SAMSUNG

SAMSUNG

CLUB de APPS

Programa tus Ideas

TUTORIAL DIARIO

DÍA 2

Inria Chile

Introducción

Hola Profesor! Bienvenido al segundo día del taller **Programa Tus Ideas** :)

El día de hoy comenzaremos a trabajar con el componente **Lienzo**, que te permite dibujar líneas y círculos, además de poder mostrar imágenes y animaciones. También conocerás y trabajarás con otros dos nuevos elementos que son de suma importancia a la hora de crear aplicaciones: los *procedimientos* y los *ciclos*. Para entender su funcionamiento, aprenderás sobre los *algoritmos*, los cuales están en el corazón de la informática y la ciencia de la computación. Los algoritmos son las instrucciones o recetas de cocina con la que tú como programador le dices al computador que tiene que hacer. Para comprender su funcionamiento aprenderás a realizar **Diagramas de Flujo**, que son representaciones gráficas que te permitirán definir los distintos pasos que estarán presentes en tu aplicación. Finalmente llevarás tus diagramas de flujo a una aplicación en **App Inventor** conocida como *Logo, ¡De esa forma podrás programar !*.

1. Tutorial: Algoritmos de Dibujos Simples

Un algoritmo es una secuencia precisa de instrucciones que controla el comportamiento del computador. En este tutorial aprenderás a dibujar figuras simples usando una plataforma similar a la tortuga *Logo*. *Logo* es un lenguaje de programación desarrollado en la década de 1960 enfocado principalmente para uso educacional. Es una excelente plataforma para crear algoritmos de dibujo simples. Con ella vas a conocer los pasos necesarios que involucran a toda creación de aplicaciones.

Un paso muy importante que deberás hacer siempre antes de comenzar a desarrollar tus aplicaciones, es el darte un tiempo para comprender la totalidad de los elementos que estarán presentes en el desarrollo para luego analizar la forma en que estos elementos estarán interactuando entre si, o bien para establecer cómo lo harán cuando estén interactuando con los usuarios. Sentarse a planificar y proyectar como se verá la aplicación una vez terminada permite optimizar de muy buena forma el proceso completo de desarrollo. Si no hacemos el ejercicio de planificar el desarrollo, es muy probable que vayan surgiendo inconvenientes en el camino que de seguro atrasarán todo el desarrollo. ¿Se imaginan qué sería de la construcción de un edificio si no se planificara con anterioridad cada una de las etapas que comprenderá la puesta en marcha del proyecto? Lo mismo ocurre con el desarrollo de aplicaciones. Es necesario darse el tiempo para planificar la forma en la cual se va a desarrollar la aplicación, estableciendo cada una de los componentes que se necesitarán y por sobre todo, entender la lógica que tendrá la aplicación; establecer cada uno de los eventos con su correspondiente acción es una práctica que te ayudará bastante.

Existe una herramienta que nos permite establecer el funcionamiento de una aplicación mediante una representación gráfica a través de diagramas; podemos señalar mediante esquemas los algoritmos que serán utilizados en la aplicación. A esta herramienta se le conoce con el nombre de *diagramas de flujo*. Un diagrama de flujo nos ayuda a representar el flujo que tendrá una aplicación mediante elementos gráficos de tal manera que es posible visualizar y entender la lógica que hay detrás de la aplicación. Por más que agreguemos elementos gráficos a a nuestra aplicación, la lógica que hay detrás, la programación, es la que permite que la

aplicación funcione. Realizando este paso antes de comenzar a desarrollar alguna aplicación verán como se les hará mucho más directa y clara la programación mediante App Inventor. A continuación mostraremos un ejemplo sencillo de cómo sería realizar un diagrama de flujo para representar la lógica que existe en un ejercicio tan cotidiano como reemplazar la ampolleta de una lámpara si es que esta estuviese mala.

El primer paso será entender el problema e identificar todos los elementos que se encuentran presentes. Buscaremos la solución y definiremos qué es lo que queremos hacer si el problema se resuelve. Definiremos también qué es lo que ocurre si es que no hay solución. Definir qué hacer en ambos casos es algo que tendrás que realizar de manera frecuente cuando estés desarrollando tus aplicaciones.

Lo primero es establecer el problema, que en este caso es: *la lámpara no funciona*. Teniendo esto en mente, analizaremos paso a paso las posibles razones por las cuales nuestra lámpara no prende y estableceremos qué hacer en cada caso:

- Una razón por la cual nuestra lámpara no enciende podría ser que ésta no estuviese enchufada. En este caso la solución es sencilla ya que bastaría con conectarla a la corriente y ver si enciende o no. Cuando desarrollamos aplicaciones, nunca debemos descartar opciones por muy obvias que estas pudiesen ser. ¡Nunca sabes qué es lo que podría hacer un usuario!
- Podría darse el caso en que la lámpara siempre estuvo enchufada a la corriente y que aún así no encendiera. En este caso, podríamos pensar que el problema no viene dado por la corriente sino que sea por algún desperfecto en la ampolleta. Lo lógico aquí sería revisar el estado de la ampolleta. Preguntaremos (revisaremos) si es que la ampolleta está quemada o no. Si es que logramos determinar que la ampolleta si está quemada, vamos a proceder a cambiarla. Con esto notaríamos que la lámpara vuelve a encender y nuestro problema inicial estaría resuelto. Si es que sigue sin encender, procederemos a revisar nuestra ampolleta y si logramos determinar que ésta se encuentra en perfectas condiciones, quiere decir que el problema es originado por otra causa y en ese caso, es mejor cambiar la lámpara. Para entender este ejercicio hemos limitado las consecuencias del desperfecto a sólo 2 posibles razones. Pueden haber sido muchos otros los problemas

que originaron el desperfecto, sin embargo, nos quedaremos con estos dos para ejemplificar la lógica detrás del problema.

El proceso que acabamos de hacer, puede ser establecido de manera gráfica mediante diagramas, tal cual se ve en la Figura 1.1.

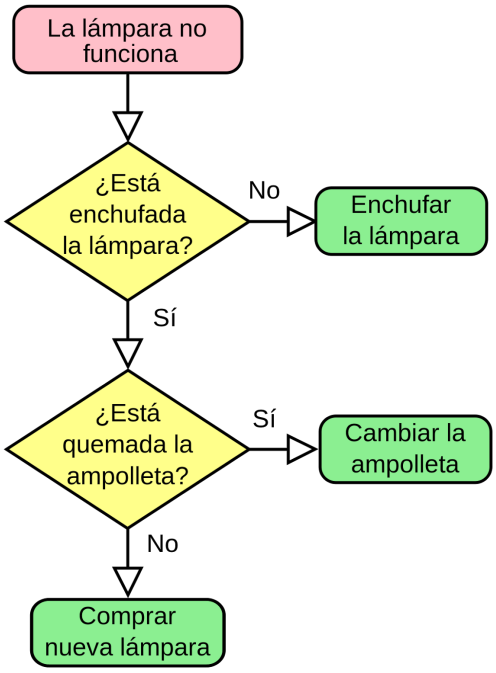


Figura 1.1: Diagrama de flujo con los pasos a seguir si es que una lámpara no funciona

Existe una simbología definida que nos permitirá entender y realizar de mejor forma nuestros diagramas de flujo. A continuación se mencionan los componentes principales y una explicación de su uso:

- El *óvalo* o *elipse* se utiliza para indicar el inicio y fin del diagrama. En el caso anterior viene a ser el problema inicial y cada una de sus posibles soluciones. Un diagrama de flujo siempre debería comenzar con un óvalo.
- El *rectángulo* representa una acción específica. Generalmente los procedimientos o actividades que pudiese tener una aplicación serán simbolizados por un rectángulo. Vamos a representar de esta manera todas las respuestas a los eventos gatillados por la acción de los usuarios.

- El *rombo* se utiliza para establecer una condición. Esta estructura permite cambiar el flujo de la aplicación según se cumpla o no una condición. En el ejemplo anterior, necesitábamos definir ciertas preguntas para ejecutar distintas acciones, según el estado que tuviese la lámpara, es lo que ocurre por ejemplo cuando preguntamos *"si es que está quemada la ampolleta, entonces la cambiamos"*. Con el paso de los días veremos que las condiciones cumplen un rol súper importante en la creación de cualquier aplicación o desarrollo de programas.
- El *romboide* permitirá representar el ingreso de información al flujo de aplicación. Por ejemplo, si queremos diagramar el inicio de sesión de facebook, necesitaremos un romboide para señalar el ingreso del usuario y la contraseña.

Si bien existen más que figuras para representar otro tipo de comportamientos, las cuatro que fueron señaladas anteriormente son las más comunes y las que utilizaremos con mayor frecuencia para describir un sin número de aplicaciones. Es muy importante que antes de comenzar a desarrollar tu aplicación te des el tiempo para desarrollar el diagrama de flujo. Como acabamos de ver, no es más que establecer la lógica de la aplicación mediante diagramas. A partir de hoy, antes de comenzar a desarrollar nuestras aplicaciones, dedicaremos un par de minutos para que juntos desarrollemos el diagrama de los proyectos que revisaremos cada día en el Taller.

1.1. Qué Aprenderás

Habiendo introducido los diagramas de flujos, veamos que nos toca aprender hoy en este tutorial:

- Usar comandos de *Logo* para dibujar figuras simples.
- A definir procedimientos simples para simplificar el proceso de dibujo.
- A definir ciclos que permiten optimizar tareas repetitivas

Para comenzar, descarga el proyecto ***Logo1*** desde `ProgramaTusIdeas/dia2` e impórtalo en **App Inventor**. Si no sabes cómo importar un proyecto pídele ayuda a tu tutor. Este

proyecto sirve como una plantilla ya que tiene una aplicación funcionando la que te permite usar comandos de dibujo Logo. La Figura 1.2 muestra la interfaz de usuario de la aplicación.

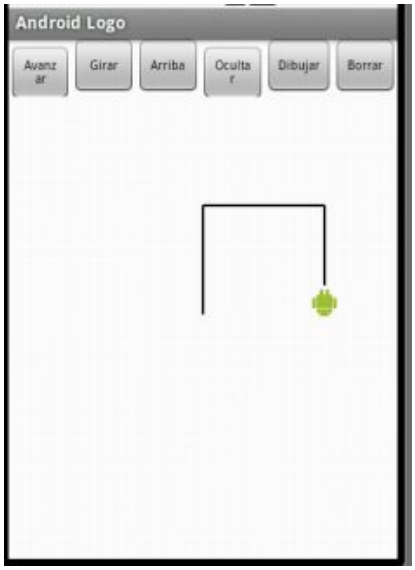


Figura 1.2: Interfaz de usuario Android Logo

Los comandos implementados en los botones son:

- *Avanzar*: hace que el Androide avance 10 pixeles.
- *Girar*: hace que el Androide gire 90° a la derecha.
- *Arriba/Abajo*: un interruptor que levanta el lápiz desde el lienzo de dibujo (pasa a modo "no dibujar"), o bien pone el lápiz sobre el lienzo (modo "dibujar").
- *Ocultar/Mostrar*: un interruptor que hace que el Androide desaparezca y aparezca.
- *Dibujar*: este botón ejecutar cualquier algoritmo que pongas en el procedimiento **dibujar** como parte de los ejercicios de este tutorial.
- *Borrar*: este botón borra el lienzo y pone al Androide de vuelta al centro del lienzo.

¿Qué es Logo? Logo es un lenguaje de programación inventado por *Seymour Papert* en la década de 1960, principalmente para usos educativos. Papert sostenía que los estudiantes aprenden mejor cuando están construyendo su propio conocimiento e ideas. El enfoque usado

por Papert se conoce como *aprendizaje constructivista*, el cual se inspira en la visión de que los individuos construyen modelos mentales para entender el mundo alrededor de ellos. Sin embargo, el constructivismo sostiene que el aprendizaje ocurre con mayor efectividad cuando las personas están activas construyendo *objetos tangibles en el mundo real*. En este tutorial, los *objetos tangibles* que construirás son los *algoritmos* para dibujar figuras simples.

La característica más conocida de *Logo* es su tortuga —realmente, un dibujo de una tortuga— que el usuario puede controlar diciéndole cómo moverse. A medida que la tortuga se mueve, deja tras de sí un rastro, o en otras palabras, dibuja al avanzar. Puedes imaginarlo como el rastro dejado por un animal cuando se mueve por la arena en la playa o la baba que deja un caracol cuando camino. *Logo* puede usarse para crear algoritmos muy sofisticados, y por lo tanto dibujos muy sofisticados (Por ejemplo, ver http://en.wikipedia.org/wiki/Turtle_graphics).

Comandos Logo

El lenguaje de programación *Logo* consiste en un conjunto de comandos primitivos que controlan a la tortuga. En esta implementación de *Logo*, hemos reemplazado la Tortuga por un Androide. Por lo tanto tus algoritmos le dirán al Androide que hacer. Además, la versión de este tutorial está deliberadamente construida con comandos mucho más "débiles" que la versión original de *Logo*. Los comandos que puedes usar para construir tus algoritmos son:

- Comando ***avanzar***: mueve al Androide hacia adelante 10 pixeles.
- Comando ***girar***: hace que el Androide gire 90° hacia la derecha.
- Comando ***subirLápiz*** : sube el lápiz sobre el lienzo de manera que nada se dibuja cuando la tortuga se mueve. El comando ***bajarLápiz*** pone el lápiz sobre el lienzo para que se vuelva a dibujar.
- Comando ***mostrarTortuga***: hace visible al Androide. El comando ***ocultarTortuga*** hace invisible al Androide.
- Comando ***dibujar***: mueve al Androide de acuerdo al código que tú específicas. Aquí es donde pondrás tus algoritmos de dibujo.
- Comando ***borrar***: limpia el lienzo y mueve al Androide de vuelta a la posición inicial al centro del lienzo.

Todos estos comandos ya están implementados como procedimientos de App Inventor en el proyecto ***Logo1*** que descargaste e importaste. Deberías ver que estos bloques están colapsados en el Editor de Bloques, como se muestra en la Figura 1.3

Figura 1.3: Bloques colapsados de los comandos básicos para mover al Androide.

No tienes que modificar estos procedimientos (sí puedes mirarlos si tienes curiosidad, ¡pero no los cambies!).

Algoritmos

Un *algoritmo* es una secuencia precisa de instrucciones que, cuando se ejecutan en un computador, controlan su comportamiento. Un algoritmo es similar a una *receta de cocina* pero que tiene que ser mucho más preciso si es que se espera que un computador la pueda seguir. Las recetas a menudo tienen instrucciones muy imprecisas que un cocinero novato no sabría como hacer. Por ejemplo, “revuelva hasta que la masa esté suave”. Un cocinero experimentado podría saber exactamente qué significa eso, pero no todos lo sabrán.

Cada instrucción en un algoritmo debe tener un significado preciso y no ambiguo. Por ejemplo, la Figura 1.4 muestra un algoritmo para dibujar un cuadrado de 10 por 10 pixeles en nuestra versión de Logo. A la izquierda, se expresa el algoritmo en *pseudocódigo* y en la derecha se muestra como sería con bloques de App Inventor.

```
para dibujar un
cuadrado de 10 por 10
pixeles:
      avanzar
      girar
      avanzar
      girar
      avanzar
      girar
      avanzar
      girar
```

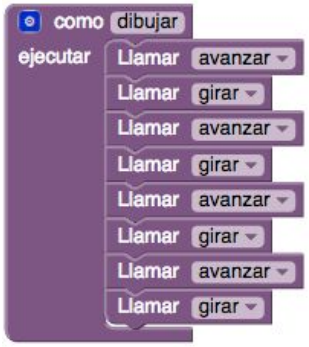


Figura 1.4: Algoritmo para dibujar un cuadrado de 10 por 10 pixeles. A la izquierda en pseudocódigo, a la derecha en bloques de App Inventor.

El *pseudocódigo* es un lenguaje para expresar algoritmos que es muy similar al Español (o Inglés) pero que también incluye código similar al del computador. Su propósito es proveer una manera conveniente de expresar algoritmos como *texto*. Es lo mismo que ocurre con los diagramas de flujo que vimos anteriormente.

Ejercicios

Usa el botón *"Guardar proyecto como"* para crear distintas versiones de la aplicación para cada uno de los ejercicios siguientes. Utiliza una hoja cuadriculada para diseñar los algoritmos de los ejercicios. ¡Vamos a crear los diagramas de flujo!

1. Diseña un algoritmo para dibujar un cuadrado de 20 por 20 pixeles. Observa que **diseñar** un algoritmo *no* es lo mismo que ***programarlo***. Al principio trata de escribirlo a mano o en una pizarra. Parte del diseño es averiguar o establecer qué es lo que tu diseño deber a hacer |es decir, imaginarte a ti mismo como el Androide— y seguir los pasos del algoritmo para ver cual es el resultado.

   Después de que hayas diseñado un buen algoritmo, implementado en App Inventor modificando el procedimiento dibujar para que dibuje un cuadrado de 20 por 20.

1.2. Procedimientos

Hasta el momento te habíamos advertido que no modificaras ningún procedimiento de los que estaban definidos en el proyecto y que solo los miraras en caso de que quisieras saber su funcionamiento. Es ahora el momento de comprender qué es lo que está ocurriendo**.** Un procedimiento es una recopilación de instrucciones las que se ordenan para cumplir una tarea puntual. A partir de un programa grande, cuya solución implique una gran cantidad de instrucciones, es posible descomponer el problema en una gran cantidad de tareas pequeñas las que resolverán instrucciones en específico. Es como si dividieramos una tarea grande en muchas tareas pequeñas. Por ejemplo, el proceso de construir un edificio está dividido en un sin número de tareas, que van desde preparar la tierra y adquirir los implementos para la construcción, hasta la construcción misma. Cuando programamos ocurre algo similar. La gran mayoría de las veces podemos darnos cuenta que hay tareas que se repiten en reiteradas ocasiones. Pensemos en lo que ocurre en facebook cada vez que iniciamos sesión. Supongamos que el proceso de autenticación requiere mucho tiempo para programar y que además, como es sabido, es un proceso que se realiza más de una vez (a diario vemos como todos los sitios requieren que iniciemos sesión al menos una vez). Ahora, hagámonos la siguiente pregunta. Si es que iniciar sesión es algo que hacemos frecuentemente, ¿Es necesario que como programadores escribamos el mismo código cada vez que quisiéramos hacer un sitio con ingreso de usuarios? La respuesta es NO. Los procedimientos vienen a realizar precisamente eso, ayudarnos a establecer tareas que se hacen de manera sistemática, entregarles un nombre y llamarlas las veces que necesitemos sin tener que volver a escribir lo mismo otra vez. **¡Optimizamos tareas, las escribimos solo una vez y la reutilizamos!**

Hasta ahora, hemos llamado a distintos tipos de procedimientos sin saber que lo eran. Por ejemplo el día de ayer cuando incluímos el sonido de nuestro animal, agregamos un bloque de sonido que decía "*sonido.reproducir*". Ese elemento *reproducir* corresponde a algún tipo especial de procedimiento definido para el componente **Sonido**. Lo maravilloso es que App Inventor te permite crear tus propios procedimiento!!

Así como debes suponer, existen cierta reglas a la hora de crear procedimientos y no solo

hay una forma de crear procedimientos, sino que lo puedes hacer de varias maneras. Pensemos un poco en las tareas domésticas que a diario realizamos. Llevando esto a un ejemplo cotidiano, cuando nosotros queremos lavar nuestra ropa generalmente utilizamos una lavadora. Este aparato nos permite lavar grandes cantidades de ropa sin tener que perder tiempo realizando el proceso a mano. Pensemos un poco en el proceso de lavar ropa. Inicialmente nosotros vamos a tener cierta cantidad de ropa a lavar la que ingresaremos a la lavadora con el fin obvio de que esta termine limpia una vez que termine el proceso. Si vemos el detalle del proceso podemos darnos cuenta que la lavadora recibe como entrada la ropa sucia y como salida, esta nos la entrega limpia. Si tuviésemos que programar, la conducta que tiene la lavadora es bastante similar a la que realizan los procedimientos.

Nos daremos cuenta en la medida que avancen los días que existen muchos tipos de procedimientos. Algunos, tal como la lavadora, van a necesitar de información de entrada para su funcionamiento (como en el caso del día de ayer, el procedimiento **vibrar** necesitaba recibir como entrada el tiempo total en el cual iba a estar vibrando el teléfono). De la misma forma, nos dimos cuenta que el procedimiento reproducir no necesitaba nada para funcionar.

App Inventor nos permite crear todos estos procedimientos. A continuación veremos en detalle la manera en como crear estos procedimientos junto a una pequeña referencia de su posible uso:

- Procedimientos que no reciben datos de entrada y que tampoco devuelven algún dato de salida. Este tipo de procedimientos generalmente lo usaremos para realizar tareas puntuales donde el resultado no repercute directamente en algún otro componente. Para poder crear estos procedimientos debemos ir al Editor de Bloques, y desde la opción procedimientos en los bloques "Integrados" arrastrar el bloque morado que dice "como procedimiento ejecutar" donde "procedimiento" es el nombre que le daremos. Un ejemplo de uso típico de este tipo de procedimiento es el de modificar textos de componentes definidas en el Diseñador o bien actualizar el valor de variables. La Figura 1.7 muestra cómo es el bloque en el Editor de Bloques para este tipo de elemento.

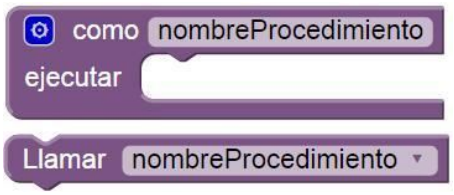


Figura 1.5: bloque para procedimiento "como ejecutar".

- De la misma manera como vimos el día de ayer en el uso de las condiciones, en App Inventor el engranaje azul que poseen algunos bloques nos indica que podemos construir aquel bloque. En este caso, si construimos el bloque anterior nos daremos cuenta que es posible darle valores de entrada (como en el ejemplo de la lavadora, los valores de entrada o parámetros, corresponden a la ropa sucia). Un uso para este tipo de procedimiento es el de actualizar elementos definidos en el Diseñador o el de actualizar variables con el valor que le demos como parámetro de entrada. Si quisiéramos hacer un procedimiento que te salude con tu nombre mediante una etiqueta que diga: **Hola "Tu Nombre"**, *Tu Nombre* debería ser el parámetro de entrada. En La Figura 1.7 podemos ver el formato que tiene este procedimiento. Podemos definir el número de parámetros de entrada que estimemos conveniente

Figura 1.6: bloque para procedimiento "como ejecutar" con parámetro de entrada

- Otro tipo de procedimiento es aquel que, al igual que el anterior, puede o no recibir parámetros de entrada pero que a diferencia de los dos ejemplos anteriores, este si devuelve un resultado. Por ejemplo, cuando inicias sesión en Facebook, el procedimiento encargado de validar que tu usuario y contraseña estén correcto devuelven como resultado la confirmación del inicio de sesión o bien el rechazo. La estructura es similar a los ejemplos anteriores, pudiéndose crear sin parámetros de entrada o construir el bloque para que reciba datos (por ejemplo, para iniciar sesión como datos de entrada debería

recibir el usuario y la contraseña). La 1.8 muestra cómo sería un procedimiento con valor de retorno.

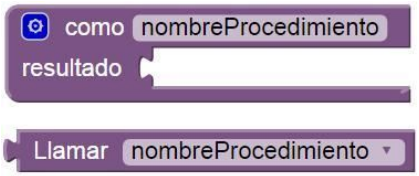


Figura 1.7: bloque para procedimiento "como resultado".

Un aspecto importante a señalar acerca de los procedimientos es que una vez que hayas creado el bloque con el código para su funcionamiento, App Inventor creará un nuevo bloque con la instrucción "llamar". Utiliza este bloque cuando desees utilizar el procedimiento que has creado!!

1.3. Más Ejercicios

1. Diseña un algoritmo para dibujar un cuadrado de 40 por 40 pixeles. Luego implementa tu algoritmo de niendo el procedimiento **cuadrado40** que dibuja un cuadrado de 40 por 40 pixeles. Modifica el procedimiento **dibujar** para que llame a **cuadrado**
2. Define un procedimiento llamado **cuadrado20** que dibuja un cuadrado de 20 por 20 y luego modifica el procedimiento **dibujar** para que llame a **cuadrado20**. Por ejemplo, en la Figura 1.8 se muestra la abstracción para el cuadrado de 10 por 10, que luego se invoca desde dibujar.

Figura 1.8: código para procedimiento **cuadrado10** y cómo se llama desde el procedimiento

Diseña primero, y después programa Este algoritmo será un poco más complejo que cualquiera de los anteriores. Tendrás que usar el procedimiento **subirLápiz** para despegar el Androide del lienzo. También tendrás que planificar cuán lejos moverte hacia adelante para

colocar correctamente los ojos y la boca. **Definitivamente querrás planificar y probar este algoritmo en papel o en la pizarra antes de intentar programarlo.**

Cuando hayas diseñado un algoritmo correcto, impleméntalo como el procedimiento **dibujarCara** que dibuja una cara. Luego prueba tu código para asegurarte que lo hiciste todo correctamente.

4. ¿Puedes dibujar un triángulo con el conjunto actual de comandos Logo? Discute por qué sí o por qué no.
5. Discute sobre qué necesitar amos cambiar sobre los comandos Logo para poder dibujar un triángulo.

2. Tutorial: Algoritmos de Dibujo con Repetición y Selección

En el tutorial anterior desarrollaste algoritmos para dibujar figuras simples. Sin embargo esto fue difícil porque los comandos que usaste eran muy débiles e inflexibles. Por ejemplo, el comando ***avanzar*** solo podía usarse para mover al Androide hacia adelante 10 pixeles. Asimismo, el comando ***girar*** solo podía girar al Androide en 90° . Con estos comandos dibujar un cuadrado de 100 por 100 pixeles sería algo muy *tedioso*[1] y además era imposible dibujar un simple triángulo! En este tutorial hemos mejorado el conjunto de comandos al hacerlos más generales. Las mejoras principales están en los comandos **avanzar(N)** y **girar(G)**:

- El comando **avanzar(N)** mueve el Androide **N pixeles hacia adelante.**
- El comando **girar(G)** hace que el Androide **gire G grados hacia la derecha**.

Los valores N y G son *parámetros* o *argumentos*. Un ejemplo sencillo debiera bastar para mostrar por qué son más general, y por lo tanto, más poderosos. En el tutorial anterior, para mover el Androide 40 pixeles hacia adelante había que llamar 4 veces al comando ***avanzar,*** como se ve en la Figura 2.1.

[1]Eliminar el tedio y el aburrimiento de hacer cosas repetitivas es una gran fuerza motivadora en el desarrollo de nuevos algoritmos, lenguajes de programación, *frameworks* y librerías. En cierto sentido, un programador " ojo", en el sentido de ahorrar trabajo innecesario, llega a ser un buen programador!

```
avanzar
avanzar
avanzar
avanzar
```

Figura 2.1: Algoritmo para avanzar 40 pixeles, usando el comando avanzar en su versión inflexible.

Compara esto con la Figura 2.2 donde, usando el nuevo conjunto de comandos, basta con una llamada ***avanzar(40).***

```
avanzar(40)
```

Figura 2.2: Algoritmo para avanzar 40 pixeles, usando el comando avanzar en su versión general.

La primera versión de ***avanzar*** era muy *específica* mientras que la nueva versión, con un parámetro, es más *general* y es precisamente la presencia del parámetro lo que le da su generalidad. En vez de siempre avanzar 10 pixeles, ahora podemos avanzar cualquier número de pixeles en una sola llamada al procedimiento. La misma observación aplica también al procedimiento ***girar.*** La abstracción anterior era muy específica, porque nos dejaba girar solo 90°. La nueva abstracción, al involucrar un parámetro, nos deja girar cualquier número de grados. *Nota*: tanto la versión anterior y la actual de los procedimientos Logo son *abstracciones.* Pero, claramente, el nuevo conjunto de abstracciones es mucho más poderoso. Como regla general, mientras más general es un procedimiento o abstracción, es mejor.

Los otros comandos Logo son los mismos que en la versión anterior:

- Comando ***subirLapiz*** : sube el lápiz sobre el lienzo de manera que nada se dibuja cuando la tortuga se mueve. El comando ***bajarLapiz*** pone el lápiz sobre el lienzo para que se vuelva a dibujar.
- Comando ***mostrarTortuga***: hace visible al Androide. El comando ***ocultarTortuga*** hace invisible al Androide.
- Comando ***dibujar*****:** mueve al Androide de acuerdo al código que tu especi cas. Aquí es donde pondrás tus algoritmos de dibujo.
- Comando ***borrar***: limpia el lienzo y mueve al Androide de vuelta a la posición inicial al centro del lienzo.

Para comenzar con los siguientes ejercicios, descarga el proyecto ***Logo2*** desde la carpeta *ProgramaTusIdeas/Dia2* e impórtalo en App Inventor. Al igual que en la versión anterior, hemos programado todos los bloques requeridos para que entres a programar tus algoritmos; estos bloques, que se muestran en la Figura 2.3, están colapsados y no es necesario que los edites.

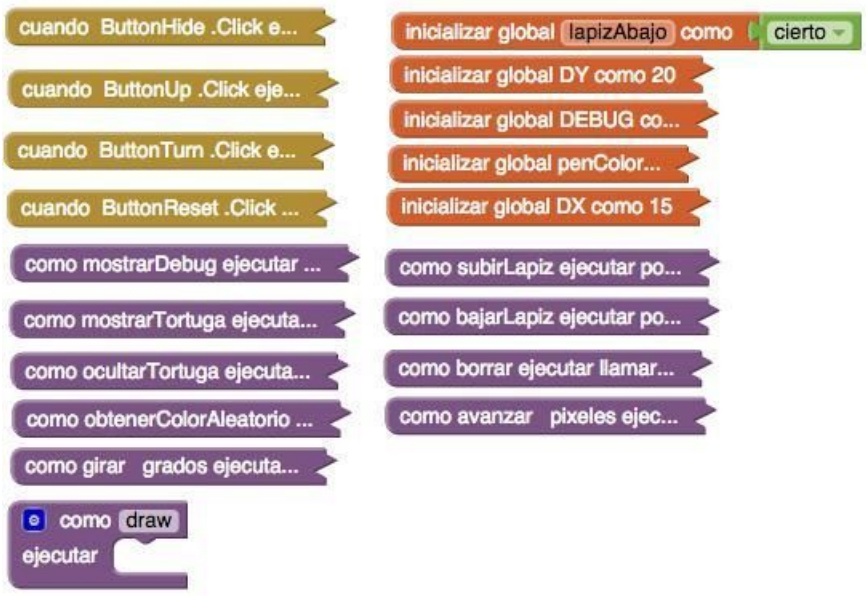


Figura 2.3: Bloques ya programados de la aplicación ***Logo2*** . Nota que hay un nuevo bloque **obtenerColorAleatorio.**

Definir un Procedimiento con Argumentos

En el tutorial anterior definiste procedimientos sin parámetros, pero pronto necesitarás definir procedimientos con parámetros. Para hacer esto, necesitas arrastrar un bloque **procedimiento** hacia el área de trabajo. Como siempre, debes darle un nombre adecuado a tu procedimiento. Luego, para especificar que el procedimiento requiere un parámetro cuando es llamado, presiona el botón azul y arrastra un bloque **entrada x** desde la izquierda, hacia los bloques de la derecha (similar a como agregas una condición **si no**, a un bloque condicional).

Observa la Figura 2.4 para ver cómo tiene que ser.

Figura 2.4: Creando un procedimiento con parámetros.

Reemplaza el nombre del parámetro, **x**, con un nombre más útil y significativo. Puedes agregar más parámetros si es necesario; una vez que termines de agregar parámetros presionando nuevamente el botón azul.

Algoritmos con Repetición y Selección

Al diseñar algoritmos hay tres tipos básicos de *estructuras de control*: secuencias, selección y repetición. Cualquier algoritmo que puedas imaginar puede construirse con estos tres tipos de control.

Secuencias: Ya estás familiarizado con las secuencias, que simplemente significan una secuencia de pasos. En App Inventor ponemos los bloques en secuencia al apilar unos sobre otros, como se ve en la Figura 2.5.

Figura 2.5: Una secuencia de llamadas a **avanzar.**

Selección: También estás familiarizado con la selección, que es simplemente un término que usamos para estructuras **si-sino**, por ejemplo como se ve en la Figura 2.6.

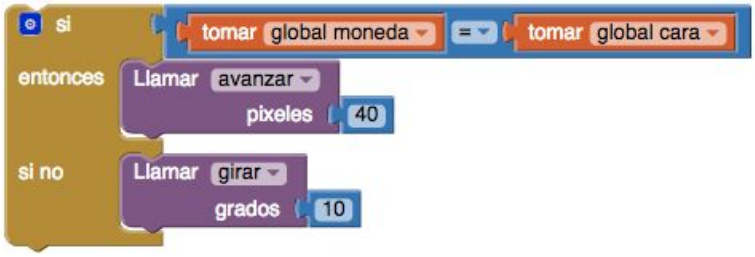
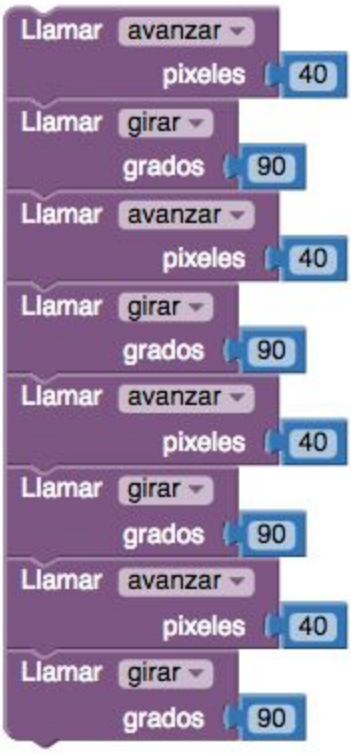


Figura 2.6: Un bloque usando selección.

Repetición: Hasta ahora no hemos usado repetición (o *looping* en inglés) en nuestros algoritmos. La Figura 2.7 muestra un ejemplo simple de cómo un loop puede ser muy útil. Compara los algoritmos a la izquierda y a la derecha.



Cómo leer este bloque: Para cada número (el contador del loop) desde 1 hasta 4, incrementando cada paso en 1, ejecuta los bloques dentro de "ejecuta". Por lo tanto el número cambiar a 1, 2, 3 y 4. En otras palabras, repite los bloques dentro de "ejecuta" 4 veces.

Figura 2.7: Ejemplo de un algoritmo que usa repetición.

El algoritmo a la izquierda usa una secuencia simple con copias de las llamadas a **avanzar** y **girar** para dibujar un cuadrado. Por otro lado, el algoritmo a la derecha utiliza un loop **por cada**; lo que es un enfoque mucho más práctico y general. En este caso, el bloque **por cada *repite*** los bloques dentro de "ejecutar" 4 veces. Hay muchos usos para este bloque, pero por ahora veremos un uso sencillo. Muchos de los ejercicios de este tutorial pueden programarse usando el loop como se muestra aquí : contando desde **inicio** hasta **fin**, con incrementos de 1 en 1.

En el ejemplo de la Figura 2.7 colocamos valores constantes para los valores de inicio, fin, e incremento. Pero también puedes poner variables. Por ejemplo, la Figura 2.8 muestra un procedimiento que dibuja una estrella un poco rara, como la de la Figura 2.9.

Figura 2.8: El procedimiento **dibujarFiguraRara** que usa una variable N en un loop.

Figura 2.9: Una figura dibujada al llamar a **dibujarFiguraRara**. ¿Qué argumentos debes usar para obtener esta figura?.

Copia el procedimiento **dibujarFiguraRara** y empieza a jugar con el número de repeticiones. ¿Cuántas repeticiones necesitas para dibujar una estrella completa?

Ejercicios

1. Usando un bloque **por cada** define un algoritmo para dibujar un cuadrado, de nombre **dibujarCuadrado(L)**, que dibujará un cuadrado de tamaño *L x L* donde *L* es el largo del lado.

Nota Importante! En App Inventor y otros lenguajes de programación, los parámetros pueden tener cualquier nombre. Por lo tanto es importante usar nombres que describen el propósito del parámetro Esto hará que leer el código sea más fácil para ti y para otros programadores.

Por ejemplo, compara los nombres en la Figura 2.10.

Figura 2.10: Una comparación sobre nombres de parámetros.

2. Diseña un algoritmo para dibujar un triángulo equilátero. Es decir, un triángulo donde los lados y los ángulos son iguales. Primero diseña el algoritmo a mano. ¿Cuánto tiene que girar el Androide?

Este ejercicio es un ejemplo de una repetición, así que puedes usar el bloque **por cada** en tu algoritmo. ¿Cuántas repeticiones son necesarias? Una vez que tengas diseñado tu algoritmo, impleméntalo en App Inventor y pruébalo. Define el algoritmo como un procedimiento con 1 parámetro, para que así lo puedas reutilizar en el futuro si es necesario. ¿Qué debe representar el parámetro?

3. Dibuja un pentágono, es decir, una figura de 5 lados donde los lados y los ángulos son iguales. Sigue el proceso del ejercicio anterior. Este ejemplo también es un ejemplo de repetición, así que usa el bloque **por cada**. ¿Cuántas repeticiones son necesarias?

Pista: para dibujar un cuadrado, el Androide tiene que girar 4 veces en 90° , lo que significa que en total gira 360° . Una vez que tengas listo tu algoritmo, impleméntalo como un procedimiento de 1 parámetro. ¿Qué representa este parámetro?

4. Los cuadrados y pentágonos son ejemplos de una figura más general, que se conoce como *polígono.* Un polígono es una figura con múltiples lados. Por lo tanto, un cuadrado es un polígono con 4 lados, y un pentágono es un polígono con 5 lados. Si pudieras diseñar un procedimiento **dibujarPolígono(N,L)**, entonces podrías usarlo para dibujar un cuadrado,

un pentágono, un hexágono (6 lados), un octógono (8 lados), o incluso una aproximación de un círculo (¿36 lados?). **¡Inténtalo!**

Pista: Tu procedimiento necesitará 2 parámetros, N y L, donde **N** es el número de lados (por ejemplo, 4, 5, 6, etc) y **L** es el largo de cada lado.

Pista: Un cuadrado tiene 4 lados y sus giros son en (360/4)° . Un pentágono tiene 5 lados y sus giros son en (360/5)° . Un hexágono... . Un N-ágono... Prueba tu procedimiento **dibujarPolígono(N,L)** usándolo para dibujar un hexágono y un octágono. Conecta el nuevo bloque dentro del bloque **dibuja**r para que cuando el usuario presione el botón "Dibujar" se ejecute tu algoritmo.

5. Usa **dibujarPolígono(N,L)** para dibujar un círculo. Este ejercicio puede requerir mucho ensayo-error para obtener el número correcto de lados y el largo de los lados. ¿Un polígono de 36 lados se parece a un círculo?
6. Para dibujar una flor, dibuja repetidamente un cuadrado y luego gira algún número de grados. Por ejemplo, ve la Figura 2.11. Para cambiar el color del lápiz debes cambiar la propiedad **Lienzo.ColorDePintura**. Puedes usar el procedimiento **obtenerColorAleatorio** que ya viene provisto para obtener colores al azar.

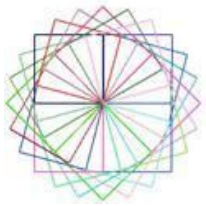


Figura 2.11: Una flor dibujada mediante repeticiones de dibujar un cuadrado y luego girar.

7. Dibuja una flor a la que le falten algunos pétalos, como en la Figura 2.12. Para ello utiliza una moneda (es decir, elige un número aleatorio entre 0 y 1) para tomar la decisión de dibujar o no algún cuadrado.

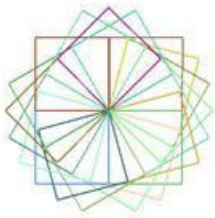


Figura 2.12: Una flor a la que le faltan algunos pétalos

8. Diseña y dibuja tus propias figuras, incluyendo flores, espirales, estrellas, etc. Por ejemplo, la flor de la Figura 2.13 se hizo dibujando círculos y girando.

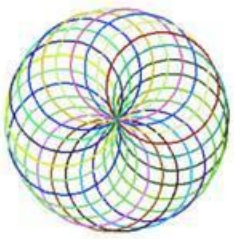


Figura 2.13: Una flor en base a círculos.

Resumen

La lección principal aquí es que la elección de nuestras abstracciones, en este caso el uso de parámetros en los comandos Logo, afecta la clase de problemas que podemos resolver y cómo los resolvemos. Es decir, nuestra elección en cuanto a abstracción tiene un enorme impacto en nuestros algoritmos. Además, la abstracción procedural (con y sin parámetros) facilita la construcción de algoritmos al elevar el nivel de abstracción.

Si te gustó jugar con la programación, te recomendamos que veas el sitio https://blockly-games.appspot.com/ donde hay juegos en línea basados en bloques de código muy similares a los de App Inventor!

3. Tutorial: PintaFotos

En este tutorial presentaremos el componente **Lienzo** para crear aplicaciones con gráficos y animaciones simples en 2 dimensiones (2D). Construirás la aplicación ***PintaFotos*** que permite al usuario dibujar en la pantalla usando distintos colores, usando una imagen de fondo y dibujando sobre ella.

Un dato histórico: PintaFotos fue uno de los primeros programas desarrollados para demostrar el potencial de los computadores, en la década de 1970. En aquel entonces, hacer algo tan simple como esta aplicación de dibujo era algo muy complejo, y los resultados no eran perfectos. Pero ahora, con App Inventor, cualquiera puede rápidamente crear una aplicación de dibujo bastante buena, lo que sirve de excelente punto de partida para luego hacer juegos 2D.

La interfaz de usuario de ***PintaFotos*** se muestra en la Figura 3.1.

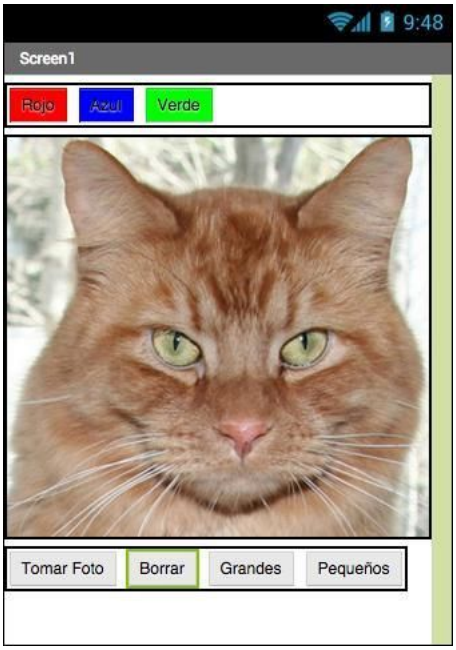


Figura 3.1: Interfaz de usuario aplicación ***PintaFotos***

En esta aplicación podrás:

- Bañar tu dedo en un tarro de pintura virtual para dibujar en este color.
- Arrastrar tu dedo en la pantalla para dibujar una línea.

- Tocar la pantalla para hacer puntos.
- Usar el botón de abajo para limpiar la pantalla.
- Agrandar o achicar el tamaño de los puntos con los botones abajo.
- Sacar una foto con la cámara y luego dibujar encima de esta foto.

Qué Aprenderás

Siguiendo este tutorial aprenderás a:

- Usar el componente **Lienzo** para dibujar.
- Manejar las funcionalidades “touch” y “arrastrar” en la pantalla del equipo.
- Configurar la disposición de los componentes en la pantalla.
- Usar los controladores de eventos que reciben argumentos o parámetros.
- Definir variables para recordar cosas como el tamaño de un punto elegido por el usuario para dibujar.

Para Empezar

Asegúrate que tu computador y teléfono están configurados para usar App Inventor. Crea un nuevo proyecto y nómbrarlo ***PintaFotos***. Abre el Editor de Bloques y comprueba que puedes probar tu aplicación en tu dispositivo mediante la conexión USB. Consulta con tu tutor ante cualquier problema. Para empezar, ve al panel de propiedades a la derecha del Diseñador y cambia el título de la pantalla a "PintaFotos". Deberías ver este cambio reflejado en el teléfono, con el nuevo título apareciendo en la barra de títulos de tu app.

Si estás preocupado por confundir el nombre de tu proyecto con el nombre de la pantalla, ¡no te preocupes! Hay tres nombres claves en App Inventor:

- El nombre que eliges para tu proyecto mientras trabajes en él. También será el nombre de la aplicación cuando la quieras publicar. Nota que puedes hacer clic en “Archivo” y seleccionar “Guardar Como” en el Diseñador para crear una nueva versión o cambiar el nombre de un proyecto.

- El nombre de la componente, **Screen1**, que verás en el panel que contiene el listado de los componentes de la app. No puedes cambiar este nombre en esta versión de App Inventor.
- El título de la pantalla, el que ves en la barra de título del teléfono. Empieza siendo **Screen1**, que es el título que usaste en ***HolaZoo***. Pero lo puedes cambiar, como lo hicimos en ***PintaFotos.***

Diseñando los Componentes

Para crear la app usarás los siguientes componentes:

- Tres componentes **Botón** para seleccionar pintura roja, azul o verde y un componente **DisposiciónHorizontal** para organizarlos.
- Un componente **Botón** para limpiar el dibujo, y otros dos para cambiar el tamaño de los puntos.
- Un componente **Lienzo**, que es la superficie para dibujar. El **Lienzo** tiene una propiedad **ImagenDeFondo,** que configuraremos como *gatito.png.* Esta vez utilizaremos un gatito desde el tutorial ***HolaZoo***. Más adelante, modificarás la app para que la imagen de fondo pueda ser una foto sacada por el usuario.

Crea los Botones de Colores

Primero, crea los 3 botones de color usando las siguientes instrucciones:

- Arrastra un **Botón** al Visor y cambia su **Texto** a “Rojo”, y su **ColorDeFondo** también a rojo.
- En la lista de Componentes, selecciona el **Botón1** y presiona “Cambiar Nombre” para cambiar su nombre por **BotónRojo**. Observa que no se puede poner espacios en los nombres de los componentes, entonces es común poner en mayúscula la primera letra de cada palabra en el nombre.
- De la misma manera, crea dos botones adicionales para azul y verde, nómbralos **BotónAzul** y **BotónVerde** respectivamente. Ponlos debajo del botón rojo. Chequea tu trabajo comparándolo con la Figura 3.2

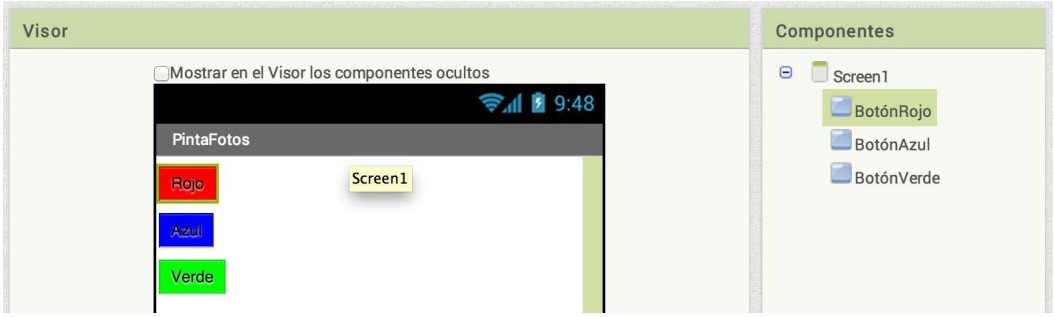


Figura 3.2: El Visor con los tres botones creados.

¡Importante! Observa que en este proyecto cambias los nombres de los componentes en vez de dejarlos con sus nombres por defecto, como lo hiciste en **HolaZoo**. El usar nombres más significativos hace que tus proyectos sean más fáciles de leer, y realmente ayuda en el momento de pasar al Editor de Bloques cuando tendrás que referirte a los componentes por su nombre. A partir de ahora y en el resto del taller usaremos la convención de que cada nombre de componente empiece por su tipo (por ejemplo: **BotónRojo**).

¡Prueba tu aplicación! Conecta tu aplicación a tu dispositivo y verifica que funcione correctamente.

Usando los Componentes de Disposición para una Mejor Presentación

Ahora deberías tener tres botones alineados verticalmente. Sin embargo para esta app, vas a necesitar que estén en la horizontal arriba de la pantalla, como en la Figura 3.3. Para ello debes usar el componente **DisposiciónHorizontal**.

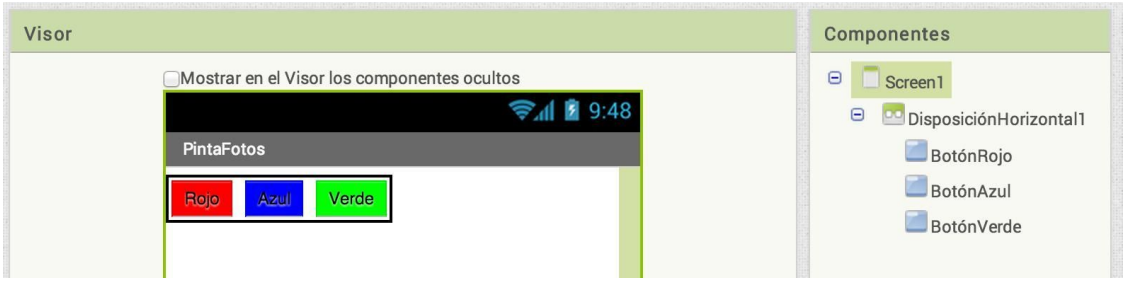


Figura 3.3: Los tres botones en disposición horizontal.

1. Desde la categoría “Disposición” en la Paleta, arrastra un componente **DisposiciónHorizontal** y ponlo debajo de los botones.
2. En el panel de Propiedades, cambia el **Ancho** de **DisposiciónHorizontal** a la opción “Ajustar al contenedor” para llenar todo lo ancho de la pantalla.
3. Mueve los tres botones uno después del otro al interior de la **DisposiciónHorizontal**. *Truco*: Verás una línea vertical azul que indica dónde irá el elemento que estás arrastrando.

Si miras en la lista de componentes del proyecto, verás tres botones listados bajo el componente **DisposiciónHorizontal**, lo que muestra que se trata de sus subcomponentes. Así mismo, observa que todos los componentes están listados bajo el componente **Screen1**.

¡Prueba tu Aplicación! Deberías ver tus tres botones en una fila horizontal en la pantalla, a pesar de que puedan verse un poco diferente a como se ven en el Diseñador. Por ejemplo, el borde de **DisposiciónHorizontal** no aparece en el dispositivo.

En general, se usan los componentes de "Disposición" como opciones de diseño para crear disposiciones simples, verticales, horizontales o en tablas. También puedes crear disposiciones más complejas insertando componentes de disposición unos dentro de otros.

Agregar el Lienzo

- El lienzo es el lugar donde el usuario dibuja círculos y líneas. Agregalo, y configura el archivo *gatito.png*, usado en ***Hola Gatito,*** como su **ImagenDeFondo**.
- Desde la categoría "Dibujo y Animación" de la Paleta, arrastra un **Lienzo** hacia el Visor. Cambia su nombre a **LienzoDeDibujo**. Configura su Ancho como "Ajustar al Contenedor" y su **Alto** a 300 pixeles.
- Recuerda que puedes descargar el archivo *gatito.png* desde ***ProgramaTusIdeas/Dia1/ HolaZoo.***
- Configura la **ImagenDeFondo** del **LienzoDeDibujo** con el archivo *gatito.png*. Si es necesario, debes subir el archivo.

- Cambia el **ColorDePintura** en el **LienzoDeDibujo** al color rojo para que cuando el usuario inicie la app pero todavía no ha presionado ningún botón, sus dibujos sean rojos. Comprueba si lo que has hecho es parecido a la Figura 3.4.

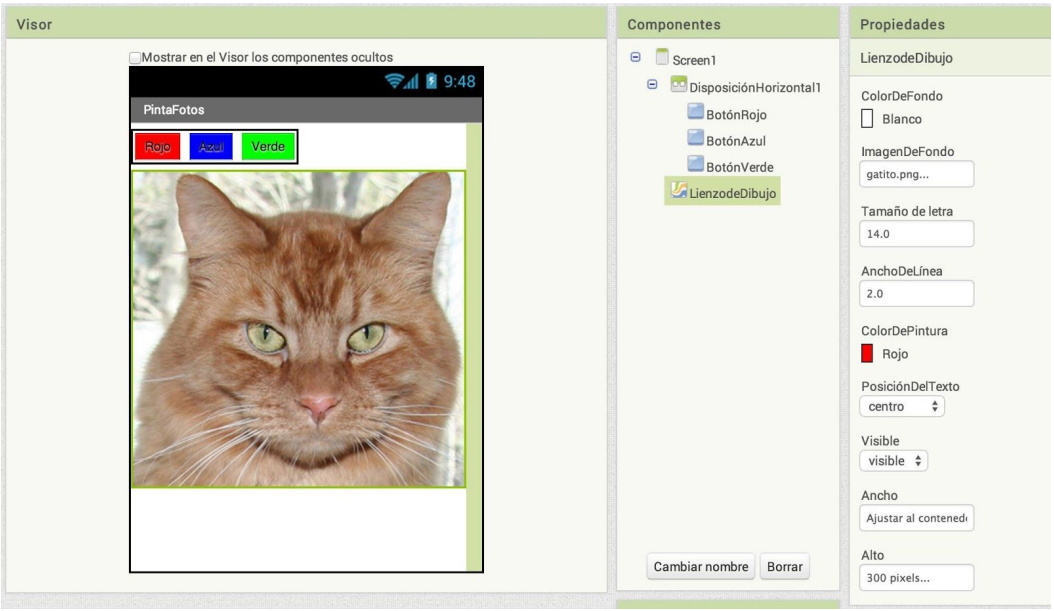


Figura 3.4: El **LienzoDeDibujo** con la imagen *gatito.png* como **ImagenDeFondo**.

Agregar los Botones Inferiores y el Componente Cámara

1. Desde la Paleta, arrastra un nuevo componente **DisposiciónHorizontal** y ponlo abajo del lienzo.
2. Luego arrastra dos componentes **Botón** adicionales en el Visor y ponlos dentro del **DisposiciónHorizontal** que acabas de agregar. Cambia el nombre del primer botón por **BotónCámara** y su **Texto** a “Sacar Foto”. Cambia el nombre del segundo botón por **BotónLimpiar** y su **Texto** por “Limpiar”.
3. Arrastra dos botones más y colócalos al lado derecho del **BotónLimpiar**.
4. Nombra los nuevos botones como **BotónGrande** y **BotónPequeño**, y pon su **Texto** como “Grande” y “Pequeño” respectivamente.
5. Desde la sección Medios de la Paleta, arrastra un componente **Cámara** hacia el Visor. Aparecerá en la sección de los componentes no-visibles.

Una vez completados estos pasos, tu aplicación debería verse como en la Figura 3.5.

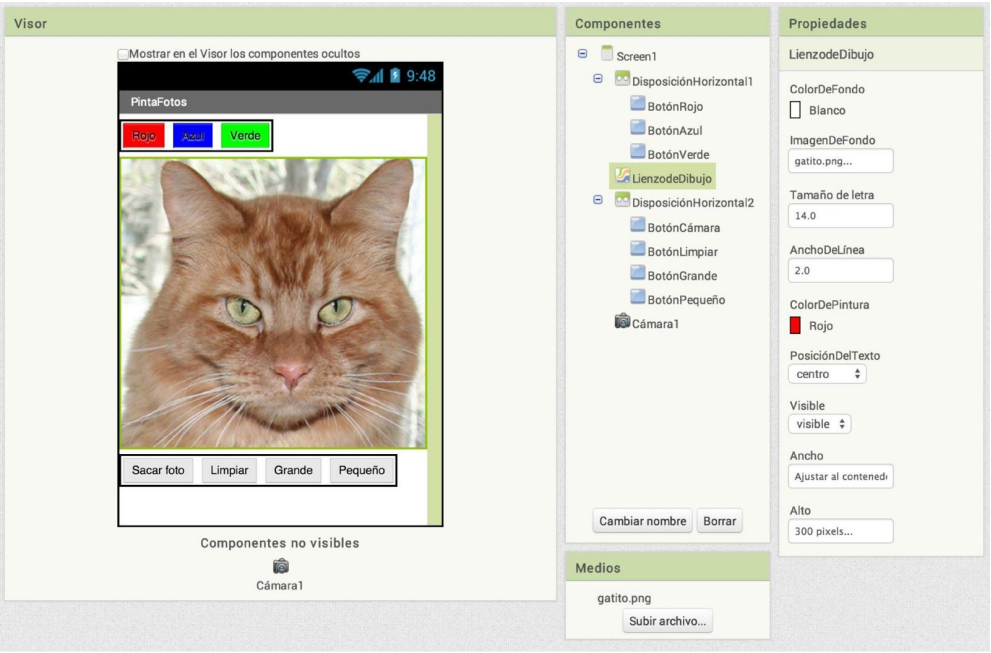


Figura 3.5: Interfaz de usuario completa para ***PintaFotos***

¡Prueba tu Aplicación! ¿Aparece la foto del gatito debajo de los botones de la primera fila? ¿Aparece la última la de botones abajo?

Agregar Comportamiento a los Componentes

El próximo paso es definir cómo se comportan los componentes. Crear un programa de pintura puede parecer un desafío insuperable, pero quedate tranquilo! App Inventor te facilita el trabajo: existen bloques muy fáciles de usar para manejar las funcionalidades de touch y arrastre, y para dibujar y sacar fotos.

En el Diseñador, agregaste un componente **Lienzo** llamado **LienzoDeDibujo**. Como todos los componentes de ese tipo, **LienzoDeDibujo** tiene un evento **Tocar** y un evento **Arrastrado**. Programarás el evento **LienzoDibujo.Tocar** de tal manera que llame a la función **LienzoDibujo.DibujarCírculo**. Programarás el evento **LienzoDibujo.Arrastrado** de tal manera que llame a la función **LienzoDibujo.DibujarLínea**. Programarás luego los botones para configurar la propiedad **ColorDePintura** usando el bloque **poner LienzoDibujo.ColorPintura**, y para limpiar el **LienzoDeDibujo**; finalmente, programarás cómo cambiar la **ImagenDeFondo** del lienzo por una foto sacada con la cámara del dispositivo.

Agregar el Evento Tocar para Dibujar un Punto

Primero, dispondrás los objetos de manera que cuando tocas el **LienzoDeDibujo**, se dibuje un punto en el lugar del lienzo que toques:

1. En el Editor de Bloques, selecciona el **LienzoDeDibujo**, y arrastra el bloque **LienzoDeDibujo.Tocado** al Visor. El bloque tiene parámetros para x e y, y SpriteTocado, como ilustrado en la Figura 3.6. Estos parámetros entregan información sobre la ubicación del evento "tocar la pantalla" hecho por el usuario.

Figura 3.6: El evento viene con información sobre la ubicación del toque en la pantalla

Nota. Si completaste la aplicación ***Hola Zoo***, ya estás familiarizado con los eventos de **Botón.clic,** pero no con los eventos del lienzo. Los eventos **Boton.clic** son bastante sencillos porque no hay nada más que saber del evento aparte del hecho que ocurrió. Algunos controladores de eventos, sin embargo, vienen con información sobre los argumentos para llamar al evento. El evento **LienzoDibujo.Tocar** te entrega las coordenadas **x** e **y** del toque dentro del **LienzoDeDibujo**. También te dice si un objeto dentro del **LienzoDeDibujo** (lo que se conoce como *Sprite*) fue tocado, pero este punto se abordará más adelante. Las coordenadas x e y son los argumentos que usaremos para grabar donde el usuario tocó la pantalla, de manera de poder dibujar el punto en este lugar.

2. Arrastra un bloque **LienzoDeDibujo.DibujarCírculo** desde el bloque y ponlo dentro del controlador de eventos **LienzoDeDibujo.Tocar**, como se ilustra en la Figura 3.7.

Figura 3.7: Cuando el usuario toca el lienzo, la aplicación dibuja un círculo

Por el lado derecho del bloque **LienzoDeDibujo.DibujaCírculo**, verás tres espacios vacíos por completar con los argumentos siguientes: **x**, **y,** y **r.** Los **x** e **y** especifican la ubicación donde el círculo se va a dibujar, y **r** determina el radio (o tamaño) del círculo.

El signo amarillo con el punto de exclamación a la izquierda inferior de la pantalla aparecer con un 1 cuando un espacio todavía permanece vacío. Construiremos los bloques para completar esto después.

Este controlador de evento puede confundir un poco porque el evento **LienzoDeDibujo.Tocar** también tiene una **x** y una **y**. Solamente acuerdate que la **x** y la **y** para el evento **LienzoDeDibujo.Tocar** te dice dónde tocó el usuario, mientras que el **x** e **y** para el evento **LienzoDeDibujo.DibujarCírculo** son espacios abiertos para que tú definas donde se dibujará el círculo. Dado que quieres que el círculo aparezca donde tocó el usuario, conectarás los valores **x** e **y** desde el **LienzoDeDibujo.Tocar** de la misma forma que los valores de los parámetros **x** e **y** en **LienzoDeDibujo.DibujarCírculo.**

Nota. Puedes acceder a los valores del evento al pasar el mouse sobre los parámetros en el bloque “Cuando”. Al hacerlo, aparecerán los bloques **tomar** y **poner a**. Arrastra el bloque **tomar x** y conéctalo como el valor faltante en para **x** en **LienzoDeDibujo.DibujarCírculo**, como se ilustra en la Figura 3.8

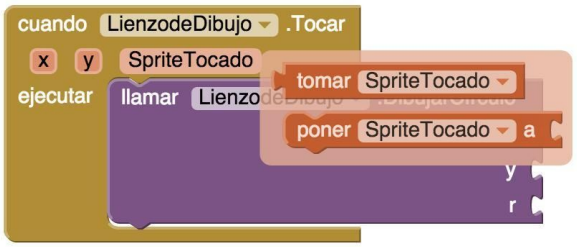


Figura 3.8: Para acceder al parámetro de un evento, arrastra un bloque **tomar** desde el bloque **LienzoDeDibujo.Toca**r

3. Arrastra los bloques **tomar x** y **tomar y** y conéctalos en los espacios abiertos en el bloque **LienzoDeDibujo.DibujarCírculo**, como se ilustra en la Figura 3.9.

Figura 3.9: La aplicación sabe donde dibujar (x,y), pero todavía necesitamos especificar cuán grande debería ser el círculo.

4. Ahora necesitas especificar el radio del círculo a dibujar. El radio se mide en pixeles, que es el punto más pequeño que puede ser dibujado en la pantalla. Por ahora, ponle como valor el número 5: haz clic en una zona blanca de la pantalla y tipea 5, luego presiona **Enter** para crear automáticamente un bloque numérico, y conecta este bloque en el espacio para el parámetro **r**. Cuando hagas esto, el signo amarillo en la esquina inferior se pondrá en 0 nuevamente, dado que todos los espacios abiertos habrán sido completados. La Figura 3.10 muestra como debiera verse el controlador de eventos **LienzoDeDibujo.Tocar**.

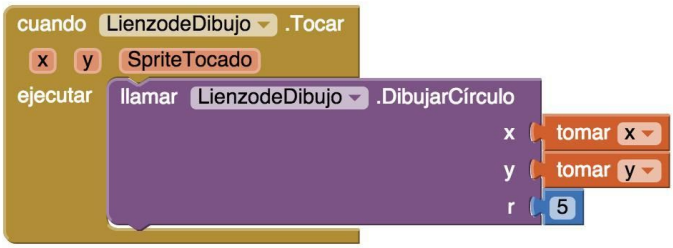


Figura 3.10: Cuando el usuario toca el **LienzoDeDibujo**, un círculo de radio 5 será dibujado en las coordenadas (x, y) correspondientes.

> **Nota.** Al poner un 5 en el Editor de Bloques y luego presionar Enter, habrás usado lo que se llama *tipeo de bloques*. Si empiezas a tipear, el editor de bloques muestra un listado de bloques cuyos nombres corresponden a los que estás tipeando. Si tipeas un número, crea un bloque numérico.

> **¡Prueba tu Aplicación!** Usa tu dispositivo para probar lo que has desarrollado hasta ahora. Al tocar el **LienzoDeDibujo**, tu dedo debería dejar un punto en cada lugar que toques. Los puntos serán rojos si configuraste la propiedad del **LienzoDeDibujo.ColorDePintura** como color rojo en el Diseñador (sino será negro, que es el color por defecto).

Agregar el Evento de Arrastre para Dibujar una Línea

Luego, agregarás el controlador de eventos de arrastre. La diferencia entre un *touch* y un *arrastre* es la siguiente:

- Un touch es cuando pones tu dedo en la pantalla y solo lo levantas para tocar, sin desplazarlo.
- Un arrastre es cuando pones tu dedo en la pantalla y lo mueves al mantener contacto con la pantalla.

En un programa de pintura, cuando arrastras tu dedo en la pantalla se dibuja una línea igual al camino que sigue tu dedo. Lo que estás haciendo en realidad es dibujar cientos de pequeñas líneas rectas; cada vez que mueves tu dedo, incluso un poco , extiendes la línea desde la última posición de tu dedo hacia su nueva posición.

1. En el editor de Bloques, desde la sección del **LienzoDeDibujo**, arrastra el bloque **LienzoDeDibujo.Arrastrado** al espacio de trabajo. Deberías ver el gestionador de eventos tal como en la Figura 3.11.

Figura 3.11: Un evento de arrastre tiene más argumentos que un evento de *touch*.

El evento **LienzoDibujo.Arrastre** viene con los siguientes argumentos:

- *XInicial, YInicial:* la posición de tu dedo donde empieza el arrastre.
- *XActual, YActual*: la posición actual de tu dedo.
- *XPrevio, YPrevio*: la posición inmediatamente anterior de tu dedo.
- *SpriteArrastrado*: un booleano, será verdad si el usuario hace el arrastre directamente sobre un imagen Sprite. En el taller no usaremos este argumento.

2. Arrastra el bloque **LienzoDeDibujo.DibujarLínea** dentro del bloque **LienzoDeDibujo.Arrastrado** como se muestra en la Figura 3.12.

Figura 3.12: Agregando la habilidad de dibujar líneas

El bloque **LienzoDeDibujo.DibujarLínea** tiene cuatro argumentos, dos para cada punto que determina la línea: un punto es *(x1, y1)*, mientras que *(x2, y2)* es el otro. ¿Puedes adivinar qué valores corresponden a cada argumento? Recuerda que el evento **LienzoDeDibujo.Arrastrado** se activará cada vez que arrastres tu dedo en el Visor, por lo tanto la aplicación dibuja una pequeña línea cada vez que mueves tu dedo, desde *(XPrevio, YPrevio)* hacia (XActual, YActual). Agreguemos esto a nuestro bloque **LienzoDeDibujo.DibujarLínea**:

3. Arrastra los bloques **tomar** para los argumentos que vas a necesitar. Los valores *XPrevio* e *YPrevio* deberían ir conectados en los espacios para los argumentos **x1** e **y1**, respectivamente, como se ilustra en la Figura 3.13.

Figura 3.13: Cuando el usuario arrastre el dedo, la aplicación dibujar una línea desde el punto anterior hacia el actual.

¡Prueba tu Aplicación! Arrastra tu dedo en la pantalla para dibujar líneas y curvas. Toca la pantalla para dibujar puntos.

Agregar Controladores de Eventos para Botones

La aplicación que has construido hasta ahora permite al usuario dibujar, pero siempre estos dibujos usan el color rojo. El próximo paso consiste en agregar controladores de eventos para los botones de colores, para que el usuario pueda cambiar el color de la pintura, y otro para el **BotónLimpiar** para que pueda limpiar la pantalla y así volver a empezar a dibujar.

En el Editor de Bloques:

1. Arrastra el bloque **BotónRojo.clic** al espacio de trabajo.
2. Arrastra el bloque poner **LienzoDeDibujo.ColorDePintura** y conéctalo en las acciones del bloque **BotónRojo.clic**.
3. Abre la sección "Colores" y arrastra el bloque de color rojo para conectarlo con el bloque poner **LienzoDeDibujo.ColorDePintura**
4. Repite los pasos anteriores para los botones azul y verde.

5. El último botón por configurar es el **BotónLimpiar**. Para limpiar el lienzo de dibujo debes utilizar el bloque **LienzoDeDibujo.Limpiar**. Confirma que tus bloques se ven como en la Figura 3.14.

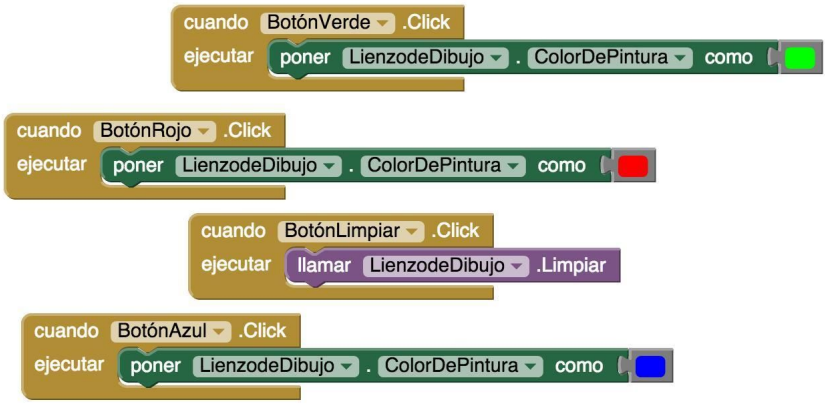


Figura 3.14: Cuando el usuario hace clic en los botones de colores cambia el color para dibujar en el lienzo. Hacer clic en "Limpiar", borra el contenido del lienzo.

Permitir al Usuario Tomar un Foto

Las aplicaciones hechas con App Inventor pueden interactuar con las funcionalidades de los dispositivos Android, incluyendo la cámara. Para personalizar la aplicación, se permite al usuario fijar la imagen de fondo de su dibujo con una foto tomada con la cámara.

1. El componente **Cámara** contiene dos bloques claves. El bloque **Cámara.TomarFoto** inicia la aplicación de cámara del teléfono. El evento **Cámara.DespuésDeTomarFoto** se activa cuando el usuario terminó de sacar la foto. Agregarás bloques en el controlador de eventos **Cámara.DespuésDeTomarFoto** para configurar el **LienzoDeDibujo.ImagenDeFondo** con la foto que fue sacada. Abre la sección de **BotónCámara** y arrastra el controlador de evento **BotónCámara.clic** hacia el espacio de trabajo.
2. Arrastra el bloque **Cámara1.TomarFoto** y ponlo en el controlador **BotónCamara.TomarFoto.**
3. Arrastra el controlador de eventos **Cámara1.DespuésDeTomarFoto** al espacio de trabajo.
4. Arrastra el bloque poner **LienzoDeDibujo.ImagenDeFondo** y ponlo en las acciones de **Cámara1.DespuésDeTomarFoto.**

5. **Cámara1.DespuésDeTomarFoto** tiene un argumento llamado imagen, que es la foto recién sacada. Puedes referirte a ella, con un bloque "tomar" desde el bloque **Cámara1.DespuésDeTomarFoto**. Conecta la imagen como el argumento para el bloque poner **LienzoDeDibujo.ImagenDeFondo**.

Los bloques deberían ser parecidos a la Figura 3.15.

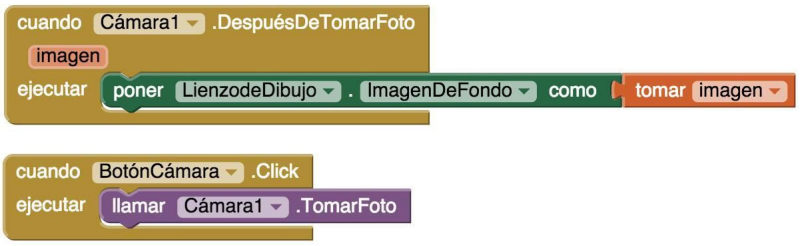


Figura 3.15: Cuando se saca la foto, se configura como la imagen de fondo del lienzo.

¡Prueba tu Aplicación! Haz una prueba al hacer clic en "Tomar Foto" y saca una foto. El gatito debería cambiar por la foto que acabas de sacar, y luego podrás dibujar encima de la foto que sacaste.

Cambiar el Tamaño del Punto

El tamaño de los puntos dibujados en el **LienzoDeDibujo** se determina al llamar a **LienzoDeDibujo.DibujarCírculo**, donde el argumento de radio está configurado en 5. Para cambiar el grueso de la línea, puedes cambiar el valor de **r**. Para probar esto, intenta cambiar el 5 por un 10 y pruébalo en el teléfono para ver cómo aparece. El único tamaño que el usuario podrá usar es el que tú indicas como argumento para el radio del círculo. ¿Pero qué pasa si el usuario quiere cambiar el tamaño de los puntos? Vamos a modificar el programa de manera que el usuario—no solo el programador—pueda cambiar el tamaño de los puntos. Lo haremos de tal manera que cuando el usuario haga clic en un botón llamado "Grandes", el tamaño será 8, y cuando hará clic en un botón llamado "Pequeños", será 2. Para usar distintos valores en el argumento radio, la aplicación necesita saber cuál queremos aplicar. Necesitamos pedirle usar un valor específico, y tiene que memorizar este valor de manera a poder seguir usándolo. Cuando tu aplicación necesita guardar algo en memoria que no sea una propiedad, tal como lo vimos anteriormente, puedes definir una *variable*. Una variable es una *celda de memoria*. Es como un

balde donde puedes almacenar datos variables, como el tamaño del punto. Empezamos por definir una variable TamañoPunto:

1. En el Editor de Bloques, desde la sección "Variables", arrastra un bloque **Inicializar global nombre**. Cambia el texto "nombre" por "TamañoPunto".
2. Fíjate que el bloque **Inicializar global TamañoPunto** tiene un espacio abierto. Ahí es donde puede especificar el valor inicial de la variable, o su valor por defecto al iniciar la aplicación. Para esta aplicación, inicializa el TamañoPunto en 2 creando un bloque con el número 2, y conectándolo en **Inicializar global TamañoPunto** como se ilustra en la Figura 3.16.

Figura 3.16: Inicializar variable global **TamañoPunto** con valor 2.

Usar las Variables

Ahora, queremos cambiar el argumento de **LienzoDeDibujo.DibujarCírculo** en el controlador de eventos **LienzoDibujo.Tocar** para que uses el valor TamañoPunto en vez de usar siempre un número fijo. (Puede dar la ilusión que fijamos TamañoPunto en 2 porque lo inicializamos de esta manera, pero verás en un minuto cómo podemos cambiar TamañoPunto y así cambiar el tamaño del punto que se dibuja.)

1. Arrastra un bloque "tomar" desde **LienzoDeDibujo.DibujarCírculo**. Deberías ver un bloque **tomar global TamañoPunto** que indica el valor de la variable.
2. Anda al controlador de eventos **LienzoDeDibujo.Tocar** y arrastra el bloque del número 5 fuera del espacio **r** y pónlo en el papelero. Luego reemplázalo con el bloque **tomar global TamañoPunto** (Ver Figura 3.17). Cuando el usuario toca el lienzo, la aplicación determinar ahora el radio a partir de la variable TamañoPunto.

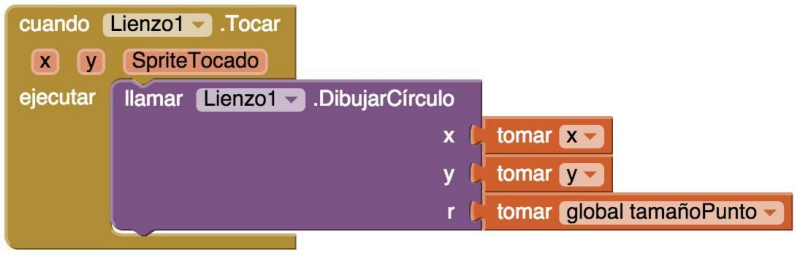


Figura 3.17: Ahora el tamaño de cada círculo depende de lo que está guardado en la memoria de la variable **TamañoPunto**.

Cambiar los valores de las variables

Aquí es donde la magia de las variables opera. La variable TamañoPunto permite al usuario elegir el tamaño del círculo, y tu controlador de eventos dibujar el círculo en función de esto. Desarrollaremos el comportamiento al programar los controladores de eventos **BotónPequeño.clic** y **BotónGrande.clic.**

1. Arrastra un controlador de eventos **BotónPequeño.clic** al espacio de trabajo. Luego arrastra desde la categoría "Variables" un bloque **poner a,** selecciona "global TamañoPunto" como referencia y conéctalo a **BotónPequeño.clic**. Finalmente, crea un bloque número 2 y conéctalo al bloque poner **global TamañoPunto.**
2. Haz un controlador de eventos similar para **BotónGrande.clic,** pero configura el TamañoPunto en 8. Ambos gestionadores de eventos deberían aparecer en el editor de bloques, como en la Figura 3.18.

Figura 3.18: Hacer clic en los botones cambia el tamaño del punto. Los toques que siguen serán con este tamaño de punto.

¡Prueba tu Aplicación! Intenta hacer clic en los botones de tamaño y luego toca el lienzo. ¿Se dibujan círculos de distintos tamaños? ¿Las líneas? El tamaño de las líneas no debería cambiar porque programaste *TamañoPunto* solamente para ser usado en el bloque de **DibujarCírculo**. En

base a esto, ¿podrías ahora cambiar tus bloques para que el usuario también pueda cambiar el tamaño de la línea? (Nota que el lienzo tiene una propiedad llamada **AnchodeLinea**)

La App Completa: PintaFotos

La Figura 3.19 ilustra el código completo de la aplicación ***PintaFotos***:

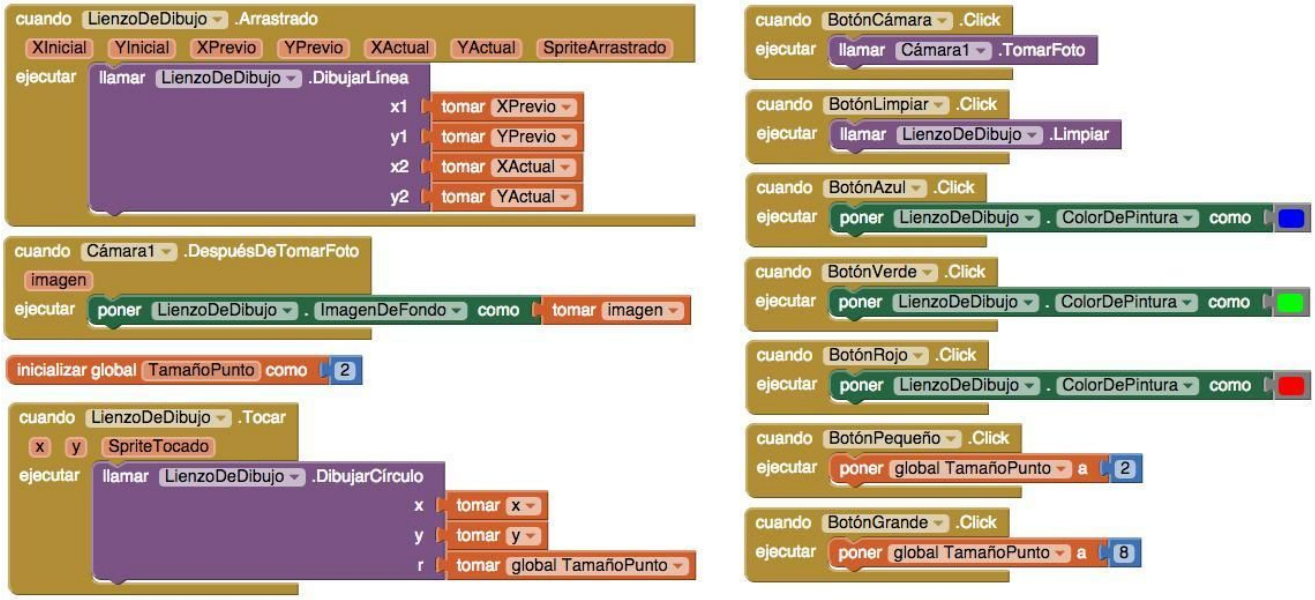


Figura 3.19: Grupos de bloques para ***PintaFotos.***

Resumen

Las ideas claves que hemos visto en este tutorial son las siguientes:

- El componente **Lienzo** te permite dibujar en un lienzo. También es sensible a eventos "*touch*" y de "*arrastre*", los cuales puedes aprovechar para desarrollar funcionalidades para dibujar.
- Puedes usar componentes de disposición en la pantalla para organizar tus componentes en lugar de ponerlos todos uno debajo del otro.
- Algunos controladores de eventos vienen con información sobre el evento, como las coordenadas de donde la pantalla fue tocada. Esta información está representada por argumentos. Cuando arrastras un controlador de eventos que tiene argumentos, App Inventor crea ítems "poner a" y "tomar", dentro del bloque seleccionado, para así poder referirse a estos argumentos.

- Crear variables al usar bloques **poner global nombre,** desde la categoría "Variables". Las variables permiten guardar información en memoria, como el tamaño del punto, que no está guardado en una propiedad de un componente.
- Para cada variable que definas, App Inventor automáticamente provee una referencia "*tomar global*" que entrega el valor de la variable, y una referencia *"poner global a"* para cambiar el valor de la variable. Para acceder a esto, arrastra un bloque "poner a" o "tomar" desde el mismo bloque donde declaras la variable.

El componente Lienzo también se puede usar para programar animaciones como las que ves en juegos en 2D.

4. Discusión y Ejercicios de Personalización

Preguntas/Discusión

1. En el Lienzo, los controladores de eventos **Tocar** y **Arrastrado**, mostrados abajo en la Figura 4.1, tienen parámetros de evento. Nombra cada parámetro e indica lo que representa.

Figura 4.1: Grupos de bloques para ***PintaFotos***.

2. Los parámetros de evento son distintos de los parámetros para llamar funciones. ¿Cómo se llaman los parámetros para llamar funciones en el controlador de eventos de la Figura

4.1? ¿Quién especifica los parámetros de función? ¿Quién especifica parámetros de eventos?

3. ¿Cuál es el propósito de definir una variable para el Tamaño de Punto en la App Pinta Fotos? Si quieres permitir cambiar el grosor de las líneas, ¿necesitarás una variable?
4. Define lo que es una variable. ¿Cuáles son sus puntos en común con una propiedad? ¿Cómo se diferencia de una propiedad?
5. Una forma de personalizar la aplicación es poner un **CuadroDeTexto** para el tamaño del punto. ¿Cuál es la diferencia entre un cuadro de texto y una variable? ¿Cuál es la diferencia entre una etiqueta y un cuadro de texto?

Ejercicios de Personalización

1. La interfaz de usuario de ***PintaFoto*** no permite mostrar con qué color se está dibujando. El usuario solo se da cuenta al dibujar. ¿Puedes agregar esta información para el usuario de tal manera que cuando él hace clic para cambiar el color, la interfaz cambia y le indica qué color fue elegido?
2. El tamaño del punto para dibujar un círculo sólo puede ser de 2 o 8. ¿Puedes cambiar esto para permitir al usuario elegir entre distintas opciones con un componente **Deslizador** o un **CampoDeTexto**?
3. Permitir al usuario controlar el grosor de las líneas, de la misma manera que puede cambiar el tamaño del punto. El lienzo tiene una propiedad **AnchoDeLinea** que controla esta característica.

5. Material de Apoyo

El Componente Lienzo

El componente **Lienzo** es una subsección dentro de tu aplicación. El lienzo se usa para dibujar y hacer animaciones—tu app puede dibujar objetos, y puedes dar al usuario la capacidad de dibujar objetos. Normalmente vas a necesitar que el lienzo llene por completo el ancho de la pantalla de la app, entonces tendrás que ajustar la propiedad **Ancho** con la opción “Ajustar al

contenedor". Por lo general vas a necesitar tener otros elementos abajo o arriba, por lo que configurarás el **Alto** como un número fijo de pixeles.

La ubicación de un objeto en el lienzo se define con coordenadas **X**, **Y** relativas a la esquina arriba e izquierda del lienzo. **X** es la ubicación horizontal del objeto, siendo 0 el borde izquierdo y **X** creciendo cuando el objeto se mueve hacia la derecha. **Y** es la ubicación vertical, con 0 siendo el borde superior e **Y** creciendo cuando el objeto se mueve hacia abajo. Conceptualmente el lienzo se comporta como una rejilla, tal como se ilustra en la Figura 5.1.

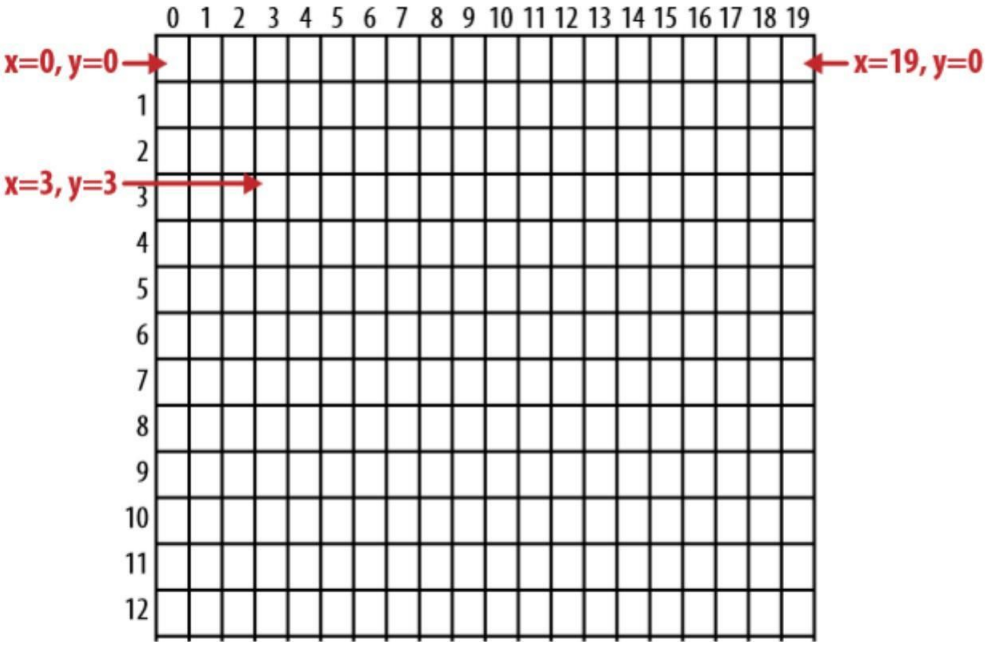


Figura 5.1: Un lienzo con ancho de 19 pixeles y altura de 12 pixeles. Se muestran los puntos en coordenadas (0,0), (3,3) y (19, 0).

Ejemplo: ¿Cómo dibujar un círculo en (10,10)? El lienzo tiene bloques funcionales para dibujar un círculo o una línea. El círculo tiene tres parámetros **x**, y **y** un radio **r**. **x** es la ubicación horizontal, **y** es la ubicación vertical, y **r** es el radio del círculo por dibujar. Si **x** tiene un valor de 10 significa que el círculo será ubicado 10 pixeles a la derecha del borde izquierdo del lienzo. Si **y** tiene un valor de 10 significa que el círculo será ubicado 10 pixeles abajo del borde superior del lienzo. La Figura 5.2 muestra cómo dibujar un círculo en la coordenada (10,10) y con radio de 5 pixeles.

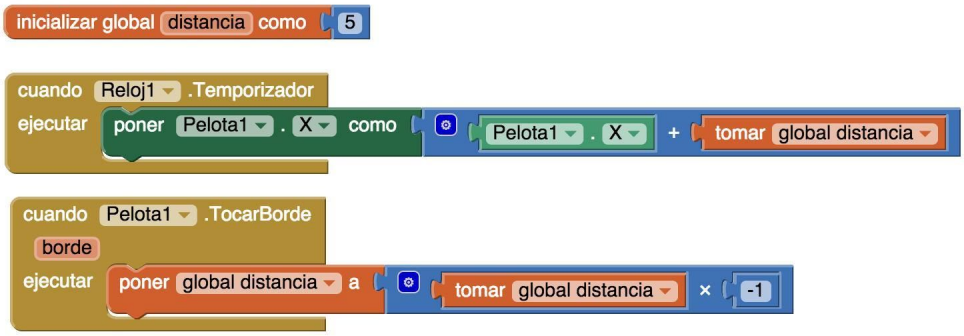


Figura 5.2: Dibujando un círculo en el lienzo, en la coordenada (10,10) y con radio 5 pixeles.

Ejemplo: ¿Cómo dibujar un círculo en el lugar donde el usuario toca el lienzo?

La funcionalidad "*touch*" (táctil) se activa cuando el usuario toca con el dedo en el lienzo. Tiene parámetros **x** e **y** que indican la ubicación del toque. El parámetro ***SpriteTocado*** especifica si el toque ocurrió o no en un **SpriteImagen** (esto no in uye en su comportamiento).

Para dibujar el círculo donde el usuario tocó el lienzo, debes pasar el mouse sobre los parámetros **x** e **y** para arrastrar los bloques tomar para cada uno, y luego conectarlos en los espacios para **x** e **y** del bloque **DibujarCírculo**. No te confundas con lo siguiente: los parámetros del evento **Tocar** tienen el mismo nombre que los espacios libres para especificar la ubicación del círculo en **DibujarCírculo**, pero son valores independientes y con conceptualización diferente. La Figura 5.3 muestra el código descrito anteriormente.

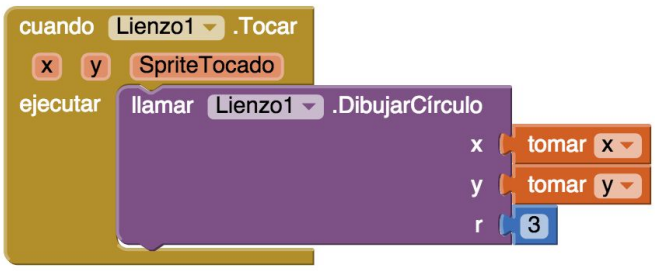


Figura 5.3: Dibujando un círculo en el lugar donde el usuario toca el lienzo.

Ejemplo: ¿Cómo mover una imagen al centro del lienzo? Los bloques de la Figura 5.4 ubican el **SpriteImagen1** en el centro del lienzo usando las propiedades **Ancho** y **Alto**. Dichas referencias abstractas significan que el código funcionar aunque el lienzo cambie de tamaño.

Figura 5.4: Código para posicionar un sprite en el centro del lienzo.

Una alternativa es poner números fijos para **X** e **Y**, después del bloque **SpriteImagen1.MoverA.** Es muy similar a como (10, 10) fue usado en el ejemplo anterior. ¿Sabes por qué es sustraída la mitad del alto y ancho del sprite?

Fíjate que no existe un bloque **DibujaImagen** como **DibujarCírculo**. En vez de eso, con el Diseñador arrastras los componentes **SpriteImagen** en un lienzo y especificas la imagen (una foto) que define su apariencia. No hay forma de crear sprites dinámicos con bloques, sólo se pueden hacer visibles o invisibles, y se puede cambiar su ubicación, tal como se muestra en este ejemplo.

Variables

¿Cuándo defines una variable? Una aplicación memoriza cosas, tiene una memoria "escondida". Puedes imaginártelo como una hoja de cálculo (o planilla) dentro del "cerebro" privado de la aplicación. Tú, el programador, controlas su memoria. Cuando arrastras un componente en tu aplicación (por ejemplo un botón, un cuadro de texto), cada componente tiene una serie de propiedades. Estas propiedades son celdas de tu hoja de cálculo que puedes modificar.

También puedes definir nuevas celdas en tu planilla, cuando necesitas recordar algo: se llaman *variables.* Defines una variable cuando no existe en el componente una propiedad para almacenar la información que necesitas.

Ejemplo: ¿Como hacer para que cada vez que el usuario hace clic se agranda el círculo?

Como se ilustra en la Figura 5.5, cuando se toca el **Lienzo1** se llama a **Lienzo1.DibujarCírculo** para dibujar un círculo donde se hizo el touch, en las coordenadas (x, y). El radio **r** no puede ser un

número fijo si queremos un tamaño de c rculo distinto cada vez. La variable TamañoCírculo se usa para seguir y recordar el tamaño de los círculos cada vez que se dibuja uno. Se inicializa TamañoCírculo en 1 para que el primer círculo que se dibuje sea sólo un pequeño punto. Luego, al final del controlador de eventos **Lienzo1.Tocar** el tamaño del círculo se duplica. Entonces la segunda vez el círculo tendrá un radio 2, luego 4, luego 8, etc. La variable TamañoCírculo se necesita porque no hay ninguna propiedad del componente que se pueda usar. Por ejemplo, el componente **Lienzo** tiene una propiedad **AnchoDeLínea**, entonces si estuvieses dibujando líneas no necesitarías definir una variable porque podrías usar esta propiedad. Pero el lienzo no tiene una propiedad "Tamaño de Círculo", por lo que debes definirla tú mismo usando una variable.

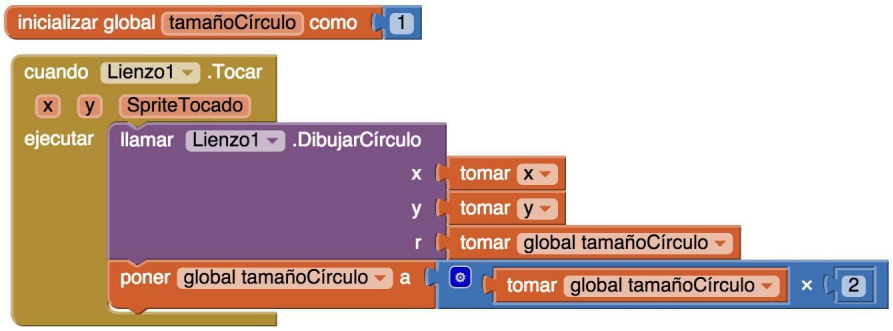


Figura 5.5: Usando una variable para controlar el radio de los círculos dibujados.

Resumen

Cuando inicias una aplicación, ésta empieza a ejecutar sus operaciones y a responder a eventos que ocurren. Al responder a eventos, la aplicación a veces necesita recordar cosas. Para un juego, puede ser que tenga que recordar el marcador de cada jugador o la dirección en la cual se mueve un objeto. Tu aplicación recuerda cosas con propiedades de componentes, pero cuando necesitas memoria adicional no asociada a un componente, puedes definir variables. Puedes almacenar valores dentro de una variable y recuperar el valor actual, de la misma manera que con las propiedades. Como los valores de propiedades, los valores variables no son visibles para el usuario final. Si quieres que el usuario final vea la información almacenada en una variable, agregas bloques que muestran esta información en una etiqueta u otro componente de la interfaz de usuario.