IMPLEMENTACIÓN

Proyecto: My First Programming Language (FiPL)

INTRODUCCIÓN
Este documento explica el proceso de la fase de implementación del sistema a desarrollar para el Proyecto Fin de Grado 2014-2015 My First Programming Language FiPL

PROPÓSITO
Además de producir el código fuente, javadoc, capturas de pantalla y videos del programa en ejecución en este documento explico las decisiones de implementación.

Para escribir el programa he usado Android Studio 1.2
https://developer.android.com/sdk/index.html

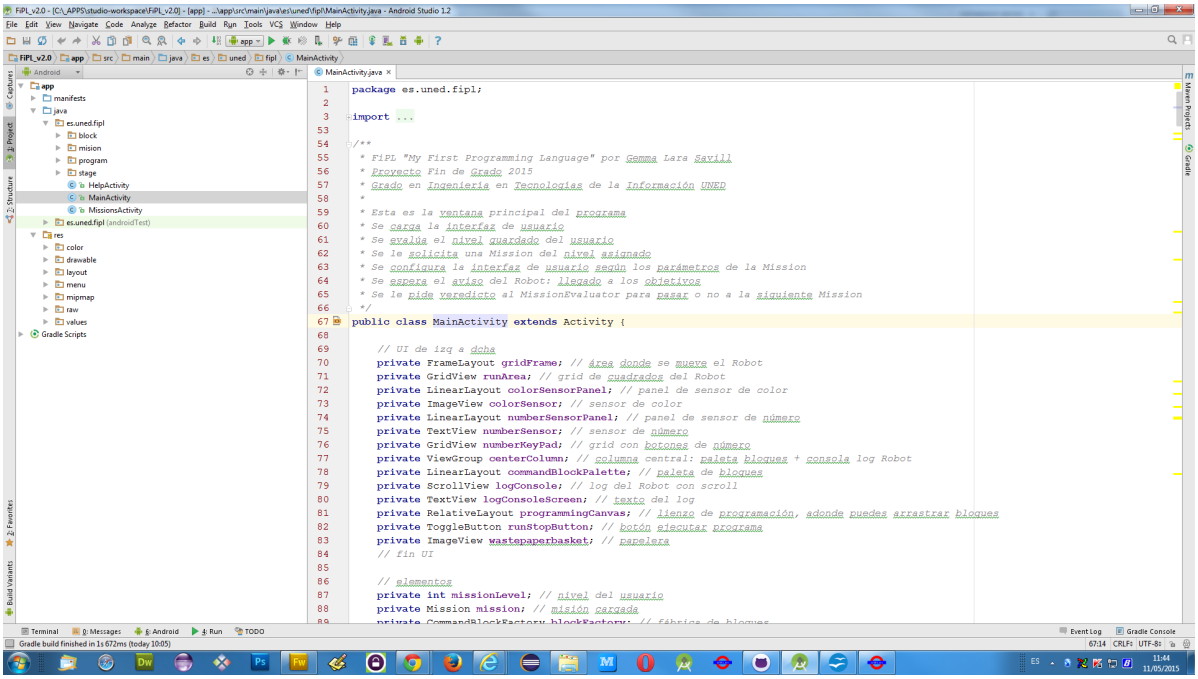


Imagen: Android Studio mostrando la estructura del programa

RESUMEN

Un ciclo completo del programa sería:

- cargar la misión
- crear los bloques necesarios y colocarlos en la paleta
- configurar el escenario del robot, colocar al Robot y el cuadrado solución
- el usuario interactúa con los bloques, arrastrando y uniéndolos
- el usuario ejecuta el programa creado mediante la manipulación de los bloques
- el Robot ejecuta el programa
- el programa informa al usuario del resultado: misión cumplida, no cumplida o Error.

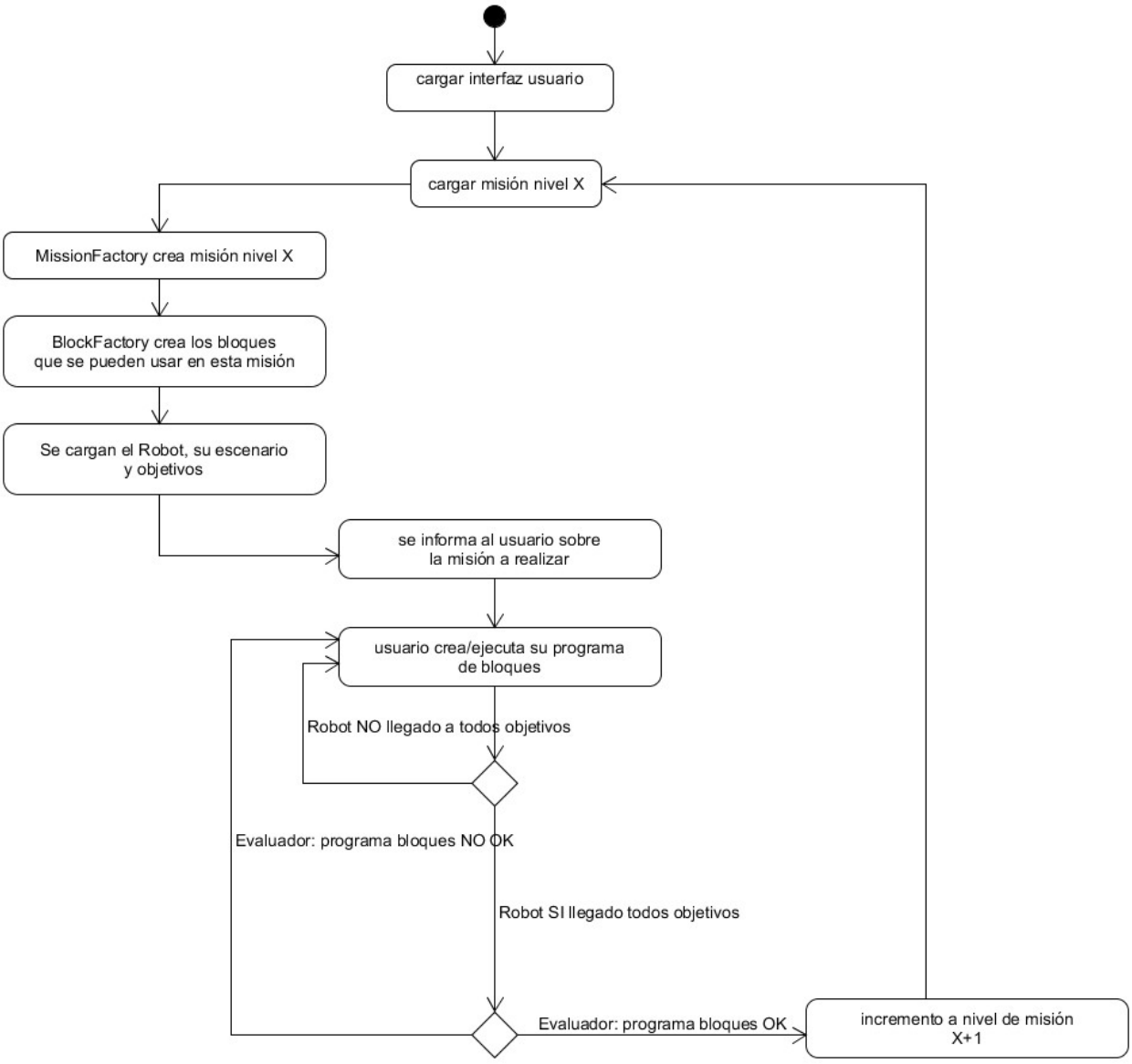


Imagen: diagrama de flujo al abrir el programa

ESTRUCTURA GENERAL DEL PROGRAMA

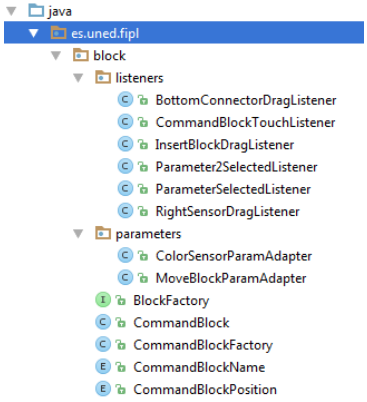
Sigue una estructura clásica de una app Android que separa muy bien la programación en Java de los recursos: gráficos, textos para el usuario, layouts en xml, etc.

En Android la convención es usar un nombre de dominio al revés para nombrar el paquete principal del programa. Yo he usado el de mi universidad: uned.es así que el paquete principal es

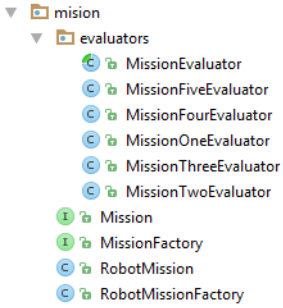
es.uned.fipl

Dentro de este paquete he dividido la programación Android (Java) en otros paquetes:

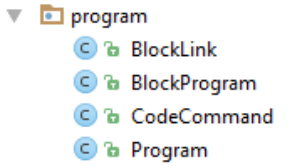
```
es.uned.fipl/
        block/              se ocupa de los bloques
```



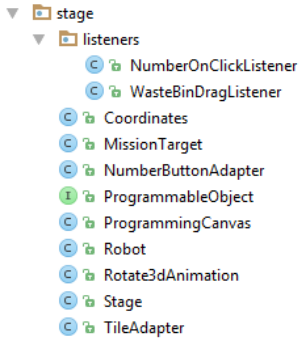

```
        mision/             todo lo referente a las misiones
```




```
        program/            todo lo referente al programa que crea el usuario
```

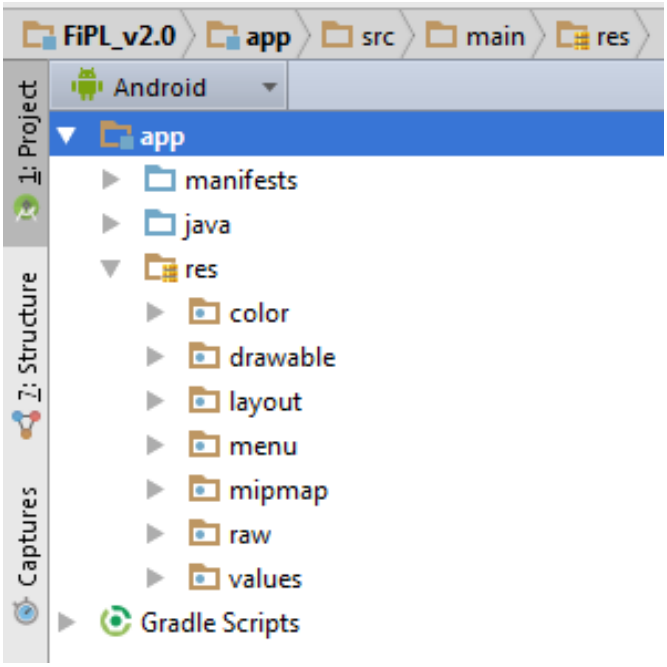



```
        stage/              los elementos relativos al escenario
```



En el proyecto además de la carpeta java (app\src\main\java) se encuentra la carpeta res (app\src\main\res) en ella se encuentran todos los elementos del programa que no son java, los resources.

La estructura aquí es estandard de Android:



res/color: colores usados en recursos gráficos
res/drawable: aquí están los recursos gráficos, ya sean png o en xml (svg)
se desglosan por densidad de píxeles
Android se ocupa de mostrar el recurso de la densidad necesaria
res/layout: aquí van los xml que representan elementos UI
res/menu: aquí van los xml que representan menus de pantallas
res/mipmap: en esta carpeta van los iconos launcher de la app, para cada resolución
res/raw: aquí están los sonidos del programación
res/values: aquí tenemos los xml de estilo y colores
xml de dimensiones independientes de densidad
strings por idioma: todos los mensajes para el usuario
incluyo los strings en inglés y español
si la tablet está en inglés el sistema mostrará inglés
y si la tablet está en español la app también lo estará

Más información en: http://developer.android.com/guide/practices/screens_support.html
Nota: faltan algunas traducciones al inglés

IMPLEMENTACIÓN

He dividido la implementación del programa en 4 fases:

1. Crear la interfaz de usuario
2. Programar la manipulación de los bloques
3. Convertir una lista de bloques en un programa y ejecutarlo en el Robot
4. Comunicación con el usuario

FASE 1. CREAR LA INTERFAZ DE USUARIO.

En Android cada pantalla debe extender la clase Activity.
Cada Activity se ocupa de crear su interfaz de usuario mediante un elemento Layout.
Al iniciarse un programa Android se carga la clase que se ha indicado como punto de inicio del programa.
En nuestro caso el punto de inicio es la clase MainActivity que crea la interfaz de la ventana especificado en el archivo /layout/activity_main.xml
Este programa es para tablets de entre 7 y 10 pulgadas, y de diferentes densidades de pantalla, así que todas las medidas en píxeles están en dp o dip (density independent pixels) ya que en una pantalla de densidad media un bloque mide 180 pixeles que en una pantalla de densidad xdpi será 320 píxeles. Los layouts o interfaces de cada pantalla también han sido diseñados para que se vean bien en pantallas más grandes (xlarge) y las medianas (large). El programa de bloques que crea el usuario lo he rodeado de una estructura ScrollView para que se pueda hacer scroll si la pantalla no es muy grande, así nunca te quedarás sin espacio para añadir tus bloques. Lo mismo con el log del Robot.

Al abrirse la primera pantalla el programa le pide al MissionFactory una misión y se configura la interfaz de usuario.

Parte izquierda de la pantalla, relativo al Robot:

- La misión nos proporciona una lista de baldosas de colores con las que creamos el suelo por donde va a moverse el Robot.
- Se crea el escenario del Robot con las baldosas y se coloca al Robot encima, en la posición 0,0.
- Se actualiza el sensor de color del Robot según el color de la baldosa bajo el Robot.
- Se cargan los botones de número y sus correspondientes Listeners.

Parte derecha de la pantalla, relativo a los Bloques:

- Se le pide al CommandBlockFactory un bloque de cada tipo indicado en la misión. Estos serán los bloques que el usuario tendrá disponible para crear su programa.
- Se añaden estos bloques a la paleta de bloques.
- Se configura el lienzo de programación, adonde se deberán arrastrar y encadenar los bloques.
- Se configuran los Listeners para arrastrar los bloques, el listener de la papelera, adonde se pueden arrastrar bloques para eliminarlos.
- Se configura el Listener del botón de Ejecutar/Detener el programa.

Ahora el programa ya está listo para la interacción del usuario.

FASE 2. PROGRAMAR LA MANIPULACIÓN DE LOS BLOQUES

El CommandBlockFactory puede crear 8 diferentes tipos de bloques. Se identifican mediante el Enumerado CommandBlockName: START, WAIT, SPEED, POSITION, MOVE, REPEAT, COLORSENSOR y NUMBERSENSOR. De esta manera es fácil añadir más tipos de bloques en un futuro.

Cada bloque puede tener hasta un máximo de 2 parámetros seleccionables por el usuario: por ejemplo un WAIT block no tiene parámetros, un MOVE block tiene uno y un POSITION block tiene dos.
Los parámetros son seleccionables por el usuario mediante selectores desplegables, en Android se llaman Spinner.

Los bloques de la paleta se pueden arrastrar. Si arrastras un bloque desde la paleta y lo enganchas con otro, éste se clona y se añade al programa de bloques.

No todos los bloques se enganchan igual, por ello tienen unos conectores tipo puzzle que indican cómo se enganchan entre ellos.

Si arrastras un bloque que esté enganchado a otro a la papelera lo puedes borrar. También los puedes reordenar, arrastrando y soltando.

En este punto del programa LA LISTA DE BLOQUES CONECTADOS que se va contruyendo sobre el lienzo de programación (ProgrammingCanvas) NO ES TRATADO COMO UN PROGRAMA, se almacena simplemente como una lista de bloques. El área que contiene a los bloques es un RelativeLayout. Esto significa que todos los elementos contenidos en él se posicionan de forma relativa unos con otros: debajo de, a la derecha de, etc.

El programa almacena la lista de bloques conectados en una estructura de datos que he llamado BlockProgram que extiende un ArrayList.

Es una lista de una estructura de datos llamada BlockLink. Cada vez que se conecta un bloque se crea un BlockLink y éste se añade al BlockProgram.

Un BlockLink se compone de un bloque y su posición relativa a otro bloque.

Al iniciarse el programa hay un bloque START como primer bloque del programa.

Al arrastrar y conectar un bloque MOVE debajo de START se creará un BlockLink: MOVE, debajo de, START

Los sensores se añadiran a la derecha, por ejemplo, COLORSENSOR, a la derecha de, MOVE

Extracto de un log mostrando cómo el programa almacena la lista de bloques:

```
05-11 11:37:41.312    2978-2978/? I/System.out: Canvas tiene 8 hijos
05-11 11:37:41.312    2978-2978/? I/System.out: Draw START[154] FIRST 2131492889
05-11 11:37:41.312    2978-2978/? I/System.out: Draw MOVE[163] BELOW 154
05-11 11:37:41.312    2978-2978/? I/System.out: Draw MOVE[165] BELOW 163
05-11 11:37:41.312    2978-2978/? I/System.out: Draw REPEAT[167] BELOW 165
05-11 11:37:41.312    2978-2978/? I/System.out: Draw MOVE[169] BELOW 167
05-11 11:37:41.312    2978-2978/? I/System.out: Draw COLORSENSOR[171] RIGHT 167
05-11 11:37:41.322    2978-2978/? I/System.out: Draw POSITION[173] BELOW 169
05-11 11:37:41.322    2978-2978/? I/System.out: Añado conector bajo 173
```

Todos los cambios de añadir, eliminar o reordenar se realizan sobre el BlockProgram (ArrayList de BlockLink).
La interfaz de usuario sólo tiene que iterar el BlockProgram e ir colocando los bloques según indica cada BlockLink.
Si en un futuro se quisiera cambiar de interfaz de usuario sólo se tendría que tocar el método drawBlockProgram() de la clase ProgrammingCanvas, que es el encargado de dibujar y encajar los bloques en la interfaz de usuario Android.
Mi intención ha sido separar la gestión de la lista de bloques de la interfaz de usuario.

FASE 3. CONVERTIR UNA LISTA DE BLOQUES EN UN PROGRAMA Y EJECUTARLO EN EL ROBOT

Una vez que el usuario esté satifecho con el programa de bloques que ha creado pulsará el botón para que el Robot ejecute el programa (runStopButton).

En este momento hay que convertir una lista de bloques con posiciones relativas entre ellos en un programa.

Para ello paso la lista de bloques (blockProgram) a un Program. Un Program es una lista de comandos (CodeCommand).

Un CodeCommand tiene la forma: id, nombre del comando, parámetro
por ejemplo ID 10 Comando: MOVE Parámetro: Derecha

El Robot no entiede una lista de bloques, pero sí entiede los comandos así que pasamos el BlockProgram (lista de BlockLink) a un Program (lista de CodeCommand).

Esto lo he hecho así para separar el objeto programado (Robot) de los bloques, así cualquier cambio o extensión de los bloques no debería afectar al Robot y viceversa.

Si en el futuro vamos a programar un Avión en vez de un Robot, éste interpretaría los comandos de otra manera, pero no tendríamos que cambiar los bloques por ello.

He usado la misma filofofía que el patrón de programación Adapter.

Así que el usuario ha pulsado Ejecutar y el listener del botón convierte la lista de bloques en un programa y se lo pasa al Robot robot.runProgram(program)

Extracto de un log mostrando la lista de comandos que ha resultado de la lista de bloques:

```
05-11 11:39:51.018    2978-2978/? I/System.out: Program: creo lista comandos desde lista bloques
05-11 11:39:51.018    2978-2978/? I/System.out: Program: tengo 7 bloques
05-11 11:39:51.018    2978-2978/? I/System.out: inicio de paso de blockList a Program
05-11 11:39:51.018    2978-2978/? I/System.out: ------------
05-11 11:39:51.018    2978-2978/? I/System.out: ## Inicio program:
05-11 11:39:51.018    2978-2978/? I/System.out: 0 ID: 154 Comando: START Parámetro: null
05-11 11:39:51.018    2978-2978/? I/System.out: 1 ID: 163 Comando: MOVE Parámetro: Derecha
05-11 11:39:51.018    2978-2978/? I/System.out: 2 ID: 165 Comando: MOVE Parámetro: Abajo
05-11 11:39:51.018    2978-2978/? I/System.out: 3 ID: 167 Comando: REPEAT Parámetro: 4
05-11 11:39:51.018    2978-2978/? I/System.out: 4 ID: 171 Comando: COLORSENSOR Parámetro:
#ff8c5aeb
05-11 11:39:51.018    2978-2978/? I/System.out: 5 ID: 169 Comando: MOVE Parámetro: Derecha
05-11 11:39:51.018    2978-2978/? I/System.out: 6 ID: 173 Comando: POSITION Parámetro: 3,2
```

```
05-11 11:39:51.018    2978-2978/? I/System.out: ## Fin program.
```

Para que el Robot ejecute su programa he decido usar elementos Android: ObjectAnimator y AnimationSet (lista de ObjectAnimator).
http://developer.android.com/reference/android/animation/ObjectAnimator.html
http://developer.android.com/reference/android/animation/AnimatorSet.html

El programa en el Robot es una lista de comandos que se van ejecutando uno a uno. Ejecutas un comando y si el programa sigue se carga el siguiente comando.
Me he inspirado en como los computadores manejan un programa: cargando el comando siguiente en el program counter, así que lo he implementado usando en el Robot un contador de programa.
Si entras en un bucle repetir y llegas al final del bucle (sin darse las condiciones de salir del bucle) se apunta el contador de programa al comando de inicio del bucle y se prosigue la ejecución.

Extracto de un log de ejecución de la lista de comandos por parte del Robot:

```
05-11 11:39:51.018    2978-2978/? I/System.out: Cargo comando PC 1 (PC es programCounter)
05-11 11:39:51.018    2978-2978/? I/System.out: ID: 163 Comando: MOVE Parámetro: Derecha
05-11 11:39:51.018    2978-2978/? I/System.out: Robot paso a la derecha a 140.0
05-11 11:39:51.018    2978-2978/? I/System.out: X: 140 Y:0
05-11 11:39:51.018    2978-2978/? I/System.out: X: 1 Y:0
05-11 11:39:52.039    2978-2978/? I/System.out: Comando ejecutado: avanzo programa
05-11 11:39:52.039    2978-2978/? I/System.out: Fin de paso -> actualizo sensor color
05-11 11:39:52.039    2978-2978/? I/System.out: X: 140 Y:0
05-11 11:39:52.039    2978-2978/? I/System.out: Advance program
05-11 11:39:52.039    2978-2978/? I/System.out: PC en 1 de 6
05-11 11:39:52.039    2978-2978/? I/System.out: Cargo siguiente PC 2
05-11 11:39:52.039    2978-2978/? I/System.out: Cargo comando PC 2
05-11 11:39:52.039    2978-2978/? I/System.out: ID: 165 Comando: MOVE Parámetro: Abajo
05-11 11:39:52.039    2978-2978/? I/System.out: Robot paso abajo a 140.0
05-11 11:39:52.039    2978-2978/? I/System.out: X: 140 Y:140
05-11 11:39:52.039    2978-2978/? I/System.out: X: 1 Y:1
05-11 11:39:53.060    2978-2978/? I/System.out: Comando ejecutado: avanzo programa
05-11 11:39:53.060    2978-2978/? I/System.out: Fin de paso -> actualizo sensor color
05-11 11:39:53.060    2978-2978/? I/System.out: X: 140 Y:140
05-11 11:39:53.060    2978-2978/? I/System.out: Advance program
05-11 11:39:53.060    2978-2978/? I/System.out: PC en 2 de 6
05-11 11:39:53.060    2978-2978/? I/System.out: Cargo siguiente PC 3
05-11 11:39:53.060    2978-2978/? I/System.out: Cargo comando PC 3
05-11 11:39:53.060    2978-2978/? I/System.out: ID: 167 Comando: REPEAT Parámetro: 4
05-11 11:39:53.070    2978-2978/? I/System.out: Advance program
05-11 11:39:53.070    2978-2978/? I/System.out: advance program in Repeat Mode
05-11 11:39:53.070    2978-2978/? I/System.out: repeatEntry 4
05-11 11:39:53.070    2978-2978/? I/System.out: repeatEscape 4
05-11 11:39:53.070    2978-2978/? I/System.out: Cargo comando PC 4
05-11 11:39:53.070    2978-2978/? I/System.out: ID: 171 Comando: COLORSENSOR Parámetro: #ff8c5aeb
05-11 11:39:53.070    2978-2978/? I/System.out: color tras REPEAT
05-11 11:39:53.070    2978-2978/? I/System.out: Robot (x,y)=(1,1)
05-11 11:39:53.070    2978-2978/? I/System.out: Inicio bucle en busca de color. PC a 4
05-11 11:39:53.070    2978-2978/? I/System.out: Advance program
05-11 11:39:53.070    2978-2978/? I/System.out: advance program in Repeat Mode
05-11 11:39:53.070    2978-2978/? I/System.out: repeatEntry 5
05-11 11:39:53.070    2978-2978/? I/System.out: repeatEscape 5
05-11 11:39:53.070    2978-2978/? I/System.out: Cargo comando PC 5
05-11 11:39:53.070    2978-2978/? I/System.out: ID: 169 Comando: MOVE Parámetro: Derecha
05-11 11:39:53.070    2978-2978/? I/System.out: Robot paso a la derecha a 280.0
05-11 11:39:53.070    2978-2978/? I/System.out: X: 280 Y:140
05-11 11:39:53.070    2978-2978/? I/System.out: X: 2 Y:1
05-11 11:39:54.091    2978-2978/? I/System.out: Comando ejecutado: avanzo programa
```

```
05-11 11:39:54.091    2978-2978/? I/System.out: Fin de paso -> actualizo sensor color
05-11 11:39:54.091    2978-2978/? I/System.out: X: 280 Y:140
05-11 11:39:54.101    2978-2978/? I/System.out: Advance program
05-11 11:39:54.101    2978-2978/? I/System.out: advance program in Repeat Mode
05-11 11:39:54.101    2978-2978/? I/System.out: repeatEntry 5
05-11 11:39:54.101    2978-2978/? I/System.out: repeatEscape 5
05-11 11:39:54.101    2978-2978/? I/System.out: Cargo comando PC 5
05-11 11:39:54.101    2978-2978/? I/System.out: ID: 169 Comando: MOVE Parámetro: Derecha
05-11 11:39:54.101    2978-2978/? I/System.out: Robot paso a la derecha a 420.0
05-11 11:39:54.101    2978-2978/? I/System.out: X: 420 Y:140
05-11 11:39:54.101    2978-2978/? I/System.out: X: 3 Y:1
05-11 11:39:55.142    2978-2978/? I/System.out: Comando ejecutado: avanzo programa
05-11 11:39:55.142    2978-2978/? I/System.out: Fin de paso -> actualizo sensor color
05-11 11:39:55.142    2978-2978/? I/System.out: X: 420 Y:140
05-11 11:39:55.142    2978-2978/? I/System.out: Advance program
05-11 11:39:55.152    2978-2978/? I/System.out: PC en 5 de 6
05-11 11:39:55.152    2978-2978/? I/System.out: Cargo siguiente PC 6
05-11 11:39:55.152    2978-2978/? I/System.out: Cargo comando PC 6
05-11 11:39:55.152    2978-2978/? I/System.out: ID: 173 Comando: POSITION Parámetro: 3,2
05-11 11:39:55.162    2978-2978/? I/System.out: X: 280 Y:140
05-11 11:39:55.162    2978-2978/? I/System.out: X: 2 Y:1
05-11 11:39:55.663    2978-2978/? I/System.out: Comando ejecutado: avanzo programa
05-11 11:39:55.663    2978-2978/? I/System.out: Fin de paso -> actualizo sensor color
05-11 11:39:55.663    2978-2978/? I/System.out: X: 280 Y:140
05-11 11:39:55.673    2978-2978/? I/System.out: Advance program
05-11 11:39:55.673    2978-2978/? I/System.out: PC en 6 de 6
05-11 11:39:55.673    2978-2978/? I/System.out: X: 280 Y:140
```

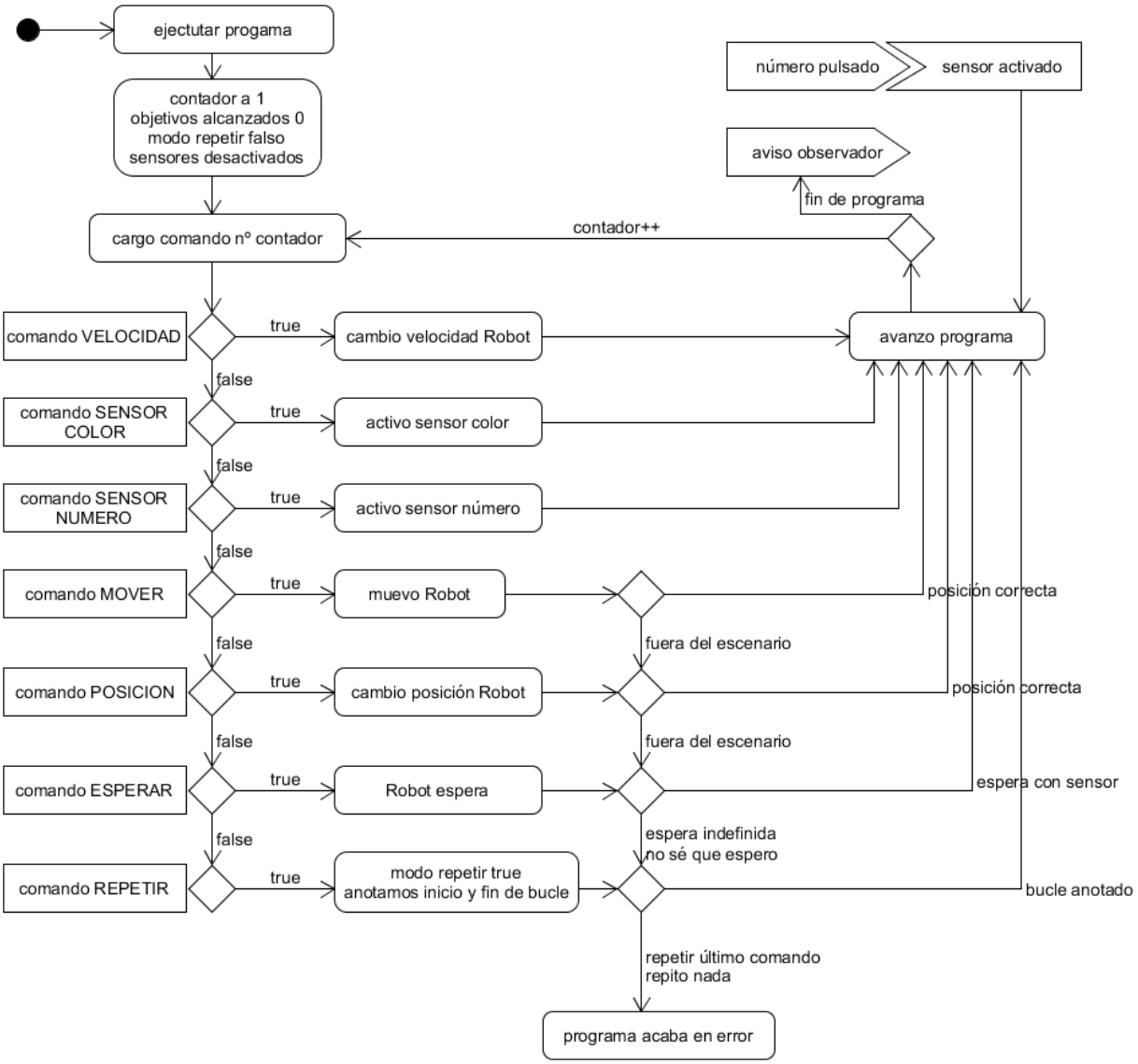


Imagen: flujo de ejecución de un programa en el ROBOT

Cuando el Robot termina de ejecutar su programa comprueba si ha pasado por los objetivos parciales (si los hubiera) y si ha acabado el programa sobre el cuadrado objetivo (x,y). Si es así avisa a su “observador”, la pantalla principal .

Terminar sobre el objetivo final, incluso pasando por los objetivos parciales (misión cuatro) no es suficiente. Hay que evaluar el programa de bloques, por ejemplo, si se ha incrementado la velociadad (misión dos). Para ello cada misión tiene un Evaluador, que evalúa lo necesario para cada misión y da un veredicto.

La idea de los Evaluadores está inspirada en los módulos pedagógicos de los Sistemas Interactivos de Enseñanza y Aprendizaje. Observan el comportamiento del alumno y deciden que mensaje o qué contenidos desbloquear. En este caso se analiza el programa de bloques que has realizado, qué bloques has usado y te ayuda a mejorar o te da una misión más compleja.

4. COMUNICACIÓN CON EL USUARIO

Lo he dejado para el final, pero creo que es una parte muy importante, igual o más que el funcionamiento. En esta fase he analizado y creado un flujo de información entre el programa y el usuario.

Hay dos principales formas de comunicarse con el usuario en Android:

- **Dialog**. Ventana emergente. El usuario debe interactuar para que se vaya (pulsar OK, cancelar, etc)
- **Toast**. Texto emergente. Se desvanece sólo tras unos segundos.

Al entrar en el programa se le informa al usuario de la misión y cómo cumplirla con un **Dialog**. La primera misión es muy fácil y sólo se utiliza un sólo bloque: MOVER

Imágen: mostrando Dialog misión uno

Al descartar la ventana he añadido un **Toast** sobre cómo empezar a arrastrar el primer bloque, sólo en esta primera misión. Creo que una vez que se arrastra un bloque el resto es bastante intuitivo. El usuario puede recuperar el Dialog de la misión desde el menú de la barra superior siempre que lo necesite.

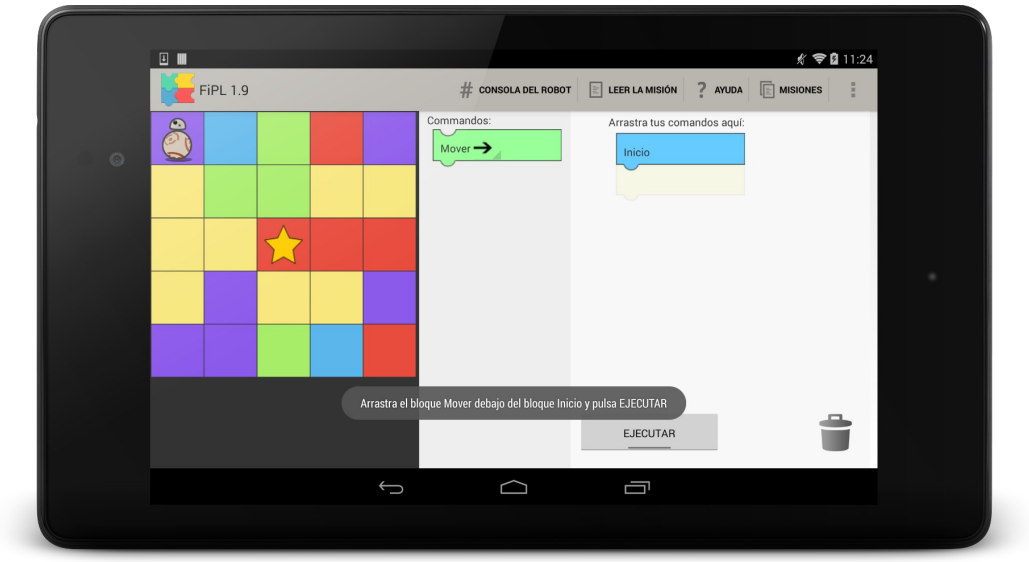


Imagen: mostrando Toast misión uno

Los bloques tienen unos “conectores” como pistas de donde puedes conectar otro bloque. Si un bloque entra en su área se iluminan para dar a entender que aquí se puede soltar. El enlazado de los bloques hace un sonido. Si intentas conectar un bloque donde no encaja se produce un sonido de error. Esto es para “guiar” al usuario en el uso de los bloques.

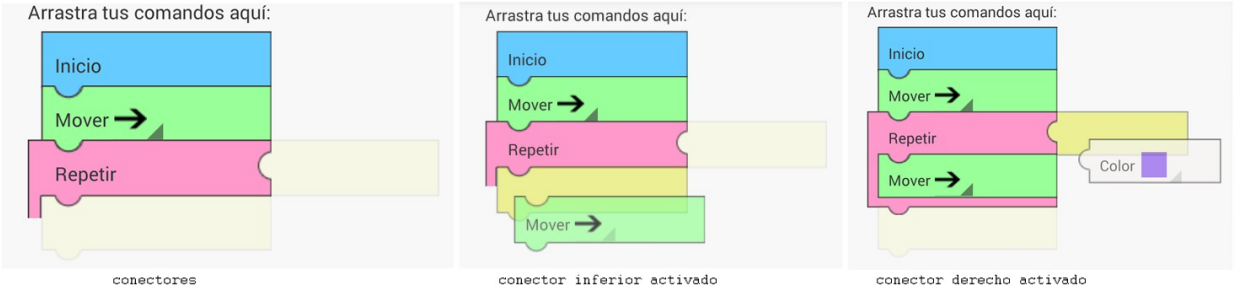


Imagen: secuencia de conectores guiando al usuario para soltar un bloque

También he añadido un log al Robot. Cuando se ejecuta el programa la columna de los bloques se gira y el Robot va informando paso a paso de la ejecución del programa.
Pienso que así es una experiencia de programación más "real". La información en los logs son muy importantes y así los niños pueden hacerse a la idea de que un sistema te puede proporcionar información sobre qué está pasando durante la ejecución de un programa.

Imagen: mostrando log del Robot

He añadido sonidos a unos eventos concretos que creo que ayudan a la comunicación entre programa y usuario: al conectar un bloque, borrarlo, misión OK, programa acaba en error y misión no cumplida.

El resultado del programa se evalúa en múltiples etapas:

- a tiempo real mediante los logs del Robot, él te va informando si ha llegado a un objetivo, ya sea parcial o total, si está esperando un número o si ha detectado un color.
- cuando el programa termina, si el Robot ha cumplido todos los objetivos de posición, es decir, ha pasado por todos los cuadrados para cumplir la misión, entra en juego el evaluador. El evaluador de cada misión te va a decir si la misión está PERFECTA, OK o MAL. Esta información es con Dialog, así que el usuario tendrá tiempo de leerlo hasta que decida descartarlo. Todos los mensajes indican que se ha hecho bien o si está mal qué se puede hacer para mejorarlo.

Imagen: misión no cumplida explicando qué falta

Imagen: Misión completada con éxito

Imagen: misión cumplida pero se puede hacer de otra manera

En la pantalla de las misiones, accesible desde el menú de la barra superior, se enumeran todas las misiones que el usuario puede repetir cuando quiera.

Imagen: pantalla de misiones

En la pantalla de ayuda, también accesible desde el menú de la barra superior, se explica como funciona el programa y cada bloque en particular.

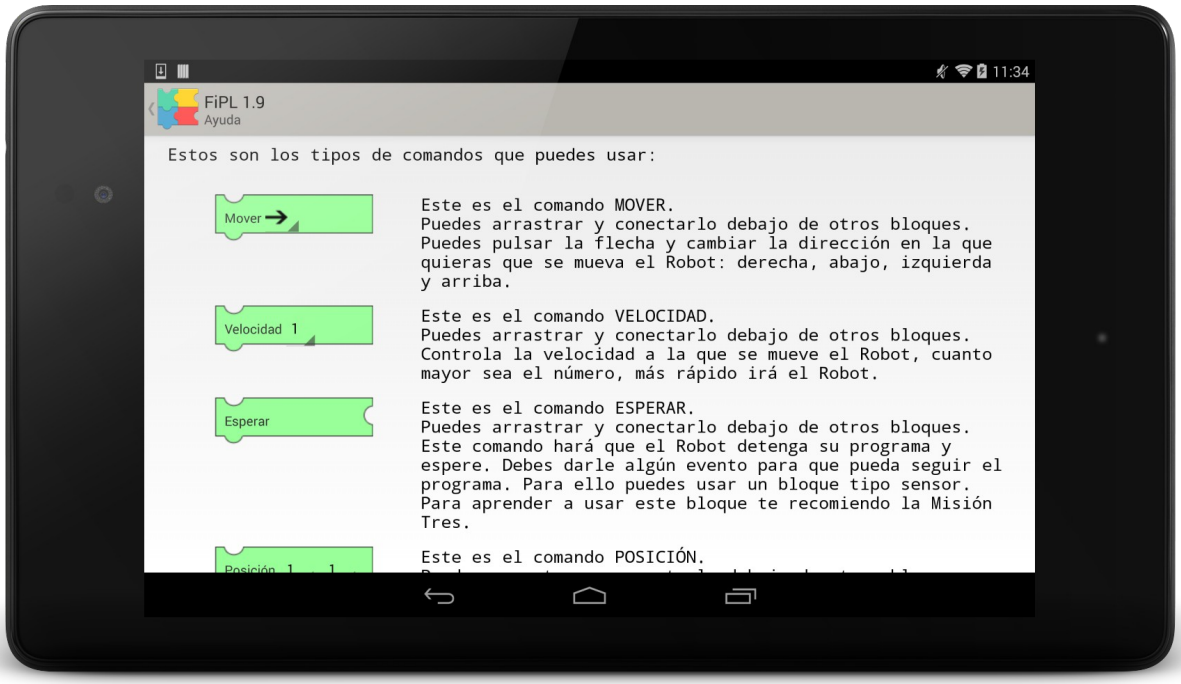


Imagen: pantalla de misiones