

Mente maestra

Es habitual que un programa tenga que realizar muchas veces seguidas ciertas acciones. Repetir en forma explícita secuencias de instrucciones es incómodo y aumenta el riesgo de introducir errores. Para evitarlo, los lenguajes proveen comandos que denotan repeticiones. Además, es habitual tener que recordar y modificar valores a lo largo de un programa; por ejemplo, para llevar registro de los puntos alcanzados por un jugador en un juego. En programación, las variables permiten almacenar valores, leerlos y modificarlos. Esta secuencia didáctica propone desarrollar una versión del juego Mente Maestra para introducir repeticiones y variables.

Actividad 1. Mente maestra, desenchufada

Esta actividad propone jugar grupalmente al juego Mente maestra en modo desenchufado. El juego consiste en adivinar un código secreto de cuatro posiciones, contando con varios intentos para hacerlo. La propia dinámica del juego da pie para presentar nociones de repetición.

Actividad 2. Mente maestra, parte I

Primera de tres actividades en las que se construye la aplicación Mente maestra con App Inventor. Con el objetivo de evidenciar la necesidad de contar con variables, se propone programar una versión muy simplificada del juego en la que hay un único intento para descubrir un código secreto de una sola posición.

Actividad 3. Mente maestra, parte II

En esta actividad las y los estudiantes programarán una versión del Mente maestra en la que los códigos son de cuatro posiciones y los jugadores cuentan con un único intento para descubrirlos. Aquí se presentan tanto las listas como estructuras útiles en programación como las repeticiones.

Actividad 4. Mente maestra, parte III

En esta actividad la propuesta es modificar el programa de la actividad anterior para llegar a la versión final del juego, en la que los jugadores cuentan con 8 intentos para descifrar el código secreto. Para lograrlo, tendrán que incorporar funciones en sus programas.

Datos curriculares

Nivel: Secundaria, ciclo orientado

Área: Programación

Eje: Soluciones a problemas computacionales

Contenido

- Diseño de soluciones computacionales: estrategias de solución, modularidad y legibilidad.

Eje: Representación de información en sistemas computacionales

Contenido

- Modelado: representación con datos estructurados.

Eje: Lenguajes de programación

Contenidos

- La semántica como el significado de los programas y sus partes en términos del problema que resuelve
- Herramientas de lenguaje de programación.

Objetivos de aprendizaje

- Introducir el uso de variables.
- Presentar las listas como estructura de datos.
- Utilizar repeticiones para resolver problemas.
- Introducir el uso de funciones.

Materiales necesarios

- Pizarrón.
- Hojas.
- Fibras rojas, verdes, azules y amarillas.
- Anexo "Mente maestra, la aplicación".
- Computadoras con acceso a Internet.
- Teléfonos con Android (opcional).
- Fichas para estudiantes.

*Todos los recursos necesarios para esta secuencia están disponibles en: <https://curriculum.program.ar/>
Podés buscarlos por el título de la secuencia.*

Acerca de esta iniciativa

Desde el sitio curriculum.program.ar tenemos por objetivo acompañar a la comunidad docente de habla hispana en el desafío de llevar las Ciencias de la Computación al aula.

Para ello, construimos un repositorio que reúne diversos recursos para el aula que desde la Iniciativa Program.AR de la Fundación Sadosky impulsamos desde 2013.

Organizados a partir de los saberes a promover con nuestras y nuestros estudiantes y los conceptos de la disciplina presentados en la [Propuesta curricular para la inclusión de las Ciencias de la Computación \(CC\) en el aula](#), encontrarán en curriculum.program.ar proyectos, secuencias didácticas y actividades desarrollados por una diversidad de autores y docentes en conjunto con instituciones y universidades de América Latina.

Estos materiales, que han sido desarrollados para responder a necesidades de diferentes contextos y países y que son heterogéneos en su formato y extensión, comparten un mismo propósito: integrar las Ciencias de la Computación en la escolaridad obligatoria para promover en el conjunto de los y las estudiantes la construcción de saberes que les permitan comprender, apropiarse y transformar la tecnología digital y computacional y así participar de manera crítica del mundo contemporáneo.

Cómo utilizar este recurso

Siguiendo la Propuesta curricular, es posible organizar una planificación escolar para el grado o el año a abordar y, a partir de ella, seleccionar del universo de recursos para el aula que ofrecemos los que sean adecuados al contexto y la realidad de cada grupo de estudiantes.

Al acceder a esta secuencia en el sitio curriculum.program.ar, encontrará los enlaces para descargar los materiales anexos que fueren necesarios.

Instituciones



Fuente

Banchoff, C.; Czemerinski, H.; Dabbah, J. et al. (2019). Ciencias de la computación para el aula: 2do. ciclo de secundaria. Ciudad Autónoma de Buenos Aires: Fundación Sadosky.

https://program.ar/descargas/cc_para_el_aula-2do_ciclo_secundaria.pdf



Fundación
SADOSKY

<Program.AR/>



Secuencia Didáctica 2

MENTE MAESTRA

Es habitual que un programa tenga que llevar a cabo muchas veces seguidas una misma serie de acciones. Repetir en forma explícita una secuencia de instrucciones resulta incómodo y, lo que es más grave, aumenta considerablemente el riesgo de introducir errores en los programas. Para evitar estas complicaciones, los lenguajes de programación proveen comandos que denotan **repeticiones** de instrucciones sin que haga falta reiterarlas en forma explícita.

Además, en muchas ocasiones hace falta recordar y modificar valores a lo largo de un programa; por ejemplo, en los juegos, hay que llevar registro de los puntos alcanzados por un jugador. En esta secuencia didáctica, también, se presentan las **variables**, que permiten almacenar valores en la memoria de la computadora, leerlos y modificarlos.

Con el objetivo de introducir repeticiones y variables, se proponen actividades para desarrollar una versión simplificada de Mente maestra, un juego de dos jugadores en el que uno arma un código secreto con fichas de colores que su oponente buscará descubrir.

.....

OBJETIVOS

- Introducir el uso de variables.
- Presentar las listas como estructura de datos.
- Utilizar repeticiones para resolver problemas.

.....

Actividad 1





Mente maestra, desenchufada

TODA LA CLASE

OBJETIVOS

- Ejercitar el razonamiento lógico deductivo.
- Presentar procesos repetitivos.

MATERIALES

-  Pizarrón
-  Hojas
-  Fibras rojas, verdes, azules y amarillas
-  Anexo "Mente maestra, la aplicación"

DESARROLLO

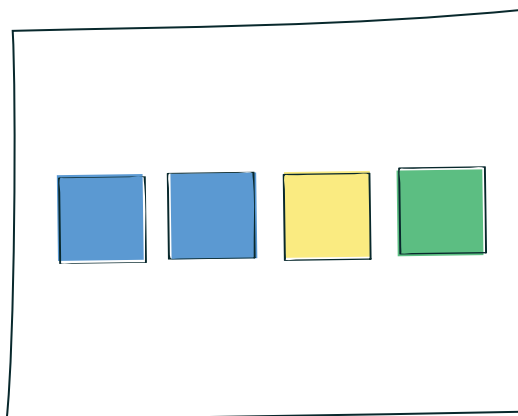
Los objetivos de esta actividad son: (i) poner en práctica mecanismos de deducción lógica y (ii) presentar procesos repetitivos. En primer lugar, se propone jugar grupalmente a una versión simplificada del juego Mente maestra (habitualmente conocido como *Mastermind*, por su nombre en inglés) en modo desenchufado –sin dispositivos electrónicos–. Este juego permite ejercitar el razonamiento lógico deductivo. A continuación, se indagará sobre los mecanismos que gobiernan la dinámica del juego, lo que dará pie para presentar nociones de repetición.

Parte 1: Mente maestra desenchufada

Para que comprendan la dinámica del juego, les propondremos a los estudiantes jugar al Mente maestra grupalmente. Se trata de un juego de mesa de dos jugadores en el que uno arma un código secreto con cuatro fichas de colores y su oponente busca descubrirlo. Nosotros elegiremos el código y los estudiantes asumirán el rol del contrincante, con el fin de descifrarlo en la menor cantidad de intentos posible.

Primer paso: selección del código secreto

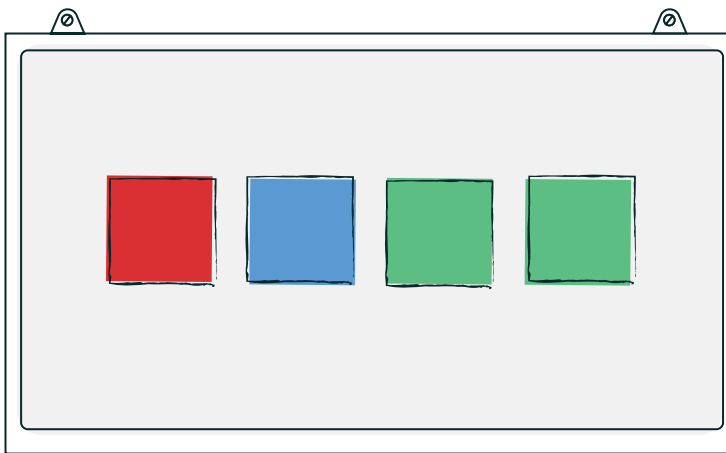
Para dar comienzo al juego elegimos un código de longitud cuatro, al que a cada posición le asignamos un color entre azul, rojo, amarillo y verde, y lo anotamos en una hoja sin que ningún estudiante pueda observarlo. Algunos ejemplos son: [verde, verde, rojo, azul], [amarillo, rojo, azul, verde] y [rojo, rojo, rojo, rojo]. Como el código tiene 4 posiciones y para cada una hay cuatro opciones de color, existen 256 códigos posibles ($4^4 = 256$). A modo de ejemplo, supondremos en la explicación que hemos elegido [azul, azul, amarillo, verde].



Hoja con el código secreto de cuatro colores

Segundo paso: intento de quebrar el código

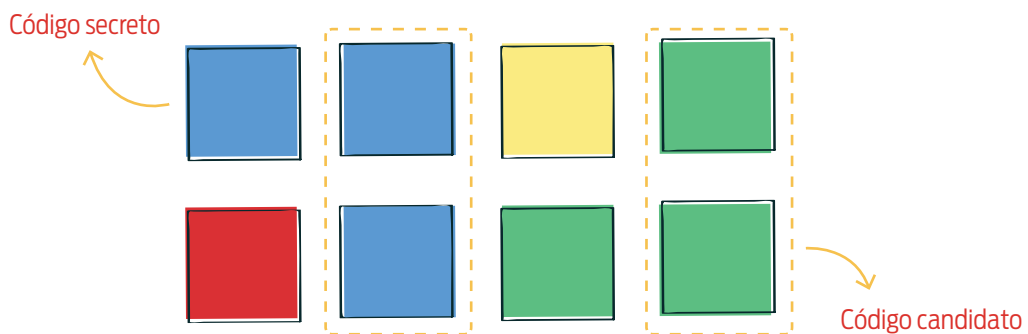
Le pedimos a algún estudiante que intente descubrir el código. Como a esta altura no cuenta con ninguna información sobre nuestra elección, simplemente intentará adivinarlo eligiendo una clave al azar. Una vez que la haya dicho, la copiamos en el pizarrón. Siguiendo con el ejemplo, supondremos que su tentativa es [rojo, azul, verde, verde].



Código de cuatro colores propuesto por un estudiante

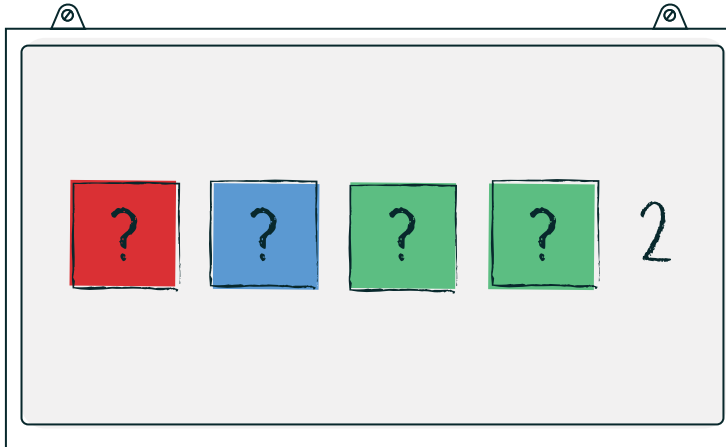
Tercer paso: proveemos *feedback*

En esta instancia les informamos a los estudiantes la cantidad de posiciones de la clave propuesta que coinciden con el código secreto elegido por nosotros, sin revelar cuáles son los aciertos –en el ejemplo, el azul de la segunda posición y el verde de la cuarta–: “Hay dos coincidencias”.



Dos coincidencias

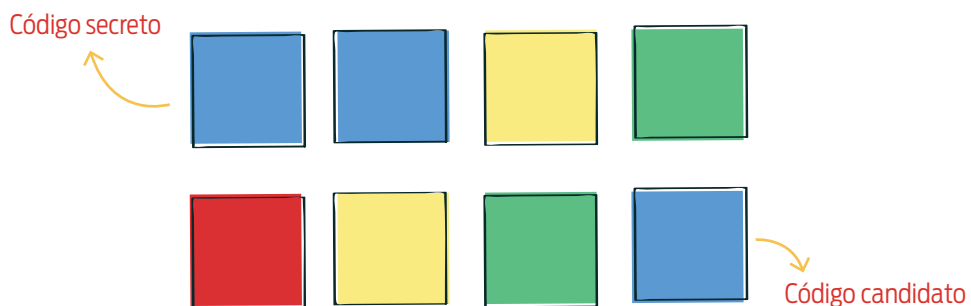
Seguidamente, anotamos el número de coincidencias en el pizarrón. Además, como no hay certeza sobre cuáles son las posiciones adivinadas, colocamos sobre todas un signo de interrogación.



Se copia en el pizarrón la cantidad de coincidencias

De aquí en adelante, se repiten los pasos dos y tres: los estudiantes arriesgan códigos y nosotros informamos la cantidad de aciertos. La diferencia con lo anteriormente descrito es que, en los siguientes intentos, pueden usar la información recabada en las manos previas. El objetivo es descubrir el código en la menor cantidad de intentos posibles.

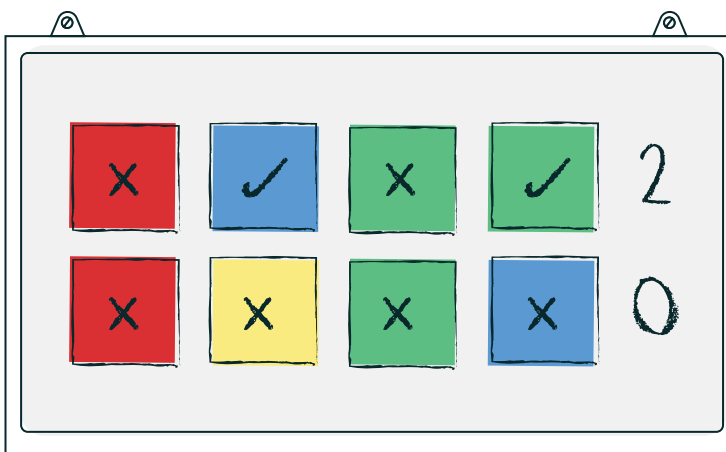
Continuando el ejemplo, los estudiantes podrían especular (erróneamente) que los dos aciertos corresponden a las posiciones 1 y 3 y arriesgar el código [rojo, amarillo, verde, azul] (solo modifican los colores de las posiciones 2 y 4 respecto de su intento anterior). Entonces, informamos que allí no hay ninguna coincidencia.



No hay coincidencias

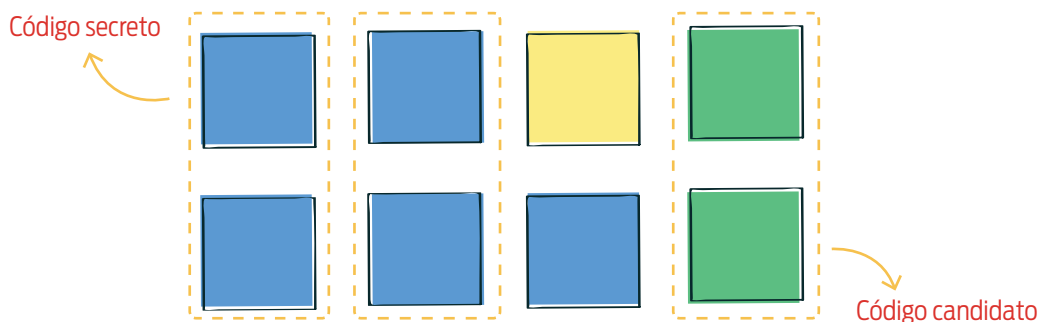
En esta instancia los estudiantes pueden empezar a usar el *feedback* recibido para deducir algunos colores del código secreto y descartar otros. Razonamos con ellos: “En primer lugar, como en el primer intento hubo dos coincidencias y en el segundo ninguna, los aciertos de la primera propuesta corresponden necesariamente las dos posiciones que modificaron en la segunda. Es decir, la segunda posición del código secreto es azul y la cuarta verde. En segundo lugar, también saben que la primera posición no es roja y la tercera no es verde”.

Al copiar nuevos códigos en el pizarrón, borramos los signos de interrogación y marcamos las posiciones de las que se tiene certeza sobre su color y las opciones descartadas, de forma que los estudiantes fijen su atención en lo que falta descubrir.

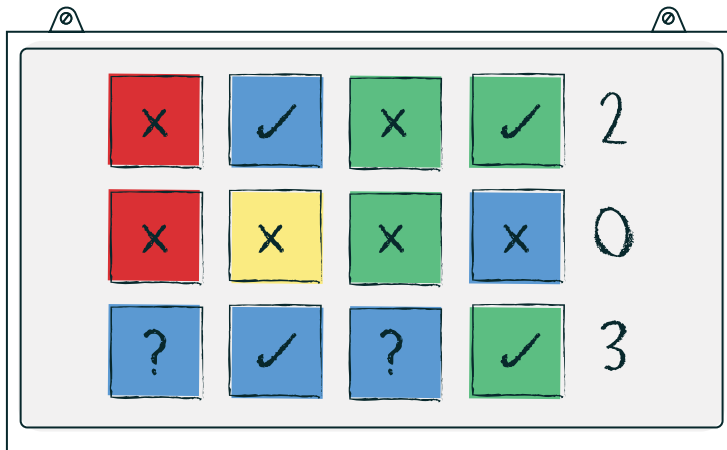


Se informa la cantidad de coincidencias

A continuación podrían arriesgar, por ejemplo, el código [azul, azul, azul, verde]. En este caso, hay tres coincidencias. Sin embargo, no pueden aún concluir si la tercera coincidencia es el azul de la primera posición o el azul de la tercera.

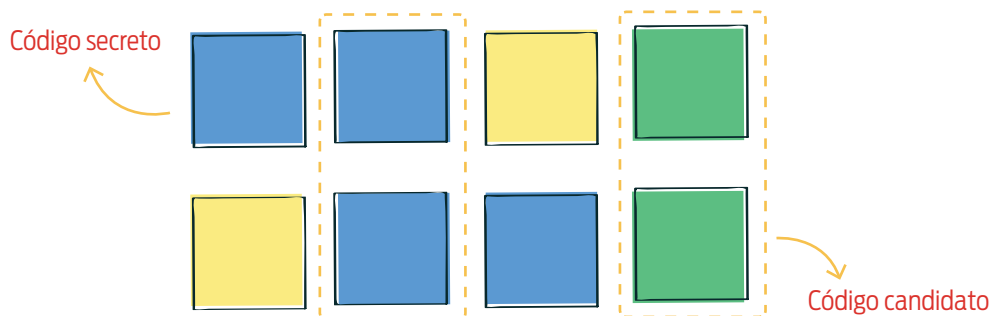


Tres coincidencias



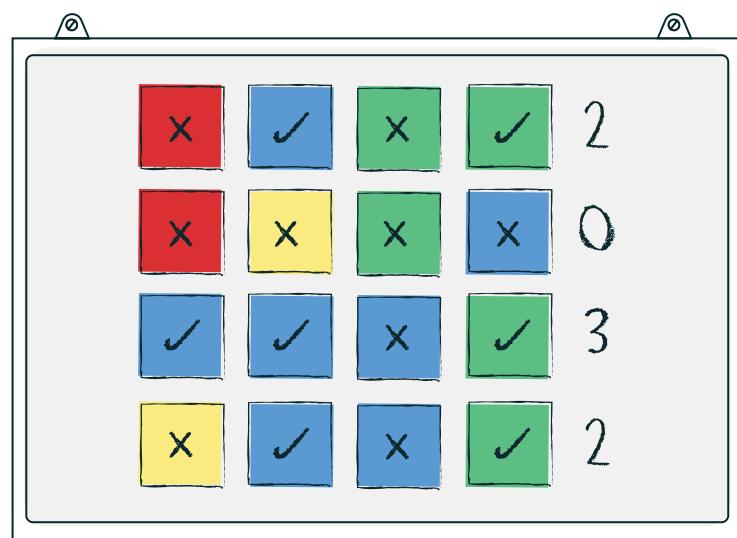
Se informa la cantidad de coincidencias

A partir de la información disponible podrían especular con que la tercera posición es azul y proponer [amarillo, azul, azul, verde], obteniendo dos coincidencias.



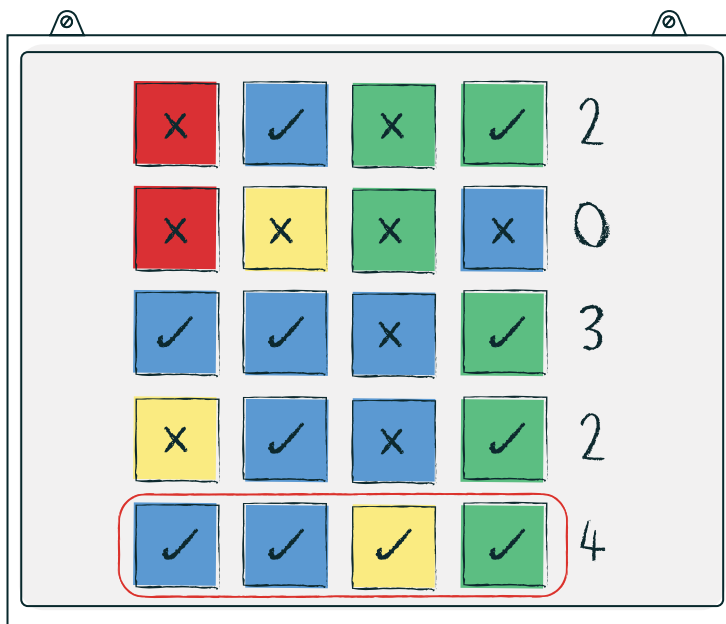
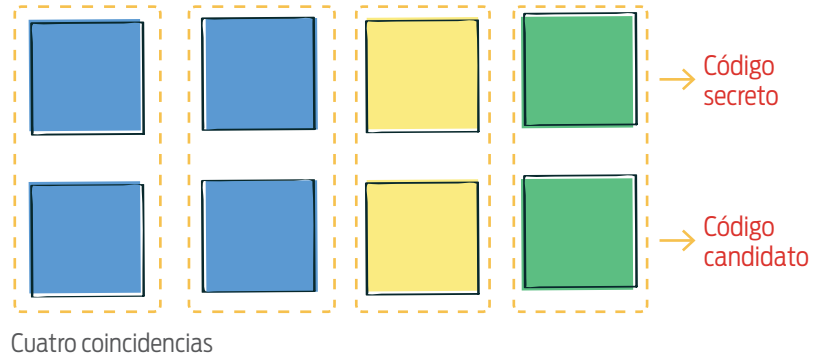
Dos coincidencias

Razonamos con ellos: “Como en este intento hubo dos coincidencias y son las que ya sabían de antemano –la posición dos azul y la cuatro verde–, la tercera coincidencia del intento previo es necesariamente el azul de la posición uno –el único que cambió entre las últimas dos propuestas–. Además, la tercera posición no puede ser ni verde –ya lo sabían– ni azul”.



Se informa la cantidad de coincidencias

Solo resta conocer el color de la tercera posición. Las únicas dos opciones son que sea roja o amarilla. Con un poco de fortuna, en esta ocasión podrían proponer el código [azul, azul, amarillo, verde], logrando descifrar el código en el quinto intento.



El código es descubierto

Los invitamos a que, en parejas, jueguen algunas partidas de Mente maestra. Después, les entregamos el anexo “Mente maestra, la aplicación” y les contamos que allí se describe una aplicación que ellos construirán luego de completar una serie de actividades.

Parte 2: Repeticiones

Comenzamos la segunda parte de la actividad simulando recordar una anécdota: “Tengo un amigo que tiene cinco hijos con un año de diferencia entre el más grande y el segundo, un año entre el segundo y el tercero y así también con los otros. ¡Cinco años seguidos tuvo un hijo! No recuerdo muy bien a qué venía, pero el otro día me comentó que cuando sus hijos eran chiquitos, hacía lo siguiente: **para cada hijo**, seleccionaba la ropa, lo despertaba, lo vestía y lo mandaba a lavarse los dientes. Repetía esto 5 veces, una vez por hijo. Qué trabajo, ¿eh?”.

Continuamos: “Volvamos a lo nuestro”. Les proponemos a los estudiantes que imaginen que en un partido de Mente maestra ellos son los que escriben el código secreto y preguntamos: “Una vez que su

oponente arriesga un código, ¿cómo harían para proporcionarle *feedback*?”. Les damos unos instantes para que lo piensen y escuchamos sus ideas. Es posible que aparezcan respuestas tales como “les decimos cuántas son iguales”, “contamos la cantidad de coincidencias” o similares. “Es cierto, contamos la cantidad de coincidencias. Detengámonos aquí y hagamos el ejercicio de desmenuzar el proceso de conteo. ¿Qué hacemos primero?”. Guiamos el intercambio para concluir que lo primero que hacemos, es mirar si el color de la primera posición del código propuesto coincide con el color de la primera posición de nuestro código y que, de ser así, la consideramos para el conteo. A continuación, copiamos en el pizarrón el mecanismo enunciado.

Si el color de la primera posición coincide con el color de la primera posición de nuestro código, entonces la tenemos en cuenta en el conteo.

Análisis de la primera posición

Luego, comentamos: “Aquí hay una estructura que podemos identificar. Primero aparece la palabra *Si*, luego enunciamos una condición que puede ser cierta o falsa, a continuación la palabra *entonces* y, finalmente, lo que hacemos en caso de que la condición sea verdadera. Como ya saben, esta es una alternativa condicional”. Agregamos en el pizarrón la descripción para las tres posiciones restantes.

Si el color de la primera posición coincide con el de la primera posición de nuestro código, entonces la tenemos en cuenta en el conteo.

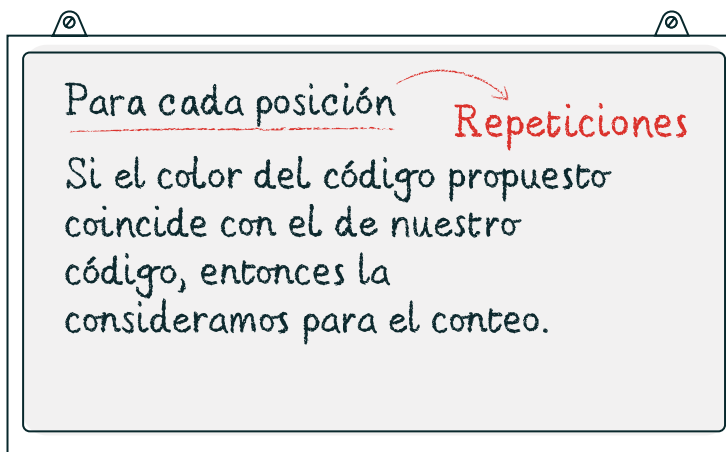
Si el color de la segunda posición coincide con el color de la segunda posición de nuestro código, entonces la tenemos en cuenta en el conteo.

Si el color de la tercera posición coincide con el color de la tercera posición de nuestro código, entonces la tenemos en cuenta en el conteo.

Si el color de la cuarta posición coincide con el color de la cuarta posición de nuestro código, entonces la tenemos en cuenta en el conteo.

Alternativas condicionales

A continuación indagamos a los estudiantes: “Como pueden observar, estamos haciendo cuatro veces cosas análogas, ¿no es cierto? ¿Se les ocurre cómo podrían escribir lo mismo de forma más sintética?”. Escuchamos sus respuestas y, a medida que las enuncian, las analizamos grupalmente. En caso de que no haya surgido, copiamos en el pizarrón la siguiente:



Repeticiones

Preguntamos: “¿Qué diferencias y similitudes encuentran entre lo que escribí antes y lo que escribí ahora?”. Guiamos la discusión para concluir que, aunque se usaron expresiones distintas, ambas connotan que para hacer el conteo solo se consideran las posiciones que comparten el color. Nos aseguramos de que todos hayan comprendido la equivalencia y redondeamos: “Así, obtenemos una forma más concisa de describir el proceso de conteo de coincidencias, ¿no? ¡Imagínense lo que sería escribirlo de la forma anterior si los códigos tuviesen mil posiciones!”.

Finalmente les decimos: “En definitiva, inspeccionamos una colección de elementos –las posiciones de los códigos– y con cada uno de ellos hicimos lo mismo –preguntarnos si había que considerarlos para el conteo de coincidencias–. A este proceso en el que se hace varias veces lo mismo se lo conoce como **repetición**”.

CIERRE

Les comentamos a los estudiantes que, en general, los lenguajes de programación proveen instrucciones que permiten incorporar en nuestros programas repeticiones. Estas resultan muy útiles cuando hay que ejecutar muchas veces una misma secuencia de instrucciones.

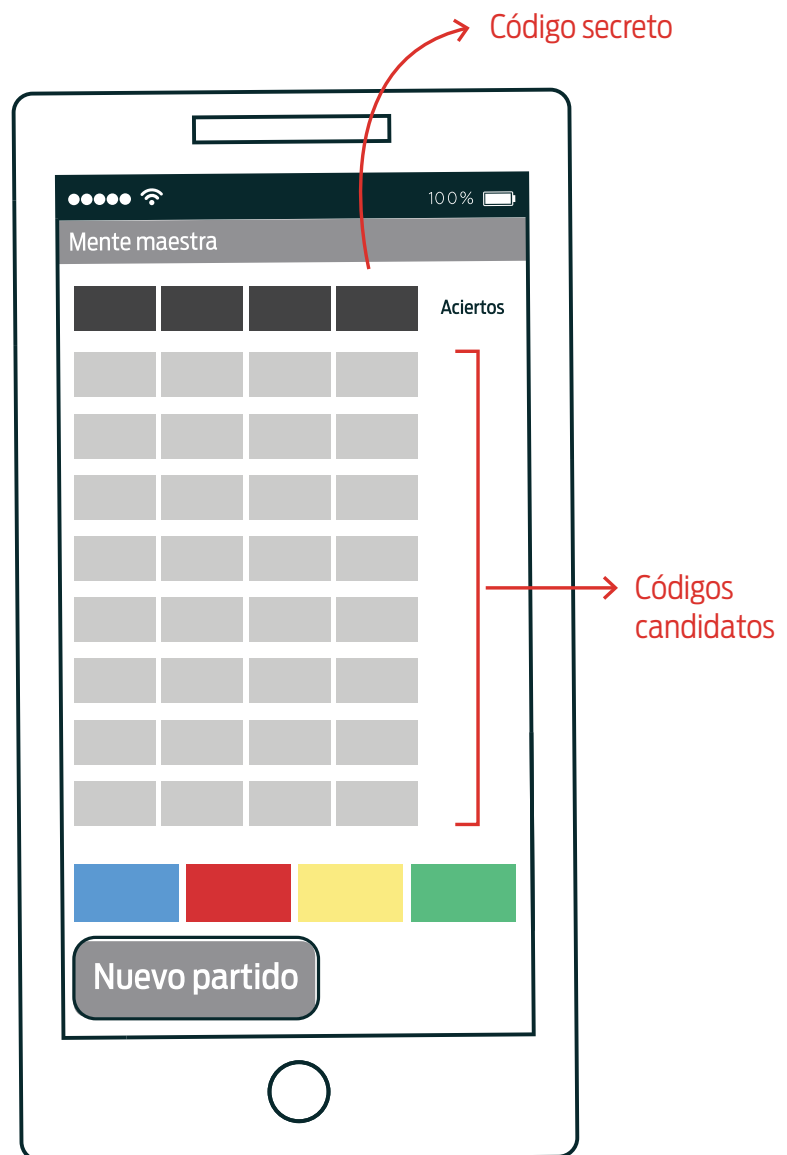
MENTE MAESTRA, LA APLICACIÓN

¡Mirá esta aplicación del juego Mente maestra! En esta versión, el jugador tiene que estar muy atento y concentrado porque, para adivinar el código secreto, cuenta solo con ocho intentos.

ANEXO

¡Comienza el juego!

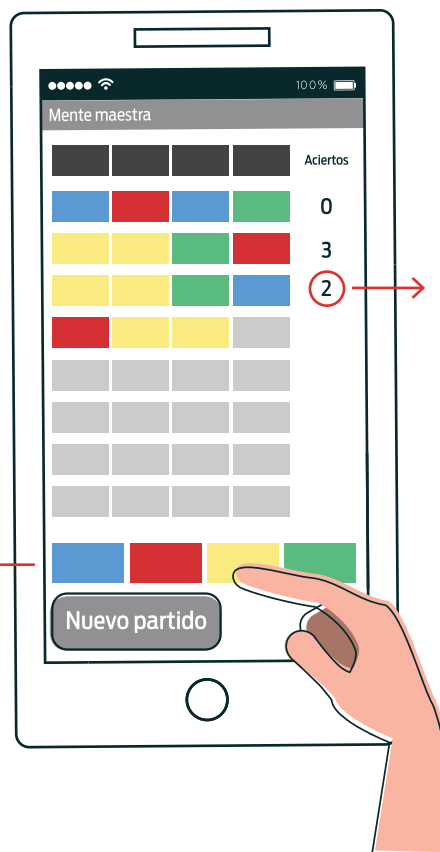
Esto es lo que se ve en la pantalla apenas la aplicación comienza su ejecución. Los rectángulos en gris oscuro de arriba ocultan el código secreto; y cada una de las ocho líneas siguientes mostrará cada uno de los intentos del jugador por descubrirlo.



Probando, probando, probando...

Para seleccionar los colores de los códigos candidatos, el usuario presionará los botones azul, rojo, amarillo y verde. A medida que lo haga, irán apareciendo los colores elegidos en los rectángulos reservados para los ocho ensayos. Además, cuando completan un intento, a la derecha se mostrará la cantidad de coincidencias entre el código ingresado y el código secreto.

Botones para ingresar códigos candidatos

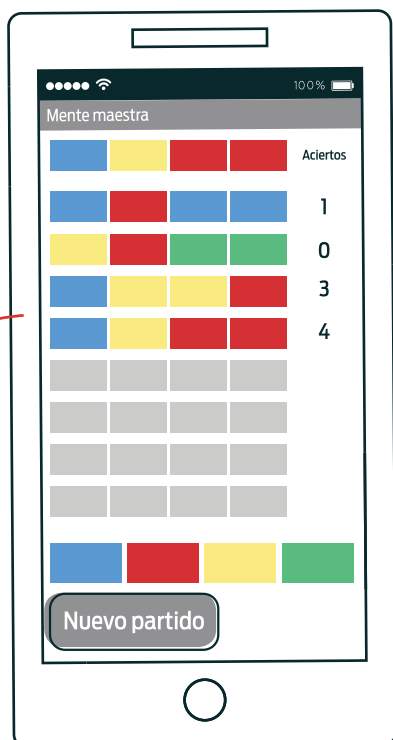


Número de coincidencias entre el código secreto y el código candidato

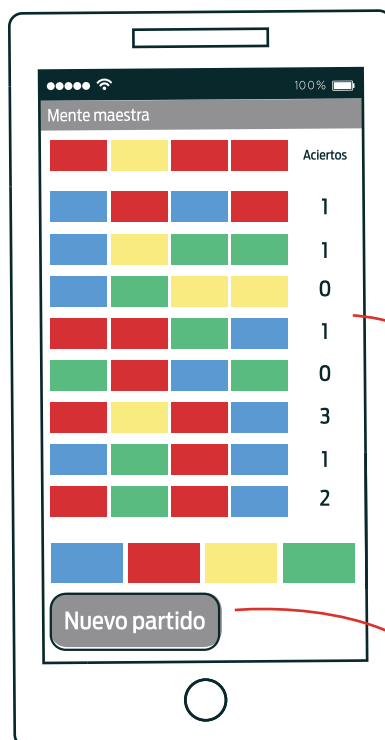
Y en un momento se acaba

El juego finaliza tanto cuando el código secreto es descubierto como cuando se agotan las ocho posibilidades. Por supuesto, en el primer caso el jugador gana y, en el segundo, pierde. Igual, este juego da revancha: presionado el botón *Nuevo partido* todo comienza otra vez.

El jugador gana



El jugador pierde



Para empezar de nuevo

Actividad 2

Mente maestra, parte I

DE A DOS

OBJETIVOS

- Introducir el uso de variables.
- Diferenciar variables locales y globales.

MATERIALES



Computadora



Internet



MIT App Inventor 2



Ficha para estudiantes



Teléfono con Android (opcional)

DESARROLLO

Esta es la primera de una serie de actividades para desarrollar una aplicación para jugar al juego Mente maestra. La división en actividades propuesta se debe a que, para completar el programa, hay que incorporar varios temas importantes de programación y, en cada actividad, se hace foco en algunos de ellos.

El objetivo de este primer ejercicio es poner de manifiesto la necesidad de usar **variables** al programar. Para ello, los estudiantes construirán un programa para jugar a una versión muy simplificada del Mente maestra: los códigos son de una sola posición –en lugar de cuatro– y los jugadores cuentan con un único intento para descubrirlos. Se trata, en definitiva, de adivinar un color que el programa elige al azar al comenzar su ejecución. Al completar la propuesta, los estudiantes diferenciarán las variables **locales** de las **globales**, advirtiendo cuándo es conveniente usar unas y cuándo las otras.

Es probable que los estudiantes esbocen distintas propuestas para completar la consigna. En el desarrollo de la actividad presentamos y analizamos una de ellas. Sin embargo, cualesquiera sean los programas que elaboren, es importante que (i) usen procedimientos para descomponer problemas en subproblemas más simples y para evitar las redundancias, introduciendo parámetros en los casos que sean necesarios, y (ii) que incorporen el uso de variables locales y globales.¹

Consigna 1: interfaz gráfica

Comenzamos contándoles a los estudiantes que esta es la primera de una serie de actividades para desarrollar una aplicación para jugar al juego Mente maestra. Luego, les repartimos la ficha y los invitamos a que resuelvan la primera consigna. Allí se dan instrucciones para armar la interfaz gráfica de la aplicación. La tabla a continuación muestra los valores de las propiedades que hay que cambiar (en relación a los que App Inventor asigna por defecto) para que la aplicación se vea tal como en la siguiente imagen.

¹Una versión completa de la aplicación se encuentra disponible en la galería de proyectos de App Inventor del usuario “programar2020”.

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto	Etiqueta	Color de fondo	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato	Etiqueta	Color de fondo	Ninguno	Gris Claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

Valores diferentes a los asignados por defecto

Consigna 2: generar el código secreto

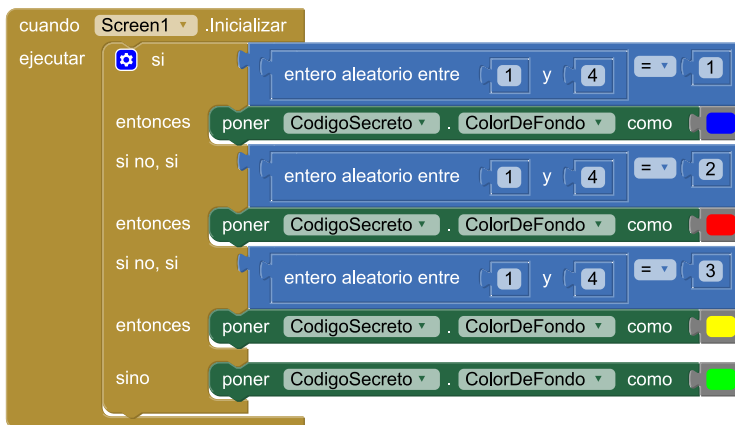
La segunda consigna propone que, al iniciar la aplicación, se genere el código secreto que luego se buscará descubrir. Por tratarse de un código de una posición, el problema se reduce a seleccionar al azar uno de los cuatro colores.

Lo primero que deben reconocer los estudiantes es que tienen que usar el bloque `cuando Pantalla.Inicializar / ejecutar { }` para manejar el evento que se produce cuando la aplicación comienza a ejecutarse. Además, como solo hay cuatro opciones de color –azul, rojo, amarillo y verde–, es esperable que intuyan que para resolverlo tendrán que generar un número al azar entre 1 y 4.



Bloques para resolver la consigna

Algunos estudiantes podrían proponer un programa como el de la figura (o similar, asociando de otra forma los números con los colores). La propuesta contiene dos problemas fundamentales: el modo de selección de color y lo que hace una vez que este fue seleccionado.¹



Programa con errores

Como puede observarse, para seleccionar uno de los cuatro colores se generan 3 números al azar. De este modo, no todos los colores tienen la misma probabilidad de ser elegidos –i.e., con probabilidad $\frac{1}{4}$ –.² Les comentamos: “En realidad, lo que hay que hacer es generar un único número y, de acuerdo al resultado, definir el color secreto. Por ejemplo, si se genera el 1 el color es azul, si se genera el 2 es rojo, etc. Pero

¹ Aunque aquí se presentan juntos, podría ocurrir que haya propuestas que tengan solo uno de los problemas descritos; en tal caso, el programa se puede corregir con las modificaciones propuestas para el error en cuestión.

² Si bien aconsejamos no realizar un análisis probabilístico con los estudiantes, se les puede mencionar que de este modo solo el color azul tiene un 25% de probabilidades de ser escogido; la probabilidad de escoger el rojo es de 19% (aprox.), el amarillo 14% (aprox.) y el verde 42% (aprox.).

recuerden: generando un único número”. Les damos tiempo para que, autónomamente, exploren el entorno y ensayen alternativas para resolver el problema al que se enfrentan.

La solución al problema planteado requiere el uso de una variable. Por tratarse de un concepto complejo, es esperable que haya estudiantes que no comprendan por sí solos qué son las variables ni cómo se utilizan. En ese caso se puede hacer una revisión grupal, buscando en todo momento que los estudiantes infieran, a partir de los obstáculos que presenta el problema, la necesidad de que un programa “recuerde” valores que luego necesitará recuperar. A continuación, un apartado sobre las variables y, más adelante, la continuación del desarrollo de la actividad.

Sobre las variables

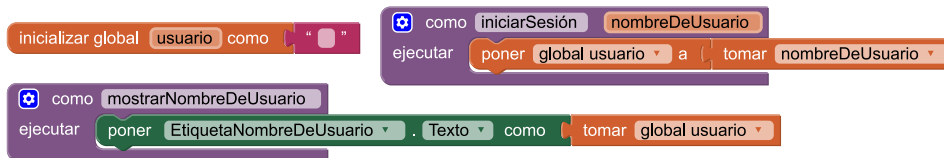
Una variable es un nombre –que elegimos nosotros al programar– que denota un espacio de la memoria de la computadora en la que se puede guardar un valor para, luego, recuperarlo o modificarlo. Las variables permiten, por ejemplo, que los juegos registren la cantidad de puntos que alcanza un jugador a medida que el juego avanza. Como siempre, es conveniente que el nombre que escojamos refleje su propósito. Siguiendo con el ejemplo de los puntos que obtiene un jugador, podría llamarse *puntos*. En App Inventor, los bloques para manipular variables se encuentran en *Bloques > Integrados > Variables*.



Bloques para manipular variables

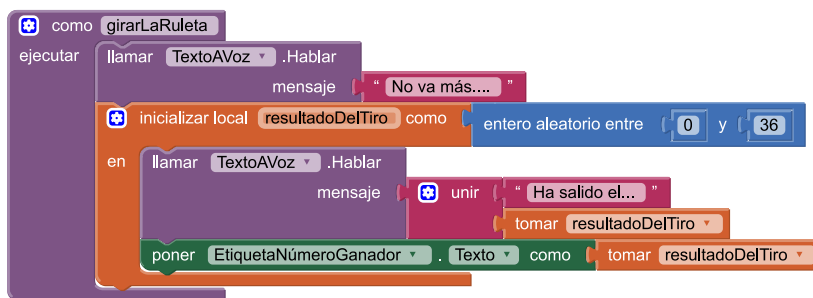
Existen dos tipos de variables: locales y globales. Una variable global puede ser referenciada en cualquier punto de un programa –por ejemplo, en distintos procedimientos, en los bloques para manejar eventos, etc.–. Para crear una hay que usar el bloque `inicializar global (nombre) como []` y, allí, asignarle un valor. Este valor es el que adquiere la variable ni bien la aplicación comienza a ejecutarse. Luego, para leer y modificar su contenido hay que usar, respectivamente, los bloques `tomar (nombre)` y `poner (nombre) a []`. Para cambiarle el nombre, alcanza con hacer clic sobre *nombre* y tipear uno nuevo.

En la imagen siguiente se observa un ejemplo en el que se crea una variable `usuario` cuyo valor inicial es un texto vacío, un procedimiento `iniciarSesión (nombreDeUsuario)` que modifica su valor; y, por último, otro `mostrarNombreDeUsuario` que lee su contenido y lo muestra como texto de una etiqueta.



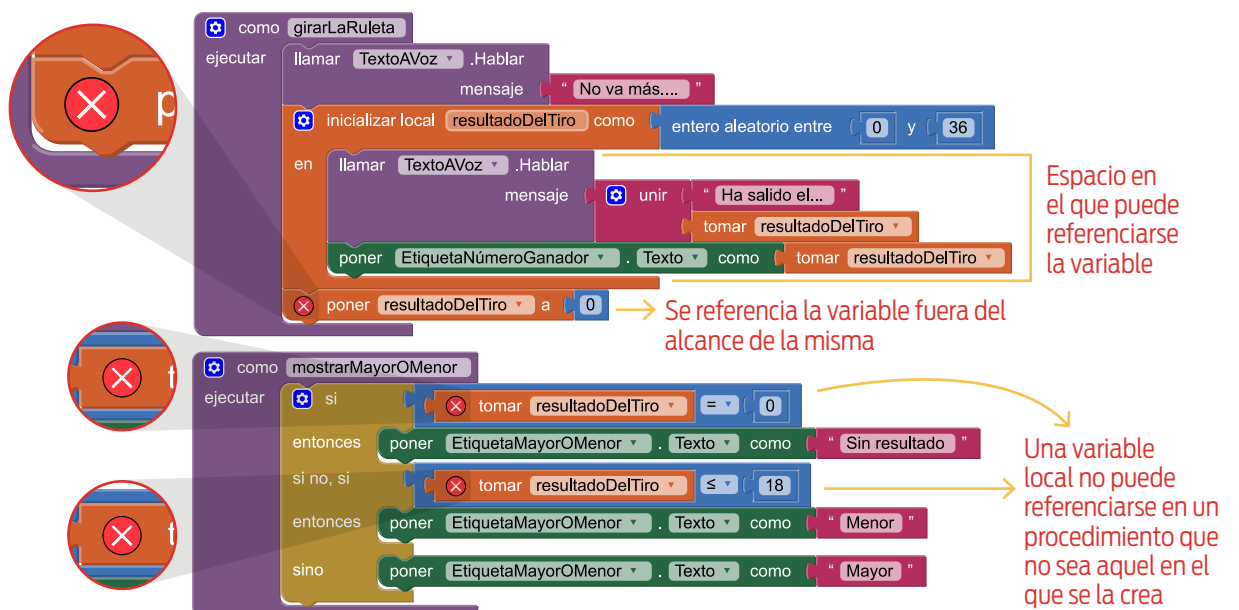
Definición, escritura y lectura de una variable global

A diferencia de las variables globales, las variables locales solo pueden referenciarse en un espacio restringido del programa. Para inicializar una variable de este tipo se usa el bloque `inicializar local (nombre) como [] / en { }`; luego, solo puede actualizarse o leerse su valor dentro del espacio delimitado por `{ }`. A continuación, se muestra un ejemplo con un procedimiento que simula una tirada de una ruleta y, luego de obtener un valor, usa el sintetizador de voz del teléfono para que la aplicación lo anuncie y, además, lo muestra como texto de una etiqueta.



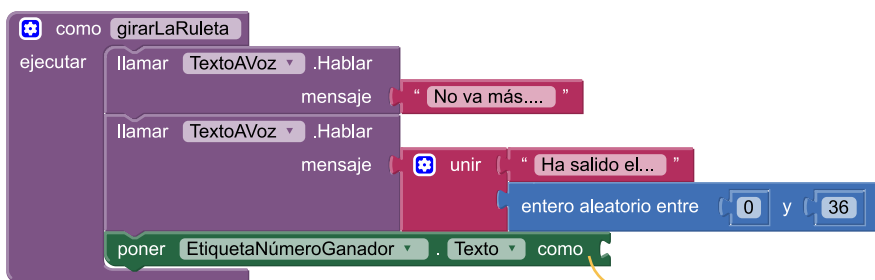
Definición y lectura de una variable local

Al intentar referenciar una variable local fuera de su campo de aplicación, App Inventor muestra que se está en presencia de un error.



Variables locales fuera de su campo de aplicación

A modo de ejemplo, se presenta un análisis de por qué hace falta una variable para producir el efecto buscado en el caso de la simulación de una tirada de la ruleta. Suponiendo que no se use una, hay que generar el número aleatorio en el primer momento en el que se lo utiliza –en este caso, cuando el número es anunciado por el sintetizador de voz–. Sin embargo, como el valor no puede recuperarse de ningún lado, no es posible mostrarlo en la etiqueta. El problema surge al querer utilizar más de una vez un valor sin previamente haberlo conservado.



Se debe usar una variable

No se puede completar el programa

SIEMPRE ES MÁS SEGURO UTILIZAR VARIABLES LOCALES QUE GLOBALES

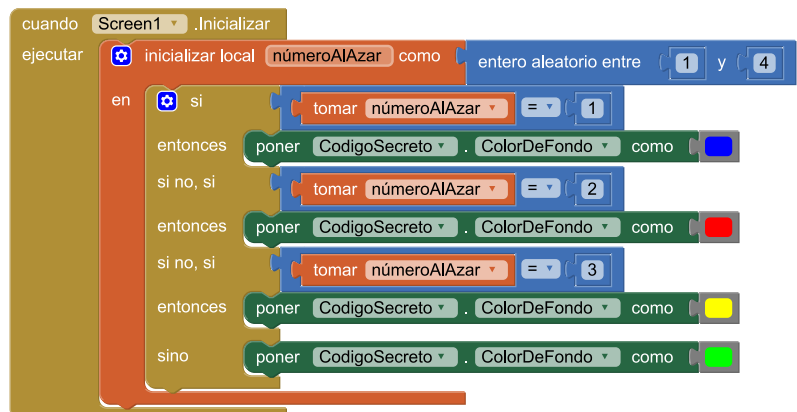
Al ser accesibles desde cualquier punto del programa, es difícil tener control sobre lo que pasa con las variables globales y poder hacer asunciones sobre su valor. Esto se potencia cuando la ejecución de distintas porciones de un programa se dispara por acciones del usuario –como sucede con App Inventor– que, *a priori*, no se sabe cuándo van a ocurrir. Las variables globales hay que usarlas únicamente cuando sea necesario acceder a ellas en distintas partes del programa.

Por el contrario, las variables locales tienen un ámbito de aplicación restringido a una pequeña porción del programa, lo que permite que, al momento de programar, sepamos cómo evolucionará su valor cuando la aplicación se encuentre en ejecución.



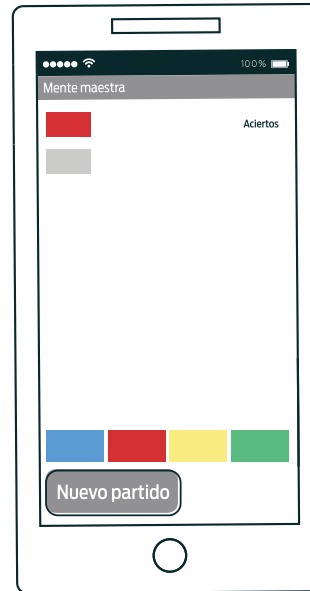
Consigna 2: generar el código secreto (continuación)

El problema que se produce al generar tres números aleatorios distintos para la selección de un color puede resolverse usando una variable local: se genera un único número y, luego, se consulta su valor. De este modo, los cuatro colores tienen la misma probabilidad de ser seleccionados.



Se genera un único número al azar

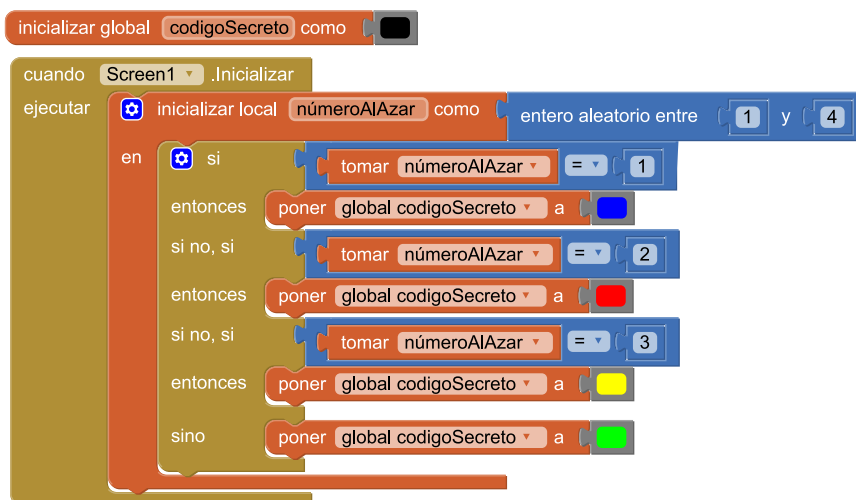
El segundo problema de la propuesta es que el color secreto, una vez definido, se lo establece como color de fondo de la etiqueta que representa la incógnita. De este modo, en lugar de conservarse oculto, el color queda expuesto ni bien la aplicación comienza a ejecutarse. Si algún estudiante propone un programa con esta característica, le sugerimos que lo corra; así, podrá observar cómo se manifiesta el error y pensar algún modo de corregirlo.



En esta primera aproximación al Mente maestra, la interacción entre la aplicación y el usuario involucra tres pasos: (i) el programa selecciona un color al azar, (ii) el usuario presiona uno de los cuatro botones intentando adivinar el color secreto, y (iii) el programa compara ambos colores y muestra el resultado. Por lo tanto, el color seleccionado aleatoriamente cuando comienza la ejecución de la aplicación debe ser recordado para evaluar, tiempo más tarde, cómo le fue al jugador. En este caso, sí, hace falta una variable global: hay que conservar un valor generado en un lugar –donde se maneja el evento que se produce cuando comienza la ejecución– y usarlo en otro –donde se maneja el evento que se produce cuando el usuario selecciona un color–.

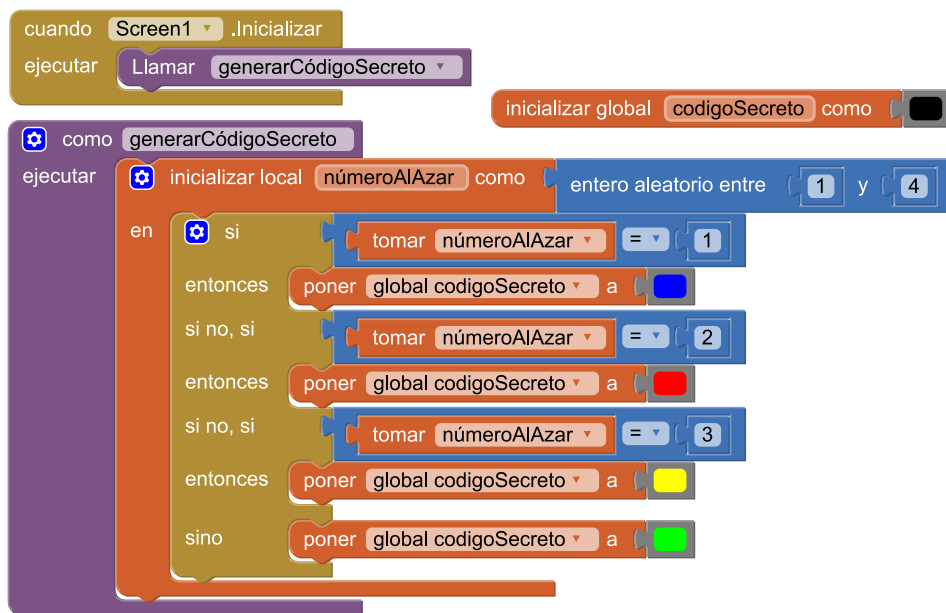
Al iniciar la aplicación el código secreto queda expuesto

En la imagen se exhibe una posible solución del desafío, que incluye una variable global: `codigoSecreto`. Si bien se inicializa con el color negro, podría inicializarse con cualquier otro valor, pues apenas la aplicación comienza su ejecución el negro se reemplaza por el color seleccionado al azar.



Solución sin procedimientos

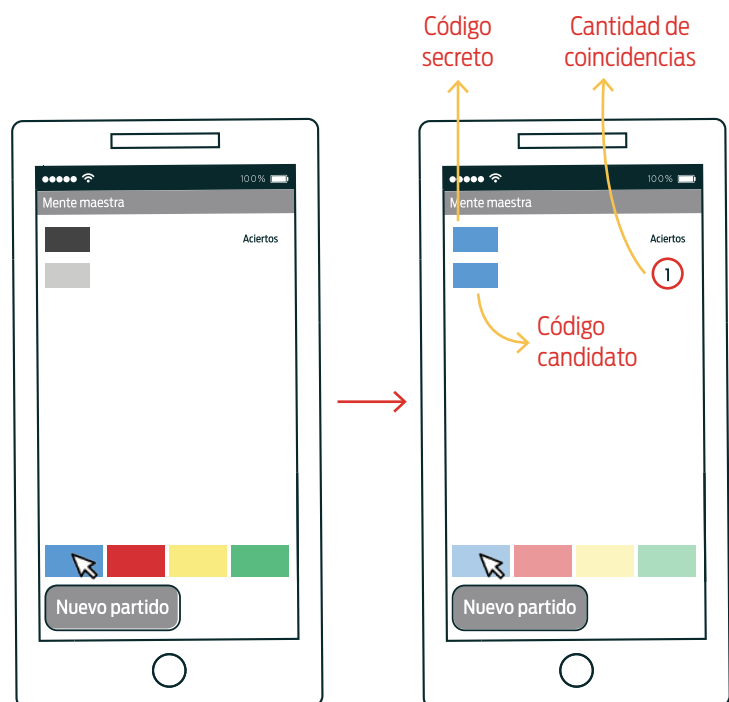
Es importante insistirles a los estudiantes sobre la ventaja de usar procedimientos (y escoger cuidadosamente sus nombres) para construir programas claros y fácilmente entendibles. A continuación se muestra una solución alternativa que define un procedimiento, en la que se aprecia a simple vista que, al iniciar la ejecución, la aplicación genera el código secreto; además, cómo hace para generarlo, queda encapsulado en un procedimiento separado.



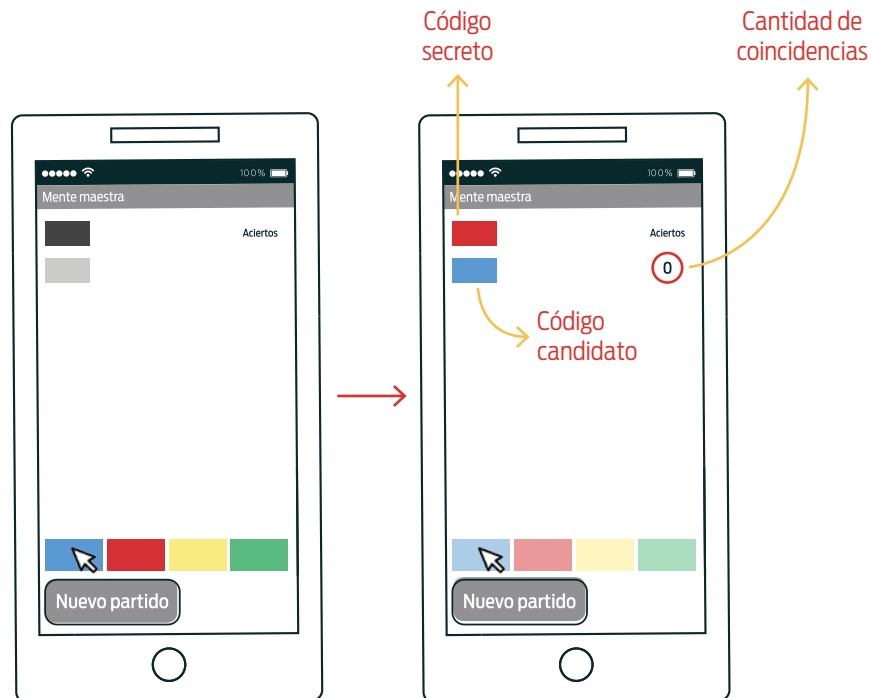
Solución con un procedimiento

Consigna 3: manejar el evento del botón azul

La tercera consigna pide que, luego de que el usuario presione el botón azul, se muestre si hay o no una coincidencia entre la elección del jugador y el color secreto. A continuación se muestra la respuesta que se espera del programa, tanto si hay coincidencia como si no, en un caso asumiendo que el color secreto es azul y en el otro que es rojo.



Hay coincidencia



No hay coincidencia

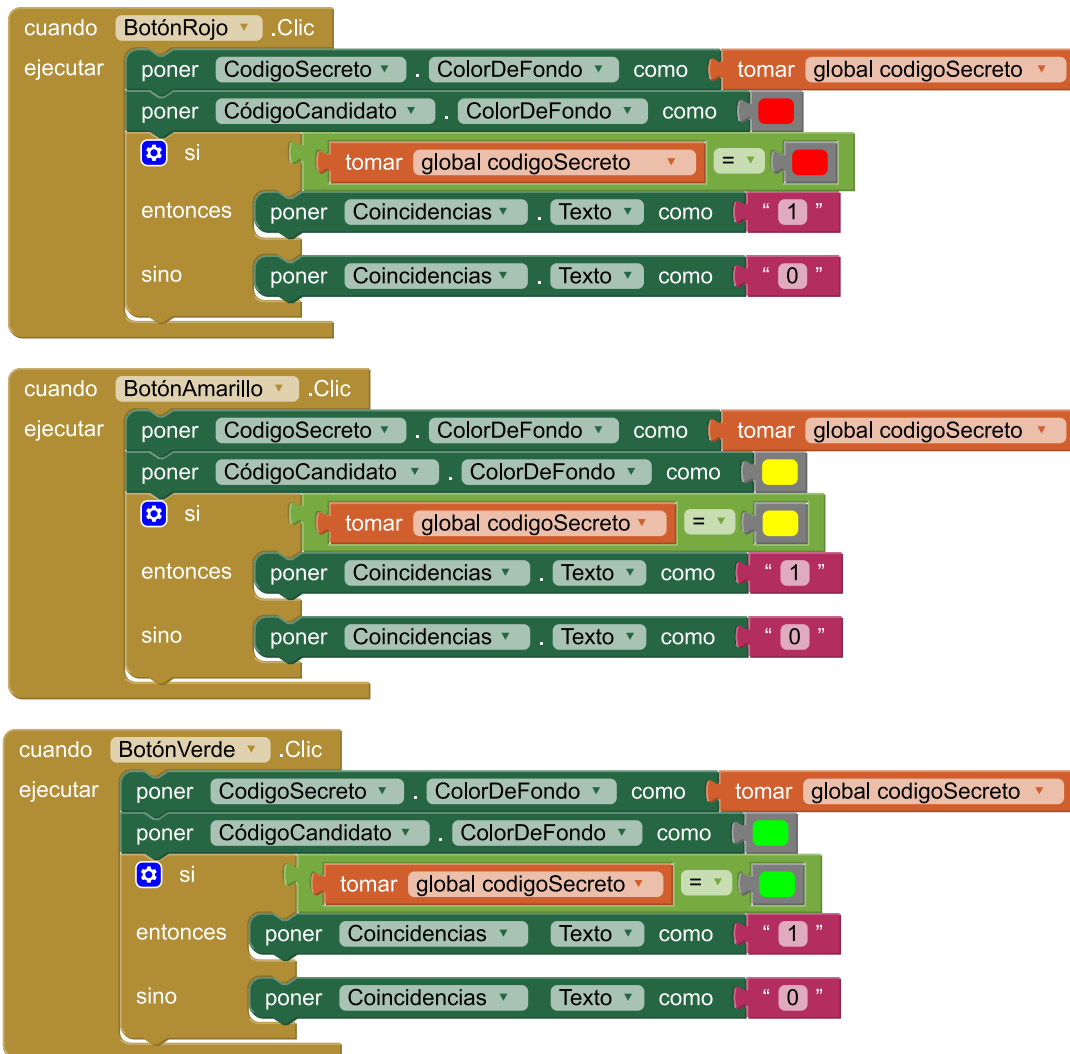
Cuando se presiona el botón azul hay que: (i) mostrar el color del código secreto, (ii) mostrar el color elegido por el usuario –azul–, y (iii) indicar la cantidad de coincidencias entre ambos códigos –1 si coinciden y 0 si no–. La figura muestra una posible solución.



Posible solución de la tercera consigna

Consigna 4: manejar el evento de los botones rojo, amarillo y verde

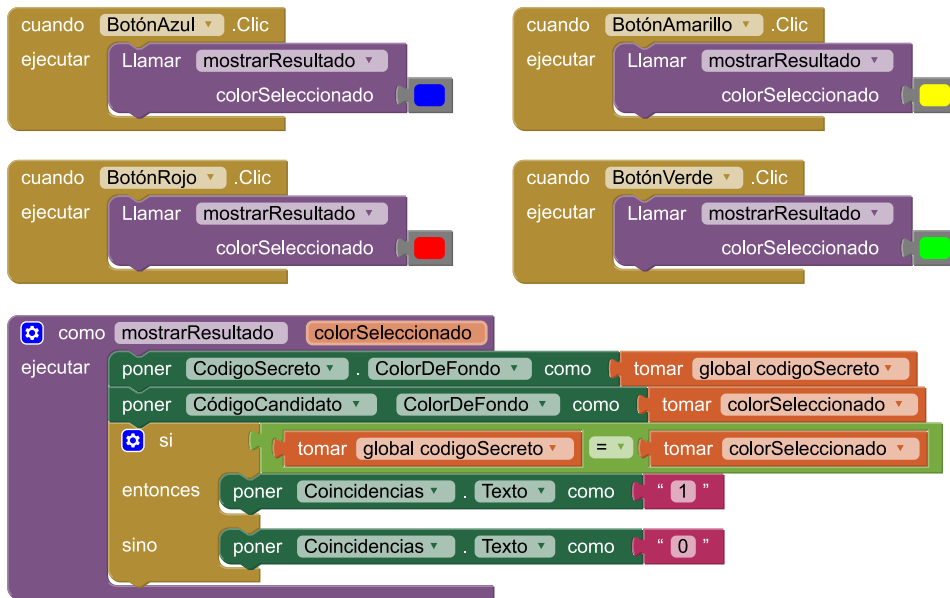
La cuarta consigna pide incorporar el manejo de los eventos que se producen al apretar los botones rojo, amarillo y verde. Podría ocurrir que algunos estudiantes propongan un programa en el que el manejo de estos tres eventos sea muy similar a lo programado para el botón azul, cambiando en cada uno el color considerado.



Fragmentos similares del programa

Si bien el programa es funcionalmente correcto, no es una solución cualitativamente buena. A los estudiantes que optasen por un programa con esta característica, les sugerimos que piensen algún programa alternativo.

La presencia de fragmentos muy parecidos sugiere que es conveniente incorporar un procedimiento en el que el color seleccionado por el usuario sea un parámetro. A continuación, se da una solución de la consigna que incorpora el procedimiento `mostrarResultado (colorSeleccionado)`.



Solución de la cuarta consigna

Consigna 5: deshabilitación de la botonera de colores

La cuarta consigna propone que, una vez que el usuario arriesga una alternativa y se devela el color secreto, se deshabiliten los cuatro botones de colores. De esta forma, se da por finalizada la jugada.

Con este propósito hay que establecer la propiedad `Botón.Habilitado` con el valor `falso` (disponible en *Bloques > Integrados > Lógica*). Nuevamente, es conveniente definir un procedimiento que se encargue específicamente de esta tarea.



Procedimiento `deshabilitarBotonesDeColores`

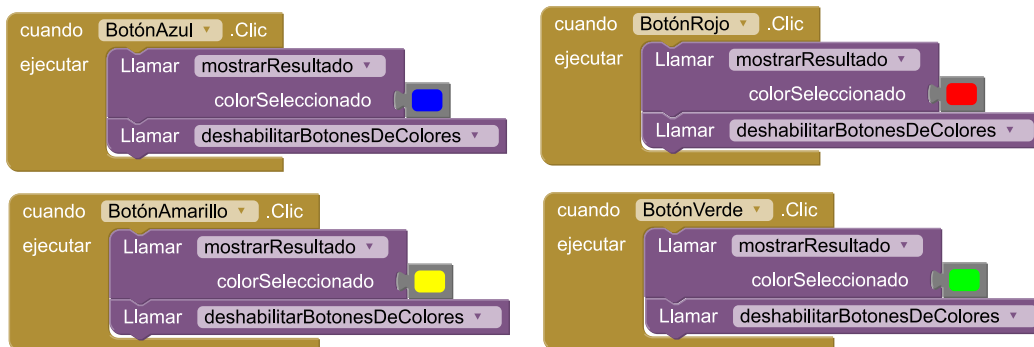
Una pregunta válida es: ¿desde dónde hay que invocar al procedimiento `deshabilitarBotonesDeColores`? Existen distintas alternativas que alcanzan el objetivo propuesto. A continuación se muestran tres, ordenadas por calidad, de menor a mayor.

La primera es hacerlo luego de mostrar el resultado, como última instrucción del procedimiento `mostrarResultado (colorSeleccionado)`.



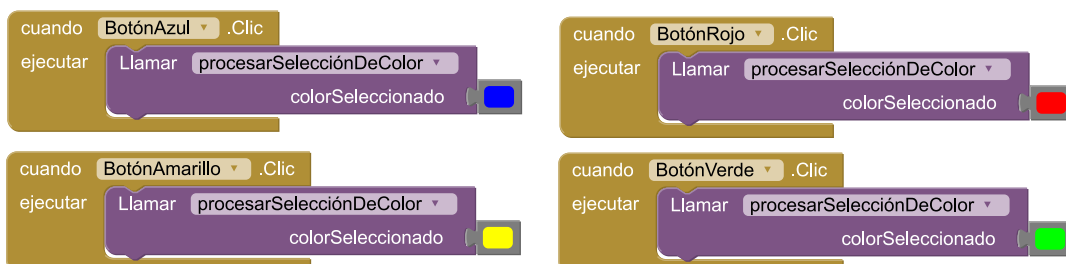
Invocación a `deshabilitarBotones` dentro de `mostrarResultado`

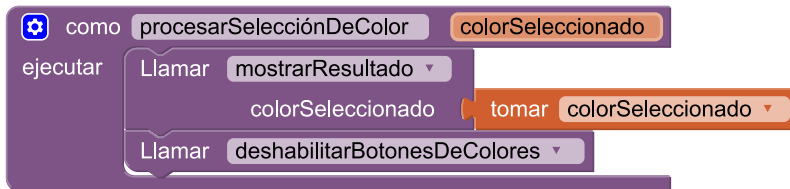
El punto débil de esta propuesta es que la deshabilitación de los botones, como unidad de sentido, comprende acciones que no forman parte de mostrar el resultado. Otra opción, más adecuada, es invocar a `deshabilitarBotones` desde los bloques de manejo de los eventos que producen los botones.



Invocación a `deshabilitarBotones` dentro de `mostrarResultado`

Esta propuesta maneja los eventos de los cuatro botones de eventos de forma muy similar. Aunque en este caso la similitud es sutil –cambia el color con el que se invoca `mostrarResultado` (`colorSeleccionado`)–, una alternativa superior consiste en crear un nuevo procedimiento que tenga un parámetro que represente el color y se encargue tanto de mostrar el resultado como de deshabilitar los botones. Podría llamarse, por ejemplo, `procesarSelecciónDeColor` (`colorSeleccionado`).





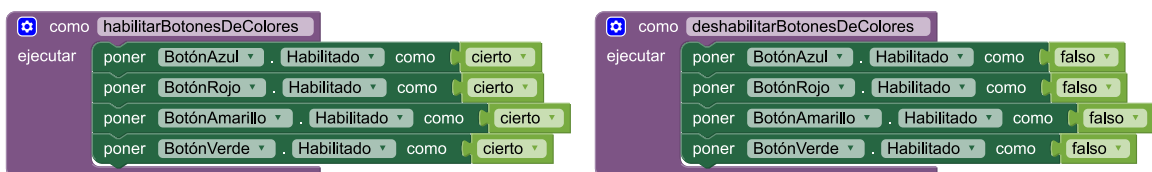
Definición de un nuevo procedimiento

Consigna 6: volver a empezar

La sexta y última consigna solicita que, cuando se presione el botón *Nuevo partido*, se pueda volver a jugar una mano; es decir, que todo quede como al comienzo. Por lo tanto hay que (i) generar un nuevo código secreto, (ii) habilitar los botones de colores, (iii) agrisar las etiquetas que representan ambos códigos –la que simboliza el color secreto y la que muestra la elección del jugador–, y (iv) borrar el texto que muestra la cantidad de coincidencias.

Dado que ya se cuenta con el procedimiento `generarCódigoSecreto`, el desafío puede completarse generando otros dos: uno para habilitar los botones de colores –que podría llamarse `habilitarBotonesDeColores`– y otro para ocultar los colores de las etiquetas que representan los códigos –que podría llamarse `agrisarEtiquetasDeLosCódigos`–.

Es probable que algunos estudiantes propongan habilitar los botones de colores de forma similar a como los deshabilitaron en la consigna anterior.



Dos procedimientos muy similares

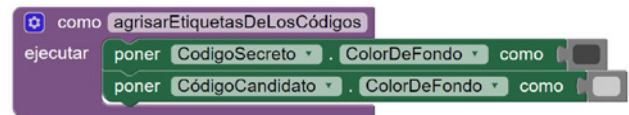
Nuevamente, por ser ambos procedimientos muy similares, lo conveniente es crear un nuevo procedimiento en el que el estado de habilitación de los botones sea un parámetro. Podría llamarse, por ejemplo, `establecerHabilitaciónDeBotonesDeColores (estadoDeHabilitación)`.





Un procedimiento para habilitar y deshabilitar botones

Para agrisar las etiquetas que representan los códigos basta con establecer la propiedad `Etiqueta.colorDeFondo` de `CódigoSecreto` y `CódigoCandidato` con gris oscuro y gris claro respectivamente.



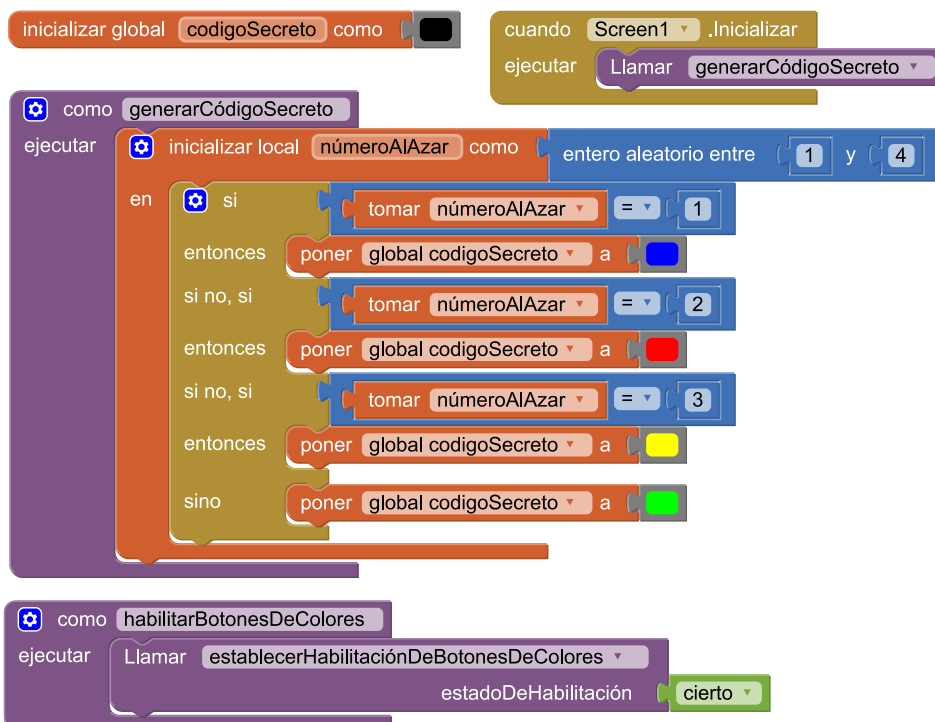
Procedimiento `agrisarEtiquetasDeLosCódigos`

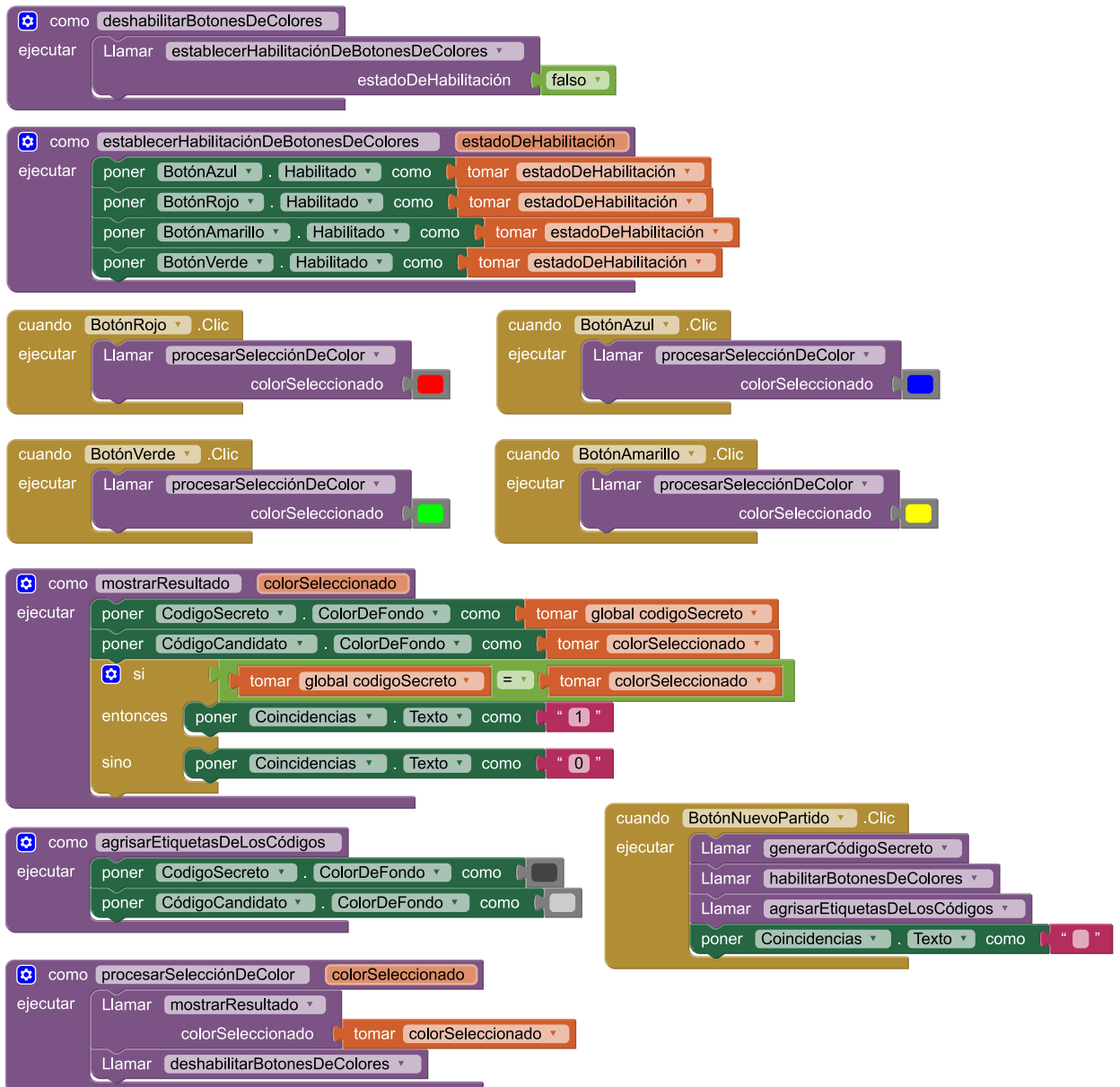
Finalmente, al hacer clic sobre `BotónNuevoPartido`, los procedimientos son invocados y se borra el texto que muestra la cantidad de aciertos.



Manejo de evento de `BotónNuevoPartido`

A continuación se muestra una solución completa de la actividad.





Solución completa de la actividad

Cierre

Repasamos con los estudiantes las posibilidades que dan las variables. Reiteramos que una variable nos sirve para recordar un valor y que, una vez definida, puede leerse y modificarse más adelante. Además, hacemos hincapié en la diferencia entre las globales y las locales: las globales pueden referenciarse en cualquier lugar del programa, mientras que las locales existen en espacios reducidos. Insistimos, por último, en que hay que ser muy cuidadosos al usar variables globales, pues fácilmente podemos perder noción de su evolución a medida que un programa se ejecuta.

NOMBRE Y APELLIDO:

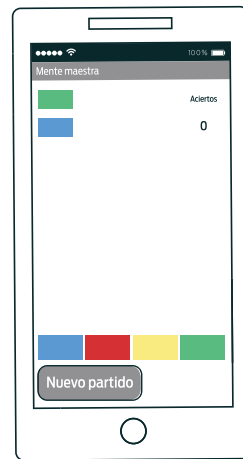
CURSO:

FECHA:

MENTE MAESTRA, PARTE I

¡Primera versión del juego! Acá, el código secreto es solo un color. Además, hay un solo intento. Para ganar, ¡hay que pegarle de una!

1. Comenzá armando la interfaz gráfica de esta primera aproximación al Mente maestra. Acá abajo podés ver los componentes y sus dimensiones. En la tabla, las propiedades que tenés que cambiar para que se vea igual a la que te mostramos. Seguí las indicaciones para componerla.



Código Secreto
(Etiqueta en posición (1,1))
19% x 7%

Panel Código Secreto
(Disposición tabular de 1x5)

Código Candidato
(Etiqueta en posición (1,1))
19% x 7%

Panel Código Candidato
(Disposición tabular de 8x5)

Botón Nuevo Partido
(Botón)
8% x Automático

Panel Botonera
(Disposición Horizontal)
10% x Ajustar al contenedor

7% x 25%
Botón Azul (Botón) | **Botón Rojo** | **Botón Amarillo** | **Botón Verde**

Etiqueta Aciertos
(Etiqueta en posición (1,5))
19% x 7%

Etiqueta Coincidencias
(Etiqueta en posición (1,5))
19% x 7%

NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto	Etiqueta	Color de fondo	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato	Etiqueta	Color de fondo	Ninguno	Gris Claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

NOMBRE Y APELLIDO:

CURSO:

FECHA:

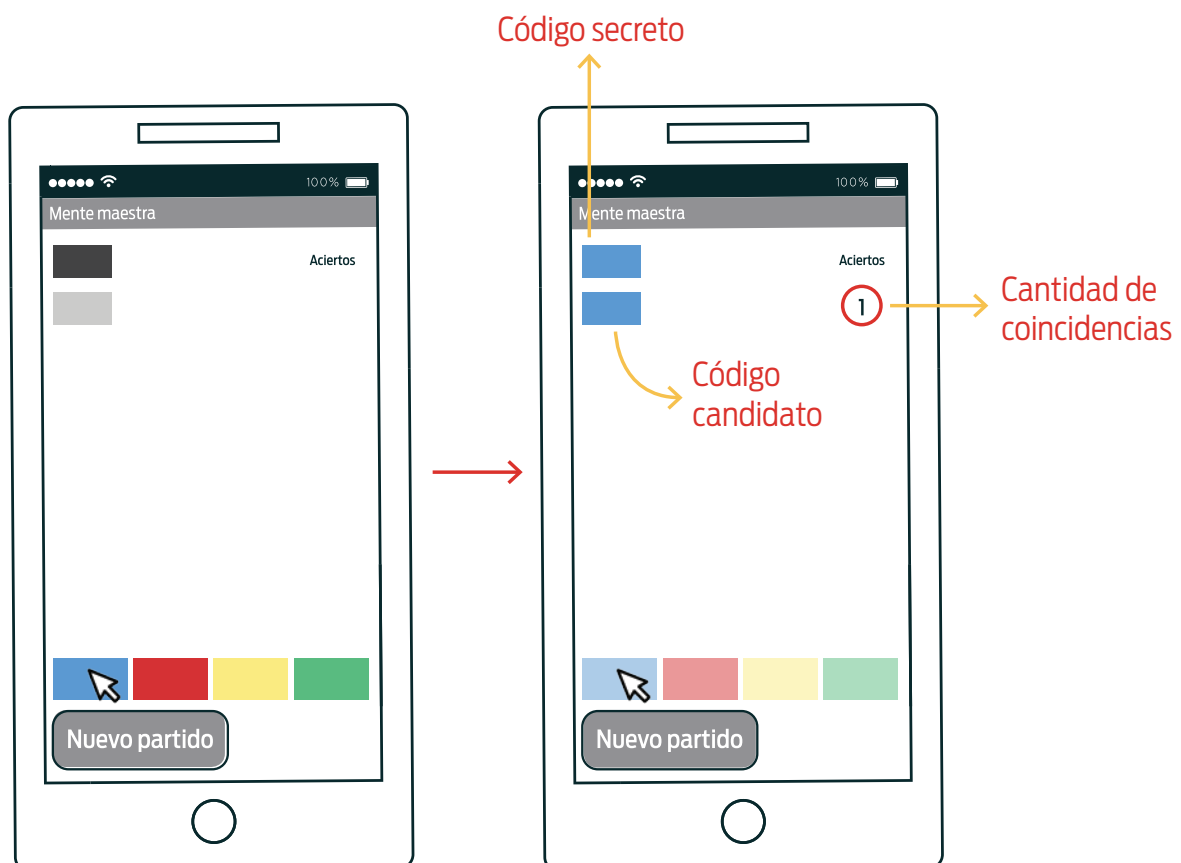
2. Ni bien comienza su ejecución, el juego tiene que generar al azar un código secreto de longitud 1. Se trata, en definitiva, de librar a la suerte la elección de un color. Tené en cuenta que, más tarde, quien juegue buscará adivinarlo. ¿Qué herramienta usaste para resolver el desafío? ¿Para qué?

PARA NO OLVIDAR...

Una variable es un nombre –que elegimos nosotros al programar– que denota un espacio de la memoria de la computadora donde se puede guardar un valor y, luego, recuperarlo o modificarlo. Las variables permiten, por ejemplo, que los juegos registren la cantidad de puntos que alcanza un jugador a medida que el juego avanza.



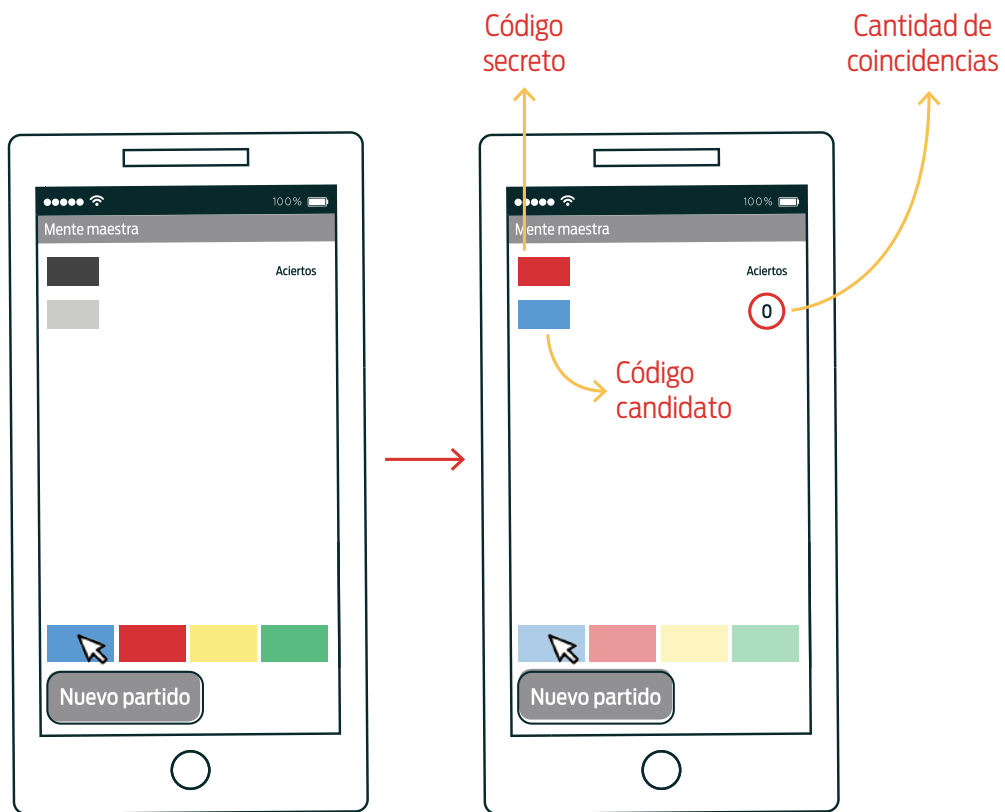
3. Ahora tenés que encargarte de que, una vez que el usuario presione el botón de azul, se muestre si hay o no una coincidencia entre la elección del jugador y el color secreto. Acá podés ver lo que tiene que pasar tanto si hay coincidencia como si no (en la página siguiente).



NOMBRE Y APELLIDO:

CURSO:

FECHA:



4. No solo de apretar el botón azul se trata este juego. También hay que resolver lo que sucede cuando se haga clic sobre el rojo, el amarillo y el verde. Resuelvelo evitando que en tu programa haya redundancia. ¿Qué hay que hacer, en general, si cuando programamos observamos que hay distintas partes del programa que son iguales o muy parecidas?

PROCEDIMIENTOS PARA SIMPLIFICAR LO COMPLEJO

Tené siempre presente que, al resolver un problema, es conveniente comenzar pensando las partes que componen una solución. En tu programa, cada una de estas partes las podés resolver en un procedimiento separado.

5. A los jugadores vamos a darles una sola oportunidad para que adivinen el color secreto. Una vez hecha la elección, los botones tienen que quedar deshabilitados. ¿Cómo lo conseguiste?
-
6. Y ahora, todo otra vez. Con un clic sobre el botón *Nuevo partido* hay que dejar todo como al principio, listo para que el jugador pueda intentar otra vez descubrir un nuevo color secreto.

Actividad 3






Mente maestra, parte II

DE A DOS

OBJETIVOS

- Construir un programa con repeticiones.
- Introducir el uso de listas.

MATERIALES

-  Computadora
-  Internet
-  MIT App Inventor 2
-  Ficha para estudiantes
-  Teléfono con Android (opcional)

DESARROLLO

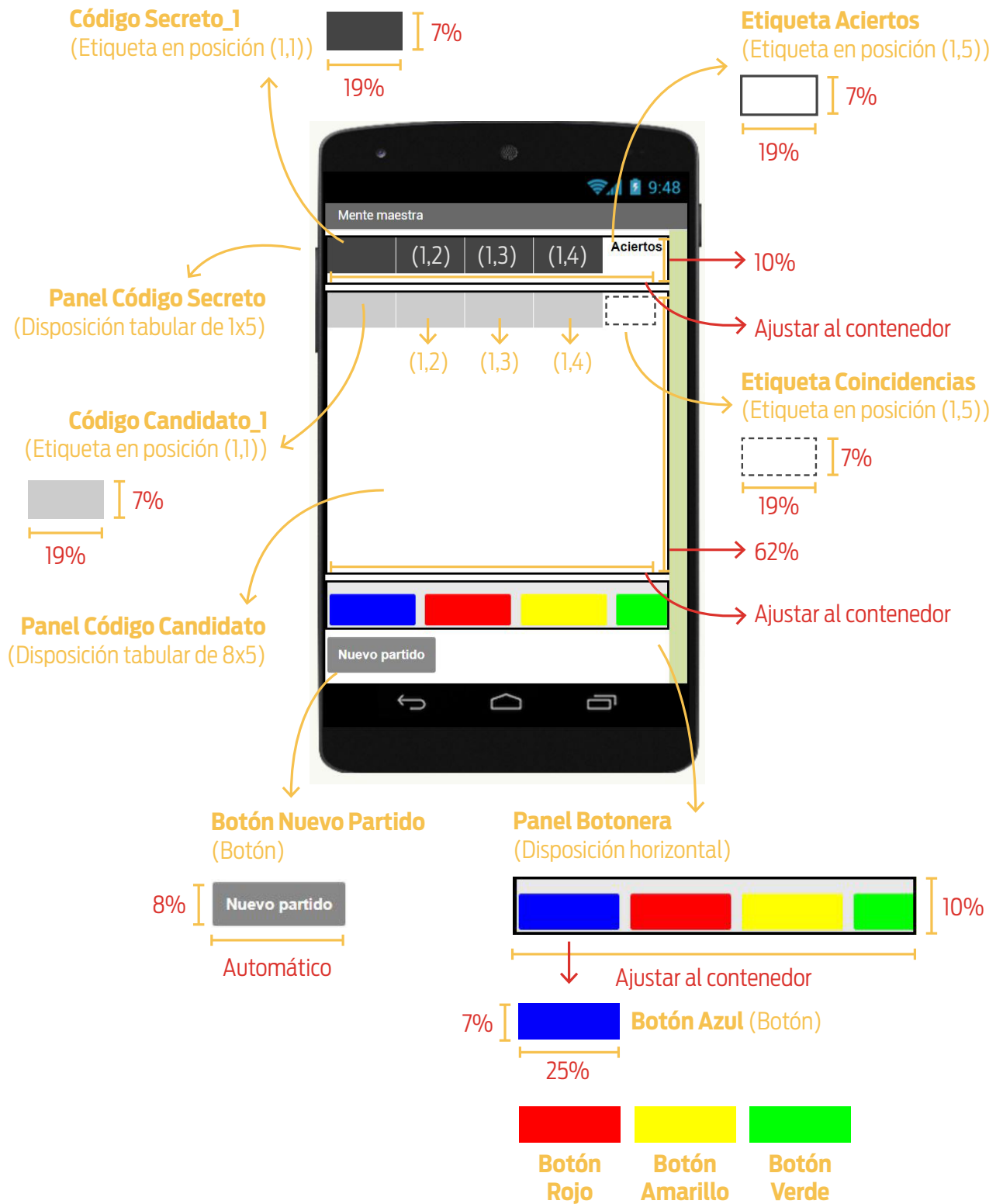
Los objetivos de esta segunda actividad de la serie son (i) presentar las listas como estructuras útiles en programación y (ii) construir una aplicación que use repeticiones. Para ello, los estudiantes programarán una nueva versión del Mente maestra en la que los códigos son de cuatro posiciones y los jugadores cuentan con un único intento para descubrirlos.

Nuevamente, puede ocurrir que los estudiantes propongan distintas soluciones para completar la consigna. A continuación, desarrollamos y analizamos una de ellas. Sin embargo, cualesquiera sean los programas que elaboren, es importante (i) que usen procedimientos para descomponer problemas en subproblemas más simples y para evitar las redundancias, introduciendo parámetros en los casos que sean necesarios; (ii) que incorporen el uso de listas; y (iii) que utilicen bloques de repetición para evitar copiar, una vez a continuación de otra, una misma secuencia de instrucciones.¹

Consigna 1: interfaz gráfica

Les repartimos la ficha a los estudiantes y los invitamos a que resuelvan la primera consigna. Allí se dan instrucciones para armar la interfaz gráfica de la aplicación. La tabla a continuación muestra los valores de las propiedades que hay que cambiar (en relación con los que App Inventor asigna por defecto) para que la aplicación se vea tal como en la siguiente imagen.

¹ Una versión completa de la aplicación se encuentra disponible en la galería de proyectos de App Inventor del usuario “programar2020”.



Diseño de la interfaz

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto_1 ¹	Etiqueta	Color de fondo	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato_1 ²	Etiqueta	Color de fondo	Ninguno	Gris claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

¹Las restantes etiquetas del código secreto requieren los mismos cambios.

²Las restantes etiquetas del código candidato requieren los mismos cambios.

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

Consigna 2: generar el código secreto

Una vez que todos hayan compuesto la interfaz gráfica, les comentamos: “Como hemos visto, un código secreto de una posición se puede representar con una variable global. En este caso, el código tiene cuatro posiciones. ¿Cómo harían para representarlo?”. Es esperable que algún estudiante conteste que se pueden usar cuatro variables globales, una para cada posición. “Sí, esa es una forma de resolverlo, pero tratemos de pensar una forma más general. Por ejemplo, ¿les parecería una buena solución usar mil variables globales si el código tuviese mil posiciones? ¡Ni siquiera podríamos verlas en la pantalla de la computadora!”. Guiamos el intercambio para concluir que, independientemente de la cantidad de posiciones del código secreto, siempre podríamos representarlo como *una única lista* de colores.

Continuamos: “Si fuese un código de cuatro posiciones, la lista tendría cuatro colores; si fuese de cinco, cinco colores; y si fuese de mil, entonces tendría mil colores, ¿qué les parece?”. Escuchamos con atención sus observaciones y les comentamos: “A pesar de que el código tiene varias posiciones, el desafío es seguir representándolo con una única variable global”. Los invitamos, entonces, a que exploren el entorno en busca de herramientas que les permitan resolver el desafío, que consiste en generar el código secreto al comenzar la ejecución de la aplicación, y conservarlo en una única variable global.

A continuación, un apartado sobre las listas y, más adelante, la continuación del desarrollo de la actividad.

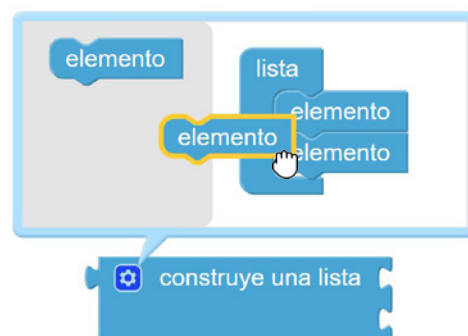
Sobre las listas

En computación, una **lista** es un tipo de datos que representa una secuencia de elementos ordenados. Se puede pensar, por ejemplo, en listas de números, listas de colores o listas de palabras. Cada elemento, además, puede aparecer más de una vez en una lista.



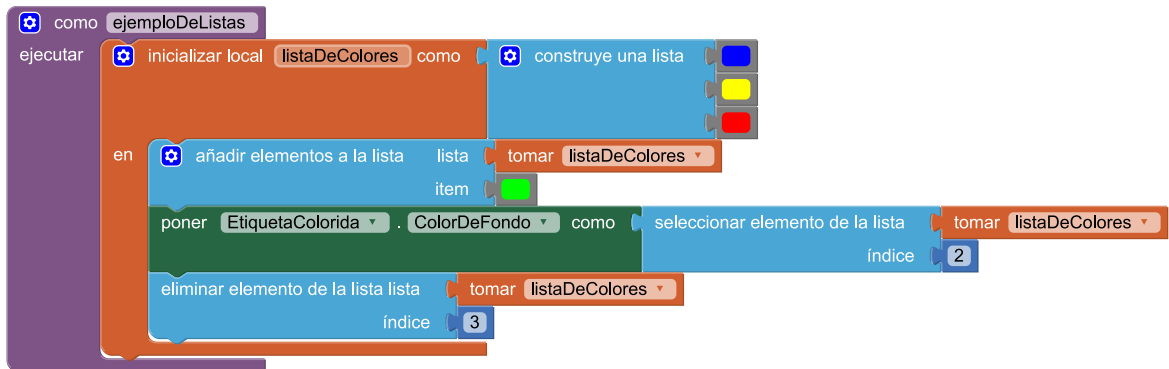
Listas de números, colores y palabras

Casi todos los lenguajes de programación modernos permiten trabajar con listas y tienen instrucciones para crearlas, agregarles y borrarles elementos, inspeccionar el valor de una cierta posición, etc. En App Inventor, una lista puede crearse a partir de una serie de elementos usando el bloque `construye una lista [] []` disponible en *Bloques > Integrados > Listas*. La cantidad de valores que se incluyen en la lista al crearla de este modo se puede modificar accediendo a las opciones que se despliegan al hacer clic sobre la tuerca que se encuentra en la esquina superior izquierda del bloque.



Bloque para construir una lista a partir de una serie de elementos

En la imagen se encuentra un ejemplo que ilustra el uso de algunas operaciones que se pueden realizar con listas.



Ejemplo de uso de listas

En primer lugar se crea una variable `listaDeColores` cuyo valor inicial es una lista en la que en la primera posición está el color azul, en la segunda el color amarillo y en la tercera el color rojo. Luego, se utiliza el bloque `añadir elementos a la lista (lista) [] (ítem) []` para agregar al final de `listaDeColores` el color verde (esta instrucción agrega un elemento al final de la lista). Aquí, el parámetro `lista` representa la lista en la que se agrega un nuevo elemento e `ítem` el elemento que se agrega. A continuación, se usa `seleccionar elemento de la lista [] (índice) []` para establecer que el color de fondo de la etiqueta `EtiquetaColorida` sea el que se encuentra en la segunda posición de `listaDeColores`; es decir, de color amarillo. Finalmente, con el bloque `eliminar elemento de la lista (lista) [] (índice) []` se remueve el elemento que está en la tercera posición; en este caso, el color rojo. En la tabla a continuación se observa la evolución de la lista `listaDeColores` a medida que avanza la ejecución del procedimiento `ejemploDeListas`.

INSTRUCCIÓN	LISTA REPRESENTADA

Pone amarillo el fondo de una etiqueta sin modificar la lista

INSTRUCCIÓN	LISTA REPRESENTADA

Evolución de `listaDeColores` a medida que se ejecuta `ejemploDeListas`

En App Inventor –al igual que en todos los lenguajes que admiten el uso de listas– es posible trabajar con **listas vacías**, es decir, con listas que no contienen ningún elemento. Estas son útiles en distintos contextos. A modo de ejemplo, puede considerarse una aplicación de *delivery* de empanadas. Las empanadas que forman parte de un pedido pueden representarse como una lista de los gustos que un cliente seleccione. Al comenzar un pedido, aún no se ha agregado ninguna empanada; por lo tanto, la lista se encuentra vacía. Luego, a medida que se agreguen empanadas, sus gustos se irán agregando a la lista. Una vez completado, el pedido pasa a la cocina para comenzar la preparación de la comida.

inicializar global `empanadas` como → Se crea una lista vacía

cuando `AgregarUnaDeCarne` .Clic
 ejecutar

cuando `AgregarUnaDeJamónYQueso` .Clic
 ejecutar

cuando `AgregarUnaDeHumita` .Clic
 ejecutar

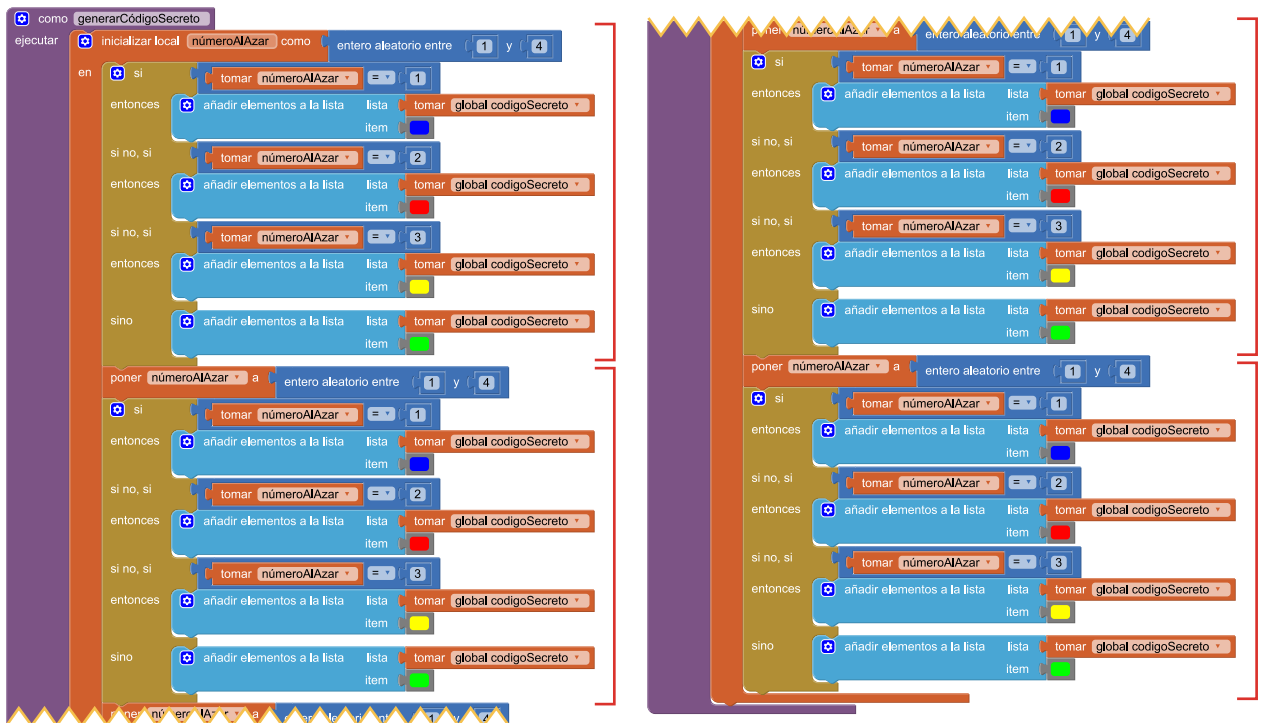
cuando `FinalizarEIPedido` .Clic
 ejecutar

Uso de una lista vacía

Consigna 2. Generar el código secreto (continuación)

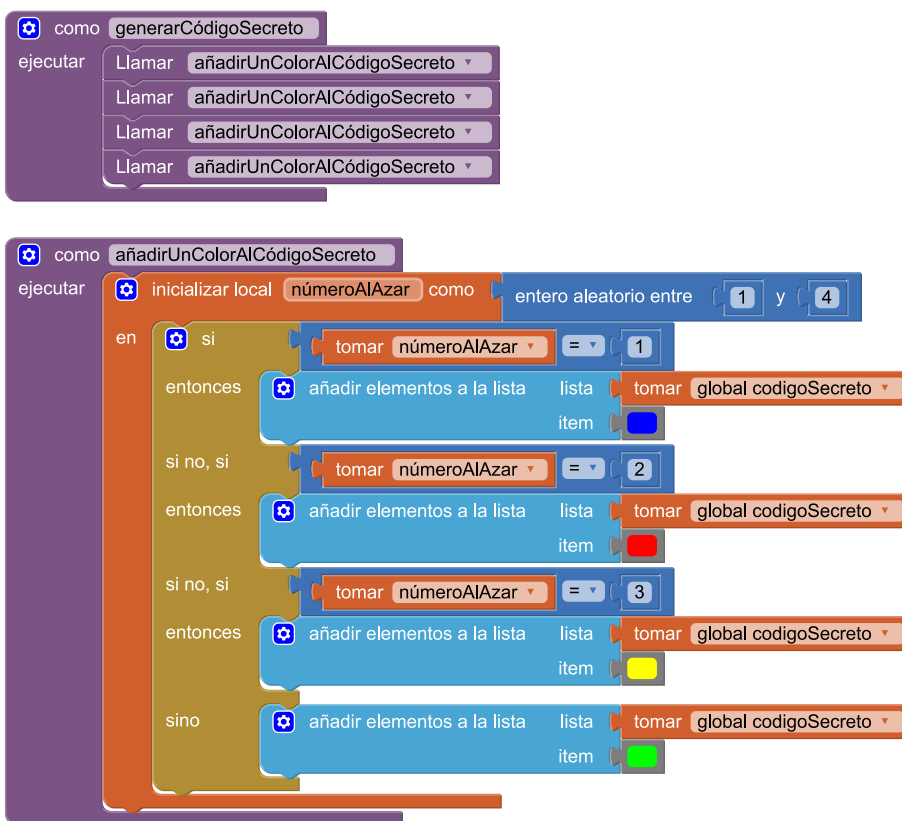
Como en distintas partes del programa hay que acceder a la variable en la que se almacenará el código secreto (por ejemplo, cuando la aplicación comienza su ejecución y se genera el código y, luego, cuando el jugador completa su código candidato y se cuentan las coincidencias), hay que usar una variable global.

Una solución que podría presentarse consiste en inicializar una variable `codigoSecreto` como una lista vacía y, manejando el evento que se produce cuando se carga la aplicación, invocar a un procedimiento –que podría llamarse `generarCodigoSecreto`– que genere cuatro números al azar, uno a continuación del otro, para definir los colores del código secreto.



Se realiza cuatro veces lo mismo

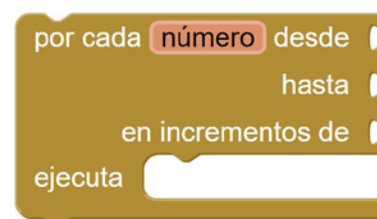
A los estudiantes que acerquen esta propuesta les hacemos notar que en `generarCódigoSecreto` están haciendo cuatro veces lo mismo y les sugerimos que piensen alguna solución alternativa. Una variación que mejora cualitativamente el programa consiste en encapsular el añadido de un color al código secreto en un procedimiento –que podría llamarse, por ejemplo, `añadirUnColorAlCódigoSecreto`– que sea invocado cuatro veces.



Se encapsula en un procedimiento la generación de un color del código

Frente a esta propuesta comentamos: “Está muy bien encapsular en un procedimiento la generación de un color del código pero, ¿no se sigue repitiendo cuatro veces lo mismo en `generarCódigoSecreto`? ¿Qué sucedería si el código secreto fuese de mil posiciones?”

Como en casi cualquier lenguaje de programación, en App Inventor hay herramientas que permiten repetir instrucciones. En particular, el bloque `por cada (número) desde [] hasta [] en incrementos de [] / ejecuta { }` sirve para expresar en forma explícita la cantidad de veces que se tienen que ejecutar las instrucciones que se encuentren dentro de `{ }`.



Bloque para expresar repeticiones

El modo más habitual de usar el bloque es completando `desde` con un 1, `hasta` con la cantidad de veces que queremos que se repita la ejecución de las instrucciones contenidas en `{ }` y en `incremento de` con un 1. En la figura se muestra una forma de completar `generarCódigoSecreto` para resolver el desafío.¹

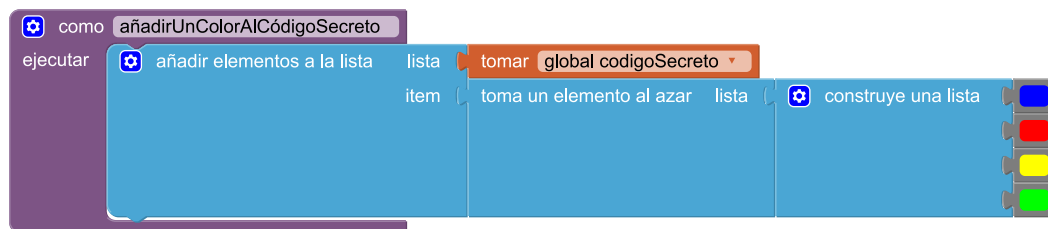


Procedimiento que usa repeticiones

Una forma alternativa (y elegante) de elegir un color al azar para el código secreto sin necesidad de generar un número aleatorio puede realizarse usando el bloque `toma un elemento al azar (lista) []`, disponible en *Bloques > Integrado > Listas*. De este modo, se puede agregar a la lista `códigoSecreto` el resultado de seleccionar un elemento al azar de otra lista (creada *ad hoc*) que contenga los cuatro colores –azul, rojo, amarillo y verde–.



Bloque para obtener un elemento al azar de una lista



Forma alternativa de `añadirUnColorAlCódigoSecreto`

Consigna 3: manejar eventos de los botones de colores

La tercera consigna pide manejar los eventos que se producen cuando se presionan los botones de colores. Hacerlo involucra (i) actualizar en la pantalla el código candidato agregándole el color elegido por el jugador, y (ii) chequear si al agregar el nuevo color se completa el código candidato, caso en el cual hay que develar el código secreto, mostrar la cantidad de coincidencias y deshabilitar los botones de colores.

En primer lugar, los estudiantes tienen que reconocer que, al igual que con el código secreto, para el código candidato también es necesario usar una variable global. Esta, que podría llamarse `códigoCandidato`, se inicializa como una lista vacía al comenzar la ejecución de la aplicación y se irá actualizando a medida que se elijan los colores.

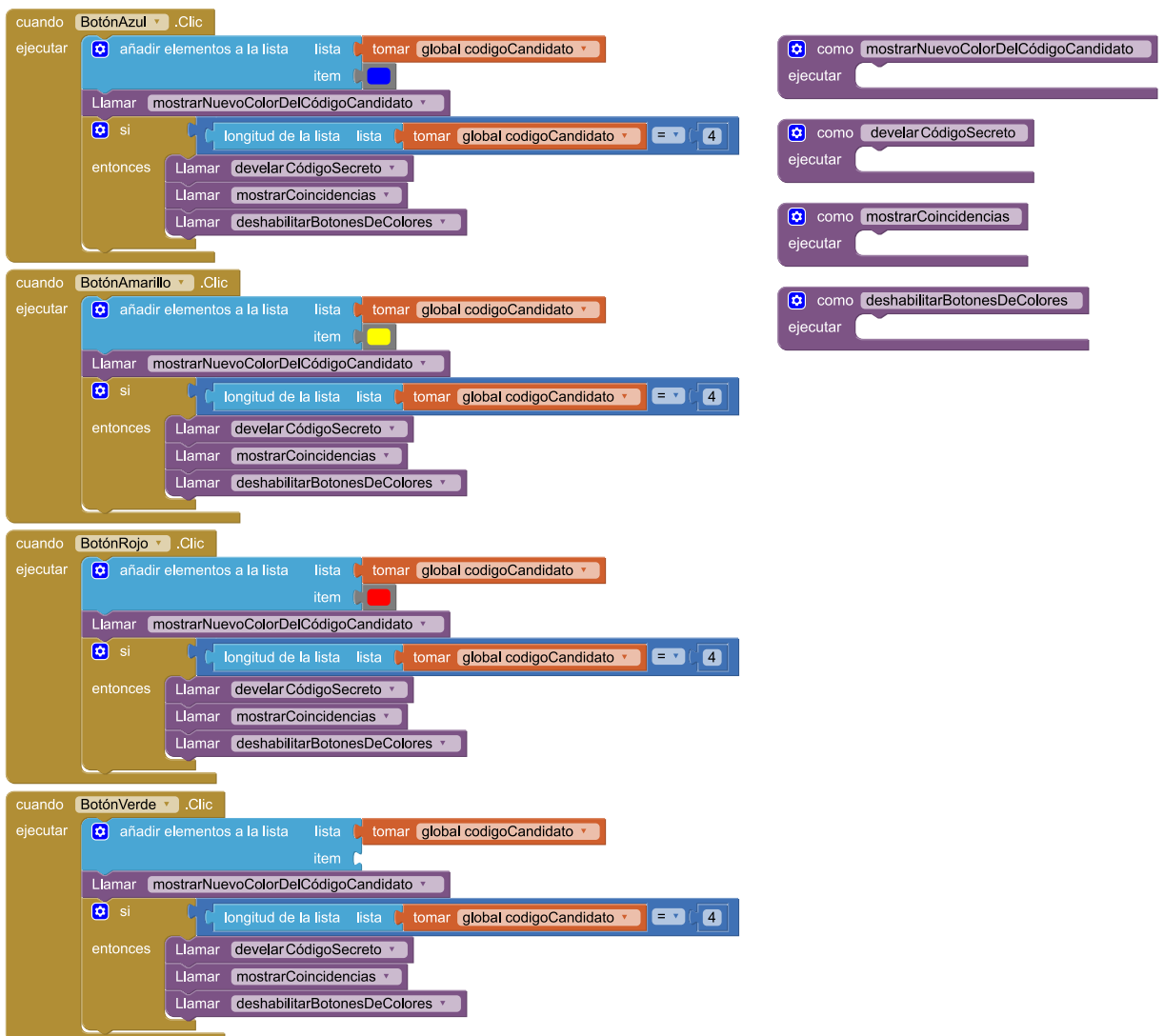
¹ Hay otras alternativas para describir un número n de repeticiones, aunque son más rebuscadas y menos recomendables. Por ejemplo, en lugar de completar (*desde, hasta, en incrementos de*) con (1,4,1), se podría usar (7,10,1) o (1,7,2), entre otros, y generar el mismo efecto: 4 repeticiones.

inicializar global `codigoSecreto` como  crear una lista vacía

Variable que guarda el código candidato

Para saber si se ha completado el código candidato, hay que chequear si la longitud de `codigoCandidato` es 4 (esto solo sucede cuando ya se le agregaron cuatro colores).

Además, para completar el manejo de los eventos que se generan al presionar los botones de colores, pueden crearse cuatro procedimientos –que luego nos encargaremos de resolver– donde cada uno encapsule una tarea específica. Podrían llamarse, por ejemplo, `mostrarNuevoColorDelCódigoCandidato`, `develarCódigoSecreto`, `mostrarCoincidencias` y `deshabilitarBotonesDeColores`.



The image displays four Scratch event blocks for buttons: BotónAzul, BotónAmarillo, BotónRojo, and BotónVerde. Each block contains the following sequence of actions:

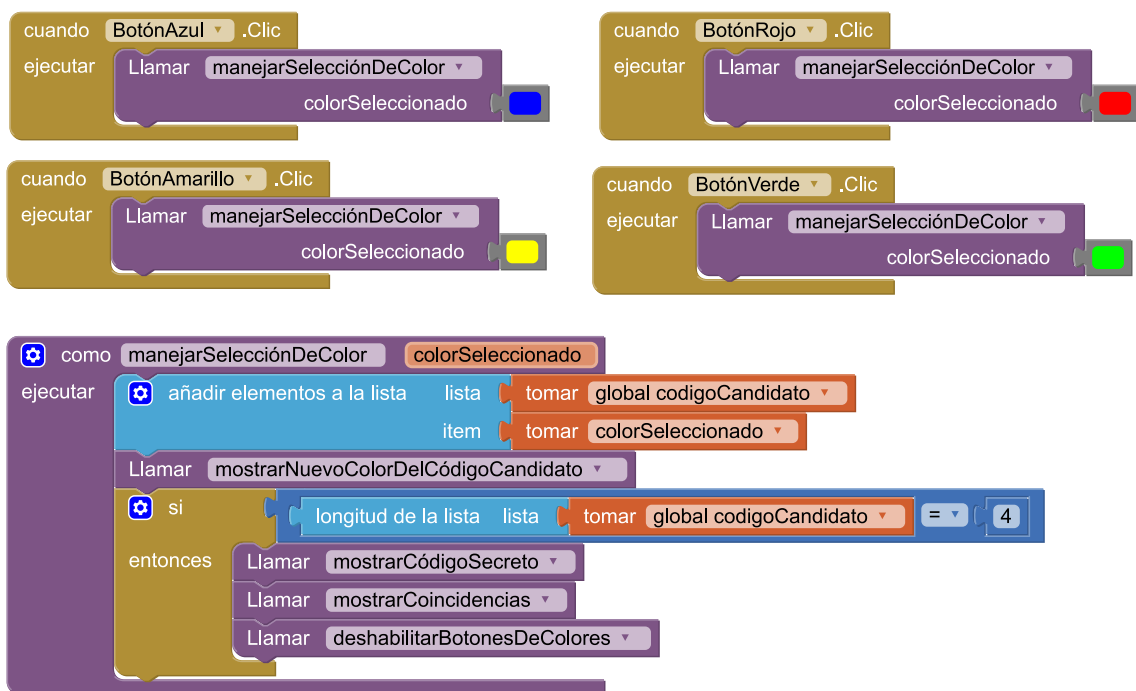
- cuando `BotónAzul` / `BotónAmarillo` / `BotónRojo` / `BotónVerde` .Clic
- ejecutar:
 - añadir elementos a la lista (lista: `global codigoCandidato`, item: `global codigoCandidato`)
 - Llamar `mostrarNuevoColorDelCódigoCandidato`
 - si `longitud de la lista` (lista: `global codigoCandidato`) `=` `4` entonces:
 - Llamar `develarCódigoSecreto`
 - Llamar `mostrarCoincidencias`
 - Llamar `deshabilitarBotonesDeColores`

To the right of the main code blocks, four procedure call blocks are shown, each with a 'como' field and an 'ejecutar' field:

- como `mostrarNuevoColorDelCódigoCandidato`
- como `develarCódigoSecreto`
- como `mostrarCoincidencias`
- como `deshabilitarBotonesDeColores`

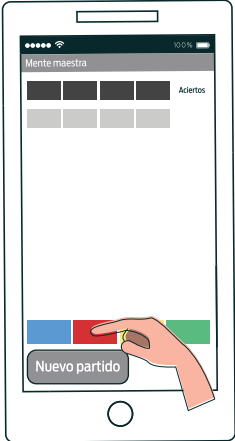

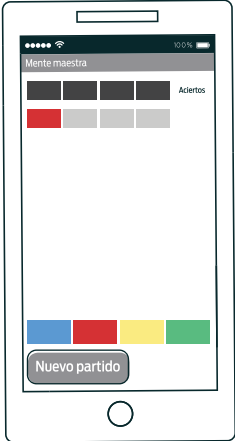
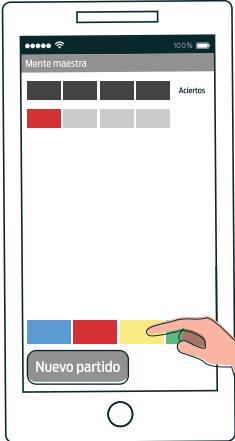

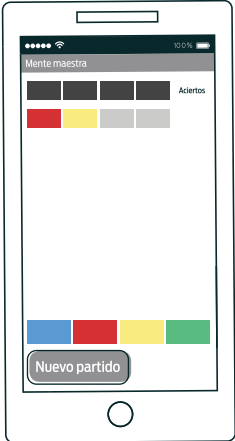
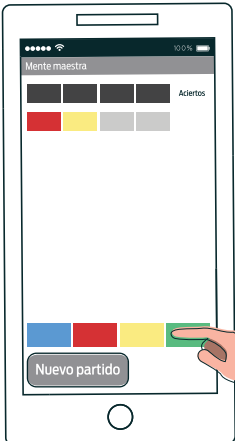

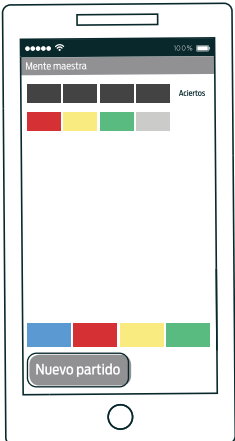
Programa incompleto y con redundancia

Como puede observarse, los manejos de los eventos de los cuatro botones son muy similares: únicamente difieren en la primera instrucción, en la que varía el color que se agrega a la lista `códigoCandidato`. Por lo tanto, se puede crear un único procedimiento que abstraiga el color en un parámetro y que sea invocado con el color adecuado en cada caso. Un nombre posible sería `manejarSelecciónDeColor (colorSeleccionado)`.



Refactorización del manejo de eventos

El objetivo de `mostrarNuevoColorDelCódigoCandidato` es reflejar visualmente la selección de un nuevo color por parte del usuario. En la pantalla, el código candidato está representado por cuatro etiquetas: `CódigoCandidato_1`, `CódigoCandidato_2`, `CódigoCandidato_3` y `CódigoCandidato_4`. ¿En cuál de ellas hay que mostrar el color elegido? Esta pregunta puede responderse a partir de la longitud de la lista `códigoCandidato`: si tiene un solo elemento, hasta el momento el usuario solo ha incorporado un color al código candidato, con lo que hay que mostrarlo en `CódigoCandidato_1`; siguiendo el mismo razonamiento, si tiene dos hay que mostrarlo en `CódigoCandidato_2`; si tiene 3, en `CódigoCandidato_3`; y si tiene 4 en `CódigoCandidato_4`. Además, el color que hay que mostrar es, siempre, el último agregado a `códigoCandidato`, pues allí se encuentra la última elección del usuario.

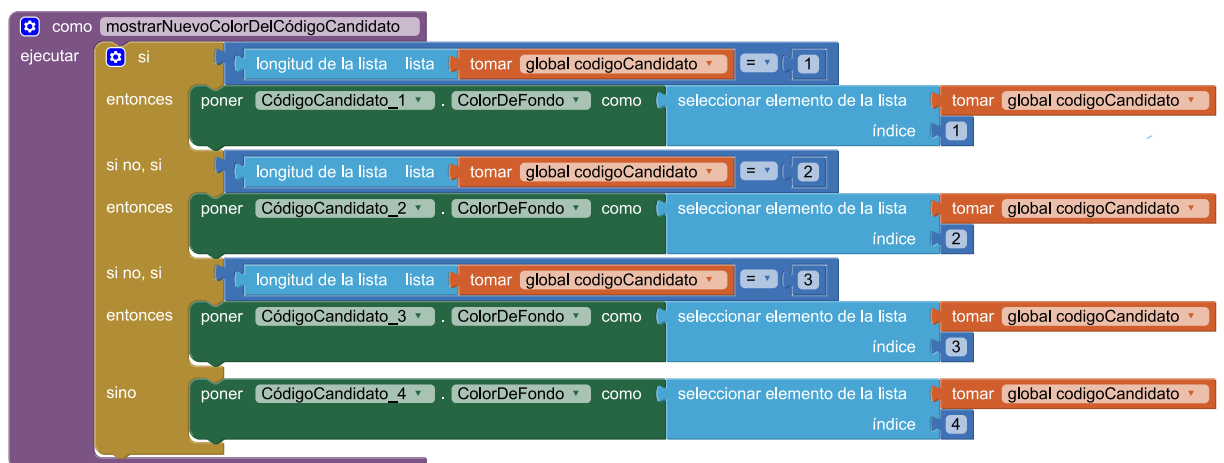
ACCIÓN DEL USUARIO	EFECTO EN LA LISTA CÓDIGOCANDIDATO	REFLEJO EN LA PANTALLA
		
		
		



Evolución de una corrida de la aplicación

El último elemento de una lista puede obtenerse a partir de su longitud. Por ejemplo, en una lista con dos elementos –es decir, de longitud 2–, el último elemento es el que se encuentra en la segunda posición; y en una con tres elementos –de longitud 3–, el que está en la tercera posición. En general, en una lista con n elementos –de longitud n –, el último se encuentra en la n ésima posición.¹

Se muestra a continuación el procedimiento `mostrarNuevoColorDelCódigoCandidato`.



Se muestra un nuevo color del código candidato

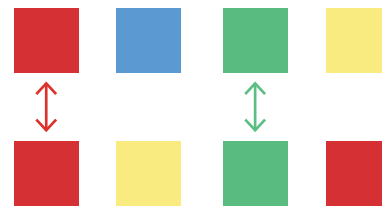
¹Esta afirmación no es válida en una lista vacía. En tal caso, no hay elemento alguno.

El propósito del procedimiento `develarCodigoSecreto` es mostrar en la pantalla el código secreto. Alcanza, en este caso, con establecer el color de fondo de las etiquetas `CódigoSecreto_1`, `CódigoSecreto_2`, `CódigoSecreto_3` y `CódigoSecreto_4`, con los colores de la primera, la segunda, la tercera y la cuarta posición (respectivamente) de la lista `códigoSecreto`.



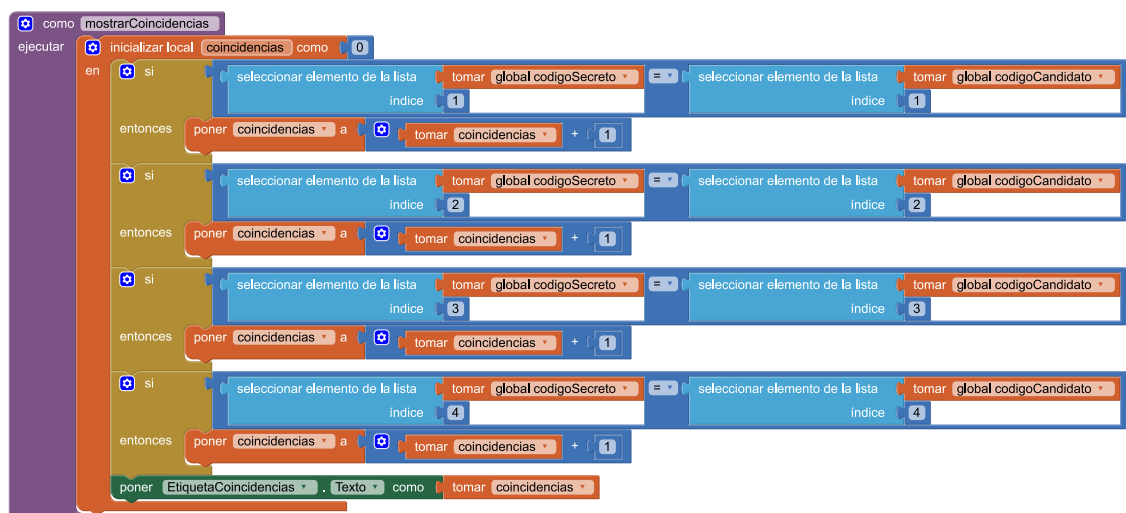
Se devela el código secreto

La cantidad de coincidencias entre el código secreto y el código candidato está dada por la cantidad de posiciones de las listas `códigoSecreto` y `códigoCandidato` que contienen el mismo color. Por ejemplo, si el código secreto es `[rojo, azul, verde, amarillo]` y el código candidato `[rojo, amarillo, verde, rojo]` hay dos coincidencias, que corresponden a las posiciones 1 y 3 de las listas.



Dos coincidencias

Para calcular las coincidencias se puede inicializar una variable en 0 y, por cada coincidencia, sumarle 1. Luego, para mostrar el resultado por pantalla, alcanza con actualizar el texto de la etiqueta `EtiquetaCoincidencias`. Una posible propuesta se muestra a continuación.



Propuesta endable para calcular coincidencias

Si bien es funcionalmente correcta, esta solución no es buena. Por ejemplo, si los códigos pasasen a ser muy largos, habría que incluir una gran cantidad de sentencias condicionales, una por cada posición del código. El resultado sería un programa muy largo, menos comprensible y más propenso a errores.

Una forma de saldar el problema descrito es usar repeticiones. De este modo, se puede analizar una posición diferente de las listas en cada ejecución del cuerpo de la repetición.

posición adquiere los valores 1, 2, 3 y 4 en la primera, la segunda, la tercera y la cuarta ejecución del cuerpo del ciclo respectivamente

Se compara el contenido de ambas listas en los distintos valores que adquiere posición

Se muestran las coincidencias usando repeticiones

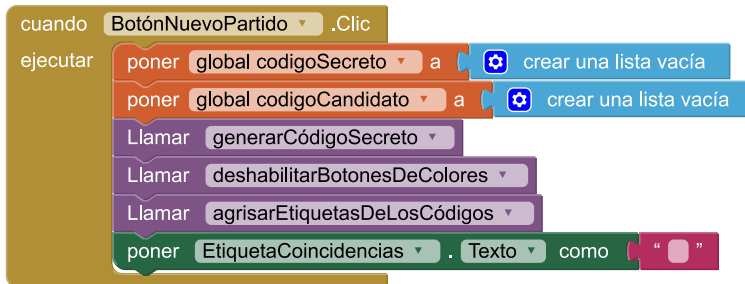
Es interesante notar que, en este caso, se ha usado el valor que adquiere `posición` en cada ejecución del cuerpo de la repetición para indicar qué posición de las listas hay que inspeccionar: la primera vez vale uno, y evalúa si son iguales el primer color de `códigoSecreto` y el primero de `códigoCandidato`; la segunda vale dos, y se compara el segundo de uno contra el segundo del otro, y así siguiendo. De algún modo, `posición` es una variable y puede referenciarse únicamente en los bloques que componen la repetición.

Para completar la consigna, solo resta completar el procedimiento `deshabilitarBotonesDeColores`. En este caso alcanza con establecer la propiedad `Botón.Habilitado` de los cuatro botones de colores como `falso`.

Se deshabilitan los botones de colores

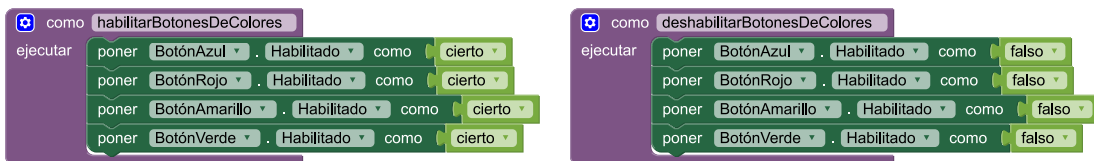
Consigna 4: todo como al principio

La cuarta y última consigna pide que al presionar el botón *Nuevo partido* se restablezca el estado inicial de la aplicación de forma tal que el usuario pueda intentar adivinar un nuevo código secreto. Esto requiere (i) vaciar las listas `códigoSecreto` y `códigoCandidato`, (ii) generar un nuevo código secreto, (iii) habilitar los botones de colores, (iv) agrisar las etiquetas que muestran los códigos en la pantalla, y (v) vaciar el texto de la etiqueta que muestra la cantidad de coincidencias.



Comenzar un nuevo partido

Al igual que cuando son inicializadas al cargar la aplicación, a las variables `codigoSecreto` y `codigoCandidato` se les puede asignar una lista vacía. Por otro lado, el procedimiento `generarCódigoSecreto` ya está programado. Para habilitar los botones de colores, se puede crear un procedimiento que establezca la propiedad `Botón.Habilitado` de los cuatro botones de colores como `cierto`.



Procedimientos similares para habilitar y deshabilitar botones

Debido a la similitud entre `habilitarBotonesDeColores` y `deshabilitarBotonesDeColores`, se puede crear un nuevo procedimiento que tenga un parámetro que indique si se quiere habilitar o deshabilitar a los botones y, en cada caso, invocarlo con el argumento que corresponde.



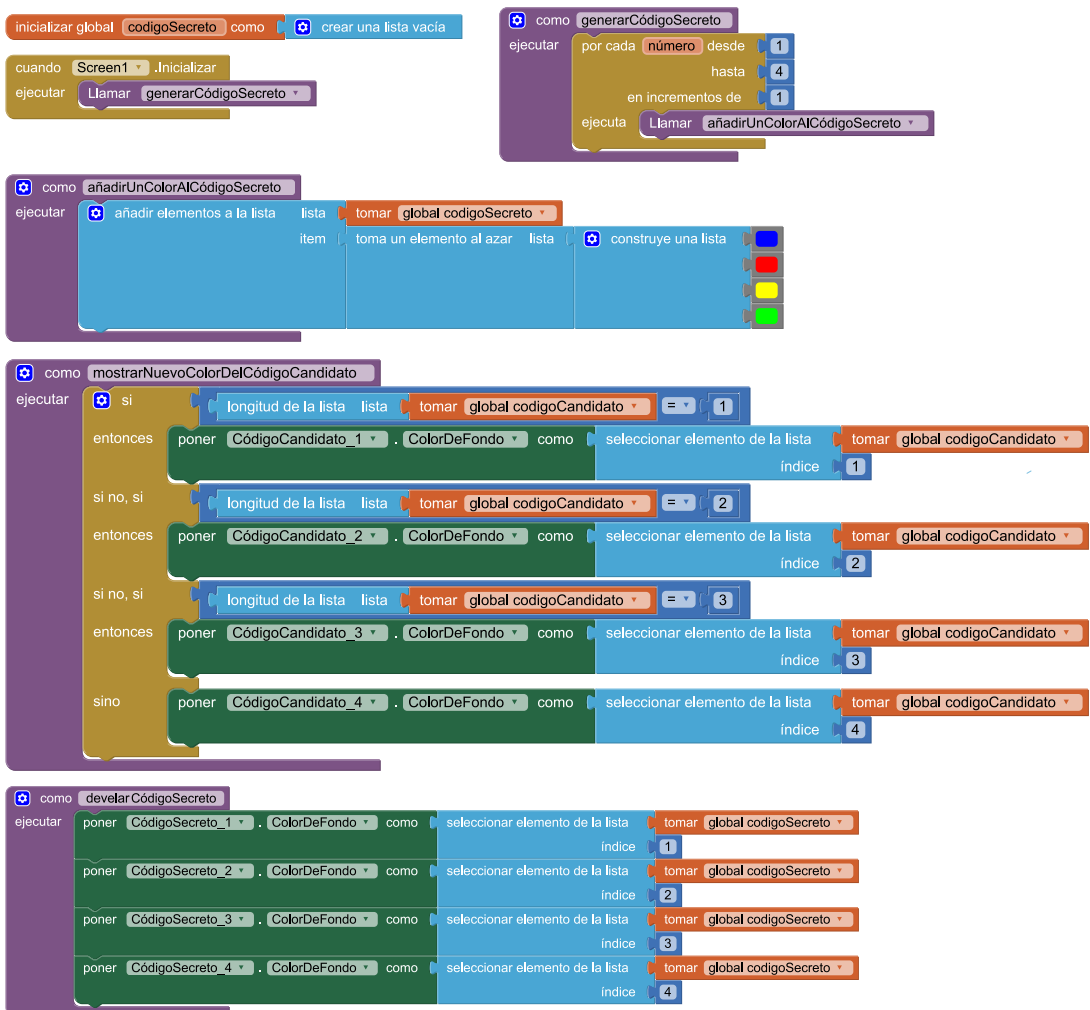
Refactorización para habilitar y deshabilitar botones

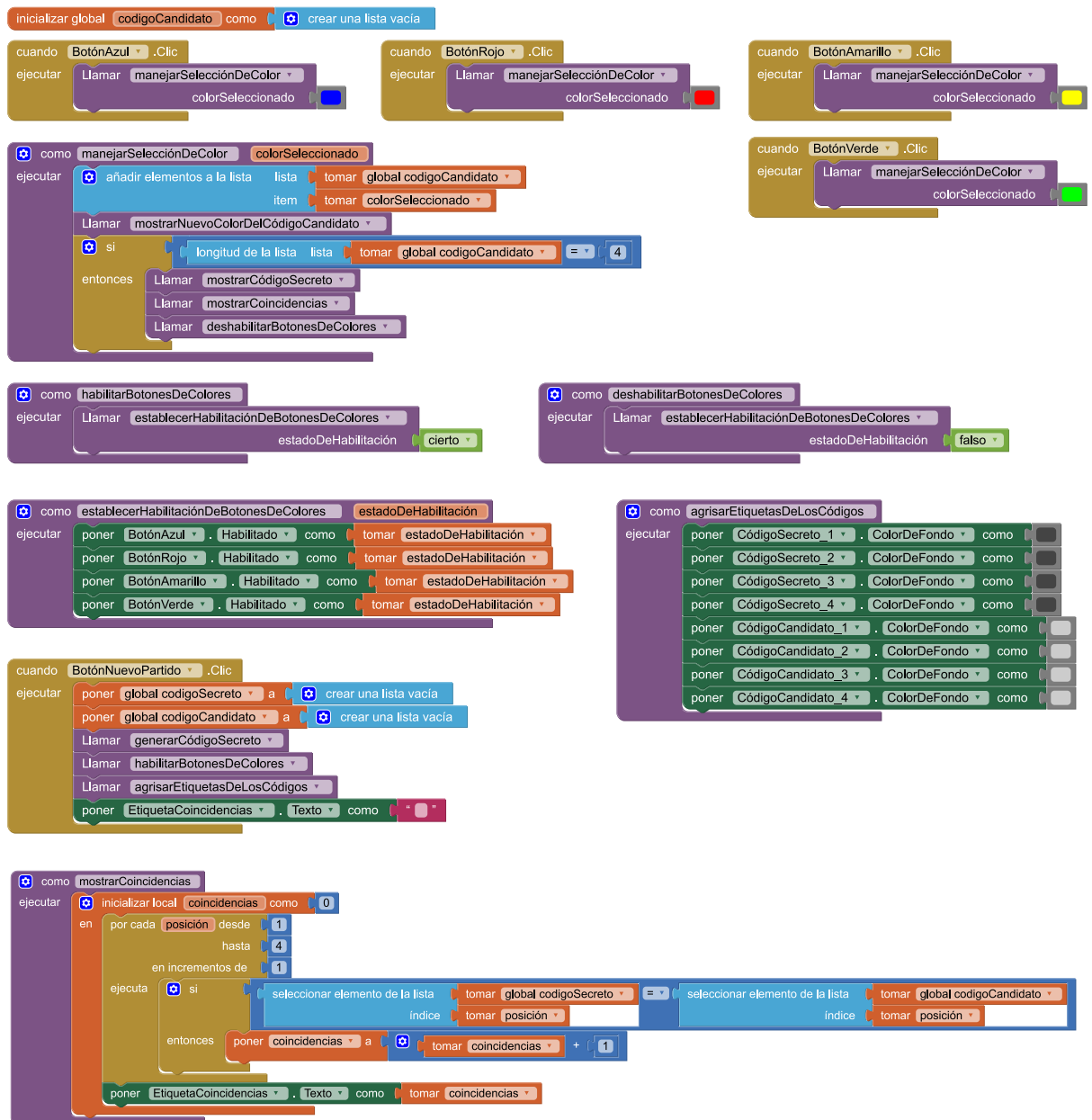
Por último, el procedimiento `agrisarEtiquetasDeLosCódigos` solo debe ocuparse de establecer el color de fondo de las etiquetas que muestran el código secreto y el código candidato como gris oscuro y gris claro respectivamente.



Se agrisan las etiquetas de los códigos

A continuación se exhibe una solución completa de la actividad.





Programa completo

CIERRE

Repasamos con los estudiantes que las listas dan la posibilidad de representar secuencias de valores ordenados sin necesidad de usar muchas variables. Además, hacemos hincapié en la utilidad de contar con bloques para hacer repeticiones, que permiten hacer muchas veces lo mismo sin tener que repetir instrucciones en forma explícita.

NOMBRE Y APELLIDO:

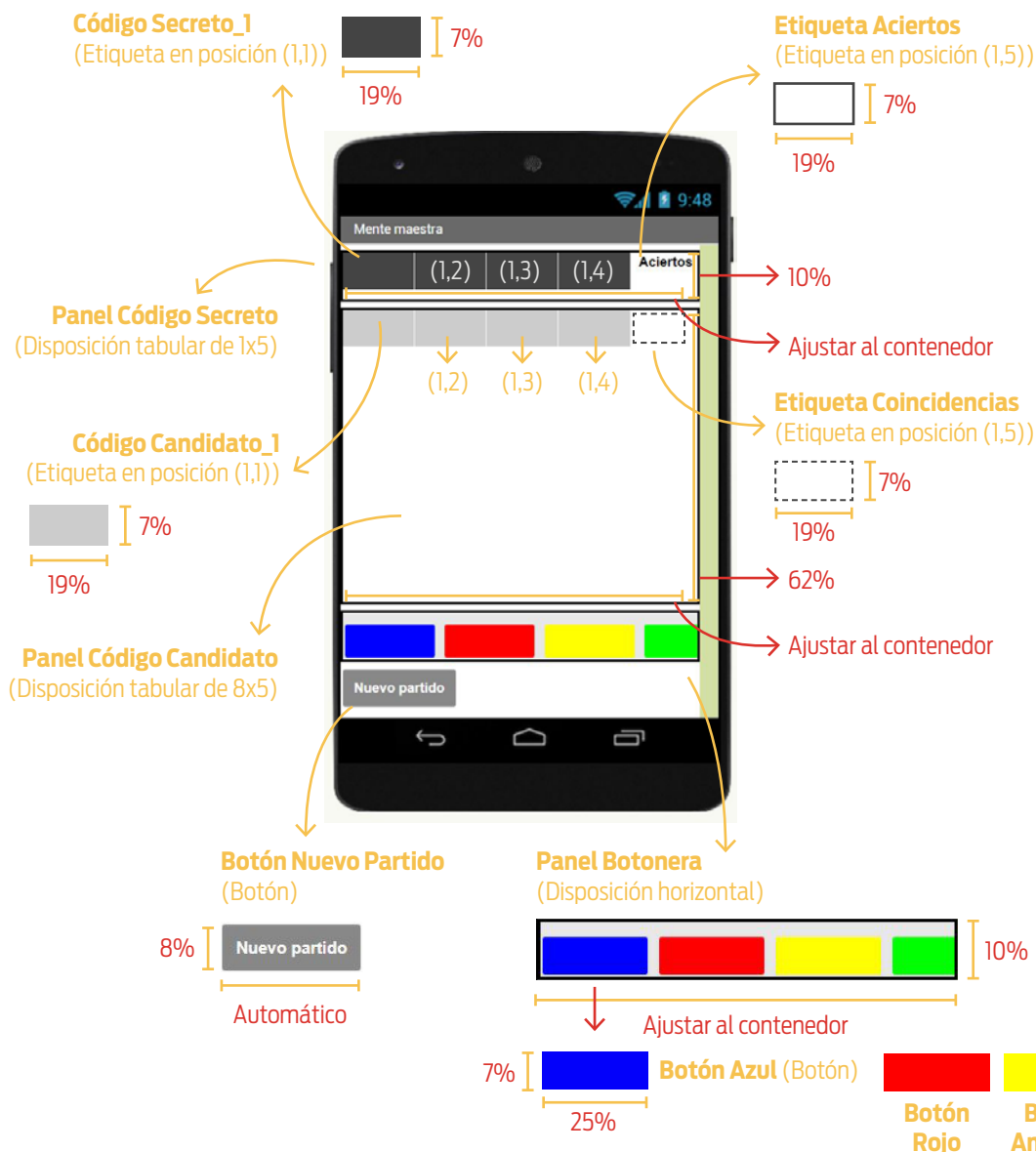
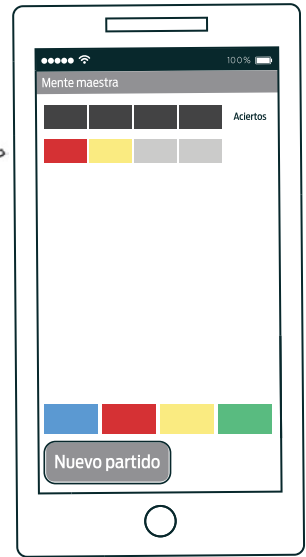
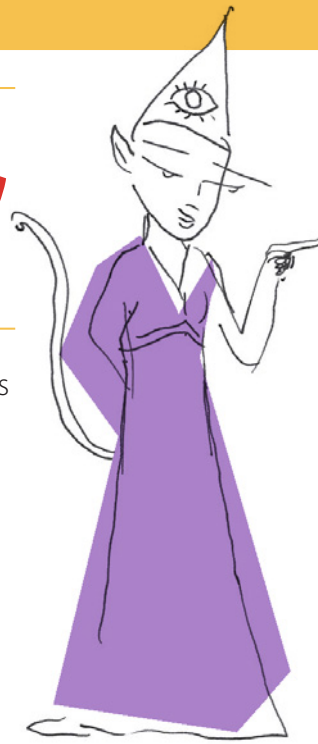
CURSO:

FECHA:

MENTE MAESTRA, PARTE II

Ahora la cosa se pone más complicada. Los jugadores tienen un intento para adivinar un código de cuatro colores. ¡Solo una chance entre 256!

1. Comenzá armando la interfaz gráfica. Acá abajo podés ver los componentes y sus dimensiones. En la tabla, las propiedades que tenés que cambiar para que se vea igual a la que te mostramos. Seguí las indicaciones para componerla.



NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto_1 ¹	Etiqueta	Color de fondo	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato_1 ²	Etiqueta	Color de fondo	Ninguno	Gris claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

¹Las restantes etiquetas del código secreto requieren los mismos cambios.
²Las restantes etiquetas del código candidato requieren los mismos cambios.

NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

NOMBRE Y APELLIDO:

CURSO:

FECHA:

2. Ni bien comienza su ejecución, la aplicación tiene que generar al azar un código secreto de 4 posiciones. Además, a pesar de tener cuatro posiciones con colores, tenés que conservar el código en una única variable.

LISTAS

En computación, una **lista** es un tipo de datos que representa una secuencia de elementos ordenados. Se puede pensar, por ejemplo, en listas de números, listas de colores o listas de palabras, etc. Cada elemento, además, puede aparecer más de una vez en una lista.

1 7 21 3



Lista de cuatro números



Lista de cuatro colores

Hola matungo



Lista de dos palabras



¿Usaste listas? ¿Por qué?

3. Ahora tenés que programar el manejo de los eventos que se producen cuando se presionan los botones de colores. Hacerlo involucra (i) actualizar en la pantalla el código candidato agregándole el color elegido por el jugador, y (ii) chequear si al agregar el nuevo color se completa el código candidato, caso en el cual hay que develar el código secreto, mostrar la cantidad de coincidencias y deshabilitar los botones de colores.

Acá abajo podés ver dos ejemplos: uno en el que al agregar un color no se completa el código candidato y otro en el que sí.



NOMBRE Y APELLIDO:

CURSO:

FECHA:

¿Cómo hiciste para que se calcule la cantidad de aciertos una vez que el jugador ingrese los cuatro colores del código candidato? ¿Usaste algún bloque que permita repetir secuencias de instrucciones? ¿Cuál?

4. Por último, el botón *Nuevo partido* tiene que dejar todo listo para intentar adivinar el código secreto otra vez. ¿Reusaste alguna parte del programa para conseguir el objetivo? ¿Cuál? ¿Para qué?

Actividad 4






Mente maestra, parte III

DE A DOS

OBJETIVOS

- Construir una aplicación partiendo de una versión previa.
- Presentar los operadores lógicos de disyunción y conjunción.
- Introducir las funciones.

MATERIALES

-  Computadora
-  Internet
-  MIT App Inventor 2
-  Ficha para estudiantes
-  Teléfono con Android (opcional)

DESARROLLO

Es habitual que las aplicaciones evolucionen y, a medida que pasa el tiempo, aparezcan nuevas versiones que cambian o agregan funcionalidades respecto a sus predecesoras. En estos casos, generalmente, el equipo que desarrolla el *software* no construye el programa desde cero; el punto de partida suele ser una versión previa de la aplicación.

En esta actividad, la propuesta es modificar el programa de la actividad “Mente maestra, parte II” para llegar a la versión final del juego, en la que los jugadores cuentan con 8 intentos para descifrar el código secreto.¹

En el desarrollo de la actividad tomaremos como punto de partida la solución de “Mente maestra, parte II” presentada en el manual. En caso de querer que los estudiantes utilicen la desarrollada por ellos, la ficha debe ser adaptada.

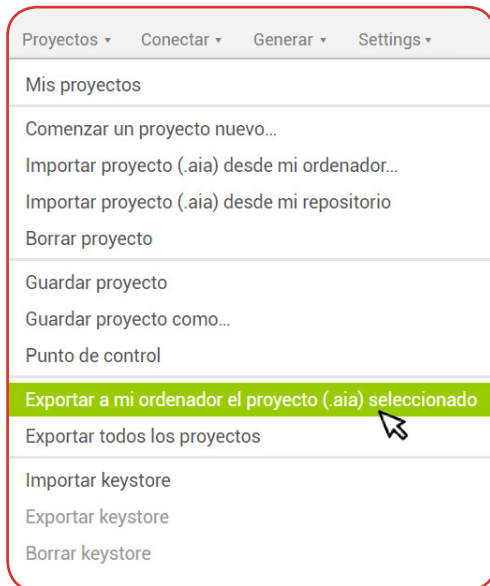
Preparación de la actividad

Antes de dar comienzo a la actividad debemos distribuir entre los estudiantes la solución de “Mente maestra, parte II” presentada en el manual. En primer lugar, para obtenerla, debemos seguir los siguientes pasos: (i) ingresar con nuestro usuario al entorno de App Inventor, (ii) acceder a la galería de aplicaciones y buscar “programar2020”, (iii) seleccionar la apertura del proyecto “Mente_maestra_parte_II” y cambiarle el nombre por “Mente_maestra_parte_III”, y (iv) exportar el proyecto para descargarlo. De este modo, obtendremos el archivo *Mente_maestra_parte_III.aia*, que podremos compartir con los estudiantes haciendo circular un *pendrive*, mandándolo en un correo electrónico, etc.



Se obtiene y renombra el proyecto *Mente_maestra_parte_II*

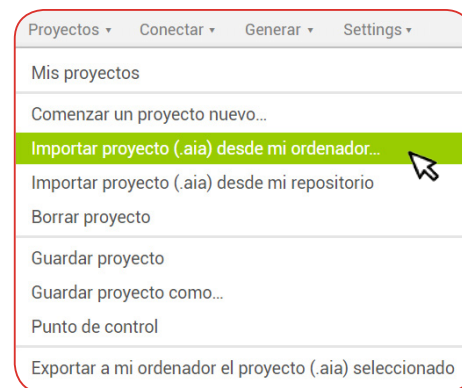
¹ Una versión completa se encuentra disponible en la galería de proyectos de App Inventor del usuario “programar2020”.



Se exporta el proyecto

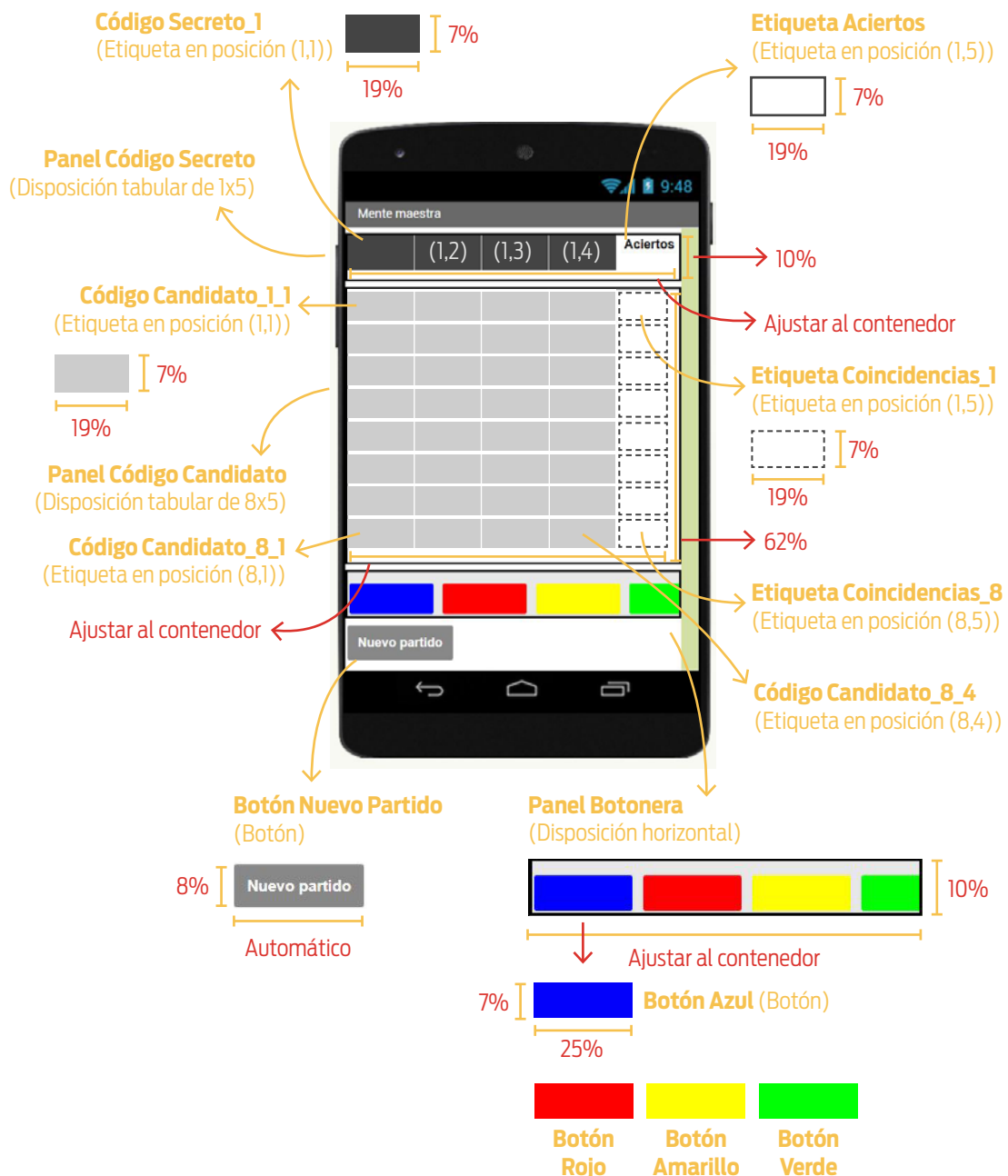
Consigna 1: interfaz gráfica

Comenzamos la actividad distribuyendo entre los estudiantes el archivo *Mente_maestra_parte_III.aia*. Luego, les indicamos que ingresen al entorno de App Inventor e importen el proyecto. Les comentamos: “Para alcanzar la versión completa del Mente maestra, este será nuestro punto de partida: una implementación de la actividad Mente maestra, parte II”.



Se importa el proyecto

A continuación, les entregamos la ficha y los invitamos a que resuelvan la primera consigna. Allí se dan instrucciones para completar la interfaz gráfica de la aplicación. La diferencia entre la interfaz de esta versión respecto de la anterior es que, en el panel de los códigos candidatos, se incorporan 7 filas de 5 etiquetas para que, sumadas a la que ya estaba, se representen 8 códigos candidatos en lugar de 1. En cada fila, las primeras cuatro etiquetas irán mostrando los colores que el usuario vaya eligiendo, y la quinta, la cantidad de coincidencias cada vez que el jugador complete un intento. Les remarcamos que en este caso es conveniente renombrar los componentes de acuerdo a lo indicado en la ficha. Por ejemplo, la etiqueta *CódigoCandidato_1* de la versión II (que correspondía al primer color del único código candidato) pasa a llamarse *CódigoCandidato_1_1*, pues ahora se trata de la primera posición del primer código candidato. Con el primer número indicaremos el número de intento y, con el segundo, la posición dentro de un código candidato. Por ejemplo, *CódigoCandidato_3_4* hará referencia a la cuarta posición del tercer código candidato.



Diseño de la interfaz

La tabla a continuación muestra los valores de las propiedades que hay que cambiar (en relación con los que App Inventor asigna por defecto) para que la aplicación se vea tal como en la imagen.

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto_1 ¹	Etiqueta	Color	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato_1_1 ²	Etiqueta	Color de fondo	Ninguno	Gris claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias_1 ³	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

¹Las restantes etiquetas del código secreto requieren los mismos cambios.

²Las restantes etiquetas del código candidato requieren los mismos cambios.

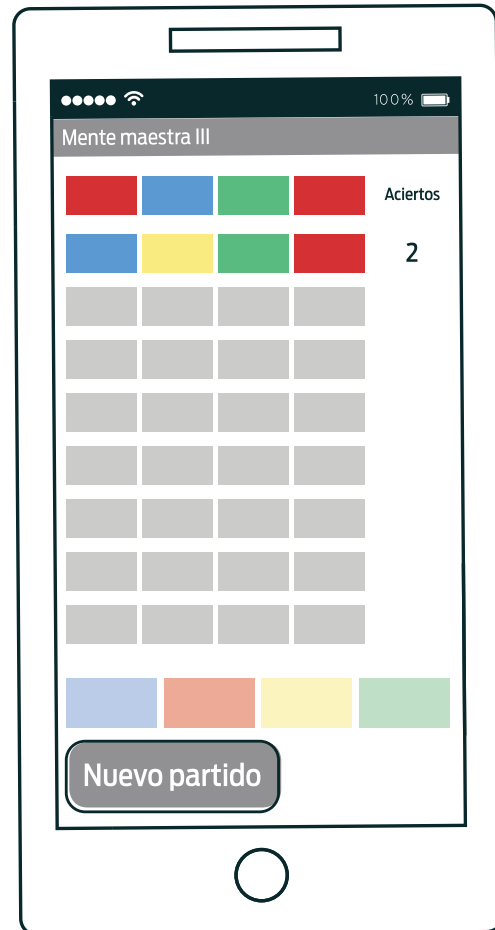
³Las restantes etiquetas para mostrar la cantidad de aciertos de un intento requieren los mismos cambios.

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

Valores diferentes a los asignados por defecto

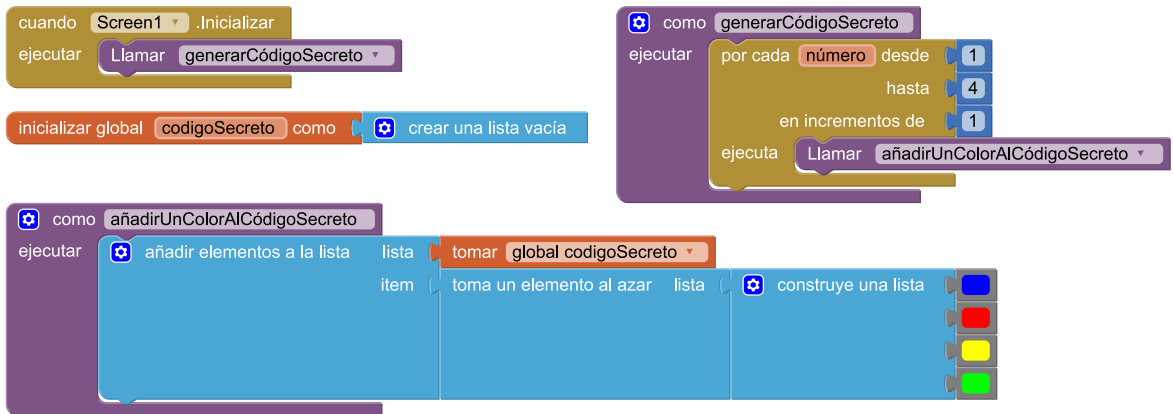
Consigna 2: ejecución de la versión inicial

Una vez que los estudiantes hayan compuesto la interfaz de la aplicación, les indicamos que la ejecuten y jueguen una partida. Les damos un pequeño tiempo para que la prueben y les preguntamos: “¿Cómo les fue con el juego? ¿Con qué se encontraron?”. Si bien al ejecutarla aparecen en la pantalla las etiquetas que representan los 8 códigos candidatos, el comportamiento de la aplicación es el mismo que el de la versión anterior: una vez ingresados cuatro colores, el juego finaliza. Continuamos: “Más allá de que al ver la pantalla nos da la impresión de que vamos a poder hacer 8 intentos, lo cierto es que luego del primero se deshabilitan los botones de colores y, entonces, lo único que podemos hacer es comenzar un nuevo partido. ¿Por qué está pasando esto?”. Escuchamos las respuestas de los estudiantes y guiamos la discusión para concluir que solo se ha cambiado el aspecto, pero no los bloques que definen el comportamiento del programa. “En definitiva, las instrucciones de un programa son las que determinan su funcionamiento, no los componentes de la interfaz gráfica”.



El juego finaliza luego del primer intento

Continuamos: “Si bien en esta versión de la aplicación vamos a permitir ocho intentos para descifrar el código secreto (en lugar de uno), ¿no hay algún requerimiento que coincida con los de la versión previa?”. Escuchamos sus comentarios y preguntamos: “¿Qué tiene que pasar, en ambas versiones, cuando comienza su ejecución?”. Se debe generar el código secreto. “¡Claro! Y eso, ¿hace falta programarlo de nuevo? ¡Por supuesto que no!”.

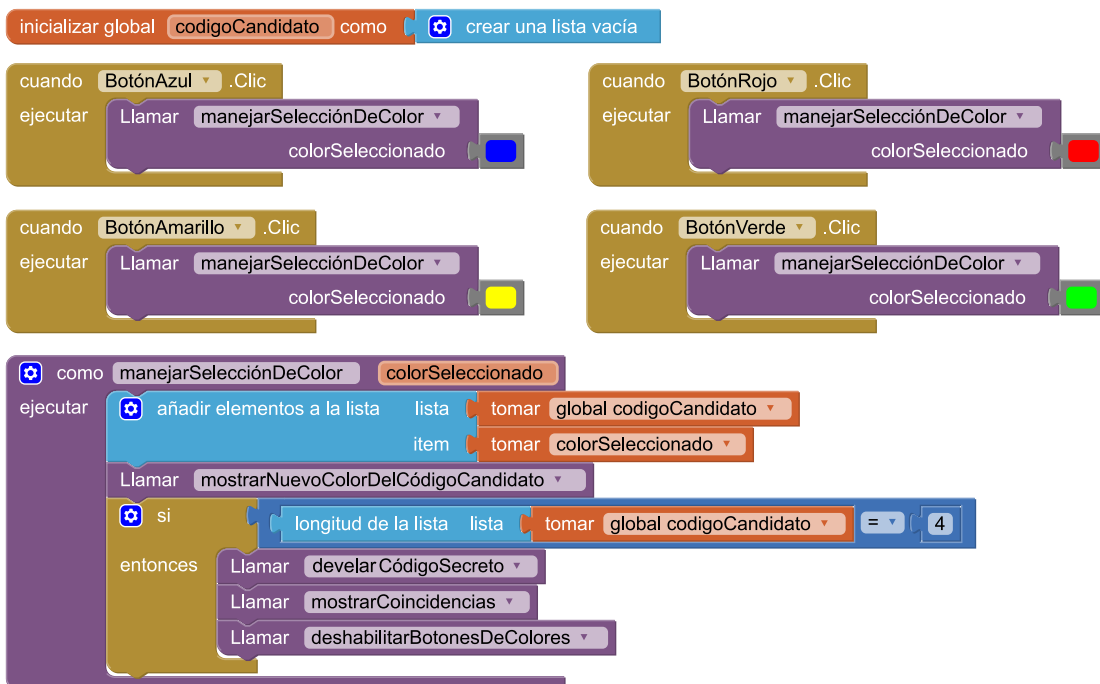


Fragmento del programa que genera el código secreto

Los invitamos a que inspeccionen por dentro el programa –desde el editor de bloques– y a que observen cómo está programado.

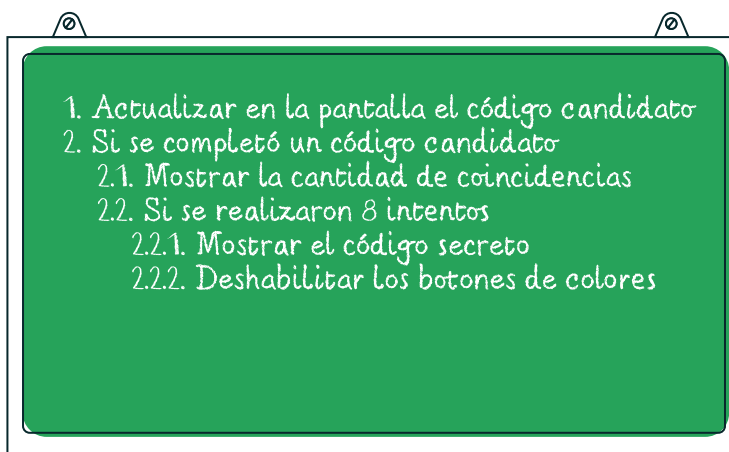
Consigna 3: los ocho intentos

Luego de hacer una puesta en común, les proponemos a los estudiantes que observen en el editor de bloques cómo está programado el manejo de eventos de los botones de colores: cuando se hace clic en cualquiera de ellos, se invoca a `manejarSelecciónDeColor (colorSeleccionado)`.



Manejo de eventos de la versión anterior del Mente maestra

Les comentamos a los estudiantes: “En la versión anterior del Mente maestra, una vez que un jugador agregaba un color al código candidato había que mostrarlo en pantalla y, además, si se completaba el código propuesto se revelaba el código secreto, se mostraba la cantidad de coincidencias y se deshabilitaban los botones de colores. En la nueva versión, tenemos que permitir ocho intentos. ¿Alguno se anima a contar cómo tendría que ser ahora el comportamiento de la aplicación?”. Guiamos el intercambio para llegar a la conclusión de que, cuando se agrega un color hay que mostrarlo en la pantalla; si se completa un código candidato, hay que mostrar la cantidad de coincidencias; y solo si se han realizado los ocho intentos, hay que mostrar el código secreto y deshabilitar los botones de colores. A continuación, copiamos en el pizarrón las ideas delineadas.



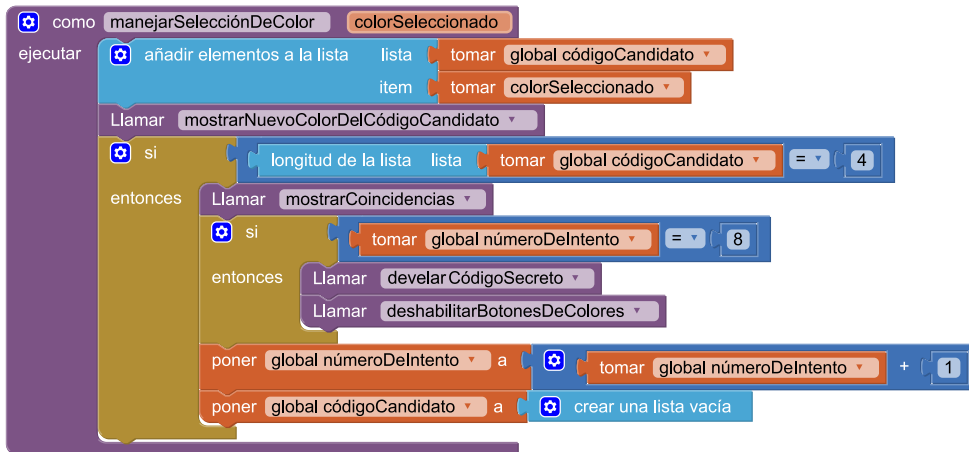
Descripción del comportamiento esperado de la aplicación

Les indicamos a los estudiantes que resuelvan la tercera consigna, que pide modificar el programa para que se comporte de acuerdo a lo descrito. Para resolver el desafío, deben notar que para poder determinar si se han completado los ocho intentos hace falta una nueva variable global que indique cuál es el número de intento que está realizando el jugador. Podría llamarse, por ejemplo, `númeroDeIntento` y ser inicializada con un 1 –cuando el juego comienza el jugador puede realizar su primer intento–.

inicializar global `númeroDeIntento` como `1`

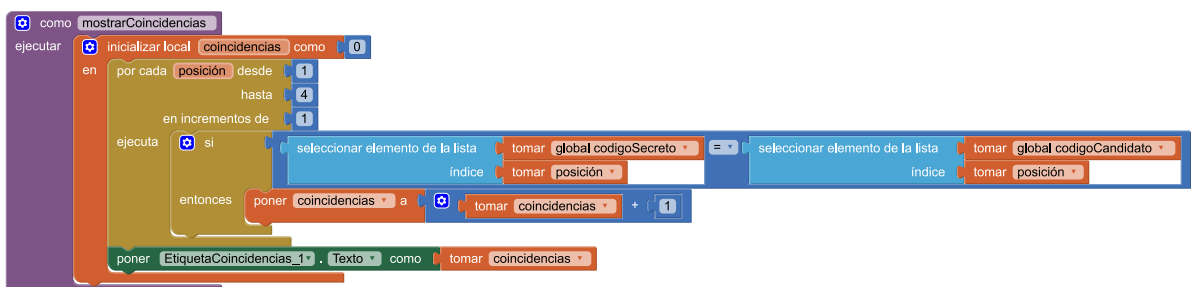
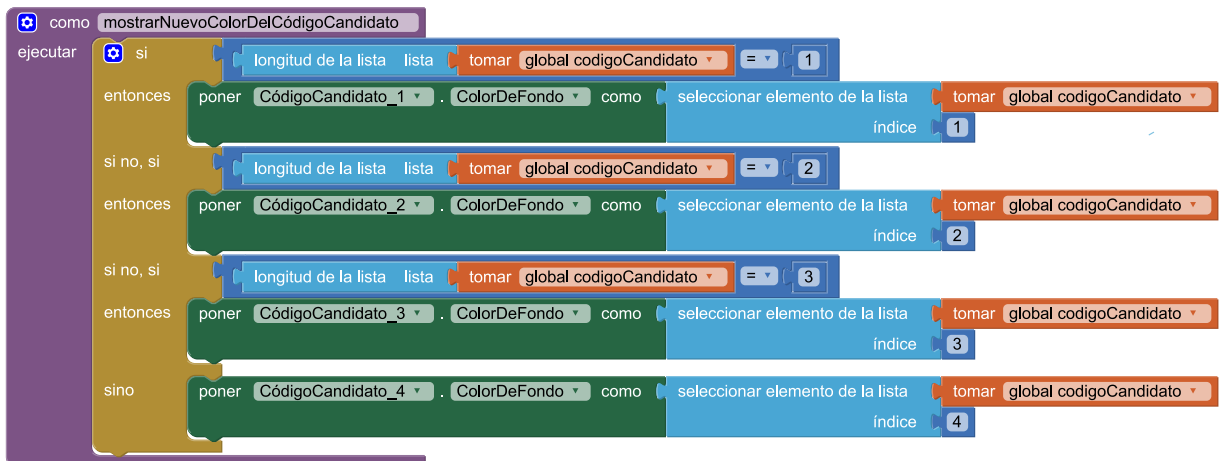
Variable que conserva el número de intento del jugador

Además, para completar la nueva versión de `manejarSelecciónDeColor` (`colorSeleccionado`), hay que tener en cuenta que cada vez que se completa un código candidato hay que: (i) sumarle 1 a `númeroDeIntento` –el jugador comenzará el siguiente intento–, y (ii) vaciar la lista que representa el código candidato –el nuevo código se arma desde cero, es independiente del anterior–.



Nueva versión de `manejarSelecciónDeColor` (`colorSeleccionado`)

Tal como vienen dados –por la versión anterior–, los procedimientos `mostrarNuevoColorDelCódigoCandidato` y `mostrarCoincidencias` solo modifican la primera fila de etiquetas del panel para los códigos candidatos. Por lo tanto, si se ejecuta la aplicación, no se mostrarían los ocho intentos uno debajo del otro.



Procedimientos de la versión anterior de la aplicación

En ambos casos, para determinar la etiqueta que hay que actualizar, alcanza con incorporar al análisis el número de intento del jugador. Así, por ejemplo, si el jugador ingresa la primera posición del segundo intento, hay que actualizar el color de la primera etiqueta de la segunda fila (*CódigoCandidato_2_1*); y si se trata de la cuarta posición del sexto intento, el color de la cuarta etiqueta de la sexta fila (*CódigoCandidato_6_4*) y el valor de las coincidencias (*EtiquetaCoincidencias_6*).



Fragmento de `mostrarNuevoColorDelCódigoCandidato`

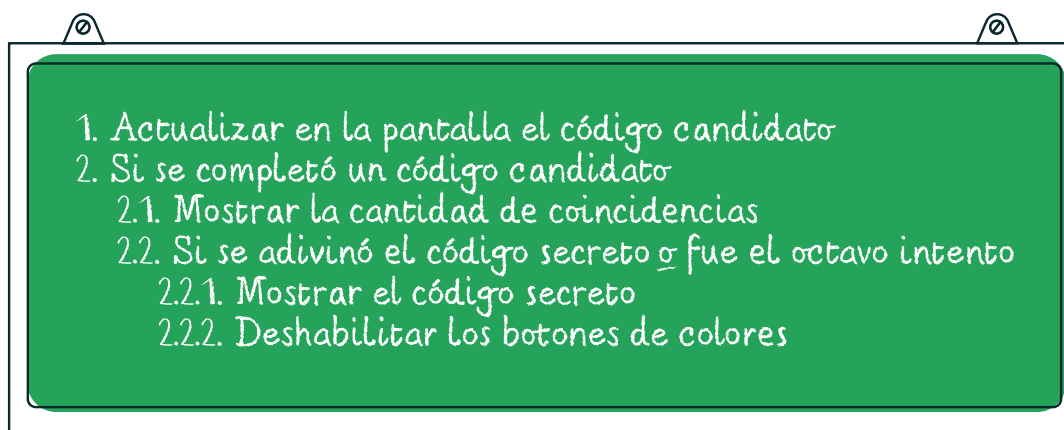


Fragmento de `mostrarCoincidencias`

Consigna 4: al ganar, ya está

La cuarta consigna pide que el juego, además de terminar cuando se hayan alcanzado los ocho intentos, también finalice cuando el código secreto sea descubierto.

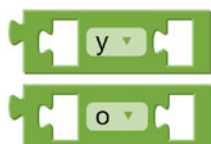
Antes de que comiencen a resolver el desafío, les proponemos diseñar grupalmente una solución que tenga en cuenta el nuevo requerimiento. Orientamos la discusión para concluir que, entonces, una partida puede finalizar por dos motivos: se descubre el código o se alcanzan ocho intentos. Copiamos en el pizarrón la estrategia que se muestra en la figura y los instamos a que modifiquen sus programas para resolver la consigna.



El juego también finaliza si se descubre el código secreto

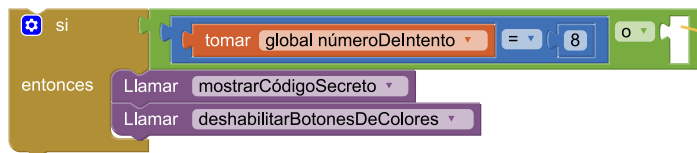
Lo primero que deben advertir los estudiantes es que, para resolver el desafío, es necesario modificar el procedimiento `manejarSelecciónDeColor` (`colorSeleccionado`). Hasta aquí, el juego solo finaliza cuando se alcanzan los ocho intentos. Resta, entonces, construir una expresión que denote la condición "se adivinó el código secreto o se alcanzó el octavo intento".

Al inspeccionar el entorno, los estudiantes observarán en *Bloques > Integrados > Lógica* dos bloques para armar expresiones booleanas a partir de otras más simples: `[] y []` y `[] o []`, que corresponden a las operaciones lógicas de conjunción y disyunción respectivamente. Dadas dos condiciones `c1` y `c2`, `[c1] y [c2]` es verdadera únicamente si ambas (`c1` y `c2`) lo son; en cambio, para que `[c1] o [c2]` sea verdadera, alcanza con que alguna de las dos sea verdadera. Si hiciese falta, orientamos a los estudiantes en el uso de estos dos bloques.



Bloques para conjunción y disyunción lógica

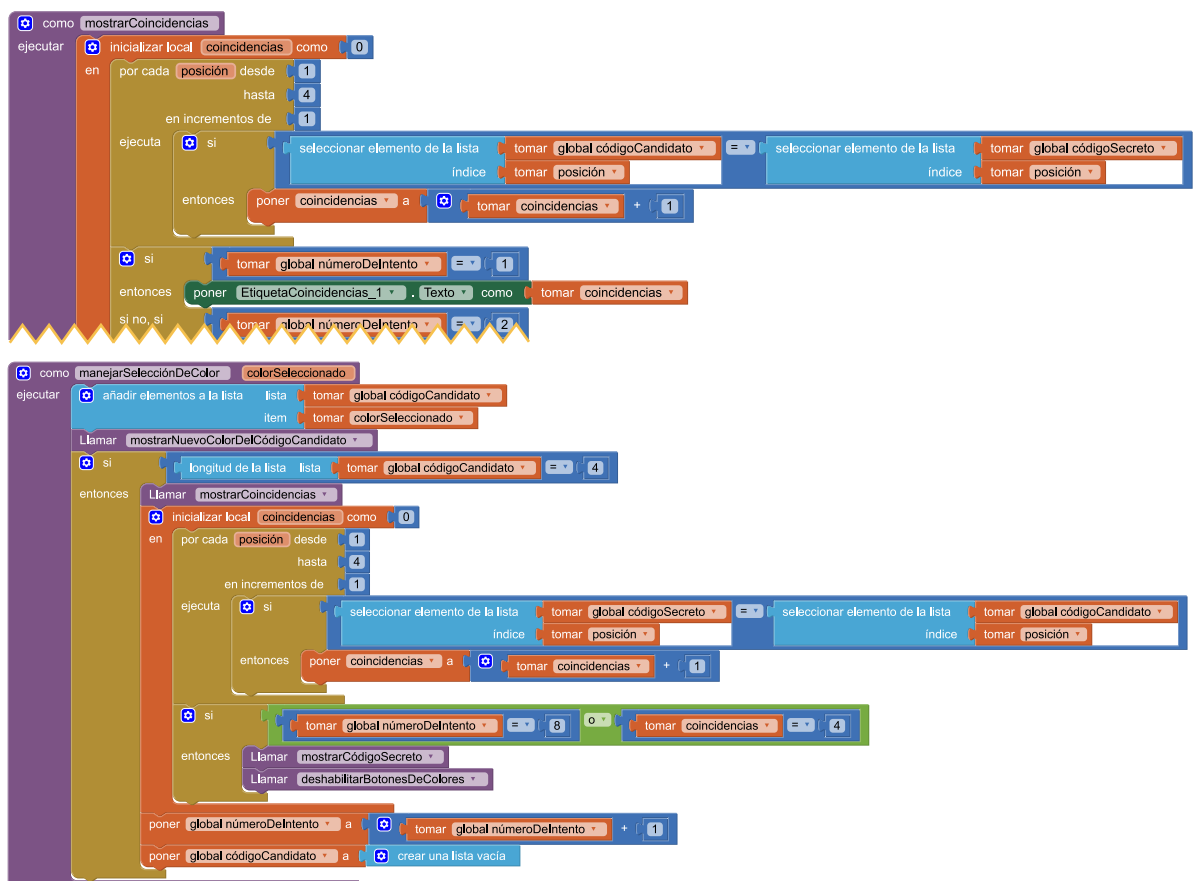
Una vez identificado el bloque de la disyunción lógica, se lo puede incorporar a la alternativa condicional.



Falta expresión que denote que se adivinó el código secreto

Se incorpora la disyunción a la alternativa

Resta definir la expresión que refleje que el código secreto ha sido descubierto. O, equivalentemente, que los cuatro colores del código candidato coincidan con los del código secreto. En `mostrarCoincidencias` se realiza un conteo de las posiciones en las que ambos códigos son iguales, con lo que es probable que algunos estudiantes armen un esquema similar dentro de `manejarSelecciónDeColor` (`colorSeleccionado`).



Reproducción del enfoque para cálculo de coincidencias

El procedimiento `manejarSelecciónDeColor` (`colorSeleccionado`) es funcionalmente correcto, aunque adolece de un problema grave: al observarlo, no resulta sencillo comprender qué hace. A aquellos estudiantes que hayan propuesto esta solución (u otra similar), les hacemos notar que en `mostrarCoincidencias` y `manejarSelecciónDeColor` (`colorSeleccionado`) hay fragmentos muy similares y les indicamos que piensen una solución alternativa.

Es probable que, a partir de reconocer que el conjunto de bloques que se repite en ambos procedimientos, lo primero que piensen sea extraer la porción común en un procedimiento nuevo (al que podrían llamar, por ejemplo, `calcularCoincidencias`) e invocarlo tanto desde `mostrarCoincidencias` como desde `manejarSelecciónDeColor` (`colorSeleccionando`).

La variable `coincidencias` solo existe dentro de `calcularCoincidencias`

La variable `coincidencias` solo existe dentro de `calcularCoincidencias`

Las variables locales de un procedimiento no pueden ser referenciadas en otro procedimiento

Como la variable `coincidencias` solo existe dentro del procedimiento `calcularCoincidencias`, no puede ser referenciada ni desde `mostrarCoincidencias` ni desde `manejarSelecciónDeColor` (`colorSeleccionando`). Frente a este problema, algún estudiante podría proponer que `coincidencias` sea una variable global (en lugar de local).

Uso inadecuado de una variable global

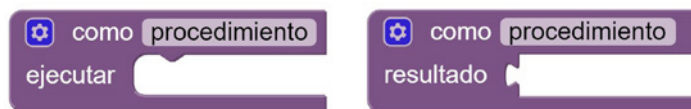
Uso inadecuado de una variable global

Se guardan las coincidencias en una variable global

Si bien de este modo el programa funciona, el uso de variables globales solo se justifica cuando hay un valor que va evolucionando a lo largo de la ejecución de un programa (por ejemplo, el puntaje que va acumulando un jugador al desarrollarse una partida de un juego). Sin embargo, en esta propuesta, se usa una para conservar el resultado de un cálculo, cuyo valor solo tiene una relevancia local (inmediatamente después de que `calcularCoincidencias` es invocado). En caso de que los estudiantes propongan resolver el problema usando una variable global, les sugerimos que exploren el entorno de App Inventor en busca de una solución alternativa.

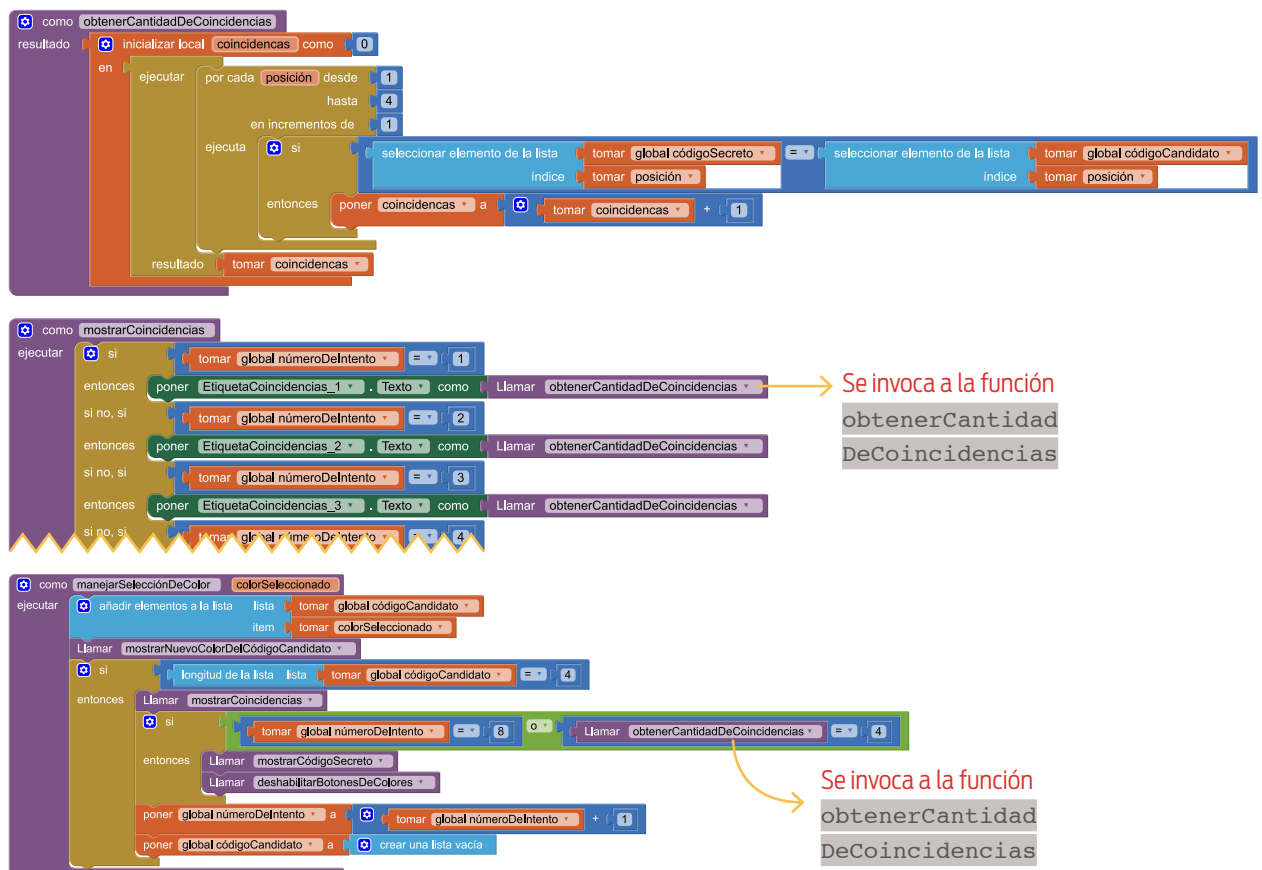
En *Bloques > Integrados > Procedimientos* se encuentran los bloques `como (procedimiento) / ejecutar { }` y `como (procedimiento) / resultado []`. Mientras que el primero se utiliza

para realizar una tarea, el segundo se usa para calcular y retornar un valor. En los lenguajes de programación, a estos últimos se los refiere como **funciones**,¹ cuya característica principal es que producen nueva información relevante para un programa.



Bloques para definir un procedimiento y una función

Para calcular la cantidad de coincidencias entre un código candidato y el código secreto, lo adecuado es resolverlo en una función, que calcule y retorne el valor buscado (podría llamarse `obtenerCantidadDeCoincidencias`). A continuación se muestra una posible solución.

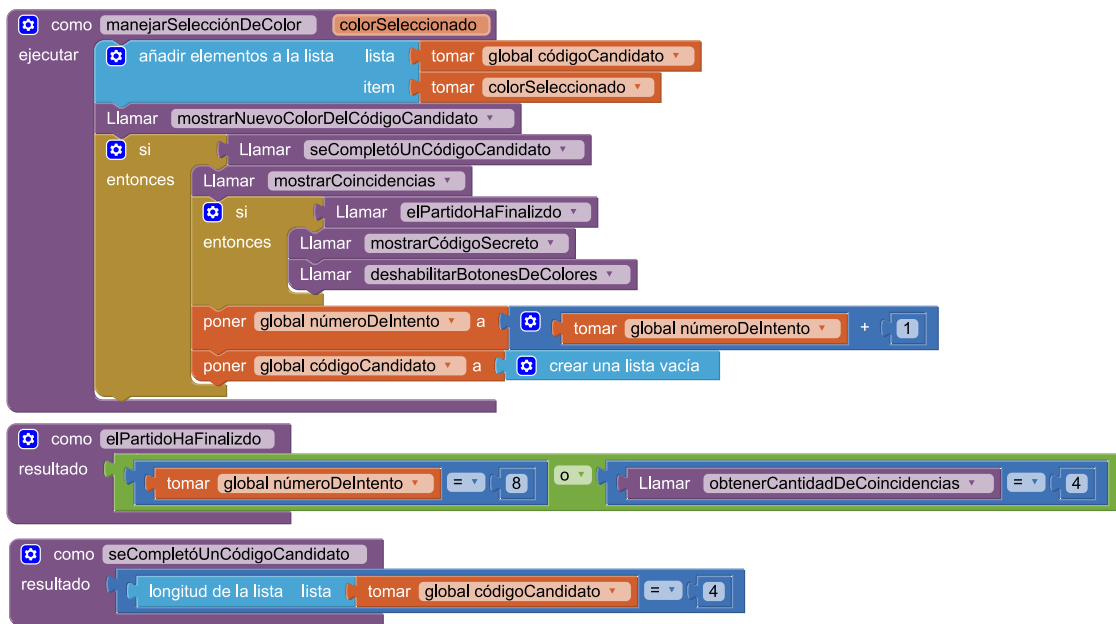


Solución que incorpora una función

Como se ha visto, los procedimientos se usan para encapsular un conjunto de acciones que componen una unidad de sentido y, al ponerles un nombre adecuado, se obtiene un programa cuya lectura resulta más clara. Con las funciones puede hacerse algo similar y, también, obtener como resultado un programa que sea más fácil de entender. En la figura a continuación se muestra una solución que incorpora dos

¹En App Inventor no se hace una diferenciación explícita entre procedimientos y funciones; a ambos se los llama procedimientos.

nuevas funciones: una que chequea si se ha completado un código candidato y, otra, que chequea si ha finalizado la partida. De este modo, al leer el procedimiento `manejarSelecciónDeColor` (`colorSeleccionado`) resulta sencillo comprender lo que hace.

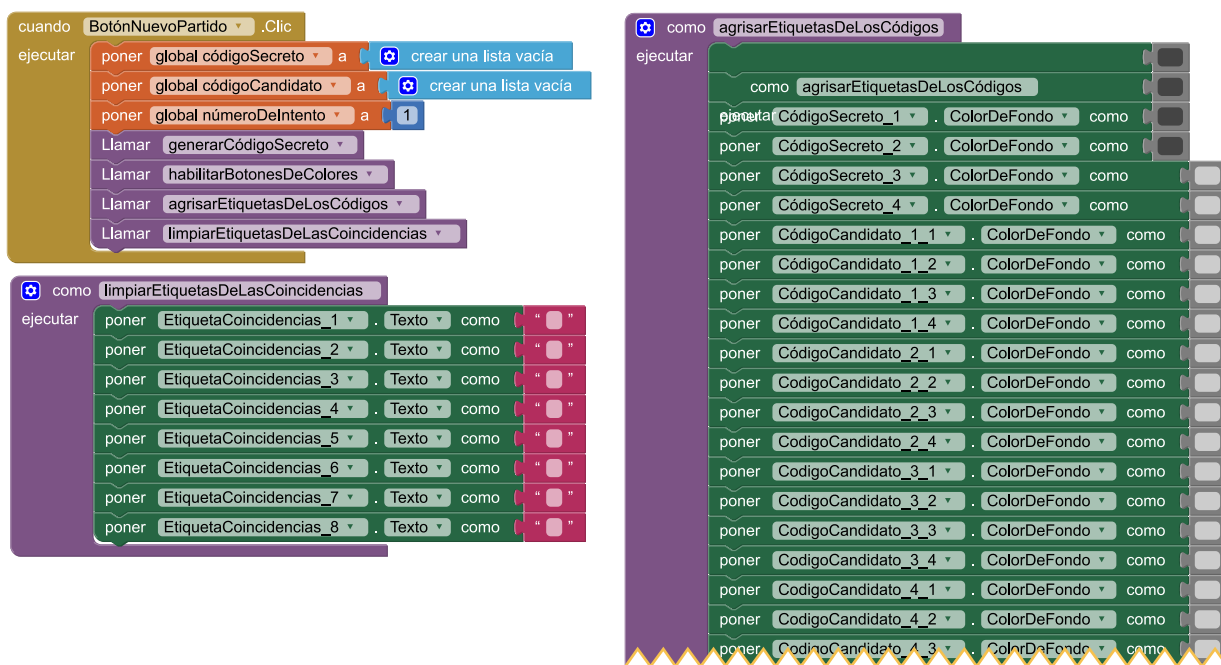


Definición de nuevas funciones

Consigna 5: un nuevo partido

La quinta y última consigna pide que, al apretar el botón *Nuevo partido*, todo quede dispuesto para empezar una nueva partida. La solución es muy similar a la que ya tienen de la actividad anterior. Concretamente hay que: (i) vaciar las listas `códigoSecreto` y `códigoCandidato` y establecer `númeroDeIntento` en 1, (ii) generar un nuevo código secreto, (iii) agrisar las etiquetas que muestran los códigos en la pantalla, y (iv) vaciar los textos de las etiquetas que muestran las cantidades de coincidencias entre los distintos intentos de un usuario y el código secreto.

Para resolver el desafío se pueden usar los procedimientos `generarCódigoSecreto` y `habilitarBotonesDeColores` tal como vienen dados desde la versión anterior de la aplicación. Por su parte, puede adaptarse `agrisarEtiquetasDeLosCódigos` para incluir las etiquetas de los ocho códigos candidatos; y crear un nuevo procedimiento `limpiarEtiquetasDeCoincidencias` que deje vacíos los textos de todas las etiquetas que muestran la cantidad de posiciones en que coinciden los códigos candidatos y el código secreto.



Reinicio del Mente maestra

CIERRE

Repasamos con los estudiantes que, como para determinar si una partida finaliza hubo que chequear si se verificaba al menos una de dos condiciones (se completaron los ocho intentos o se descubrió el código secreto), en esta actividad hemos utilizado el operador lógico de disyunción. Además, comentamos que cuando se tenga que chequear si se cumplen dos condiciones (y no al menos una de las dos), hay que usar el operador de conjunción. Finalmente, subrayamos que, así como los procedimientos permiten encapsular acciones, las funciones permiten encapsular operaciones que retornan un valor; es decir, cálculos que producen información.

NOMBRE Y APELLIDO:

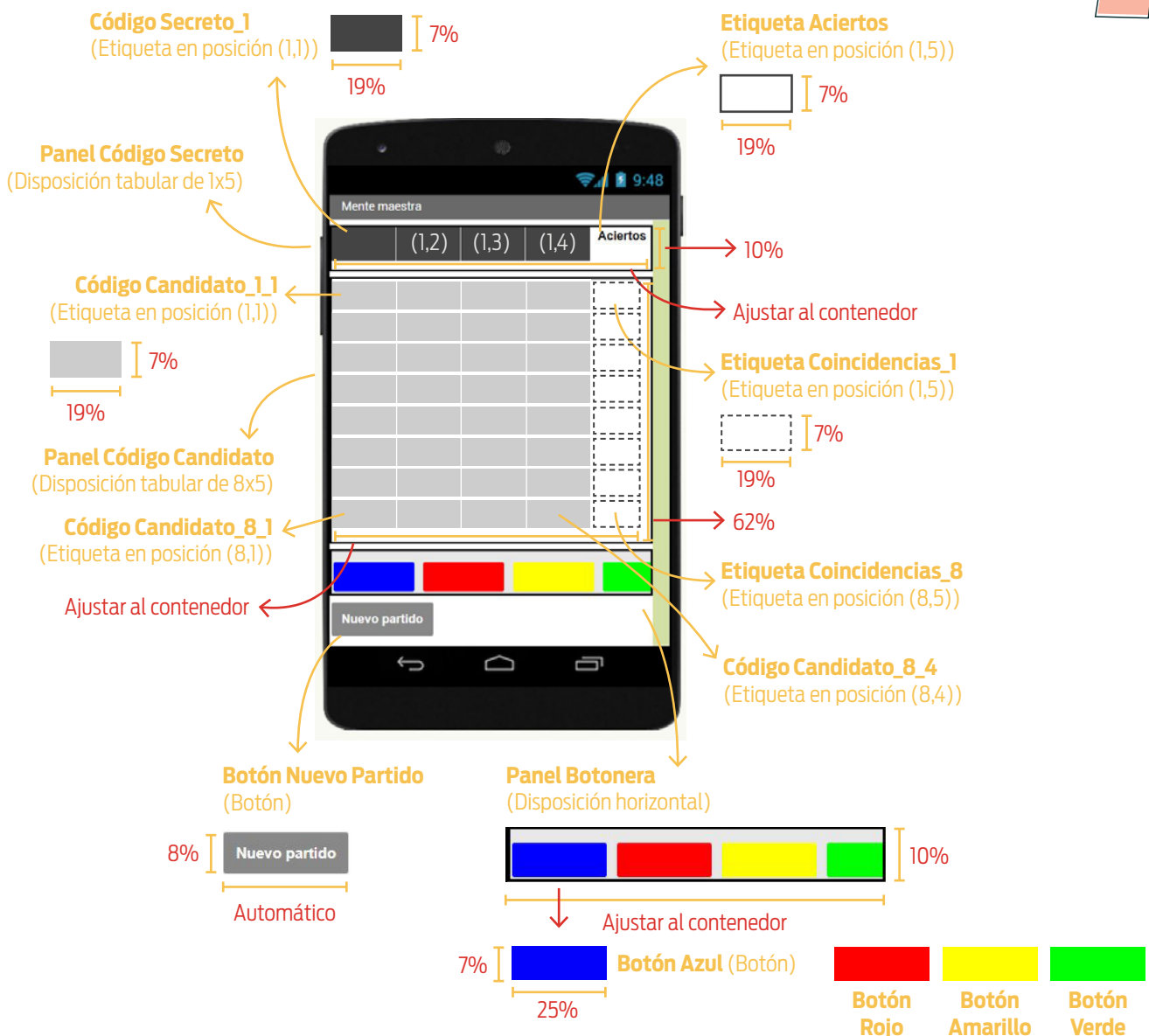
CURSO:

FECHA:

MENTE MAESTRA, PARTE III

Y ahora sí: códigos de cuatro colores y ocho intentos. ¡A armar la versión completa del Mente maestra!

1. Comenzá armando la interfaz gráfica. En la imagen están los componentes y sus dimensiones. En la tabla, las propiedades que tenés que cambiar para que se vea igual a la que te mostramos. Seguí las indicaciones para componerla.



NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Screen1	Pantalla	Título	Screen1	Mente maestra
Panel Código Secreto	Disposición tabular	Columnas	2	5
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	1
Código Secreto_1 ¹	Etiqueta	Color	Ninguno	Gris oscuro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Aciertos	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	Aciertos
		Posición del texto	Izquierda	Centro
Panel Código Candidato	Disposición tabular	Columnas	2	5
		Alto	Automático	62%
		Ancho	Automático	Ajustar al contenedor
		Registros	2	8
Código Candidato_1_1 ²	Etiqueta	Color de fondo	Ninguno	Gris claro
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
Etiqueta Coincidencias_1 ³	Etiqueta	Negrita	(ninguno)	✓
		Alto	Automático	7%
		Ancho	Automático	19%
		Texto	Texto para etiqueta	(ninguno)
		Posición del texto	Izquierda	Centro

¹Las restantes etiquetas del código secreto requieren los mismos cambios.

²Las restantes etiquetas del código candidato requieren los mismos cambios.

³Las restantes etiquetas para mostrar la cantidad de aciertos de un intento requieren los mismos cambios.

NOMBRE Y APELLIDO:

CURSO:

FECHA:

NOMBRE SUGERIDO	TIPO DE COMPONENTE	PROPIEDAD	VALOR PREVIO	VALOR NUEVO
Panel Botonera	Disposición horizontal	Disp. horizontal	Izquierda	Centro
		Disp. vertical	Arriba	Abajo
		Alto	Automático	10%
		Ancho	Automático	Ajustar al contenedor
Botón Azul	Botón	Color de fondo	Por defecto	Azul
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Rojo	Botón	Color de fondo	Por defecto	Rojo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Amarillo	Botón	Color de fondo	Por defecto	Amarillo
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Verde	Botón	Color de fondo	Por defecto	Verde
		Alto	Automático	7%
		Ancho	Automático	25%
		Texto	Texto para botón	(ninguno)
Botón Nuevo Partido	Botón	Color de fondo	Por defecto	Gris
		Negrita	(ninguno)	✓
		Alto	Automático	8%
		Texto	Texto para botón	Nuevo partido
		Color de texto	Por defecto	Blanco

NOMBRE Y APELLIDO:

CURSO:

FECHA:

- 2.** Ejecutá la aplicación. ¿Con qué te encontraste? ¿Cuántos intentos tuviste para descubrir el código secreto? ¿Por qué?

- 3.** El juego tienen que permitir que un usuario cuente con ocho intentos para develar los cuatro colores ocultos. Usando lenguaje coloquial, delineá una estrategia para conseguirlo.

Programá en App Inventor la estrategia propuesta.

UNA AYUDA

El procedimiento `manejarSelecciónDeColor (colorSeleccionado)` describe la estrategia de la versión anterior, en la que solo había un intento. A lo mejor te sirve como referencia.



NOMBRE Y APELLIDO:

CURSO:

FECHA:

4. No solo cuando se completan los ocho intentos el juego tiene que terminar. ¡Si el jugador descubre el código secreto tampoco tiene sentido seguir! Modificá el programa para que también finalice cuando el jugador triunfa.

PARA QUE TENGAS EN CUENTA

En App Inventor hay dos bloques para armar expresiones booleanas que se construyen a partir de otras más simples: `[] y []` y `[] o []`. La primera corresponde a la operación lógica de **conjunción**, y la segunda, a la de **disyunción**. ¿Te imaginás cómo usar alguna de ellas para resolver la consigna?



5. Por último, ocupate de que cuando se presione el botón *Nuevo partido* se pueda jugar otra vez. ¿Qué cambios tuviste que hacer para conseguirlo?

PROCEDIMIENTOS Y FUNCIONES

¿Sabés cuál es la diferencia entre `como (procedimiento) / ejecutar { }` y `como (procedimiento) / resultado []`? Mientras que el primero se utiliza para realizar una tarea, el segundo se usa para calcular y retornar un valor. En los lenguajes de programación, a estos últimos se los refiere como **funciones**, cuya característica principal es que producen información nueva que es relevante para un programa. ¡Incorporá funciones en tu programa y elegí concienzudamente sus nombres!

