UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universidad politécnica de Valencia

Sistema Android generador de pruebas de esfuerzo basado en el test de Conconi

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autor: Enrique Bodí Pastor

Director: Juan Carlos Ruiz

Fecha: 24/09/2012

Índice

1 Introducción

1.1 Motivación

1.2 Objetivos

1.3 Aplicaciones similares

2 Elección del Sistema Operativo

2.1 Android

2.2 iOS

2.3 Symbian OS

2.4 Windows Phone

2.5 Blackberry

2.6 Decisión

3 Análisis y diseño

3.1 Información de las zonas de entrenamiento cardiovascular

3.2 Funcionamiento del sistema cardiovascular bajo diferentes condiciones

3.3 Especificación de requisitos

3.3.1 Especificación funcional

3.3.1.1 Diagrama casos de uso

3.3.1.2 Diagrama de flujo

3.3.1.3 Diagrama de clases

3.3.1.3 Gestión y verificación de datos

3.3.1.4 Algoritmo de procesamiento de datos

3.3.2 Especificación no funcional

3.4 Diseño de la App

4 Implementación

5 Conclusión

6 Bibliografía

1- Introducción:

El proyecto final de carrera a realizar va a ser una aplicación para Android, en la cual se simula el test de Conconi, que ofrece al usuario información precisa sobre las zonas cardíacas de entrenamiento y sus límites.

1.1- Motivación

La idea de realizar este proyecto surgió al ver la metodología de entrenamiento utilizada en un gimnasio de artes marciales.

Uno de los entrenamientos de resistencia consistía en la simulación de un combate a intensidad máxima contra un saco de boxeo por varios ciclos, de aproximadamente 2 minutos y 1 minuto de descanso.

Dependiendo el día, se aumentaba la duración o se reducía el tiempo de descanso basándose en el rendimiento de cada uno.

De esto surgió la idea de desarrollar una aplicación Android para simular este tipo de entrenamiento.

Ésta consistiría en un simple cronómetro que se comunicaría con el usuario por una serie de mensajes sonoros para indicarle el inicio del ciclo de trabajo, del tiempo de descanso y de los ciclos restantes.

Sería una buena idea, si no tuviéramos en cuenta que estos ciclos deben realizarse a intensidad máxima y la motivación al entrenar solo, no es la misma que con compañeros, por lo tanto había que idear algo para mejorar esta motivación.

Un buen indicador fisiológico de la intensidad a la que se está realizando un ejercicio es la frecuencia cardíaca.

Si a la aplicación anterior la conectamos con un pulsometro podemos monitorizar la intensidad y dar instrucciones al usuario si se está trabajando a una intensidad muy baja o por el contrario, demasiado alta.

Pero ahora tenemos otro problema ¿cómo podemos saber cuál es la intensidad máxima de una persona para no exigirle mas de lo que su cuerpo puede dar?

Existen varias formulas para calcular las la diferentes zonas de trabajo del corazón, pero estas se basan en un dato dudoso o no determinante como es la edad.

Por ello, no podemos confiar en estos datos, ya que pueden no ser ciertos y le estaríamos exigiendo al usuario más intensidad de la que puede dar y éste podría sufrir un infarto u provocar lesiones de considerable gravedad.

Por esta y muchas razones debemos ceñirnos al rigor más científico para desarrollar una aplicación de este tipo.

Existen una serie de test de esfuerzo, con los que podríamos calcular las zonas de trabajo con total exactitud para cada individuo en concreto.

Por una parte tenemos el test de consumo máximo de oxígeno (VO2max), este test es el más confiable, pues detecta en que momento el cuerpo no puede consumir más oxígeno y pasa de la

zona aeróbica a la zona anaeróbica. La desventaja es que necesitaríamos un equipo especial para medir este consumo.

Por otra parte, tenemos el test de Conconi. Este test calcula el punto de inflexión entre la zona aeróbica y la zona anaeróbica mediante una prueba de esfuerzo en la que se mide la ruptura de una correlación lineal entre pulsaciones y velocidad.

Finalmente nos decidimos por el segundo test, pues el pulsometro necesario para realizar esta prueba también lo utilizaremos para el entrenamiento de resistencia.

Una vez llegados a este punto nos damos cuenta de que, por un lado, tenemos un entrenador personal enfocado a la resistencia en los deportes de contacto y por otro lado un test de esfuerzo.

El conjunto de las dos aplicaciones excede las competencias del PFC, por tanto tuvimos que elegir entre una de estas dos.

Dado su rigor científico, ampliamente documentado de el test de Conconi, sumado a la necesidad de utilizar conocimientos aprendidos durante la carrera como programación, matemáticas, estadística, etc (aparte de otros conocimientos aprendidos por cuenta propia) para realizar una correcta interpretación de los datos, nos decantamos por la opción de la prueba de esfuerzo para proponerla como Proyecto final de carrera.

1.2- Objetivos

El objetivo de esta aplicación es permitir a deportistas de nivel medio-alto realizar este tipo de pruebas para conocer de forma muy exacta las diferentes zonas de trabajo de su sistema cardiovascular y que con ello, puedan planificar entrenamientos mas acordes con sus objetivos.

Hay que destacar que se ha ampliado el rango de usuarios, pues, no solo está limitada a los que practican deportes de contacto, sino que puede ser de utilidad para la gran mayoría de deportistas.

1.3- Aplicaciones similares

En este apartado realizaremos un estudio de las aplicaciones similares existentes para el *Sistema operativo Android*.

En el cual tendremos en cuenta, las aplicaciones relacionadas con el deporte, que más se parezcan a la que queremos desarrollar, para encontrar similitudes, metodologías y aspectos que pueda suplir nuestra aplicación.

Actualmente, no hay ninguna aplicación en el Android Market, que simule una prueba de esfuerzo.

Sin embargo, podríamos utilizar algunas aplicaciones diseñadas para el running para realizar por nosotros mismos un test de esfuerzo como el *test de Cooper*, pero para esto el usuario debería tener previamente conocimientos sobre como realizarlo y configurar el entrenamiento para este fin.

De todas formas, hemos analizado algunas de estas aplicaciones como *Google My Tracks*, *Endomondo*, *RunKeeper* y *Nike+ Running* y hemos extraído varios puntos a tener en cuenta.

Todas estas aplicaciones tienen un denominador común. Por ejemplo todas utilizan localización por GPS que combinado con *GoogleMaps* muestran un reporte de el recorrido.

Por otra parte, como es normal, se muestra información sobre tiempo total, distancia recorrida, velocidad media, etc.

Algunas de ellas permiten el uso de un pulsómetro, así como compartir el entrenamiento en alguna red social.

Ninguna de ellas cuenta con una prueba de esfuerzo. Es en este escenario donde nuestra aplicación tendrá cabida.

Sin embargo intentaremos utilizar algunos aspectos de las aplicaciones ya existentes como mostrar información de interés al finalizar la prueba y poder compartir dichos resultados en las redes sociales.

2- Elección del Sistema Operativo

En este apartado procederemos a realizar un estudio, así como presentar la historia de los principales Sistemas Operativos para dispositivos móviles

2.1- Android

Este Sistema Operativo de código abierto basado en *Linux* y orientado a dispositivos móviles fue desarrollando en un principio por *Android Inc.*

Esta empresa fue comprada por la multinacional Google en 2005, aunque la existencia del sistema operativo no se hace publica hasta el año 2007, momento en que se anuncia la creación de la *Open Handset Aliance*. Esta organización es un consorcio de compañías relacionadas con la informática y las telecomunicaciones, cuyo principal objetivo es la promoción del sistema operativo *Android*.

Actualmente, la cuota de mercado de los dispositivos con *Android* es del 60%, muy por delante de sus competidores.

Google play (anteriormente llamado *Android Market*) es la tienda de software en linea desarrollada para *Android*. Esta tienda cuenta con mas de 600.000 aplicaciones, la mayoría de ellas gratuitas.

Uno de los motivos por el que existe este gran número de aplicaciones es por las facilidades ofrecidas a la plataforma de programación, con un kit de desarrollo de software (*Android SDK*), basado en *Java*, aunque también se puede programar en C y C++.

Otro podrida ser el coste de la membresía como desarrollador, de unos 25 dolares para toda la vida. Precio que, comparado con otras plataformas es irrisorio.

Por último, que casi el 60% de los terminales móviles esté basado en *Android* es un buen incentivo, pues de esta forma tienen la posibilidad de llegar a mas usuarios finales.

2.2- iOS (Apple)

Este Sistema Operativo fue desarrollado a partir del *Mac OS X* de *Apple*, para los dispositivos móviles. En un principio, *iOS* iba destinado a la gama iPhone y se llamaba *iPhone OS*.

Pero en 2010, tras el anuncio del iPad, Steve Jobs, uno de los CEOs de Apple, comunicó que el sistema operativo pasaría a llamarse *iOS* (pues a partir de ese momento iba a ser utilizado también por los iPad).

Este sistema operativo es sin duda unos de los principales atractivos de la marca, está instalado en todos los dispositivos móviles de esta y su uso esta restringido a terceros.

La cuota de mercado actual es del 23%, lo cual es razonable teniendo en cuenta que dicho sistema operativo solo se utiliza en dispositivos *Apple*.

En este caso, la tienda de software disponible para este SO se llama *App Store*. El número de aplicaciones disponibles no es tan elevado como el anterior Sistema Operativo por diferentes razones.

Una de las principales es la alta cuota de membresia como desarrollador de Apple, en la cual, tan solo un año cuesta unos 99 dolares. Además, si no se pagáramos esta membresia, no podríamos ejecutar aplicaciones en dispositivos *Apple*, sino que deberíamos utilizar el "*iPhone Simulator*"

El kit de desarrollo de software of SKD fue liberado en 2008. Este tipo de aplicaciones deben ser programadas en el lenguaje de programación *Objective-C*.

2.3- Symbian OS

Symbian fue desarrollado por un conjunto de empresas de telefonía móvil, que se unieron para crear un sistema operativo capar de competir con *Palm OS* o *Windows mobile*.

Sin embargo, el desarrollo de *Android* e *iOS* ganaron la cuota de mercado a la que iba destinado *Symbian* (sobretodo *Android*) que se quedó con un 6,8%

Se pueden desarrollar aplicaciones para *Symbian* en *Java, C++ Visual Basic, Python, Perl, Flash Lite* (entre otros), descargando su correspondiente SDK.

2.4- Windows phone

El sistema operativo *Windows Phone* (basado en *Windows CE*) fue desarrollado por *Microsoft* como sucesor de la plataforma *Windows Mobile*.

Aún no existen cálculos fiables de la cuota de mercado de este sistema operativo.

Respecto al desarrollo de aplicaciones, se pueden programar siguiendo dos tipos diferentes de implementación.

La primera, basada en *Silverlight*, permite realizar aplicaciones que contengan transiciones y efectos visuales

La segunda, utilizando XNA Framework, que incluye bibliotecas de clases, específicas para el desarrollo de juegos.

Marketplace es la tienda de contenidos de *Windows Phone*, donde está disponible a la venta musica, vídeos y por supuesto aplicaciones.

2.5- Blackberry OS

Este Sistema Operativo fue desarrollado por *Research In Motion (RIM)* para sus dispositivos *BlackBerry*.

BlackBerry OS tiene soporta diferentes métodos de entrada utilizados por *RIM* para sus dispositivos móviles.

Una de sus principales características es la posibilidad de sincronizar un dispositivo con correo electrónico, tareas, notas y contactos de *Microsoft Exchange server.*

Al igual que en los demás Sistemas Operativos, los desarrolladores independientes también pueden crear programas para *BlackBerry* pero en el caso de querer tener acceso a ciertas funcionalidades restringidas necesitan ser firmados digitalmente para poder ser asociados a una cuenta de desarrollador de *RIM*.

2.6- Decisión

De todos los Sistemas operativos móviles anteriormente mencionados, podemos destacar principalmente *Android* e *iOS* como posibles elegidos por diversas razones.

Por una parte tendremos en cuenta la cuota de mercado. Sumando estas dos plataformas tenemos un 83% de los terminales existentes. Aunque el proyecto no esté dedicado a un uso masivo de usuarios, tener una cuota de mercado amplia facilitará al usuario interesado utilizar dicha aplicación.

Por otra parte, las facilidades ofrecidas al desarrollador por parte de la plataforma son muy importantes. En este caso, a nivel de lenguaje de programación las dos plataformas cuentan un lenguaje de programación de uso muy extendido. En el caso de *Android*, el lenguaje utilizado es *Java*, mientras que en *iOS* es una variante del lenguaje *C*, siendo estos dos ampliamente utilizados en el mundo de la informática.

Además, los dos cuentan con una tienda de software, con la cual lo usuarios pueden descargar e instalar directamente las aplicaciones.

Una vez hemos decidido para cuales de estos Sistemas Operativos podríamos desarrollar la aplicación, hay que elegir una de las dos.

Si tenemos en cuenta la cuota de mercado, *Android* es el Sistema Operativo idóneo para esta aplicación. Tiene un 60% de cuota de mercado (y esta va al alza) mientras que *iOS* tiene un 23%.

Por otra parte, en el caso de *iOS* podemos tener por supuesto que el rendimiento del terminal va a ser el correcto y no vamos a tener ningún problema a la hora de ejecutar tareas complicadas. No podemos decir lo mismo con *Android*, que aunque existen unos requisitos básicos, estos son mínimos.

Si tenemos en cuenta el lenguaje de programación, tanto *Java* como *C* son utilizados por un gran número de desarrolladores, sin embargo, las facilidades para desarrollar y publicar las aplicaciones Android son mucho mayores que en *iOS*. Por ejemplo, en el caso de *iOS*, para poder ejecutar la aplicación en el terminal, antes deberíamos para la cuota de membresia de 99 dolares (por año) o ejecutarla en una máquina virtual. Como nuestro objetivo es desarrollar una aplicación dinámica aplicada al deporte, es evidente que no la podríamos testear con comodidad.

Esta es una razón por la cual desarrollar en *Android*, se pueden testear las aplicaciones libremente y si el funcionamiento es correcto y deseamos agregarla la tienda de software de Android tan solo tenemos que darnos de alta en *GoogleCheckout*, pagando una cuota de 25 dolares. Hay que tener en cuenta que al contrario de *AppMarket,* en *GoogleCheckout* la membresia es de por vida.

Otro punto a favor es que el lenguaje de programación necesario para programar en Android es *Java*, un lenguaje que hemos utilizado mucho en varias asignaturas de la carrera y el cual dominamos con bastante soltura.

Para finalizar, la política de licencias de *Android* se basa en una *libre* y de *código abierto*, a diferencia de *iOS* que mantiene una política de licencias propietaria.

En nuestro caso, apoyamos las políticas de código libre, por tanto esto es un punto a favor del *Sistema Operativo Android*.

En definitiva, por estas y varias razones más, decidimos que el Sistema Operativo sobre el cual vamos a desarrollar la aplicación será *Android*.

3- Análisis y diseño

Vamos a empezar este proyecto con un estudio previo a la implementación, en el que definiremos claramente como funcionará nuestra aplicación, sin entrar en detalles de implementación.

Por tanto, en este apartado mostramos los pasos previos realizados a la programación de la App.

3.1 Información de las zonas de funcionamiento cardiovascular

Podemos dividir el funcionamiento del sistema cardiovascular en dos zonas, dependiendo de como se produce el consumo de oxígeno y la forma en la que éste se utiliza, una vez en el músculo para generar energía.

La primera de estas sería la *zona aeróbica*.

Cuando esta zona de trabajo está en ejercicio se produce en equilibrio de oxígeno, esto quiere decir que disponemos de mas oxígeno del que necesitamos.

Esto es fundamental para la realización óptima de las reacciones químicas necesarias en las células para producir energía.

Por otra parte tenemos la *zona anaeróbica*.

En esta zona el ejercicio se realiza en un entorno de déficit de oxígeno, es decir que consumimos más oxígeno del que inhalamos.

En este caso las producción de energía en las células no será tan óptima y se realizará con déficit de oxígeno.
El grado de déficit dependerá de la intensidad a la que se esté realizando un ejercicio y dependiendo de éste, intervendrán el sistema aeróbico y el anaeróbico en mayor o menor medida.
Hay dos tipos de sistemas anaeróbicos para generar energía.
El primero, llamado *ATP-PC*, usa fosfato de creatina en los primeros 10 segundos.
El segundo utiliza la glucosa en ausencia de oxígeno como combustible, pero esto genera algunas sustancias de desecho como el ácido láctico, que perjudican la función muscular.

En resumen, tenemos dos zonas que se diferencian por la disponibilidad de oxígeno para realizar las reacciones químicas que producen energía.
En la zona aeróbica la disponibilidad de oxígeno es mayor o igual que la necesidad de este.

Disponibilidad de oxígeno > Demanda de oxígeno

En la zona anaeróbica, la disponibilidad de oxígeno es menor que la demanda.

Disponibilidad de oxígeno < Demanda de oxígeno

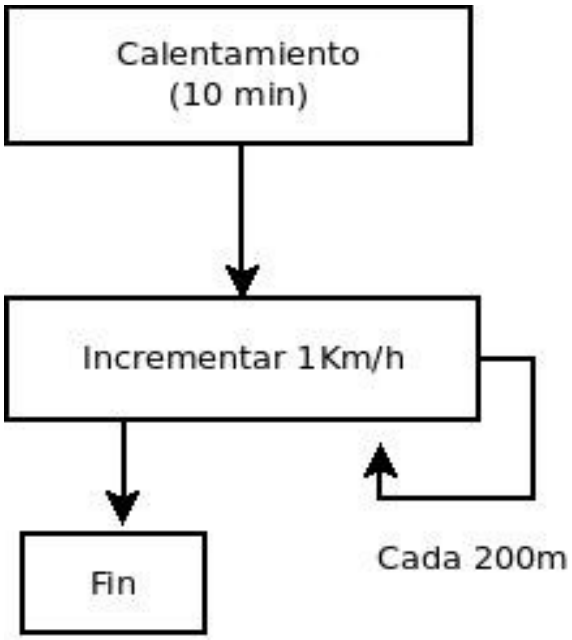
3.2 Funcionamiento del sistema cardiovascular bajo diferentes condiciones

Basándonos en la investigaciones de Francesco Conconi y plasmada en el documento "*Determination of the anaerobic threshold by noninvasive field test in runners, 1981*", podemos definir el comportamiento del sistema cardiovascular bajo diferentes condiciones.

En este estudio, Conconi observó que había una concordancia lineal entre velocidad y frecuencia cardiaca en la zona aeróbica y que esta relación se rompía al pasar a la zona anaeróbica.

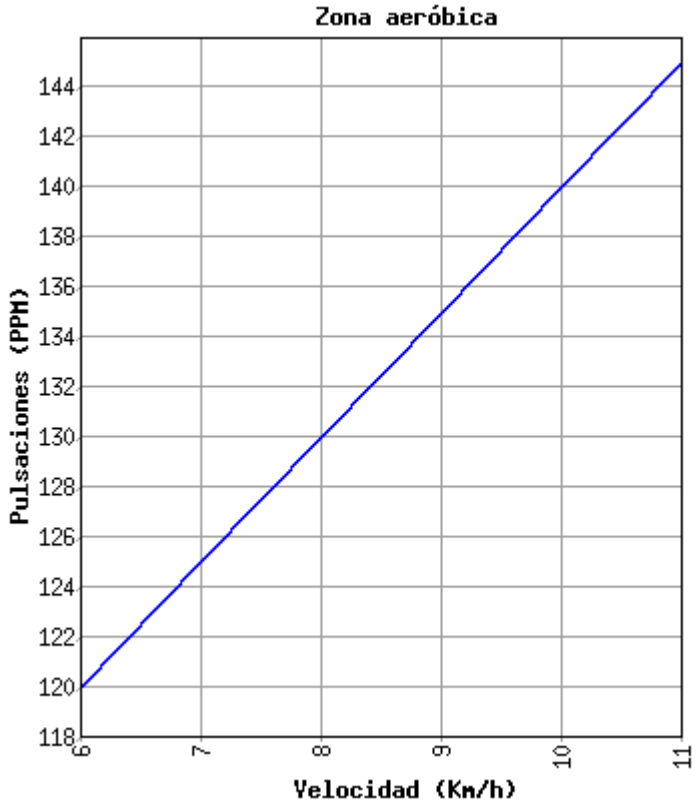
Basándose en estos principios, Conconi ideó una prueba de esfuerzo en la que utilizando una cinta de correr, para definir una velocidad exacta, podríamos modelar la actividad del sistema cardiovascular y extraer de ella una gráfica con la que poder analizar dicha actividad.

Esta prueba consiste en, situados en una cinta de correr y tras 10 minutos de calentamiento a aproximadamente 120 PPM, incrementar la velocidad 1Km/h cada 200 metros.

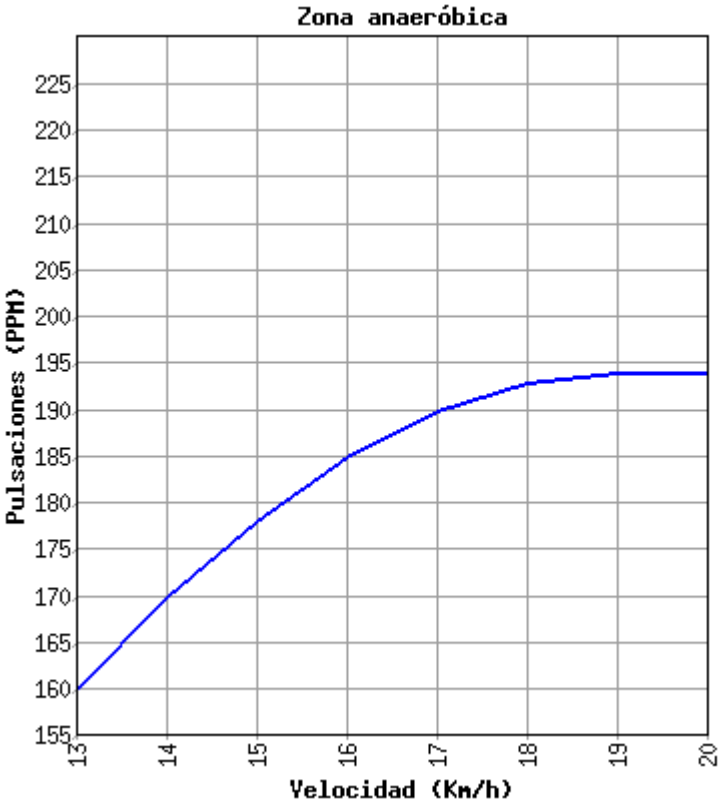


Como dijimos anteriormente, este tipo de prueba va a provocar comportamientos diferente dependiendo en que zona estemos trabajando.

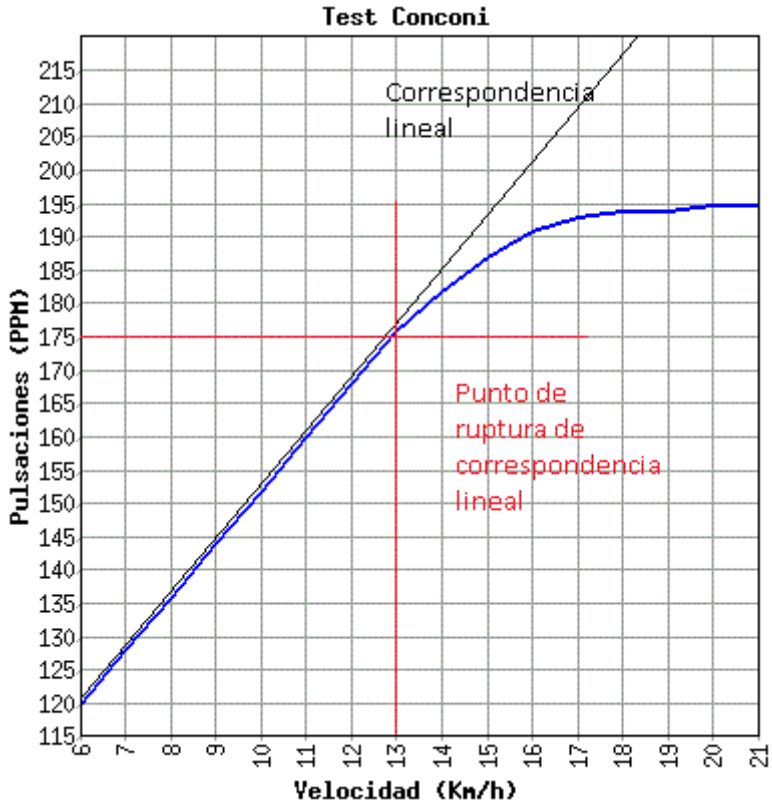
El la zona aeróbica tendremos una correlación lineal:



En cambio, cuando pasamos a la zona anaeróbica la correlación pasa de ser lineal a ser logarítmica :



Es en esta transición donde podemos calcular el límite de la zona aeróbica.



Como se aprecia en la tabla anterior, el punto de ruptura es fácilmente localizable si recreamos una recta de regresión a partir de los primeros valores y los comparamos con los los nuevos.

Si la dispersión de un dato es mayor de la que esperamos, lo marcamos como posible punto de inflexión y validamos este si en el siguiente punto la dispersión aumenta también.

3.3 Especificación de requisitos

En este apartado realizaremos un estudio de los requerimientos de la aplicación y la implementación de los mismos

3.3.1- Especificaciones funcionales

Para comenzar se indican varios escenarios que representen el uso que puedan darle los usuarios a la aplicación.

Por un lado tenemos a Juan. El es un corredor experimentado, participa en varias competiciones, usa un pulsometro para optimizar sus entrenamientos y ahora mismo está en fase de pretemporada.

Buscando por internet alguna formula confiable para calcular sus zonas cardiacas descubre la aplicación "*ConconiTest*" y se interesa por ella.

Juan ya había pensado en acudir a un centro especializado para realizar una prueba de esfuerzo, pero no llegó a decidirse por el alto precio de estas y los extraños aparatos a los que se tendría que conectar, pero en el caso de "*ConconiTest*" el único dispositivo que debe utilizar es un pulsometro y ya está acostumbrado a su uso.

Por todas estas razones Juan decide utilizar esta aplicación y se siente muy satisfecho con ésta.

Juan recomienda la aplicación a muchos de sus compañeros de entrenamiento.

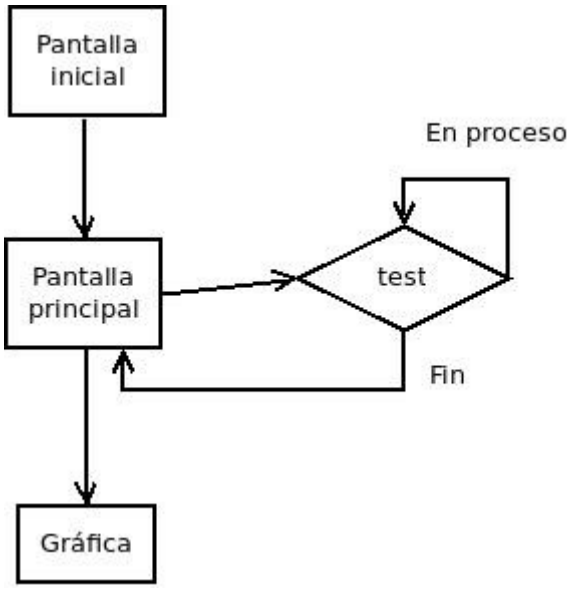
Por otro lado tenemos a Carlos.

Carlos está desarrollando el hábito de salir a correr 3 veces por semana, como las aplicaciones deportivas están de moda, Carlos empieza a utilizar aplicaciones como *Endomondo, RunKepper, etc.*

Buscando una aplicación nueva que probar encuentra "*ConconiTest*". No es lo que quería, pero decide probarla.

Tras la prueba se siente satisfecho con la aplicación y utiliza los valores proporcionados por esta en sus posteriores entrenamientos.

A continuación mostramos y explicamos el diagrama de flujo del usuario:



En la pantalla inicial se muestran las instrucciones y varios consejos para sacar el máximo rendimiento a la aplicación, además se invita al usuario a continuar.

Además se pedirá permiso para conectar el *Bluetooth* si este no está conectado todavía.

En la pantalla principal, el usuario dispone toda la información proporcionada por el pulsometro, como distancia recorrida, frecuencia cardiaca, velocidad y además un campo de texto donde se le informa de el desarrollo del test y se le da información básica.

Esta pantalla contiene 3 botones:

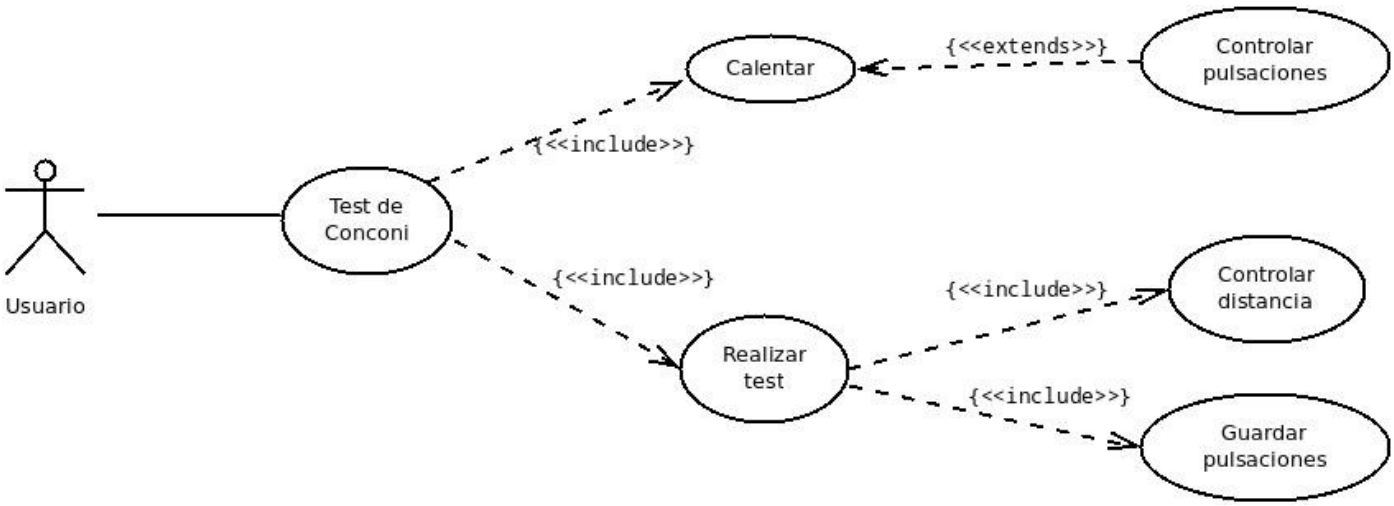
El primer botón, con título "*Empezar*", inicia la prueba de esfuerzo

El segundo botón con título ("*Cancelar/Fin*"), dependiendo si la prueba de esfuerzo está iniciada, detendrá la prueba. En el caso de que la prueba no haya sido iniciada, no hará nada.

El tercer botón estará deshabilitado en un primer momento y se habilitara una vez la prueba de esfuerzo haya sido finalizada. Tiene como título "*Dibujar*" y nos dirigirá hasta la pantalla donde se representan los datos y se dibujan las gráficas después del test.

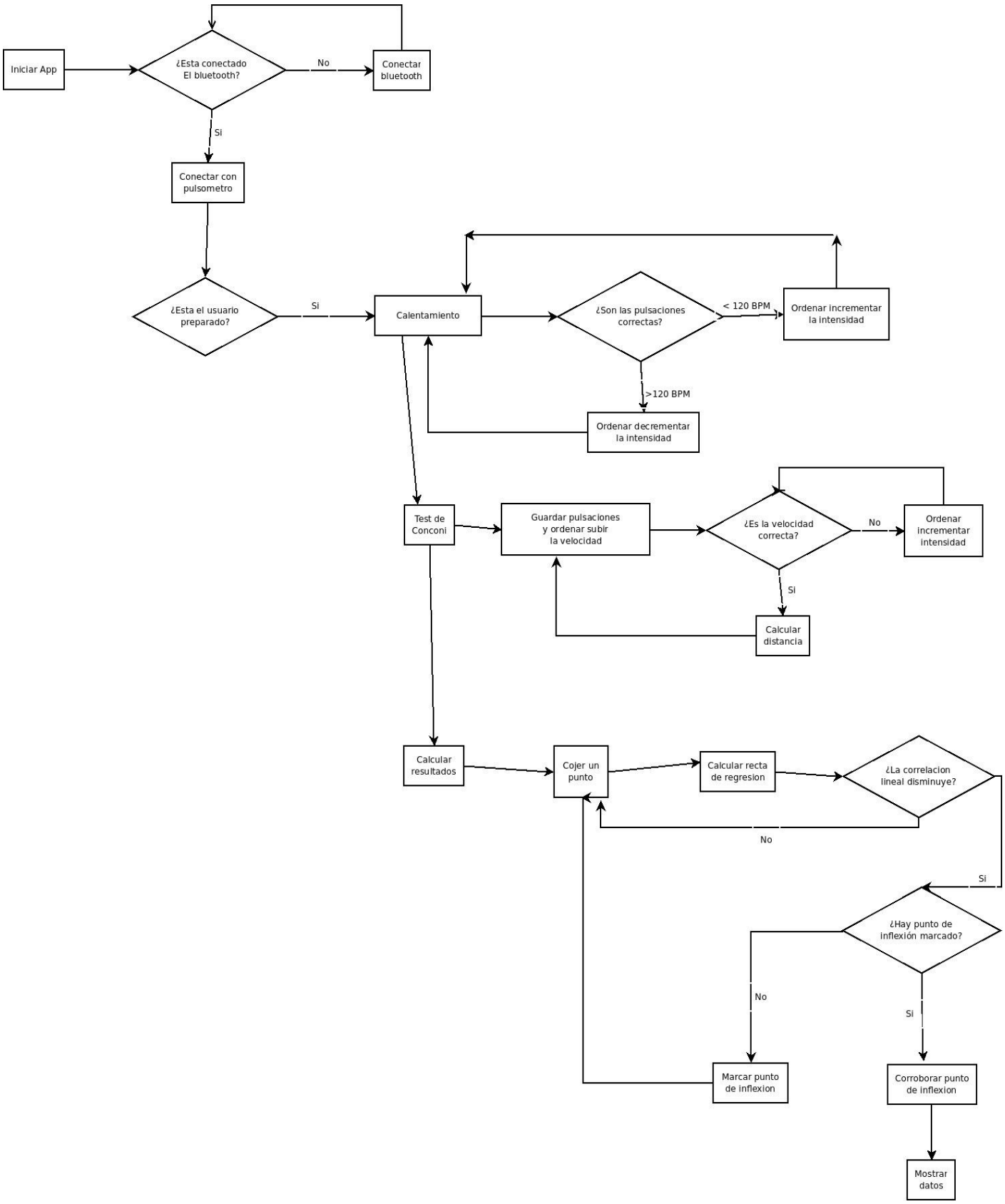
3.3.1.1 Diagrama casos de uso

En este diagrama mostramos las actividades que el usuario puede realizar.



3.3.1.1- Diagrama de flujo

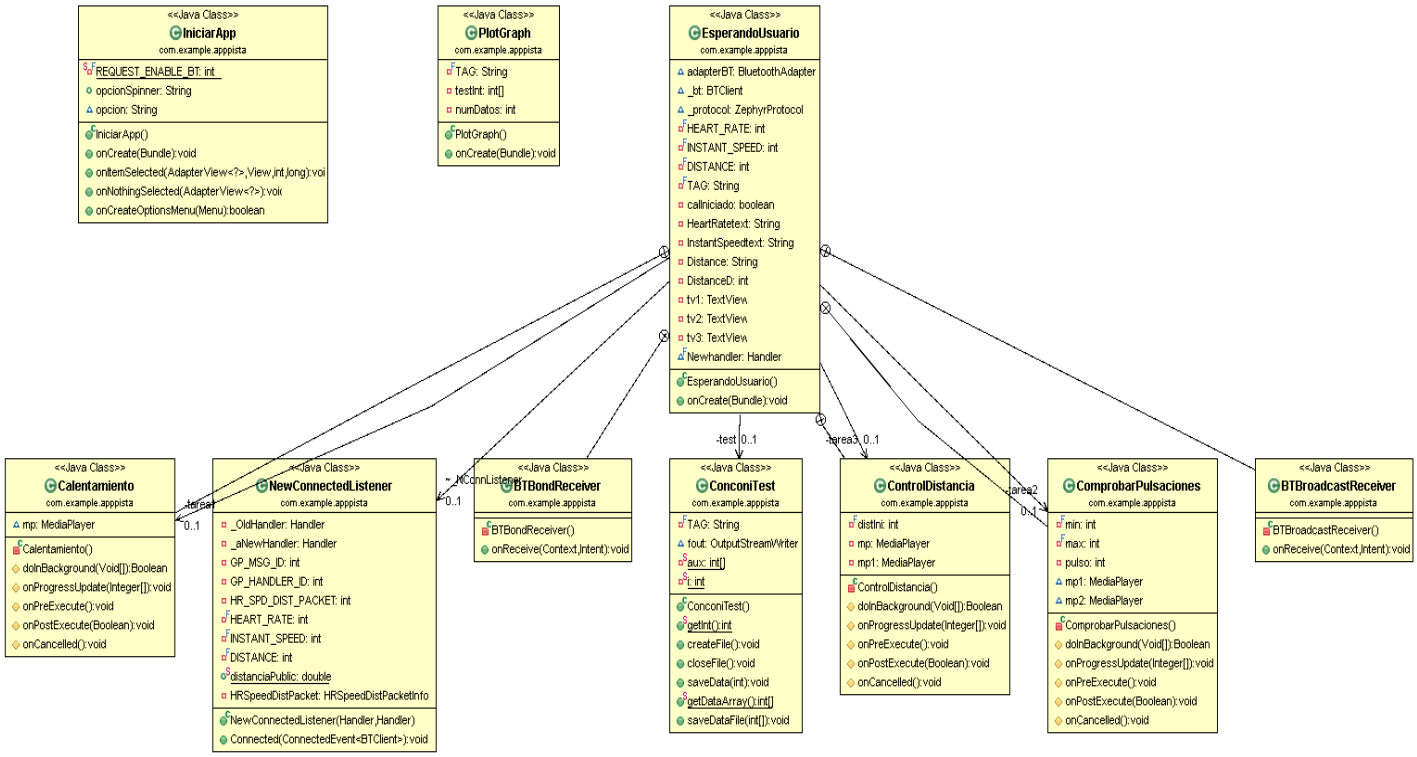
En el siguiente diagrama se muestra el flujo de la aplicación, de principio a fin, dependiendo de las acciones realizadas por el usuario.



3.3.1.1 Diagrama de clases

En este apartado se muestra el diagrama de clases, el cual representa las clases que necesitamos implementar.

Debemos comentar en este caso que la aplicación consta de tres pantallas diferenciadas como se podrá apreciar.



3.3.1.2 Gestión y verificación de datos

Como hemos dicho anteriormente, los datos que vamos a recoger son las pulsaciones durante el test de esfuerzo. En dicho test, las instrucciones son muy claras, no obstante, durante la ejecución de la prueba pueden surgir algún tipo de complicaciones, ya sea por no entender las instrucciones o no querer seguirlas.

Esto puede repercutir en los valores de los datos que estamos recogiendo, destruyendo la tendencia que estos deberían tener.

Nosotros, como desarrolladores, no podemos evitar este tipo de irregularidades, en cambio, podemos intentar corregirlas, sabiendo que los resultados que podamos obtener a partir de estas pueden ser erróneos.

A continuación, comentaremos cuales son los principales fallos y como los hemos solventado:

- El usuario no sigue la instrucción de incrementar la velocidad:

En este caso, no incrementar la velocidad significa que en el *array* de resultados habrá dos datos para una misma velocidad. Este caso lo podemos reconocer fácilmente si el coeficiente de correlación varia mucho entre dichas variables o directamente tienen el mismo valor.

- El usuario no pulsa el botón finalizar al acabar el test:

Si el usuario no pulsa dicho botón, las pulsaciones continuarán almacenándose hasta que lo pulse, por tanto, habrá un segmento de datos falseados.

Podemos corregir este error comprobando si hay algún dato con valor inferior al anterior y eliminarlo.

Otra variante de este problema es cuando el usuario retira el pulsómetro antes de presionar el botón finalizar.

En este caso la solución es la misma, incluso mas fácil de detectar pues las pulsaciones pasan directamente a 0.

- El usuario incrementa menos de 1Km/h:

Este escenario puede ocurrir por dos motivos.

El primero es que el usuario se siente cansado y no quiere hacer el esfuerzo necesario para incrementar 1Km/h. En este caso, el dato recogido en el siguiente ciclo estará falseado y nosotros no podremos hacer nada para detectarlo, por tanto la gráfica resultante puede no ser del todo exacta.

El segundo motivo puede ser por un error en la máquina de correr. Si por ejemplo, aumentamos 1Km/h en la maquina y en realidad el incremento es de 0'8Km/h, suponiendo que cada incremento será de 0'8Km/h la gráfica podría generarse correctamente, pues el incremento de velocidad será el mismo para cada ciclo.

- El usuario da por finalizada la prueba antes de haber llegado al fallo muscular.

Este es el mas común de los errores producidos en este test.

Afortunadamente, esto no supone un gran error siempre y cuando se haya llegado a la zona anaeróbica.

En este caso el error no se produciría para el punto límite entre la zona aeróbica y la zona anaeróbica, sino en el cálculo de la frecuencia cardíaca máxima.

En definitiva, existen varios errores comunes realizados por los usuarios y estos pueden ser resueltos en mayor o menor medida, aunque pueden variar los resultados proporcionados por la aplicación.

En el caso de detectar algún error, éste se le será comunicado al usuario al finalizar la prueba para informar que los resultados no son fiables al 100% y animar a usar la aplicación de nuevo correctamente.

3.3.1.2 Procesamiento de datos

Una vez finalizado el test y teniendo todos los datos verificados pasamos a procesarlos para encontrar umbral anaeróbico y la frecuencia cardiaca máxima.

Para ello, como dijimos anteriormente tendríamos que construir la gráfica resultante y distinguir entre el segmento formado por una recta y el segmento formado por una curva. En este caso, el punto intermedio entre estos dos segmentos seria el umbral anaeróbico.

Sin embargo, desde el punto de vista de la programación no podemos distinguir gráficamente dichos segmentos por lo que tenemos que usar otro método para calcular el umbral.

Un buen método seria usar las propiedades matemáticas de la gráfica proporcionada junto a operaciones estadísticas para calcular los puntos en los que la tendencia cambia.

Podríamos formular la sentencia anterior de forma matemática como: "*Encontrar el punto en el que la correlación pasa de ser lineal a ser logarítmica*".

Para ello vamos a utilizar algunos métodos de la estadística descriptiva para el análisis de variables unidimensionales.

Podemos descubrir este cambio de tendencia utilizando en *coeficiente de correlación*.

El *coeficiente de correlación,* como su propio nombre indica, mide el grado de correlación que tienen las variables a estudiar y está comprendido entre -1 y +1.

Dichos valores solo se alcanzarán en el caso de que haya una relación lineal exacta . Se tiene +1 si la recta es creciente o -1 si es decreciente.

Por tanto, nuestra gráfica tendrá un coeficiente de correlación aproximado de 1, que en un punto determinado degenerará, pues ya no será una relación lineal.

La única forma de encontrar dicho punto es calcular para cada nuevo punto, el coeficiente de correlación con todos los demás puntos, para posteriormente compararlos entre ellos y decidir en que punto la correlación lineal decae.

El *coeficiente de correlación* se calcula mediante esta formula:

$$r = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

Para utilizar esta formula, primero debemos calcular la *covarianza* y la *desviación típica* con las siguientes fórmulas:

Covarianza:

$$\sigma_{xy} = \frac{\sum f_i (x_i - \bar{x})(y_i - \bar{y})}{N}$$

Desviación típica:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{N}}$$

Hay que tener en cuenta que estas operaciones habría que calcularlas para cada punto, lo cual se ha haría trabajoso, por tanto hemos decidido que una vez recogidos y verificados todos los datos los introduciremos uno a uno en una tabla, calculando adicionalmente algunos valores que nos facilitarán los cálculos como "x²", "y²" y "x·y", así como el sumatorio total de todos estos valores.

Explicaremos esta metodología con un ejemplo practico.

En el *array* de resultados tenemos estos datos:

120,130,140,150,160,170,178,184,188,190,190,191

Colocamos los datos en una tabla.

x	y
1	120
2	130
3	140
4	150
5	160
6	170
7	178
8	184
9	188
10	190
11	190
12	191

Una vez tenemos la tabla, agregamos las funciones que hemos comentado anteriormente (“x^2”, “y^2” y “$x \cdot y$”) , además de los sumatorios de cada una (“$\sum x$”,”$\sum y$”,“$\sum x^2$”, “$\sum y^2$” y “$\sum x \cdot y$”).

	x	y	x^2	y^2	$x \cdot y$
	1	120	1	14400	120
	2	130	4	16900	260
	3	140	9	19600	420
	4	150	16	22500	600
	5	160	25	25600	800
	6	170	36	28900	1020
	7	178	49	31684	1246
	8	184	64	33856	1472
	9	188	81	35344	1692
	10	190	100	36100	1900
	11	190	121	36100	2090
	12	191	144	36481	2292
∑	78	1991	650	337465	12912

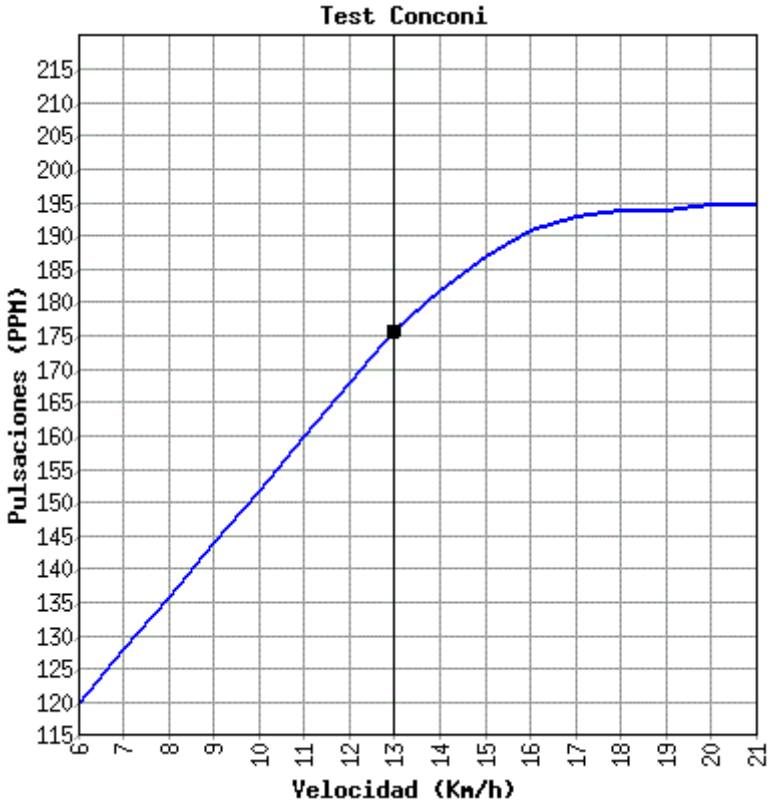
Hay que notar que el sumatorio se calcula para cada punto, por tanto, tendrá un valor diferente cada vez que se utilice en las funciones anteriormente descritas, pues el coeficiente de correlación se calcula para cada uno de estos puntos.

Los resultados obtenidos son los siguientes:

x	y	Coeficiente de correlación
1	120	NaN
2	130	1
3	140	0.9999999999999933
4	150	0.9999999999999999
5	160	0.9999999999999998
6	170	0.9999999999999966
7	178	0.9996006442779504
8	184	0.9978553562450141
9	188	0.9938742420458652
10	190	0.9865796039333553
11	190	0.9743693781510674
12	191	0.9614741810442838

Como vemos, hay una relación lineal muy fuerte en los primeros 6 puntos*, que cambia cuando llegamos al 7º punto y que, a partir de este punto, decrece rápidamente.

Podemos observar en la gráfica, que en este punto en concreto es donde la gráfica pasa de ser *lineal* a ser *logarítmica.*



3.3.2- Especificación no funcional

En esta aplicación vamos a implementar una aplicación Android que simule la realización del test de Conconi en una cinta de correr. Esta aplicación recogerá datos de un pulsómetro conectado al dispositivo mediante Bluetooth, por tanto los dispositivos necesarios son:

- Teléfono o tableta con tecnología Android.
- Pulsómetro con tecnología Bluetooth.
- Cinta de correr

Los siguientes consejos no son obligatorios pero si recomendables:

- Consulte la opinión de su médico, recuerde que es un test de esfuerzo.
- Realizar ejercicios de movilidad articular y estiramientos dinámicos.
- Estar descansado.
- Hidratase bien antes del esfuerzo.
- Usar ropa cómoda.
- Tener a mano una toalla.

3.4 Diseño de la Aplicación

En este apartado vamos a explicar el diseño de las interfaces utilizadas en esta aplicación.

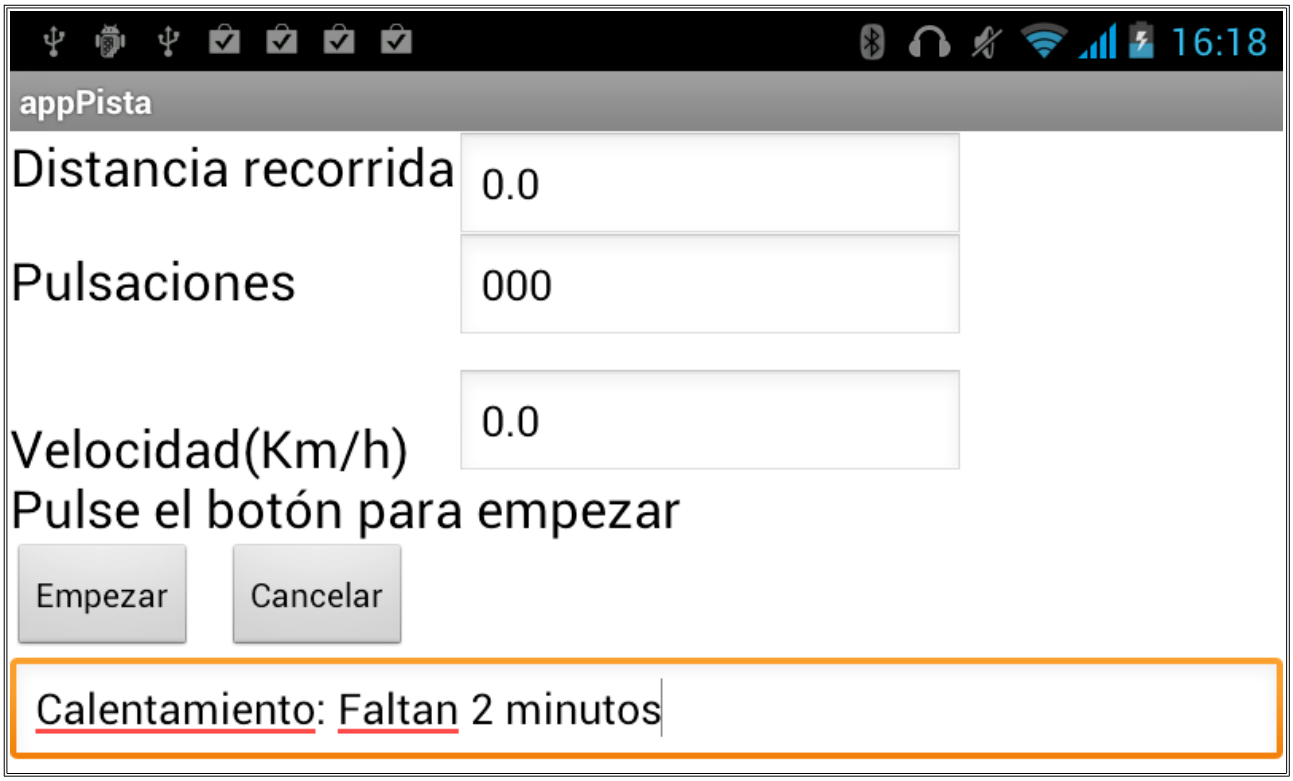
La aplicación está dividida en 3 pantallas, en las que cada una representa un paso en el flujo de la aplicación.

La primera pantalla es la pantalla de inicio. En ella seleccionamos el escenario en el que vamos a realizar el test.

Como vemos, la pantalla esta formada por una lista desplegable y un botón.

Las opciones de la lista indican los escenarios posibles, de los cuales el usuario debe elegir uno.

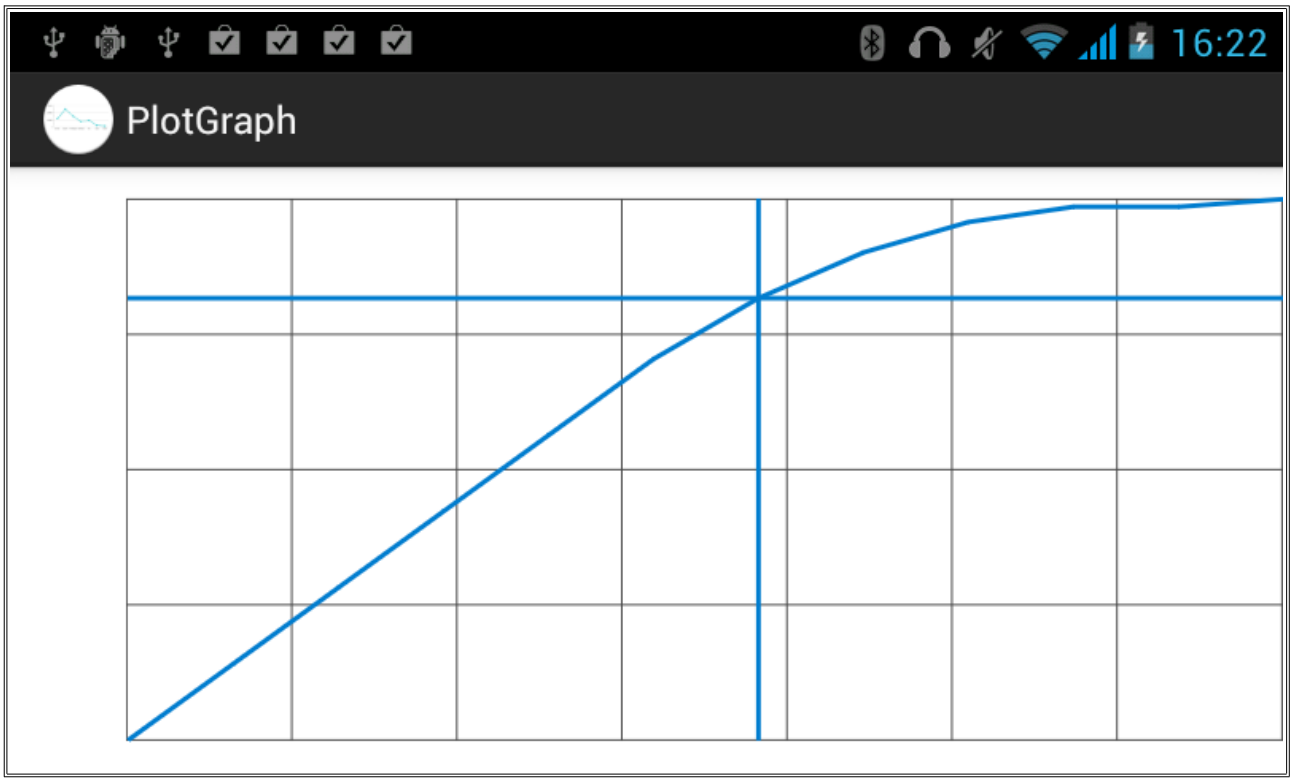
Mientras con el botón pasamos a la siguiente pantalla.



En esta segunda pantalla es donde de va a desarrollar todo el test.
Como vemos, en la parte superior tenemos tres campos de texto donde mostramos la información recibida por el programa.
Mas abajo están situados 3 botones (el 3º está oculto al inicio), que controlaran el flujo de la prueba de esfuerzo.
El primero inicia la prueba de esfuerzo.
Es segundo cancela la prueba en caso de que haya algún inconveniente durante el calentamiento.

Un vez finalizado el calentamiento, este botón se llamará "*Fin*", que se deberá pulsar cuando el usuario haya llegado al fallo muscular en la prueba de esfuerzo y supondrá el fin de esta.

Una vez la prueba de esfuerzo haya sido realizada, aparecerá el tercer botón "*Dibujar*", el cual dará paso a la tercera y última pantalla, en la cual se dibuja una gráfica partiendo de la información que ha sido recolectada durante la prueba.



Además de la gráfica, se dibujarán dos lineas que indicarán el punto en el que el usuario pasa de la zona aeróbica a la zona anaeróbica, siendo este, el punto del umbral de la zona anaeróbica.

4 Implementación

En este apartado, vamos a explicar como hemos implementado la aplicación a partir de los diagramas propuestos anteriormente.

Vamos a recalcar la parte de programación, mostrando, documentando y explicando los algoritmos mas interesantes.

Dividiremos este apartado en tres partes. Cada una corresponderá a cada una de las pantallas, explicando primero la programación de la interfaz y posteriormente la conexión con la aplicación y la lógica interna.

En la primera pantalla preparamos todos los servicios que vayamos a necesitar durante la ejecución de la aplicación. Por tanto, el primer paso es comprobar si el *Bluetooth* esta conectado y sino, consultar al usuario si desea activarlo.

```
if (!mBluetoothAdapter.isEnabled()){
            Log.d("out", "bluetooth habilitando");
            Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
        }
```

Como dijimos en el apartado de 3.5 (Diseño de la aplicación), la interfaz consta de una lista, en la que podemos elegir el escenario en el cual vamos a realizar la prueba.

En *Android* dicha lista se llama *spinner*, el cual debemos conectar con las opciones que queremos que ofrezca.

Para ello creamos un fichero XML en la carpeta res (recursos) con el siguiente código:

```
<string-array name="tests_array">
        <item>Pista de atletismo</item>
        <item>Cinta de correr</item>
        <item>Bicicleta estatica</item>
    </string-array>
```

Una vez creado este fichero, tan solo tenemos que conectar el *spinner* a la lógica del programa e importar esta información.

Con la siguiente instrucción creamos un objeto *Spinner* en la lógica del programa.

```
Spinner spinner1 = (Spinner)findViewById(R.id.spinner1);
```

A continuación creamos el adaptador entre el fichero XML y el spinner de la interfaz

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
             R.array.tests_array, android.R.layout.simple_spinner_item);
```

Para posteriormente asignarlo al spinner de la lógica del programa

```
spinner1.setAdapter(adapter);
```

Un botón es una de las formas mas comunes de pasar de una pantalla a otra o de realizar cualquier actividad.
En este caso, mediante el botón situado en esta pantalla, lanzaremos un *Intent*, para pasar a la siguiente pantalla.

```
public void onClick(View arg0) {
                Intent intent = new Intent(IniciarApp.this,
EsperandoUsuario.class);
                Bundle bundle = new Bundle();
                bundle.putString("opcion", opcionSpinner);
                intent.putExtras(bundle);
                startActivity(intent);
            }
```

Como podemos observar, antes de iniciar la actividad *EsperandoUsuario*, hemos creado un *Bundle* y lo hemos incluido en el *Intent.*
Esta es una de las formas de pasar argumentos de un *Activity* a otro.

Un vez se ha dado la orden de pasar de una pantalla a otra, creamos un IntentFilter que informará al Sistema operativo Android que va a efectuar una petición de emparejamiento.

```
IntentFilter filter =
new IntentFilter("android.bluetooth.device.action.PAIRING_REQUEST");
```

Creamos un objeto *BTBroadcasReciever*

```
BTBroadcastReceiver aux = new BTBroadcastReceiver();
```

Que empareja automáticamente con el pulsómetro.

Posteriormente, creamos un objeto *BTClient*, de la librería proporcionada por el fabricante, mediante el cual, conectamos con el dispositivo.

Una vez hemos conectado el pulsómetro, pasamos a implementar la información que se mostrará en la interfaz.

En primer lugar, inicializaremos los campos de texto donde mostraremos la información recibida por el pulsometro. Desde el punto de vista de la programación, no es necesario inicializar las dichas variables, aunque de esta manera, la aplicación quedará más atractiva cara al usuario.

```
tv1 = (EditText)findViewById(R.id.pulse);
             tv1.setText("000");

             tv2 = (EditText)findViewById(R.id.speed);
             tv2.setText("0.0");

             tv3 = (EditText)findViewById(R.id.distance);
             tv3.setText("0.0");
```

Cabe comentar que las variables *tv1*, *tv2* y *tv3* son variables globales, que se irán actualizando, gracias a un objeto *Handler* que crearemos y que será el encargado de recibir la información recibida por el pulsómetro y formatearla.

```
final  Handler Newhandler = new Handler(){
           public void handleMessage(Message msg)
           {

           switch (msg.what)
           {
           case HEART_RATE:
                 HeartRatetext = msg.getData().getString("HeartRate");
                 if (tv1 != null)tv1.setText(HeartRatetext);
           break;

           case INSTANT_SPEED:
                 InstantSpeedtext = msg.getData().getString("InstantSpeed");
                 double aux = Double.parseDouble(InstantSpeedtext) * 3.6;
                 InstantSpeedtext = Double.toString(aux);
                 if (tv2 != null)tv2.setText(InstantSpeedtext);
           break;

           case DISTANCE:
                 Distance = msg.getData().getString("Distance");
                 DistanceD = (int)Double.parseDouble(Distance);
           break;
           }
     }
   };
```

Como vemos, hemos hecho una pequeña modificación en el apartado *INSTANT_SPEED*, en el cual, inicialmente, los datos son enviados en m/s, con lo que multiplicando por 3'6 los pasamos a Km/h, que es la medida que utilizaremos durante la prueba de esfuerzo.

En el apartado *HEART_RATE,* tenemos un problema de desbordamiento de datos. El paquete enviado tiene 8bits con signo, para representar el valor de la frecuencia cardiaca. Tomando en cuenta que se utiliza un bit para el signo, nos quedan 7 bits, con los que podemos extraer un número como máximo de 127.

Posiblemente las pulsaciones del usuario que esté usando la aplicación superarán esta número por tanto deberemos adecuar los datos para su correcta representación.
Una vez la frecuencia cardiaca ha llegado a 127, al superar este umbral pasará a -127 e irá decreciendo.
Por lo que utilizaremos el siguiente algoritmo para normalizar el valor

```
if (HRate<0) HRate=128+HRate+128;
```

Tenemos el mismo problema con la variable *DISTANCE*, en la que el paquete de datos le concede 8bits sin signo para representar el valor.
Esto supone un máximo de 256m, a partir de los cuales se reinicia.
En la prueba de esfuerzo necesitamos calcular tramos de 200m, con lo que este valor podría ser suficiente si no fuera porque realizaremos varios ciclos de 200m.
En principio, la mejor solución seria reiniciar dicho contador cada 200m, pero esto no es posible ya que la cuenta proviene del pulsómetro y no la podemos modificar.

Otra opción posible es realizar una cuenta con los metros recorridos por el usuario y dividirlos por 200 para averiguar cuando se produce un cambio de ciclo.

```
//Calculo de overflow
if(aux >= 150 && !flag1){
	flag1 = true;
	}
else if(aux<100 && flag1){
	nOverflows++;
	flag1 = false;
}
//se suma siempre 256 por si hubiera habido overflow
dist = ((aux+256-distIni)%256)+256*nOverflows;
```

En este punto, vamos a obviar la programación de los botones, sin embargo, estos realizaban las funcionalidades básica del programa.
Por ejemplo, el botón *Empezar,* ejecutaba una tarea asíncrona, en la cual se inicia el calentamiento.

```
private class Calentamiento extends AsyncTask<Void, Integer, Boolean> {
	MediaPlayer mp = MediaPlayer.create(EsperandoUsuario.this, R.raw.inical);
	@Override
	protected Boolean doInBackground(Void... params) {

		for(int i=1; i<=10; i++) {

			try {
				// 1s/180s = 0,0055s hay que restar 5ms que tiene
				de desfase
				Thread.sleep(995);
				//wait(10000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			publishProgress(i);
			if(isCancelled())
				break;
```

```
                }
            //Calentamiento finalizado
            return true;
      }

      @Override
      protected void onProgressUpdate(Integer... values) {
            int progreso = values[0].intValue();
            ((TextView) findViewById(R.id.editText1)).setText("Calentamiento:
"+progreso+" segundos.");
      }
@Override
      protected void onPreExecute() {
            calIniciado = true;
            Log.d(TAG,"Antes de iniciar la tarea2");
            mp.start();
            tarea2 = new ComprobarPulsaciones();
                  tarea2.execute();
            ((TextView) findViewById(R.id.editText1)).setText("Iniciamos el
calentamiento");
      }

      @Override
      protected void onPostExecute(Boolean result) {
            if(result){
            Toast.makeText(EsperandoUsuario.this, "Calentamiento finalizado!",
Toast.LENGTH_SHORT).show();
            ((TextView) findViewById(R.id.editText1)).setText("Almacenando
pulsaciones");
            tarea2.onCancelled();
            tarea3 = new ControlDistancia();
                  tarea3.execute();
            calIniciado = false;
            }

      }

      @Override
      protected void onCancelled() {
            tarea1.cancel(true);
            Toast.makeText(EsperandoUsuario.this, "Tarea cancelada!",
Toast.LENGTH_SHORT).show();
            ((TextView) findViewById(R.id.editText1)).setText("Calentamiento
cancelado");
            calIniciado = false;
      }
    }
```

Como podemos observar, hemos utilizado *Thread.sleep()* para medir los segundos, sin embargo había un pequeño desfase que solucionamos de la siguiente forma.
Extrajimos como muestra el desfase provocado en 3 minutos y lo dividimos entre 180s, como mostramos a continuación.
1s/180s = 0,0055s hay que restar 5ms que tiene de desfase.
Por lo que un segundo equivalía a 995ms

Durante el calentamiento, el usuario debería mantener las pulsaciones al rededor de 120BPM, para ello, desde la tarea Calentamiento, lanzamos otra tarea asíncrona para controlar el pulso.
Donde situaremos el siguiente bucle.

```
pulso = Integer.parseInt(HeartRatetext);
                  if(pulso<=min){
                        //Log.d(TAG,"Pulsaciones bajas");
                        mp1.start();
                  }else if(pulso >= max){
                        //Log.d(TAG,"Pulsaciones altas");
                        mp2.start();
                  }
```

Donde *mp1* y *mp2* son objetos *MediaPlayer*, que contienen instrucciones sonoras de incrementar o decrementar la intensidad.

Una vez finalizado el calentamiento, empezaremos el test.
Por lo tanto, iniciaremos otra actividad asíncrona, que controlará la distancia y recogerá datos.

```
while(true){
                  aux = DistanceD;
                  //Calculo de overflow
                  if(aux >= 150 && !flag1){
                        flag1 = true;
                        }
                  else if(aux<100 && flag1){
                        nOverflows++;
                        flag1 = false;
                  }
                  //se suma siempre 256 por si hubiera habido overflow
                  dist = ((aux+256-distIni)%256)+256*nOverflows;
                  //dist es la distancia real desde que empezamos el
                  monitoreamiento.
                  attr = (int)dist;
                  publishProgress(attr);
                  dist = dist%200;
                  if(dist>190 || dist<5){
                        test.saveData(Integer.parseInt(HeartRatetext));
                        mp.start();
            }
```

Este bucle se repetirá hasta que el usuario dé por terminado el test.

Una vez dado este caso se procederá a dibujar la gráfica resultante en la tercera pantalla.
En este caso, usaremos la biblioteca *GraphView*, la cual nos proporciona herramientas muy sencillas para dibujar nuestra gráfica.

```
GraphViewData[] arrayDatos = new GraphViewData[aux.length];
for(int i=0;i<aux.length;i++)
      arrayDatos[i] = new GraphViewData(i+1,aux[i]);
GraphViewSeries exampleSeries = new GraphViewSeries(arrayDatos);
GraphView graphView = new LineGraphView(this,"GraphViewDemo");
graphView.addSeries(exampleSeries);
```

Este código, es la parte donde se extraen los datos del *array* de resultados, para convertirlos en puntos que, posteriormente serán unidos al dibujar la gráfica.

Sin embargo, en esta apartado es donde se realizan los cálculos estadísticos para extraer el punto del umbral aeróbico.

Como explicamos anteriormente, formamos una tabla, con algunos valores predefinidos que favorecerán la claridad del código y las operaciones que necesitaremos más adelante.
Para explicar el hilo del código, mostraremos primero la tabla resultante y posteriormente el código que la generará.

	x	y	x^2	y^2	$x \cdot y$
	1	120	1	14400	120
	2	130	4	16900	260
	3	140	9	19600	420
	4	150	16	22500	600
	5	160	25	25600	800
	6	170	36	28900	1020
	7	178	49	31684	1246
	8	184	64	33856	1472
	9	188	81	35344	1692
	10	190	100	36100	1900
	11	190	121	36100	2090
	12	191	144	36481	2292
∑	78	1991	650	337465	12912

```
double covarianza, desvTipicaX, desvTipicaY;
boolean marca1 = false, marca2 = false;
int punto = 0;
//Ponemos los resultados a 0
for(int i=0;i<5;i++)
      tabla[i][numDatos] = 0;
//Empezamos a rellenar la tabla
for(int y=0;y<numDatos;y++){
      // X
      tabla[0][y] = y+1;
      // Y
      tabla[1][y] = testInt[y];
      // X al cuadrado
      tabla[2][y] = Math.pow(tabla[0][y],2);
      // Y al cuadrado
      tabla[3][y] = Math.pow(tabla[1][y],2);
      // X + Y
      tabla[4][y] = tabla[0][y] * tabla[1][y];
      //Implementamos sumatorio de todo
      //sum X
      tabla[0][numDatos] += tabla[0][y];
      //sum Y
      tabla[1][numDatos] += tabla[1][y];
      //sum X al cuadrado
      tabla[2][numDatos] += tabla[2][y];
      //sum Y al cuadrado
      tabla[3][numDatos] += tabla[3][y];
      //sum X numDatos* Y
      tabla[4][numDatos] += tabla[4][y];
      //Calculamos la desviacion tipica de X
      double aux1, aux2, aux3;
      aux1 = tabla[2][numDatos] / tabla[0][y];
      aux2 = tabla[0][numDatos] / tabla[0][y];
      //aux3 media de X
      aux3 = aux2;
      desvTipicaX = Math.sqrt(aux1-Math.pow(aux2, 2));
      aux1 = tabla[3][numDatos] / tabla[0][y];
      aux2 = tabla[1][numDatos] / tabla[0][y];
      desvTipicaY = Math.sqrt(aux1-Math.pow(aux2, 2));
      //falta covarianza
      aux1 = tabla[4][numDatos] / tabla[0][y];
      covarianza = aux1 - (aux2 * aux3);
      Log.a(TAG,"En X = "+(y+1));
      Log.a(TAG,"Desviacion tipica de X: "+desvTipicaX);
      Log.a(TAG,"Desviacion tipica de Y: "+desvTipicaY);
      Log.a(TAG,"Covarianza: "+covarianza);
      resultados[y] = covarianza / (desvTipicaX *
      desvTipicaY);

      Log.a(TAG,"Coeficiente de corelación: "+ resultados[y]);
      Log.a(TAG,"Punto de inflexion: "+punto);
      }
```

El coeficiente de correlación, calculado para cada punto se almacena en un *array* de resultados y su contenido podría representarse de la siguiente manera.

x	y	Coeficiente de correlación
1	120	NaN
2	130	1
3	140	0.9999999999999933
4	150	0.9999999999999999
5	160	0.9999999999999998
6	170	0.9999999999999966
7	178	0.9996006442779504
8	184	0.9978553562450141
9	188	0.9938742420458652
10	190	0.9865796039333553
11	190	0.9743693781510674
12	191	0.9614741810442838

Para averiguar en que punto la correspondencia lineal decrece, calculamos la media de los 4 primeros términos, que estarán situados en la zona aeróbica y nos darán una pista de la dispersión que pueden tener los datos entre ellos en la parte lineal, para posteriormente compararlo con los sucesivos términos hasta encontrar alguno que se salga de la zona de confianza.

Como fue explicado en el diagrama de flujo, se realizarán dos comprobaciones. La primera marcará el punto como posible punto de inflexión y se seguirá con el cálculo. Si la correspondencia sigue decreciendo, aceptaremos el punto anterior como umbral anaeróbico y detendremos la ejecución.

```
double correlacionMin =
(resultados[1]+resultados[2]+resultados[3]+resultados[4]) / 4.0;
                correlacionMin *= 0.985;
                for(int i=0;i<numDatos;i++){
                if(resultados[i]<=correlacionMin){
                      if(marca1 == false) marca1 = true;
                      else if(marca2 == false) {
                                  marca2 = true;
                                  punto = i-1;
                                  break;
                                  }
                }else {marca1=false; marca2=false;}
                }
```

Cuanto hayamos calculado el punto, lo marcaremos mediante el cruce de dos rectas.
Crearemos la primera situando el punto de la primera en el eje X.

```
limite1[0] = new GraphViewData(punto,aux[0]);
limite1[1] = new GraphViewData(punto,aux[aux.length-1]);
```

y la segunda, situando el punto en el eje Y.

```
limite2[0] = new GraphViewData(1,tabla[1][punto-1]);
limite2[1] = new GraphViewData(aux.length,tabla[1][punto-1]);
```

Para posteriormente crear e insertar las gráficas.

```
GraphViewSeries exampleSeries2 = new GraphViewSeries(limite1);
GraphViewSeries exampleSeries3 = new GraphViewSeries(limite2);

graphView.addSeries(exampleSeries2);
graphView.addSeries(exampleSeries3);
```

Estas dos rectas marcarán el punto exacto en el que se sitúa el umbral anaeróbico del usuario.

5 Conclusión

Como conclusión podemos decir que todos los objetivos que nos habíamos propuesto han sido realizados, tanto los relacionados con la aplicación como los objetivos personales.

Con respecto a la aplicación, ahora cualquier usuario con un teléfono Android, puede realizar un test de esfuerzo para conocer cuales son sus umbrales de entrenamiento, con los cuales puede planificar sus entrenamientos.

La aplicación es rápida, ágil y tiene la misma apariencia que las aplicaciones Android que el teléfono lleva instaladas por defecto, por lo que el usuario podrá usarla sin tener ningún tipo de inconveniente con el flujo de la aplicación.

Además los resultados se presentan en forma gráfica y muy clara, lo que facilita el entendimiento del usuario.

Respecto a los objetivos personales, he aprendido mucho durante la realización del proyecto, sobre todo en la fase de búsqueda de información, en la cual tuve que adquirir muchos conocimientos sobre el sistema cardiorespiratorio, mecanismos de obtención de energía en las células, zonas de entrenamiento, etc y posteriormente durante la parte de implementación pues nunca antes había desarrollado una aplicación en Android.

Además de tener que recordar conocimientos de asignaturas anteriores como Estadística o Ingeniería del Software.

Desde el inicio de la carrera hemos estado programando varias aplicaciones sencillas, como método de entrenamiento, que cada vez fueron más y más complejas pero siempre han sido ejercicios predeterminados, sin embargo, con este proyecto he hecho una cosa que quería hacer desde hace tiempo, esta es crear y desarrollar una idea propia, dentro de un contexto que me motive y que esta sea útil para todos aquellos a los que había pensado que iría dedicada.

Para finalizar, debo decir que ha sido una gran experiencia la realización de este proyecto, pues realmente he disfrutado con él y ha sido la culminación de mi vida como estudiante y una preparación para el mundo laboral.

6 Bibliografía

http://developer.android.com/index.html

http://stackoverflow.com/

http://www.zephyr-technology.com/

http://www.wikipedia.org

http://www.vitutor.com

Métodos estadísticos en ingenieria - Rafael Romero Villafranca y Luisa Rosa Zúnica Ramajo. Editorial UPV

Statistics. Concepts and Controversies 5th Edition -David S. Moore