe su condición de prueba, este se ejecute al menos una vez.

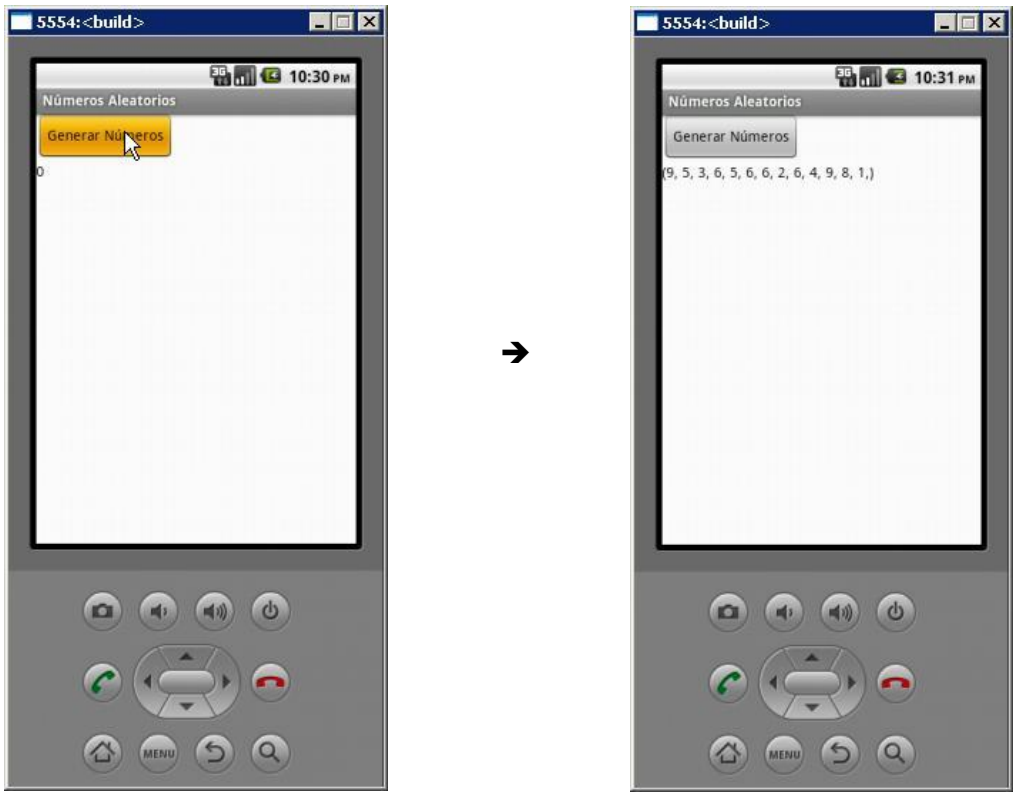
Esto se debe a que, como habrán notado, al salir de dicho **while**, el valor de "**Numero**" queda en '**1**', lo que impediría que se vuelva a ejecutar en cualquier otro momento.

Colocaremos todos los bloques que acabamos de mencionar debajo de nuestro **while**, para que se ejecuten a continuación de este, pero antes de que culmine la ejecución de **when_Generador.Click**.

El **algoritmo** logrado es el siguiente:



En la imagen a continuación podemos ver lo que sucede cuando presionamos el botón **Generar Números**, dentro del emulador:



Generador de Números Aleatorios 2

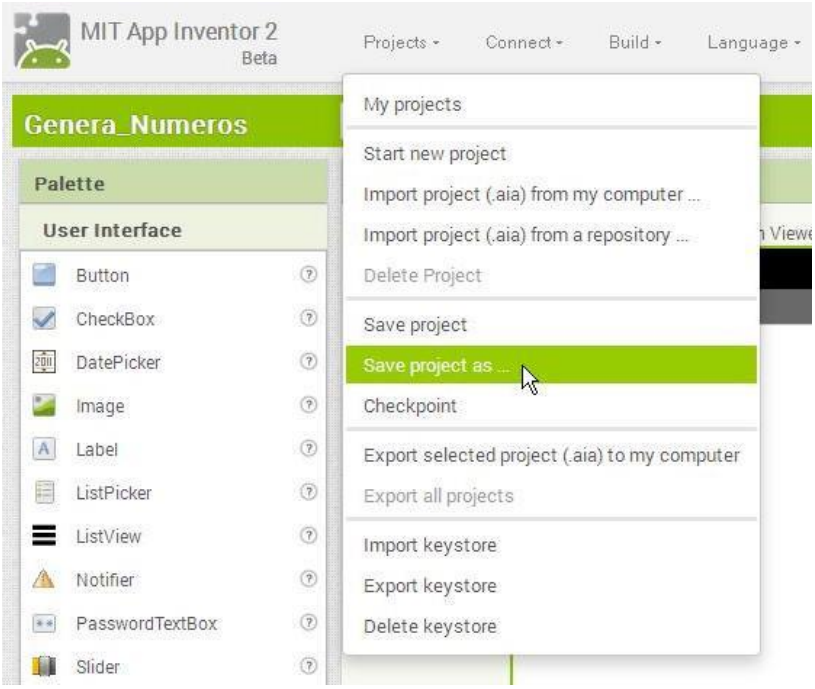
Como habrán notado en el ejemplo anterior, el listado numérico tendrá un largo que variará según el momento en el cual, el bloque **random_integer_from_1_to_10** genera un valor igual a '**1**'.

Para "salvar" esta condición, vamos llevar a cabo algunas modificaciones a lo antes realizado, a fin de mostrar además, que es posible resolver un mismo problema con distintas soluciones.

No vamos a realizar ninguna modificación en lo que respecta al aspecto de la pantalla que creamos anteriormente, sólo vamos a realizar modificaciones en lo que respecta al comportamiento de la **App**.

Crear una Copia de un Proyecto Existente

Para evitar perder lo que ya hemos conseguido hasta el momento, vamos a crear una copia del proyecto anterior. Para esto, ingresamos a dicho proyecto, cliqueamos en el menú **Projects**, y seleccionamos la opción **Save project as…**



Nos aparecerá un cuadro de diálogo, en el cual podremos ingresar el nombre del nuevo proyecto. Una vez ingresado el texto, presionamos en OK y el proyecto se guardará con ese nombre.

Este será un proyecto nuevo, totalmente aparte del original. Todo lo que hecho anteriormente quedará intacto con su anterior nombre.

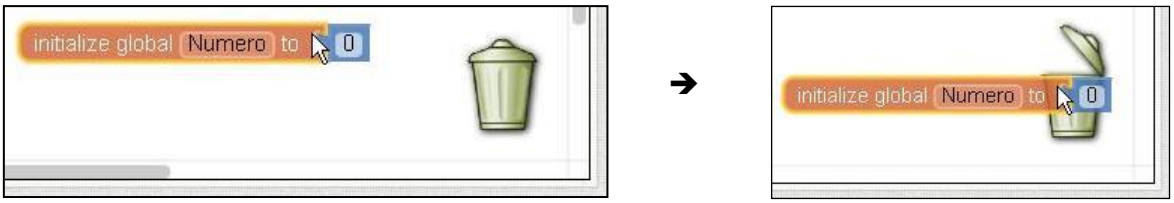
Supongamos que al proyecto anterior lo llamamos "Genera_Numeros", de manera que podríamos llamar al presente "Genera_Numeros_02".

Eliminar Bloques de un Programa

Procederemos ahora a realizar las modificaciones en la programación. Para ello, ingresamos editor de bloques y comenzamos por eliminar la **variable global Número**, la cual reemplazaremos por otra del tipo **local**. Ya explicaremos porque.

Podremos hacer esto seleccionando el conjunto **initialize_global_Numero_to=0**, para luego presionar la tecla "**Supr**" o "**Del**" en el teclado de nuestra **PC**, o bien, utilizar el **icono** en forma de "tacho", el cual se encuentra en la esquina inferior derecha de la ventana **Viewer**, por medio del cursor de nuestro mouse.

Al cliquear sobre dicho conjunto y colocarlo sobre dicho tacho, este se "abrirá" para que al soltarlo aquí (soltando clic), los bloques sean eliminados.



Importante: De momento, **App Inventor 2** **no** cuenta la funcionalidad "deshacer" o "undo", por lo que una vez eliminado un conjunto de bloques, ya sea con el teclado o el tacho, no lo podremos recuperar, a excepción de rehacer dicho conjunto paso a paso.

Variables Locales vs Globales

Como mencionamos más arriba, vamos a reemplazar la **variable global Número** por otra del tipo **Local**. Para crear dicha variable local, necesitamos utilizar un bloque del tipo **initialize_local_name_to**, el cual encontraremos dentro del menú **Variables**.

Colocaremos dicho bloque dentro del evento **when_Generador.Click**, sacando primeramente aparte todo lo que este contenía.

Renombramos dicha variable como "**Numero**" y le asignamos un bloque numérico '**0**':

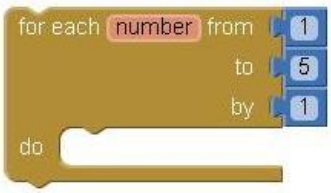
Como mencionamos en módulos anteriores, las **variable locales** son **sólo visibles** dentro del bloque en el que se las creo, y una vez fuera de este, la **variable desaparece** junto con **su valor**. Cuando el bloque vuelve a ser ejecutado, la variable **vuelve a ser inicializada**, y en el presente caso, con un valor igual a cero.

Debido a esto, no será necesario tener en cuanta (ni modificar) el valor de la nueva **variable Numero** antes de salir del bloque **when_Generador.Click**, lo cual sí era necesario cuando se trataba de una **variable global**.

En este caso, resulta algo trivial esta modificación, pero a la hora de crear programas de **mayor complejidad**, nos puede ahorrar el dolor de cabeza que representa el no tener en cuenta lo que sucede con la variable al salir de un bloque, cuando esta es global.

Uso del Bloque For (Para)

La siguiente modificación que vamos a realizar es el reemplazo del bloque de control **while** por uno del tipo **for_each_number_from**. El mismo lo encontramos dentro del menú **Control**.



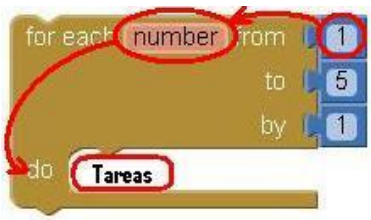
Dentro de dicho bloque encontramos a "**number**", la cual es una **variable local** propia e interna al bloque **for**. A pesar de esto, podremos renombrarla como nos resulte conveniente, así como también podremos modificar los valores asignados a sus encastres **from**, **to** y **by**.

Podríamos traducir "**for each number from… to… by… do…**" como: "**Para cada número, desde… hasta… en pasos de a…, hacer…**". A continuación explicaremos como es

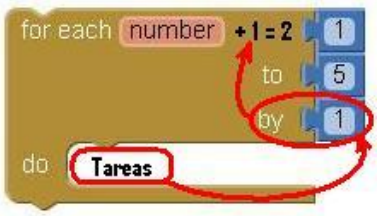
el funcionamiento, paso por paso, para dicho bloque **for**, de manera que todo esto resulte más claro.

Funcionamiento del Bloque For Paso por Paso

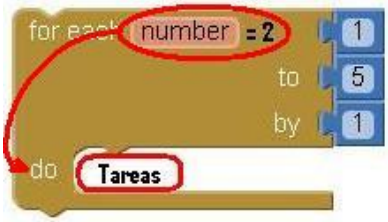
1. Al ejecutarse el bloque **for**, inicializa la **variable local number**, con un valor igual al del encastre **from** (en este caso, '**1**'). Luego, ejecutará los bloques que se encuentran dentro de su área de tareas (**do**):



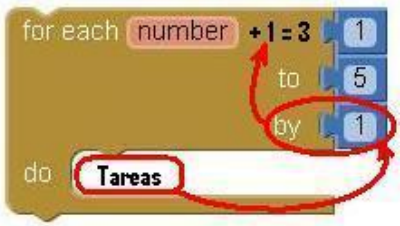
2. Al terminar de ejecutar los bloques que están dentro del área de tareas, se incrementa a **number** en un valor igual al del encastre **by**, que en este caso es igual a '**1**', por lo que **number** pasará a valer '**2**'.



3. Luego, los bloques dentro del área de tareas son ejecutados nuevamente, esta vez **para** (**for**) un valor de **number** igual a '**2**'. A esto es a lo que nos referimos con la expresión "**para cada número**" (o "**for each number**").



4. Nuevamente, al terminar de ejecutar bloques dentro del área de tareas, se incrementa a **number** en un valor igual al del encastre **by**, por lo que **number** pasará a valer '**3**'.



5. Los bloques dentro del área de tareas (do) son ejecutados nuevamente, esta vez **para** un valor de number igual a '3'. Al terminar de ejecutar dichos bloques, se volverá a incrementar number en '1', para luego, volver a ejecutar los bloques que hay dentro el espacio do.

   Esto seguirá así, **hasta** el momento en el cual number alcance un valor igual al que se encuentra en el encastre to, que en este caso es igual a '5':

6. Luego de esta última ejecución (para number igual a '5'), el programa "saldrá" del for, y continuará con la ejecución de los bloques que se encuentren debajo de este.

Bloque For vs While

Como podemos ver, el bloque for nos permite repetir una serie de tareas, de manera similar a como lo haría un while, pero en lugar de evaluar el valor de verdad de una condición dada, evalúa el valor de una variable destinada para esta función.

Dicho esto, procedemos a reemplazar nuestro while (y su condición de prueba), por un bloque for al que le modificaremos algunos parámetros, quedándonos tan sólo con la secuencia de bloque que utilizamos anteriormente dentro de dicho while.

En nuestro caso vamos a renombrar la variable number como 'N', a fin de que no resulte tan similar a la local "Numero". Así mismo, podríamos ponerle cualquier otro nombre, sin que esto afecte a la funcionalidad antes descripta.

Modificaremos además el bloque numérico asignado al encastre to por un valor igual a '20', de manera que ahora el listado de números tendrá ese largo (un largo de '20' números), y terminará con cualquiera de los números del '1' al '10', y no sólo en uno, como sucedía hasta ahora.

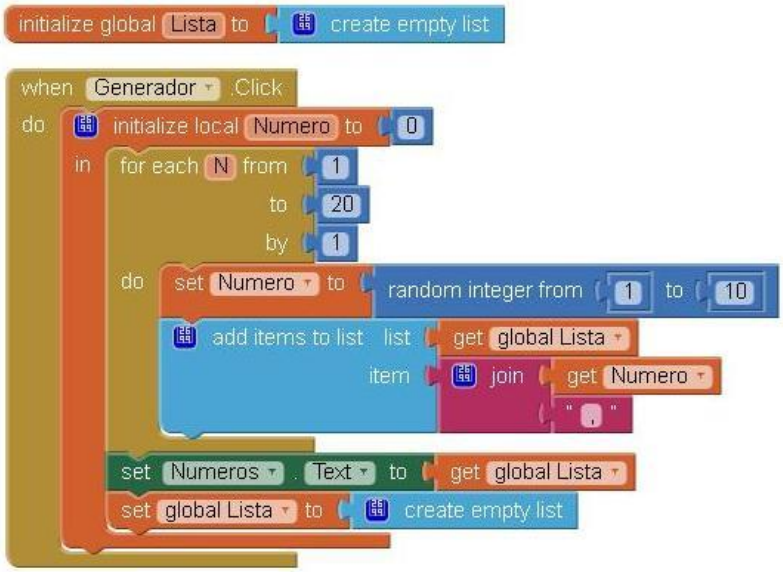
Continuando con la programación, y una vez que hayamos intercambiado el bloque **while** por el bloque **for** (de la manera que se describió más arriba), podremos observar que los bloques **set_Numero_to** y **get_Numero** presentarán sendos **signos de error** (⚠).

Dichos errores se deben a que la **variable global Numero** ya no "existe", ya que la misma fue eliminada. La que sí existe, es la **variable local Número**, que como ya explicamos, **sólo es visible** dentro del bloque en el que se la **inicializó**.

Esta es la forma en la que **App Inventor** nos advierte que dichos bloques están "fuera de lugar". De cualquier forma, dichos errores desaparecerán una vez que coloquemos nuestro **for** dentro del bloque **when_Generador.Click**, que es el que contiene a la **variable local Numero**.

La otra secuencia de bloques que vamos a reutilizar es la que se encontraba debajo del bloque **while**. De dicha secuencia eliminaremos al conjunto **set_global_Numero_to=0**, por lo que la misma nos quedara de la siguiente forma:

El algoritmo completo al que arribaremos al combinar todo esto, será el siguiente:



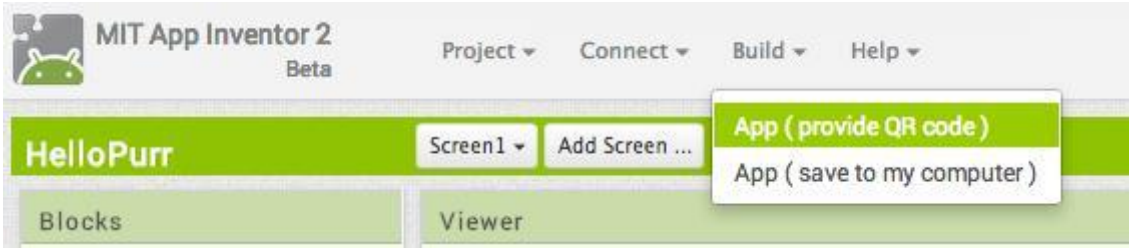
Empaquetado de una App

Si durante la construcción de alguna de las **Apps** que hasta aquí les presentamos, han podido enlazar sus **dispositivos Android** (o **emulador**) a la página de **App Inventor**, habrán notado que las mismas han estado funcionando en tiempo real dentro de dicho dispositivo, o sea, se han ido "armando" dentro del mismo.

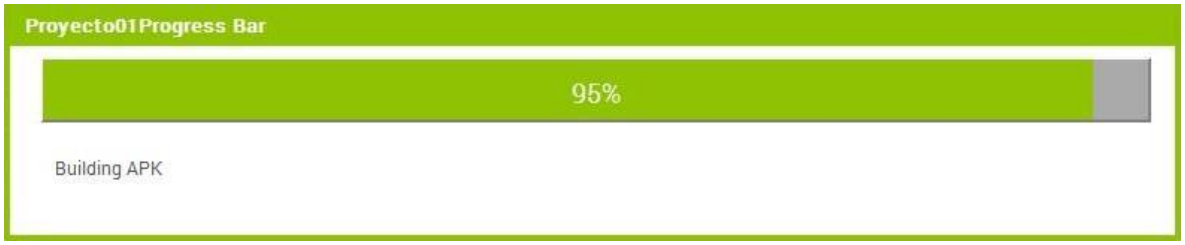
En cuanto cerramos la **App** y desconectamos nuestro dispositivo (o **emulador**), la aplicación desaparecerá del mismo. Podremos realizar dicho enlace todas las veces que queramos, de manera que la **App** volverá a ejecutarse.

Pero lo deseable es tener la **App** en ejecución sin tener que estar conectados a **App Inventor**. Para esto necesitamos **empaquetar** dicha aplicación, de manera que obtengamos un archivo **"paquete"**, cuya extensión será del tipo **.apk**.

Para realizar el empaquetado de una aplicación, haremos clic en el menú **Build**, dentro de la barra gris que se encuentra en la parte superior de la pantalla. Dentro de este menú, tendremos dos opciones posibles; "**App (provide QR code)**" y "**App (save to my computer)**":



Una vez que hayamos cliqueado cualquiera de las dos opciones, nos aparecerá una ventana con una **barra de progreso** que nos indicará que la aplicación está **siendo empaquetada**.



Al terminar dicho empaquetado, se nos presentará una ventana diferente en cada caso, dependiendo de cuál haya sido la opción elegida.

Instalar una App por Medio de Código QR

En el caso de haber optado por cliquear en **App (provide QR code)**, se nos presentará en pantalla un **código QR**, con el cual vamos a poder instalar la **App** directamente dentro de aquellos dispositivos que cuenten cámara de fotos y conexión a internet.

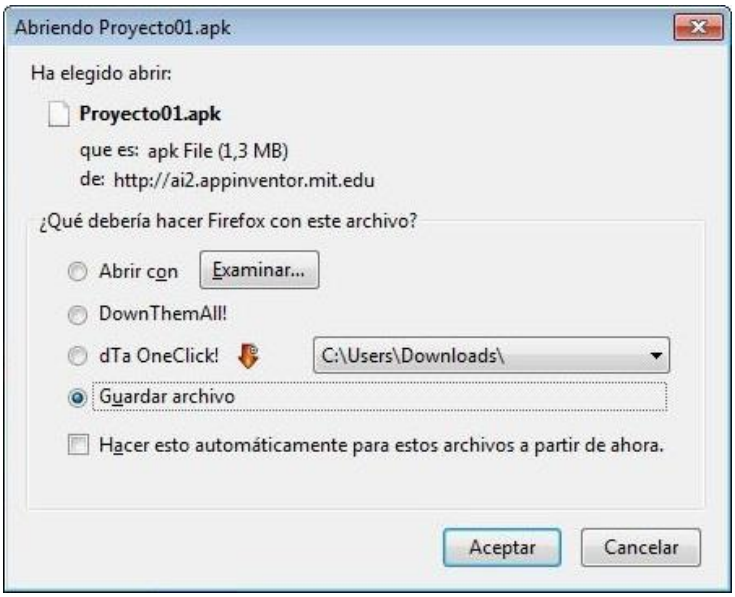
Para poder utilizar esta opción, será necesario contar con alguna aplicación de escaneo de códigos **QR**, tal como Googles o cualquier otra de las que se encuentran disponibles gratuitamente en **Google Play**. Una vez realizada la instalación, daremos clic en OK para continuar.

Nota: el código QR funcionará sólo con aquellos dispositivos que tengan asociada la misma cuenta de Google que la utilizada dentro de App Inventor.

Guardar la App como Archivo en la PC

Por otro lado, si optado por la opción **App (save to my computer)**, vamos a poder descargar la **App** a nuestra **PC** en la forma de un archivo del tipo **.apk**, el cual podremos compartir con otras personas, sin necesidad de que estas hagan uso de **App Inventor**.

Una vez que hayamos cliqueado en dicha opción y de que el proceso de empaquetado haya terminado, nuestro navegador nos preguntará si queremos **descargar un archivo** de nombre **"Project_name".apk**, donde "Project_name" será el nombre que le hayamos dado al proyecto.



Procederemos a descargar dicho archivo seleccionando la opción "Guardar archivo" (o similar), y una vez que lo tengamos en nuestra **PC**, deberemos copiarlo dentro de la **memoria SD** del dispositivo al cual deseamos instalarlo.

Luego, utilizando alguna de muchas las aplicaciones de exploración archivos que podemos encontrar gratuitamente en **Google Play**, intentaremos abrir dicho **"archivo".apk**.

El dispositivo reconocerá automáticamente que se trata de un archivo de instalación **Android**, y nos propondrá instalar la **App** que este contiene. Tocaremos en **aceptar**, y listo! Ya tendremos instalada nuestra **App** para utilizarla en cualquier momento que deseemos!!!

Dibujando con App Inventor

Objetivo

En este módulo introduciremos al componente **Canvas** (lienzo), el cual utilizaremos para dibujar. Aprenderemos como se organizan los componentes en pantalla con el uso de **Arrangements**, así como también el uso de **Argumentos** dentro de ciertos bloques de control de eventos.

Para esto, vamos a construir una **App** que nos permitirá dibujar en la pantalla del teléfono con diferentes colores.

PaintPot: Vamos a Dibujar!

La App que vamos a construir se llamará **PaintPot**, en honor a uno de los primeros programas que se desarrollaron en la década de 1970, el cual demostraba el gran potencial de las computadoras personales. La aplicación cumplirá los siguientes requisitos:

- Se seleccionará con el dedo el color con el que se va a dibujar.
- Al arrastrar el dedo por la pantalla, se dibujará una línea de dicho color.
- Con un solo toque sobre la pantalla se dibujará un punto.
- Al tocar un botón en la parte inferior de la pantalla, está se limpiará.
- Se incluirá una imagen como fondo del dibujo.

Diseño y Configuración

Para comenzar con el armado de la App, vamos a necesitar los siguientes elementos:

- Cuatro botones: **Tres botones** para la selección del color; uno **rojo,** otro **azul,** y uno **verde**; más un **botón adicional** para limpiar o borrar el dibujo.
- Un componente **Canvas**, el cual proveerá la superficie sobre la cual dibujar. Dentro de dicho componente podremos colocar una imagen de fondo.
- Un componente **HorizontalArrangement**, para hacer que los tres botones de color aparezcan alineados.

- Un archivo de imágenes con la cara de un gatito. Hemos colocado el mismo a continuación para que puedan copiarlo a sus **PCs**:

Gatito.jpg

Títulos y Nombres en App Inventor

Empecemos por cambiar el título de la pantalla. Para esto, deberán seleccionar el componente **Screen1** dentro del panel **Components**. Luego, dentro del panel **Properties** encontraremos la propiedad **Title** (título). En ella escribiremos el texto "**PaintPot**".

Hay tres elementos dentro de **App Inventor** a los cuales se puede considerar como **títulos** o **nombres**, y es fácil confundirlos:

1. El nombre que elegimos para el **proyecto** en el cual hemos de trabajar (en este caso, **PaintPot**). Este será también el nombre de la **App**, para el caso en que se la empaquete en un archivo de instalación con extensión "**.apk**".

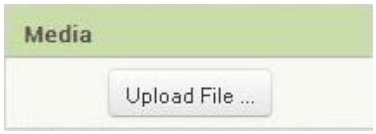
2. El nombre del componente, el cual aparece dentro del panel **Components**, en este caso "**Screen1**", que es el nombre del componente **pantalla**. De momento no es posible cambiar el nombre del primer componente de pantalla, pero sí se pueden crear nuevas pantallas, a las cuales se les podrá cambiar el nombre de componente.

3. La propiedad **Title** (título) de la pantalla, que es lo que aparece en la barra de título del dispositivo **Android** al ejecutar la **App**. **Title** es una propiedad del componente de **pantalla** "**Screen1**". El texto por defecto de la propiedad **Title** será "**Screen1**". Sin embargo, este sí puede ser modificado, como hemos visto más arriba.

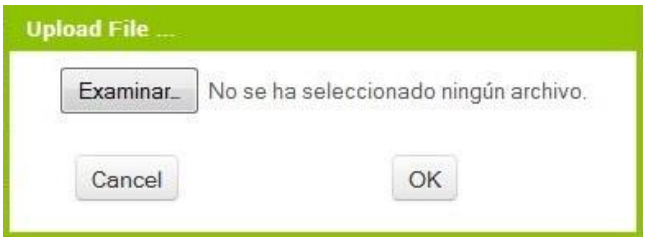
Resumiendo esto último, el **nombre** y el **título** del componente **Screen1** son inicialmente iguales, pero podremos cambiar el título, modificando la propiedad **Title**.

Subiendo Archivos a App Inventor

Como mencionamos en el módulo uno, para poder utilizar los archivos que deseemos dentro de nuestras **Apps**, primero necesitamos subirlos al entorno de trabajo. Para esto, cliquearemos sobre el botón **Upload File…** que se encuentra dentro del cuadro **Media**.



Al hacerlo, nos aparecerá un cuadro de dialogo, el cual nos permitirá buscar dentro de la **PC** los archivos que deseamos integrar a nuestra **App**, por medio del uso del botón **Examinar** o **Browse**.



Al cliquear sobre dicho botón, se nos abrirá una ventana de exploración de archivos, donde podremos seleccionar el archivo de deseado. Una vez seleccionado el archivo, daremos clic en aceptar, para retornar nuevamente al cuadro de diálogo, donde nos aparecerá el nombre del archivo que vamos a subir:

Luego, cliquearemos en **OK** para que el archivo comience a subir a la página de **App Inventor**. Una vez terminada la subida, podremos repetir este proceso para agregar todos los archivos que consideremos necesarios.

Adicionalmente, podremos ver que dentro del cuadro **Media** nos aparece el listado de los archivos cargados. Esto es válido tanto, dentro del entorno de diseño, como dentro del editor de bloques.

Para el caso de la presente App, subiremos el archivo "**Gatito.jpg**" a fin de utilizarlo dentro del Canvas.

Continuando con el diseño, procederemos a colocar el botón para el color **rojo**. Luego, modificaremos su texto para que dentro de él se lea la palabra "Rojo". Modificaremos también la propiedad **BackgroundColor** para que se vea del color correspondiente.

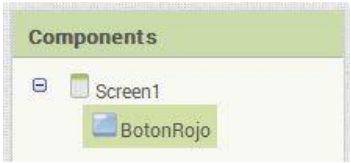
Renombrando los Componentes

Como explicamos más arriba, existen distintos tipos elementos que podemos entender como nombres dentro de App Inventor. Lo que nos interesa modificar ahora es el nombre el componente Button1.

Para esto, necesitamos seleccionar el componente deseado, es este caso Button1, dentro del panel Components. Veremos que más abajo y dentro del mismo panel encontraremos el botón Rename… (Renombrar).

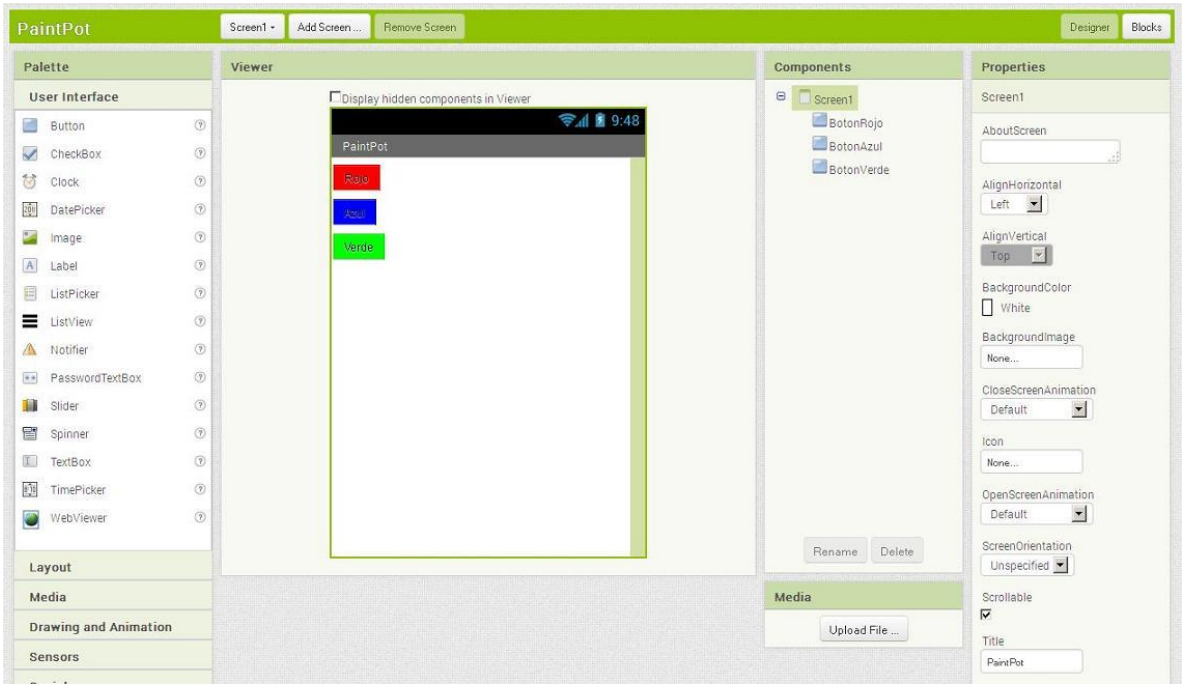
Al cliquear sobre dicho botón, se nos presentará una ventana donde podremos ingresar el nuevo nombre dentro de la casilla "New name".

En nuestro caso, renombraremos dicho botón de Button1 a "**BotonRojo**". Una vez ingresado el nuevo nombre, daremos clic en OK y listo; podremos corroborar el cambio de nombre dentro del panel Components.



Del mismo modo, vamos a colocar otros dos botones; uno para el color **azul** y otro para el **verde**, a los cuales renombraremos como "**BotonAzul**" y "**BotonVerde**" respectivamente. Modificaremos además la apariencia de los mismos de manera similar a como lo hiciéramos con el **BotonRojo**.

El entorno **Designer** nos debería quedar de manera similar a como aparece a continuación, con los nombres los componentes **Buttons** cambiados. El uso de nombres significativos hace que los proyectos sean más legibles y fáciles de entender a la hora de revisarlos y/o programarlos.



Uso de Arrangements

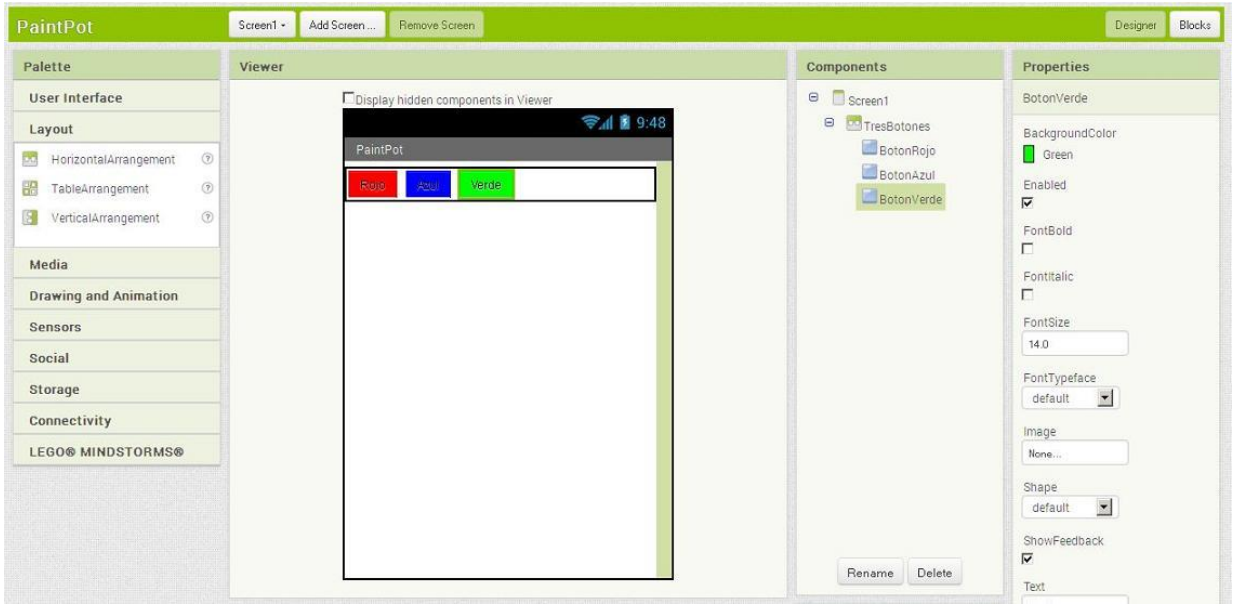
De momento tendremos tres botones, alineados uno encima del otro. El siguiente paso es hacer que estos se alineen horizontalmente. Para ello, utilizaremos el componente **HorizontalArrangement** (alineación horizontal).

El mismos se encuentra dentro de la columna **Palette**, en la categoría **Layout**. Colocaremos uno de estos debajo de los botones de color, y lo renombraremos como "**TresBotones**".
Luego, dentro del panel **Properties**, cambiaremos su propiedad **Width** (ancho) a "**Fill parent…**" (Completar cuadro). Esto hará que el mismo ocupe todo el ancho de la pantalla. Moveremos ahora los tres botones dentro de dicho **HorizontalArrangement**, de manera que estos queden uno al lado del otro.

Nota: Al mover un componente dentro de la pantalla, veremos una línea de color azul que nos indicará donde quedará situada la pieza al momento de soltarla. Utilice dicha guía para lograr el orden deseado.

Si miramos en la lista de componentes del proyecto, veremos que los tres botones aparecen ahora debajo del componente **TresBotones**, con un espacio de sangría. Esto quiere decir que los botones son ahora "subcomponentes" de **TresBotones**.

Observemos además que todos los componentes aparecen a su vez sangrados debajo de **Screen1**, o sea que son subcomponentes de dicha pantalla.



De tener enlazado nuestro dispositivo (o emulador) a **App Inventor**, deberíamos ver los tres botones alineados en la pantalla del mismo, aunque no todo se verá exactamente como en el entorno de diseño. Por ejemplo, vemos un contorno alrededor del **HorizontalArrangement** dentro del entorno de diseño, pero no dentro del dispositivo.

En general, los distintos tipos de "**Arrangements**" se utilizan para crear diseños sencillos, pero se pueden crear diseños más complejos, colocando varios de estos, unos dentro de otros.

Sobre el Componente Canvas

El componente **Canvas** es como un cuadro sensible al tacto que cumple la función de un **lienzo**, el cual nos permite hacer dibujos en dos dimensiones y/o mover imágenes en su interior.

Muchas de sus propiedades pueden ser ajustadas tanto dentro del **Designer** como del **editor de bloques**. Conviene aclarar que la unidad de medida de las propiedades **Width** (Ancho) y **Height** (Alto) es el píxel, por lo que los valores a asignar deben ser positivos.

Podemos especificar un punto cualquiera dentro del **Canvas** como un par ordenado (**X**, **Y**), donde '**X**' será la distancia en píxeles desde el borde izquierdo del **Canvas** al punto, mientras que '**Y**' será la distancia en píxeles desde el borde superior al mismo punto.

En nuestro caso vamos a utilizar el componente **Canvas** para que el usuario dibuje dentro del mismo. Dicho componente se encuentra en la columna **Palette**, dentro del menú "**Drawing and Animation**". Lo tomamos y arrastramos hacía **Screen1**.

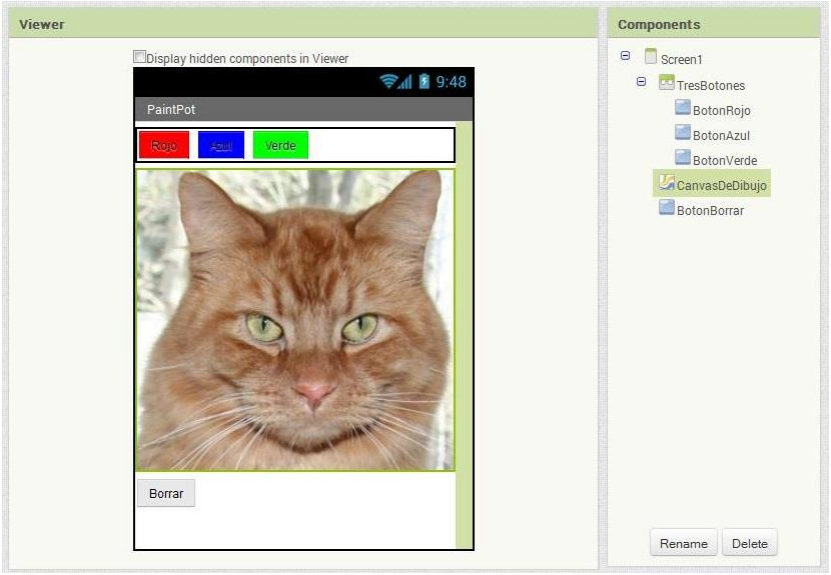
Luego, cambiaremos su nombre a "**CanvasDeDibujo**" y estableceremos su propiedad **Width** como " **Fill parent…**", mientras que su propiedad **Height** la estableceremos en "300 pixels".

Colocando una Imagen dentro del Canvas

Colocaremos ahora el archivo "**Gatito.jpg**" como imagen de fondo. Para esto, nos dirigiremos hacía la propiedad **BackgroundImage** y haremos clic sobre "**None…**". Se nos desplegará una pequeña ventana dentro de la cual podremos seleccionar dicho archivo, siempre y cuando lo hayamos cargado previamente.

Pueden utilizar cualquier imagen que deseen, pero para conseguir los mejores resultados, el tamaño de la imagen debe ser cercano al tamaño del **Canvas** (en píxeles). Además, cuanto **mayor** sea el tamaño de la imagen, **mayor** será el tiempo que tardará en cargarse, y podrían exceder la capacidad de memoria que el teléfono le asigna a las **Apps**.

Continuando con el diseño, colocamos el último botón debajo del **CanvasDeDibujo**. Cambiaremos su nombre a "**BotonBorrar**" y su propiedad **Text** a "**Borrar**". Ya hemos completado los pasos para configurar el aspecto de la **App**. Así es como puede verse dentro del entorno **Designer**.



A continuación, vamos a definir cuál será el comportamiento de la misma.

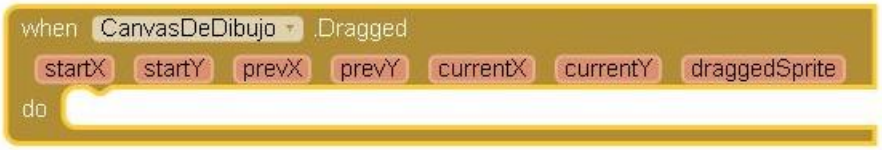
Conceptos para la Programación

Hagamos clic en el botón de **Blocks** para cambiar al **Editor de Bloques**. Primeramente, vamos explicar algunos conceptos necesarios para llevar a cabo la programación. Luego, definiremos las secuencias de bloques necesarias para especificar lo que sucederá cuando alguien toque la pantalla o deslice el dedo sobre ella.

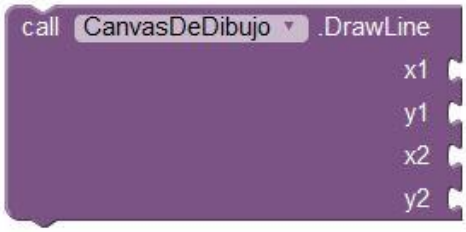
Uso del Componente Canvas

Como ya dijimos, vamos a utilizar nuestro **Canvas** a fin de que el usuario pueda dibujar dentro del él. Por esta razón, vamos a necesitar hacer uso del bloque de control de evento **when_CanvasDeDibujo.Touched**, el cual se dispara al hacer un "toque" sobre algún punto de nuestro **Canvas**.

Otro de los eventos que vamos a utilizar será **when_CanvasDeDibujo.Dragged**, el cual se disparará luego de que el usuario haya deslizado su dedo dentro del **Canvas**.



Dentro de dichos eventos, podremos colocar algunos de los bloques que nuestro **Canvas** tiene disponible, como por ejemplo el bloque **call_CanvasDeDibujo.DrawLine**, el cual nos permitirá dibujar una línea recta desde el punto inicial (**x1**,**y1**) al punto final (**x2**,**y2**).



Otro de los bloques que vamos a utilizar es el **call_CanvasDeDibujo.DrawCircle**, el cual nos permite dibujar un círculo de centro (**x**,**y**) y radio '**r**'.

Por último, tenemos al bloque **call_CanvasDeDibujo.Clear**, el cual borrara por completo todo lo que hayamos dibujado, dejando intacta la imagen de fondo.

Adicionalmente, en el caso de nuestra **App** necesitaremos utilizar el setter de la propiedad **PaintColor**, o sea **set_CanvasDeDibujo.PaintColor_to**. Dicha propiedad es la que define el color con el cual se van a dibujar todos los elementos dentro del **Canvas**.

Concepto de Argumento

Como pudimos ver en módulos anteriores, los bloques del tipo **when_"AlgúnBotón".Click** son bastante fáciles de utilizar, ya que no hace falta que sepamos nada más sobre el "Click", más allá del hecho de que este sucedido.

Por otro lado, existen otro tipo de bloques de **control de eventos**, los cuales nos devolverán información adicional sobre el evento en particular. Dicha información estará almacenada dentro de lo que conocen como "**Argumentos**".

Los **argumentos** funcionan como **variables locales** internas al bloque del evento particular, y su valor inicial dependerá del lugar y el momento en que dicho evento haya sucedido.

Podemos ver ejemplos de este concepto tanto dentro del bloque de control de eventos **when_CanvasDeDibujo.Touched**, como del bloque **when_CanvasDeDibujo.Dragged**. Los mismos aparecen en la parte superior de los bloques mencionados, dentro de una serie de casillas de color **naranjas**, las que contienen el nombre de cada uno de los **argumentos**.

Para operar con dichos **argumentos**, podemos utilizar los getters y setters que aparecen disponibles dentro del menú **Variables**, como lo hemos hecho hasta el momento para cualquier otra variable.

Sin embargo, podremos acceder también a ellos posicionando el cursor del mouse sobre el nombre del **argumento** deseado, y seleccionando el bloque necesario dentro del menú desplegable que se nos presentará.

Concepto de Bloque de Comando

Algunos de los bloques que acabamos de describir, pertenecen a lo que en **App Inventor** se conoce como **bloques de comando**. Dichos bloques se caracterizan por comenzar con la palabra "call" (llamada) y por su color purpura, el cual coincide con el de los **bloques de procedimiento**, ya que su uso es similar.

Los **bloques de comando** especifican una determinada acción a realizar. Estos bloques se colocan dentro de la **sección de tareas** (**do**) de los bloques de **control por eventos**, de manera que cuando se produzca el evento esperado, se "llame" al **bloque de comando** para realizar la acción deseada.

El uso de algunos de ellos es muy simples, como es el caso del bloque **call_CanvasDeDibujo.Clear**, el cual no cuenta con ningún encastre o entrada, mientras que otros requieren que se les informe ciertos parámetros, como sucede con el bloque **call_CanvasDeDibujo.DrawLine** o el bloque **call_CanvasDeDibujo.DrawCircle**.

Otra forma en que se conoce a estos bloques dentro de **App Inventor** es con el nombre de **Method** o **Método**.

Programación de la App

Cambiando el Color de la Pintura

Comenzaremos por establecer las secuencias de bloques que necesitaremos para que el usuario pueda modificar el color con el que se ha de dibujar.

Para ello, nos dirigiremos a la columna **Blocks**, cliquearemos sobre el menú "**BotonRojo**" y seleccionaremos el bloque **when_BotonRojo.Click**. Luego, dentro del menú **Canvas-**

DeDibujo, seleccionamos el bloque **set_CanvasDeDibujo.PaintColor_to** y lo colocamos en la sección de tareas de **when_BotonRojo.Click**.

Cliqueamos luego sobre el submenú **Colors**, seleccionamos el bloque de color **rojo** y lo encastramos al bloque **set_CanvasDeDibujo.PaintColor_to**. Repetiremos estos pasos para hacer lo propio con los botones de color **Azul** y **Verde**.

Vamos ahora a ver los pasos que necesitamos seguir para dibujar dentro de nuestro **Canvas**. La secuencia de bloques deberá ser tal que al tocar dentro del mismo, se dibuje un punto donde se realizó el toque, mientras que si se desliza el dedo lentamente, se dibuje una línea.

Dibujando un Círculo

Continuamos la programación definiendo la secuencia de bloques que necesitaremos para que el usuario pueda dibujar un punto. Para ello, cliquearemos sobre **CanvasDeDibujo** y seleccionaremos el bloque **when.CanvasDeDibujo.Touched**. Tan pronto como lo seleccionamos, podremos observar los nombres de sus tres **argumentos**, los cuales son '**x**', '**y**', y "**touchedSprite**".

Los dos primeros argumentos representarán las coordenadas '**x**' e '**y**' del punto donde ocurrió el contacto, mientras que **touchedSprite** hace referencia a un componente que veremos más adelante.

Lo que necesitamos lograr, es que cuando este evento se produzca, dentro del **Canvas** se dibuje un pequeño círculo, con centro en las coordenadas (**x**, **y**). Por dicha razón, vamos a seleccionar el bloque **call.CanvasDeDibujo.DrawCircle** y lo colocaremos dentro de la sección de tareas del bloque **when.CanvasDeDibujo.Touched**.

Como ya mencionamos, del lado derecho del bloque **call.CanvasDeDibujo.DrawCircle** encontraremos tres encastres, en los cuales debemos especificar los valores para las coordenadas (**x**,**y**) en las cuales se dibujará el círculo, mientras que **r** será el radio del mismo.

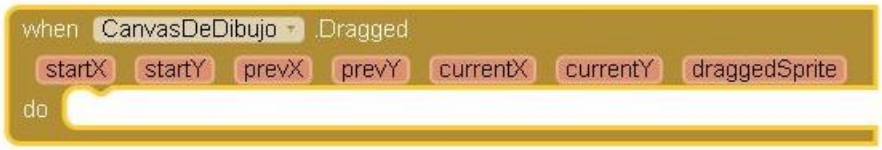
Para los parámetros '**x**' e '**y**' vamos a utilizar los valores de los argumentos suministrados por el bloque **when.CanvasDeDibujo.Touched**. Para esto, colocaremos el cursor del mouse sobre el argumento '**x**', seleccionamos el bloque **get_x** dentro del menú desplegable, y lo colocamos en el encastre '**x**' del bloque **call.CanvasDeDibujo.DrawCircle**. Haremos lo mismo con el argumento '**y**'.

También tendremos que especificar el radio del círculo a dibujar. Cinco (píxeles) es un buen valor para nuestra **App**. Si hacemos clic en un área vacía de la ventana **Viewer**, escribimos el número '**5**' y presionamos **enter**, crearemos un bloque de numérico para este valor. Luego, colocamos dicho el bloque numérico en el encastre **r**.

En el caso de que tengan un dispositivo **Android** enlazado, podrán probar lo realizado hasta el momento. Prueben de tocar alguno de los botones de color y luego en cualquier parte del **Canvas**; deberán ver como se "mancha" cada lugar donde se tocan.

Dibujando una Línea

Continuando con la programación, vamos a agregar los bloques que necesitamos para que al deslizar el dedo, se dibuje una línea. Para cumplir con la tarea, vamos a utilizar un bloque **when_CanvasDeDibujo.Dragged**, el cual encontraremos en el menú **CanvasDeDibujo**. Dicho bloque cuenta con seis **argumentos**.

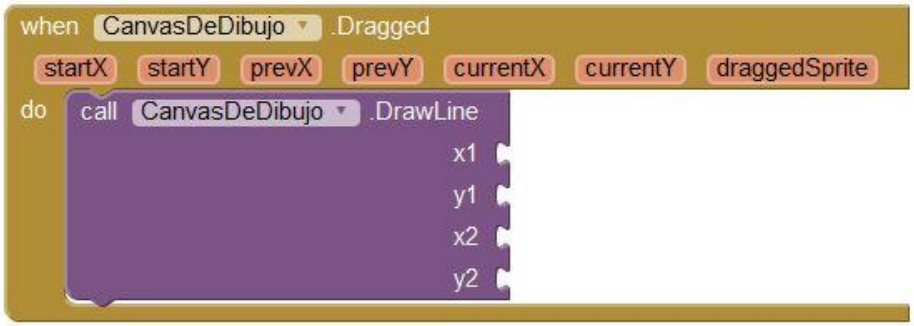


Dichos argumentos representan a tres pares de coordenadas (x, y), las cuales describen, respectivamente:

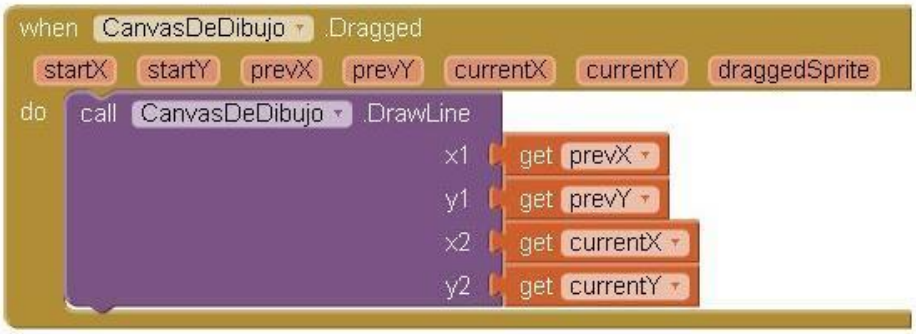
- La posición del dedo al comenzar a deslizarlo (**startX** y **startY**).
- La posición actual del dedo (**currentX** y **currentY**).
- La posición del dedo inmediatamente anterior a la actual (**prevX** y **prevY**).

Vemos también que existe un argumento llamado **draggedSprite**, el cual no vamos a utilizar para el presente ejemplo.

Colocaremos ahora los bloques que necesitamos para dibujar una línea. Dentro del menú **CanvasDeDibujo** encontraremos el bloque **call.CanvasDeDibujo.DrawLine**. Lo seleccionamos y lo colocamos dentro del área de tareas del bloque **when.CanvasDeDibujo.Dragged**.



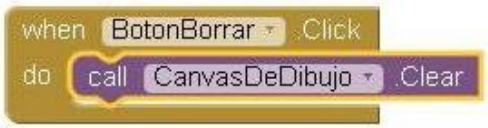
Luego, lo completaremos las asignaciones a cada uno de sus encastres de siguiente forma: **x1=get.prevX**, **y1=get.prevY**, **x2=get.currentX**, e **y2=get.currentY**. El resultado será el siguiente:



Al arrastrar el dedo sobre la pantalla, describiendo por ejemplo una línea curva, lo que sucede en realidad es que se irán dibujando cientos de pequeñas líneas rectas entre cada punto del recorrido: cada vez que se mueve el dedo, aunque sea por muy poco, se extenderá la línea graficada desde la última posición que ocupó el dedo hasta la posición inmediatamente posterior, y así sucesivamente.

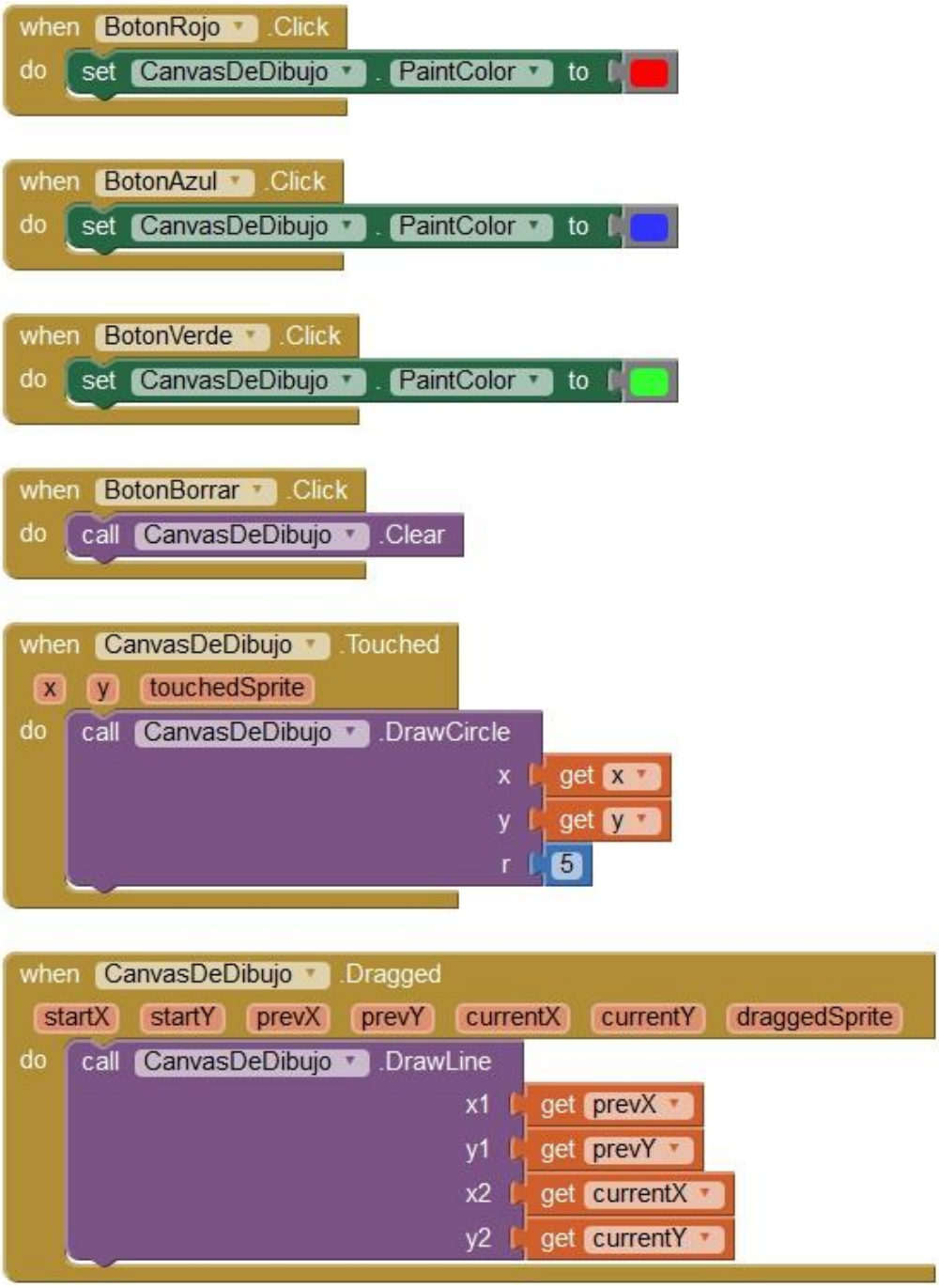
Borrando lo Dibujado

Por último, vamos a configurar el botón **Borrar**. Dentro de la columna **Blocks**, cliquearemos sobre el menú **BotonBorrar**, para luego seleccionar el bloque **when_BotonBorrar.Click**. Luego cliqueamos sobre **CanvasDeDibujo**, seleccionamos el bloque **call_CanvasDeDibujo.Clear** y lo colocamos en la sección de tareas de **when_BotonRojo.Click**.



Recordemos que esta acción sólo borrará aquello que el usuario haya dibujado, dejando intacta la imagen de fondo del **CanvasDeDibujo**.

A continuación podemos ver el algoritmo completo de esta aplicación:


when BotonRojo .Click
do set CanvasDeDibujo . PaintColor to
when BotonAzul .Click
do set CanvasDeDibujo . PaintColor to
when BotonVerde .Click
do set CanvasDeDibujo . PaintColor to
when BotonBorrar .Click
do call CanvasDeDibujo .Clear
when CanvasDeDibujo .Touched
x y touchedSprite
do call CanvasDeDibujo .DrawCircle
x get x
y get y
r 5
when CanvasDeDibujo .Dragged
startX startY prevX prevY currentX currentY draggedSprite
do call CanvasDeDibujo .DrawLine
x1 get prevX
y1 get prevY
x2 get currentX
y2 get currentY

Construcción de Apps Multimedia

Objetivo

En este módulo vamos a aprender sobre el uso de los compontes del tipo **media** (multimedia) y del tipo **storage** (almacenamiento), así como también algunos del tipo **User Interface** que nos serán de particular utilidad para con los dos primeros tipos.

HelloPurr: Acariciar al gatito para que haga miau!

Vamos a construir **App** se llama **HelloPurr**, la cual a pesar de su sencillez y de que la podremos construir en un tiempo muy corto, nos servirá para demostrar algunas de las características más interesantes de **App Inventor**.

La idea de esta **App** es utilizar la imagen de un gatito, la cual al ser tocada por el usuario reproducirá el sonido de un "miau", para simular el maullido de un gato en respuesta a una caricia.

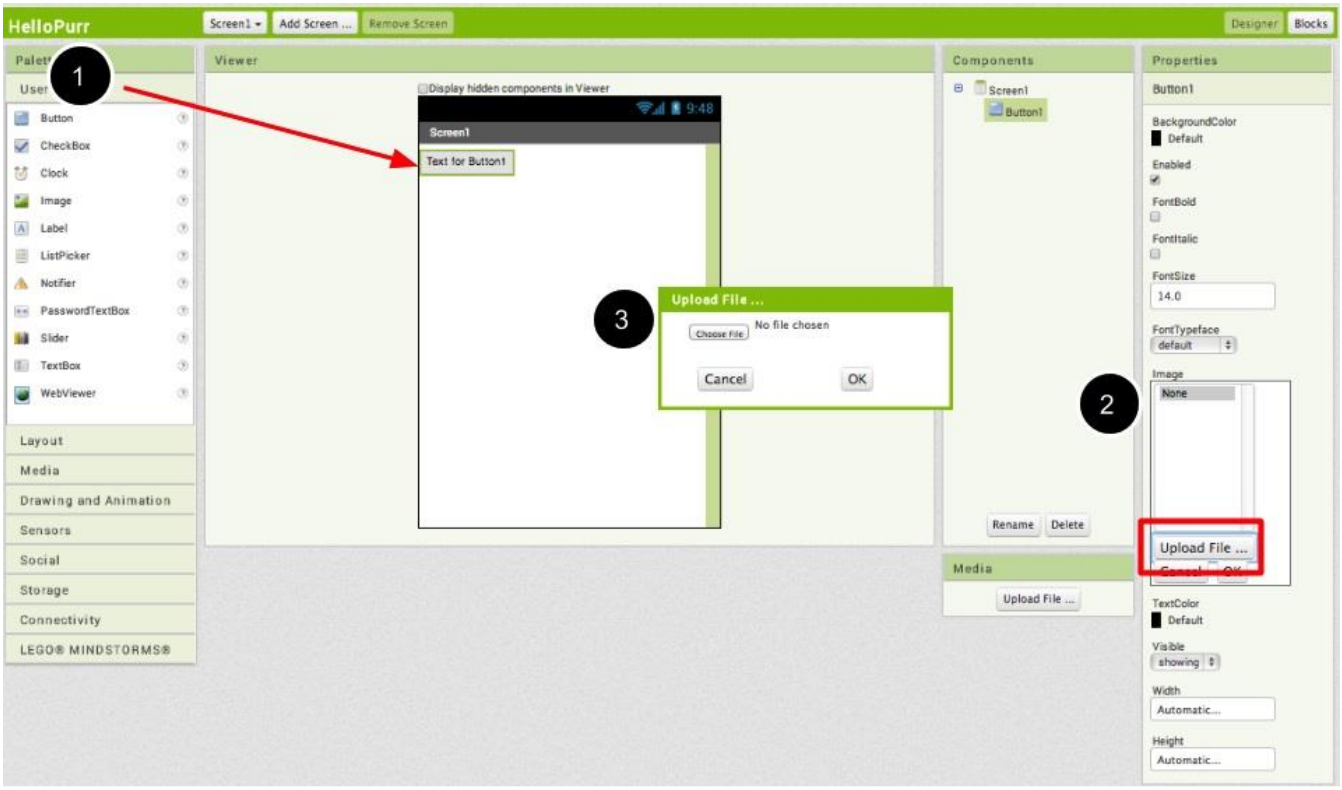
Con este fin, vamos a necesitar hacer uso de dos archivitos: un archivo de imágenes con la foto del gatito, y un archivo de audio con el sonido de un "miau", los cuales hemos colocado a continuación, para que puedan copiarlos a su PC:


Gatito.jpg


Miau!.mp3

Colocando una Imagen Dentro de un Botón

Como primer paso del diseño, colocaremos un componente **Button** dentro del cual vamos a mostrar la imagen del gatito. Utilizaremos dicho **Button** para disparar la reproducción del "Miau". Para esto, luego de colocar dicho componente **Button(1)**, iremos a la columna **Propierties**, y dentro de la propiedad **Image**, haremos clic donde dice "**None…**"**(2)**:



Al hacer esto, se nos desplegará una ventana o menú, dentro de la cual tendremos la posibilidad de seleccionar el archivo de imagen que deseamos para **Button1**, siempre y cuando lo hayamos subido son anterioridad.

De no ser así, veremos que la lista de archivos disponibles estará vacía. En ese caso, podemos cliquear sobre el botón "**Upload File…**"**(2)**, el cual nos aparecerá dentro de dicha ventana o menú.

Una vez cliqueado, nos aparecerá un cuadro de dialogo similar al descripto anteriormente, donde al cliquear sobre **Browse**, podremos para ubicar al archivo "Gatito.jpg" dentro de nuestra **PC (3)**.

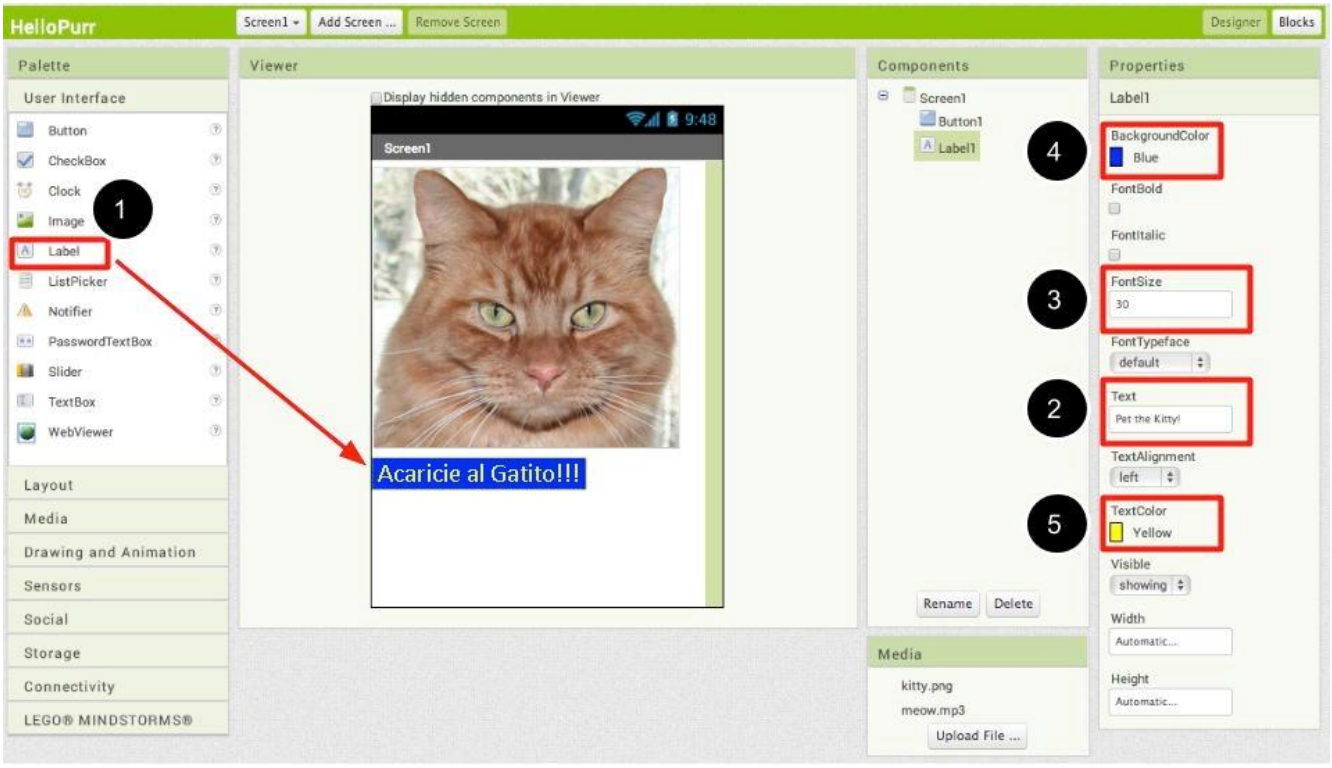
Seleccionamos este archivo, damos clic en **Abrir** y a continuación en "**OK**". Una vez completada la subida, dicho archivo se establecerá automáticamente como imagen de fondo para **Button1**.

Cabe aclarar que si bien en este caso, la subida del archivo se ha realizado dentro de la columna **Properties** y para un componente determinado, este parecerá como parte del listado del cuadro **Media** y estará disponible para usarlo dentro de cualquier otro componente que queramos.

Continuando con el diseño, modificaremos ahora la propiedad **Text** de **Button1** de modo que no aparezca nada escrito sobre la cara del gatito. Bastará con eliminar la leyenda "**Text for Button1**" de dicha propiedad (dejando esta en blanco).

Por otro lado, si sucede que la imagen aparece incompleta, podremos solucionar esto modificando las propiedades **Height** (alto) y **Width** (ancho) del botón. Para ello, cliqueamos sobre "**Automátic…**" y seleccionamos la opción "**Fill parent**" (completar cuadro), dentro de las listas desplegables que nos aparecerá para ambas propiedades.

Colocamos ahora un componente **Label(1)**, situándolo debajo de la imagen del gatito, y cambiamos su texto por "**Acaricie al Gatito!!!"(2)**. Modificaremos también el tamaño de la letra de **Label1**. Para esto, deberemos hacer clic en la propiedad **FontSize**, y cambiar su valor, en este caso, a '**30**' **(3)**.



Por último, cambiaremos el color de fondo de **Label1**, haciendo clic en la propiedad **BackgroundColor(4)**. También se puede cambiar color del texto haciendo clic en **TextColor (5)**. Puede elegir el color que más le guste. En este caso, hemos elegido para el fondo el color **azul** (**Blue**), mientras que para el color del texto elegimos el **amarillo** (**Yellow**).

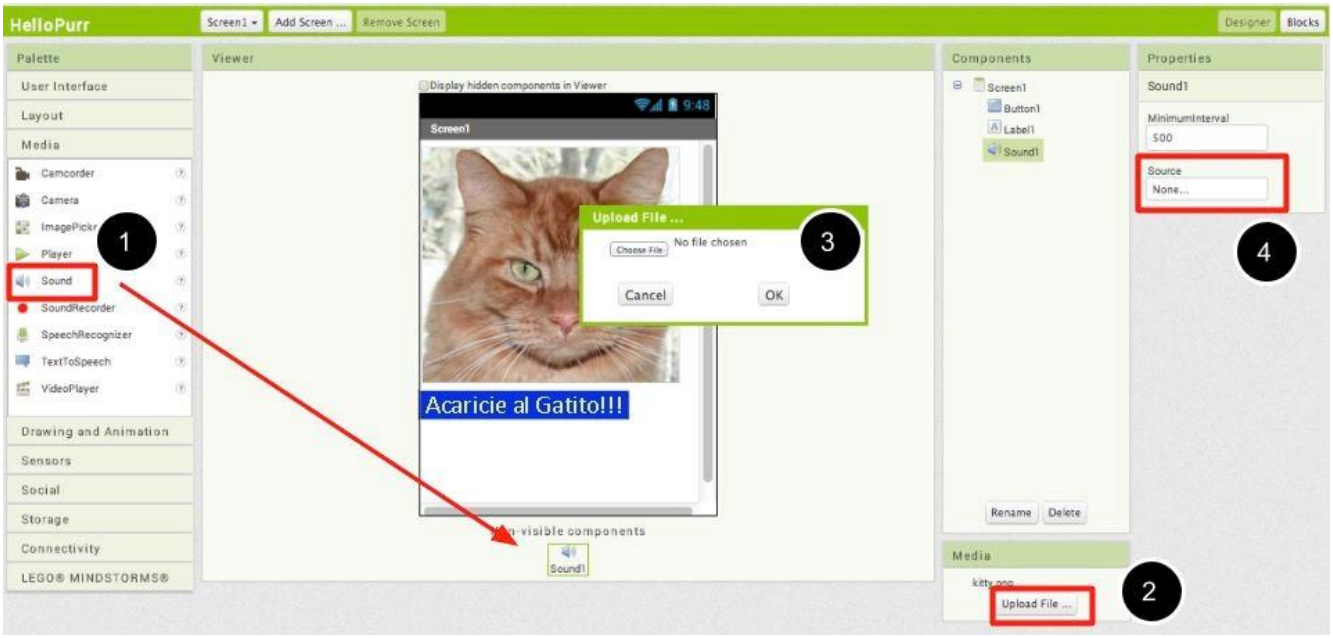
Uso del Componente Sound (Sonido)

Agregaremos ahora un componente **Sound**, el cual se encuentra en el submenú **Media**, dentro de la columna **Palette**. El mismo aparecerá en la parte inferior de dicha ventana, en el área de marcada como **Non-visible components** (componentes no visibles).

Dicho componente sirve para reproducir archivos de sonido. El nombre del archivo de sonido a reproducir puede ser especificado tanto dentro del entorno de diseño como dentro del editor de bloques.

En este caso, haremos clic en el botón **Upload File** del cuadro **Media** **(2),** dentro del **Designer**. Dentro de la ventana que se nos presenta **(3)**, ubicaremos al archivo de sonido que copiamos a la **PC** con anterioridad (**Miau!.mp3**); lo seleccionamos y lo subimos al proyecto dando clic en "**OK**".

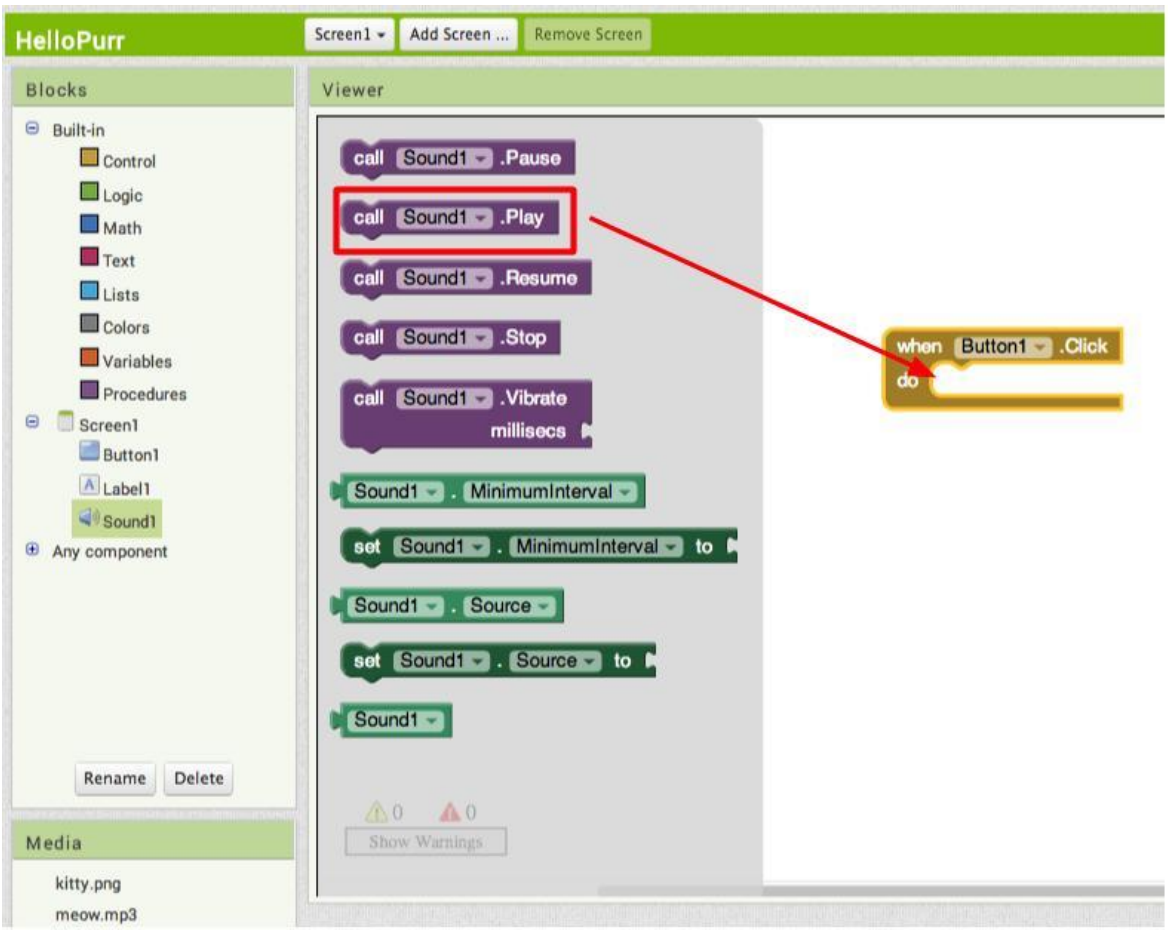
Si seleccionamos ahora al componente **Sound1**, veremos que dentro del panel **Properties**, la propiedad **Source** (fuente) muestra la leyenda **None** (Ninguno). Haremos clic en dicha leyenda para cambiar la fuente de sonido de **Sound1** por **Miau!.mp3 (4)**.



Pasaremos ahora a programar el comportamiento del botón. Como mencionamos más arriba, cuando el usuario toque la imagen del gatito, o sea, presione el botón, deberá producirse la reproducción del archivo **Miau!.mp3**.

Esto se logra mediante el uso del bloque de comando **call_Sound1.Play**. En nuestro caso, colocaremos el bloque **call_Sound1.Play** dentro del evento **when_Button1.Click**.

Encontraremos dicho bloque de comando en el menú de bloques para el componente **Sound1**, dentro de la columna Blocks.



Podemos entender el conjunto de bloques obtenido como: "**Cuando** se haga **clic** sobre **Button1**, se **reproducirá** el sonido de **Sound1**".

Podríamos decir que los **bloques de control de eventos** nos indican la categoría de la acción a realizar (por ejemplo, el presionar un botón), mientras que los **bloques de comando** especifican el tipo de acción y sus detalles (por ejemplo, la reproducción de un sonido específico).

Ya podemos probar la **App**! Al tener nuestro dispositivo **Android** enlazado a la **PC** (o al estar usando el emulador), veremos que dentro de mismo tenemos la pantalla con la

imagen del **gatito** tal y como la armáramos dentro del **entorno de diseño**. Verán que al tocar (o hacer click) sobre el gatito se escuchara un maullido.

Grabadora de Voz

El siguiente proyecto hará uso de los componentes multimedia **Player** y **SoundRecorder**, con el fin de armar una grabadora de voz. El primero de ellos se usa para reproducir sonidos de manera similar al componente **Sound**, mientras que el segundo se utiliza para grabar sonidos por medio del micrófono del dispositivo.

Crearemos un nuevo proyecto dentro del cual vamos a colocar dichos componentes multimedia, como parte de la **Screen1**. Estos, al igual que **Sound**, son componentes del tipo **no visible**, por lo que aparecerán debajo de dicha pantalla.

Sobre el Componente TinyDB (Tiny Data Base)

Como podemos ver en la imagen anterior, entre los componentes no visibles se encuentra además el componente **TinyDB1** (pequeña base de datos), el cual encontraremos como parte de los componentes del tipo **Storage** (Almacenamiento).

Este es un componente que sirve para almacenar los datos de una **App** antes de su cierre. Las **Apps** que creamos dentro **App Inventor** se reinicializan cada vez que se las ejecuta, o sea, comienzan siempre desde el punto inicial o, por así decir, desde cero.

Esto quiere decir que los nuevos valores asignados a las variables dentro de la **App**, se perderán cuando el usuario la cierre, y no podrán ser recuperados la próxima vez que la aplicación se ejecute.

Por el contrario, la información contenida por un **TinyDB** es almacenada dentro de un espacio especial de la **App** conocido como "data storage". Esto quiere decir que los datos almacenados dentro de un **TinyDB** estarán disponibles cada vez que la aplicación sea ejecutada.

Los datos guardados dentro de los **TinyDB** se almacenan en la forma de "cadenas de caracteres" (o sea, como textos), y se los identifica de manera separada por medio de

tags (obleas). Debido a esto, al momento de almacenar un dato, será necesario especificar el **tag** bajo el cual debe ser guardado.

En nuestro caso, utilizaremos para la presente **App** el **TinyDB1**, de modo que podamos guardar y recuperar las rutas a los archivos de voz que hayamos grabado en anteriores ejecuciones de la aplicación.

Continuando con el diseño, colocaremos un **VerticalArrangement** con el fin de que contenga a todos los componentes visibles. Si bien, de por sí todos los componentes que agreguemos se organizarán dentro de la pantalla de esta forma, veremos que resulta conveniente hacer esto con el fin de ajustar el ancho de los mismos, acotándolos a dicho **Arrangement** por medio de la opción **Fill parent**.

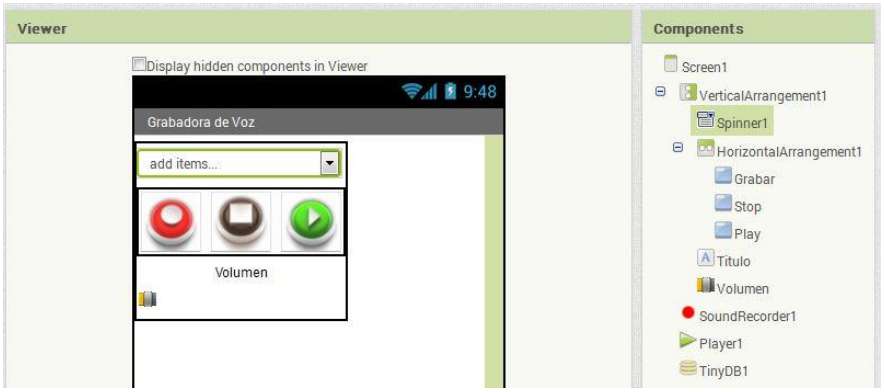
Sobre el Componente Spinner

Colocamos ahora un **Spinner**, el cual encontraremos como parte de los componentes del tipo **User Interface**. Dicho componente sirve para mostrar en pantalla (y dentro de una ventana), una lista de **elementos** determinada de la cual el usuario podrá escoger la opción deseada.

Dichos elementos pueden ser establecidos dentro del **Designer** o del **Blocks Editor**, tanto mediante el uso de la propiedad **ElementsFromString** como de la propiedad **Elements**.

En nuestro caso, utilizaremos el **Spinner1** para listar una serie de números que representarán a los archivos de audio que vayamos grabando. En principio no tendremos ninguna lista disponible. La intención es que dicha lista se vaya armando durante la ejecución de la **App**, lo cual definiremos luego dentro del **Blocks Editor**.

Continuando con el diseño, necesitamos colocar ahora tres botones, los cuales cumplirán la función de grabar, parar o reproducir un determinado archivo de sonido. Como se puede ver en la siguiente imagen, hemos colocado los mismos alineados dentro de un **HorizontalArrangement**.



Modificaremos además el aspecto de dichos botones, colocando como fondo de los mismos los siguientes archivos de imagen, cada uno con el icono correspondiente a la función deseada:

Una vez que hayan guardado los archivos aquí incluidos, deberán subirlos al entorno de diseño por medio del cuadro **Media**, o como lo hiciéramos para el ejemplo del proyecto anterior, dentro de la columna **Properties**. Luego, y debajo de una **Label** que hemos utilizado como "Titulo", colocamos un componente de tipo **Slider**, al cual hemos renombrado como "Volumen".

Sobre el Componente Slider (deslizador)

El componente **Slider** es similar a una barra de progreso, la cual cuenta con un cursor deslizable conocido como "**thumb**". El usuario podrá arrastrar dicho cursor, luego de lo cual se ejecutará el evento **PositionChanged**, el que informará la nueva posición de dicho cursor.
Utilizaremos la posición informada por dicho evento para modificar la propiedad **Volume** del componente **Player**. Dicha propiedad determina el nivel de sonido para la reproducción, con un valor por default de '**50**' y un máximo de '**100**'.

Por dicha razón, estableceremos para nuestro **Slider** las siguientes propiedades con los siguientes valores: **MaxValue**='**100**', **MinValue**='**0**' y **ThumbPosition**='**50**'. De este modo, el usuario podrá modificar todo el rango de valores de volumen por medio de dicho cursor o **thumb**.

Programación

En el caso de la presente programación, comenzaremos por explicar de manera sencilla, el uso de los bloques que necesitamos para operar con los distintos componentes hasta aquí presentados, para luego combinarlos y lograr de esa forma el comportamiento deseado.

Uso del Componente SoundRecorder

Comenzaremos por explicar cómo grabar y utilizar un nuevo archivo de audio. Para ello, bastará con utilizar el bloque **call_SoundRecorder1.Start**. Colocaremos este dentro del evento **when_Grabar.Click**, el que se disparará cuando el usuario presione el botón "Grabar".

El proceso de grabación continuará hasta momento que se ejecute el bloque **call_SoundRecorder1.Stop**. Por esta razón, lo hemos colocado dentro del evento **when_Stop.Click**, el cual se dispara cuando el usuario presiona el botón "Stop".

Vamos a utilizar ahora el evento **when_SoundRecorder1.AfterSoundRecorded**, el cual se ejecuta una vez que haya sido completada la grabación de un nuevo archivo de audio. En nuestro caso, esto sucederá cuando el usuario presione el botón "Stop", luego de haber iniciado una grabación.

Dicho evento nos devolverá, por medio de su argumento **sound**, una cadena de caracteres, la cual representa la ruta al archivo recientemente grabado dentro del dispositivo **Android**.

En líneas generales, dicha ruta estará compuesta por los nombres de los directorios que conforman el camino hasta nuestro archivo, y finalizará con el nombre del archivo al que se hace referencia. Estos nombres estarán separados por el carácter "/", conocido como carácter delimitador.

Al grabarse un nuevo archivo, la ruta hacia el mismo se determina automáticamente, por lo que no debemos preocuparnos más que en copiar dicha ruta al lugar donde deseemos utilizarla.

De momento asignaremos dicha ruta sólo al bloque **set_Player1.Source_to**, que es el setter para la propiedad que determina cual será el archivo que **Player1** deberá reproducir.

Posteriormente vamos a incluir los bloques que necesitamos para almacenar la ruta mencionada dentro de la base de datos **TinyDB1**.

Uso del Componente Player

Este es un componente del tipo **multimedia** que nos permite reproducir distintos archivos de audio, así como también hacer funcionar el vibrador del dispositivo.

Para poder reproducir un archivo, deberá especificase la ruta del mismo por medio de la propiedad **Source** (fuente), lo que se puede realizar tanto dentro del **Designer**, como dentro del **Block Editor**. Esto último es lo que se muestra dentro de la imagen anterior.

La diferencia principal entre el componente **Player** y el componente **Sound** es que el primero resulta mejor para aquellos archivos de larga duración, como es el caso de las canciones, mientras que el segundo es más eficiente a la hora de reproducir archivos pequeños, como lo son los efectos de sonido.
Una vez que la propiedad **Source** haya sido correctamente establecida, podremos reproducir el archivo indicado utilizando el bloque **call_Player1.Start**. Haremos esto simplemente colocando dicho bloque dentro del espacio de tareas del evento "**Click**" del botón "**Play**".

De este modo, cuando el usuario toque sobre el botón Play, la reproducción comenzará, y continuará hasta alcanzar el final del archivo. Sin embargo, si deseamos detener dicha reproducción manualmente, podemos utilizar el bloque **call_Player1.Stop**.

Bastará con colocar dicho bloque de comando dentro de un bloque de control de eventos. En nuestro caso, vamos a colocarlo dentro del bloque **when_Stop.Click**, a continuación de **call_SoundRecorder1.Stop**, lo que permitirá al usuario detener además la reproducción que **Player1** pueda estar llevando a cabo.

No hemos tenido en cuenta aquí, si es que se encuentran o no en ejecución alguno de los dos procesos (grabación o reproducción) ya que no es necesario, y el llamado a

cualquiera de estos dos **métodos "Stop"** no produce ningún tipo de error, aun cuando no haya nada en ejecución.

Uso del Componente Slider

Otro elemento que debemos tener en cuenta para la reproducción es el **Slider** "**Volumen**", el cual hemos colocado con la intensión de que el usuario sea capaz de modificar el nivel de sonido.

Por esta razón, utilizaremos el evento **when_Volumen.PositionChanged**, el cual se ejecutará luego de que el usuario haya cambiado la posición del cursor, devolviendo como valor el argumento **thumbPosition**. Utilizaremos dicho argumento para modificar la propiedad **Volume** de **Player1**, como se muestra a continuación:

El valor del argumento **thumbPosition** será proporcional a la posición del cursor dentro del **Slider**, por lo que en nuestro caso estará entre '**0**' y '**100**', que fueron los valores mínimo y máximo con los cuales hemos configurado a dicho componente dentro del **Designer**.

Uso del Componente TinyDB

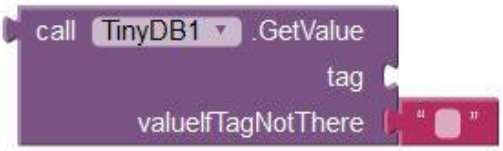
Como explicamos más arriba, vamos a utilizar el **TinyDB1** para guardar y recuperar las rutas a los archivos de voz que el usuario vaya grabando, y para hacer referencia a cada una de ellas, utilizaremos unos nombres conocidos como **tags**.

Los **tags** son similares a los nombres de las variables dentro de las **App**, en el sentido de cada uno de estos sirven para identificar unívocamente a un dato determinado. La diferencia sustancial se debe a que estos estarán almacenados en un espacio físico distinto dentro del dispositivo, por lo que la manera de trabajar con ellos es algo diferente.

Para poder almacenar un nuevo dato dentro de la **TinyDB1**, necesitamos utilizar el bloque **call_TinyDB1.StoreValue** (almacenar valor en **TinyDB1**). Dicho bloque requiere que se le informe tanto el nuevo **valor a guardar** como el **nombre** bajo el cual debe ser guardado, utilizando respectivamente para ello sus encastres **valueToStore** y **tag**.

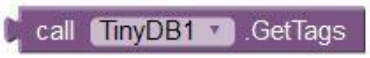
En caso de que deseemos realizar le operación inversa, podremos utilizar el bloque **call_TinyDB1.ClearTags**, el cual nos permitirá borrar el dato almacenado para un **tag** determinado.

Una vez que dispongamos de algún dato dentro de la **TinyDB1**, podremos recuperarlo utilizando el bloque **call_TinyDB1.GetValue**, al cual nos devolverá el valor almacenado para el **tag** que determinemos en el encastre que lleva dicho nombre.



Conviene comentar además que el bloque **"_"**, el cual vemos conectado al encastre **valueIfTagNotThere**, viene asignado a **call_TinyDB1.GetTags** por defecto. Dicho encastre sirve para definir un texto que será devuelto sólo en el caso de que el **tag** requerido resulte erróneo.

Por esta razón, resulta deseable que al momento de utilizar dicho comando, contemos con el nombre de alguno de los **tags** existentes. Para ello vamos a utilizar el comando **call_TinyDB1.GetTags**, el cual nos devolverá un listado con todos los **tags** que **TinyDB1** tiene disponibles. En el caso de que la **TinyDB** no cuente con ningún dato, el valor devuelto será igual al de una lista vacía.



Para el caso de la presente **App**, vamos a utilizar una variable con la que podamos contener la información devuelta por dicho comando. Inicializaremos para ello una **variable** del tipo **list** a la que llamaremos **ListaTags**.

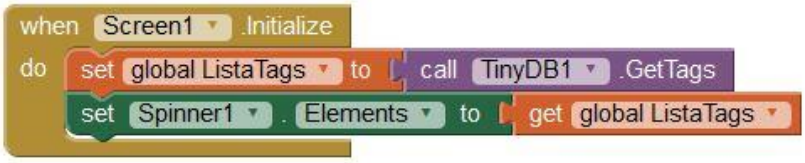
La ventaja que nos da el tener el listado de los **tag** disponible en una variable del tipo **list**, es que esta nos permitirá el uso de todos los operadores que **App Inventor** tiene disponible para este tipo de variables, facilitando así la programación.

Por esta razón, necesitaremos que al iniciar la **App**, la **variable ListaTags** recupere el listado de las **tags** mencionados, asignando dicho bloque de comando al setter **set_global_ListaTags**. Colocaremos para ello a dicho conjunto de bloques dentro del evento **when_Screen1.Initialize**, el cual sirve para ejecutar una secuencia de bloques al ingresar a **Screen1**, o lo que es lo mismo para este caso, al iniciar la **App**.

En encontraremos este bloque de control de eventos dentro de la columna **Blocks**, en el menú **Screen1**.

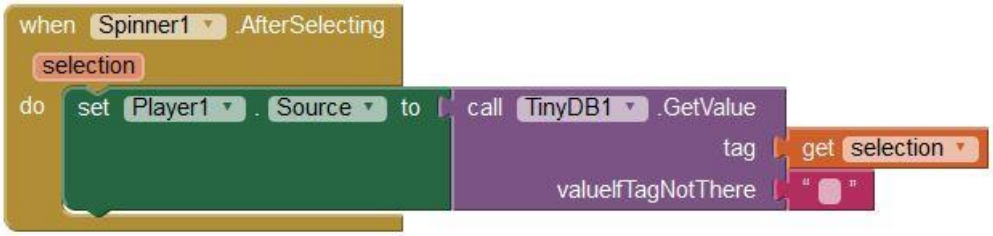
Uso del Componente Spinner

Como comentamos anteriormente, el componente **Spinner** sirve para mostrar en pantalla una lista de **elementos**, dentro de la cual el usuario podrá escoger la opción deseada. Para establecer la lista de elementos a mostrar, utilizaremos el bloque **set_Spinner1.Elements_to**, al cual le asignaremos el bloque **get_global_ListaTags**.



Como se puede apreciar, hemos colocado dicho conjunto de bloques dentro del evento **when_Screen1.Initialize**, a continuación del conjunto anterior.
Continuando con la programación, supongamos que disponemos ya de grabaciones de voz, por lo que podremos reproducirlas utilizando nuestro componente **Player1**. Con esto en mente, utilizaremos el evento **when_Spinner1.AfterSelecting** para definir una nueva fuente del audio.

Dicho evento se ejecutará luego de que el usuario haya realizado una elección dentro de la ventana que se presenta al tocar sobre **Spinner1**, e informará el valor de dicha elección por medio del argumento "**selection**".



Como podemos ver en la imagen, dentro del área de tareas de **when_Spinner1.AfterSelecting** hemos colocado el setter **set_Player1.Source_to**, asignándole a este el bloque **call_TinyDB1.GetValue**, el cual nos devuelve el valor almacenado en **TinyDB1** para un **tag** igual al argumento **selection**.

Nótese aquí que el argumento **selection** coincidirá siempre con alguno de los **tags** del listado informado anteriormente por **call_TinyDB1.GetTags**, logrando de este modo el comportamiento buscado.

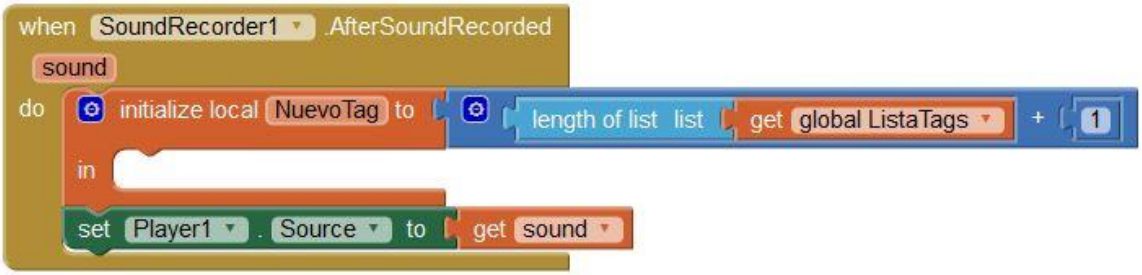
Almacenando Datos dentro de la TinyDB

Como comentamos más arriba, deseamos que para la presente **App** cada **tag** sea nombrado de forma numérica, comenzando la cuenta a partir del número "**uno**". Teniendo en cuenta que dichos **tags** estarán disponibles dentro de la **ListaTags**, podremos determinar cuál será el siguiente de ellos teniendo en cuenta el largo de la misma.

Para averiguar el largo actual de la **ListaTags**, vamos a utilizar el bloque **length_of_list**, el cual nos devuelve la cantidad de elementos de la lista que se le haya asignado.

Luego, bastará con aumentar dicha cantidad en '**1**' para obtener el valor de nuestro nuevo **tag**.

Asignaremos este conjunto de bloques a una **variable local** de nombre **NuevoTag**, la que colocaremos dentro del evento **when_SoundRecorder1.AfterSoundRecorded**.



Una vez hecho esto, vamos a almacenar la ruta informada por **sound**, utilizando para ello el bloque **call_TinyDB1.StoreValue**. Asignaremos a dicho bloque los getters **NuevoTag** y **sound** en los encastres **tag** y **valueToStore**, respectivamente.

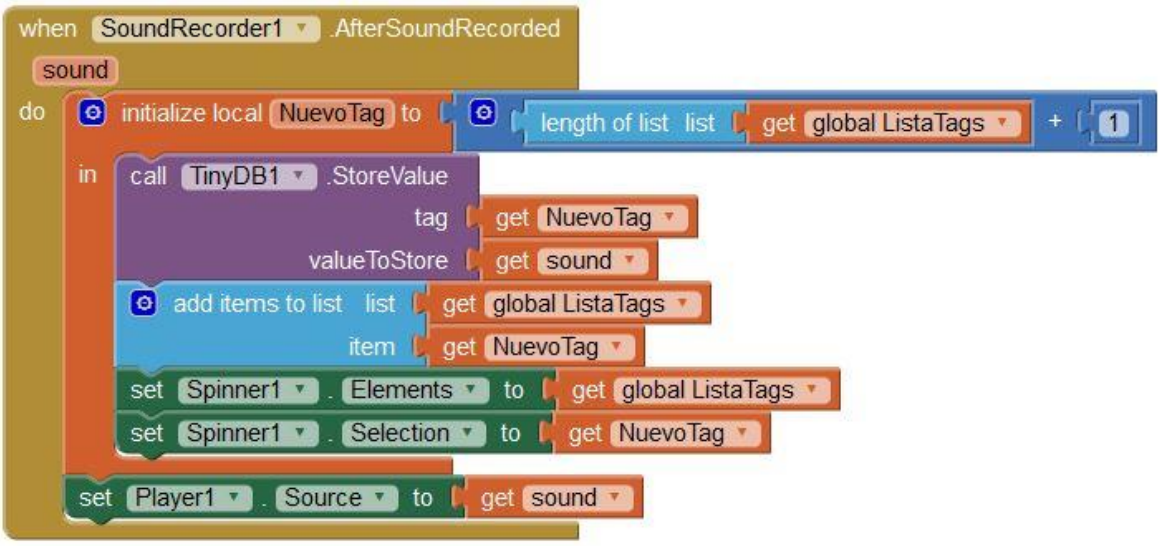
De este modo lograremos que la cadena de caracteres que representa la ruta del archivo recién grabado, sea "salvada" para su posterior lectura.

Nótese que en el caso de que esta sea la primera grabación realizada, no se dispondrá de ningún **tag**, por lo que la **ListaTags** estará vacía y **NuevoTag** resultará igual a '**1**', indicando de este modo que es la primera ruta salvada. Luego, necesitaremos que el **NuevoTag** sea agregado a la **ListaTags**.

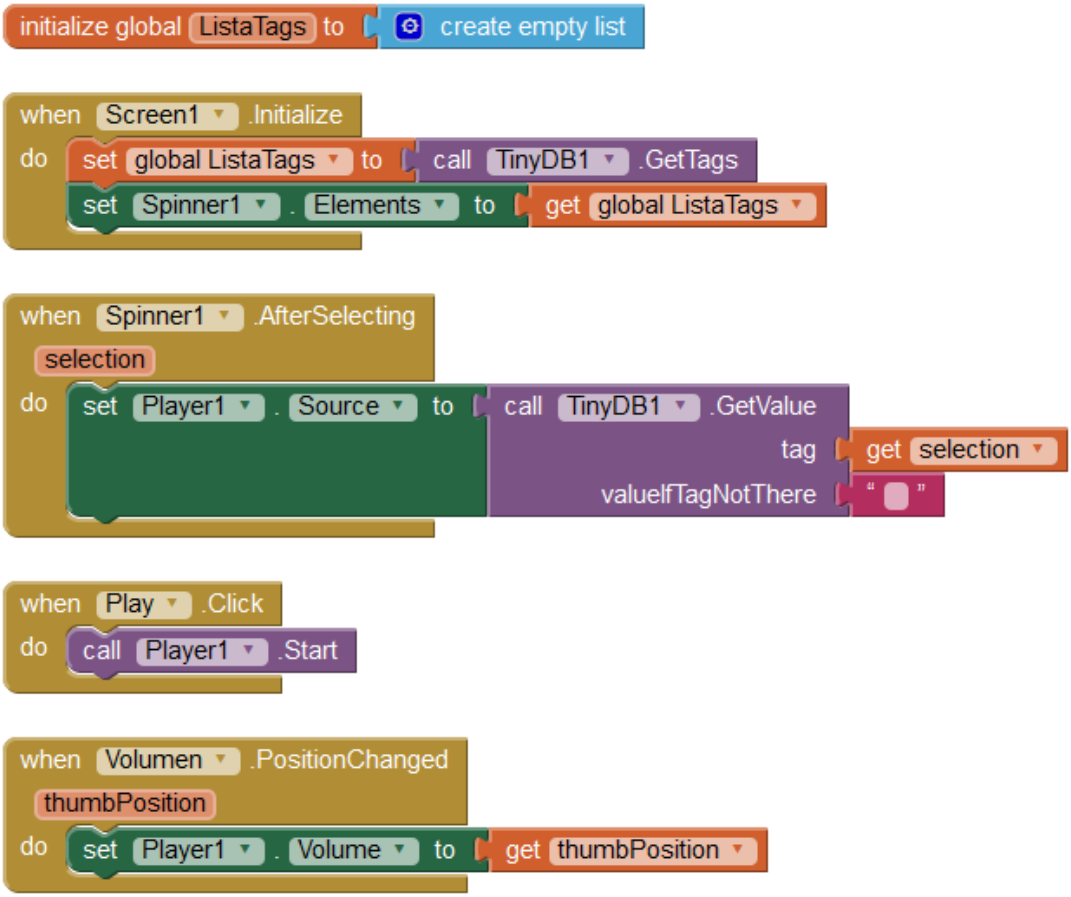
Utilizamos para ello el bloque **add_items_to_list**, el que requiere se le informe el nombre de la lista y el ítem a agregar.

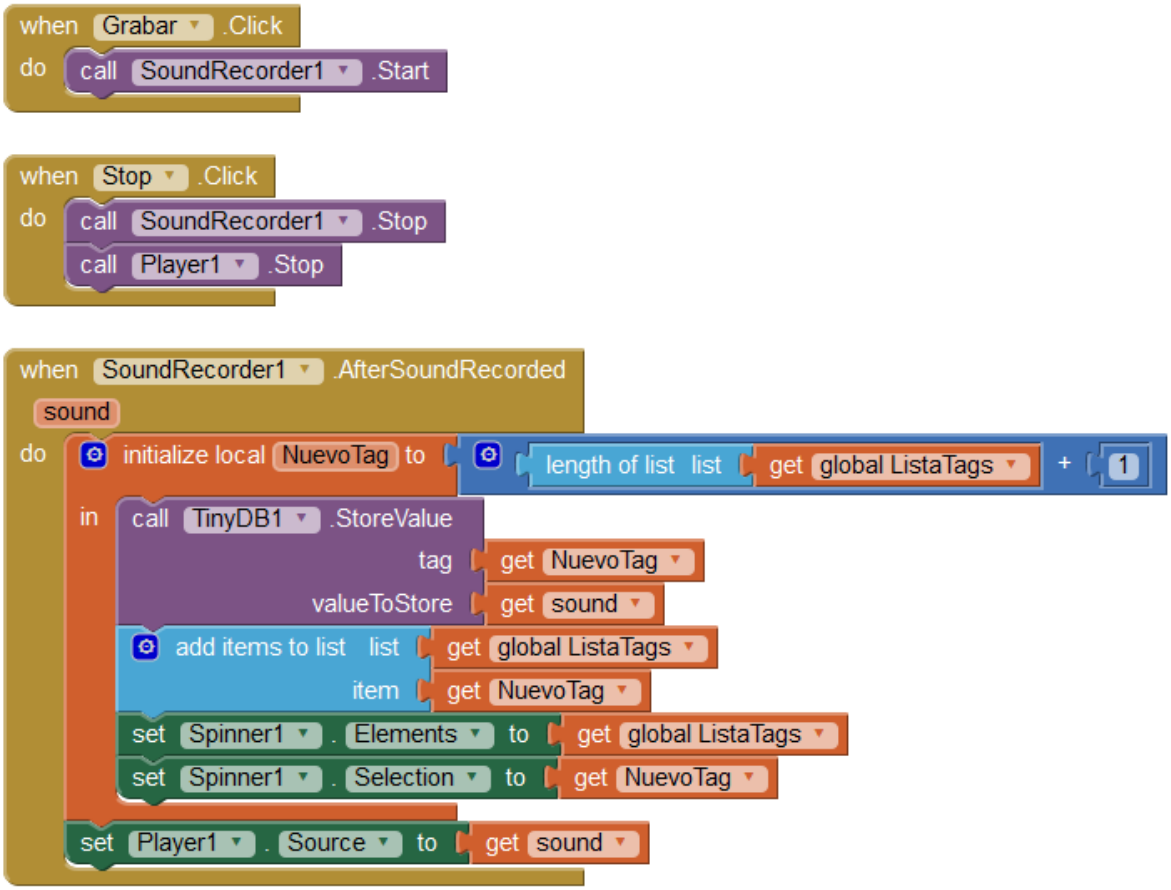
Adicionalmente, vamos renovar la lista de elementos asignada al **Spinner1**, para que esta incluya al nuevo **tag**. Modificaremos también la selección actual de dicho **Spinner** para que coincida con el archivo recientemente grabado.

Para esto último, utilizaremos los setters **Spinner1.Elements** y **Spinner1.Selection**, asignándoles respectivamente los getters **global_ListaTags** y **NuevoTag**:



El algoritmo completo nos quedará de la siguiente forma:





Sensores en App Inventor

Objetivo

En este módulo vamos a aprender sobre el uso de los componentes del tipo "**Sensor**". Aprenderemos también sobre uso del componente **Notifier**, uso de **ImageSprites** para desplazar imágenes dentro de un **Canvas**, así como otros conceptos propios del editor de bloques.

Obteniendo Coordenadas y Direcciones

Vamos a construir una **App** que utilizará las características del componente **Location-Sensor**. Este es un componente del tipo sensor, por lo que lo vamos a encontrar en menú **Sensors**, dentro de la columna **Palette**.

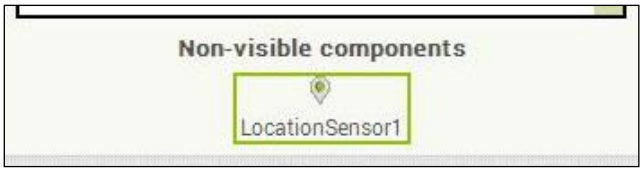
Sobre el Componente LocationSensor

Este es un componente que nos proporciona información sobre la ubicación geográfica del dispositivo **Android**, incluyendo la dirección, la longitud, la latitud y la altitud (esta última sólo para los dispositivos que cuenten con tal capacidad).

También nos permitirá obtener la longitud y la latitud de una dirección dada, sin necesidad de que esta coincida a la ubicación actual. Para poder utilizar el **LocationSensor**, será necesario corroborar que su propiedad **Enabled** este definida como **true**, mientras que el dispositivo **Android** deberá tener habilitado el uso de redes inalámbricas y/o del GPS, al momento de utilizarla **App**.

Coloquemos ahora uno de estos **LocationSensors** dentro la **Screen1** de un nuevo proyecto. Nótese que al soltar este componente dentro de dicha pantalla, el mismo no aparece dentro de la misma, sino debajo de ella y bajo el título **Non-visible components** (componentes no visibles).

Esto es así porque la funcionalidad de los componentes sensores es **interna** a la **App**, y no serán visibles al usuario más que por sus efectos sobre otros componentes.

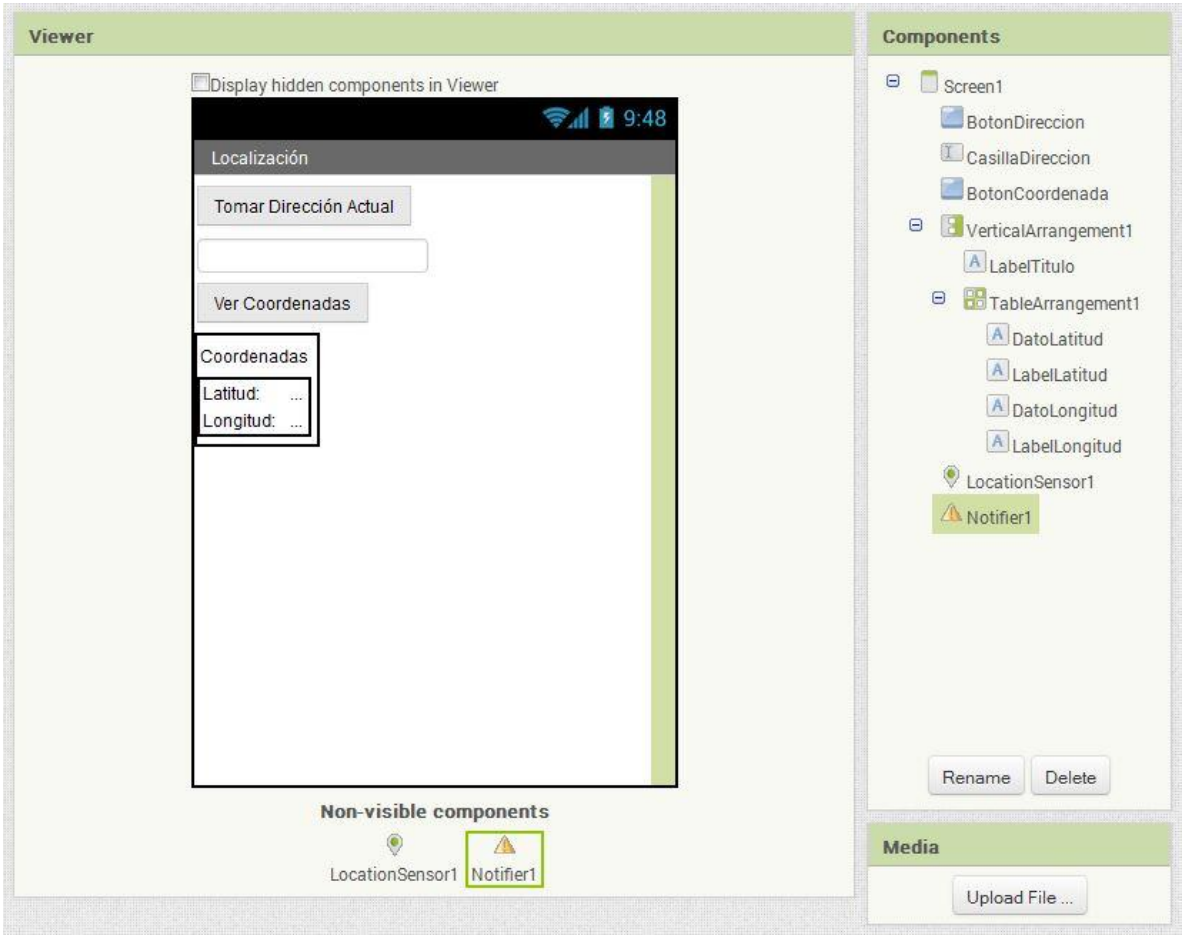


En nuestro caso, utilizaremos para tal fin, un componente del tipo **TextBox** (**casilla de texto**), el cual encontraremos dentro del menú **User Interface**.
Usaremos esta **TextBox** tanto para mostrar la dirección de la ubicación actual, como para permitir el ingreso de una dirección deseada. Renombraremos la misma como "**CasillaDireccion**".

Dentro de la propiedad **Hint** de la "**CasillaDireccion**" colocamos el siguiente texto: "Obtenga dirección actual o escriba aquí para buscar coordenadas".

Encima y debajo de "**CasillaDireccion**", colocaremos dos **botones**, uno destinado a tomar la **dirección actual**, y el otro a informa las **coordenadas** de una dirección dada.

Más abajo colocamos unas **Labels** dentro de una serie de **Arragements**, todo esto destinado a formar un cuadro que nos servirá para mostrar las coordenadas de la dirección de "**CasillaDireccion**".



Hemos agregado también un componente del tipo **Notifier**, el cual nos aparece debajo de **Viewer** como un **componente no visible** de nombre **Notifier1**.

Sobre el Componente Notifier

El componente **Notifier** sirve para mostrar cuadros de diálogo, mensajes y alertas temporales, con el fin de notificar al usuario acerca de un suceso o situación, la cual programaremos dentro del **Blocks Editor**, por medio de bloques de comando específicos para ello.

Programación

Uso del Componente Notifier

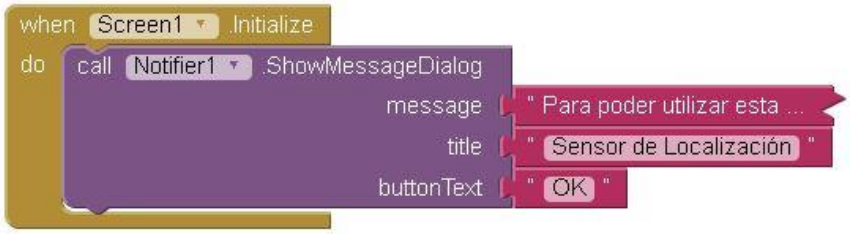
Una vez dentro del **Blocks Editor**, encontramos en la columna **Blocks** al menú **Notifier1**, del cual seleccionaremos el bloque **call_Notifier1.ShowMessageDialog**. Este es

un bloque de comando que sirve para mostrar un mensaje en la pantalla, el cual se cerrará cuando el usuario pulse un botón destinado para tal fin.

Dicho bloque cuenta con tres encastres de entrada, los cuales son: **message**, **title** y **buttonText**. El primero de ellos recibirá el texto de mensaje a mostrar, el segundo será el título del mensaje, mientras que el tercero sirve para establecer la leyenda del botón que el usuario debe presionar para cerrar el mensaje.

En nuestro caso utilizaremos este bloque al inicio de la **App**, con la intensión de informar que: **"Para poder utilizar esta aplicación, es necesario tener activados los servicios de localización. Asegúrese de activarlos en Ajustes -> Ubicación y seguridad -> Mi Ubicación"**.

Con esto en mente, colocaremos nuestro bloque de comando dentro del evento **when_Screen1.initialize**, asignándole el texto del mensaje por medio de un bloque del tipo **"_"**.



Como podemos ver en la imagen de arriba, también colocamos sendos bloques del tipo **"_"** con los textos "Sensor de Localización" y "OK", asignados respectivamente a los encastres **title** y **buttonText**.

Colapsar y Expandir los Bloques

En la imagen anterior podemos ver que el bloque de texto que asignamos al encastre "**message**" aparece **truncado**. Esto es debido a que hemos **colapsado** el bloque para disminuir su largo, al hacer clic derecho sobre este y seleccionar "**Collapse Block**" dentro del menú desplegable que se nos presenta.

Para deshacer esta acción bastará con volver a hacer clic derecho y seleccionar "**Expand Block**" dentro del nuevo menú desplegable.

LocationSensor – Obteniendo una Dirección Física

Continuaremos la programación de la **App** explicando el uso de algunas de las funciones del componente **LocationSensor**. Comenzaremos por hacer uso del bloque **LocationSensor1.CurrentAddress**, el cual nos devuelve la dirección física (**calle** y **altura**) de la ubicación geográfica actual del dispositivo **Android**.

Para ello, seleccionaremos el bloque **LocationSensor1.CurrentAddress**, el cual se encuentra dentro del menú **LocationSensor1**. Asignaremos dicho bloque al setter de la propiedad **Text** de **CasillaDireccion**.

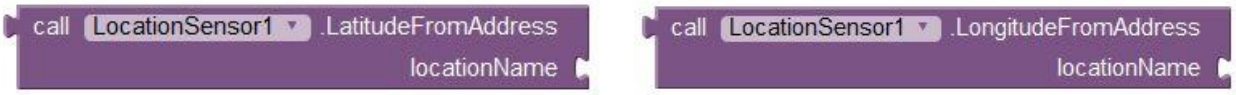
Luego, colocaremos dicho conjunto dentro del espacio de tareas del bloque **when_BotonDireccion.Click**, de modo que nos quede como podemos ver a continuación.

Cuando el usuario presione el **BotonDireccion**, **LocationSensor1.CurrentAddress** asignará la dirección física del dispositivo a **CasillaDireccion**, logrando de este modo mostrar dicha información en pantalla; todo esto sucederá siempre y cuando el usuario haya activado los servicios de localización de **Android**.

LocationSensor – Obteniendo Coordenadas Geográficas

Haremos algo similar con el **BotonCoordenada** pero a la inversa. Esto es, obtener las coordenadas (**latitud** y **longitud**) a partir una dirección dada, la que podremos ingresar dentro de la **CasillaDireccion** al tocar sobre ella.

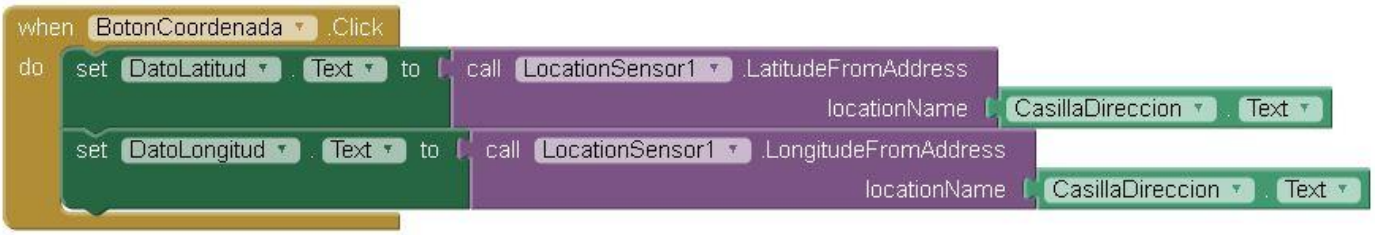
Utilizaremos para tal fin los bloques **call_LocationSensor1.LatitudeFromAddress** y **call_LocationSensor1.LongitudeFromAddress**.



El primero de estos nos devuelve la latitud geográfica de la ubicación informada en el encastre **locationName**, mientras que el segundo nos informa la longitud. La información suministrada al encastre **locationName** deberá ser una cadena de texto que represente una dirección física valida (**calle** y **altura**).

En nuestro caso, utilizaremos la propiedad **Text** de **CasillaDireccion** para obtener la dirección a consultar, por lo cual el usuario ha de ingresarla en dicho **TextBox** con anterioridad.

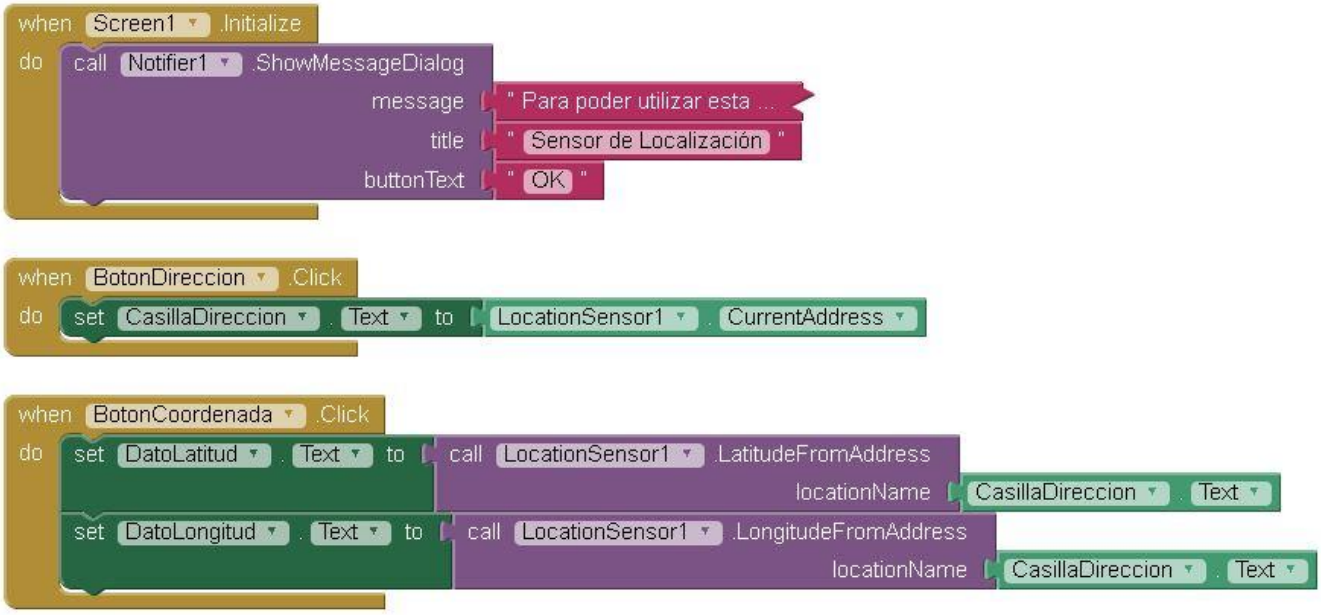
Por último, vamos a asignar los mencionados bloques de comando a los setters **set_DatoLatitud.Text_to** y **set_DatoLongitud.Text_to**. Colocaremos la secuencia de



bloque completa dentro del evento **when_BotonCoordenada.Click**

Cuando el usuario presione el **BotonCoordenada**, la latitud y longitud de la dirección física informada en **CasillaDireccion** se mostrará en pantalla, dentro de las **Labels** destinadas para tal fin.

Nuevamente, para que todo esto suceda, el usuario deberá haber habilitado los servicios de localización dentro del dispositivo **Android**. El programa completo nos queda de la siguiente forma:



Haciendo uso del AccelerometerSensor

La siguiente **App** simulará el lanzamiento de dos dados, de modo que la misma podrá ser utilizada para llevar a cabo algún tipo de juego de mesa.

Diseño y Configuración

A continuación les presentamos una imagen, en la que podrán apreciar el diseño y los componentes que hemos utilizado para esta **App**, a la que hemos nombrado simplemente como "Dados".

En ella tenemos los siguientes elementos:

- Un componente **Canvas** de nombre **Canvas1**
- Dos componentes **imageSprites** de nombres **Dado1** y **Dado2**
- Un componente **AccelerometerSensor** de nombre **AccelerometerSensor1**
- Un componente **Player** de nombre **Player1**
- Seis archivos de imagen de extensión "**.jpg**"
- Un archivos de sonido de extensión "**.mp3**"

Sobre el Componente ImageSprites

En la imagen anterior podemos ver el diseño de esta **App**, en la que hemos colocado un componente **Canvas1**. Dentro de este hemos colocado dos **imageSprite**, a los que renombramos como **Dado1** y **Dado2**. Todos estos componentes los podemos encontrar dentro de "**Drawing and Animation**".

Los **imageSprites** son un tipo especial de componente para imágenes, cuyo uso está destinado a los **Canvas**, permitiendo el desplazamiento de dichas imágenes en el interior de dichos cuadros. Estos pueden responder a las acciones del usuario (tocar y

deslizar), moverse de acuerdo al valor de ciertas propiedades, e interactuar con otros **ImageSprites** y/o con los bordes del **Canvas** al que pertenecen.

La imagen a visualizar dentro de este componente se especificará por medio de su propiedad **Picture**. Por otro lado, si deseamos que el **imageSprites** deje momentáneamente de estar visible, podremos utilizar la propiedad **Visible**, definiendo su valor como **false**.
Si deseáramos tener un **imageSprite** moviéndose hacia la izquierda a, por ejemplo, 10 píxeles por segundo, deberíamos ajustar las siguientes propiedades, a los siguientes valores:

- **Speed** (velocidad): 10 [píxeles]. Indica la velocidad en píxeles por unidad de tiempo.
- **Interval** (intervalo): 1000 [milisegundos]. Define la unidad de tiempo o intervalo entre los cambios de posición.
- **Heading** (Inclinación): 180 [grados]. Indica la dirección de la imagen.
- **Enabled** (Habilitado): Tildada (o sea, **true**). Para habilitar o deshabilitar el movimiento.

Por otro lado, un **imageSprite** cuya propiedad **Rotate** sea igual a **true**, podrá girar su imagen según la dirección y los cambios de rumbo de esta. Cualquiera de todas estas propiedades puede ser modificada, tanto dentro del **Designer**, como durante la ejecución del programa por medio del **editor de bloques**.

Sobre el Componente AccelerometerSensor

Continuando con el diseño, colocaremos ahora un componente sensor del tipo **AccelerometerSensor** (sensor de aceleración), el cual aparecerá debajo de la pantalla seleccionada, ya que es un **Non-visible component**.

Usaremos este componente con la intensión de "agitar" los dados dentro del **Canvas1**. Podremos hacerlo gracias a que este componente es capaz de detectar si nuestro dispositivo **Android** está siendo sacudido, o se lo ha movido bruscamente en alguna dirección, ya que como lo indica su nombre, mide la aceleración de dichos movimiento.

También es capaz de decirnos la inclinación del dispositivo respecto de la superficie terrestre. Esto es posible gracias el efecto que el campo gravitatorio de la tierra tiene sobre el **AccelerometerSensor**. Este efecto se mide en [m/s^2], que es la unidad SI en la que se mide la aceleración de la gravedad.

Por último, colocaremos un componente **Player**, el cual vamos a utilizar para reproducir el sonido de unos dados siendo mezclados dentro de un cubilete. Necesitaremos además subir al proyecto los siguientes archivos multimedia:


Cara1.JPG


Cara2.JPG


Cara3.JPG


Cara4.jpg


Cara5.jpg


Cara6.jpg


Sonido de Dados.mp3

Los archivos de imagen "**Cara'N'.jpg**" representan las diferentes caras de un dado, según el número de '**N**'. Utilizaremos respectivamente los archivos "**Cara1.jpg**" y "**Cara2.jpg**", para establecer inicialmente el valor de las propiedades **Dado1** y **Dado2**.

Estableceremos también, dentro del entorno de diseño, el archivo "**Sonido de Dados.mp3**" como **Source** para el componente **Player1**.

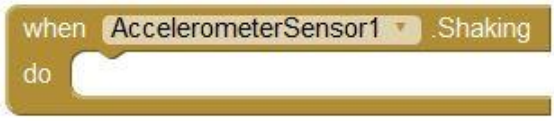
Como comentamos más arriba, los **imageSprites** serán capaces de moverse dentro del **Canvas**. Para que esto suceda, necesitamos que la propiedad **Speed** (velocidad) de dichos **imageSprites** tenga un valor distinto de '**0**', por lo que le hemos dado un valor de '**25**'.

Por otro lado, necesitamos que los dados se muevan **sólo** luego de que el dispositivo **Android** haya sido agitado, por lo que vamos a destildar la propiedad **Enabled** de ambos **imageSprites**.

Programación

Uso del Componente AccelerometerSensor

Antes de empezar con la programación, vamos a comentar un poco sobre los bloques que vamos a utilizar. Uno de ellos es el evento **when_AccelerometerSensor1.Shaking**.



Dicho evento se disparará en el momento en que el **AccelerometerSensor1** detecte un movimiento brusco del dispositivo, o sea **cuando** detecte el "**Shaking**" (agitado) del dispositivo.

Uso del Componente ImageSprite

En el caso de los **imageSprites**, tendremos en cuenta varios bloques, entre ellos los eventos **when_Dado1.EdgeReached** y **when_Dado2.EdgeReached**.

Podemos entender los mismos como: **cuando** el **Dado'N'** **Alcance** uno de los **Bordes** Haremos…

En efecto, estos son bloques de control de eventos que se dispararán cuando sus respectivos **imageSprites** alcanzan a alguno de los bordes del **Canvas** al que pertenecen. Adicionalmente, el argumento "**edge**" nos informará a cuál de los cuatro bordes alcanzó el **imageSprite** en cuestión.

Como complemento de los eventos mencionados tenemos a los bloques de comando **call_Dado1.Bounce** y **call_Dado2.Bounce**. Se puede entender a los mismos como **llamar** (**call**) al **rebote** (**Bounce**) del **imageSprite** **Dado'N'** contra el borde "**edge**".

La función de estos es muy sencilla. Al ser llamados, harán que el **imageSprite** al cual pertenecen "rebote" en la dirección contraria al borde informado en su encastre **edge**. Por dicha razón, están especialmente pensados para su uso con los bloques del tipo **when_imageSprite.EdgeReached**, los cuales cuentan con el argumento "**edge**", que es el que nos brinda la información específica sobre los bordes alcanzados.

Otro bloque útil para el uso de los **imageSprites** es el bloque de control de eventos **when_Dado1.CollidedWith**, el cual se disparará cuando el **imageSprite** el que pertenece (en este caso, **Dado1**), "colisione" con otro **imageSprite**.

Vemos que dicho evento cuenta con el argumento "**other**", el cual nos informa cuál de los otros **imageSprite** fue el que disparo el evento.

Continuaremos ahora mencionando algunos de los bloques para propiedades que vamos a necesitar para programar nuestra **App**. Tomemos por ejemplo a los setters

set_Dado1.Enabled_to y **set_Dado2.Enabled_to**, los cuales vamos a utilizar para activar el movimiento de los **Dado1** y **Dado2** cuando sea necesario.

Otros setters a tener en cuenta son el **set_Dado1.Picture** y el **set_Dado2.Picture**. Con ellos podremos modificar la imagen que se mostrará dentro de sus respectivos **imageSprite**.

También vamos a utilizar los bloques **set_Dado1.Heading** y **set_Dado2.Heading**. Estos son los setters que nos permiten establecer el ángulo de inclinación o giro de nuestros **imageSprites**, respecto del eje horizontal del **Canvas** al que pertenecen. Dicho ángulo se definirá en grados sexagesimales (de '0' a '360').

Gracias a ellos, podremos cambiar la dirección del movimiento de nuestros **imageSprites** en cualquier momento.

Uso de Bloques del Tipo Any component

Otros de los bloques que vamos a utilizar son el **set_imageSprite.Heading** y el **imageSprite.Heading**, los cuales se encuentran en el menú **Any ImageSprite**.

Estos son bloques del tipo **Any component**, lo que quiere decir que nos van a permitir definir a que componente se los aplicará en el momento mismo de ser ejecutados. El componente al cual se aplicarán deberá ser informado por medio de los encastres "**of_component**", incluido en cada uno de ellos.

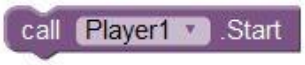
Todo esto permite aplicar una misma secuencia de código a distintos componentes, siempre y cuando estos sean del mismo tipo, como sucede en nuestra App con los imageSprites "**Dado1**" y "**Dado2**".

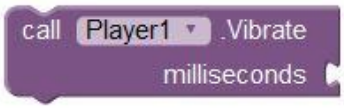
Funciones Alternativas del Componente Player

Para la programación de la presente App, vamos a necesitar utilizar algunas funciones del componente Player1 que van más allá de la sola reproducción de un sonido. Una de ellas nos la brinda el bloque when_Player1.Completed.

Este es un evento que se dispara cuando Player1 completa la reproducción del archivo indicado por su propiedad Source. En nuestro caso, será muy útil para lograr que los dados se detengan justo en el momento en que la reproducción del archivo "**Sonido de Dados.mp3**" concluya, haciendo coincidir de esta forma el movimiento de los imageSprites "**Dado1**" y "**Dado2**", con el sonido de la tirada.

Otros elementos que vamos a utilizar en la programación son los bloques de comando call_Player1.Start y call_Player1.Vibrate.





El primero de estos cumplirá con la función básica de la reproducción del "**Sonido de Dados.mp3**", mientras que el segundo está destinado a utilizar la capacidad de vibración del dispositivo Android. Dicha vibración es la misma que la que se produce en nuestro teléfono cuando tenemos activado el modo silencioso y recibimos una llamada.

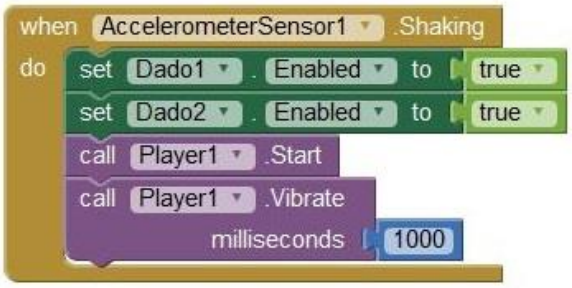
Para el uso del comando call_Player1.Vibrate, será necesario establecer el tiempo (en milisegundos) que se desea dure la vibración, asignando para ello un bloque numérico a su encastre "milliseconds".

Programación de la App

Animando los ImageSprites

Vamos a comenzar por programar la secuencia de bloques que activará el movimiento de los dados cuando el dispositivo Android sea agitado. El bloque a utilizar es el

when_AccelerometerSensor1.Shaking (cuando_SensorAcelerómetro1_seaAgitado), el que se encuentra en el menú de dicho componente.



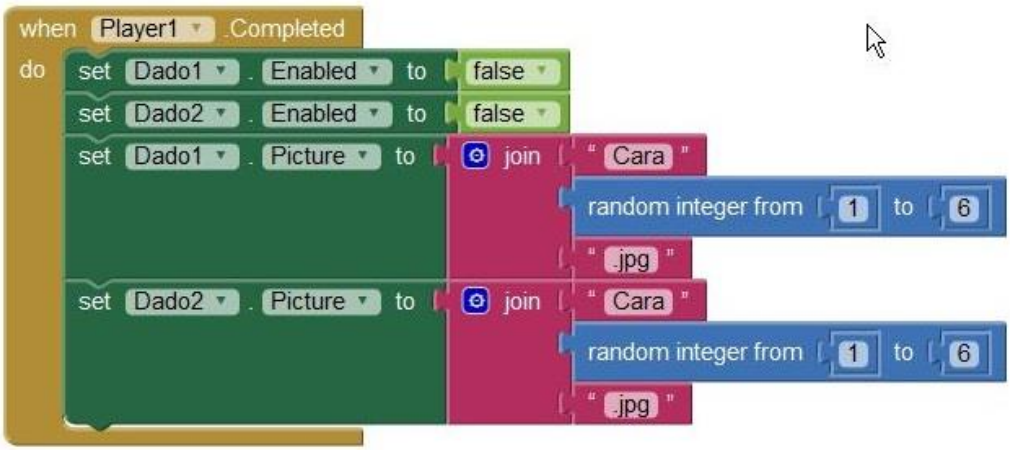
Comenzamos colocando dentro de este los bloques **set_Dado1.Enabled_to** y **set_Dado2.Enabled_to**, a los que les hemos asignado un valor de verdad igual a **true**, lo que permite de este modo que los **imageSprite** se desplacen dentro del **Canvas1**.

A continuación están los bloques **call_Player1.Start** y **call_Player1.Vibrate**, los cuales reproducen el "**sonido de dados.mp3**" y hacen vibrar al dispositivo, respectivamente. Estos se encuentran en el menú **Player1**.

Para el bloque **call_Player1.Vibrate** hemos establecido su encastre "**milliseconds**" igual a '**1000**' [mSeg], lo que hará que la vibración dure un total un segundo.

Obteniendo un Resultado

Necesitaremos luego una secuencia de bloques que detenga el movimiento de los dados, una vez que el "**sonido de dados.mp3**" deje de escucharse. Lograremos esto utilizando el bloque **when_Player1.Completed**. Dentro de él colocamos los bloques **set_Dado1.Enabled_to** y **set_Dado2.Enabled_to**, a los cuales les hemos asignado en este caso, un valor de verdad igual a **false**.



También hemos colocado aquí los bloques **set_Dado1.Picture** y **set_Dado2.Picture**, asignándoles la unión entre la palabra "**Cara**", un número aleatorio entre '**1**' y '**6**', y la extensión "**.jpg**". De este modo lograremos obtener aleatoriamente el nombre de uno de los seis archivos de imagen, de manera que cada dado nos muestre una cara distinta cada vez que se los "mezcle".

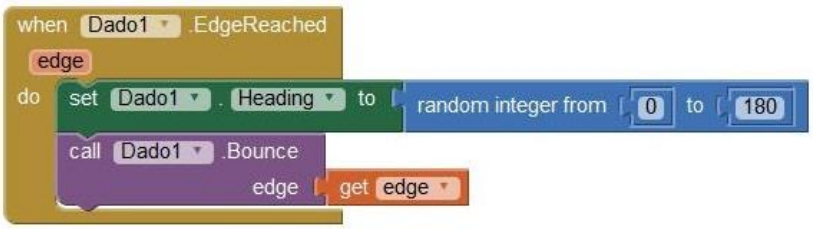
Hacer que los ImageSprites Reboten

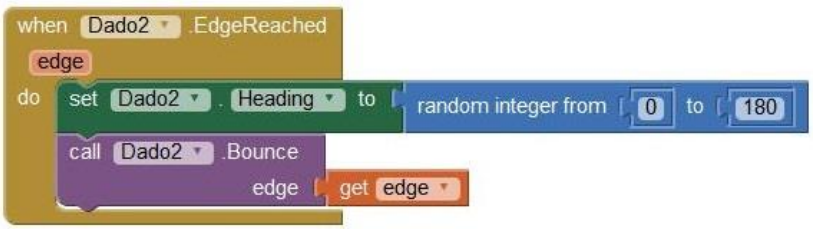
Si prueban la **App** hasta este punto, notaran que una vez que los dados llegan a un bo-de, estos se quedaran "pegados" a él. Para evitar esto, necesitamos que los mismos "reboten" cada vez que alcanzan cualquiera de los bordes.

Para esto necesitamos usar los bloques **call_Dado1.Bounce** y **call_Dado2.Bounce** que cumple con esta función específica, asociados a los bloques **when Dado1.EdgeReached** y **when Dado2.EdgeReached** de la siguiente forma:

Notarán ahora que si dejamos esto así, ambos dados rebotarán siempre perpendicu-larmente contra los bordes derecho e izquierdo del **Canvas1**. Para modificar esto y darle un aspecto más irregular (propio del mezclado), agregaremos dentro de los pre-sentes eventos a los bloques **set_Dado1.Heading** y **set_Dado2.Heading.**
Como mencionamos antes, dichos **setters** nos permitirán modificar la propiedad **Hea-ding**, cambiando así la orientación nuestros **imageSprites**, y con ella la dirección de sus respectivos movimientos.

Para ello, le asignaremos a dichos bloques **set_Dado'N'.Heading** sendos bloques **ran-dom_integer**, con un rango de '**0**' a '**180**', de manera que el ángulo del rebote varíe aleatoriamente entre estos dos valores.





Interacción entres los ImageSprites

Por último, nos falta tener en cuenta que sucederá con los dados cuando estos se encuentren. Para esto, haremos uso del bloque **when_Dado1.CollidedWith**:

Dentro del mismo hemos agregado una variable local con el fin de almacenar transitoriamente el valor del "**Angulo**" de orientación del **Dado1**, el cual intercambiaremos luego por el valor del ángulo de orientación del **Dado2**.
Para esto, utilizaremos los bloques **set_imageSprite.Heading** e **imageSprite.Heading**, los cuales se encuentran en el submenú **Any ImageSprite**, dentro del menú **Any component**.

Como comentamos anteriormente, en estos menús encontraremos bloques de funciones similares a los de un componente específico, con la diferencia de que estos se podrán aplicar a cualquier componente que sea de su mismo tipo.

Para que estos bloques funcionen correctamente necesitamos informarles a cuales componentes deben ser aplicados. Haremos esto asignándoles a ambos encastres "**of_component**" el argumento "**other**", propio del evento **when_Dado1.CollidedWith** en el que está incluidos.

Ya que en la presente **App** sólo tenemos dos **imageSprites**, parece innecesario utilizar este tipo de bloques. Sin embargo, deseábamos incluirlos en ella con el fin de dar a conocer su funcionalidad de la manera más sencilla posible.

Este tipo de bloques nos resultarán muy útiles cuando deseemos referirnos a más de un componente de un mismo tipo, dentro de una misma secuencia de bloques, dando como resultado un programa con menor cantidad de bloques.

Uso de Procedimientos e IF Multiples

Objetivo

En este módulo aprenderemos sobre el uso de los bloque **de procedimiento (Procedures)** así como también el modo de expandir nuestros **bloques if** para llevar a cabo selecciones múltiples, lo que nos permitirá hacer uso del concepto de máquina de estados.

Programemos una Calculadora!!

Vamos a hacer una clásica calculadora, la cual luego podremos modificar para agregar otras funciones. Crearemos para esto, un nuevo proyecto al que llamaremos **Calculadora**.

Diseño y Configuración

Ya dentro del entorno de diseño, cambiaremos el título de la pantalla para que lleve el mismo nombre del proyecto. Luego, colocaremos un **HorizontalArrangement** al que vamos a renombrar como "**Marco**", ya que servirá como tal para nuestro display. Dentro de este colocaremos un componente **Label**, al que renombraremos como "**Display**".

Uso de TableArrangements

Debajo del **Arrangement** que colocamos recién, colocaremos otro pero en este caso del tipo **TableArrangement**, el cual se encuentra en el mismo submenú. Renombraremos a este como "**Teclado**".

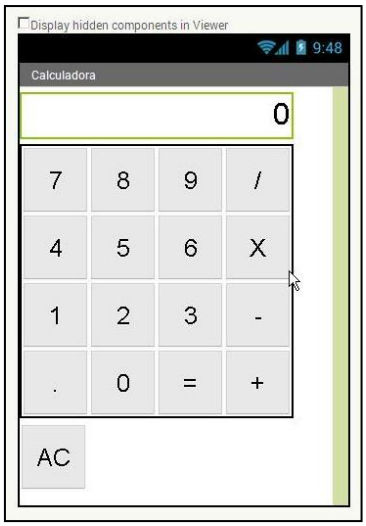
Los **TableArrangements** nos sirven para organizar distintos componentes en forma de tabla, ya que los mismos forman cuadrículas de varias celdas, ordenadas por filas y columnas. Si bien es posible colocar más de un componente dentro de cada celda, esto no resulta conveniente ya que de ser así, sólo el último componente colocado será visible.

En nuestro caso, vamos a utilizar este **TableArrangement** para colocar una serie de botones que formarán el teclado de la calculadora. Con esto en mente, vamos a modificar el valor de las propiedades **Columns** (columnas) y **Rows** (filas), ambas en un valor igual a '**4**', obteniendo así un total de 16 cuadros.

Ahora, dentro de cada uno de los cuadro de esta tabla vamos a colocar un componente **Button**, de manera que vayamos dando forma al teclado de la calculadora.

Determinando el Tamaño de los Componentes

En la imagen que aparece a continuación, podremos ver el diseño descripto hasta el momento, en el cual hemos modificado varias de las propiedades correspondientes al tamaño (**Width** y **Height**), tanto de los **Buttons** como del componente **Label**.



En nuestro caso, hemos utilizado un valor de "65 pixels", tanto para la altura como para el ancho de los botones, mientras que para el alto y ancho del "**Display**", hemos utilizado un valor de "40 pixels" y "272 pixels", respectivamente.

Por otro lado, las propiedades de tamaño correspondientes a los **Arrangements** hasta aquí utilizados, las hemos dejado como **Automatic**, de manera que los mismos se adaptarán a las medidas de los compontes colocados en su interior.

Hemos modificado además el tamaño de las fuentes de texto (**FontSize**, dentro de **Properties**) a un valor igual a '**28**' para los botones y de '**35**' para el **Display**, de modo que resulten más legibles.

Establecimos también las propiedades **Text** y **TextAlignment** de la **label** "**Display**" como '**0**' y "**right**" respectivamente, de manera que al arrancar la **App** su lectura sea '**0**' y su texto se **alinee** siempre a la derecha.

Por último y debajo de todo esto, colocamos un botón para "limpiar" al que renombramos como **BotonBorrar**. Cambiaremos su leyenda por **AC**, de manera que resulte similar al botón que encontraríamos en una calculadora comercial.

Conceptos para la Programación

Ya que vamos a realizar una calculadora, tenemos que tener en cuenta varias cuestiones, como por ejemplo que para resolver cualquier operación aritmética simple, será necesario el ingreso de las cifras y el operador antes de poder obtener una respuesta.

Esto quiere decir que la programación que necesitamos realizar será de naturaleza **secuencial**, siguiendo un orden determinado, por lo que el comportamiento de la **App** no va a depender **sólo** de que tecla **presionemos**, sino también de **cuándo** ha sido presionada esta.

Por todo esto, vamos a necesitar planificar nuestra programación cuidadosamente, utilizando para ello algún tipo de modelo conceptual. En nuestro caso, vamos a utilizar lo que en programación se conoce como **máquina de estados**.

Concepto de Máquina de Estados

Muchas veces cuando estamos programando, necesitamos hacer un plan o modelo de lo que queremos que suceda en determinadas situaciones. Existen muchas formas de llevar a cabo dicha tarea. Una de ellas implica utilizar la mencionada **máquina de estados**.

Se denomina **máquina de estados** al modelo que se hace de un programa, donde el comportamiento actual del mismo no depende sólo del evento actual, sino también de lo que haya sucedido en eventos anteriores.

Dicha máquina de estados se define por el conjunto de los estados que el programa puede tomar. Dicho conjunto sirven para indicar cuál deberá ser el comportamiento de la **App** según el estado en el que se encuentra.

Si bien este concepto no forma parte de **App Inventor** como tal, creemos que resultará muy conveniente el tener un conocimiento básico del mismo, el cual nos permitirá llevar a cabo programas que implican cierto grado de complejidad.

En nuestro caso, vamos a definir los **estados** que necesitamos tener en cuenta para que la presente **App** tenga un comportamiento similar al de una calculadora estándar, como la que cualquiera de nosotros puede tener en su hogar o en su computadora.

Con esto en mente, hemos definido los siguientes estados, asignándole a cada uno un número, el cual nos permitirá identificarlos. Dichos estados son:

- **Estado 0**: Este va a ser nuestro **estado inicial**, en el cual estarán definidas las condiciones necesarias para comenzar las operaciones.
- **Estado 1**: En este estado se ingresa el **primer operando**.
- **Estado 2**: Aquí se define el **operador**.
- **Estado 3**: En este estado se ingresa el **segundo operando**.
- **Estado 4**: A este estado se llega luego de presionar la tecla igual, realizar la operación deseada y presentar su resultado.
- **Estado 5**: En este estado se estará ingresando un nuevo operando, luego de que fuera presionada la tecla **igual**.
- **Estado 6**: El programa alcanzará este estado luego de que el usuario haya presionado el **BotonBorrar** una sola vez, lo que borrará **sólo** lo último que haya sido ingresado, y cambiará la leyenda de dicho botón a "**CE**" (**C**lear to **E**mpty = limpiar hasta vaciar). En caso de que el **BotonBorrar** sea presionado una segunda vez, se borrará **todo**, se restablecerán las condiciones iniciales, y se pasará al **Estado 0**.
- **Estado 7**: Este es un estado de **error**, el cual se alcanza cuando de intenta dividir cualquier número por un valor igual a '**0**'.

Concepto de Procedimiento

Como sucede muchas veces en programación, en la presente **App** vamos a contar con **tareas** o **secuencias de bloques** las cuales necesitamos repetir, ya que se requiere realizarlas en más de un **estado** o **punto del programa**. Esto sucede por ejemplo con la **secuencia de cálculo**, cualquiera sea el operador implicado.

Para poder hacer esto sin tener que programar dichas secuencias más de una vez, vamos a utilizar lo que en **App Inventor** se conocen como **Procedures** o **Procedimientos**.

Como comentamos en otra oportunidad, se conoce como **Procedimientos** a una **secuencia de bloques** o pieza de código, la cual se agrupa dentro de un solo **nombre**. Este nombre lo determinaremos al momento de crear el procedimiento.

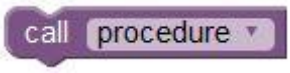
Esto permite que en lugar de tener que colocar la misma **secuencia de bloques** una y otra vez, podamos crear un **procedimiento**, el cual contenga todos los bloques tan **sólo una vez**, para luego **llamar** a dicha secuencia por medio de su **nombre**, todas y cada una de las veces donde necesitemos que la misma sea ejecutada.

Existen dos tipos de procedimientos dentro de **App Inventor**: los del tipo **do** y los del tipo **result**.

Bloques de Procedimiento "Do"

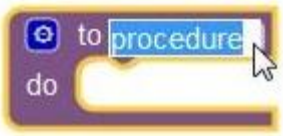
Los procedimientos del tipo **do** sirven para agrupar secuencias de bloques, colocando estas dentro de sus espacios de tareas.

Luego, podremos ejecutar dicha secuencia de bloques todas las veces que sea necesario, utilizando para ello un llamando a dicho procedimiento, de manera similar a como lo hiciéramos con los bloques de comando.



Cuando creamos un nuevo procedimiento, **App Inventor** genera automáticamente un bloque del tipo **call_procedure**, y lo coloca en el menú **Procedures** de la columna **Blocks**. Este es el bloque que utilizaremos para invocar al procedimiento.

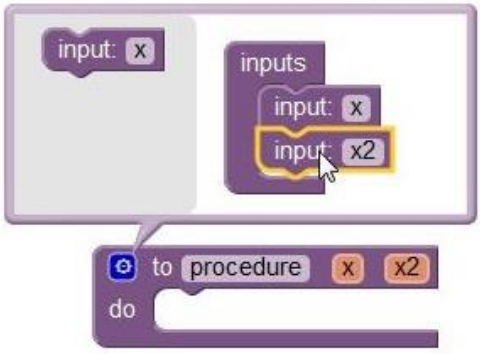
Por otro lado, al crear un nuevo procedimiento, **App Inventor** definirá el nombre del mismo de forma automática. Podremos modificar este nombre haciendo clic dentro del cuadro que lo contiene.



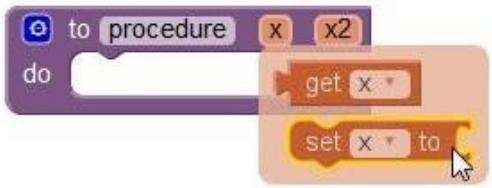
El nombre de cada procedimiento dentro de una misma **Screen** será único, por lo que no será posible definir dos procedimientos con el mismo nombre.

Podremos modificar el nombre de cualquier procedimiento en cualquier momento. Al hacerlo, **App Inventor** cambiará automáticamente el nombre de los bloques del tipo **call** asociados al mismo.

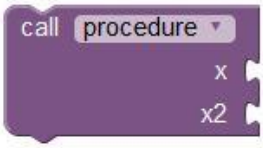
A su vez, habrán podido notar que este tipo de bloques de procedimiento pertenecen a la categoría **mutators**, ya que cuentan con un cuadrito azul en su esquina superior izquierda.



Como podemos apreciar en la imagen anterior, al cliquear en dicho cuadro azul se nos despliega una ventana en forma de globo, dentro de la cual podremos utilizar los pseudo-bloques **input** para definir nuestros propios **argumentos**. Luego, podremos utilizar dichos **argumentos** como lo hiciéramos en los bloques de control de eventos.



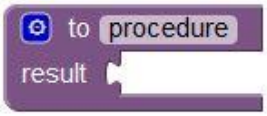
A su vez, cuando definimos nuevos **argumentos** un procedimiento, veremos que los bloques **call** asociados al mismo son automáticamente modificados, incluyendo en ellos nuevos encastres de entrada con el nombre de dichos argumentos.



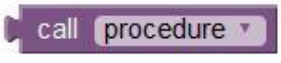
A dichos encastres les asignaremos los **parámetros** con los cuales se inicializaran los **argumentos** al momento de que el procedimiento sea ejecutado. Por esta razón, las palabras "parámetro" y "argumento", muchas veces son utilizadas en programación como sinónimos.

Bloques de Procedimiento "Result"

Los bloques de procedimiento del tipo **result** se utilizan de manera similar a los del tipo **do**, con la diferencia de que luego de ser ejecutados, nos devolverán un valor como resultado. Dicho resultado tomará el valor de la secuencia de bloques que haya sido asignada a su encastre "**result**".

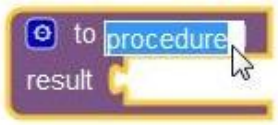


Una vez definida, podremos ejecutar dicha secuencia de bloques todas las veces que sea necesario, utilizando para ello su respectivo bloque **call** o de llamada al procedimiento.



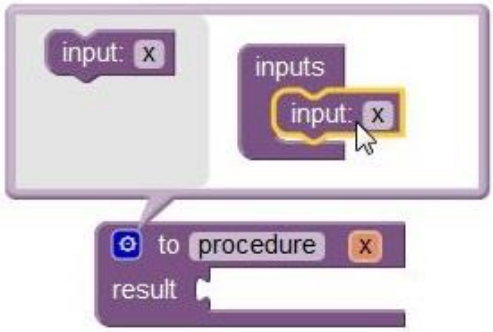
Después de la creación de este procedimiento, **App Inventor** genera automáticamente un bloqueo del tipo **call_procedure** y lo coloca dentro el menú **Procedures** de la columna **Blocks**. Asignaremos dicho bloque donde deseemos ejecutar y copiar el resultado del nuestro procedimiento.

Además, al crear un nuevo procedimiento, **App Inventor** define automáticamente el nombre del mismo. Podremos modificar este nombre haciendo clic dentro del cuadro que lo contiene.

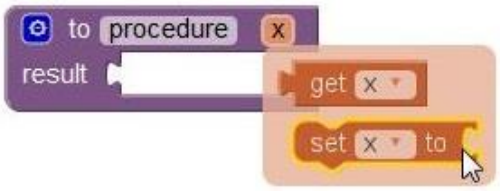


Como sucede con los bloques de procedimiento del tipo **do**, el nombre de cada procedimiento es único, y lo podremos modificar en cualquier momento. Al hacerlo, también cambiarán los nombres de los bloques asociados.

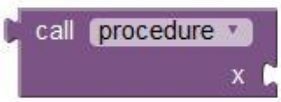
A su vez, también este tipo de bloques pertenece a la categoría **mutator**, de modo que podremos agregarles **argumentos**, de manera similar a los del tipo **do**.



Luego, podremos utilizar estos como lo hiciéramos en los bloques de control de eventos.



A su vez, al definir estos **argumentos**, los bloques **call** asociados serán modificados para incluir en ellos los encastres de entrada a los **parámetros**, con los cuales se inicializaran los **argumentos**.



Programación de la App

Definiendo el estado inicial

Vamos a comenzar creando las **variables globales** que necesitamos. Estas serán **Estado**, **Nro1**, **Nro2**, **Op** y **Acumulador**. Como puede verse a continuación, inicializaremos las mismas con un valor igual a '**0**', a excepción de **Op**, a la cual le hemos asignado el carácter "**=**", a modo de **operador neutro**.

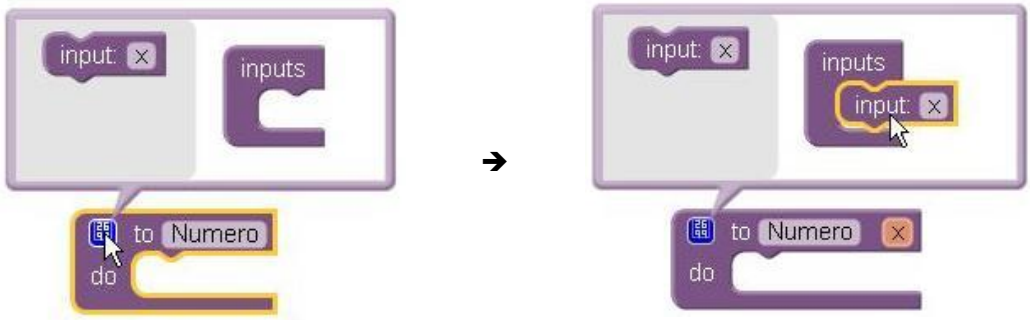
Estados, como su nombre lo indica, será la variable que por medio de su **valor** indicará en cuál de los estados se encuentra el programa. **Nro1** y **Nro2** guardaran respectivamente al **primer** y al **segundo** operando, hasta el momento en que se realice la operación. **Op** indicará la operación a realizar, mientas que **Acumulador** guardará el resultado.

Definiendo los Procedimientos

Para el caso de la presente **App**, vamos a utilizar tres **procedimientos** del tipo **do**. Comenzaremos por definir el primero, al que renombraremos como "**Calculo**":

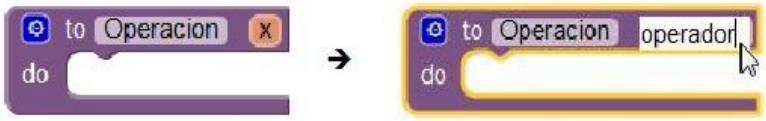
Como su nombre lo indica, este será el procedimiento dentro del cual se llevaran a cabo los cálculos de las operaciones deseadas.

Definimos ahora un segundo procedimiento, al que vamos a renombrar como "**Numero**". Utilizaremos este para ingresar los operandos, por lo que vamos a agregar un **argumento**, con el cual vamos a recibir a cada una de las cifras que formarán a dichos operandos:



Renombraremos este **argumento** como "**valor**", y su función será la de recibir la cifra que se está ingresando en ese momento.

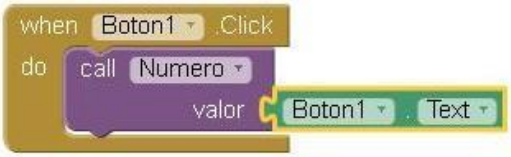
Por último, definimos un procedimiento que llamaremos "**Operación**", el cual vamos a utilizar para recibir el operador a utilizar dentro de "**Calculo**", por lo que le agregaremos un argumento de nombre "**operador**".



Asignando un Parámetro a un Procedimiento

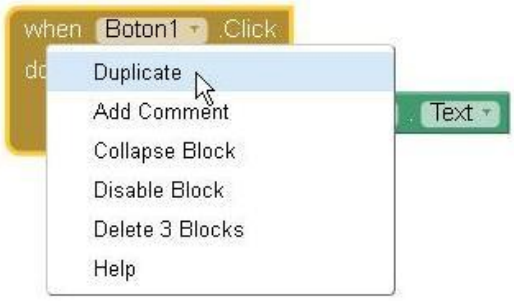
Vamos a analizar ahora cual será el modo en que utilizaremos los llamados a los procedimientos. Comenzaremos por ejemplificar esto para el procedimiento **Número**. Una vez que comprendamos como y donde realizaremos los llamados a este, pasaremos a explicar cuál será la secuencia de bloques a ejecutar.

Para esto, colocaremos el primero de varios bloques del tipo **when_Boton….Click_do**, por ejemplo el del **Boton1**. Dentro de este colocamos el bloque **call_Numero_valor**, que parecerá dentro del submenú **Procedures** luego de que lo hayamos creado.



Como asignación para la entrada "**valor**" hemos colocado el bloque **Boton1.Text**, que devuelve el texto que se lee dentro del dicho botón, o sea, el carácter '**1**'. Repetiremos esto para los restante números del **2** al **9** (el cero y el punto serán casos particulares).

Podremos hacer esto duplicando el bloque **when_Boton1.Click** ya completo. Para ello, haremos clic derecho sobre él mismo, para luego cliquear en la opción **Duplicate** del menú desplegable.



Veremos que nos aparecerá un nuevo conjunto de bloques idéntico al anterior. En ambos conjuntos de bloques nos aparecerá un símbolo de error. Al cliquear sobre este, se nos informará que tenemos más de uno de estos bloques controladores de evento, para el mismo componente.

Modificaremos este nuevo conjunto de bloques cliqueando sobre la leyenda **Boton1**, tanto del bloque **when_Boton1.Click**, como de **Boton1.Text**. Se nos desplegarán sendos menús, en los que podremos seleccionar cualquiera de los otros botones; para el caso de este procedimiento, utilizaremos los botones numéricos.

Como mencionamos antes, haremos esto mismo para los restantes botones numéricos, excluyendo el **0**. Al hacer esto, logramos que un mismo conjunto de bloques (los del procedimiento "**Numero**"), sea ejecutado en distintos momentos, para distintos eventos, y con un **valor** distinto en cada oportunidad, lo que simplificará por mucho nuestra programación.

If para Selección Múltiple – Procedimiento Numero

Vamos a programar ahora la secuencia de bloques de nuestro procedimiento "**Numero**". Colocaremos dentro de este un bloque **if_then**, el cual vamos a utilizar para determinar en cuál de los estados nos encontramos.

Colocaremos pues como primera condición de este **if**, a la **comparación matemática** entre la variable **Estado** y el número '**0**'.

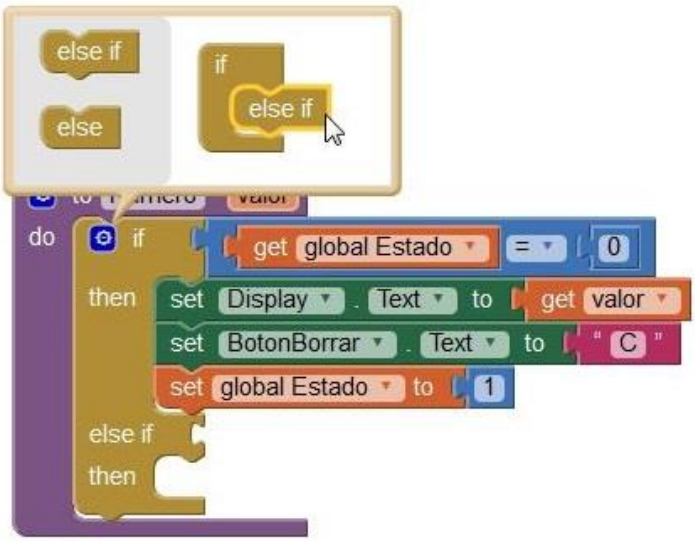
Si dicha condición se cumple, significa que nos encontramos en el estado inicial, por lo que comenzaremos a ingresar el primer operando. Dicho operando lo formaremos colocando las cifras ingresadas **una a una** dentro de nuestro **Display**.

Para este primer estado, vamos a asignar la cifra indicada por **valor** directamente al bloque **set_Display.Text**, ya que nuestro **Display** se encontrará inicialmente en cero.

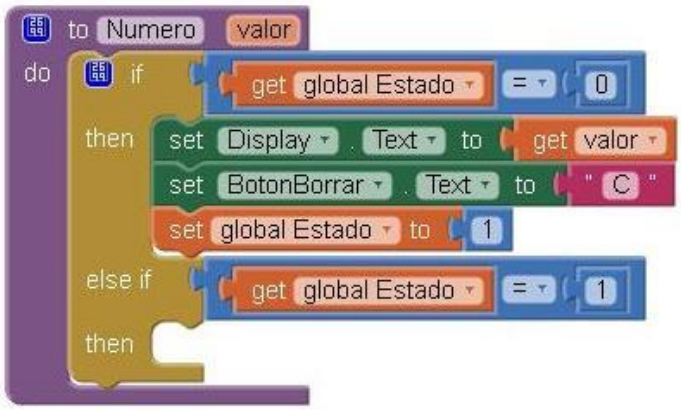
Además de hacer esto, será necesario establecer la variable **Estado** en '**1**', ya que es el estado que definimos para el ingreso del primer operando. De esta forma, la próxima vez que se llame al procedimiento **Numero**, sabremos que el argumento **valor** contienen una nueva cifra.

Vamos a cambiar además, la leyenda del **BotonBorrar** a "***C***" (**C**lear), lo que le indicará al usuario que de presionar dicho botón, sólo se borrara lo último ingresado (programaremos esto más adelante, dentro del evento **when_BotonBorrar.Click**)

Procedemos ahora a ampliar nuestro **if_then** para que admita una segunda condición, la cual se comprobará **sólo** en el caso de que no se cumpla la primera (**Estado=0**). Para esto, cliquearemos dentro del pequeño **cuadrado azul** que se encuentra en su esquina superior izquierda, para luego agregarle una entrada del tipo **else_if**, de la siguiente forma:



Veremos que nos aparecerá tanto un nuevo encastre **if** (marcado como **else_if**), como un nuevo espacio de tareas **then**, debajo del anteriormente utilizado. Una vez agregado el nuevo encaste **else_if**, colocaremos como condición del mismo la comparación entre la variable **Estado** y el número '**1**'.



Como hemos mencionado, esta nueva condición se evaluará **si y sólo si** la primera condición resulta falsa, la cual se volverá a evaluar recién cuando el procedimiento **Numero** vuelva a ser llamado, y no inmediatamente después del cambio del valor de la variable **Estado**. Por ello decimos que nuestro **if** nos permitirá realizar una **selección múltiple**; en este caso, elegir que acción realizar según el **estado** del programa.

Supongamos ahora que el procedimiento **Numero** vuelve a ser ejecutado, y que la variable **Estado** es igual a '**1**', de modo que el primer espacio **then** resulta ignorado. En este caso, la segunda condición del **if** resultará verdadera, lo que implicará que una nueva cifra ha sido ingresada, por lo que necesitamos unirla a las anteriores para ir formando nuestro primer operando.

Así pues, colocaremos dentro de nuestro nuevo espacio **then**, un nuevo bloque **set_Display.Text**, al que le vamos a asignar la unión entre su texto actual y la nueva cifra. Para esto usaremos el bloque **join** (unir), el cual encontraremos dentro del submenú **Text**.

No cambiaremos aquí de **Estado** (o sea, no cambiamos el valor de dicha variable), debido a que necesitamos permanecer en el mismo hasta que se presione alguna de las teclas de operación disponibles.

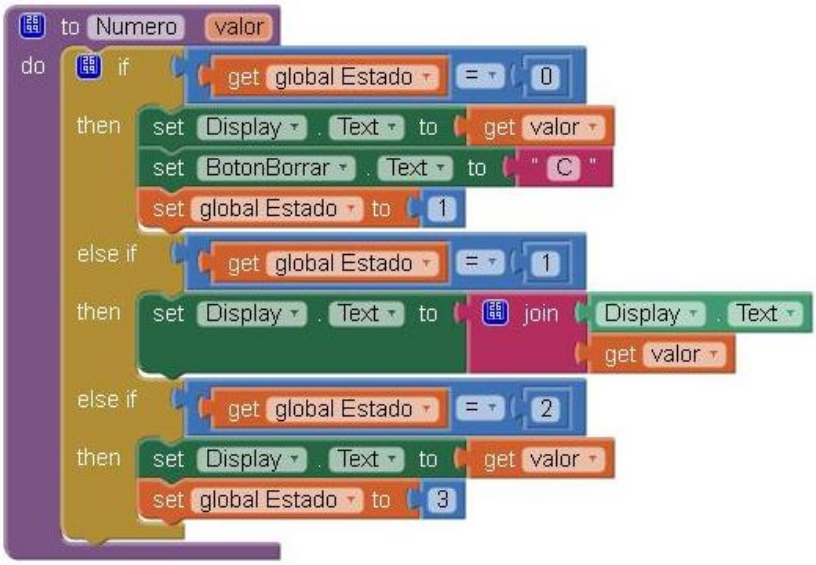
Continuamos ampliando nuestro **if** con el agregado de un nuevo encaste **else_if**, al cual vamos a asignarle la comparación entre **Estado** y el número **dos**. Si esta condición resulta verdadera, quiere decir que tanto el operador como el primer operando ya habrán sido ingresados, por lo que comenzaremos a ingresar aquí al segundo operando.

En este punto valdría preguntarse ¿En qué momento ha sucedido esto? ¿Cómo puede cumplirse que nos encontremos en el **Estado 2**, y que tanto el operador como el primer operando hayan sido ingresados, siendo que todavía no hemos programado tales acciones?

Haremos esto debido a que programar muchas veces requiere que nos adelantemos a pasos posteriores. Aquí estamos suponiendo el caso de que se cumpla una condición que va a ser establecida en otra parte del código, pero que de todas formas afectará al comportamiento del actual procedimiento.

Dicho esto, suponemos que si estamos en el **Estado 2**, es porque efectivamente tanto el primer operando como el operador ya han sido ingresados, por lo que dentro del **argumento valor** tendremos a la primera cifra del segundo operando, de modo que se la asignaremos directamente al **Display**.

Luego, estableceremos el valor de la variable **Estado** en '**3**', indicando de este modo que nos encontramos ingresando del segundo operando.



Para el caso del **Estado 3**, tenemos que realizar una tarea similar a la del **Estado 1**, agregando el **valor** ingresado al **Display**, por lo que nuestro **if** quedará así:

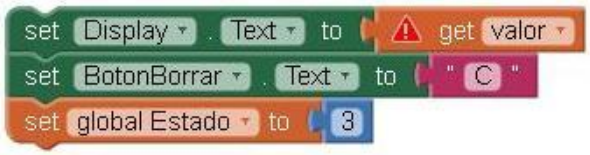
Recordemos que la comparación entre la variable **global Estado** y el número '**3**', no se llevará acabo sino hasta la próxima ejecución de **Número**, y sólo en caso de que las primeras dos condiciones de nuestro **if** no se cumplan.

Con respecto al **Estado 4** ("ya se ha efectuado un cálculo") y al **Estado 5** ("nuevo operador luego de un cálculo"), tendremos que efectuar acciones similares a las de los **Estados 2** y **3**, respectivamente. Por dicha razón, agregaremos sendos encastres **else_if**, a cuyos espacios **then** completaremos de la siguiente forma:

Para el caso del **Estado 6**, supondremos que se ha presionado el **BotonBorrar** una sola vez, por lo que si bien el segundo operando habrá sido borrado, contaremos todavía con el primer operando y con el operador.

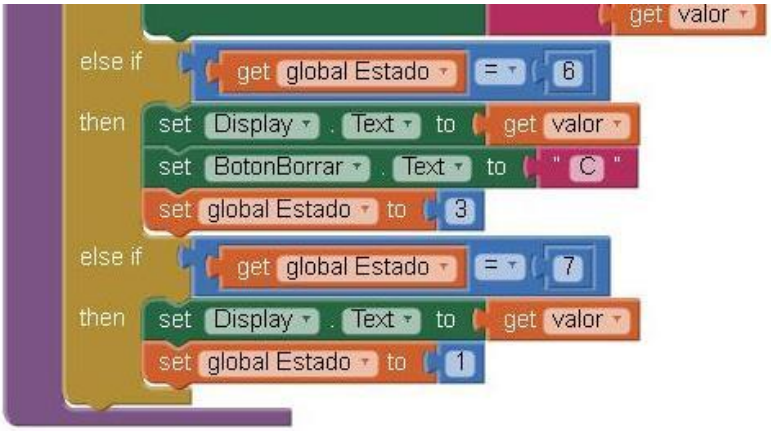
Adicionalmente, la leyenda de dicho botón será igual a **"CE"**, por lo que las acciones a realizar aquí serán:

- El ingreso de un nuevo operando dentro del **Display**
- El cambio de la leyenda del **BotonBorrar**, de **"CE"** a **"C"**
- El establecer la variable **Estado** en un valor igual a '**3**'

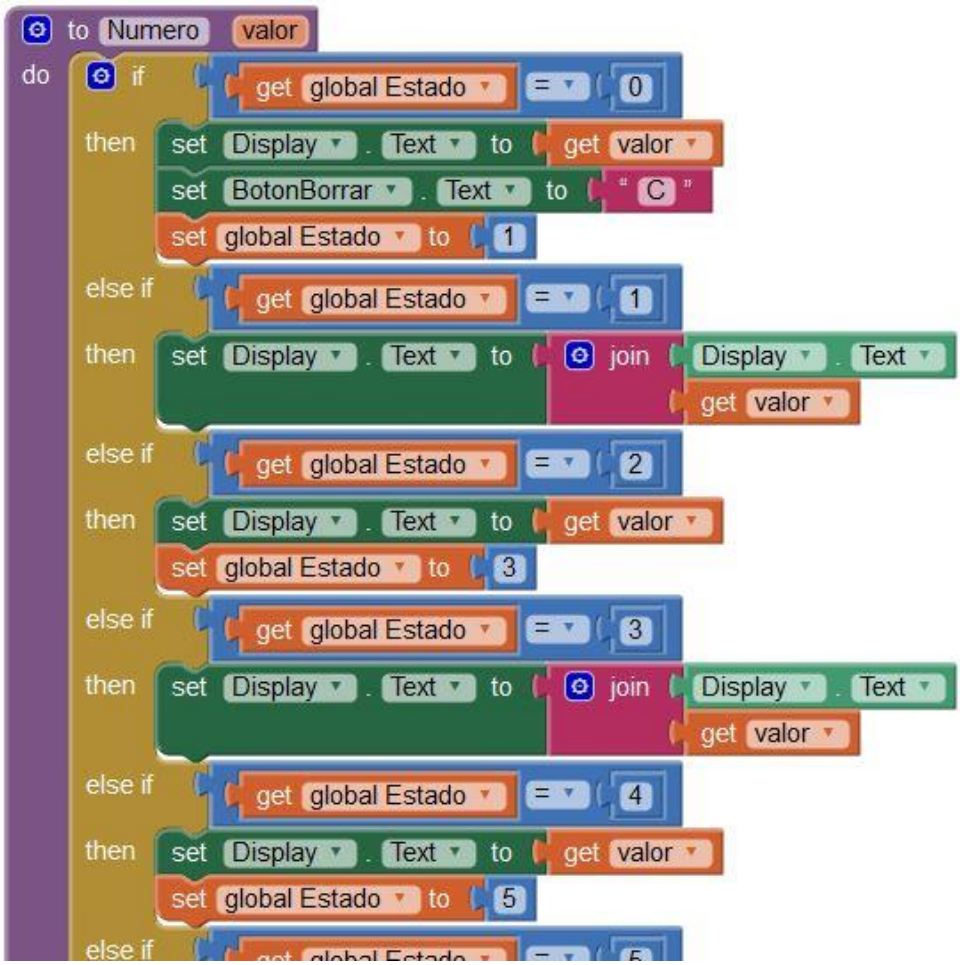


Para el caso del **Estado 7**, que es el estado de **error** de división por cero, sobre escribiremos la leyenda del **Display** (que será igual **error**) y pasaremos al **Estado 1**. Haremos esto para que el comportamiento de nuestra **App** sea similar al de cualquier calculadora.

Pueden apreciar lo descripto para estos dos últimos casos en la imagen a continuación:



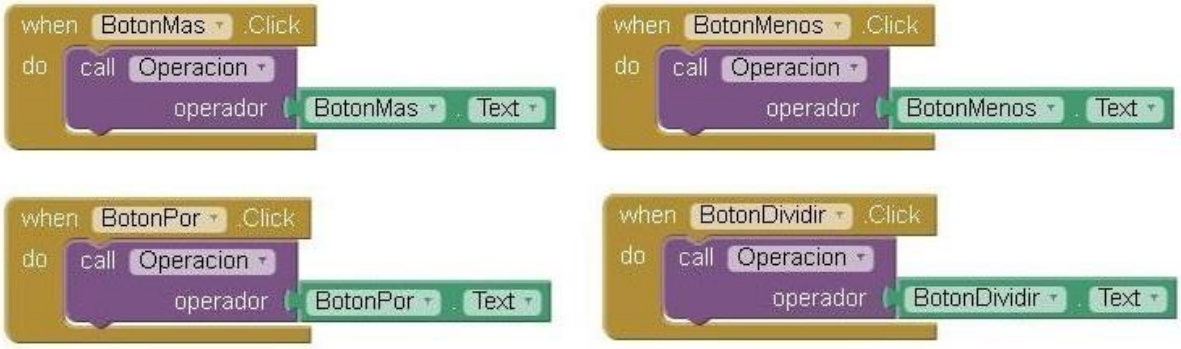
Y el procedimiento completo nos quedará de la siguiente forma:



Programación para las Teclas de Operación

Vamos a programar ahora lo que sucederá cuando se presione alguna de las teclas para los operadores. Utilizaremos para ello el procedimiento **Operación**, de manera similar a como lo hiciéramos con las teclas numéricas, asociando en este caso los operadores al **argumento operador**.

Para ello, vamos a colocar dentro de la ventana **Viewer**, los 4 controladores de evento del tipo **when_Boton….click** correspondientes a cada operación. Luego, colocaremos dentro de ellos la llamada al procedimiento **Operación**, asignándole a cada uno el texto correspondiente a cada botón como parámetro del encastre **operador**:



Pasemos ahora a realizar la programación de dicho procedimiento. Colocaremos dentro del mismo un nuevo **if_then**, el cual iremos ampliando de modo que podamos seleccionar cada uno de los **7 estados** posibles.

La programación que se detallará a continuación, se ha realizado partiendo del comportamiento observado al utilizar una aplicación "calculadora" estándar. Describiremos aquí los pasos que se deben realizar para que el comportamiento de nuestra **App** sea el mismo.

Dicho esto, podemos describir lo observado para el **Estado 0** de la siguiente forma:

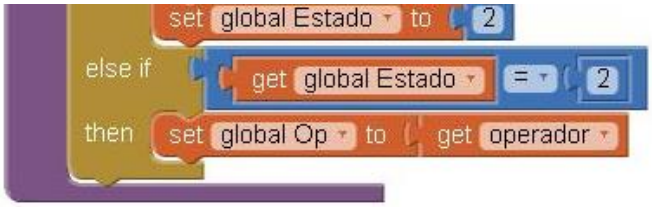
- Con la calculadora recién iniciada, presionamos un operador cualquiera.
- Al hacer esto, lo que hay en el **Display**, o sea '**0**', se guarda como primer operando (**Nro1=0**) .
- El operador es memorizado (**Op=operador**) .
- La leyenda del **BotonBorrar** cambia a **"C"**.
- La calculadora queda a la espera del segundo operando, lo que en nuestro caso es igual pasar al **Estado 2**.

Para el **Estado 1**, operaremos de manera similar:

- Luego de ingresar el primer operando, se presiona un operador cualquiera.
- Al hacer esto, lo que hay en el **Display**, se guarda como primer operando (**Nro1=Display.Text**) .
- El operador es memorizado (**Op=operador**) .
- La calculadora queda a la espera del segundo operando, lo que en nuestro caso es igual pasar al **Estado 2**.

A diferencia del caso anterior, no será necesario cambiar le leyenda del **BotonBorrar**, debido que esto habrá sucedido ya dentro del procedimiento **Numero**, justo antes de que **Estado** haya pasado a valer '**1**':

Para el caso del **Estado 2**, sólo tendremos que asignarle el valor de la variable **operador** a la variable **Op**, ya que el único modo de que **Estado** sea igual a '**2**' es debido a una anterior ejecución del procedimiento **Operacion**, lo que indica que usuario ya ha ingresado un **operador**, pero desea cambiarlo:



Como podemos apreciar, no hemos modificado aquí el valor de la variable **Estado**, permitiendo de este modo que el usuario pueda cambiar de operador todas las veces que necesite, ya que la secuencia a ejecutar será siempre la misma.

Para el caso del **Estado 3**, tendremos que:

- El usuario termino de ingresar el segundo operando, pero en lugar de apretar "igual", presionó alguna de las tecla de operación.
- El contenido del **Display** se convierte en el segundo operando, por lo que se lo asigna a la variable **Nro2**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** es asignado tanto al **Display** como a la variable **Nro1**, para que sirva de primer operando en la siguiente operación.
- El nuevo **operador** es asignado a la variable **Op**.
- **Estado** se establece en **2**, indicando de esta forma que un nuevo operador ha sido ingresado.

De encontrarnos en el **Estado 4**, significa que una operación ya se ha sido completada, y su resultado se encuentra en el **Acumulador**, por lo que:

- Utilizamos el resultado de la operación anterior como nuevo primer operando, por lo que asignaremos **Acumulador** a **Nro1**.
- Asignamos el nuevo **operador** a la variable **Op**.
- **Estado** se establece en **2**, indicando de esta forma que un nuevo operador ha sido ingresado.

En el caso del **Estado 5**, tenemos que:

- El usuario acaba de ingresar un nuevo primer operando, luego de haber completado una anterior operación. Luego, presionó alguna de las teclas de operación.
- El contenido del **Display** se convierte en el primer operando, por lo que se lo asigna a la variable **Nro1**.
- El nuevo **operador** es asignado a la variable **Op**.
- **Estado** se establece en **2**, indicando de esta forma que un nuevo operador ha sido ingresado.

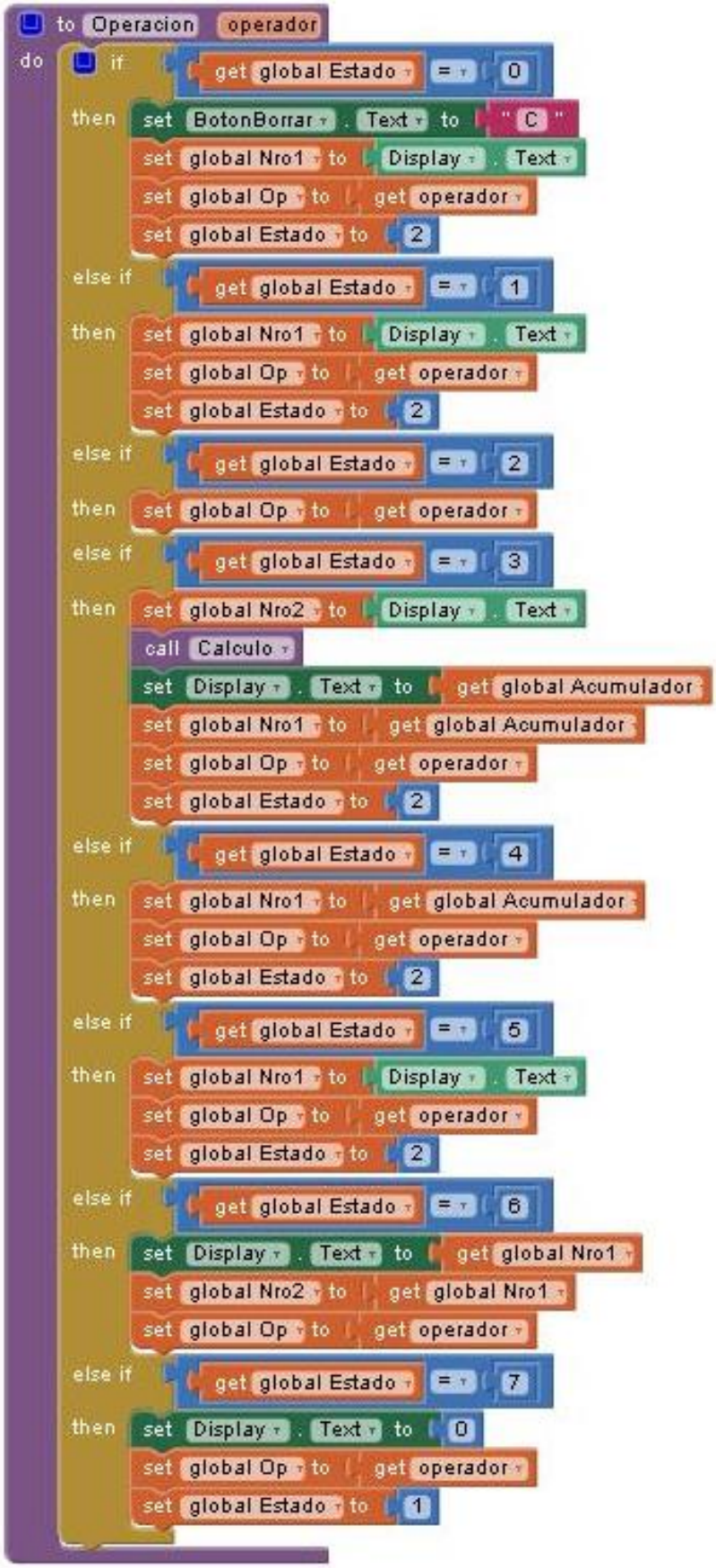
De encontrarnos en **Estado 6**, significa que se ha presionado el **BotonBorrar**, por lo que el **Display** será igual a '0'. En este caso tenemos que:

- Cuando el usuario presione alguna de las teclas de operación, el primer operando será asignado al **Display**.
- El segundo operando deberá ser igual al primero, por lo que asignaremos **Nro1** a **Nro2**.
- El nuevo **operador** es asignado a la variable **Op**.
- **Estado** se establece en **2**, indicando de esta forma que un nuevo operador ha sido ingresado.

Por último, de encontrarnos en el **Estado 7** significa que se produjo un error de división por cero, por lo que en el **Display** se lee "Error". En este caso tendremos que:

- Restablecemos el **Display** con un valor igual a '0'.
- El nuevo **operador** es asignado a la variable **Op**.
- **Estado** se establece en **1**, permitiendo de esta forma que un nuevo primer operando se ingresado.

El algoritmo completo para este procedimiento nos quedará de la siguiente forma:



Programación del Procedimiento de Calculo

Vamos a definir ahora la secuencia de bloques que vamos a utilizar dentro del procedimiento "**Calculo**". Nuevamente, vamos a utilizar aquí un bloque **if_then**, con el cual determinaremos cuál de las **4 operaciones aritméticas** es la que se necesita realizar.

Como primera condición de este **if**, colocamos una **comparación lógica** entre la variable **global Op** y el carácter **"+"**. De resultar verdadera, la operación a realizar será la "**suma**", por lo que dentro de su espacio de tareas, vamos a colocar un bloque **set_global_Acumulador_to**, al que le vamos a asignar el bloque matemático "**suma**" (submenú **Math**), el cual completaremos con las variable **Nro1** y **Nro2**.

Luego, modificaremos el bloque **if_then** para que admita una segunda condición. Completamos esta nueva condición y su espacio de tareas de manera similar al anterior, pero esta vez para el carácter **"-"** y el bloque matemático "**resta**", con lo que nos quedará de la siguiente forma:

Ahora haremos todo esto mismo nuevamente, para el carácter **"X"** y el bloque matemático "**multiplicación**", quedándonos de la siguiente forma:

Tengamos nuevamente presente aquí, que el término **else** quiere decir "**sino…**" o "**en caso de que no se cumpla…**", por lo que cada encastre **else_if** se evaluará **sólo** en el caso de que las condiciones en los encastres anteriores no hayan resultado verdaderas, por lo que si se cumple por ejemplo la igualdad de **Op** con **"+"**, no se evaluará ninguna de las posteriores condiciones (**"-"** y **"X"**).

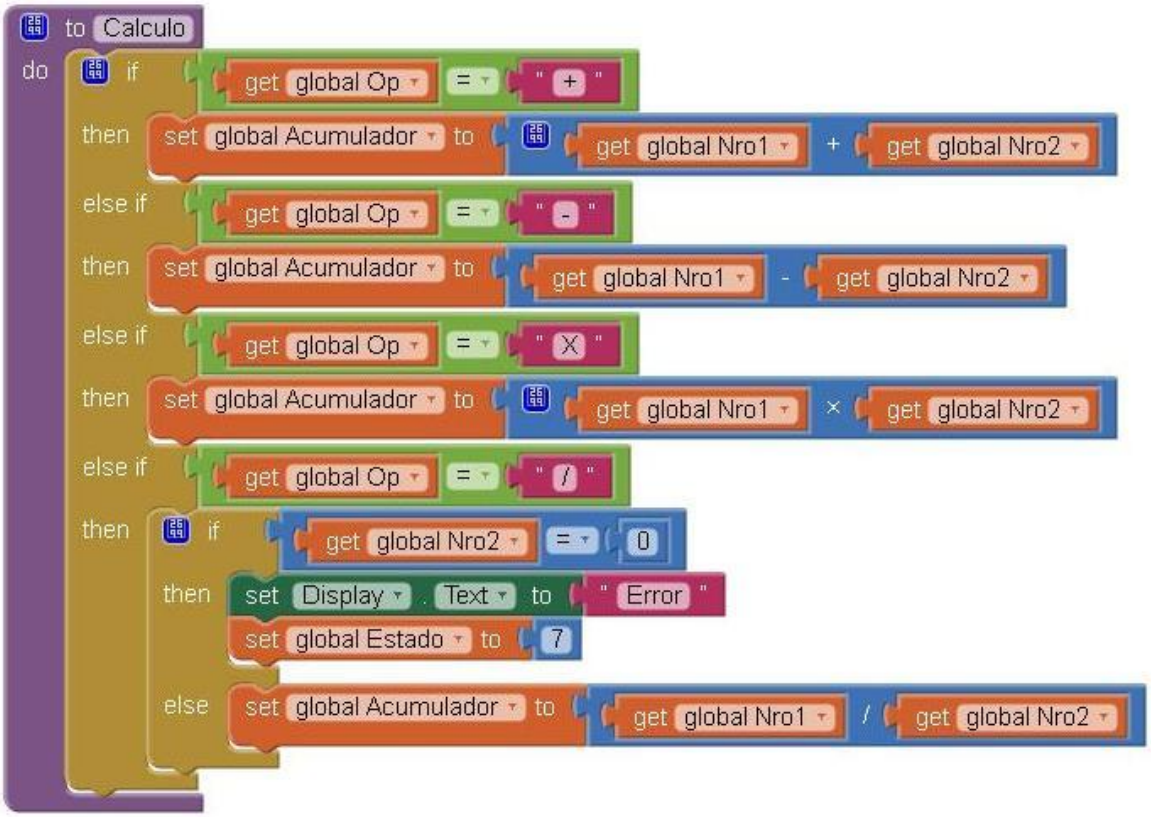
Ahora bien; para el caso de la división, será necesario evaluar previamente si el segundo de los operandos resulta o no igual a '**0**', ya que de ser así se establecerá la condición de **error**, cambiando además el valor de la variable **Estado** a '**7**', que es el estado que representa este error. Todo esto lo haremos con un nuevo **if_then_else**, cuya condición será una comparación del tipo matemática entre **Nro2** y '**0**'.

Si encontramos que se cumple la igualdad, informaremos al usuario de dicha condición de **error**, asignando el texto **"Error"** al **Display**; usaremos para esto el bloque **set_Display.Text_to** de la siguiente forma:

En caso contrario, podremos llevar a cabo la división, por cual colocamos dicha operación dentro del espacio de tareas **else** restante, asignando su resultado a la variable

Acumulador. Este último **if** lo colocamos a su vez dentro del anterior, agregando un nuevo encastre **else_if**, al que estableceremos como condición la comparación entre **Op** y el carácter **"/"**.

Todo esto en conjunto nos quedará de la siguiente forma:



Programación para el Botón Igual

La programación para este botón es bastante simple. Bastará con asignar los valores correspondientes a las variables **Nro1** y **Nro2** según el **estado actual**, para luego y en los casos que corresponda, realizar la operación deseada, por medio de la llamada al procedimiento **Calculo**. Aquí está el algoritmo:

Tanto para el **Estado 0**, como para el **Estado 1**, sucede lo mismo, o sea, no contamos con ningún operando, por lo que permaneceremos en el **Estado 0**:

Para el caso del **Estado 2**, sucederá que ya se habrán ingresado tanto el primer operando como el operador, por lo que tendremos que:

- Al presionar la tecla "igual", se llevará a cabo la operación, igualando el segundo operando al primero, por lo que **Nro1** es asignada a **Nro2**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** (**Acumulador**) se asigna tanto al **Display** como a la variable **Nro1**, para que sirva de primer operando en la siguiente operación.
- **Estado** se establece en **4**, indicando de esta forma que se acaba de completar una operación por medio de la tecla "igual".

De encontrarnos ahora en el **Estado 3**, significa que se está ingresando el segundo operando, por lo que tendremos que:

- Al presionar la tecla igual, se considera que el ingreso del segundo operando se ha completado, por lo que el contenido del **Display** es asignado a **Nro2**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** (**Acumulador**) es asignado al **Display**.
- **Estado** se establece en **4**, indicando de esta forma que se acaba de completar una operación por medio de la tecla "igual".

Si ahora, nos encontramos en el **Estado 4**, quiere decir que se acaba de completar una operación, por lo que tendremos que:

- Al presionar la tecla "igual", se repetirá la última operación, utilizando como primer operando el último resultado, por lo **Acumulador** es asignada a **Nro1**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** (**Acumulador**) es asignado al **Display**.
- No cambiaremos de **Estado**, el cual permanecerá igual a **4**, indicando de esta forma que se acaba de completar una nueva operación.

Para el **Estado 5**, en el cual se estará ingresando un nuevo operando, luego haber obtenido un resultado, tendremos que:

- Al presionar la tecla "igual", se repetirá la última operación, utilizando el nuevo número ingresado como primer operando, por lo **Display** será asignado a **Nro1**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** (**Acumulador**) es asignado al **Display**.
- **Estado** se establece en **4**, indicando de esta forma que se acaba de completar una nueva operación, por medio de la tecla "igual".

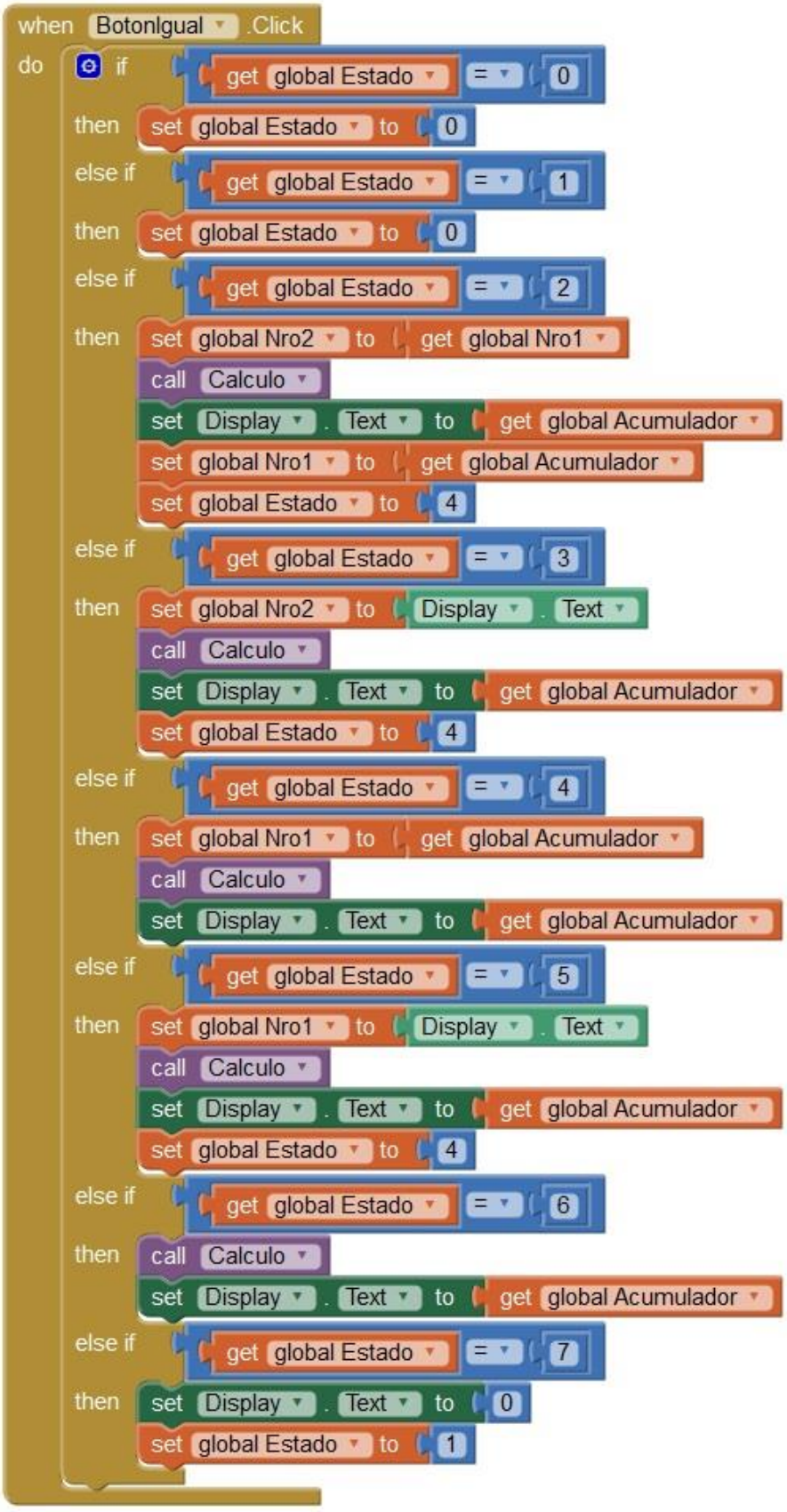
De encontrarnos en el **Estado 6**, quiere decir que el **BotonBorrar** ha sido presionado una sola vez, por lo que sólo se ha borrado lo último ingresando, de manera que todavía disponemos tanto de uno de los operandos, como del operador, por lo que:

- Al presionar la tecla "igual", se repetirá la última operación, sin modificar lo que sea que se encuentre en **Nro1** y **Nro2**.
- Se llama al procedimiento **Calculo** para obtener un resultado, el cual es almacenado por dicho procedimiento dentro de la variable **Acumulador**.
- El resultado de dicho **Calculo** (**Acumulador**) es asignado al **Display**.
- Permaneceremos en el **Estado 6**, aun cuando acabamos de completar un nuevo **Calculo**, por lo que la variable **Estado** no sufrirá cambios.

Por último, para el **Estado 7**, que es el estado de error, sólo tendremos que:

- Al presionar la tecla "igual", se sale del estado de error, restableciendo el **Display** a '0'.
- **Estado** se establece en **1**, indicando de esta forma que se puede comenzar a ingresar un nuevo primer operando.

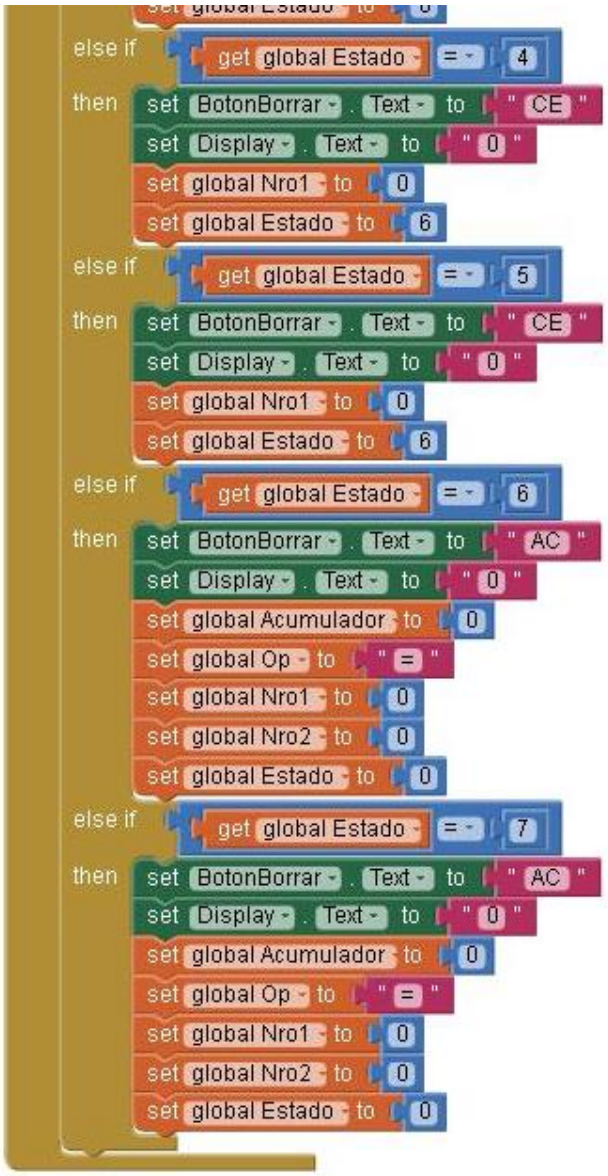
El algoritmo completo para este evento resulta de la siguiente forma:


when BotonIgual .Click
do if get global Estado = 0
then set global Estado to 0
else if get global Estado = 1
then set global Estado to 0
else if get global Estado = 2
then set global Nro2 to get global Nro1
call Calculo
set Display . Text to get global Acumulador
set global Nro1 to get global Acumulador
set global Estado to 4
else if get global Estado = 3
then set global Nro2 to Display . Text
call Calculo
set Display . Text to get global Acumulador
set global Estado to 4
else if get global Estado = 4
then set global Nro1 to get global Acumulador
call Calculo
set Display . Text to get global Acumulador
else if get global Estado = 5
then set global Nro1 to Display . Text
call Calculo
set Display . Text to get global Acumulador
set global Estado to 4
else if get global Estado = 6
then call Calculo
set Display . Text to get global Acumulador
else if get global Estado = 7
then set Display . Text to 0
set global Estado to 1

Programación para el BotónBorrar

La función de este botón será la volver atrás la última operación o la de limpiar todo, dependiendo del **estado** en que se encuentre el programa cuando este es presionado.

Por esta razón, para los **estados** que van del **2** al **5**, se procederá a borrar sólo lo último ingresado, estableciendo la leyenda del mismo como **"CE"** (**C**lear to **E**mpty; Borrar hasta vaciar), y pasando al **Estado 6**, que es el estado previo al borrado total. Para los **Estados 0**, **1**, **6** y **7**, se restablecerán los valores de **todas** las variables, retornando al estado de **inicio** (**Estado 0**):

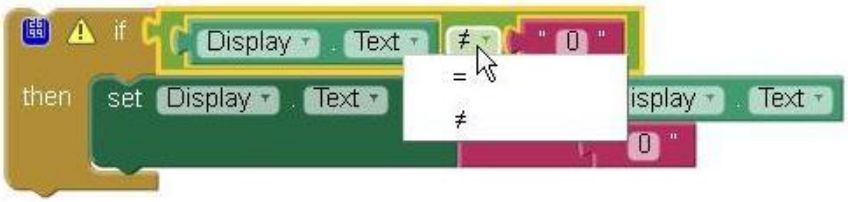


Casos particulares

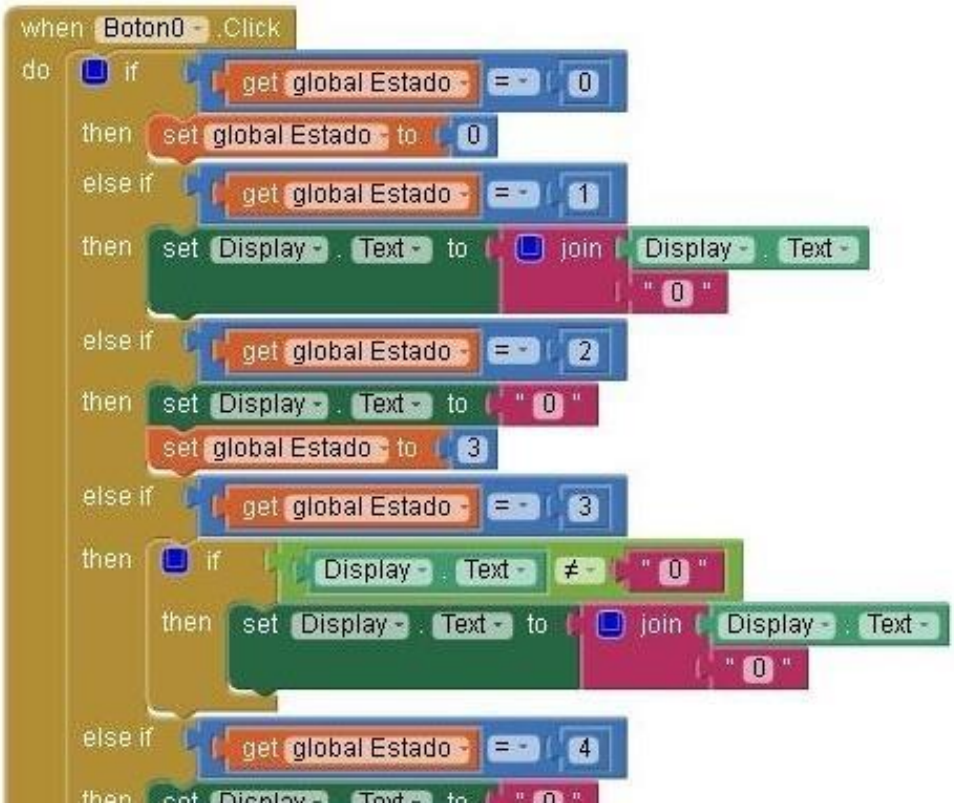
Programación para el Botón Cero

En el caso del botón '0', vamos a utilizar prácticamente el mismo algoritmo que utilizamos para el procedimiento **Numero**, con le salvedad de que **sólo** tendremos que modificar la etiqueta **Display** cuando esta sea distinta de '0' (no tiene sentido agregar ceros a la derecha de otro cero).

Por esta razón, dentro de los **Estados 3** y **5**, necesitaremos agregar unos bloques **if_then**, en los cuales utilizaremos como condición el bloque **lógico "desigual"**, el cual se obtiene a partir de un bloque **lógico "igual"**, cliqueando sobre el cuadrito "=", el cual nos desplegará un pequeño menú, como podemos ver en la imagen a continuación:



Seleccionamos la segunda opción y acomodamos los bloques de manera que nos quedará algo como esto:



Programación para Botón Punto

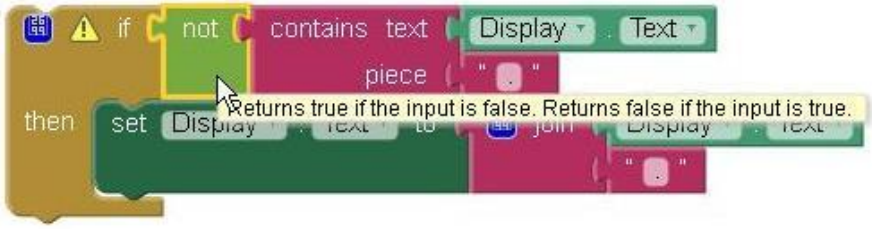
De manera similar al caso del botón cero, utilizaremos para el **BotónPunto** un algoritmo parecido al del procedimiento **Numero**, pero con ciertas modificaciones para los **Estados 1**, **3** y **5**.

La idea es utilizar el punto como separador decimal, por lo cual si el usuario presiona sobre dicho botón, será necesario que agreguemos un "**.**" (punto) a continuación de las cifras ya ingresadas dentro del "**Display**", siempre y cuando este no haya sido ingresado con anterioridad.

Para determinar si nuestro **Display** contiene o no un "**.**", necesitamos utilizar el bloque **conteins_text_piece**, el cual sirve para establecer si dentro de una determinada cadena de texto, se encuentra o no una determinada pieza texto.

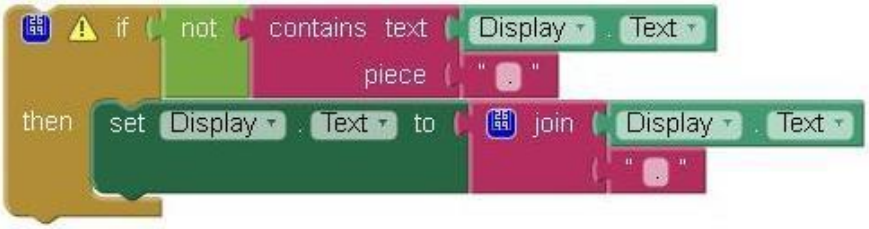
En caso de que el texto asignado al encastre **piece**, este contenido dentro de la cadena asignada a **text**, el bloque **conteins** nos devolverá un valor de verdad igual a **true**.

Ahora, lo que necesitaremos será "preguntar por negativo", anteponiendo para ello el bloque **lógico not**, para luego utilizarlo como parte de la condición de un nuevo bloque **if** de la siguiente forma:

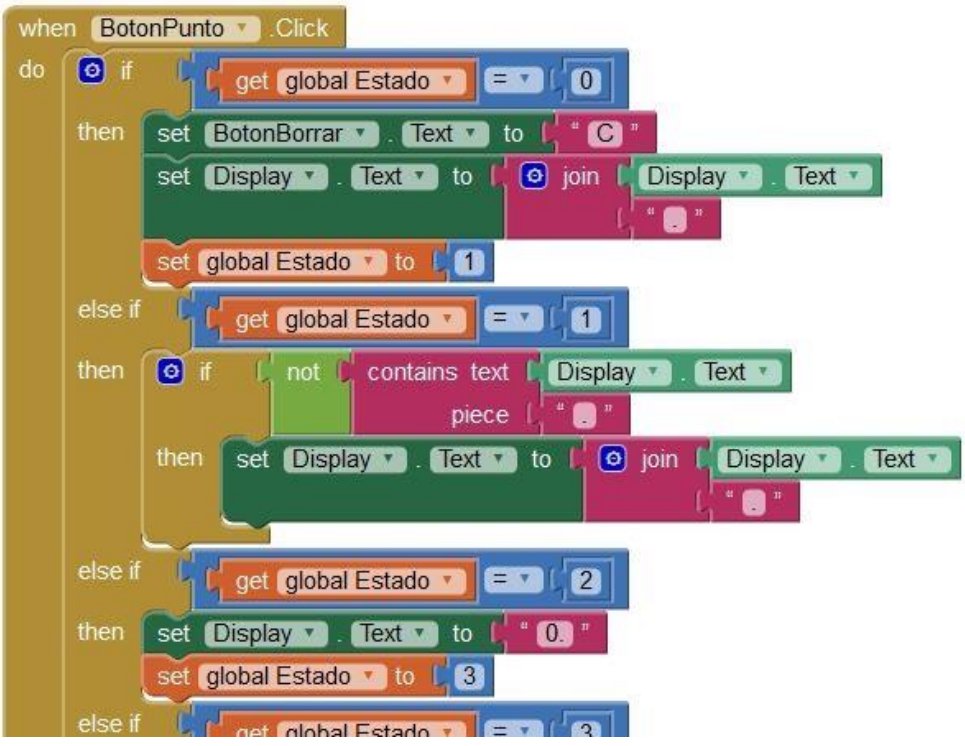


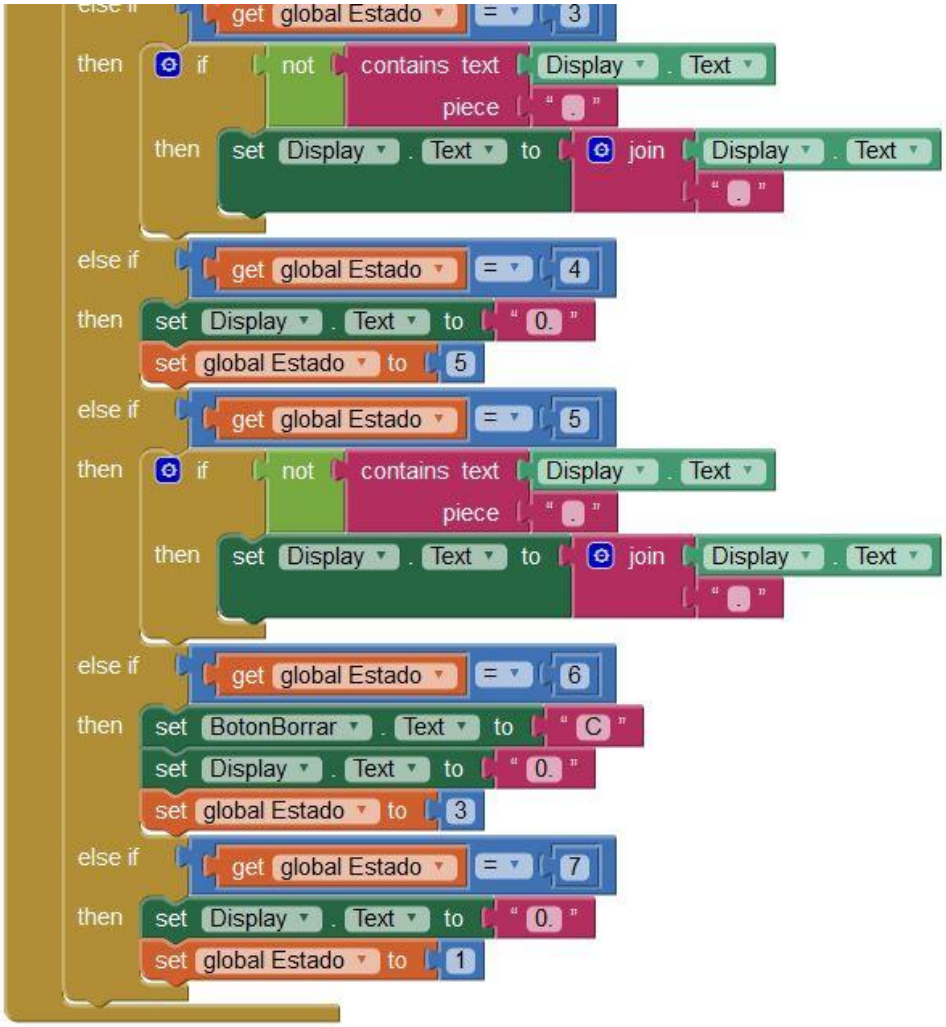
Como dice el cuadro de ayuda desplegado sobre el cursor en la imagen, el bloque **not** "devuelve un valor **verdadero** si a la entrada del mismo se ingresa un valor **falso**, y devuelve un valor **falso** si a la entrada del mismo se ingresa un valor **verdadero**".

Con esto se logra hacer que "**si el texto del Display no contiene "." (un punto), entonces este sea agregado al final**".



El algoritmo completo para este botón nos quedaría así:




get global Estado = 3
then if not contains text Display . Text
piece " . "
then set Display . Text to join Display . Text
" . "
else if get global Estado = 4
then set Display . Text to " 0. "
set global Estado to 5
else if get global Estado = 5
then if not contains text Display . Text
piece " . "
then set Display . Text to join Display . Text
" . "
else if get global Estado = 6
then set BotonBorrar . Text to " C "
set Display . Text to " 0. "
set global Estado to 3
else if get global Estado = 7
then set Display . Text to " 0. "
set global Estado to 1