ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

"Aplicación de aprendizaje de tareas cotidianas en personas con discapacidad mediante dispositivos móviles"

Asier Alejo Siles

Dr José Javier Astrain Escola

Pamplona, 14 de Noviembre de 2012

AGRADECIMIENTOS

Tras estos duros años académicos en los cuales se han sucedido muy buenos momentos con otros quizá no tanto, en los que he tenido que superarme y crecer como persona, me gustaría dedicar unas palabras a todas esas personas que no me permitieron tirar la toalla en los momentos difíciles y agradecerles profundamente su apoyo, ya que sin ellos es posible que hoy no fuera como soy.

A mis padres y a mi hermano, que siempre me han apoyado y arropado en los momentos de más dificultad, no solo académica, sino también en los momentos difíciles de la vida. Que siempre han estado ahí para lo que hiciera falta y han celebrado mis logros y me han dado fuerza para continuar. Los mismos que me han ayudado y evaluado este proyecto final y muchos otros. Gracias por indicarme todos esos matices de los que nunca me hubiera dado cuenta. Gracias por todos los consejos y cariño. Gracias por todo.

A Amaia, que siempre ha estado motivándome y valorando mi trabajo. La misma con la que he compartido gran número de horas de estudio y buenos momentos, sin la que nunca hubiera podido llegar tan lejos. Gracias por inspirarme en todos y cada uno de mis trabajos, gracias por no tirar la toalla conmigo, por estar siempre a mi lado y por todo el tiempo que has pasado conmigo y que, lógicamente, no has dispuesto para ti. Muchas gracias por todo, nunca lo olvidaré.

A José Javier Astrain, tutor de este proyecto, el cual me permitió desarrollar esta aplicación y me brindó la oportunidad y la idea para comenzar un proyecto en el cual me ha encantado trabajar y me ha permitido demostrar todo lo aprendido y adquirir muchos nuevos conocimientos. Gracias por la idea y por abrir un nuevo camino ante mí.

A mi amigo Aitor, el cual me ha soportado como el que más, que ha pasado por alto todas mis rarezas y falta de atención en alguna que otra conversación. El mismo que ha pasado aislado largas horas de estudio en la biblioteca, laboratorio, clase… Y que en muchas ocasiones ha ejercido de buena conciencia. Gracias.

A mis amigos, Eneko, Naroa, Iñaki, Ekaitz, Asier, y toda la cuadrilla, por haberme acogido como a uno más y gracias a los cuales también he podido desconectar de tantas horas de estudio o estrés acumulado. También por ejercer de conciencia menos buena en muchos momentos, los cuales son tan necesarios como las intensas jornadas de estudio.

A mis amigos y compañeros de clase, buenos compañeros que me han aceptado como soy y me han soportado durante tantos años. Gracias por todos esos ratos de relajación y estudio. Nunca os olvidaré.

Y gracias a todas las personas que han estado conmigo y me han animado por seguir adelante y nunca conformarme por no dedicar más esfuerzo para obtener algo mejor.

A todos ellos, gracias.

Índice

1.- Introducción

1.1 Problema a resolver

La idea de este proyecto surgió de una necesidad real de la sociedad, concretamente de la dificultad para conocer el manejo de las monedas y billetes, en este caso de €, que tienen las personas con discapacidad intelectual. Dicha dificultad hace más compleja su integración en la sociedad debido al desconocimiento del valor práctico del dinero en situaciones de la vida cotidiana, especialmente en la compra diaria y de productos de coste no muy elevado. En concreto, el desarrollo se ha centrado en el perfil de niños con Síndrome de Down, aunque puede aplicarse perfectamente a otro tipo de usuarios.

Se realizaron diversas investigaciones, destacando conversaciones con familiares con niños dependientes y búsqueda de información, destacando un documento de Anna Hortsmeier [1] en el que se detallan algunos aspectos que a continuación se indican.

"El manejo del dinero por parte de las personas con síndrome de Down (SD) es un campo en el que suelen darse limitaciones importantes, por cuanto se relaciona con sus dificultades con el cálculo matemático.

El aprendizaje de los cálculos más elementales es costoso para ellas. Tienen dificultades con los ejercicios matemáticos, numéricos y con las operaciones.

Los niños con SD necesitan un trabajo sistemático y adaptado en matemáticas y que se les proporcionen estrategias para adquirir conceptos matemáticos básicos.

Sin embargo, el dinero es un instrumento fundamental en el desenvolvimiento social de cualquier ciudadano y por tanto, ha de ser trabajado para favorecer la adecuada normalización de las personas con SD.

Habitualmente muestran poco interés por el dinero, probablemente porque no lo necesitan en su vida cotidiana.

En general, la relación de los niños con síndrome de Down con el dinero es escasa, encargándose los adultos que les rodean de proporcionarles aquello que precisan. De esta forma no llegan a encontrar la relación entre lo que tienen o compran y su valor económico real.

En ocasiones muestran conductas con el dinero, de ahorro exagerado de todo lo que les dan, rozando en algún caso la "tacañería" o en el otro extremo, se muestran excesivamente "generosos" o despreocupados con sus gastos.

Esas conductas llegan a extenderse a la etapa adulta, incluso entre los trabajadores. En este caso es aún más necesario que se les permita administrar una parte del dinero que ganan, aunque también hagan su aportación a la economía familiar.

El dinero es, sin duda, uno de los principales elementos motivadores de un trabajador a la hora de realizar sus tareas. Cuando tienen un sueldo, es recomendable que dispongan de su propia cartilla o tarjeta de crédito y que tengan acceso a su dinero, aunque se establezcan limitaciones en cuanto a la cantidad definitiva, si no se les permite manejar dinero, no pueden llegar a saber el valor del mismo y siempre serán dependientes de otros a la hora de administrarlo.

Algunas medidas que pueden ir incorporando los padres a su vida cotidiana para ayudar a sus hijos con SD en el manejo del dinero pueden ser:

- *Hacerles responsables de una cierta cantidad de dinero desde que son pequeños, por ejemplo a través de una paga que ellos podrán utilizar y administrar de acuerdo con sus propios criterios.*
- *Proporcionarles una cartera o monedero en la que llevarán su propio dinero. Lógicamente, si los padres deciden proporcionarles cierta cantidad de dinero para que manejen, deberán permitirles que lo administren de acuerdo con sus intereses. No obstante, siempre es recomendable orientarles respecto a la mejor forma de hacerlo (por ejemplo, distribuyéndolo entre una cantidad para gastos y otra para ahorro).*
- *Permitirles comprar y pagar cuando sea algo para ellos. Pedirles su colaboración en recados o compras sencillas, haciéndoles ver lo que cuestan las cosas y el dinero que precisan para pagarlas.*
- *La calculadora puede ser un instrumento muy útil cuando se trate de manejar dinero en la vida cotidiana.*
- *Trabajar en casa el reconocimiento y discriminación de las distintas monedas y billetes.*
- *Relacionar cada uno de ellos con alguna compra cotidiana de su interés (Con un euro, ¿puedo comprar una coca-cola?; con 10 euros, ¿puedo comprar una entrada de cine, una película en DVD o un CD de música?).*
- *El manejo de su propio dinero les permitirá participar en actividades en las que el dinero es protagonista, por ejemplo, invitar a un conocido en una cafetería, hacer regalos a sus amigos en sus cumpleaños, participar en juegos como "el amigo invisible" o solidarizarse con otras personas, ayudando en campañas para recaudar fondos de ONGs.*

Cuando se inician programas de autonomía con desplazamientos independientes por la ciudad, es útil contar con cierta cantidad de dinero disponible siempre, para poder responder a imprevistos (coger un autobús o un taxi, llamar por teléfono, tomar un refresco, etc.).

Puede argumentarse que permitir que las personas con SD manejen dinero es algo que les va a introducir en la sociedad consumista en la que vivimos. Sin embargo, la máxima independencia de las personas con SD pasa por la consecución de los máximos niveles de autonomía en todos los campos y el manejo del dinero es uno fundamental en nuestra sociedad.

En todo caso, se ha de ser conscientes de que el nivel de autonomía que pueden llegar a alcanzar en este campo es limitado, ya que siempre necesitarán de alguien que les apoye cuando se trate de administrar cantidades elevadas, debido a las dificultades propias de su discapacidad intelectual."

Gracias a todo ello, pude observar que las mayores dificultades con las que se encuentran es tanto distinguir las diferentes monedas y billetes, conocer si pueden comprar un producto con una cantidad de monedas establecidas, es decir, si disponen de suficiente dinero para comprar dicho artículo, y el cálculo de "las vueltas" tras la compra de un producto cuyo importe sea menor que la cantidad de dinero entregada para su pago.

Por último, para el desarrollo de esta aplicación emplearé la herramienta de desarrollo Eclipse para poder programar en la plataforma de dispositivos móviles Android, por la cual me decanto debido a su política de distribución, gracias a la cual los dispositivos son más accesibles para las familias y futuros clientes debido a su precio y a su ecosistema de aplicaciones, Google Play. He elegido Android como plataforma, ya que la alternativa era iOS, debido a que para desarrollar para Android tiene menos restricciones, bien sean relacionadas con el Hardware (para programar y probar aplicaciones de iOS es necesario disponer de un dispositivo de Apple, mientras que para Android se puede emplear cualquier ordenador con eclipse que es distribuido sin coste alguno) como por el lenguaje de programación empleado, que en mi caso será JAVA, el cual he podido conocer y aprender a lo largo de la carrera. También ha influido en la decisión, la gran variedad de dispositivos con el sistema operativo Android en su interior, gracias a la cual, es más asequible económicamente un dispositivo Android que uno de Apple.

1.2 Objetivos

Los objetivos, propuestos por familiares, por el tutor del presente proyecto y por mí, que se quieren lograr a medio y largo plazo con esta aplicación son los siguientes:

- Conocimiento e identificación de las distintas monedas de euro.

- Aprendizaje del valor de las monedas para comprar un producto.

- Conocimiento de los distintos billetes de la moneda euro.

- Aprendizaje del valor de los billetes a la hora de comprar un artículo.

- Práctica de pago de precio exacto en una simulación de compra en una tienda.

- Ejercicio de cálculo para conocer el valor del dinero.

- Ejercitamiento visual de los cambios recibidos tras el pago de un producto.

Todos estos objetivos de aprendizaje se pretenden alcanzar con una aplicación eminentemente interactiva que permita extraer el máximo rendimiento de la capacidad de aprendizaje visual que poseen las personas con discapacidad intelectual.

Dada la baja capacidad de abstracción de las personas a las que está destinada esta aplicación, un objetivo de crucial importancia es ser lo más realista posible para tratar de minimizar esta carencia de capacidad de abstracción, por lo que los requisitos de la aplicación deberán tener en cuenta esta y otras limitaciones similares.

Los objetivos anteriormente descritos deben ser alcanzados sin obviar un objetivo primordial que es la usabilidad de la aplicación, que en este caso concreto se traduce en una gran simplicidad de uso, en el manejo de objetos de la vida cotidiana con un grado de abstracción muy bajo, en el empleo de elementos de grandes dimensiones y buena visibilidad, etc.

Un objetivo secundario que podría tenerse en cuenta en el diseño sería la posibilidad de personalización de la aplicación que permitiría que esta fuera más amigable. Esta capacidad permitiría que el usuario pudiera introducir en la aplicación (en el formato y forma que se establezca) nuevos elementos de uso cotidiano, que pudiera modificar los precios de los productos acorde a los valores reales de mercado, que pudiera personalizar los escenarios de trabajo, etc. Todo ello con el objetivo último

de mejorar la amigabilidad de la aplicación, y por ende, de mantener la atención del usuario para lograr un mejor y más rápido aprendizaje.

También puede tratarse de disponer de distintos niveles de abstracción, trabajando con detalladas ilustraciones que representen los objetos, monedas y billetes, fotos reales de estos elementos o ilustraciones con un menor grado de detalle. El usuario podría trabajar con unos u otros en función de su capacidad de abstracción, trabajando así no sólo la vertiente matemática, sino la capacidad de abstracción.

1.3 Estado del Arte

El sistema operativo Android, fue desarrollado inicialmente por una pequeña empresa llamada Android Inc, que en el año 2005 fue comprada por Google. A parte de este hecho, gracias a la ayuda de Google, Android comenzó a experimentar una evolución impresionante hasta la actualidad.

Desde que comenzó a implementarse en dispositivos móviles hasta la actualidad, se han sucedido gran número de versiones y actualizaciones. Esto provoca la denominada fragmentación, pero a su vez, permite que dispositivos más económicos, si nos centramos en el hardware que implementan, puedan disponer de Android.

Todas las versiones de Android han sido bautizadas por parte de Google y sus usuarios con nombres de postres, hecho que diferencia este sistema operativo del resto, que a pesar de ser un simple matiz o una sencilla anécdota, ha servido de publicidad y le ha otorgado una mayor sencillez para difundirlo que, si por el contrario, se hubieran referido a las mismas con los números de las respectivas versiones.

La primera vez que se pudo observar Android (Android 1.0 – Apple Pie) en funcionamiento fue en un evento el 22 de octubre de 2008 en Estados Unidos, corriendo en el interior de un G1 de T-mobile, cuyas características eran un procesador de 528MHz, 256MB de ROM y 192MB de RAM. Estas especificaciones muestran lo básico y lo poco exigente, en cuanto a recursos físicos se refiere, que son los dispositivos físicos necesarios para disponer de Android. Tras esta versión, Google implementó Android 1.1 – Banana Bread, que básicamente se empleó para corregir errores de la versión anterior. Desde estas versiones, se implementó la pantalla de notificación desplegable (característica de Android e imitada por la competencia posteriormente), Widgets en la pantalla de inicio, integración con Gmail (debido a que Android pertenece a Google), Android Market (que ha pasado a llamarse Google Play, y se trata del distribuidor oficial de las aplicación para Android) y con Android 1.1 se comenzó a implementar las actualizaciones denominadas “over the air” (actualizaciones vía WiFi o datos) que hasta ese momento ninguna otra plataforma lo estaba implementando.

Después, presentaron Android 1.5 Cupcake, cuya principal novedad fue la introducción del teclado virtual, y no uno físico. Además, Google permite a sus desarrolladores implementar sus propios teclados, otorgando grandes posibilidades de personalización, y que aún hoy en día, diferencia a Android de iOS, ya que este último no permite a sus usuarios estas personalizaciones. También, comenzó la distribución del SDK para desarrolladores, que permitió mejoras en los Widgets y el desarrollo de los mismos por programadores externos al sistema operativo, se implementaron mejoras en el portapapeles, captura y reproducción de vídeo, y los fabricantes comenzaron a implementar sus capas de personalización (las cuales hoy en día son un problema en Android debido a su demora en las actualizaciones de sus dispositivos) y por último, la posibilidad de administrar las cuentas personales de Youtube y Picasa y la sincronización entre cuentas.

Más adelante presentaron Android 1.6 – Donut. En esta distribución se incorporó soporte para redes CDMA, lo que ampliaba el mercado de clientes de Android a Estados Unidos y Asia. También se actualizó para que fuera compatible con las pantallas del

momento, al ser posible correr el sistema operativo en múltiples resoluciones de pantalla.

La siguiente versión que presentaron fue Android2.0/2.1 – Eclair gracias al cual se incorporaron Google Maps Navigation (empleado como sistema de navegación de automóviles en el móvil), apoyo de varias cuentas (necesario para el correo-electrónico y redes sociales) se integró el soporte de HTML5, fondos de pantalla animados, *speech-to-text* (los usuarios podrían dictar al teléfono y éste lo escribía en texto) y una nueva pantalla de bloqueo haciéndola más parecida a la de iOS.

A continuación apareció Android 2.2 – Froyo y con esta versión nuevas mejoras en su funcionamiento y la posibilidad de compartir datos con tecnología 3G (UMTS) entre dispositivos.

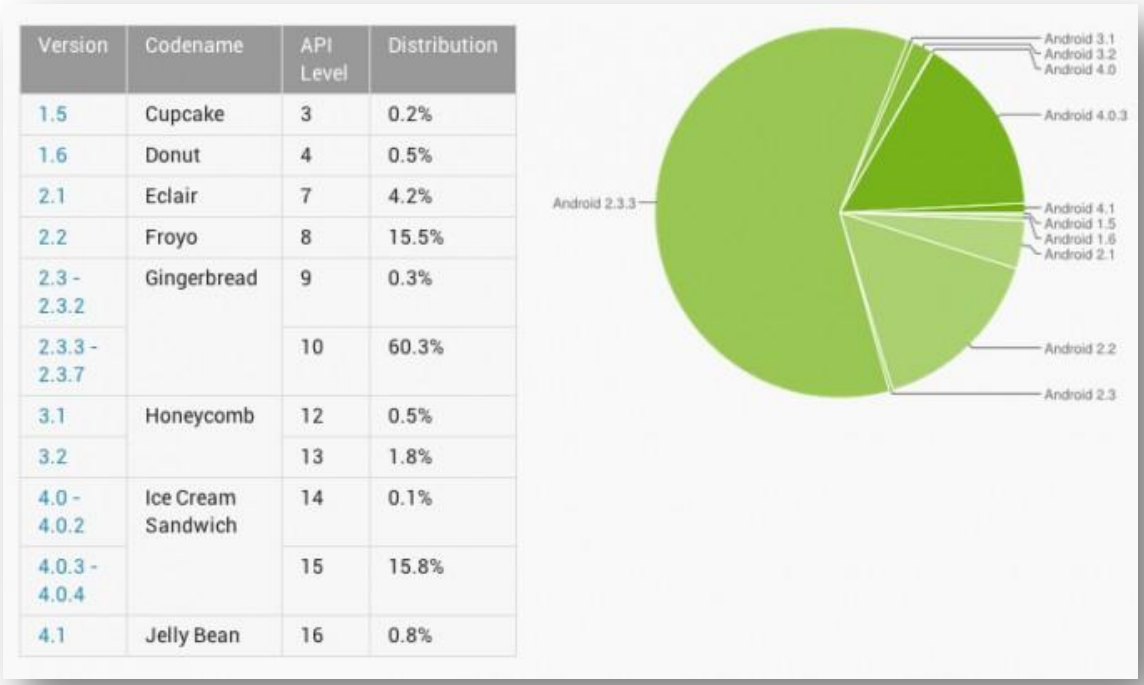
Más adelante, lanzaron Android 2.3 – Gingerbread y con ella se incorporaron mejoras como la maximización y aprovechamiento de la batería, soporte para cámara frontal y juegos en 3D. En la actualidad, esta versión es la más extendida como sistema operativo de dispositivos móviles. Actualmente cuenta con un 57.2% de la cuota de mercado android, y Froyo y Eclair, un 14% y 3.7% respectivamente [2]. Esto hace que la gran mayoría de aplicaciones sean compatibles para estas versiones en adelante.

Para tablets, Google realizó especialmente Android 3.0 – Honeycomb con la que eliminaron los botones físicos de sus dispositivos y mejoraron la multitarea. Esta versión no tuvo un gran éxito debido a que las tablets con Android se vieron eclipsadas por el iPad de Apple y por una rápida distribución de Android 4.0 – Ice Cream Sandwich.

La penúltima versión de este sistema operativo es Android 4.0 – Ice Cream Sandwich, que incorporó compatibilidad con NFC (Near Field Communication), desbloqueo facial, análisis de datos y muchas mejoras visuales más. Esto provocó un cambio radical en Android ya que a partir de esta versión será completamente compatible tanto para tablets como para móviles.

Por último, en septiembre de este año 2012, Google presentó Android 4.1 – Jelly Bean, que incorporó el soporte multicuenta y un aumento considerable de la fluidez del sistema operativo.

Estas últimas dos versiones son el futuro más cercano de Android, ya que poco a poco van ganando cuota de mercado (20.8% para ICS y 1.6% para JB). En la Figura 1 se puede apreciar un gráfico que recoge las cifras mencionadas anteriormente:



Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.2%
2.2	Froyo	8	15.5%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	60.3%
3.1	Honeycomb	12	0.5%
3.2		13	1.8%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	15.8%
4.1	Jelly Bean	16	0.8%

Figura 1, Gráfica de versiones de Android

Debido a que las versiones más antiguas de Android cada día se encuentran en menor número de dispositivos, he decidido desarrollar la aplicación para Android 2.2 en adelante porque creo que es una cuota de mercado que fácilmente empleará mi aplicación y que he considerado como la más acertada para trabajar.

Dentro del campo más didáctico, he buscado aplicaciones que persigan objetivos similares a los ya descritos. En Android he probado Kids Numbers and Math Lite [3] y Aprende a contar 123 Gratis [4], gracias a las cuales he podido observar mejor el funcionamiento de la enseñanza con dispositivos móviles. En comparación con el otro gran sistema operativo para dispositivos móviles, iOS, no he podido probar de manera práctica ninguna aplicación, por los motivos expuestos anteriormente (especialmente los económicos). Sin embargo, he comprobado dos aplicaciones desde la AppStore (lugar en el que se distribuyen las aplicaciones de Apple) como son Euro€: monedas y matemáticas [5] (ver Figura 1) y Monedas en la selva – aprende a calcular monedas (para iPad) [6]. Mi aplicación será distinta en diversos aspectos. Según las imágenes prácticas de estas aplicaciones, que tienen mayor relación con la temática de la mía, se puede observar cómo para pagar hay que seleccionar las monedas, por el contrario en la mía se deberán arrastrar las mismas para pagar un artículo. Esto lo desarrollaré con esta idea ya que, en mi opinión es más realista "entregar" el dinero para pagar que no simplemente seleccionarlo. El concepto de "dar y recibir" se trabaja frecuente y ampliamente con las personas con discapacidad intelectual.

Figura 1. Pantalla principal de Euro€: Monedas y matemáticas

Por otro lado, mi aplicación se centrará en la interacción que surge con el pago de un producto, es decir, no se quedará en el simple hecho del valor de las monedas, sino que les otorgará un valor práctico añadido al tratarse de un juego de simulación de compra de productos con precios lo más reales posibles, algo que no ocurre en las otras aplicaciones probadas. Véanse como ejemplo ilustrativo la Figura 2 y la Figura 3.

Figura 2. Pantalla del juego Euro€: monedas y matemáticas con el precio de un artículo, en este caso un trozo de tarta.

Figura 3. Una pantalla de juego de Monedas en la Selva.

También se intentará motivar y guiar al usuario constantemente mediante grabaciones de voz y animaciones dentro de la aplicación (dispondrá de tutorial para empezar a jugar, el dependiente se moverá al hablar, la pantalla de inicio dispondrá de movimiento, etc.).

Además, se premiarán los aciertos haciendo que el usuario disponga de más dinero para la compra de artículos en el último nivel.

Por último, se estructurará en temas, empezando por el precio exacto con monedas, después se tratará de saber si nos llega, a continuación los billetes y finalmente el nivel en el que disponemos de lo acumulado al completar los anteriores y podemos disponer de monedas y billetes. Esta estructura ha sido elegida en función de la dificultad que se ha observado para reconocer monedas y billetes, ya que no siempre se ve de manera clara la proporción que existe entre ellos.

Finalmente, mi aplicación será desarrollada para cualquier dispositivo Android a partir de la versión 2.2 y no para un solo dispositivo. Esto quiere decir que no importa la pantalla o resolución que tenga el terminal del usuario, ya que todas las imágenes serán

reajustadas en cada dispositivo. Gracias a esta estrategia, es posible solventar el mayor problema que podemos encontrar en Android, que es la fragmentación, y nos evitaremos tener que diseñar un gran número de versiones de la misma aplicación en función del dispositivo que la instale.

1.4 Solución propuesta

Una vez observados los problemas diarios que pueden tener estas personas con discapacidad, los cuales los he observado personalmente ya que trabajo como dependiente de una tienda el fin de semana, y gracias al auge tecnológico relacionado con los dispositivos móviles que estamos viviendo en la actualidad, se propone una aplicación para Android, tanto para Smartphone como tablet, a modo de juego, que permita conocer las monedas y billetes de euro.

Además, la solución propuesta puede servir para cualquier persona que esté empezando a conocer la moneda, aunque lógicamente superar los niveles supondrá una menor dificultad.

También se puede considerar un juego que simula una situación de compra realista sencilla, y que tiene como objetivo principal que sirva como aprendizaje y ayuda para situaciones reales.

Por otro lado, se deberá tener en cuenta problemas que pueden surgir con el desarrollo de la misma, debido a que la tecnología relacionada con los dispositivos móviles avanza realmente rápido, y encontramos muchas diferencias entre versiones. Por lo tanto, voy a implementar la aplicación con compatibilidad desde Android 2.2 hasta el actual Android 4.1 y de este modo lograr que mi aplicación tenga un mayor número de dispositivos compatibles, especialmente tablets más económicas que pueden contener Froyo en su interior hasta las más actuales con Ice Cream Sandwich o Jelly Bean.

2.- Análisis

En este capítulo se van a analizar los requisitos, tanto funcionales como no funcionales, y los casos de uso.

2.1 Análisis de requisitos

A continuación se muestran las funcionalidades que serán necesarias en la aplicación final. Estos requisitos se deberán tomar en cuenta para el diseño y la codificación de la aplicación. Se han definido mediante reuniones con el tutor, consultas con una familia con una hija con Síndrome de Down y por inventiva propia.

2.1.1 Requisitos funcionales

- La aplicación deberá guiar al usuario en los distintos niveles. Para ello se deberá emplear ayuda auditiva.

- La aplicación deberá mostrar cómo se juega al menos la primera vez que es lanzada.

- También se seguirá un orden en el aprendizaje, para ello se emplearán temas que se irán desbloqueando conforme se superen los anteriores.

- Además, la aplicación debe motivar al usuario para que siga jugando y aprendiendo. Por tanto, se deberá incentivar al jugador con una recompensa por superar los diversos temas.

- Al tratarse de un juego para niños, debe tener animaciones y movimiento.

- El sonido debe acompañar tanto los aciertos como los fallos.

- Tras cierto número de fallos, se deberá mostrar ayuda visual en pantalla, para que el nivel de interés del usuario por la aplicación no decaiga y desista por estar atascado en un nivel.

- Deberá simular la interacción con un trabajador de una tienda y realizar el pago de algún artículo.

- Tendrá que enseñar diversas monedas y billetes, por el lado común de todos ellos, independientemente del país.

- Deberá ayudar a que el usuario aprenda a saber si tiene suficiente dinero para pagar un artículo.

- La aplicación deberá enseñar la cantidad exacta necesaria para el pago de un artículo, por otro lado, también deberá mostrar los cambios recibidos tras el pago de un producto, si este no se ha pagado con la cantidad exacta.

- Será necesario que el juego muestre las monedas y billetes usados más comúnmente.

A continuación, se detallan una serie de requisitos, que sin ser imprescindibles, también se deben tener en cuenta a la hora de desarrollar la aplicación.

2.1.2 Requisitos no funcionales

- La aplicación debe ser lo más intuitiva posible, sin demasiadas opciones secundarias que puedan distraer al usuario.

- Por ser un juego para niños, deberá tener unos gráficos acordes para los mismos, en este caso, dibujos coloridos y simples, sin grandes efectos pero con dotes de dinamismo.

- Para los tutoriales y guías se intentará disponer de una voz femenina, ya que estadísticamente, son más claras y eficaces para las explicaciones.

  Esto se debe a la influencia de un artículo de la revista Icono [7], en el que se puede leer "*Las voces medias son las más comunes, y a las que estamos más habituados. Dentro de esta categoría podemos encontrar voces medias-agudas, voces medias y voces medias-graves. Este tipo de voces tiene la ventaja de poderse modular dentro de una gama más amplia de tonos que si se tratase de voces agudas o de voces graves, lo que da lugar a una mayor variedad. Son más ricas, tonalmente hablando. Esto hace que la escucha del mensaje, por parte del oyente, sea más agradable, más amena y, por lo tanto, se preste más atención al contenido de la información.*

  Sin embargo, y a pesar de que las voces graves son las más aceptadas, las más creíbles y las que más confianza dan, en el medio radiofónico son las voces medias, sobre todo si se trata de voces femeninas, las que parecen ser bastante aceptadas y dignas de credibilidad.

  En la credibilidad de la voz, no sólo hay que tener en cuenta el tipo de voz que se tenga o el dominio de las distintas técnicas vocales, sino también otro factor muy importante, que es la actitud que tenga el hablante a la hora de dirigir su discurso."

- El usuario debe tener opción de volver a empezar todos los temas en orden y bloquearlos si lo considera necesario.

- El código fuente deberá ser claro y estar comentado, para futuras modificaciones y mejoras.

- La aplicación deberá visualizarse correctamente en pantallas de cualquier tamaño, otorgándole así una mayor cuota de mercado en la que emplearla.

- Todo el diseño se intentará realizar de manera dinámica y reajustándose tanto a las versiones del sistema operativo, desde Android 2.2 – Froyo hasta Android 4.1 – Jelly Bean.

- Se emplearán todas las mejoras de las últimas versiones posibles, siempre que siga siendo compatible con el margen de versiones establecido.

- Todas las imágenes y recursos empleados, serán creados de la manera más dinámica posible para disminuir el consumo de recursos físicos del terminal, siempre que no afecten a la fluidez y dinamismo del juego.

- Por último, se intentará que toda la interfaz gráfica tenga relación con las monedas y billetes. Generalmente se tratará de establecimientos, centros comerciales y artículos disponibles para comprar.

- Como funcionalidad complementaria, se intentará otorgar al usuario la capacidad de captar sus propias imágenes de los productos y asignarles su precio correspondiente para lograr mayor efecto educativo al personalizar la enseñanza con objetos con los que el usuario objetivo se sienta más familiarizado.

2.2 Análisis de los casos de uso

Ahora vamos a mostrar los casos de uso:

Para representar los casos de uso se ha detallado un usuario y diversos niveles, que representan las diversas partes de la aplicación. Se ha empleado la herramienta Rational Rose que ha sido de gran ayuda, como se vio en clase, para representar los casos de uso.

En primer lugar tenemos la pantalla principal:

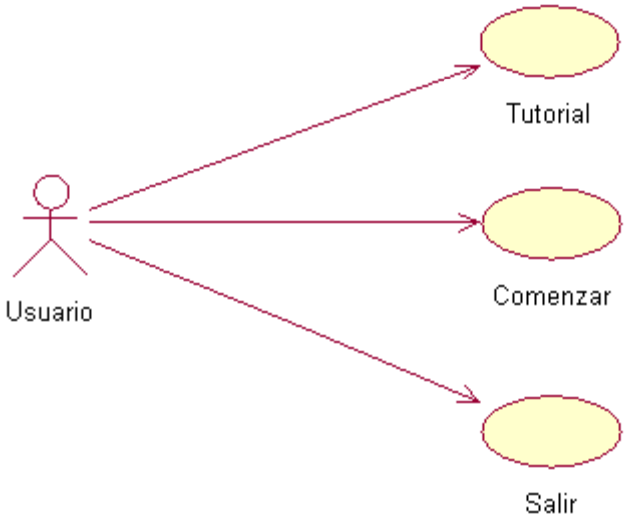


Figura 2.1: casos de uso, pantalla principal

Casos de uso: Pantalla principal

Descripción	Pantalla principal
Antecedentes	Ninguno
Opciones disponibles	Tutorial (El usuario puede seleccionar ver el tutorial o no hacerlo)
	Comenzar (Si el usuario lo selecciona pasará a la pantalla de selección de temas)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice)

A continuación tenemos el Tutorial:

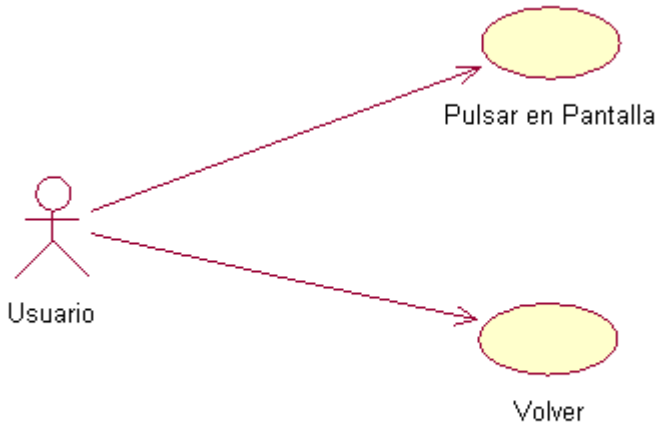


Figura 2.2: casos de uso, tutorial

Casos de uso: Tutorial

Descripción	Pantalla Tutorial
Antecedentes	Haber seleccionado Tutorial
Opciones disponibles	Pulsar en pantalla (Cuando finaliza la animación, el usuario puede pulsar en pantalla para acceder a los temas y comenzar a jugar)
	Volver (El usuario puede volver a la pantalla principal en cualquier momento de la animación. Esto supone la finalización del hilo de ejecución que se encarga de la animación y el sonido)

Ahora mostramos la selección de temas:

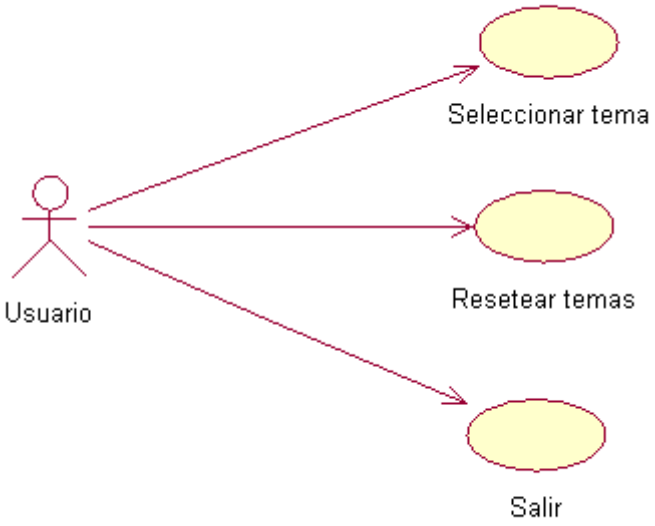


Figura 2.3: casos de uso, selección de temas

Casos de uso: Selección de temas

Descripción	Pantalla de selección de temas
Antecedentes	Haber seleccionado Comenzar
Opciones disponibles	Seleccionar tema (El usuario podrá seleccionar cualquier tema que se encuentre desbloqueado, lo que supondrá la iniciación del juego correspondiente)
	Resetear temas (El usuario podrá seleccionar si desea bloquear los temas con la finalidad de volver a superarlos en orden. Esto ayudará a su aprendizaje porque volverá a realizarlas en orden y con la meta de volver a superarlos todos)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento)

Vamos a ver los casos de uso del tema 1:

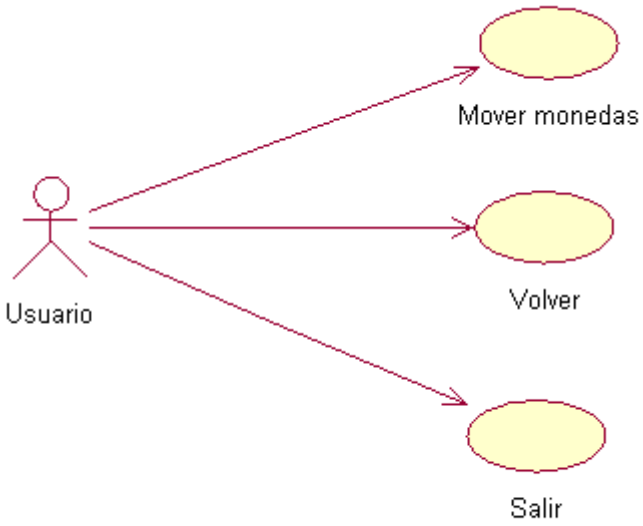


Figura 2.4: casos de uso, tema 1

Casos de uso: Tema 1

Descripción	Jugando al tema 1
Antecedentes	Haber seleccionado Tema 1
Opciones disponibles	Mover monedas (El usuario tendrá total libertad de mover las monedas por toda la pantalla, aunque para pagar deberá arrastrarlas hasta la caja registradora)
	Volver (El usuario siempre puede volver a la pantalla de selección de nivel. En caso de hacerlo perderá el numero de aciertos acumulados en dicho nivel hasta el momento)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento. También se guardará el objeto y el número de aciertos.)

A continuación, los casos de uso del tema 2 y 4:

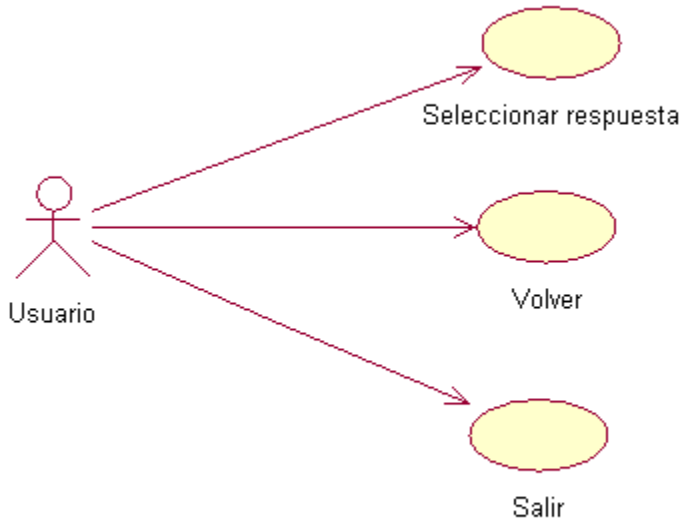


Figura 2.5: casos de uso, tema 2 y 4

Casos de uso: Tema 2 y Tema 4

Descripción	Jugando al tema 2 o al tema 4
Antecedentes	Haber seleccionado Tema 2 o Tema 4
Opciones disponibles	Seleccionar respuesta (El usuario podrá seleccionar entre "si" y "no" para resolver la pregunta. En caso afirmativo se acumularán los aciertos y sonará una melodía alegre, en caso de error no)
	Volver (El usuario siempre puede volver a la pantalla de selección de nivel. En caso de hacerlo perderá el numero de aciertos acumulados en dicho nivel hasta el momento)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento. También se guardará el objeto y el número de aciertos.)

Después analizamos los casos de uso del tema 3:

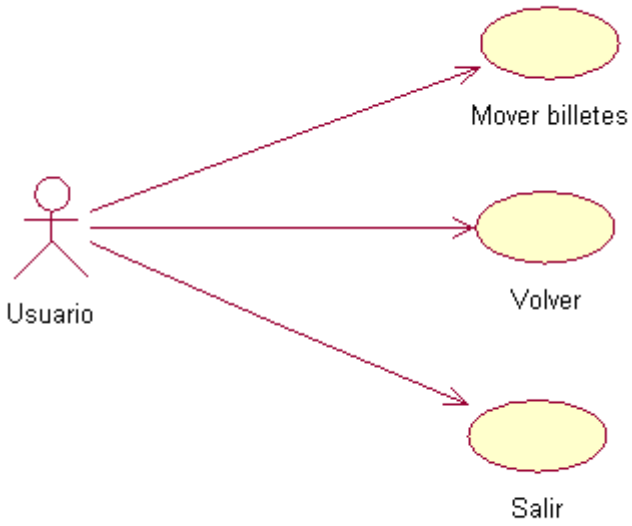


Figura 2.6: casos de uso, tema 3

Casos de uso: Tema 3

Descripción	Jugando al tema 3
Antecedentes	Haber seleccionado Tema 3
Opciones disponibles	Mover billetes (El usuario tendrá total libertad de mover los billetes por toda la pantalla, aunque para pagar deberá arrastrarlos hasta la caja registradora)
	Volver (El usuario siempre puede volver a la pantalla de selección de nivel. En caso de hacerlo perderá el numero de aciertos acumulados en dicho nivel hasta el momento)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento. También se guardará el objeto y el número de aciertos.)

Finalmente, analizamos los casos de uso del tema 5:

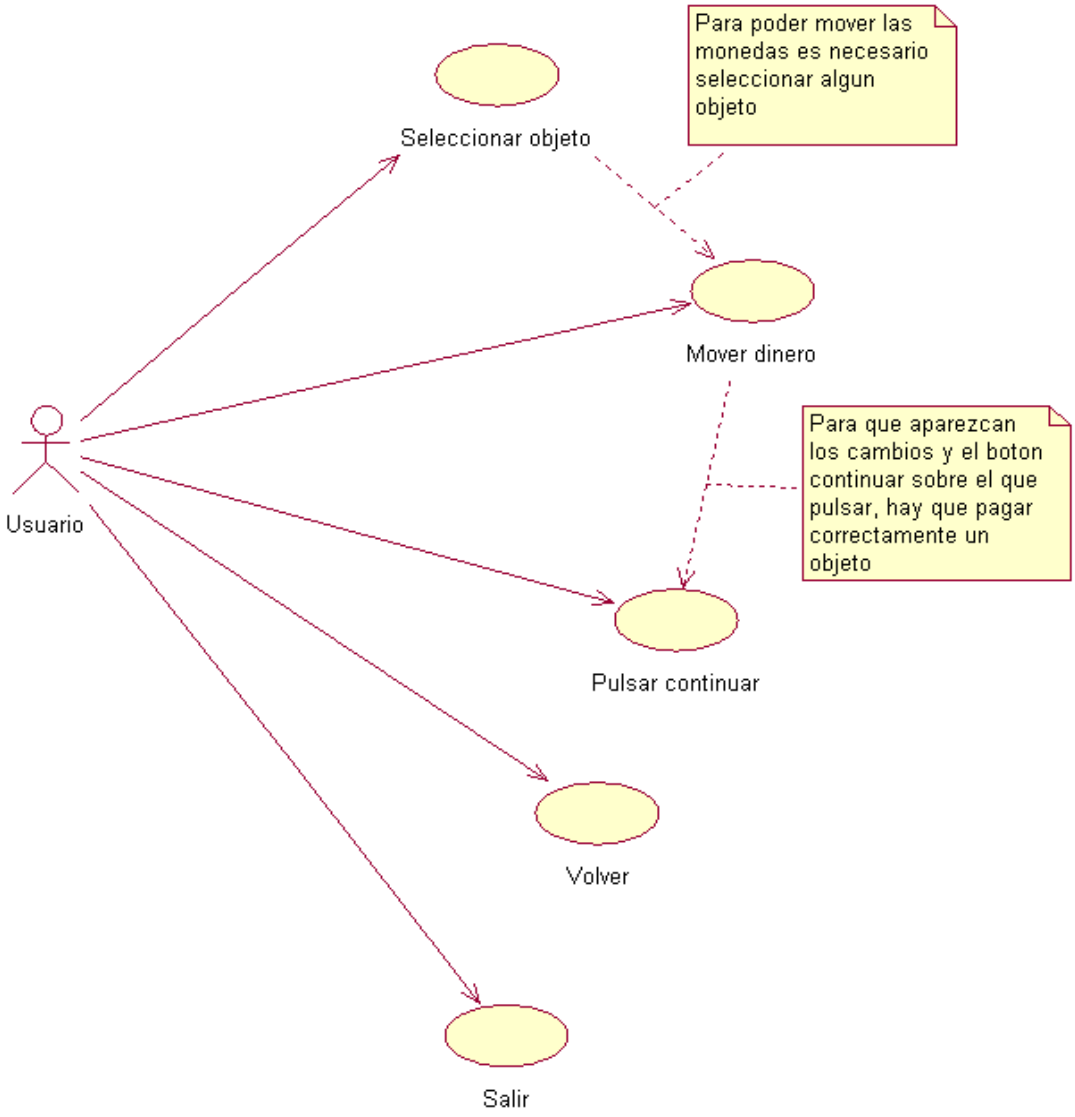


Figura 2.7: casos de uso, tema 5

Casos de uso: Tema 5

Descripción	Jugando al tema 5
Antecedentes	Haber seleccionado el tema 5
Opciones disponibles	Seleccionar objeto (El usuario podrá elegir cualquier artículo que aparezca en pantalla con la intención de comprarlo o de ver su precio)
	Mover dinero (El usuario podrá mover el dinero, tanto monedas como billetes, por toda la pantalla. Sin embargo, para poder realizar dicho movimiento, deberá haber seleccionado previamente un objeto.)
	Pulsar continuar (El botón continuar aparecerá únicamente si el usuario ha introducido una cantidad de dinero mayor o igual

Opciones disponibles	que la del artículo a comprar, en cuyo caso se generará una pantalla con los cambios y la posibilidad de pulsar en continuar para proseguir con el juego.)
	Volver (El usuario siempre puede volver a la pantalla de selección de nivel. En caso de hacerlo perderá el numero de aciertos acumulados en dicho nivel hasta el momento)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento. También se guardará el objeto y el número de aciertos.)

Por último, analizamos los casos de uso de la pantalla de premio:

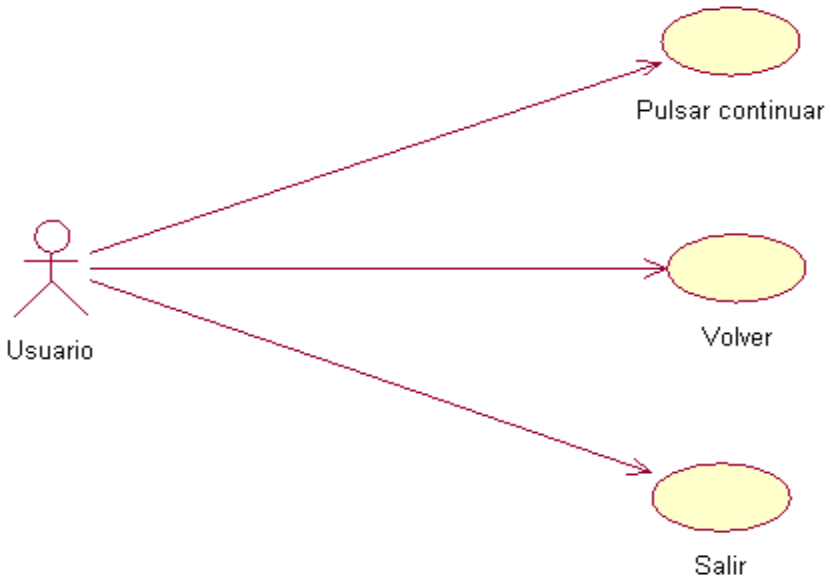


Figura 2.8: casos de uso, pantalla de premio

Casos de uso: Pantalla de premio

Descripción	Pantalla de premio
Antecedentes	Haber acertado un número determinado de veces en los diferentes niveles de la aplicación (temas del 1 al 4)
Opciones disponibles	Pulsar continuar (Cuando se finalice la animación se mostrará al usuario el botón de continuar, que le devolverá al juego en el que se encontraba)
	Volver (El usuario siempre puede volver a la pantalla de selección de nivel. En caso de hacerlo, se guardarán las ganancias hasta ese momento)
	Salir (Puede seleccionar salir de la aplicación, haciendo que su hilo de ejecución interno finalice, pero además, se guardará el estado de los niveles y la cantidad ganada hasta ese momento)

3.- Diseño

3.1 Metodología

Lo primero que debemos hacer es analizar la metodología seguida para enfocar el proyecto.

He empleado un modelo evolutivo, debido a que permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final. Dentro de este tipo encontramos más concretamente el iterativo incremental.

Este modelo se basa en la idea de desarrollar una implementación inicial, exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasta que se desarrolla un sistema adecuado.

Las actividades de especificación, desarrollo y validación se entrelazan, en vez de separarse, con una rápida retroalimentación entre éstas.

Ahora mostramos un esquema sencillo de la metodología empleada, Figura 1

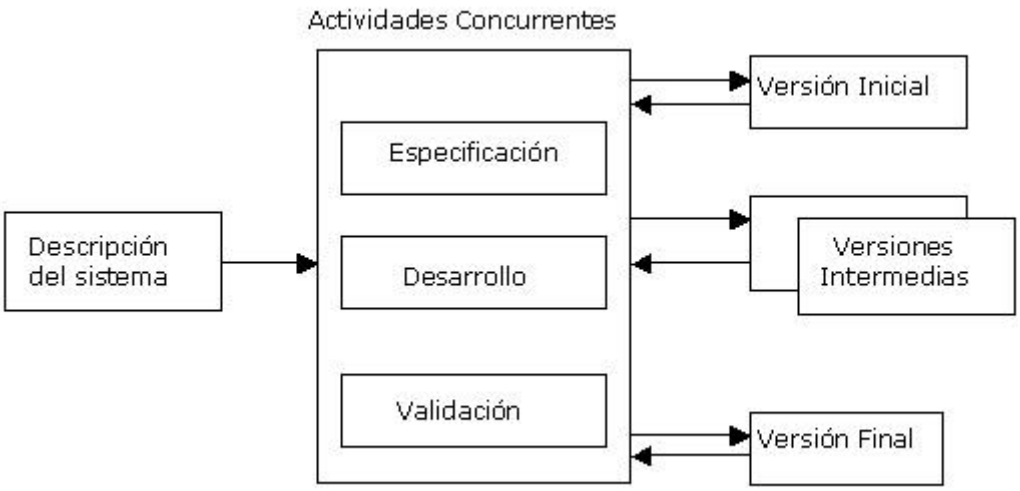


Figura 1

El modelo incremental es un modelo de tipo evolutivo que permite actividades de desarrollo realizadas en paralelo o concurrentemente con otros diseños o desarrollos de otras actividades. Además con este modelo se entrega software por partes funcionales pequeñas pero reutilizables, ya que cada modificación y mejora, y en ocasiones nuevos temas, se trabajan sobre las anteriores ya probadas.

Se emplean secuencias en cascada en forma escalonada a lo largo de toda la duración del proyecto.

El cliente valora ese trabajo inicial y puede aportar ideas al plan para el desarrollo de las siguientes versiones, teniendo en todo momento un producto parcial pero completamente operacional en cada una de las versiones.

Por último, este enfoque resulta muy útil cuando se dispone de baja dotación de personal, en este caso una única persona, para el desarrollo. También si no hay clara una fecha límite del proyecto por lo que se entregan versiones incompletas pero que proporcionan al usuario funcionalidad básica y cada vez mayor.

Al realizar el proyecto con esta metodología en mente, se han entregado y probado diversas versiones iniciales, como mostraremos más adelante, en las cuales se han corregido los errores detectados. Cada versión emitida incorpora a los anteriores incrementos las funcionalidades y requisitos que fueron analizados.

Gracias al desarrollo conjunto de la especificación y el desarrollo, se han incorporado niveles, modificado los ya existentes y, como en el caso del último nivel propuesto al inicio y que posteriormente fue descartado, la eliminación y sustitución de los niveles que no cumplían con los requisitos establecidos.

Dado que un requisito fundamental es que la aplicación sea intuitiva y sencilla, este modelo es el que mejor se adapta porque podemos comprobar con cada versión si el usuario al que está destinada, niños con Síndrome de Down, encuentra que la aplicación cumple con los requisitos establecidos.

Esta metodología también nos permite continuar desarrollando la aplicación en un futuro, debido a que se puede seguir consultando al cliente qué funcionalidades le gustaría incorporar en la aplicación, así como qué aspectos mejoraría o cambiaría.

Con este esquema se ha podido desarrollar la aplicación independientemente de la fecha de entrega de proyecto ya que, como se trata de un proyecto de fin de carrera que consta de una fecha límite de entrega, se ha podido desarrollar paso a paso, cumpliendo objetivos a corto y medio plazo, disponiendo en todo momento de una aplicación funcional que ha sido probada en cortos espacios de tiempo.

Como se dijo anteriormente, con este modelo se permiten y esperan probables cambios en los requisitos en tiempo de desarrollo y se admite cierto margen para que el software pueda evolucionar, y es aplicable cuando los requisitos son medianamente conocidos pero no completamente estáticos y definidos. En nuestro caso comenzamos planteando una aplicación para ayuda genérica para gente con Síndrome de Down y finalmente nos hemos centrado en enfocarla al aspecto de la enseñanza del dinero.

También al tratarse de un juego, se puede considerar desde la etapa de análisis, que se posee de áreas a tratar bien definidas y diferenciadas entre sí que pueden ser desarrolladas de manera independiente.

En conclusión, considero que es una metodología apropiada para enfocar un proyecto final de carrera, especialmente de una ingeniería técnica, porque se debe realizar de manera individual o en grupos reducidos de personas, se deben cumplir el mayor número de requisitos previos y es más sencillo llevar un control de todos los

progresos que se van logrando, todos ellos analizados en cada una de las pruebas que se pueden realizar a lo largo del ciclo de vida del proyecto.

También matizar que gracias a un paradigma incremental se reduce el tiempo de desarrollo inicial, ya que se implementa una funcionalidad parcial. En el caso de este proyecto, desarrollado en un corto periodo de tiempo, este aspecto ha sido fundamental para su correcto desarrollo y entrega dentro de los plazos fijados.

3.2 Interfaz gráfica

3.2.1 Primera fase

A continuación, se presentan una serie de esquemas y bocetos empleados para realizar la primera aproximación del proyecto. En esta fase se puede apreciar la simplicidad de las imágenes y la falta de experiencia adquirida a la hora de implementarlas. También se puede observar que no se redimensionan las imágenes, ya que todo está creado de manera estática en la pantalla. Al tratarse de una primera aproximación, ya se puede probar el movimiento de las monedas por la pantalla, hecho que es muy importante y distintivo de la aplicación. También la portada es sencilla, ya que se emplea únicamente para acceder a las partes de la aplicación que son importantes, el tutorial, los temas y salir. En esta etapa, el trabajo con programas de edición de imagen y audio (como Photoshop y Audacity) ha sido muy escasa debido a que se buscaba más la funcionalidad y eficacia que el cuidado visual.

Esto dio lugar a las pantallas de la Figura 4.

Figura 4. Pantallas correspondientes al tutorial (izquierda) y al primer tema (derecha).

En la Figura 4 (izda.), se puede apreciar que para comprar un objeto se debe arrastrar la moneda hasta la caja registradora. Todo el tutorial está acompañado de audio que indica cómo se deben mover las monedas para pagar el precio del producto. Por otro lado, la Figura 4 (dcha.),, hace referencia al primer tema. Esta imagen en verdad es la segunda versión del primer tema, ya que en la anterior las monedas estaban representadas en línea, en vez de en dos filas de tres monedas cada una, y esto hacia muy difícil seleccionarlas debido a que para que entrasen en pantallas pequeñas, se debía reducir su tamaño considerablemente haciendo que pulsar en el centro de la moneda para moverla fuera muy complicado.
Este fallo fue descubierto al probar la aplicación por un familiar, posible usuario de la aplicación, ya que en la primera prueba le costó especialmente seleccionar la moneda de cinco céntimos.

Por otro lado, la pantalla inicial de la aplicación, distaba mucho de la considerada como final. En dicha pantalla simplemente aparecen tres botones que sirven para iniciar el tutorial, comenzar los temas o salir de la aplicación. La Figura 5 corresponde con la situación inicial de la pantalla principal.

Figura 5. Pantalla inicial.

En este primer prototipo de la aplicación, los temas también tenían una imagen diferente. En primer lugar, solo se disponían de mover las monedas y billetes para pagar, y la idea del último nivel acabará siendo desechada y sustituida por un nivel a modo de máquina expendedora en la cual los usuarios pueden practicar eligiendo un producto y obteniendo las vueltas, pero esto lo veremos más adelante. A continuación se muestra la Figura 6, que ilustra la pantalla de los temas.

Figura 6. Imagen de los temas.

Los temas siguen el orden lógico ya mencionado anteriormente, y es una estructura que se mantendrá hasta la aplicación final, aunque como veremos, faltan niveles que estarán en mitad de los disponibles en esta versión. También se aprecia que falta un fondo que le otorgue un toque animado y distintivo, así como la posibilidad de desplazar los temas de arriba abajo o viceversa. También falta el título donde se indicará que se tiene que seleccionar un tema.

Por último, el nivel final "Vamos de compras", la idea era simular una tienda y comprar los productos, que finalmente ha sido descartado debido a que su diseño dio lugar a un nivel poco intuitivo, incumpliendo un requisito fundamental para la aplicación

El boceto del ultimo nivel corresponde a la Figura 3.5, en la cual vemos cómo la idea era tener dos objetos y, con el dinero que se tuviera en cada momento, comprar el que fuera posible; en caso de no disponer de más dinero salir. Esta idea llegó a ser implementada, pero finalmente se descartará y será sustituida por otra como veremos más adelante.

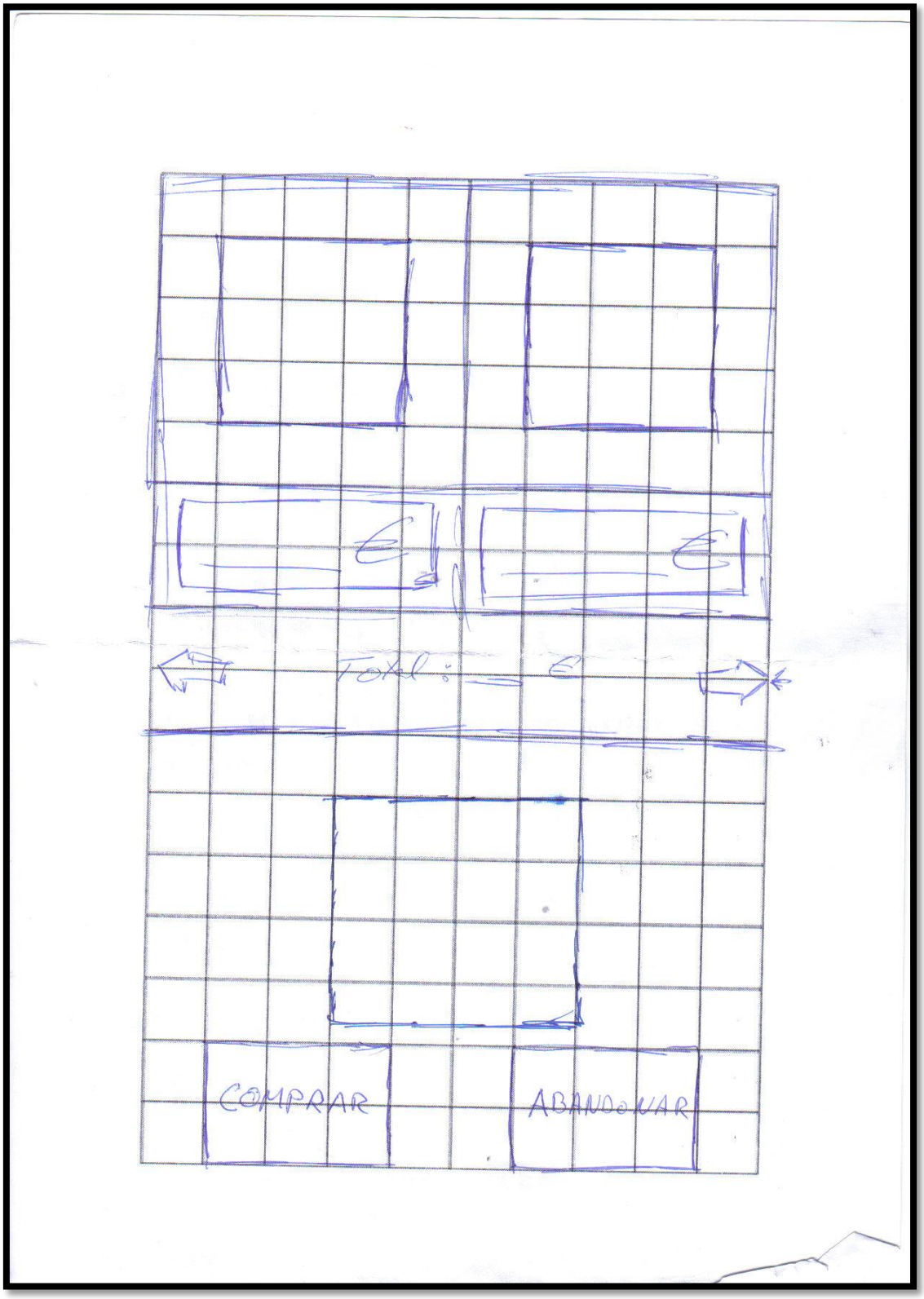


Figura 3.5

3.2.2 Segunda fase

Para esta segunda fase, ya se ha probado la aplicación de modo que se han corregido diversos fallos. También se han incorporado niveles adicionales, que se han considerado necesarios en el desarrollo del aprendizaje. La estructura a partir de este punto comienza a parecerse a la estructura final, sin embargo visualmente aún sufrirá diversos cambios, así como algún cambio en los niveles.

A partir de esta fase, se comienza a realizar el diseño adaptando toda la aplicación a diversas pantallas. Aunque al principio es más costoso de realizar, permite disponer de un mayor abanico de terminales en los que probar la aplicación.

En esta fase de desarrollo se va a empezar a emplear código específico de canvas, del cual hablaremos más adelante en la implementación.

A partir de este punto, se mostrarán esquemas realizados a mano sobre una cuadrícula. Dicha cuadricula representa las dimensiones de la pantalla, y mediante ajustes en las imágenes acordes a las coordenadas en la cuadricula, se reajustan en función de la pantalla. Esta metodología se explica paso a paso con los ejemplos y los esquemas realizados para el proyecto.

Comenzamos con el desarrollo de la pantalla del primer tema:

1. Diseñamos una cuadrícula 10x16, que representará el ancho x alto de la pantalla.

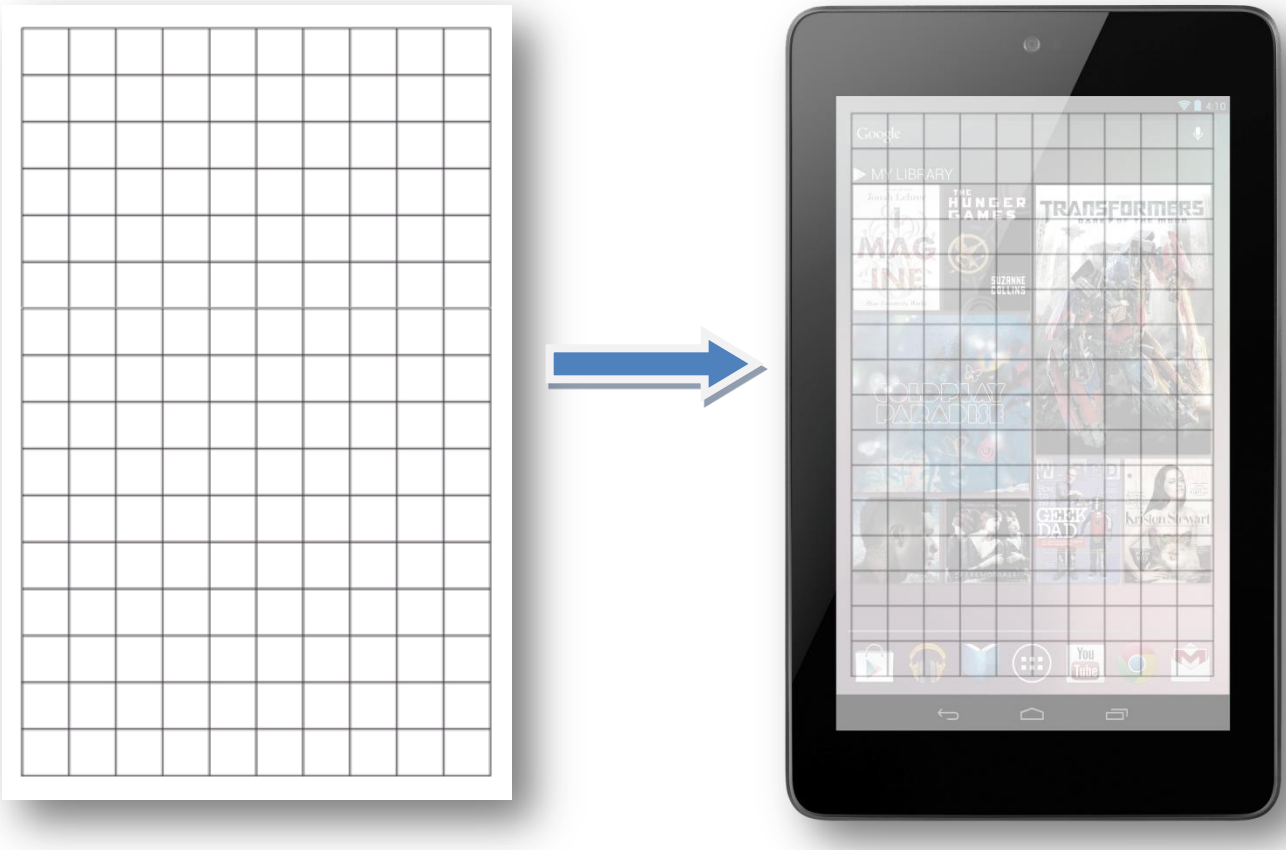


Figura 3.6

2. Una vez disponemos de dicha cuadrícula la numeramos para facilitarnos el trabajo de situar en los puntos las imágenes (Figura 3.7).

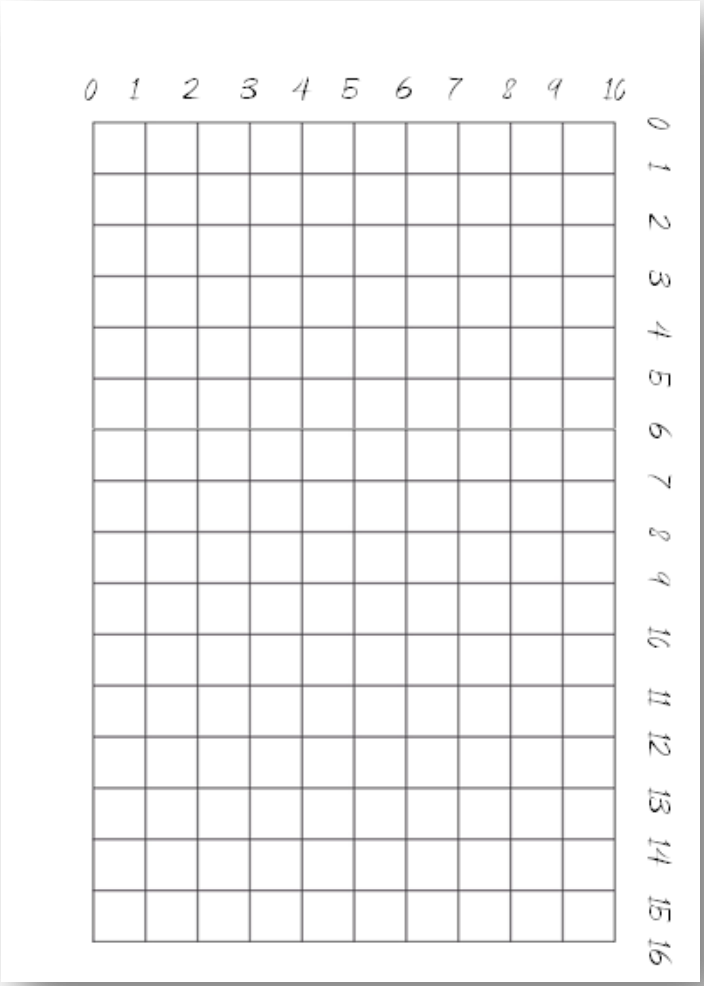


Figura 3.7

3. Con estas medidas podremos situar fácilmente en pantalla cualquier imagen. Para esta fase utilizaba el punto (i,j) como inicio. Más adelante comencé a diseñar mediante dos puntos de la cuadricula, es decir, el punto1(i,j) y el punto2(i+x,j+y), con lo que diseño un cuadrado en el cual ajusto la imagen, tal y como se observa en la Figura 3.8.

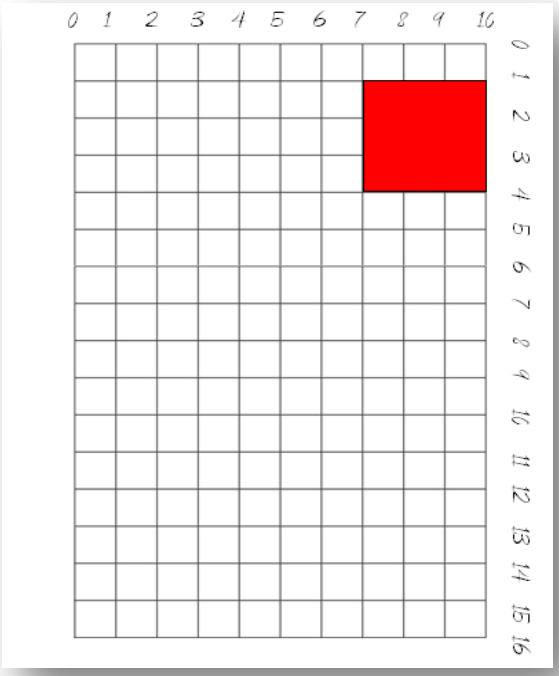


Figura 3.8

En este ejemplo, vamos a situar la caja registradora del primer nivel dentro del cuadrado rojo, como se muestra en la imagen anterior (Figura 3.8).

Esta es la imagen de la caja, diseñada con una buena resolución (500x500). Gracias al ajuste que estamos mostrando, se verá correctamente y sin pixelar, tanto en pantallas con una resolución media (800x480) hasta los de alta resolución (1280x800), quedando siempre en la misma sección de la pantalla y permitiendo un uso sencillo de la misma.

Figura 3.9

A continuación, se ajusta la imagen para que se sitúe dentro del área formada desde la posición *canvas.getWidth()*7/10,canvas.getHeigth()*1/16*, que representa el punto1, hasta *canvas.getWidth(),canvas.getHeigth()*4/16*, en cuyo caso corresponde con el punto2.

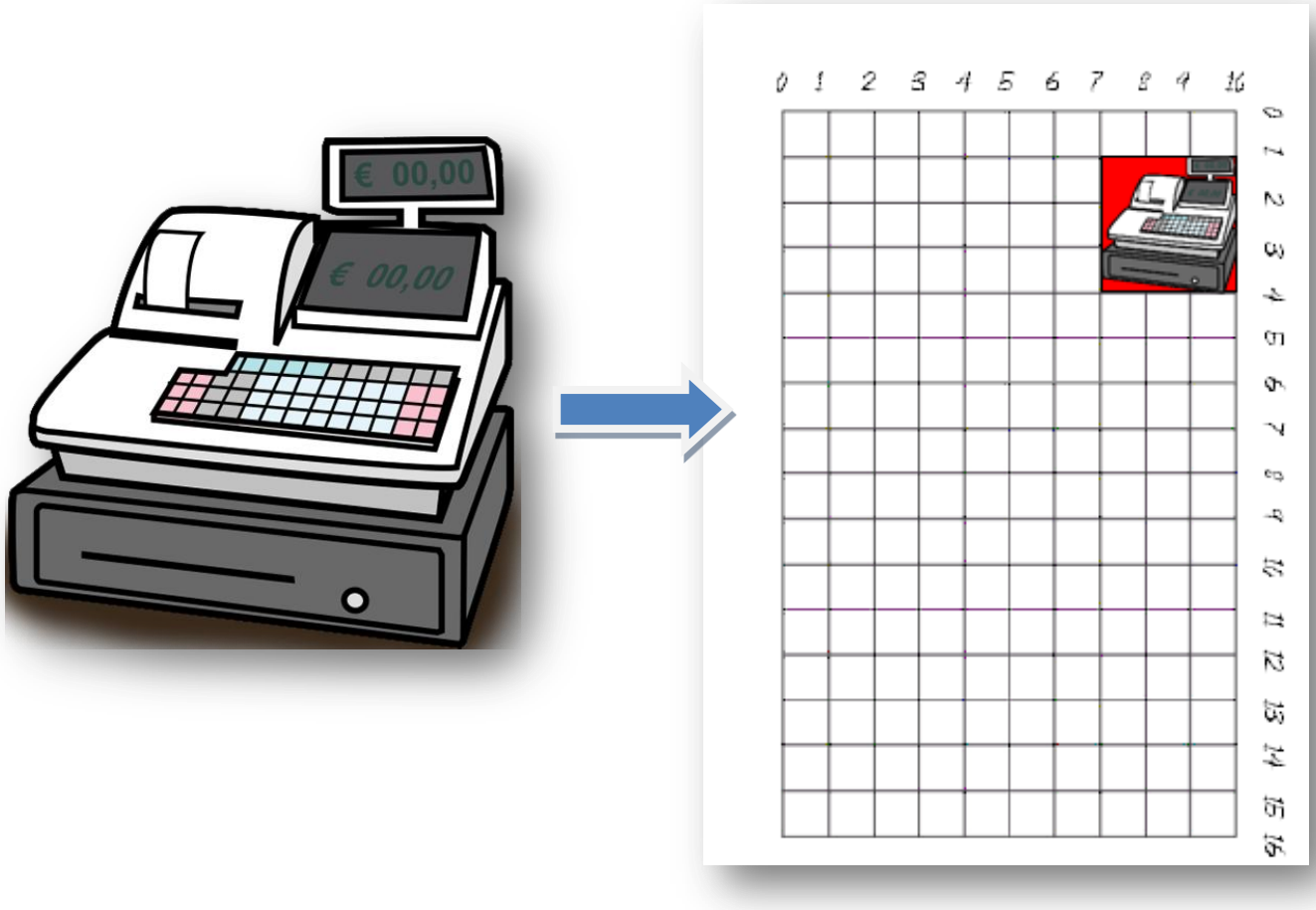


Figura 3.9

En la fase de implementación, explicaré el código necesario más específicamente, pero empleamos esta idea para el desarrollo gráfico de todos los niveles, teniendo siempre en mente la posibilidad de emplear la aplicación tanto en teléfonos móviles como tablets con Android (ver Figura 3.9).

Ahora voy a mostrar un esbozo a mano, realizado para el cálculo de las posiciones de los objetos en el primer nivel. También fue necesario ajustar las monedas, en cuyo caso el proceso de dibujo y tratamiento de las imágenes fue diferente y lo explicaré a continuación.

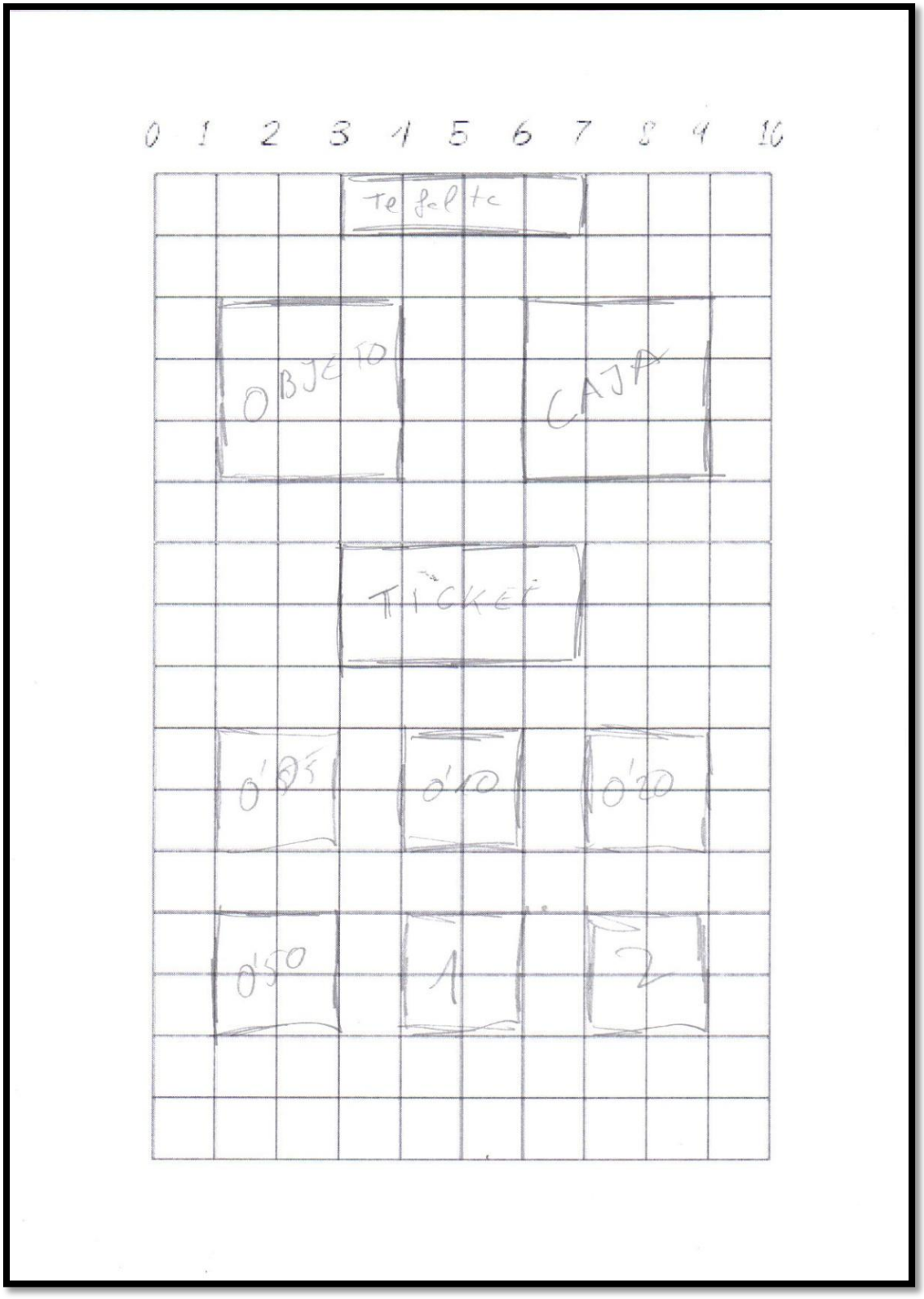


Figura 3.10

En esta imagen (Figura 3.10) se puede apreciar la localización del objeto (el cuadrado superior a la izquierda), la situación de la caja (cuadrado superior derecho), la posición del ticket en donde se muestra el precio, y la primera prueba para situar las monedas de uno y dos euros.

Esta primera aproximación da lugar a las siguientes pantallas (Figura 3.11, 3.12 y 3.13) para los temas de monedas y el de los billetes, el cual sigue la misma estructura y el mismo esquema:

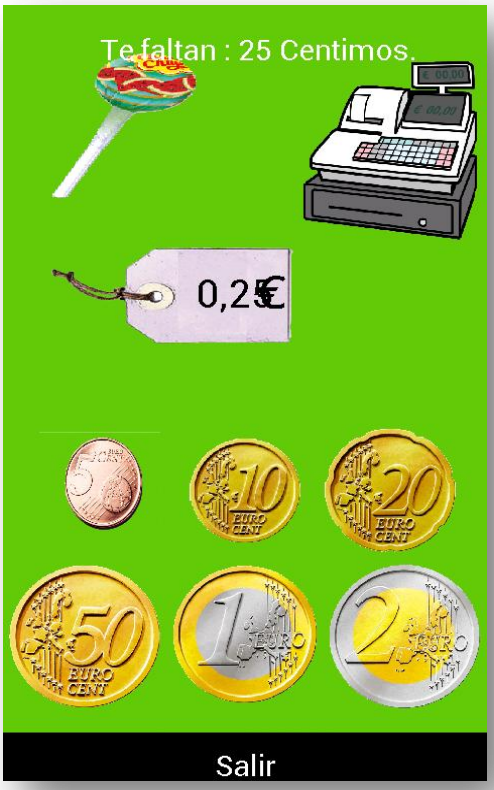


Figura 3.11

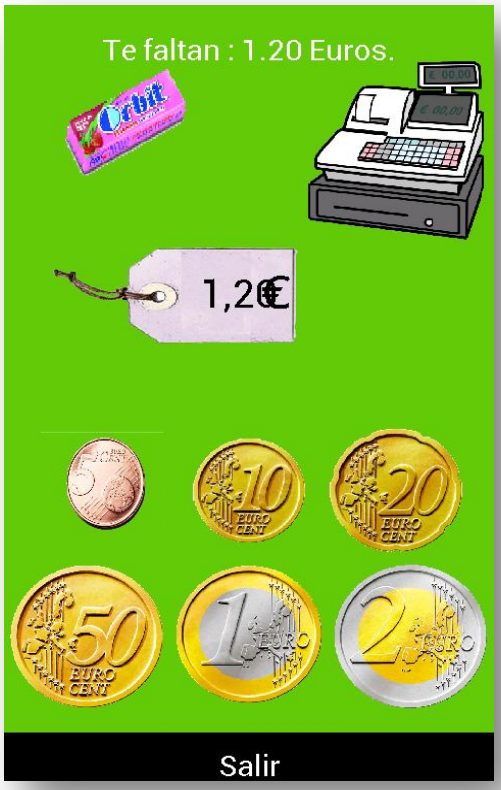


Figura 3.12

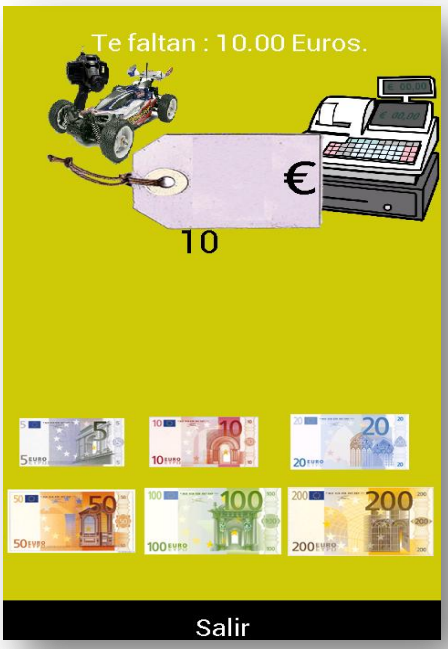


Figura 3.13

La imagen de la izquierda (Figura 3.13), corresponde a una primera aproximación del tema de los billetes. Se puede apreciar la mala localización del precio y del ticket, sin embargo, se aprecia claramente el objeto a comprar, la caja y los billetes, que suponía la meta para esta fase.

A continuación, vamos a ver el resultado de aplicar el esbozo del último nivel, el cual estaba sin implementar a estas alturas de proyecto, obteniendo la imagen correspondiente a la Figura 3.14.

Figura 3.14

En esta imagen se puede apreciar que el objetivo era comprar uno o varios productos, desplazarse a izquierda y derecha y cuando se finalizase la comprar pulsar en listo. Este nivel será descartado debido a que en las pruebas se comprobó que no era intuitivo para los usuarios, que no quedaba clara su finalidad y que se podían agotar los objetos sin haber gastado la totalidad del dinero disponible. Por tanto, como no cumplía con el principal objetivo, el de la enseñanza, y no estaba claro su diseño del todo, acabé por descartarlo.

3.2.3 Tercera fase

En esta fase he observado detenidamente otras aplicaciones disponibles para la enseñanza, concretamente las mencionadas en el estado del arte, que eran *Kids Numbers and Math Lite* y *Aprende a contar 123 Gratis*. Estas aplicaciones difieren totalmente de la mía en esta fase, debido a que me falta añadirle fondos más coloridos y relacionados con mi aplicación. Desde un principio se ha tenido la idea de que las pantallas de juego no resulten cargadas de dibujos que puedan distraer al usuario final, sin embargo he considerado que un fondo liso y de un solo color hace que la aplicación parezca un prototipo constantemente.

Este es el motivo por el cual comencé el diseño de los fondos del juego de la siguiente manera.

1- En primer lugar para simular una tienda es necesario un dependiente. En mi caso he tomado la figura de un pastelero (Figura 3.15.) y lo he retocado y pintado (Figura 3.16).

Figura 3.15

Figura 3.16

2- A continuación se ha diseñado una mesa en la cual apoyar la caja registradora el producto de venta y poner detrás al vendedor. Dicha mesa ha sido diseñada completamente en 3dmax y exportada para emplearla en la aplicación. La mesa corresponde a la Figura 3.17.

Figura 3.17

3- Ahora muestro el boceto (Figura 3.18) para ajustar todas las nuevas imágenes dentro de la pantalla. En este caso están de manera estática, y para la siguiente fase comenzaremos a animar todas las imágenes.

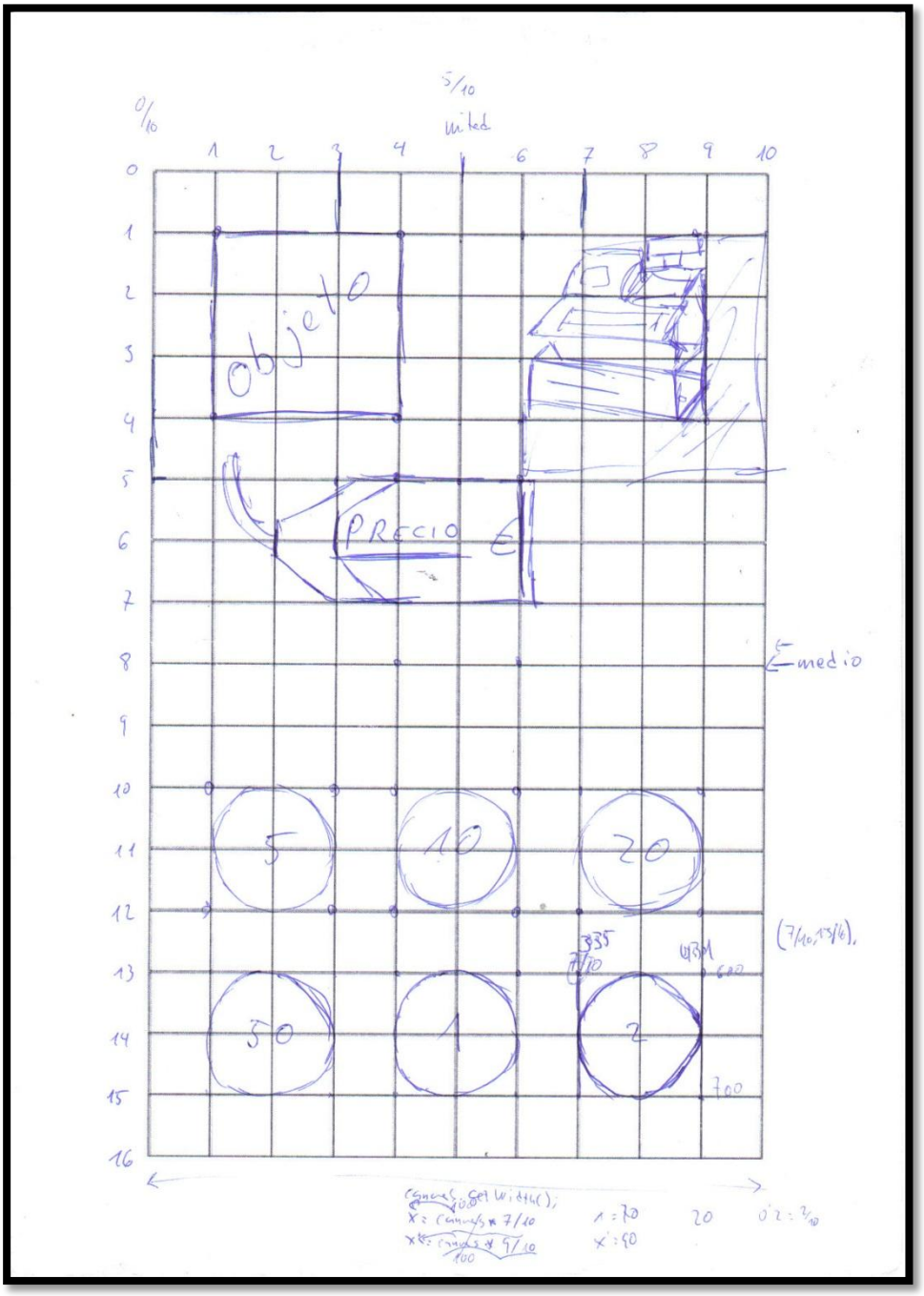


Figura 3.18

Obtenemos como resultado la imagen correspondiente a la Figura 3.19, en la cual vemos cómo comienza a tomar forma de lo que podría llegar a ser una situación real de ir a comprar un producto, representada gráficamente de una manera sencilla pero eficaz, cumpliendo con el objetivo de no sobrecargar la pantalla de objetos que puedan distraer en exceso a los usuarios.

Figura 3.19

Se pueden apreciar diversos cambios, por ejemplo el color del fondo se ha sustituido por uno menos llamativo, con la intención de resaltar los objetos importantes en pantalla. También disponemos de dependiente, caja, objeto, precio y monedas. Por último, nos fijamos que como ayuda nos indica cuánto nos falta por pagar. Esta etiqueta será eliminada para la versión final, pero en estos momentos es necesaria para el desarrollo de la implementación y corregir errores de la aplicación.

Además, en esta fase incluimos un nuevo nivel, gracias al cual el usuario aprenderá el valor de las monedas para saber si es capaz de comprar un determinado producto. Este nivel era estrictamente necesario que fuera sencillo, de modo que su interfaz gráfica también debía serlo. Por este motivo en pantalla aparecerán únicamente el objeto, una moneda o billete, y dos botones con los que indicar si somos capaces de comprar el objeto o no. En caso de acertar se mostrará una mano indicando que la

respuesta ha sido correcta acompañada de una melodía pegadiza y alegre, y si se falla sonará un sonido desagradable.

En la Figura 3.20 se muestra el esbozo con el cual calculamos la distribución en pantalla de todos los ítems necesarios.

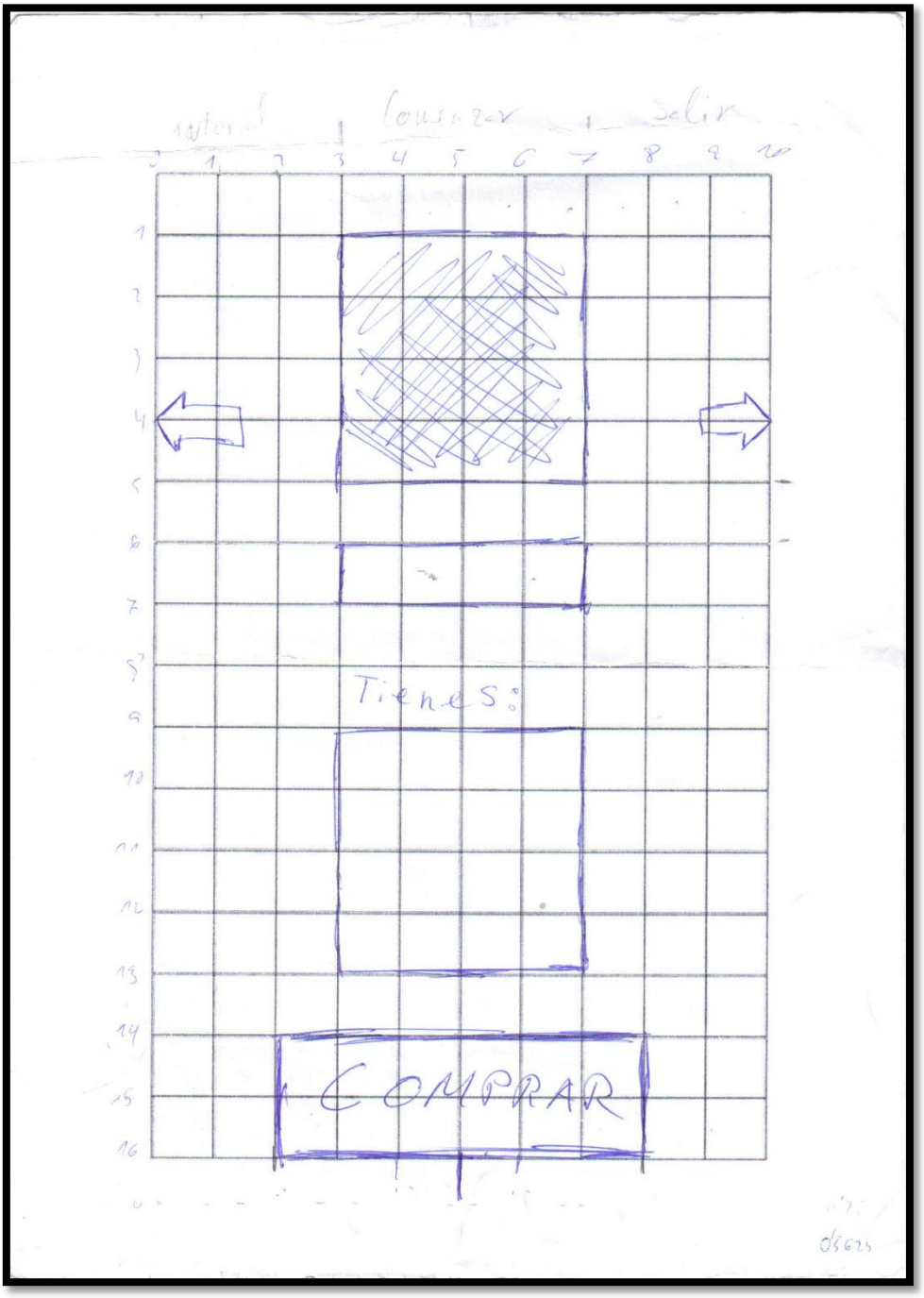


Figura 3.20

Este esquema da lugar a la siguiente pantalla (Figura 3.21) en la cual vemos también la interacción producida al acertar correctamente la pregunta propuesta.

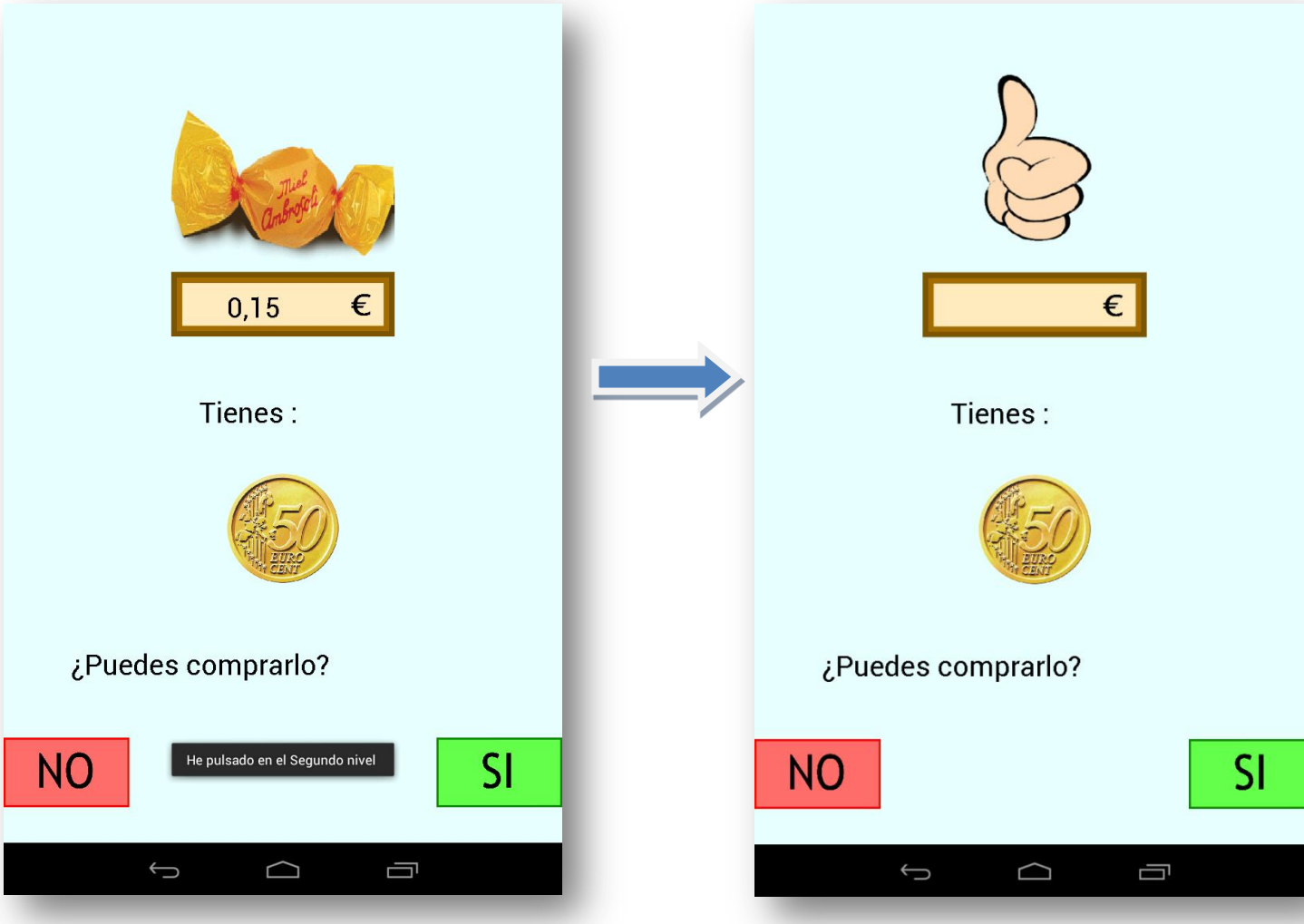


Figura 3.21

Como consecuencia de este nuevo nivel, la lista de temas también ha cambiado (Figura 3.22) y hemos incorporado un nuevo botón, que generará la actividad anterior.

Figura 3.22

Por último, en esta fase se implementó la pantalla de premio. En dicha pantalla se ve una hucha de cerdito, una mano con moneda, la cual introducirá en la hucha, y se mostrará que ha ganado dinero al hacerlo bien (Figura 3.23 y 3.24). Todo ello acompañado de música alegre, con el objeto de reforzar el aprendizaje.

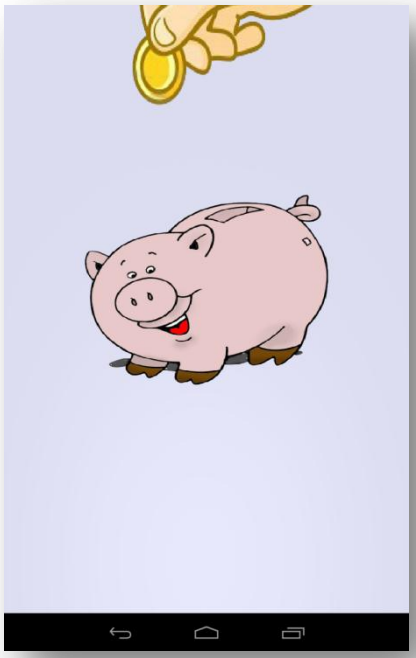


Figura 3.23

Figura 3.24

3.2.4 Cuarta fase

Esta fase es la fase del dinamismo, es decir, a lo largo de la misma voy a explicar cómo he logrado otorgar a la aplicación movimientos que han hecho que sea mucho más completa y eficaz con los objetivos propuestos. Además, gracias a estos movimientos, se logra darle un toque mucho más profesional, tanto de jugabilidad como de simulación, ya que está claro que si el dependiente te saluda y habla, su boca se moverá, los ojos y las cejas también, etc.

Finalmente, la portada también será cambiada, ya que al ser un juego para niños he creído conveniente que salga un niño animado y mirando a los botones personalizados para iniciar el juego, ver el tutorial o salir. A continuación veremos todas estas ideas y otros detalles que se han añadido en esta fase.

De modo que vamos por partes. Comenzaremos mostrando desde el inicio de la aplicación todas las pantallas y analizaremos sus cambios.

En primer lugar la portada (Figura 3.24).

Figura 3.24

En la imagen 3.24 podemos apreciar a un niño delante de un centro comercial y con los botones de “tutorial”, “comenzar” y “salir” sobre él. El centro comercial va a hacer referencia a las compras, ya que se trata de un juego que simulará esta situación de manera sencilla. Además, cuando se ejecuta el juego, se puede apreciar en esta pantalla cómo el niño mueve los ojos mirando a cada uno de los bocadillos que tiene

sobre él, cómo se mueve el avión y el globo de lado a lado y sonará una música de fondo agradable para los usuarios. El desarrollo del niño ha sido el siguiente:

1- Para comenzar, se tomó la imagen en blanco y negro (Figura 3.25) y se le dio color.

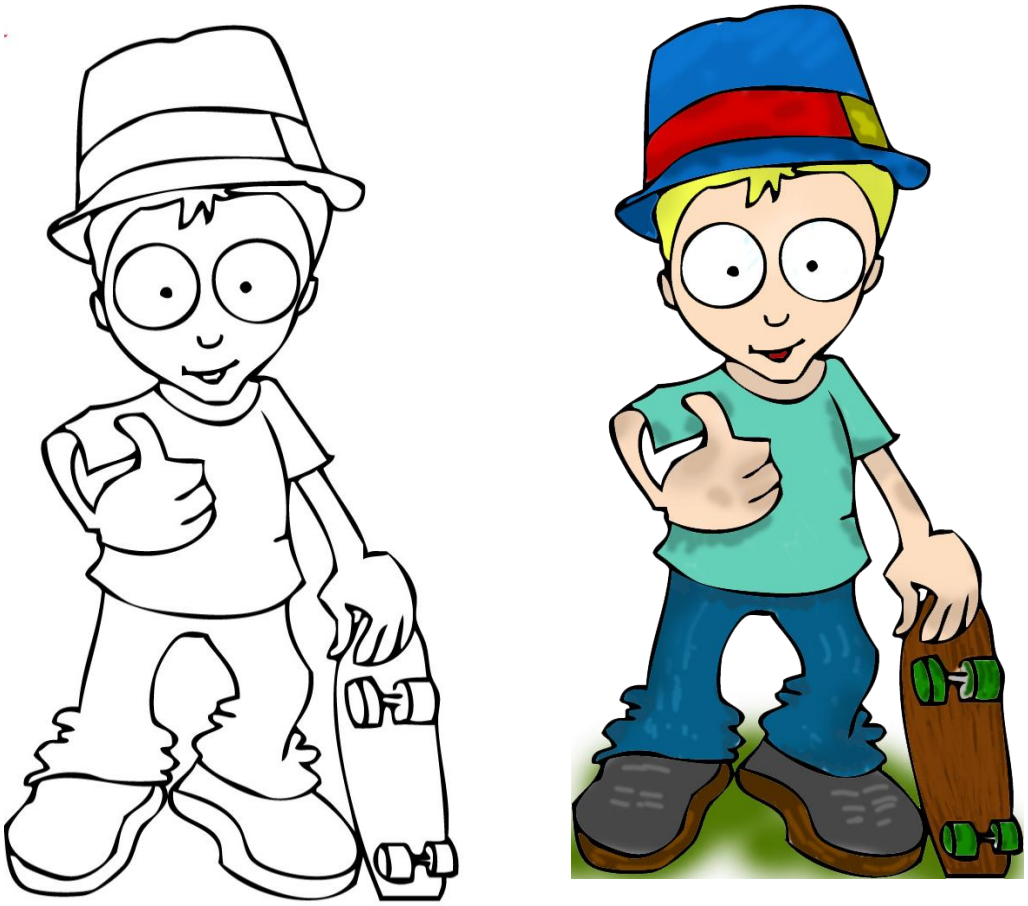


Figura 3.25

Se tuvo especial cuidado con los detalles como son las sombras, los colores vivos, y la imagen seleccionada para representar la animación. También se tuvo en cuenta la sombra que proyectará el niño en el fondo, todo con la intención de lograr los efectos más realistas posibles.

2- A continuación se realizó la copia del niño de modo que tenemos la misma imagen repetida varias veces. En cada una de las repeticiones realizamos pequeñas variaciones en función de los movimientos que deseemos, por ejemplo, si queremos que mire por cada parpadeo a cada uno de los bocadillos, deberemos de disponer de una imagen mirando al primero, otra al segundo y otra mirando al tercero. En mi caso mira al frente cada vez que completa una serie mirando hacia arriba (Figura 3.26). Más adelante, en la implementación, mostraré el código necesario para animar las imágenes.

Figura 3.26

3- Se ajustan todas las imágenes a la pantalla en función de la cuadricula. Como muestra del planteamiento tenemos el siguiente esquema realizado previamente (Figura 3.27):

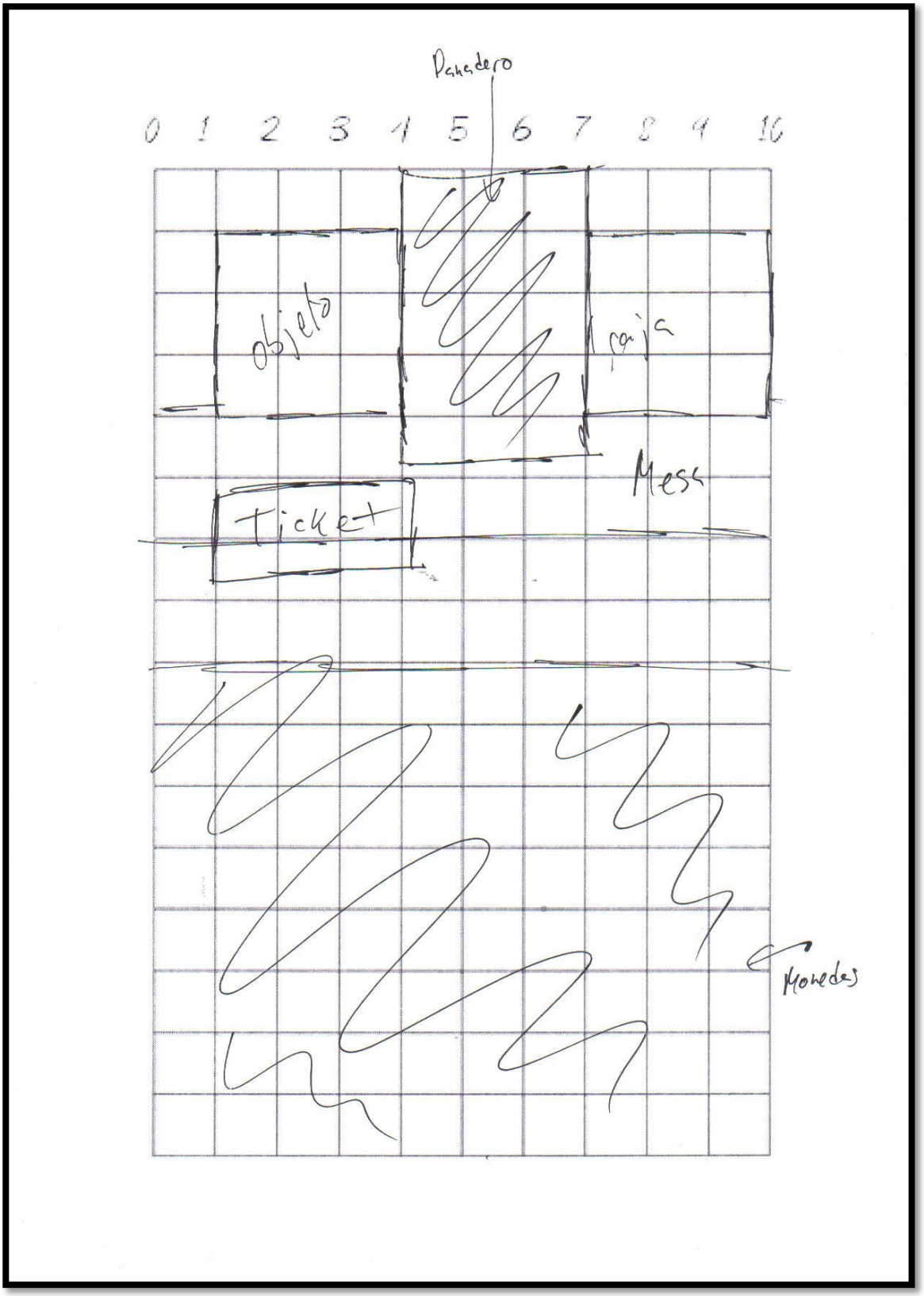


Figura 3.27

Después, vamos a ver el tutorial (Figura 3.28):

Figura 3.28

En la imagen superior, se puede apreciar el resultado final que tendrá el tutorial. En ella vemos el objeto con buena definición y ajustado a la cuadricula como hemos visto anteriormente, la caja correctamente situada a su lado, el precio en su posición, y las monedas moviéndose por la pantalla, mientras suena una voz indicando lo qué se debe hacer. Cuando se ha finalizado el movimiento y la voz, se muestra en pantalla un mensaje como se muestra en la Figura 3.29:

Figura 3.29

Todo ello ajustado según el esquema de la Figura 3.30.

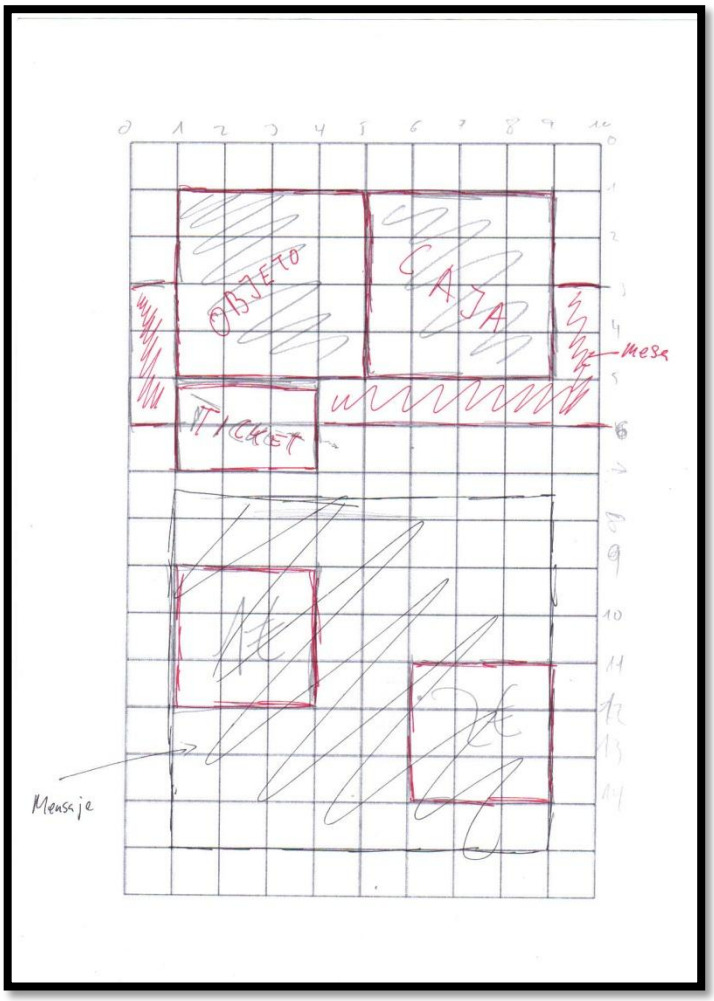


Figura 3.30

Con esto damos por concluida la pantalla del tutorial, que se mantendrá sin cambios en la aplicación.

Ahora procederemos a analizar la pantalla de temas.

Para empezar, hemos reordenado los temas para que sigan el siguiente orden:

1- Precio exacto con monedas.
2- Saber si nos llega con monedas.
3- Precio exacto con billetes.
4- Saber si nos llega con billetes.
5- Comprando en una maquina expendedora con el dinero ganado.

Con todo esto hemos reemplazado el último tema por otro más sencillo y visual, gracias al cual mostraremos los cambios y será el último nivel de nuestro juego.

Por lo tanto, vamos a analizar el interfaz gráfico del último nivel, paso a paso.

En primer lugar el esquema que vamos a seguir para representarlo en pantalla es el siguiente (Figura 3.31):

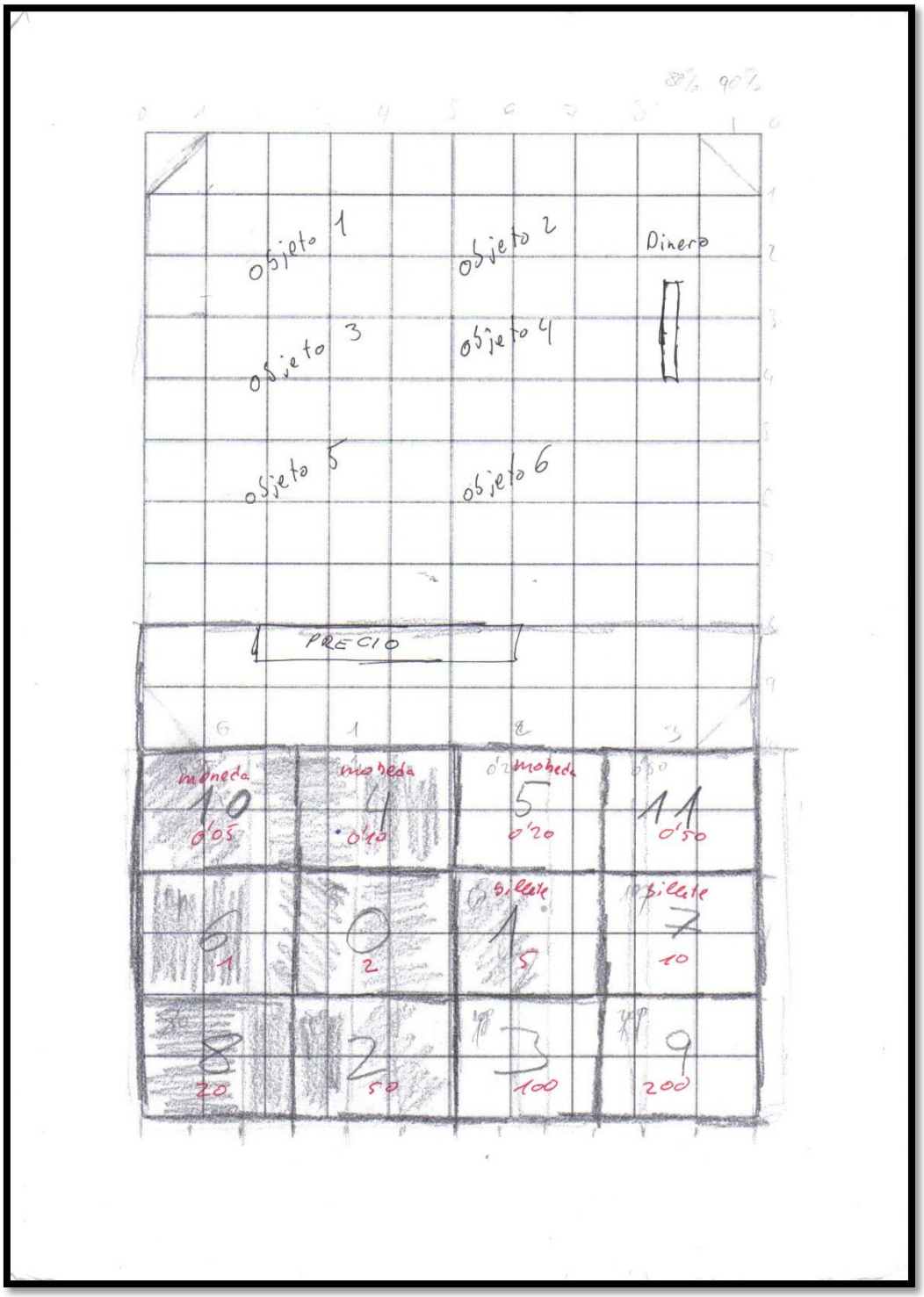


Figura 3.31

Para simular la máquina expendedora he creado con 3dmax la siguiente Figura que emplearé en la aplicación (Figura 3.32).

Figura 3.32

En dicha máquina, se mostrarán seis objetos, que al seleccionar uno de ellos se iluminará por detrás, y en función de si tenemos dinero suficiente se iluminará el letrero de "Dinero" de color verde (en caso de disponer una cantidad suficiente para el pago), como se muestra en la Figura 3.33, o en rojo (en caso contrario), por ejemplo la Figura 3.34.

Además, en la parte baja se mostrará el precio del producto y en la parte inferior de la pantalla se mostrarán las monedas, si después de jugar a los temas anteriores hemos ganado dinero o si nos ha sobrado de anteriores veces (Figura 3.33 y 3.34); y en caso de no disponer de dinero, se mostrará un mensaje indicando que debemos jugar a los temas para ganarlo (Figura 3.35). A continuación se muestran las imágenes mencionadas.

Figura 3.33

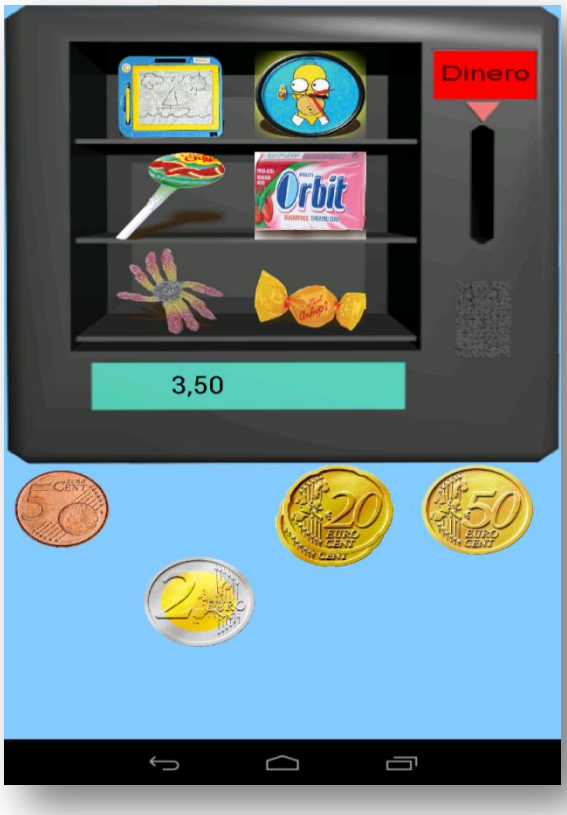


Figura 3.34

Figura 3.35

Ahora bien, en caso de disponer el dinero suficiente para comprar el artículo seleccionado, y pagarlo correctamente arrastrando las monedas hasta la ranura, he creado una transición animada gracias a la cual mostraré los cambios recibidos de pagar el artículo. En cuanto se ha pagado correctamente un producto, sonará una melodía alegre, y se creará una transición de un cuadrado azul desde el centro hasta que ocupe toda la pantalla. Una vez está toda la pantalla cambiada se mostrará el objeto comprado (y tendrá una estrella animada detrás como si de un gran tesoro se tratase) y se mostrarán los cambios recibidos (en caso de haberlos), como se puede apreciar en la Figura 3.36.

Figura 3.36

Por otro lado, también se ha modificado al dependiente, ya que ahora ha adquirido movimiento, los pasos han sido los mismos que se tuvieron que realizar con el niño y su movimiento de ojos. La imagen resultando es la Figura 3.37, en donde se puede apreciar que cambian los ojos, la boca, el bigote y las cejas.

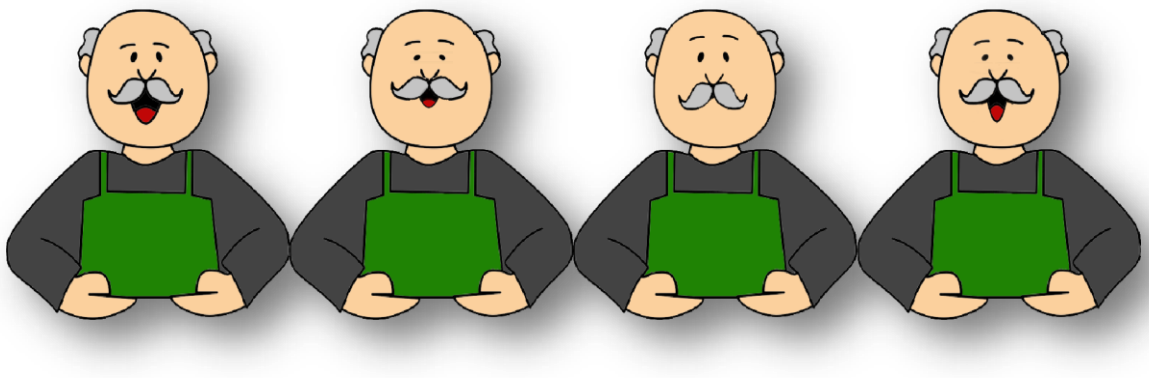


Figura 3.37

Durante el juego, si se detecta que está sonando el sonido de la voz del dependiente, se actualizará la imagen simulando que el dependiente habla.

Con esto tendrá lugar la finalización de la cuarta fase del diseño de la aplicación.

Por último, se mostrará la última fase, en la cual veremos el estado final de la aplicación.

3.2.5 Fase final

Esta es la última fase de desarrollo grafico, en la cual voy a mostrar el resultado final de la aplicación paso a paso. En este caso no se trata de un desarrollo de los niveles desde el principio, sino que se han añadido fondos o más imágenes en movimiento, empleando las técnicas descritas en las fases anteriores. También se han corregido fallos menores relacionados con el apartado gráfico.

Pantalla principal (Figura 3.38).

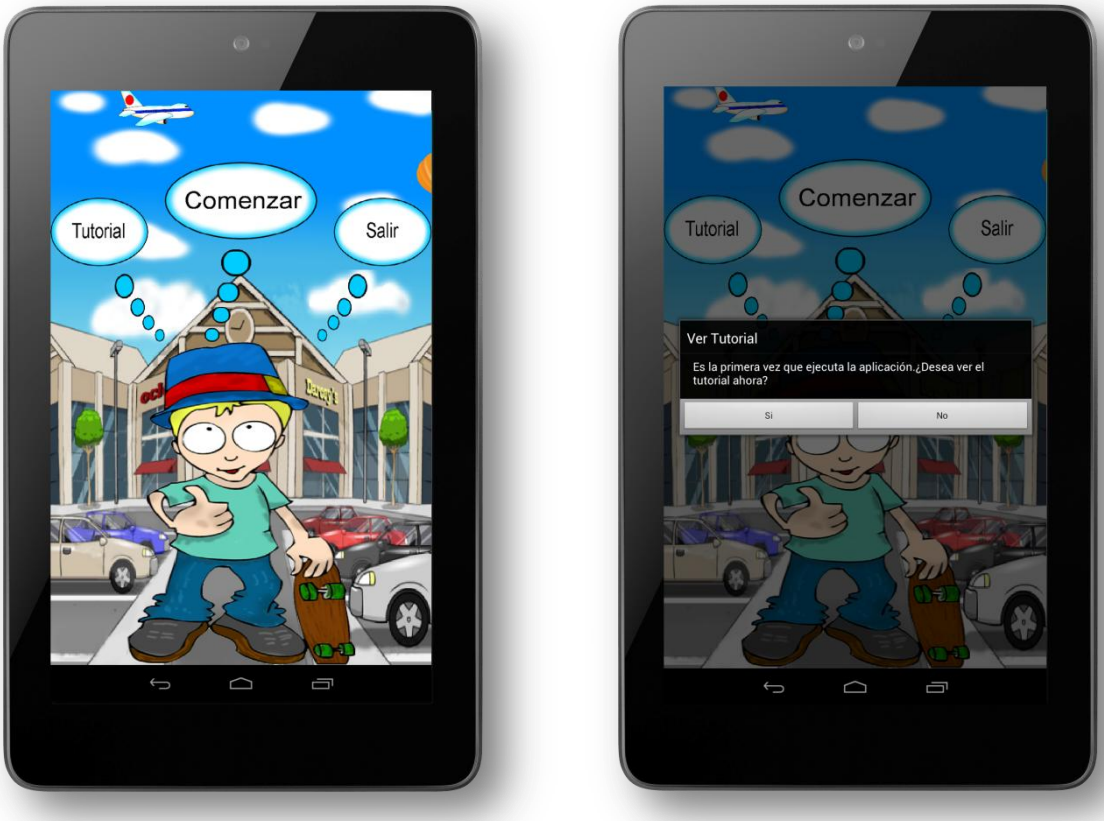


Figura 3.38

Pantallas del tutorial, Figura 3.40

Figura 3.40

Pantallas de selección de tema, Figura 3.41

Figura 3.41

Pantallas del primer nivel Figura 3.42

Figura 3.42

Pantallas de logro conseguido, Figura 3.43

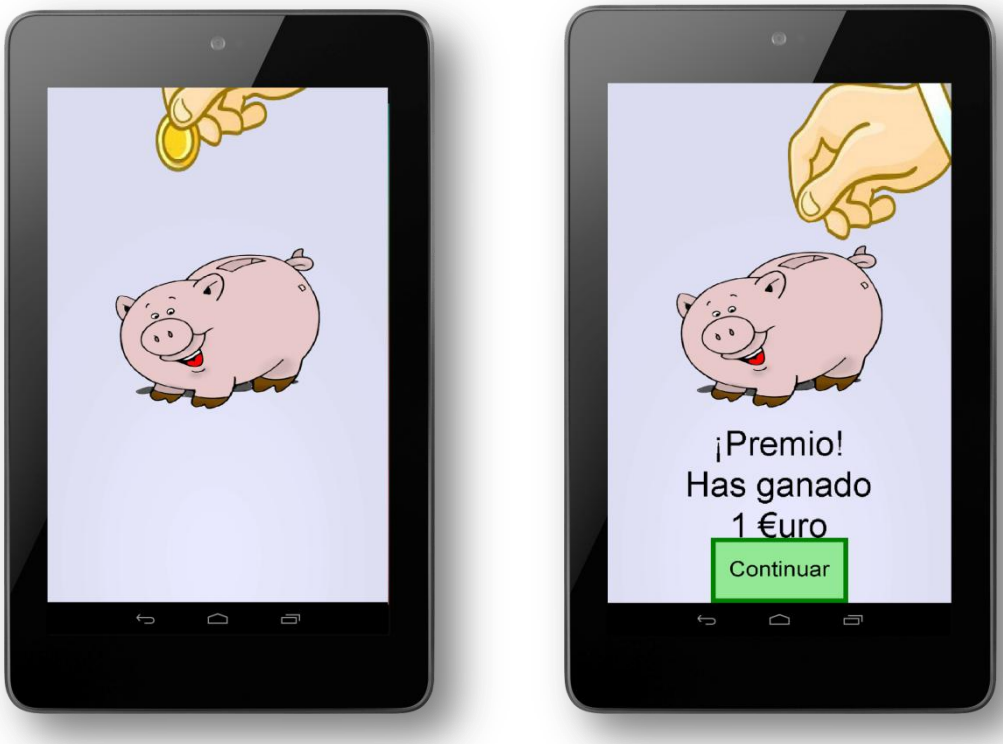


Figura 3.43

Pantallas del segundo nivel, Figura 3.44

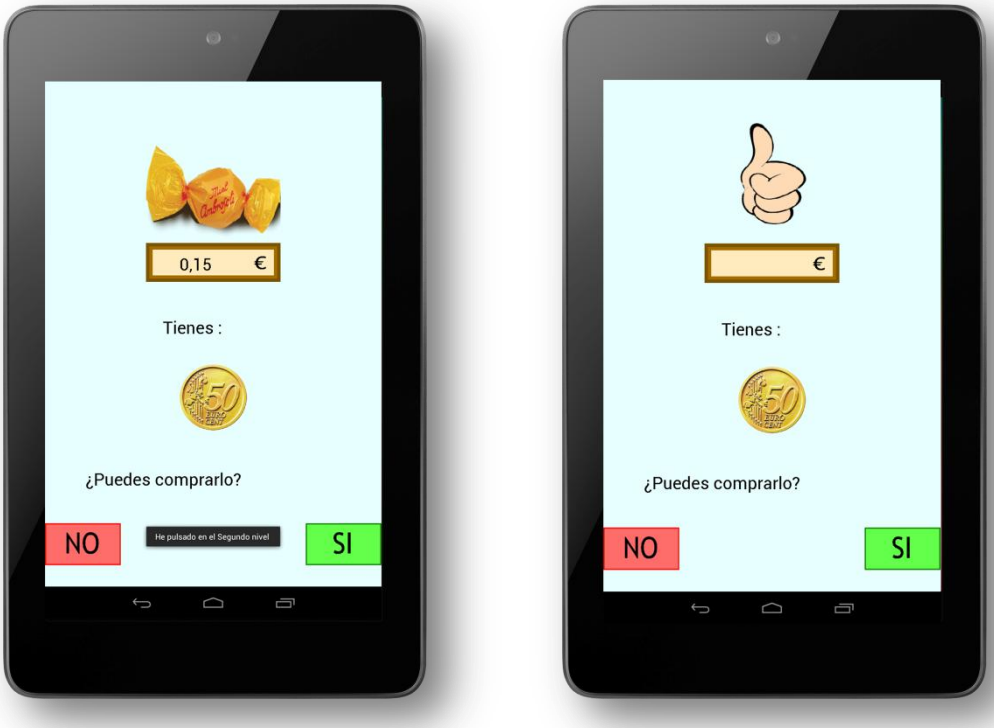


Figura 3.44

Pantalla del tercer nivel, Figura 3.45

Figura 3.45

Pantallas del cuarto nivel, Figura 3.46

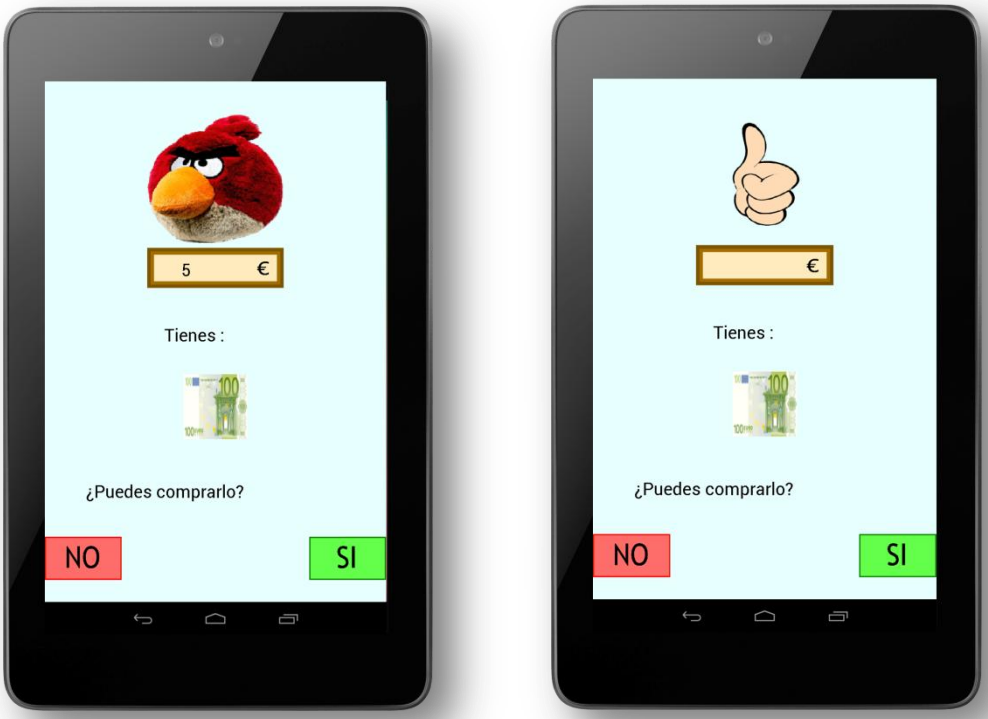


Figura 3.46

Pantallas del último nivel, Figura 3.47 y 3.48

Figura 3.47

Figura 3.48

3.2.6 Diseño del icono

Para desarrollar el icono, he tomado como idea la temática de mi aplicación, es decir los euros. Además, Google exige una resolución mínima de 500x500 aproximadamente. Mi icono tiene actualmente una resolución de 800x800. Para ello he tomado una montaña de monedas y le he añadido el logo de los euros como imagen principal. El resultado se puede observar en la Figura 3.49 y se comprueba que se visualiza claramente en la Figura 3.50.

Figura 3.49

La montaña de monedas ha sido aclarada para resaltar más el logo del euro, que a su vez se le ha aplicado sombra para lograr aún mayor efecto visual.

Por último, se comprueba que se visualiza correctamente en la pantalla, probando a arrastrarlo hasta la pantalla principal del terminal.
Tal y como se puede apreciar en la Figura 3.50, el logo ha quedado sencillo y acorde al resto de iconos de las aplicaciones para Android.

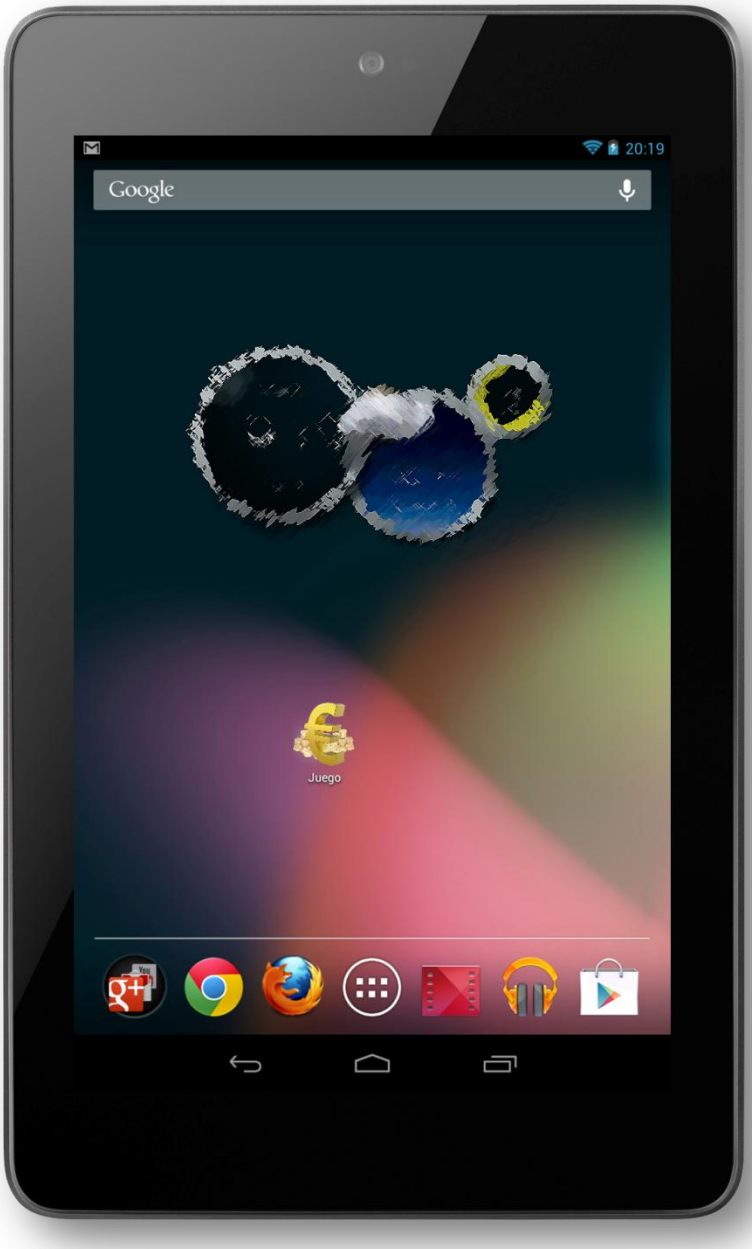


Figura 3.50

3.3 Diseño de los casos de uso

Ahora veremos cada caso de uso mediante diagramas de secuencia, los cuales comienzan cuando el usuario accede a cada una de las pantallas de la aplicación, todas mostradas anteriormente. Los diagramas de secuencia permitirán una mejor implementación de la aplicación.

De nuevo seguiremos el orden descrito hasta ahora, comenzando por la pantalla principal, el tutorial, la selección de temas, los diversos temas disponibles y la pantalla de premio para finalizar con el diseño de los casos de uso. Este diseño lo he basado en función de las posibilidades de las que dispone el usuario en cada pantalla, ya que me ha parecido más sencillo y razonable realizarlo de esta manera y no analizar cada caso de uso.

Comenzamos con la pantalla principal:

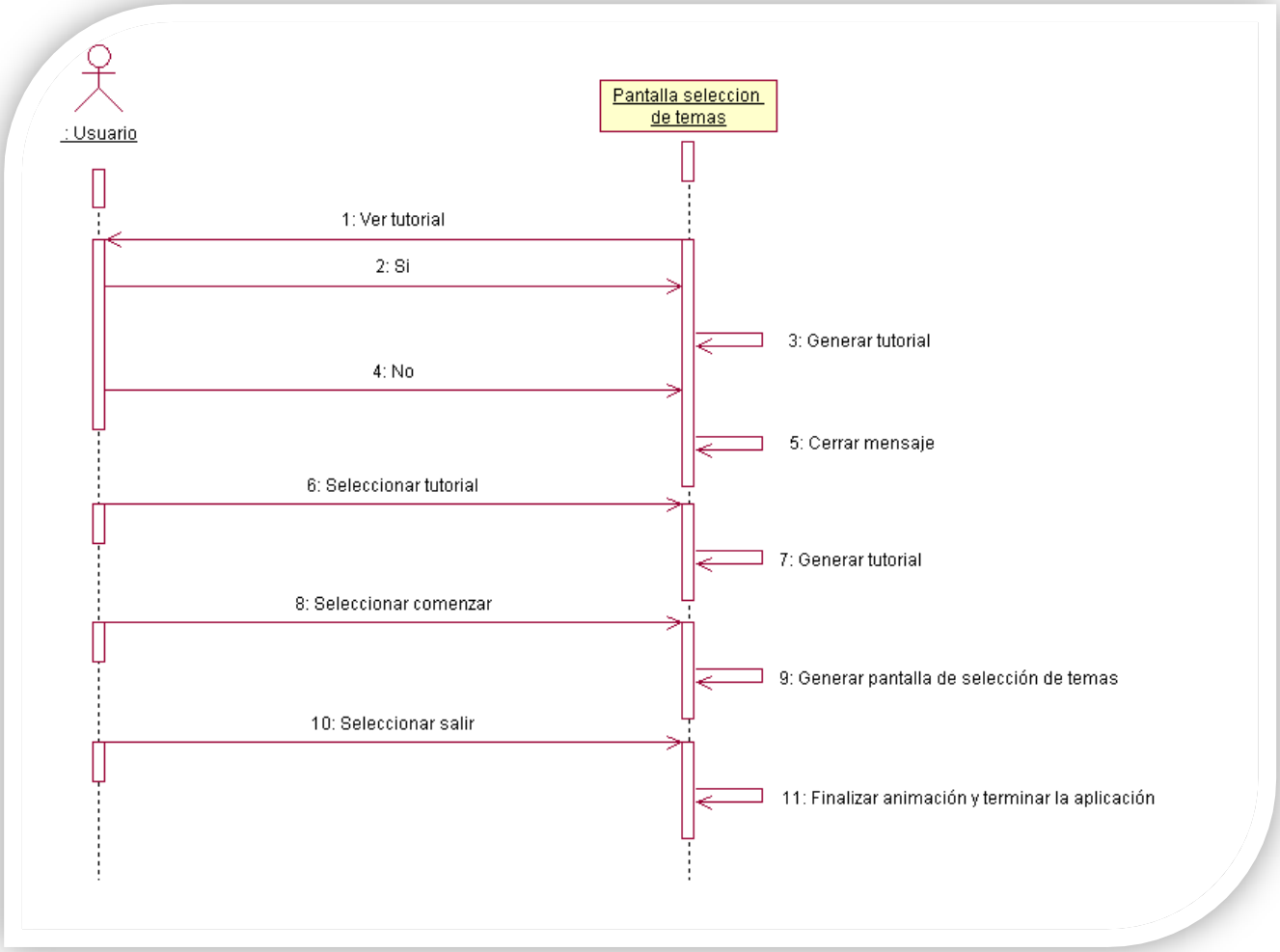


Figura 3.51

Pantalla tutorial:

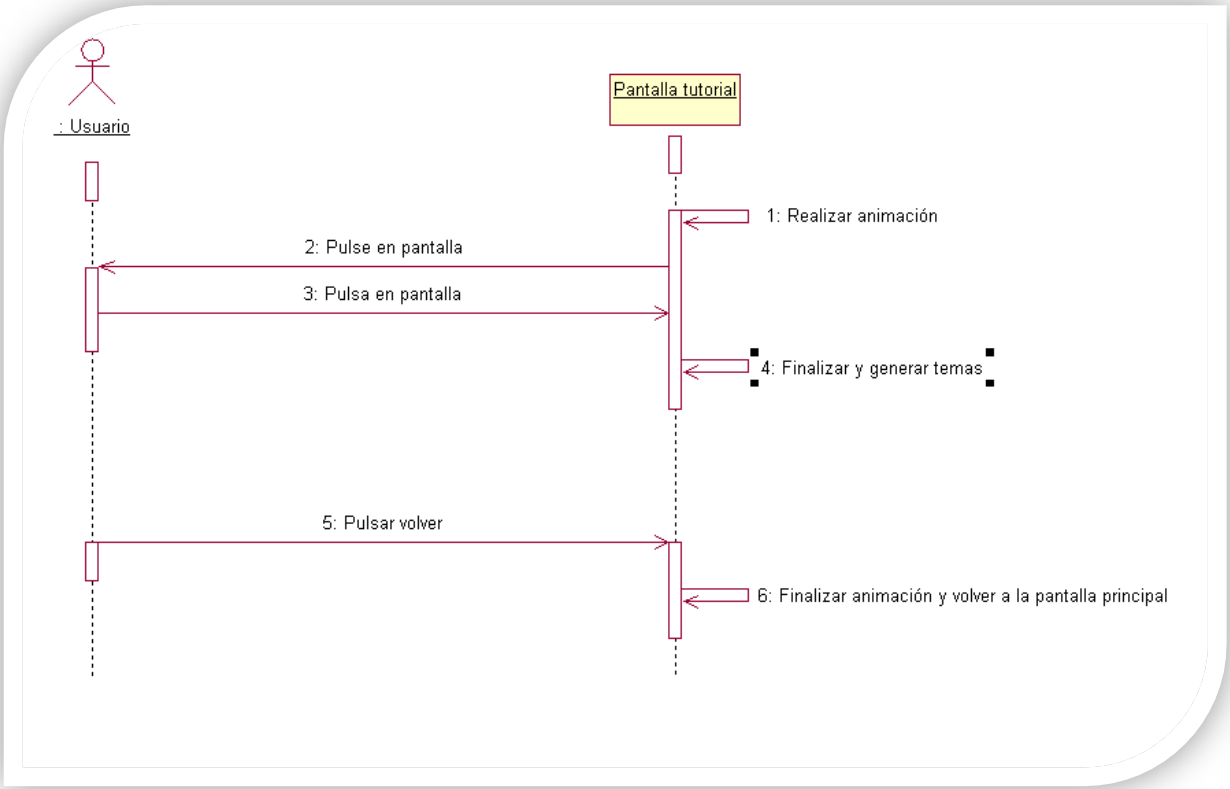


Figura 3.52

Pantalla de selección de temas

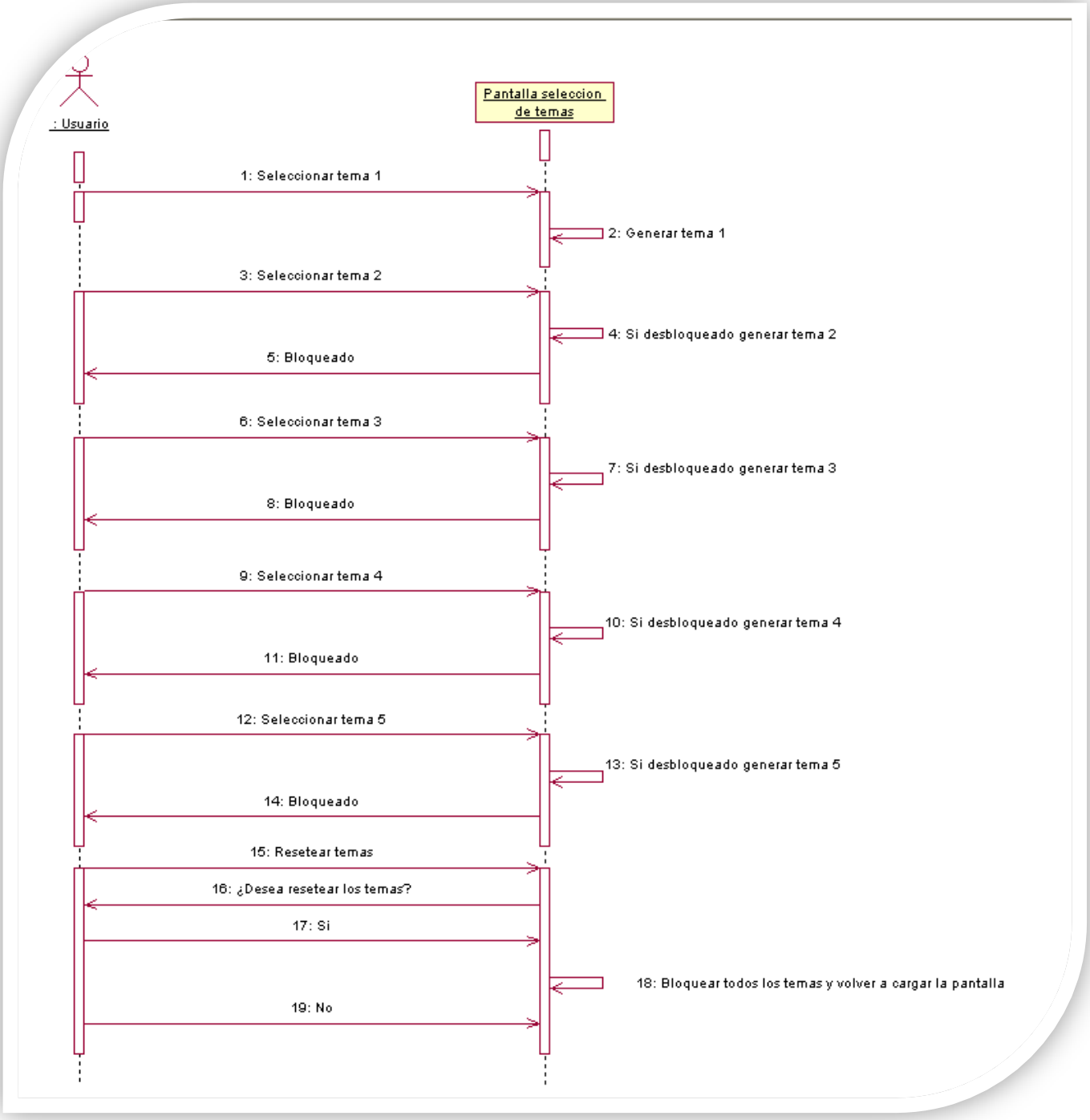


Figura 3.53

Pantalla tema 1

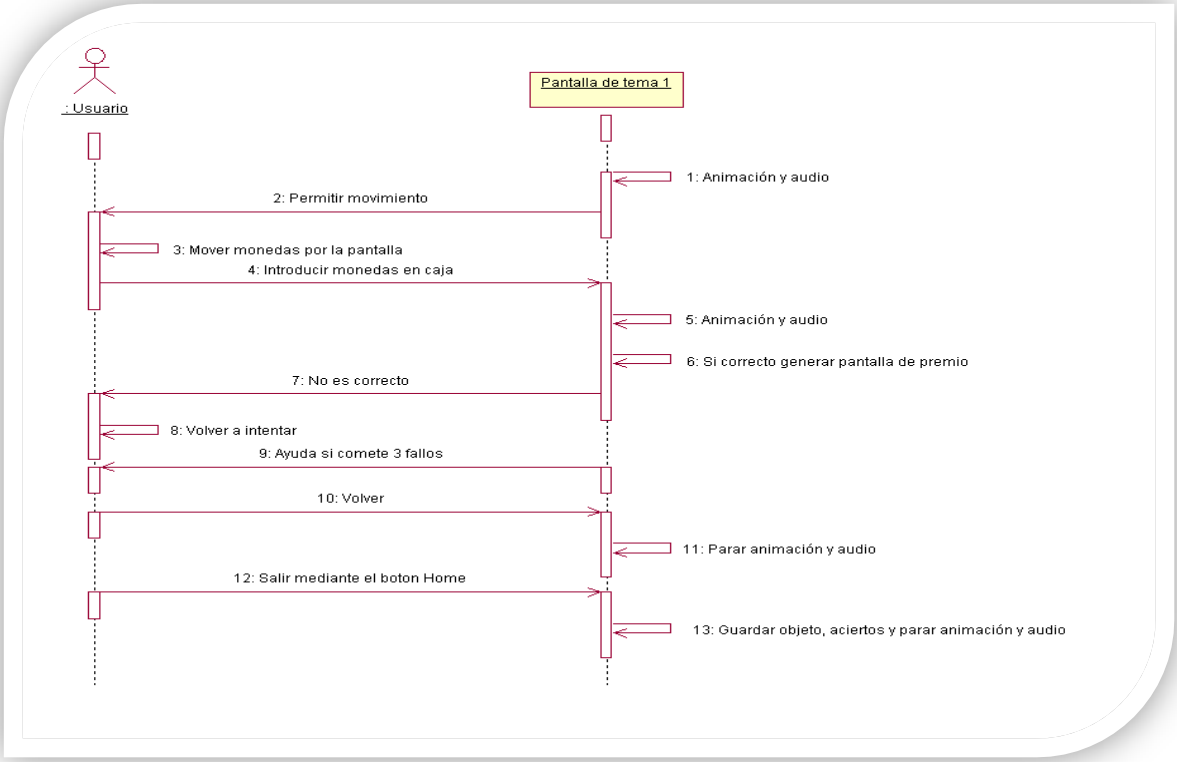


Figura 3.54

Pantalla tema 2

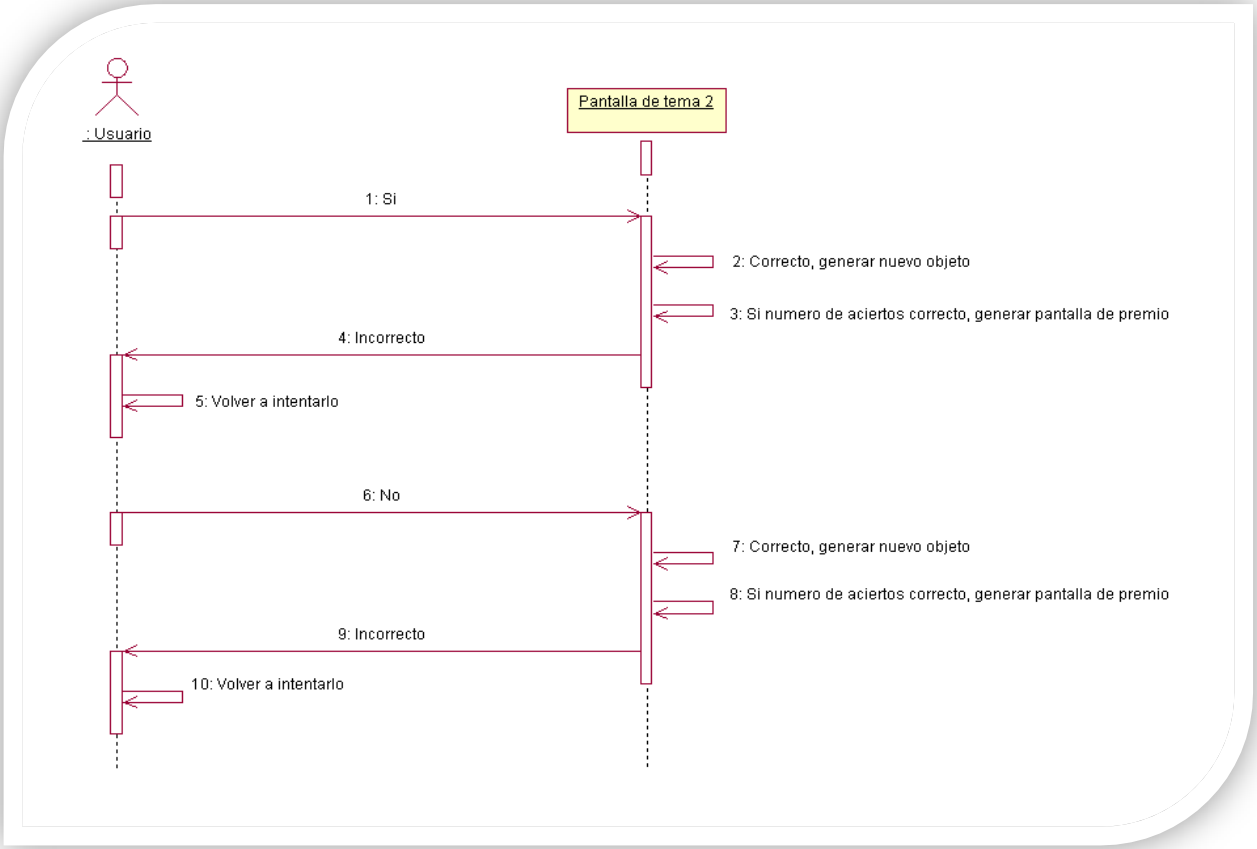


Figura 3.55

Pantalla tema 3

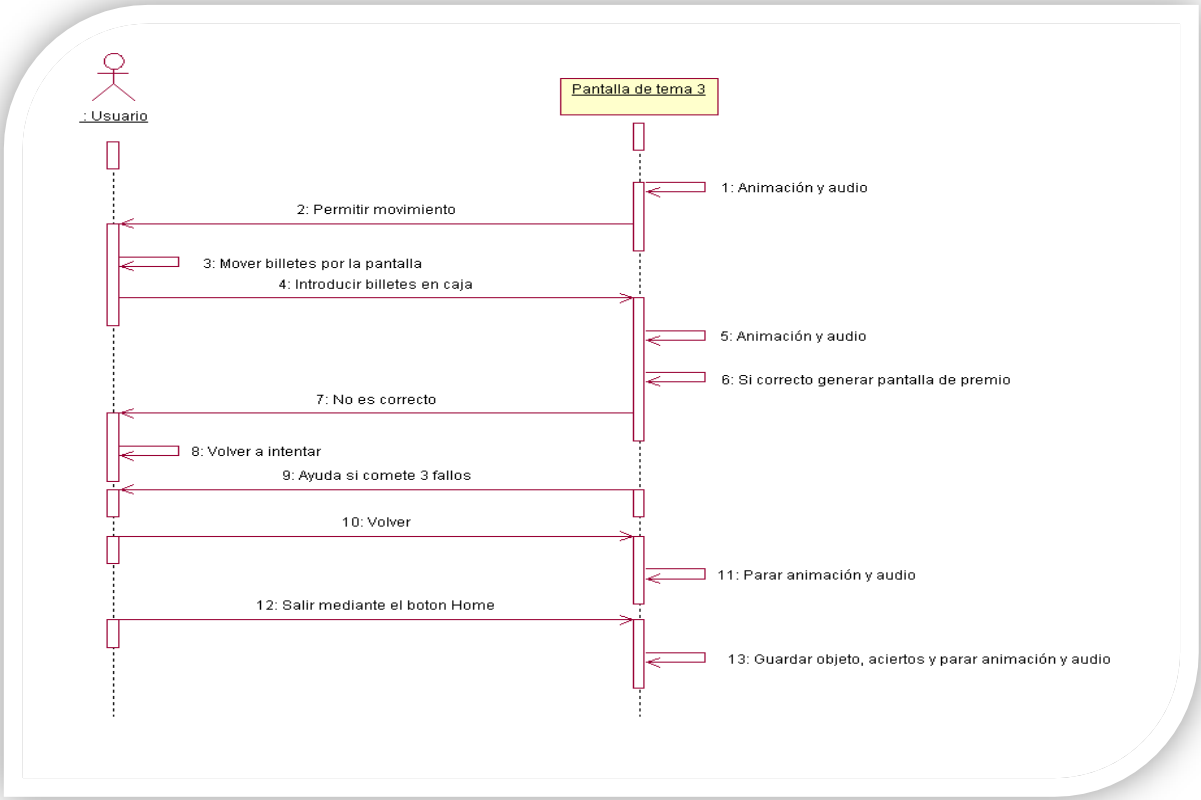


Figura 3.56

Pantalla tema 4

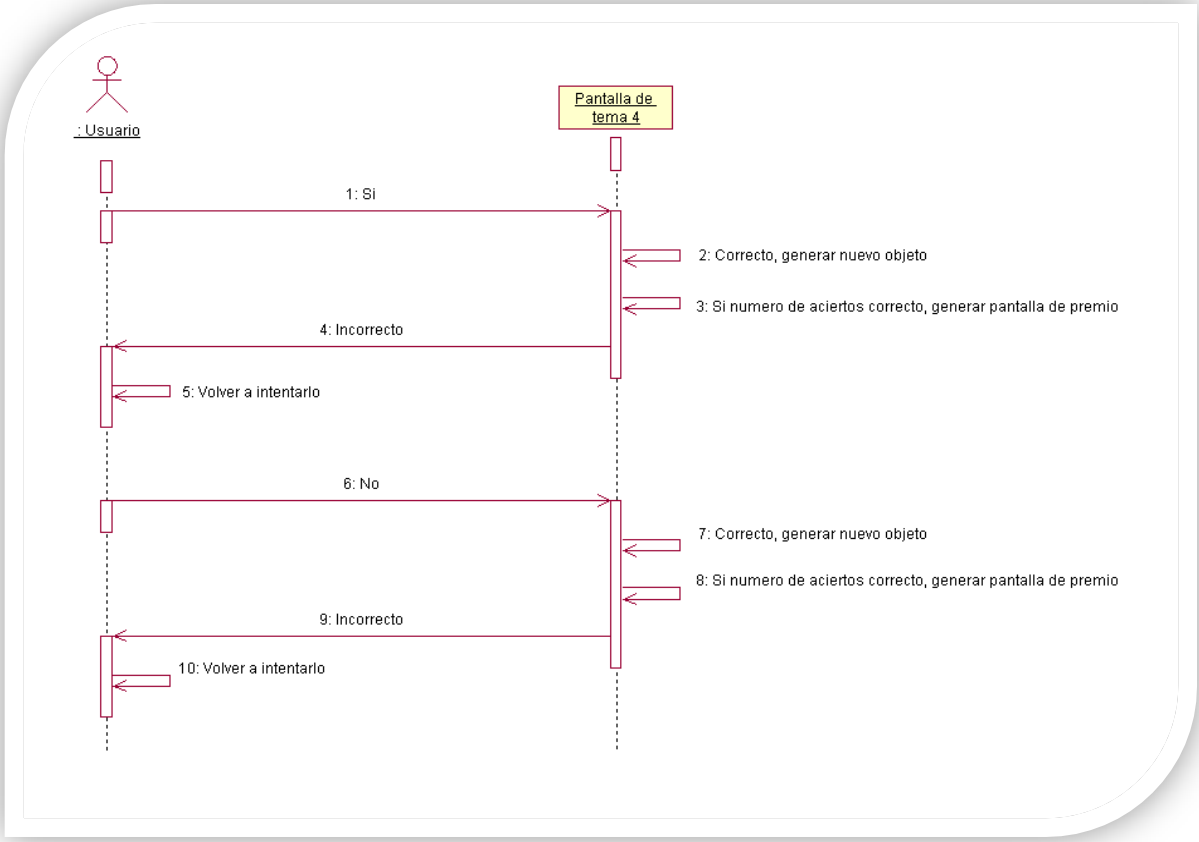


Figura 3.57

Pantalla tema 5

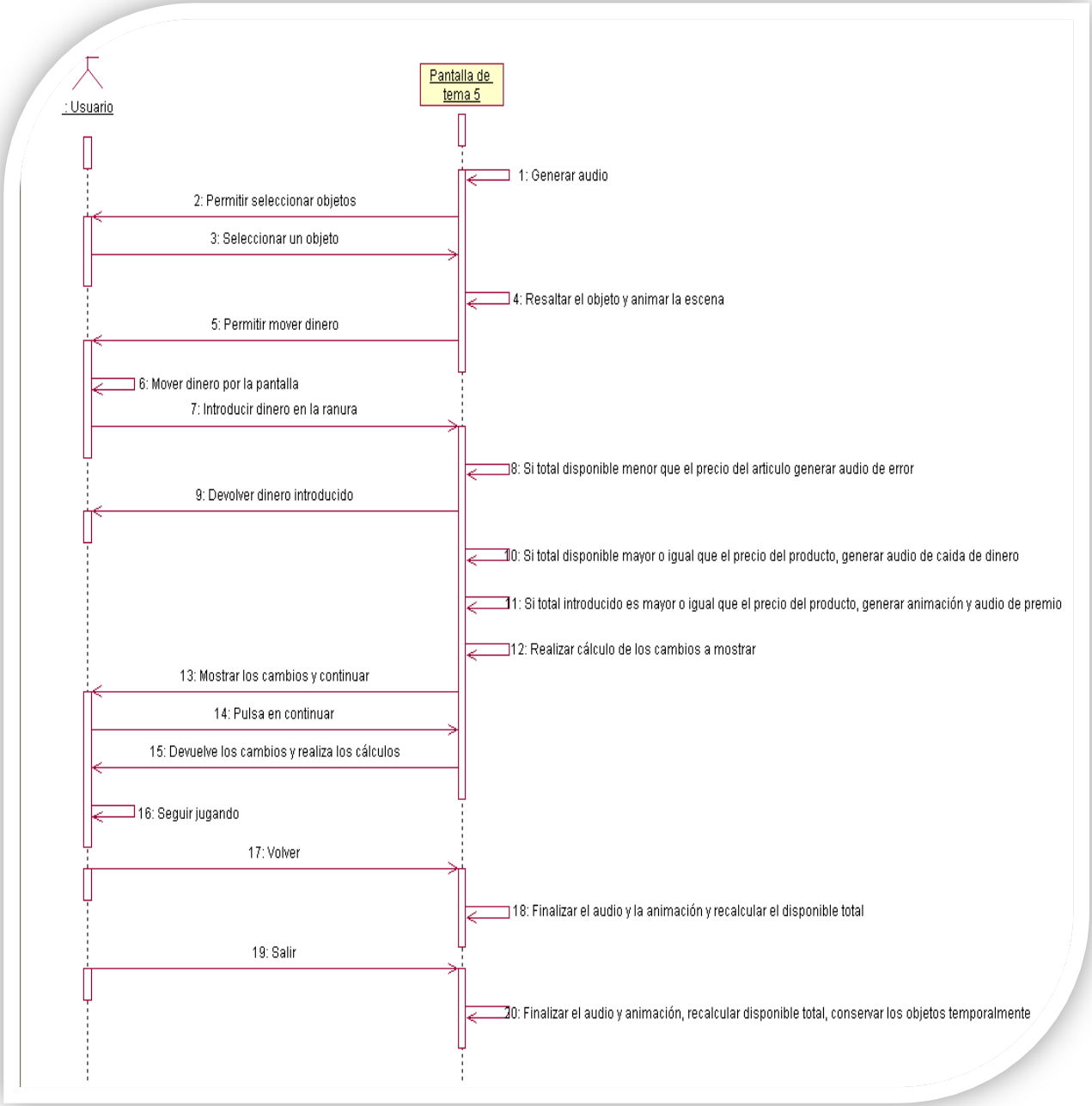


Figura 3.58

Pantalla de premio

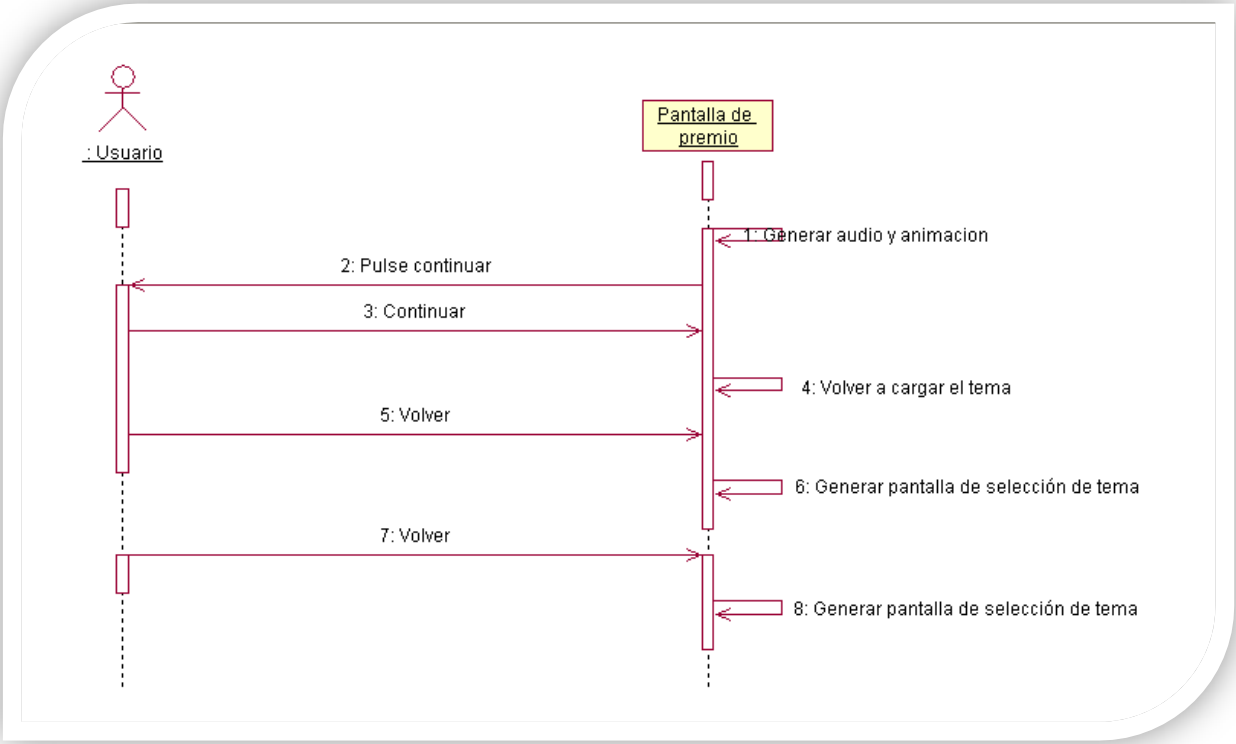


Figura 3.59

3.4 Diagrama de clases

A continuación se van a mostrar las figuras correspondientes al diagrama de clases de la aplicación. La Figura 3.60 muestra un esquema general de las clases empleadas.

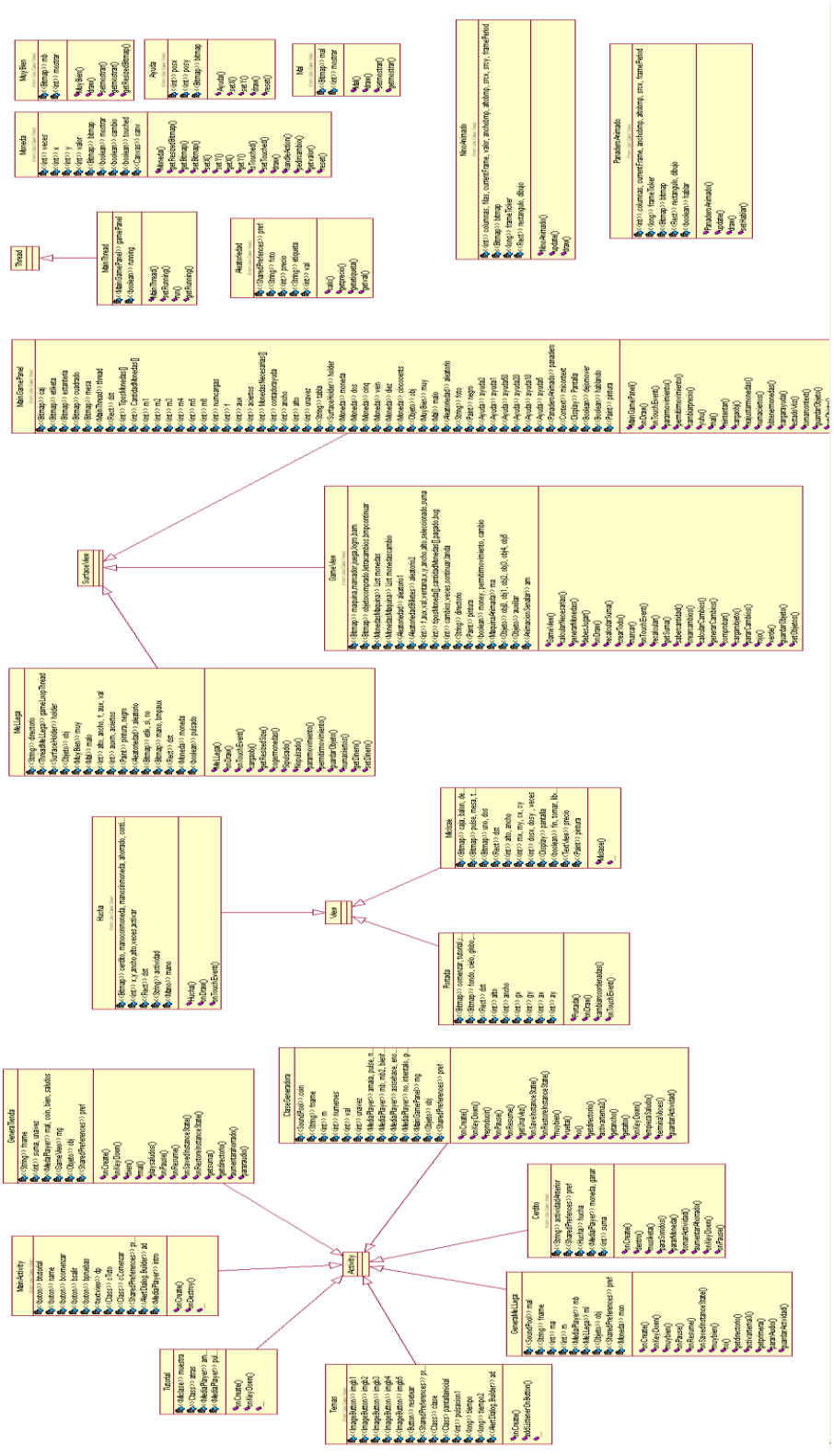


Figura 3.60

Debido a que se trata de un juego por niveles, que a su vez son independientes entre sí, sus clases también lo son. Por lo tanto se va a proceder a mostrarlas cada una en particular.

En la Figura 3.61 se muestra el diagrama de la primera clase, que es generada al ejecutar la aplicación:

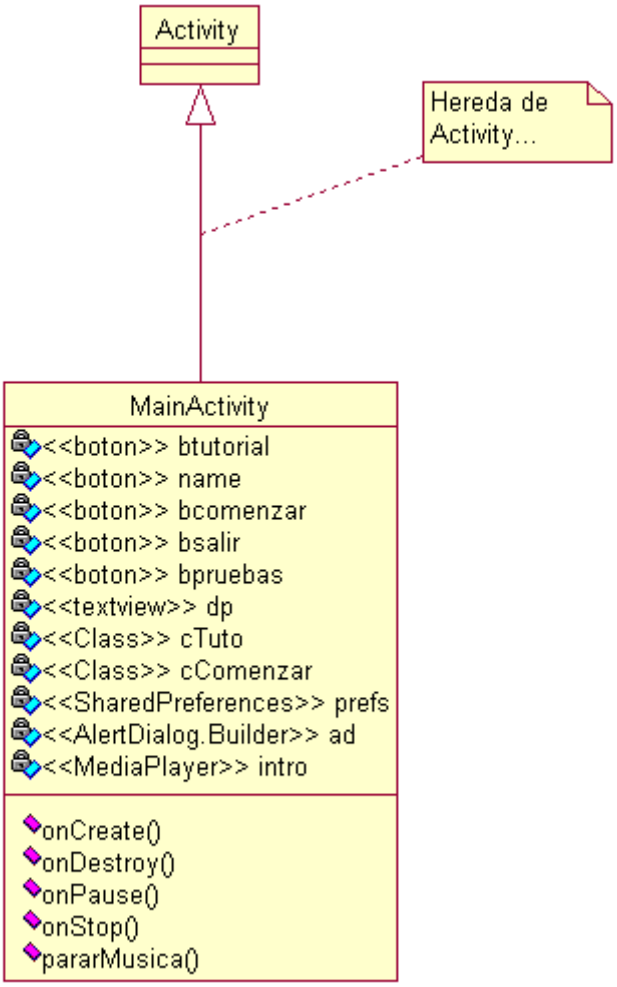


Figura 3.61, MainActivity

Esta clase será la encargada de generar la animación de la pantalla principal, que a su vez es una clase de tipo View como se puede apreciar en la Figura 3.62:

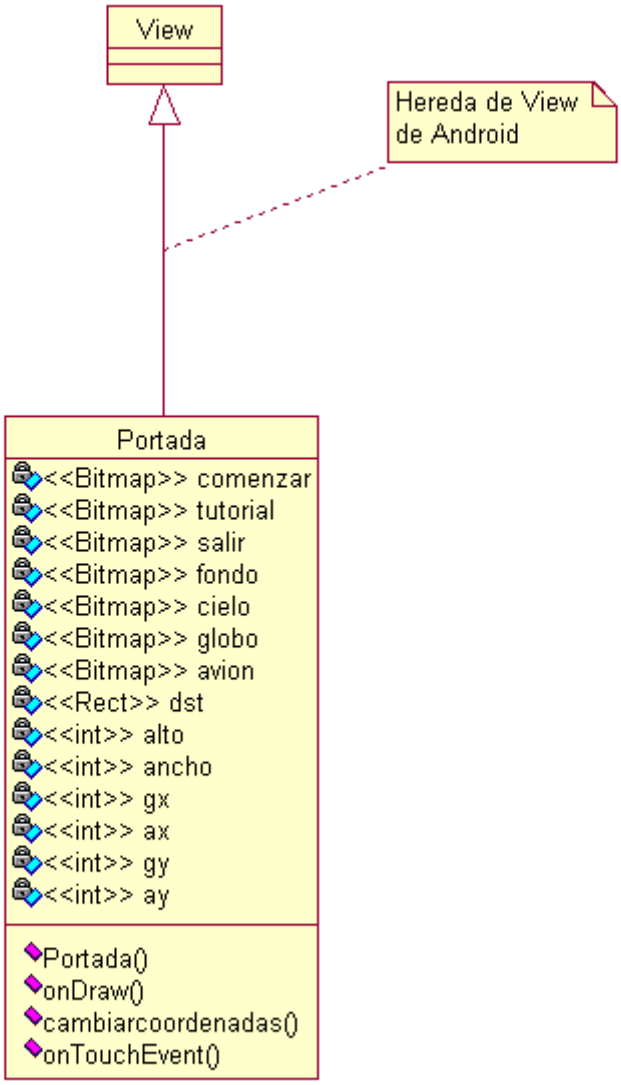


Figura 3.62

Esta clase será la encargada de llamar a ejecutar el tutorial, la pantalla de temas o salir en función del método onTouchEvent() que detecta qué opción hemos pulsado.

Ahora vamos a realizar el diagrama de clase de la clase que es llamada al seleccionar “Tutorial” en la pantalla de la portada (Figura 3.63):

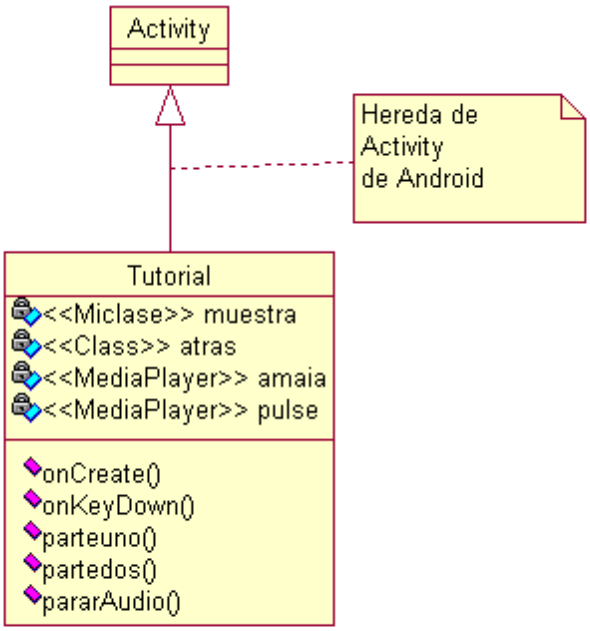


Figura 3.63

Dentro de onCreate llamará a muestra, y creará el tutorial. Se puede observar el diagrama de clase de Miclase en la Figura 3.64.

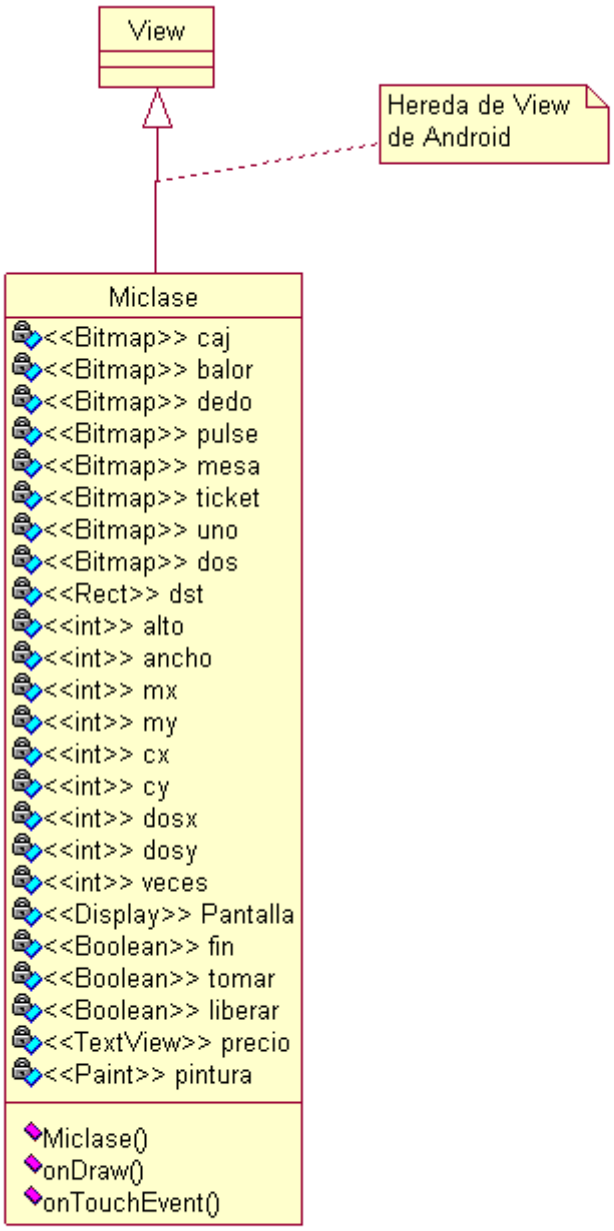


Figura 3.64

Ahora vamos a analizar el diagrama de clases de la clase Temas (Figura 3.65), que se encarga de mostrarnos en pantalla los diferentes botones con los temas. Esta clase se encarga de mostrar en pantalla un Layout en el que se encuentra la imagen a mostrar.

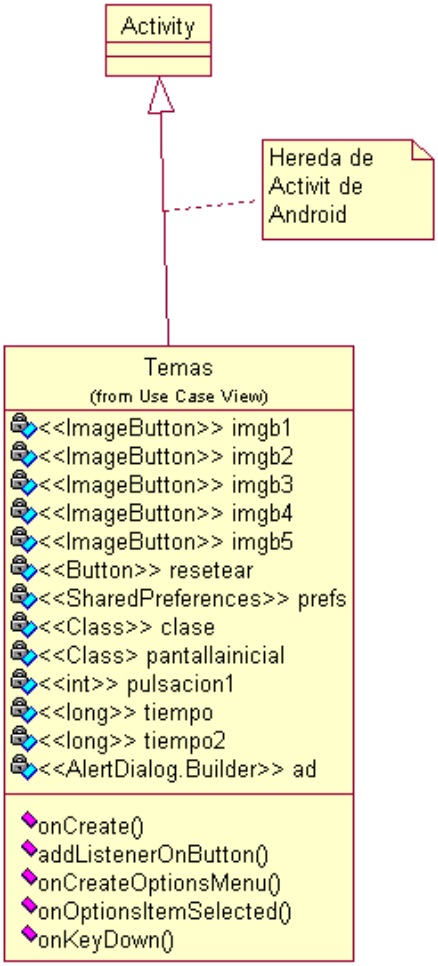


Figura 3.65

Esta clase permite resetear los temas de dos maneras posibles. Una de ellas es mediante las opciones, por eso es necesario disponer de OptionsMenu y OptionsItemSelected, y también es posible resetearlos mediante el botón resetear, que se encargará de mostrar en pantalla un mensaje para confirmar que se desea volver a bloquear los temas.

Además, en función del botón pulsado, o mejor dicho, en función del ImageButton pulsado, se llamará a la actividad encargada de crear el tema correspondiente, siendo el imgb1 el botón del tema 1, el imgb2 el del segundo tema, etc.

A continuación, la Figura 3.66 muestra el diagrama de clase de ClaseGeneradora, que se encargará de ser la base del primer tema.

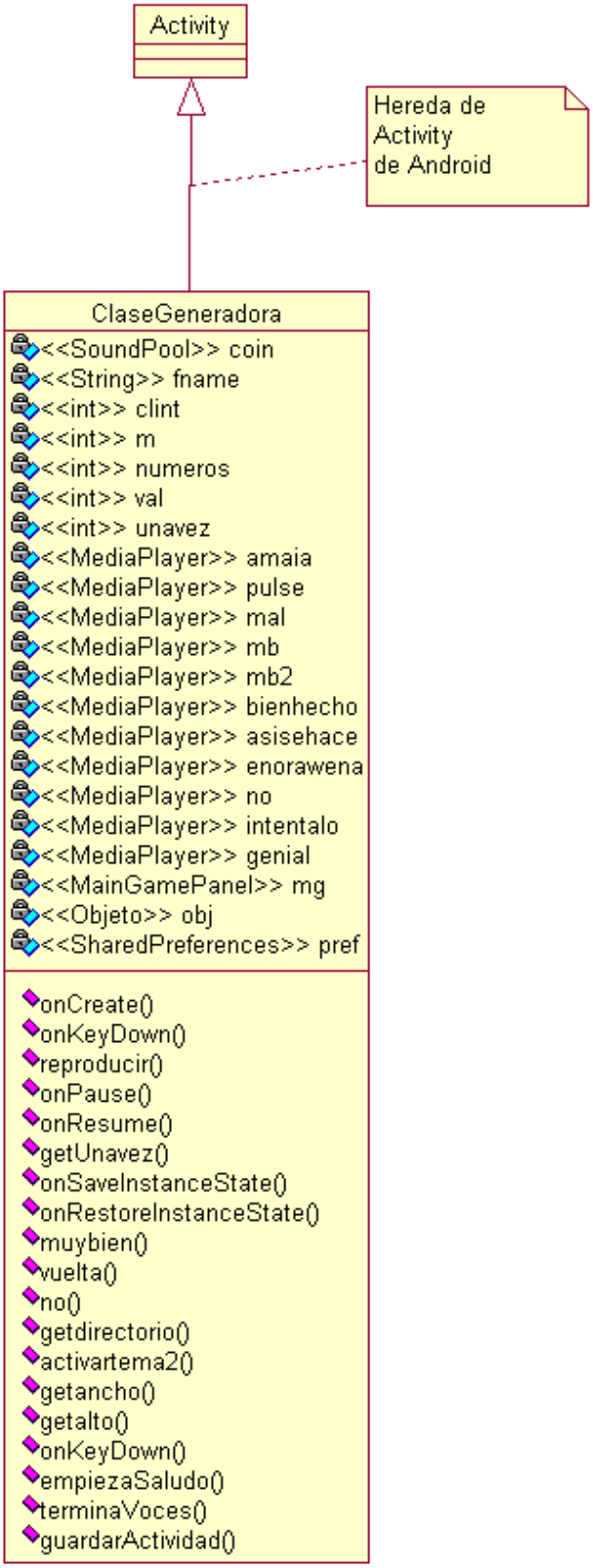


Figura 3.66

La clase ClaseGeneradora, es la encargada de generar el nivel de juego de las monedas con el dependiente. La clase GeneraBilletes, está basada en esta clase. Ambas comparten la misma estructura de clase, pero ha sido necesario realizarlas por separado ya que se emplean distintos objetos para su invocación y construcción. Aunque se podían haber diseñado de manera conjunta, a la hora del tratamiento de las mismas en Android, y de todas las pruebas que se han realizado a lo largo del desarrollo del proyecto, las he creado de manera separada por sencillez.

La clase anterior se encargará de crear una clase de tipo SurfaceView, que me permite una mayor edición de todos los objetos dibujados en pantalla. Esta clase de Android se podría explicar de manera coloquial como si se tratase de un cuaderno de dibujo, es decir, tu le vas indicando dónde dibujar, qué dibujar y cómo dibujarlo.

La clase SurfaceView es la que más libertad me ha permitido tener a la hora de implementar todos los diseños que se han mostrado anteriormente. Sin embargo, el problema de esta clase es que se debe implementar prácticamente en su totalidad, es decir, si yo quiero que un dibujo funcione como un botón, debo detectar su posición en pantalla y asignarle unas acciones a realizar si se pulsa en esa región de la pantalla. Además, se precisa de un hilo de ejecución interno, que ha ocasionado problemas debido a que en cuanto comienza a ejecutarse, el resto del sistema operativo considera todo lo mostrado en pantalla como una actividad independiente del resto de la aplicación. Es decir, si en cualquier momento del juego pulso el botón “Home” del terminal, la actividad debería entrar en el método “onPause” implementado dentro de la clase “Activity” de Android pero no sucede así con la clase “SurfaceView”. Más adelante, en la implementación, veremos cómo se ha solucionado este grave problema.

Este tipo de clase, me permite mayor libertad de movimientos, que he aprovechado para dotar de movimiento a los objetos en pantalla.

He elegido esta clase principalmente por mis ganas de aprendizaje, ya que, como bien se ha mencionado antes, se comporta como un cuaderno de dibujo en el que dispones de más libertad de diseño, sin embargo se podría haber empleado clases más sencillas para lograr efectos similares a los finalmente mostrados en la aplicación, como por ejemplo la clase View (que es la clase más genérica que permite mostrar dibujos en pantalla). Gracias a esta clase se podrían crear juegos con mayor detalle gráfico y dinamismo, y esa es la principal razón que me convenció para usar esta clase y no otra.

Finalmente, si se precisa que nuestra aplicación tenga gráficos 3D, Android también dispone de la librería OpenGL, y de su motor gráfico, pero al tratarse de un juego sencillo gráficamente, buscando en todo momento su simplicidad de uso, no se ha empleado este tipo de desarrollo.

A continuación, la Figura 3.67 muestra la clase MainGamePanel, en la que se basa también MainBilletePanel (la primera representa el juego de las monedas y la segunda de los billetes) de tipo SurfaceView:

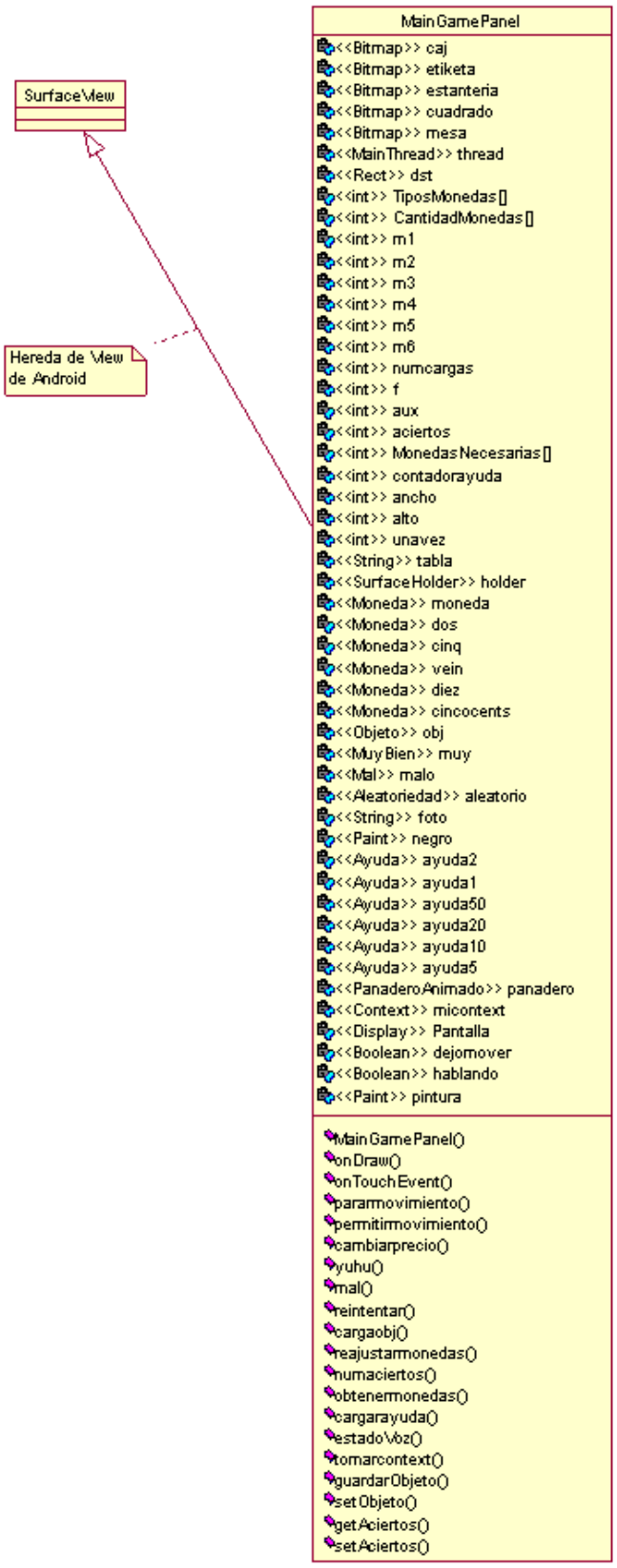


Figura 3.67

La siguiente muestra el diagrama de clases correspondientes a los hilos necesarios para cada SurfaceView. Todos los hilos siguen esta estructura de modo que se muestra el perteneciente al primer tema y servirá para definir el resto.

La clase a mostrar es MainThread (Figura 3.68) que hereda de Thread.

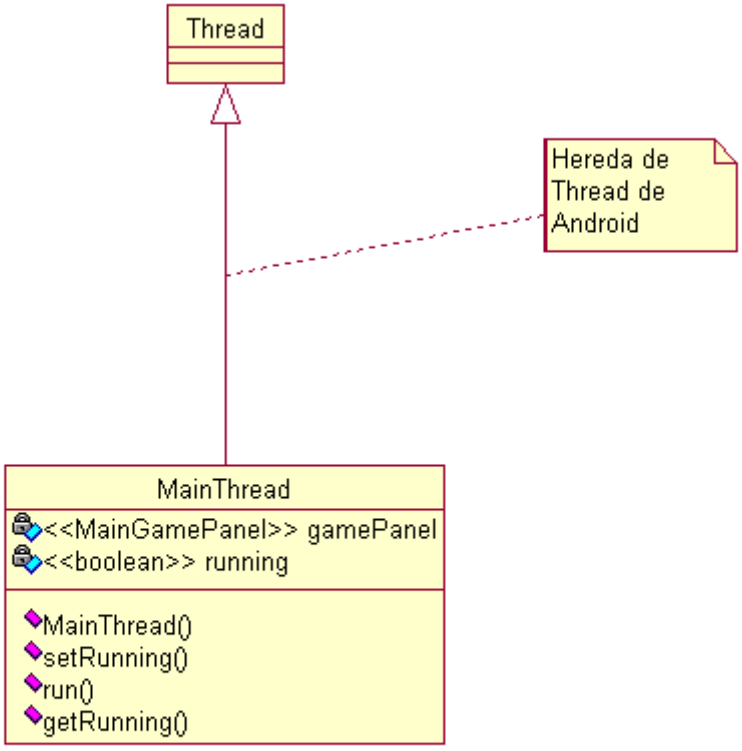


Figura 3.68

Ahora vamos a analizar GeneraMeLLega, que es la clase encargada de generar el tema 2. Con clase y la clase GeneraMeLLegaBilletes sucede como con las dos anteriores. La Figura 3.69 representa el esquema de estas clases:

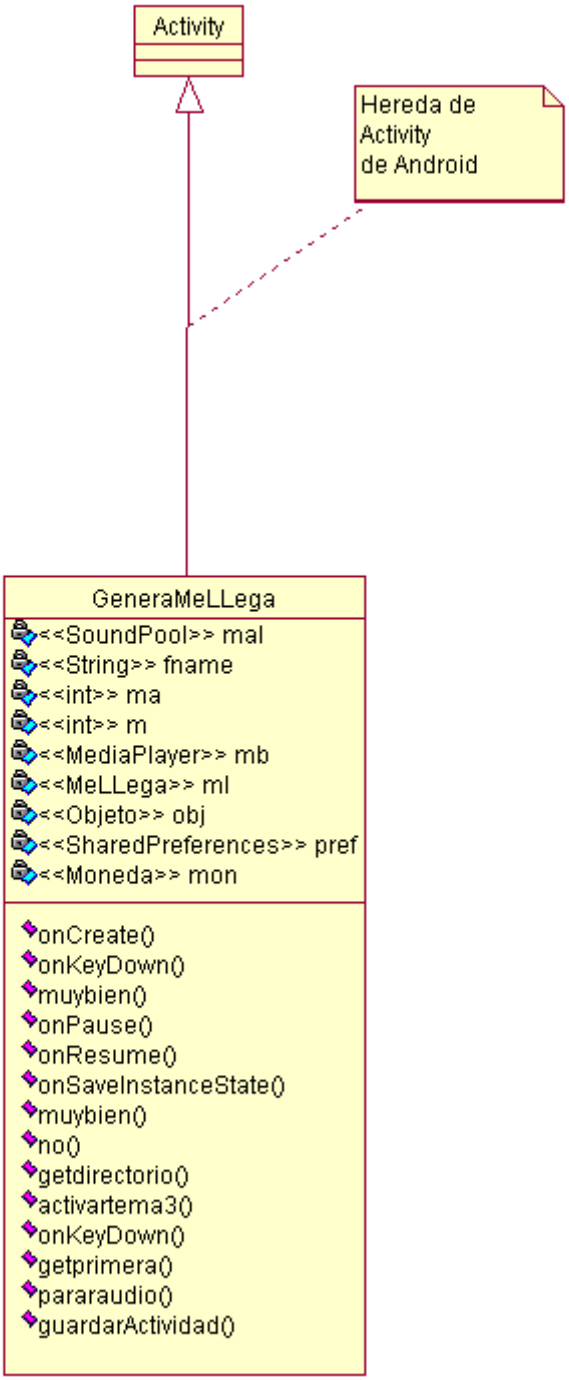


Figura 3.69

A continuación, se va a mostra el SurfaceView que es llamado con la clase anterior (Figura 3.70):

Figura 3.70

Finalmente, vamos a mostrar el diagrama de clases de la clase encargada de generar el último tema (Figura 3.71)

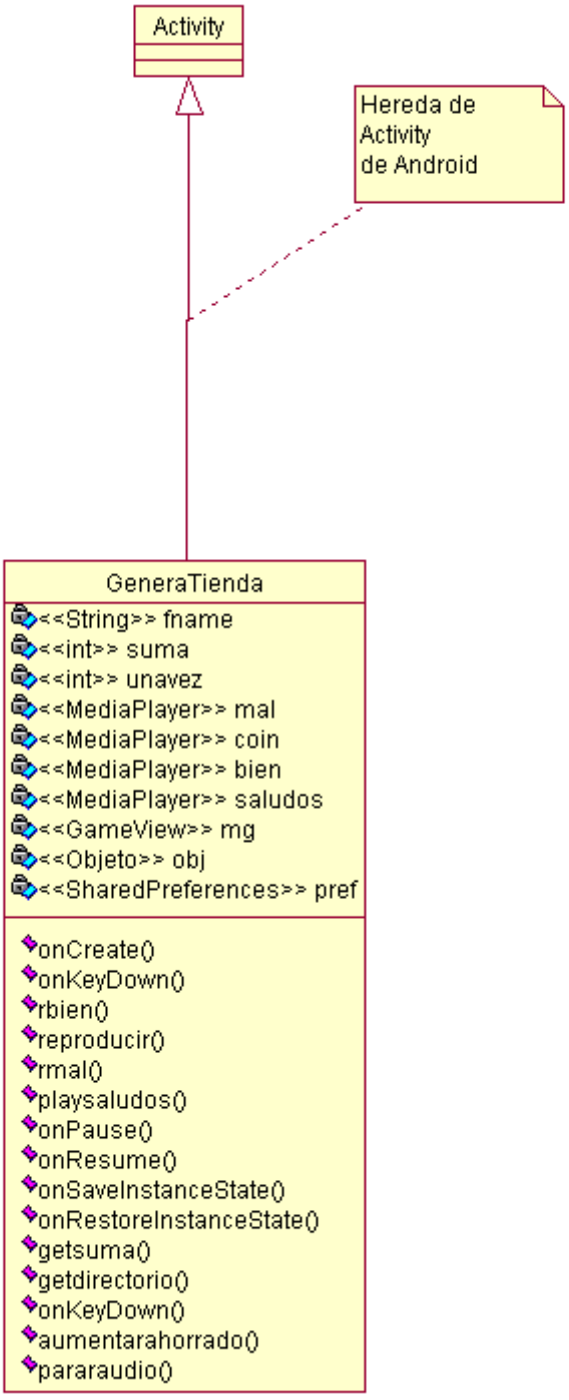


Figura 3.71

Que llamará a un SurfaceView como el mostrado en la Figura 3.72, que hace referencia a la clase GameView:

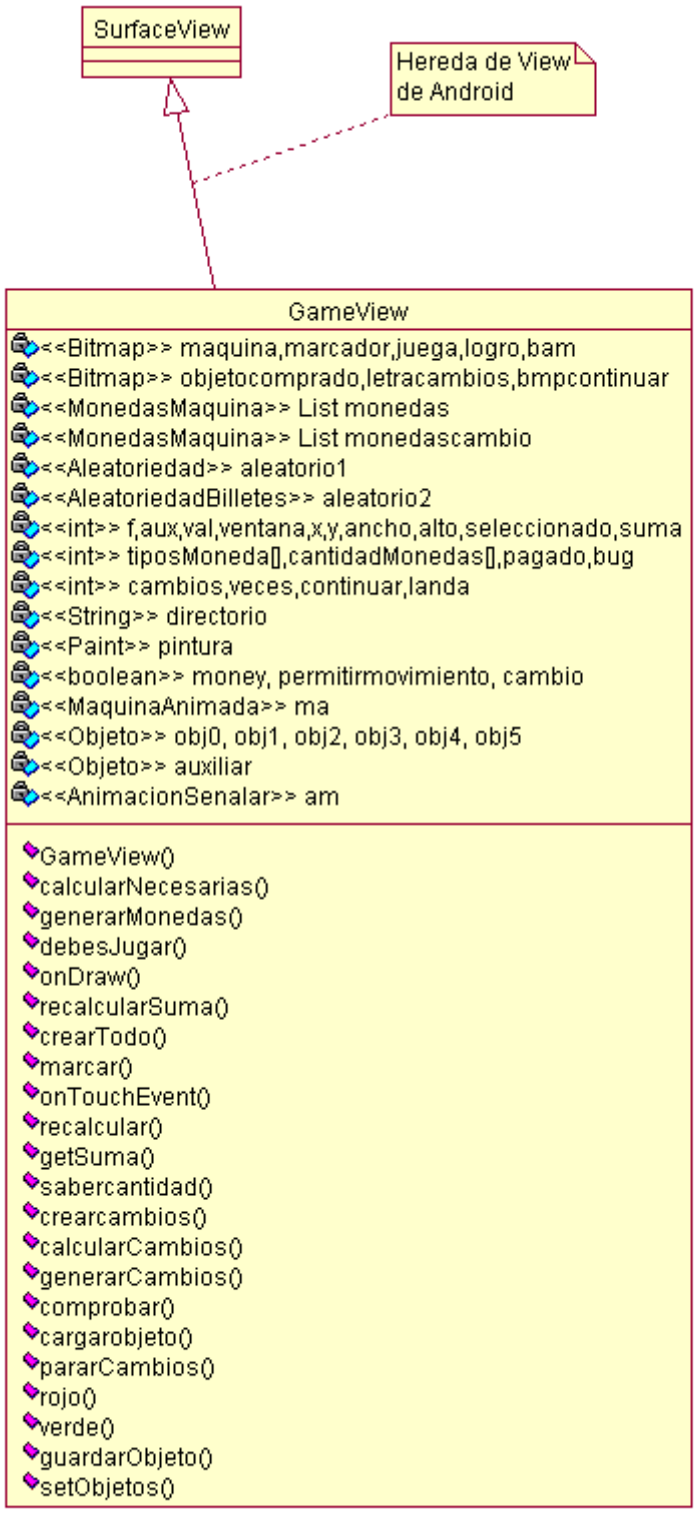


Figura 3.72

Ahora nos queda ver los diagramas de las clases que generan la animación del premio. Por último veremos las clases que crean los objetos, monedas y billetes y diversas animaciones.

La Figura 3.73 hace referencia al diagrama de clases de la clase Cerdito, que se encargará de llamar a la clase Hucha de tipo View.

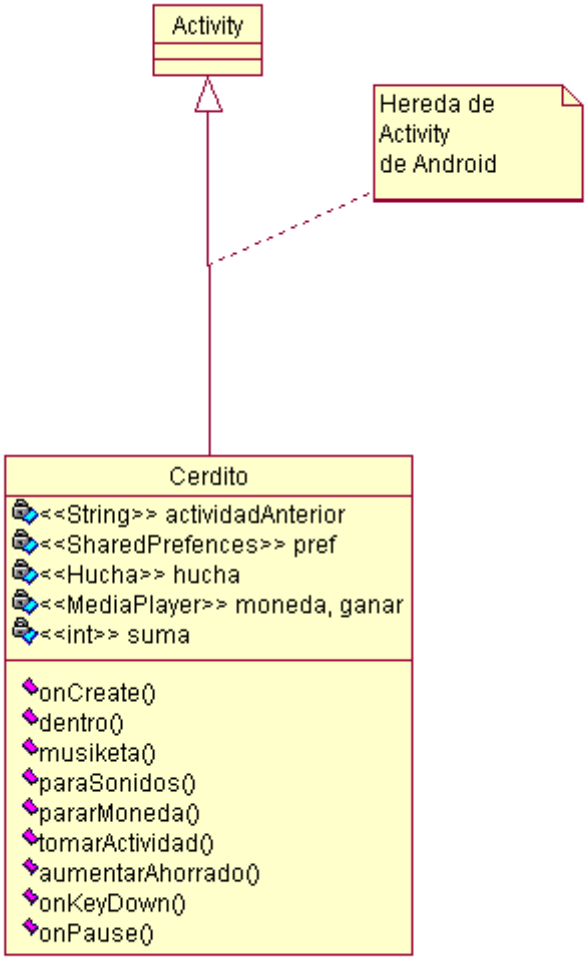


Figura 3.73

La Figura 3.74 muestra el diagrama de la clase generada mediante la clase Cerdito:

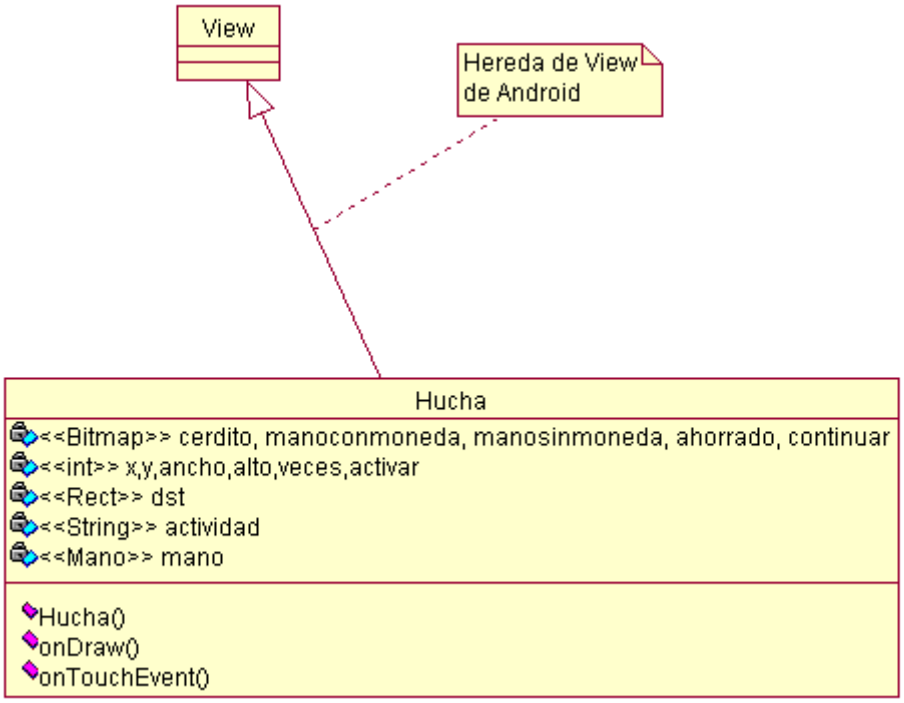


Figura 3.74

A continuación vamos a analizar el resto de clases, necesarias para crear las monedas, billetes, artículos y animaciones. En primer lugar la clase Moneda (Figura 3.75).

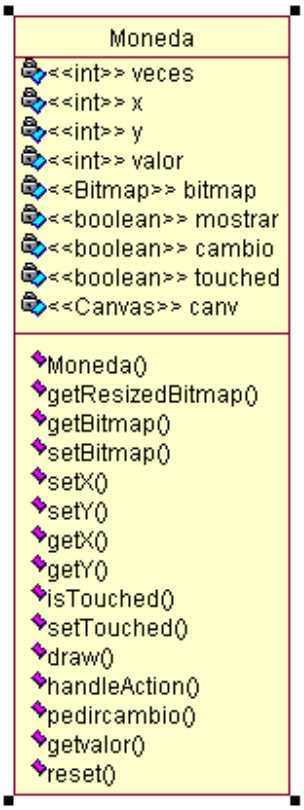


Figura 3.75

Esta clase tiene la misma estructura para los billetes, de modo que no es necesario mostrarla también.

Ahora vamos a ver las clases encargadas de generar los objetos y de generarlos. Estas clases son Aleatoriedad (crea se encarga de indicar qué objeto se crea) y Objeto (que contiene todos los atributos de los mismos). Se pueden observar ambas clases en la Figura 3.76.

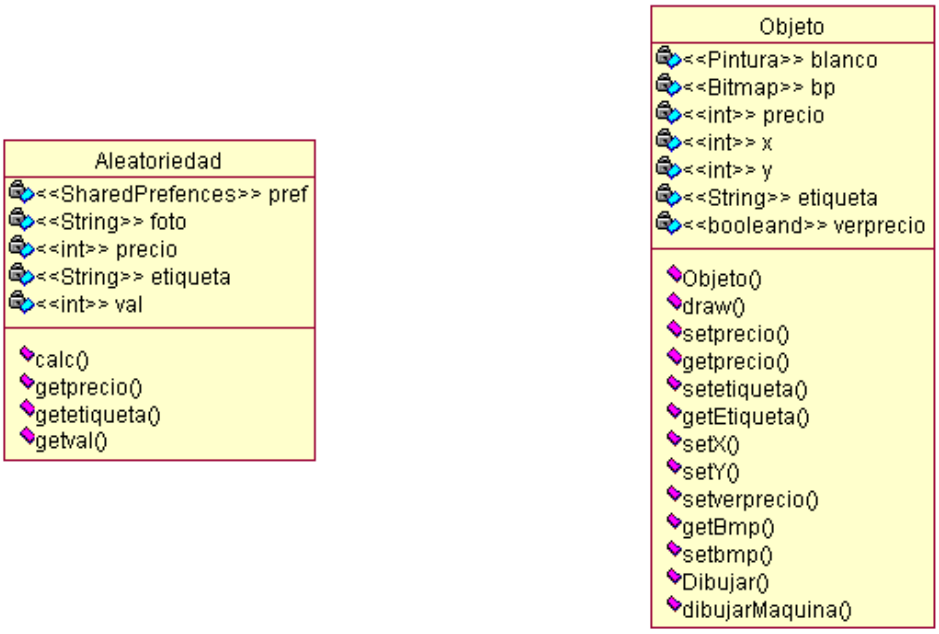


Figura 3.76

Las clases que quedan por mostrar, son clases encargadas de dotar a la aplicación de movimiento, es decir, de animarla. Vamos a mostrar las más completas, que son PanaderoAnimado y NinoAnimado (que se encargan de que el dependiente hable y el niño mueva los ojos, Figura 3.77), MuyBien y Mal (que muestran temporalmente una imagen en pantalla cuando se realiza una actividad bien o mal, Figura 3.78), MaquinaAnimada (encargada de cambiar de color de letrero en función de si disponemos de suficiente cantidad de dinero para pagar un artículo, Figura 3.79) y Ayuda (gracias a la cual mostramos unas señalizaciones a modo de ayuda cuando se falla consecutivamente en el tema 1 y en el tema 3, Figura 3.80).

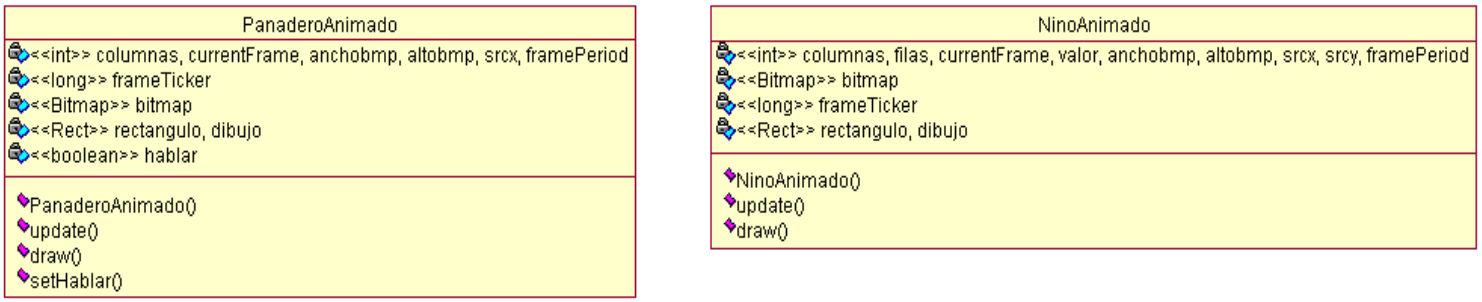


Figura 3.77

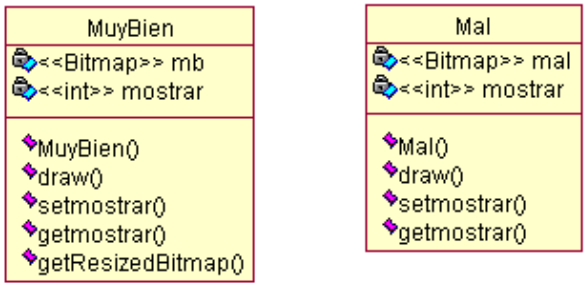


Figura 3.78

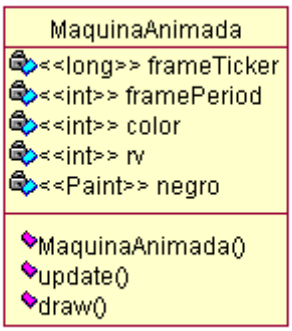


Figura 3.79

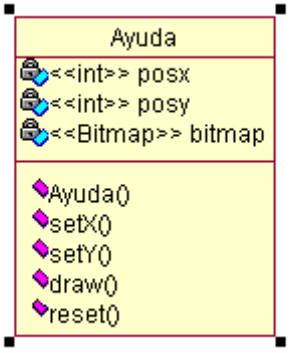


Figura 3.80

4.-Implementación

4.1 Plataforma de desarrollo

La plataforma elegida para el desarrollo de esta aplicación a sido el sistema operativo Android. En este caso se han optado por diversas versiones del sistema operativo, desde Android 2.2 – Froyo hasta el actual, Android 4.1 – Jelly Bean, por los motivos explicados al comienzo de este documento.

Para desarrollar toda la aplicación ha sido necesaria la herramienta Eclipse, debido a que para programar para Android es requisito fundamental.

Esta herramienta ha permitido disponer de los SDK (conjunto de herramientas y programas de desarrollo para crear aplicaciones) de las versiones de Android, así como la posibilidad de emplear emuladores en las fases iniciales de desarrollo.

Se ha elegido la última versión de Eclipse IDE for Java EE Developers [8], debido a las ayudas que aporta y por ser el más completo y actualizado. Además, el código de programación empleado para Android es JAVA, de modo que disponer de un conjunto de herramientas enfocadas a dicho lenguaje es lo más acertado.

Este programa nos da la posibilidad de compilar el código y generar los apk, que instalaremos en los dispositivos de prueba.

También da la posibilidad de probar las aplicaciones en dispositivos físicos y no sólo en los emuladores. Esto permite, por ejemplo, comprobar si los accesos a la memoria interna del dispositivo son correctas, si funcionan características de algunos dispositivos, como puede ser la vibración, que en mi caso la empleo para entrar en el último tema y que únicamente funciona en Smartphones, ya que las tablets no disponen de estos dispositivos.

Además, permite ejecutar el código mediante un Debugger, que nos permitirá encontrar fallos que a priori, no somos capaces de controlar a lo largo del hilo de ejecución.

En mi caso al tratarse de un juego, es muy importante poder comprobar el estado de las imágenes a mostrar en el propio dispositivo al que va destinada. También facilita la incorporación de nuevas imágenes y contenido multimedia (simplemente es necesario arrastrarlos hasta la carpeta que deseemos).

Por último, la programación con esta clase de programas es mucho más eficaz ya que completa las estructuras que indiquemos, asesora constantemente al programador si está empleado métodos ya definidos por el SDK del sistema al que va destinado, y nos indica los errores desde la fase de desarrollo, por ejemplo incompatibilidades de tipo entre variables, sin necesidad de realizar pruebas para ello.

Por otro lado, para crear y modificar todas las imágenes se ha empleado el programa de edición de imagen Photoshop en su versión 3, debido a que disponíamos de la misma de manera portable. Para futuros desarrollos que precisen de imagen será recomendable disponer de la última versión, pero para realizar este proyecto, en el cual el diseño de todos los ítems a mostrar en pantalla ha sido sencillo, no ha sido necesario.

Además, para la edición de Audio se ha empleado Audacity y Hi-Q MP3 Recorder.

Hi-Q MP3 Recorder se trata de una aplicación para Android, que dispone de dos versiones (Lite y de pago) que ha permitido grabar en el dispositivo la voz que sonará a lo largo de la aplicación para guiar al usuario. Una vez lograda una buena grabación, se modificaba con Audacitiy, que permite la eliminación de ruido, aplicar efectos de sonido y modificar el tono de la voz entre otras muchas funcionalidades.

Estos programas han sido más que suficientes para lograr nuestro propósito en la aplicación. En caso de necesitar mejores efectos acústicos será necesario disponer de otras herramientas más potentes.

4.2 Funcionamiento general

A continuación vamos a desglosar las partes más importantes y fundamentales del código empleado en el desarrollo, y que anteriormente se indicó que explicaríamos.

Comenzaremos con la clase MainActivity.

Esta clase nos genera la actividad por primera vez. Además, se encargará de generar la animación de la pantalla principal. Muchas de estas acciones las realiza gracias a los métodos implementados en Android. En caso de considerar importantes su función, también serán descritos en este apartado.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Si queremos poner que la aplicacion funcione a pantalla completa lo
    // que debemos hacer es lo siguiente
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

    // Vamos a activar el audio de los botones fisicos dentro de la
    // aplicacion
    setVolumeControlStream(AudioManager.STREAM_MUSIC);

    // quiero que la pantalla esté encendida siempre que mi aplicación se
    // esté ejecutando de modo que activo el keep screen on
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    intro = MediaPlayer.create(this, R.raw.intro);
    intro.start();
```

Figura 4.1

Observando los comentarios de la Figura 4.1, se puede aperciar como al crear la clase se llamará a este método y ajustaremos la aplicación a pantalla completa y sin título. También activaremos los botones de audio y haremos que la pantalla permanezca encendida siempre que estemos en esta pantalla.

En la figura 4.2 se puede apreciar el código necesario para generar el mensaje que obliga al usuario a elegir si ver o no el tutorial. Este mensaje ha sido necesario implementarlo para motivar al usuario a ver el tutorial y enseñarle a usar la aplicación, en caso de que fuera necesario. Además, tras realizar la elección, se guardará el estado de no volver a mostrar dicho mensaje, ya que como sucede en todos los juegos, lo normal es mostrarlo una vez al principio y no más, debido a que esto provocaría una situación repetitiva y monótona. También se aprecia las opciones a realizar en caso de responder si (en este caso cargaríamos el tutorial) o si se selecciona que no.

```
// Vamos a acceder a las preferencias para saber si es la primera vez
// que ejecutamos la aplicación o no y mostrar si queremos ver el
// tutorial
prefs = getSharedPreferences("Superados", Context.MODE_PRIVATE);
if (prefs.getBoolean("Tutorial", true)) {
    ad = new AlertDialog.Builder(this);
    ad.setTitle("Ver Tutorial");
    ad.setMessage("Es la primera vez que ejecuta la aplicación.¿Desea ver el tutorial ahora?");
    ad.setNegativeButton("No", new DialogInterface.OnClickListener() {
        // En esta parte deberemos introducir todo el código que
        // queremos que se genere de manera automática para mostrar las
        // opciones

        public void onClick(DialogInterface dialog, int which) {
            // Aqui lo que sucederá es que marcaremos como no el
            // tutorial y mostraremos la pantalla normal
            SharedPreferences.Editor editor = prefs.edit();
            editor.putBoolean("Tutorial", false);
            //Finalizamos las preferencias
            editor.putInt("aux", 999999);
            editor.commit();
        }
    });
    ad.setPositiveButton("Si", new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {
            // Aqui lo que haremos sera cambiar el valor de que ya hemos
            // visto el tutorial
            // y generar el tutorial para verlo
            SharedPreferences.Editor editor = prefs.edit();
            editor.putBoolean("Tutorial", false);
            //Finalizamos las preferencias
            editor.putInt("aux", 999999);
            editor.commit();

            // Generamos el tutorial
            try {
                cTuto = Class.forName("com.example.juego.Tutorial");
                Intent i = new Intent(MainActivity.this, cTuto);
                finish();
                startActivity(i);
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    });
    ad.show();
}
;
```

Figura 4.2

Debido a que esta clase genera audio será necesario que la música de la aplicación finalice cuando se salga del juego o se seleccione alguna opción de las disponibles. Para realizar estas tareas se debe modificar los métodos onDestroy(), onStop() y onPause() de la actividad. En la figura 4.3 se muestran dichos métodos y el método creado para detener la música en caso de ser necesario.

```
@Override
protected void onDestroy() {
    // TODO Auto-generated method stub
    super.onDestroy();
    pararMusica();
    finish();
    System.exit(0);
}

@Override
protected void onPause() {
    // TODO Auto-generated method stub
    super.onPause();
    pararMusica();
    finish();
    System.exit(0);
}

@Override
protected void onStop() {
    // TODO Auto-generated method stub
    super.onStop();
    pararMusica();
}

public static void pararMusica(){
    //pararemos la música
    if(intro.isPlaying()){
        intro.stop();
    }
}
```

Figura 4.3

Para cambiar lo mostrado en pantalla y que se pueda apreciar al niño y todos los ítems en movimientos de los que dispone, se debe realizar una llamada a Portada, que al generarla cambiaremos la vista actual a ésta última. Se puede apreciar en la figura 4.4.

```
// Vamos a crear la pantalla inicial, que cargaremos desde un Portada

setContentView(new Portada(this));
```

Figura 4.4

Ahora vamos a ver las peculiaridades de la clase Portada. Como se trata de una clase de tipo View, debemos implementar el código necesario para mostrar las imágenes que deseemos en pantalla.

```
public Portada(Context context) {
    super(context);

    // Como queremos conocer el tamaño de la pantalla será necesario
    // analizarla cuando queramos cargar cada nivel
    WindowManager wm = (WindowManager) context
            .getSystemService(Context.WINDOW_SERVICE);
    Display display = wm.getDefaultDisplay();

    ancho = display.getWidth();
    alto = display.getHeight();

    // aqui crearemos la portada con el chico y todos los bocadillos para
    // darle otro toque a la aplicacion
    nino = new NinoAnimado(BitmapFactory.decodeResource(
            getResources(), R.drawable.ninoanimado), 4);
    comenzar = BitmapFactory.decodeResource(getResources(),
            R.drawable.comenzar);
    salir = BitmapFactory.decodeResource(getResources(), R.drawable.salir);
    tutorial = BitmapFactory.decodeResource(getResources(),
            R.drawable.tutorial);
    fondo = BitmapFactory.decodeResource(getResources(), R.drawable.park);
    cielo = BitmapFactory.decodeResource(getResources(), R.drawable.cielo);
    globo = BitmapFactory.decodeResource(getResources(), R.drawable.globo);
    avion = BitmapFactory.decodeResource(getResources(), R.drawable.avion);
    //Estas serán las posiciones del globo y del avion
    gx = ancho + ancho/10;
    ax = 0-ancho/10;
    gy = alto*1/10;
    ay = 0;
    dst = new Rect();
}
```

Figura 4.5

En la figura 4.5 se muestra el código ejecutado cuando se crea esta clase. En ella se puede apreciar como tomamos el ancho y alto de la pantalla mediante display.getWidth() o display.getHeight(). Se puede observar claramente en la imagen como se encuentran tachados. Esto es así debido a que Google ha considerado obsoletos porque van a implementar algún nuevo método. Dicho método va a ser sustituido por Point(), que contará con dos valores que representarán ancho y alto. Sin embargo, en las versiones actuales, hasta Android 4.1, aun no se ha implementado, de modo que aunque aparezcan visualmente como obsoletos, son los únicos métodos disponibles que se pueden emplear para saber las dimensiones de la pantalla.

Estas dimensiones son muy importantes para el desarrollo de todo el juego. Este desarrollo ha venido impuesto por la idea de que nuestra aplicación debe funcionar tanto en Smartphones como en tablets. Para ello es necesario redimensionar las imágenes y todos los objetos a mostrar en pantalla, en función de la resolución de cada dispositivo, cuyo planteamiento se explicó anteriormente en la fase de diseño.

Toda la metodología de ajuste, ha sido calculada en el método onDraw(), donde como se puede apreciar en la Figura 4.6, definimos las coordenas de la variable dst y dibujamos en su interior el objeto deseado. El código señalado hace referencia al dibujo de comenzar situado sobre el niño.

```
@Override
protected void onDraw(Canvas canvas) {
    // Aqui vamos a dibujar todas las imagenes cargadas
    canvas.drawARGB(100, 200, 200, 255);
    // definimos el espacio del niño
    dst.set(0, 0, canvas.getWidth(), canvas.getHeight() * 5 / 16);
    canvas.drawBitmap(cielo, null, dst, null);
    dst.set(0, canvas.getHeight() * 5 / 16, canvas.getWidth(),
            canvas.getHeight());
    canvas.drawBitmap(fondo, null, dst, null);
    dst.set(gx,gy,gx+canvas.getWidth()*2/10,gy+canvas.getHeight()*2/16);
    canvas.drawBitmap(globo, null, dst, null);
    dst.set(ax, ay, ax+canvas.getWidth()*2/10, ay+canvas.getHeight()/16);
    canvas.drawBitmap(avion, null, dst, null);
    cambiarcoordenadas(canvas);
    nino.draw(canvas);
    dst.set(0, canvas.getHeight() * 3 / 16, canvas.getWidth() * 3 / 10,
            canvas.getHeight() * 7 / 16);
    canvas.drawBitmap(tutorial, null, dst, null);
    dst.set(canvas.getWidth() * 3 / 10, canvas.getHeight() * 2 / 16,
            canvas.getWidth() * 7 / 10, canvas.getHeight() * 7 / 16);
    canvas.drawBitmap(comenzar, null, dst, null);
    dst.set(canvas.getWidth() * 7 / 10, canvas.getHeight() * 3 / 16,
            canvas.getWidth(), canvas.getHeight() * 7 / 16);
    canvas.drawBitmap(salir, null, dst, null);
    invalidate();
}
```

Figura 4.6

En el código mostrado sobre estas líneas, se puede apreciar que si deseamos otorgar dinamismo a la imagen, se puede conseguir definiendo un área variable en la que mostraremos tanto el avión como el globo, y que modificaremos mediante cambiarcoordenadas(), como se aprecia en la figura 4.7.

```
public void cambiarcoordenadas(Canvas canvas){
    if(gx<0-canvas.getWidth()/10){
        gx = ancho + ancho/10;
    }else{
        gx--;
    }
    if(ax<0-canvas.getWidth()/10){
        ax = 0 - ancho/10;
    }else{
        ax+=2;
    }
}
```

Figura 4.7

Por otro lado, se debe implementar la funcionalidad de pulsar en cada una de las opciones disponibles, bien sea tutorial, comenzar o salir (Figura 4.8).

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    // Veremos qué es lo que hemos pulsado
    if (MotionEvent.ACTION_DOWN == event.getAction()) {
        if ((event.getX() > ancho * 3 / 10)
                && (event.getX() < ancho * 7 / 10)) {
            if ((event.getY() > alto * 2 / 16)
                    && (event.getY() < alto * 6 / 16)) {
                //Aqui habremos pulsado comenzar
                try {
                    Intent i = new Intent("com.example.juego.TEMAS");
                    // MainActivity.pararMusica();
                    ((Activity) getContext()).finish();
                    ((Activity) getContext()).startActivity(i);
                } catch (Exception e) {
                    System.out.println("No hago nada");
                }
            }
        }
        if ((event.getX() > 0)
                && (event.getX() < ancho * 3 / 10)) {
            if ((event.getY() > alto * 3 / 16)
                    && (event.getY() < alto * 6 / 16)) {
                //Aqui habremos pulsado Tutorial
                try {
                    Intent i = new Intent("com.example.juego.TUTORIAL");
                    // MainActivity.pararMusica();
                    ((Activity) getContext()).finish();
                    ((Activity) getContext()).startActivity(i);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
        if ((event.getX() > ancho * 7 / 10)
                && (event.getX() < ancho)) {
            if ((event.getY() > alto * 3 / 16)
                    && (event.getY() < alto * 6 / 16)) {
                //Aqui habremos pulsado salir
                try {
                    // MainActivity.pararMusica();
                    ((Activity) getContext()).finish();
                    System.exit(0);
                } catch (Exception e) {
                    System.out.println("No hago nada");
                }
            }
        }
```

Figura 4.8

A continuación se va a analizar la clase que generará la pantalla de tutorial. En dicha clase se carga el audio que servirá de guía para el mismo (Figura 4.9).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);

    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCRE
    amaia = MediaPlayer.create(this, R.raw.tutorialparteuno);
    pulse = MediaPlayer.create(this, R.raw.tutorialpartedos);
    //Vamos a invocar la clase que hemos empleado
    muestra = new Miclase(this);
    parteuno();
    setContentView(muestra);
}
```

Figura 4.9

Como se puede apreciar, también creamos una instancia a Miclase y asignamos la vista actual a la misma. A lo largo de todas las clases generadoras, se puede apreciar el código necesario para que se muestra a pantalla completa y sin título de aplicación alguno, situación totalmente lógica al tratarse de un juego.

De la Figura 4.10, cabe destacar la implementación del código necesario para realizar tareas diferentes a las implementadas de serie al pulsar el botón back. Como se puede apreciar en la Figura 4.11 lo que se hace es modificar la unción de volver para llamar a la clase que genera la Portada. Esto es necesario debido a que cada vez que se selecciona alguna opción de cambio de pantalla, lo que se hace es finalizar la actividad actual y generar una nueva actividad con la pantalla nueva que deseemos. De este modo nos aseguramos de controlar perfectamente la pila de ejecución de toda la aplicación. A lo largo de todas las clases, implementamos este método para sustituir su función normal, debido a que los hilos de ejecución necesarios para dotar de dinamismo a la aplicación nos daban problemas al quedarse sin finalizar haciendo que la aplicación se cerrase inesperadamente. De esta manera nos aseguramos que se finalizan correctamente y logramos mayor estabilidad.

En la Figura 4.11 también se puede apreciar métodos empleados para controlar el audio a lo largo del tutorial. Dichos métodos serán llamados en los momentos correspondientes desde la clase Miclase, que se encargará de dibujar en pantalla lo necesario y, en función de si es el momento oportuno, hará que suenen las grabaciones pertinentes.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        try {
            Class pantallaInicial = Class
                    .forName("com.example.juego.MainActivity");
            pararAudio();
            Intent in = new Intent(Tutorial.this, pantallaInicial);
            finish();
            startActivity(in);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    return super.onKeyDown(keyCode, event);
}

public static void parteuno(){
    amaia.start();
}

public static void partedos(){
    pulse.start();
}

public static void pararAudio(){
    if (amaia.isPlaying()){
        amaia.stop();
    }
    if(pulse.isPlaying()){
        pulse.stop();
    }
}
```

Figura 4.11

A continuación, vamos a proceder con el análisis de la clase Miclase, la cual es generada como pantalla por la clase anterior. Dado que se trata de una clase de tipo View, el método onCreate() sigue el esquema de la clase Portada. Por otro lado, el método onDraw() es más complejo, ya que realizamos el movimiento de dos monedas y mostramos un mensaje al finalizar dicha animación, todo ello acompañado de un control sobre las voces que suenan en cada momento. Por estos motivos vamos a observar éste método (Figura 4.12).

```
@Override
protected void onDraw(Canvas canvas) {
    // TODO Auto-generated method stub
    super.onDraw(canvas);
    canvas.drawColor(Color.rgb(130, 200, 255));
    dst.set(0, 0-canvas.getHeight()/16, canvas.getWidth(),
            canvas.getHeight() * 6 / 16);
    canvas.drawBitmap(mesa, null, dst, null);
    dst.set(canvas.getWidth() / 2, canvas.getHeight() / 16,
            canvas.getWidth() * 9 / 10, canvas.getHeight() * 5 / 16);
    canvas.drawBitmap(caj, null, dst, null);
    dst.set(canvas.getWidth() / 10, canvas.getHeight() / 16,
            canvas.getWidth() / 2, canvas.getHeight() * 5 / 16);
    canvas.drawBitmap(balon, null, dst, null);
    dst.set(0, canvas.getHeight() * 5 / 16, canvas.getWidth() * 4 / 10,
            canvas.getHeight() * 7 / 16);
    canvas.drawBitmap(ticket, null, dst, null);
    canvas.drawText("  3.00", canvas.getWidth() * 2 / 10,
            canvas.getHeight() * 6 / 16, pintura);
    if (moveruno == 1) {
        if (mx < canvas.getWidth() * 6 / 10) {
            mx += 2;
            dst.set(mx, my, mx + canvas.getWidth() * 3 / 10,
                    my + canvas.getHeight() * 3 / 16);
            canvas.drawBitmap(uno, null, dst, null);
        } else {
            if (my > canvas.getHeight() * 2 / 16) {
                my -= 2;
                dst.set(mx, my, mx + canvas.getWidth() * 3 / 10, my
                        + canvas.getHeight() * 3 / 16);
                canvas.drawBitmap(uno, null, dst, null);
            }else{
                moveruno=0;
            }
        }
        dst.set(dosx, dosy, dosx+canvas.getWidth()*3/10, dosy + canvas.getHeight()*3/16);
        canvas.drawBitmap(dos, null, dst, null);
    }else{
        if(dosx < canvas.getWidth()*6/10){
            dosx+=2;
            dst.set(dosx, dosy, dosx + canvas.getWidth() * 3 / 10,
                    dosy + canvas.getHeight() * 3 / 16);
            canvas.drawBitmap(dos, null, dst, null);
        }else{
            if(dosy > canvas.getHeight() * 2 / 16){
                dosy-=2;
                dst.set(dosx, dosy, dosx + canvas.getWidth() * 3 / 10,
                        dosy + canvas.getHeight() * 3 / 16);
                canvas.drawBitmap(dos, null, dst, null);
            }else{
                dst.set(canvas.getWidth()/2-pulse.getWidth()/2, canvas.getHeight()/2, canvas.getWidth()/2+pulse.getWidth()/2, canvas.getHeight()*15/16);
                canvas.drawBitmap(pulse, null,dst, null);
                liberar=true;
                if(veces==0){
                    Tutorial.partedos();
                    veces++;
                }
            }
        }
    }
    invalidate();
}
```

Figura 4.12

En la imagen anterior, podemos observar en rojo el recorrido que vamos calculando para la moneda de un euro y en azul el de la moneda de dos euros, cuyo movimiento se produce cuando a finalizado la de uno. También se centra en pantalla un mensaje, el cual está creado como imagen para poder centrarlo, como se aprecia en el recuadro verde. En dicho recuadro se ve como si ha finalizado todo el movimiento y se está mostrando el mensaje, reproducimos la segunda parte del audio, que nos indica de manera acústica, que debemos pulsar en pantalla.

En la figura 4.13 se aprecia el código que se ejecuta al pulsar en pantalla cuando ha finalizado la animación. Dicho código realizará la finalización de la actividad tutorial y comenzará la actividad Temas, como ya habíamos indicado anteriormente.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    // Este método nos va a servir para que al pulsar la pantalla, generemos
    // los temas. Esta correctamente situado ya que anteriormente habia
    // probado con onDestroy() pero parecía que la aplicacion se cerraba y
    // se volvia a abrir, provocando una mala transicion entre actividades
    if (liberar == true) {
        if (MotionEvent.ACTION_DOWN == event.getAction()) {
            Intent i = new Intent("com.example.juego.TEMAS");
            Tutorial.pararAudio();
            ((Activity) getContext()).finish();
            ((Activity) getContext()).startActivity(i);
        }
    }
    return super.onTouchEvent(event);
}
```

Figura 4.13

Ahora veremos la clase Temas, que enlazando con la imagen anterior, ha sido generada al pulsar en pantalla.

La clase Temas, es la más sencilla de implementar debido a que no contiene dinamismo, y no ha sido necesario mayor diseño que el de un Layout. Sin embargo, dicho layout está creado en XML, código con el cual no estaba muy familiarizado, y que también mostraremos. Ahora la figura 4.14 muestra el código de creación de esta clase.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    // Y hacemos que la pantalla sea full_screen
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

    // Vamos a activar el audio de los botones fisicos dentro de la
    // aplicacion
    setVolumeControlStream(AudioManager.STREAM_MUSIC);

    pulsacion1 = 0;
    setContentView(R.layout.temario);
    imgb1 = (ImageButton) findViewById(R.id.botonTema1);
    imgb2 = (ImageButton) findViewById(R.id.botonTema2);
    imgb3 = (ImageButton) findViewById(R.id.botonTema3);
    imgb4 = (ImageButton) findViewById(R.id.botonTema4);
    imgb5 = (ImageButton) findViewById(R.id.botonTema5);
    resetear = (Button) findViewById(R.id.resetear);

    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD) {
        resetear.setVisibility(1);
        ad = new AlertDialog.Builder(this);
    } else {
        resetear.setVisibility(View.GONE);
    }
```

Figura 4.14

De la Figura 4.14 cabe destacar la comprobación de la versión de Android en la que se está ejecutando la aplicación, es decir, si la versión es Froyo o Gingerbread, uno de los botones no se mostrará, mientras que si es Honeycomb o superior mostraremos en la esquina superior un botón que permitirá resetear los temas. Esto ha sido necesario implementarlo debido a que no todos los dispositivos cuenta con un botón físico específico para mostrar las opciones de la aplicación, especialmente a partir de Android 3.0 debido a que Google decidió cambiar el estilo de Android a partir de dicha versión. Bien es cierto que la mayoría de Smartphones que se siguen diseñando cuentan con dicho botón, sin embargo todos los terminales y tablets desarrolladas por Google (los terminales Nexus) ya no lo incorporan. Por lo tanto cumplimos con las especificaciones de Google sin perder funcionalidad en nuestra aplicación.

```
// Tomamos las preferencias que queremos
prefs = getSharedPreferences("Superados", Context.MODE_PRIVATE);
addListenerOnButton();
imgb1.setImageResource(R.drawable.monedastema1);
if (prefs.getBoolean("nivel2", true)) {
    imgb2.setImageResource(R.drawable.mellegamonedas);
} else {
    imgb2.setImageResource(R.drawable.mellegamonedasblokeado);
}
if (prefs.getBoolean("nivel3", true)) {
    imgb3.setImageResource(R.drawable.billetesdesblokeados);
} else {
    imgb3.setImageResource(R.drawable.billetesblokeado);
}
if (prefs.getBoolean("nivel4", true)) {
    imgb4.setImageResource(R.drawable.mellegabilletes);
} else {
    imgb4.setImageResource(R.drawable.mellegabilletesblokeados);
}
if (prefs.getBoolean("nivel5", true)) {
    imgb5.setImageResource(R.drawable.escaparate);
} else {
    imgb5.setImageResource(R.drawable.escaparateblokeado);
}

SharedPreferences.Editor editor = prefs.edit();
editor.putInt("aux", 999999);
editor.commit();
```

Figura 4.15

En la Figura 15 podemos apreciar el resto del tratamiento que hacemos al cargar la clase de los Temas. En dicho tratamiento lo que comprobamos el qué temas han sido desbloqueados y cuáles no. Para ello accedemos a las preferencias de la aplicación, que emplearemos para guardar el estado de los temas de manera fija en la memoria del dispositivo. Dicha imagen muestra como se asignan o bien la imagen desbloqueada o la que muestra el candado, a cada uno de los botones que se encuentran en pantalla.

A todos los botones hay que implementarles qué funciones deben realizar al pulsarlos. Esto se aprecia en la figura 4.16, donde nos encargamos en función de cada botón, cargar el tema que correspone y finalizar la actividad Temas.

```
private void addListenerOnButton() {

    imgb1.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {
            // TODO Auto-generated method stub
            Toast.makeText(Temas.this, "He pulsado en el primer nivel ",
                    Toast.LENGTH_SHORT).show();
            if (prefs.getBoolean("nivel1", true)) {
                try {
                    Intent i = new Intent(
                            "com.example.juego.CLASEGENERADORA");
                    finish();
                    startActivity(i);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }
        }

    });
    imgb2.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {
            // TODO Auto-generated method stub
            Toast.makeText(Temas.this, "He pulsado en el segundo nivel ",
                    Toast.LENGTH_SHORT).show();
            if (prefs.getBoolean("nivel2", true)) {
                try {
                    Intent i = new Intent("com.example.juego.GENERAMELLEGA");
                    finish();
                    startActivity(i);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }
        }

    });
```

Figura 4.16, con dos de los botones de la clase

En caso de que estemos ejecutando la aplicación en una versión superior a Android 3.0, podremos seleccionar el botón de resetear temas, el cual mostrará en pantalla un mensaje para confirmar nuestra decisión. Se puede apreciar su función y cómo se carga una nueva pantalla de temas en caso de seleccionar la respuesta afirmativa. Esto se debe a que para cambiar la imagen mostrada dentro de estos botones estáticos, se debe finalizar la clase y volverla a crear con las modificaciones guardadas en preferencias (Figura 4.17).

```
ad.setTitle("Resetear");
ad.setMessage("¿Desea tener los temas bloqueados?");
ad.setNegativeButton("No", new DialogInterface.OnClickListener() {

    public void onClick(DialogInterface arg0, int arg1) {
        //Desblowueamos los temas
        SharedPreferences.Editor editor = prefs.edit();
        editor.putBoolean("nivel1", true);
        editor.putBoolean("nivel2", true);
        editor.putBoolean("nivel3", true);
        editor.putBoolean("nivel4", true);
        editor.putBoolean("nivel5", true);
        editor.putBoolean("Tutorial", false);
        editor.commit();
        Intent in = new Intent("com.example.juego.TEMAS");
        startActivity(in);
        finish();
    }
    // Aqui no haremos nada si pulsa en no
});
ad.setPositiveButton("Si", new DialogInterface.OnClickListener() {

    public void onClick(DialogInterface dialog, int which) {
        // Aqui resetearemos los temas en caso de pulsar que si
        SharedPreferences.Editor editor = prefs.edit();
        editor.putBoolean("nivel1", true);
        editor.putBoolean("nivel2", false);
        editor.putBoolean("nivel3", false);
        editor.putBoolean("nivel4", false);
        editor.putBoolean("nivel5", false);
        editor.putBoolean("Tutorial", true);
        editor.commit();
        Intent in = new Intent("com.example.juego.TEMAS");
        startActivity(in);
        finish();
    }
});
ad.show();
```

Figura 4.17

Como hemos mencionado, nuestra aplicación también empleará el botón de menú hasta Android 2.3, de modo que tenemos el código con las opciones mostrado en la figura 4.18. Para dichas opciones, se crea un pequeño layout, en el cual indicamos cada parte del mensaje a mostrar, y llamamos a su visualización cuando lo pulsamos. También vemos las modificaciones de las preferencias, al igual que sucedía anteriormente.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Creamos las opciones del menu
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.activity_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Vamos a tratar lo que debemos hacer en cada caso de los posibles del
    // menu
    switch (item.getItemId()) {
    case R.id.Subsi:
        SharedPreferences.Editor editor = prefs.edit();
        editor.putBoolean("nivel1", true);
        editor.putBoolean("nivel2", false);
        editor.putBoolean("nivel3", false);
        editor.putBoolean("nivel4", false);
        editor.putBoolean("nivel5", false);
        editor.putBoolean("Tutorial", true);
        editor.commit();
        Intent in = new Intent("com.example.juego.TEMAS");
        startActivity(in);
        finish();
        break;
    case R.id.salida:
        finish();
        break;
    }
    return super.onOptionsItemSelected(item);
}
```

Figura 4.18

Como se puede deducir, ya nos hemos encontrado con diversos problemas provenientes de la fragmentación que sufre Android, debido al rápido avance en cuanto a actualizaciones se refiere. Aunque no ha resultado compleja su solución, si se tratase de aplicaciones más grandes y complejas podría suponer un problema para dar soporte a todas las versiones.

Por último, esta clase de nuevo implementa el método onKeyDown(). Este caso es un caso particular, debido a que se ha realizado una modificación con el objetivo de otorgar a la aplicación mayor usabilidad. En la parte señalada se realiza un cálculo de tiempo que nos permite definir que el usuario deba pulsar dos veces seguidas para salir definitivamente de la aplicación. Esta manera de actuar se suele emplear para liberar de carga a las aplicaciones y juegos, ya que no tiene sentido volver a la pantalla principal si lo que se desea es salir. De esta manera logramos que la aplicación sea mucho más ágil cuando se está usando y el usuario no sentirá una aplicación repetitiva.

El cálculo de dichas pulsaciones consecutivas, se realiza controlando que el tiempo entre ambas sea menor de cuatro segundos, y mostrando un mensaje en pantalla que indica qué se debe hacer para salir de la aplicación (Figura 4.19).

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        Toast.makeText(Temas.this, "Pulse de nuevo para salir ",
                Toast.LENGTH_LONG).show();
        Log.d(TAG, "pulsacion" + pulsacion1);
        pulsacion1++;
        if (pulsacion1 == 1) {
            Log.d(TAG, "pulsacion if 1" + pulsacion1);
            tiempo = System.currentTimeMillis();
        }
        if (pulsacion1 == 2) {
            Log.d(TAG, "pulsacion if 2" + pulsacion1);
            tiempo2 = System.currentTimeMillis();
            if ((tiempo2 - tiempo) < 4000) {
                Log.d(TAG, "finish");
                finish();
                System.exit(0);
            } else {
                pulsacion1 = 0;
            }
        }

    }
    // return super.onKeyDown(keyCode, event);
    return false;
}
```

Figura 4.19

Ahora vamos a mostrar la clase encargada de generar el tema 1, ClaseGeneradora. Esta clase y GeneraBilletes siguen las mismas pautas, de modo que analizaremos únicamente una de ellas. Al igual sucederá con las clases MainGamePanel y MainBilletePanel, ambas de tipo SurfaceView, que son generadas con las anteriores.

Lo primero a analizar es el método onCreate(). La principal diferencia con los otros métodos vistos es la creación de los diversos archivos de audio, guardar la dirección en la que se ejecuta y guardar el nombre de la actividad, que será empleado por la clase encargada de mostrar la pantalla de premio. Todas estas funcionalidades están resaltadas en rojo, azul y verde respectivamente, en la Figura 4.20.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Tomamos el directorio
    fname = this.getPackageName();
    // Vamos a quitar el titulo de la aplicacion
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    // Y hacemos que la pantalla sea full_screen
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    saludo = MediaPlayer.create(ClaseGeneradora.this, R.raw.hola);
    mal = MediaPlayer.create(ClaseGeneradora.this, R.raw.no);
    mb = MediaPlayer.create(ClaseGeneradora.this, R.raw.bien);
    mb2 = MediaPlayer.create(ClaseGeneradora.this, R.raw.denuevo);
    bienhecho = MediaPlayer.create(ClaseGeneradora.this, R.raw.bienhecho);
    asisehace = MediaPlayer.create(ClaseGeneradora.this, R.raw.asisehace);
    enorawena = MediaPlayer.create(ClaseGeneradora.this, R.raw.enorawena);
    nono = MediaPlayer.create(ClaseGeneradora.this, R.raw.nono);
    intentalo = MediaPlayer.create(ClaseGeneradora.this, R.raw.intentalo);
    genial = MediaPlayer.create(ClaseGeneradora.this, R.raw.genial);
    // ejecutando = 0;
    unavez = 0;
    mg = new MainGamePanel(this);
    // Vamos a activar el audio de los botones fisicos dentro de la
    // aplicacion
    setVolumeControlStream(AudioManager.STREAM_MUSIC);
    Log.d(TAG, "creado mg = " + mg.getId());
    // Abajo llamare a la creacion de la pantalla de juego
    coin = new SoundPool(10, AudioManager.STREAM_MUSIC, 100);
    clint = coin.load(this, R.raw.clin, 1);

    // Abrimos las preferencias que queremos
    pref = getSharedPreferences("Superados", Context.MODE_PRIVATE);

    guardarActividad();
}
```

Figura 4.20

En el método guardar actividad (Figura 4.21), añadimos a las preferencias el nombre que tiene la actividad, con la finalidad de volver a la misma cuando acaba la animación del premio.

```
public void guardarActividad() {
    SharedPreferences.Editor editor = pref.edit();
    editor.putString("Actividad", "com.example.juego.CLASEGENERADORA");
    editor.commit();
}
```

Figura 4.21

Como ya se ha mencionado anteriormente, uno de los problemas que se encuentran al emplear SurfaceView, es el manejo del hilo de ejecución cuando se paraliza la actividad, así como el proceso de recuperación de la misma. Para solucionar este problema es conveniente explicar el ciclo de vida de una actividad en Android, tal y como se muestra en developer.android.com [9] al que hace referencia la Figura 4.22.

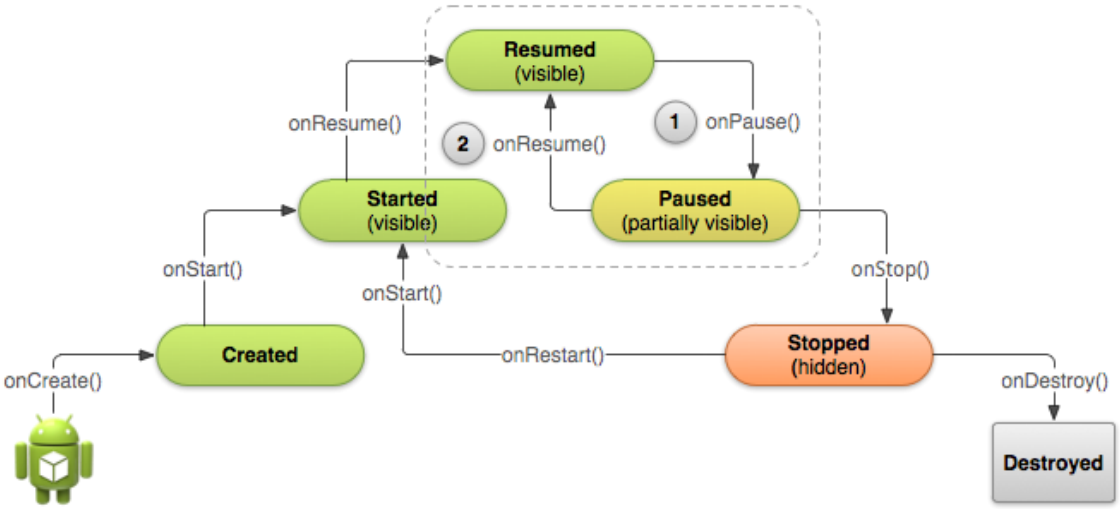


Figura 4.22

En dicha imagen se puede apreciar los métodos que invoca Android para manejar la información. Gracias a esto se pudo modificar los métodos correctos en los cuales se deben guardar los objetos necesarios si es preciso que se ejecute onResume().

Es decir, si el usuario pulsa el botón “HOME” o “BACK” la actividad entra en modo pausa, quedando a la espera de volver a estar activa o en caso de ser necesario por falta de recursos en el terminal parar definitivamente mediante Stopped. Si el usuario vuelve a la actividad, ésta se reanuda mediante onResume(). También cuando es creada se llama a éste método, de modo que modificaremos el existente en Android para que realice las acciones que consideremos oportunas.

Para guardar todos los datos que son necesarios para restaurar correctamente la pantalla de un nivel, se empleará en las diferentes clases que generan los temas dos métodos que son llamados por onPause() y onResume(). Estos métodos son onSaveInstanceState() y onRestoreInstanceState(). Todos ellos (Figura 4.23) son necesarios para que la aplicación se mantenga estable.

```
@Override
protected void onPause() {
    // Voy a intentar solucionar el problema que tengo a la hora de parar la
    // aplicacion debido al onpause() method
    Log.d(TAG, "Dentro de onPause");
    Log.d(TAG, "Contexto de mg = " + mg.tomarcontext());
    super.onPause();
    unavez = 1;
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    //Guardamos el estado
    super.onSaveInstanceState(outState);
    Log.d(TAG, "aciertos pause" + mg.getAciertos());
    outState.putInt("ACIERTOS", mg.getAciertos());
    numeros = mg.getAciertos();
    obj = mg.guardarObjeto();
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    // Restauramos el estado guardado
    super.onRestoreInstanceState(savedInstanceState);
}

@Override
protected void onResume() {
    Log.d(TAG, "Dentro de onResume");
    // Y ahora cargamos la vista del juego
    mg = new MainGamePanel(this);
    mg.setAciertos(numeros);
    if (unavez != 0) {
        mg.setObjeto(obj);
    }
    setContentView(mg);
    Log.d(TAG, "Cargando juego Dentro de onResume con mg = " + numeros);
    Log.d(TAG, "Contexto de mg = " + mg.tomarcontext());
    super.onResume();
}
```

Figura 4.23

Por último, todas las clases que generan los temas contienen un método (Figura 4.25) para activar el tema siguiente.

```
public static void activartema2() {
    SharedPreferences.Editor editor = pref.edit();
    editor.putBoolean("nivel1", true);
    editor.putBoolean("nivel2", true);
    editor.commit();
}
```

Figura 4.24

Ahora vamos a analizar MainGamePanel, que se trata de la pantalla de juego. Esta clase es de tipo SurfaceView y tiene una estructura idéntica a MainBilletePanel.

En primer lugar, creamos una instancia a esta clase mediante su constructor, en el cual, mandamos a ejecutar el hilo encargado de otorgarle dinamismo a la pantalla. También se debe definir un holder (Figura 4.25), que permitirá procesar todas las acciones que el usuario realice con los objetos. Dicho holder es necesario para sincronizar el hilo de ejecución con el hilo general del sistema operativo, logrando de este modo controlar de mejor manera la sucesión de métodos dentro de la actividad.

```
public MainGamePanel(Context context) {
    super(context);
    micontext = context;
    Log.d(TAG, "Dentro del juego con context = " + context);
    // creamos el hilo del juego
    thread = new MainThread(this);
    Log.d(TAG, "Dentro del juego con id = " + this.getId());
    // Vamos a hacer que detecte los distintos "eventos", que en nuestro
    // caso serán las pulsaciones
    holder = getHolder();
    holder.addCallback(new SurfaceHolder.Callback() {

        public void surfaceDestroyed(SurfaceHolder holder) {
            Log.d(TAG, "saliendo del juego con el hilo");
            // Esto será llamado cuando se destruya toda la vista que
            // tenemos
            boolean retry = true;
            // Como queremos terminar con los hilos de ejecución de una
            // manera lo
            // más limpia posible será necesario
            // hacer lo siguiente
            thread.setRunning(false);
            while (retry) {
                try {
                    ClaseGeneradora.terminaVoces();
                    thread.join();
                    retry = false;
                } catch (InterruptedException e) {
                }
            }
        }

        public void surfaceCreated(SurfaceHolder holder) {
            // Aqui vamos a iniciar el hilo del juego
            thread.setRunning(true);
            thread.start();
            Log.d(TAG, "iniciando el hilo dentro del juego");
        }

        public void surfaceChanged(SurfaceHolder holder, int format,
                int width, int height) {
            Log.d(TAG, "No hago na que yo sepa");
        }
    });
```

Figura 4.25

En esta parte del programa también generamos todas las imágenes a mostrar en un principio, y realizamos el cálculo de las monedas necesarias (Figura 4.26) mediante

el algoritmo de las monedas [10]. Dicho algoritmo (Figura 4.27) nos permite calcular el número mínimo de monedas necesarias para mostrar la ayuda, logrando así que además de otorgar el precio exacto, se señalen las monedas de manera correcta.

```
// Iniciamos todos los valores de las monedas que tenemos
TiposMonedas = new int[6];
TiposMonedas[0] = 200;
TiposMonedas[1] = 100;
TiposMonedas[2] = 50;
TiposMonedas[3] = 20;
TiposMonedas[4] = 10;
TiposMonedas[5] = 5;

CantidadMonedas = new int[6];
CantidadMonedas[0] = 1;
CantidadMonedas[1] = 1;
CantidadMonedas[2] = 1;
CantidadMonedas[3] = 1;
CantidadMonedas[4] = 1;
CantidadMonedas[5] = 1;

MonedasNecesarias = new int[6];
for (int i = 0; i < 6; i++) {
    MonedasNecesarias[i] = 0;
}
```

Figura 4.26

```
public void obtenermonedas() {
    // float aPagar = convertirprecio(obj.getprecio());
    int aPagar = obj.getprecio();
    int i = 0;
    int numMonedas;
    while ((aPagar > 0) && (6 > i)) {
        numMonedas = (int) (aPagar / TiposMonedas[i]);
        System.out.println(i);
        MonedasNecesarias[i] = numMonedas;
        valores = valores + " " + i;
        // Aqui como monedas necesarias será 0 o 1 lo que hago es
        // multiplicar por el valor de la moneda que representa, si es 0
        // restara 0 pero si tiene valor si que bajara el apagar
        aPagar = aPagar - (MonedasNecesarias[i] * TiposMonedas[i]);
        // aPagar = convertirprecio(aPagar);
        i++;
    }
    // Al final de esta funcion tendremos en MonedasNecesarias un array con
    // el numero de monedas de cada una necesarias, en este caso si tueviera
    // monedas infinitas
    resetearAyudas();
}
```

Figura 4.27

Una vez hemos realizado el cálculo de las monedas, si el usuario falla varias veces consecutivas se ejecutará el método cargarayuda() (Figura 4.28), el cual empleará dicho cálculo para situar las flechas indicando que monedas o billetes son los correctos, ayudando de esta manera a la persona que se encuentre jugando.

```
public void cargarayuda() {
    // Esta funcion la voy a emplear para saber cuantos circulos debo cargar
    // para ayudar a seleccionar las monedas correctas,gracias a esto
    // mostrare los circulos necesarios
    for (int k = 0; k < 6; k++) {
        if (MonedasNecesarias[k] != 0) {
            if (k == 0) {
                ayuda2.setX(dos.getX());
                ayuda2.setY(dos.getY());
            }
            if (k == 1) {
                ayuda1.setX(moneda.getX());
                ayuda1.setY(moneda.getY());
            }
            if (k == 2) {
                ayuda50.setX(cinq.getX());
                ayuda50.setY(cinq.getY());
            }
            if (k == 3) {
                ayuda20.setX(vein.getX());
                ayuda20.setY(vein.getY());
            }
            if (k == 4) {
                ayuda10.setX(diez.getX());
                ayuda10.setY(diez.getY());
            }
            if (k == 5) {
                ayuda5.setX(cincocents.getX());
                ayuda5.setY(cincocents.getY());
            }
```

Figura 4.28

Esta clase también cuenta con el método onTouchEvent() (Figura 4.29) implementado. Dentro del mismo se realiza un llamamiento a funciones de las monedas o billetes, que serán los que decidan el comportamiento a seguir.

```
if (event.getAction() == MotionEvent.ACTION_DOWN) {
    // Delegamos el tratamiento de la accion a la moneda si es que se
    // puede mover
    if (dejomover) {
        moneda.handleActionDown((int) event.getX(), (int) event.getY());
        dos.handleActionDown((int) event.getX(), (int) event.getY());
        cinq.handleActionDown((int) event.getX(), (int) event.getY());
        vein.handleActionDown((int) event.getX(), (int) event.getY());
        diez.handleActionDown((int) event.getX(), (int) event.getY());
        cincocents.handleActionDown((int) event.getX(),
                (int) event.getY());
    }
}
if (event.getAction() == MotionEvent.ACTION_MOVE) {
    // los gestos
    if (moneda.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        moneda.setX((int) event.getX());
        moneda.setY((int) event.getY());
    }
    if (dos.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        dos.setX((int) event.getX());
        dos.setY((int) event.getY());
    }
    if (cinq.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        cinq.setX((int) event.getX());
        cinq.setY((int) event.getY());
    }
    if (vein.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        vein.setX((int) event.getX());
        vein.setY((int) event.getY());
    }
    if (diez.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        diez.setX((int) event.getX());
        diez.setY((int) event.getY());
    }
    if (cincocents.isTouched()) {
        // La moneda la hemos cogido y la debemos mover
        cincocents.setX((int) event.getX());
        cincocents.setY((int) event.getY());
    }
}
if (event.getAction() == MotionEvent.ACTION_UP) {
    // Deberemos dejar la moneda donde levantemos el dedo
    if (moneda.isTouched()) {
        moneda.setTouched(false);
    }
```

Figura 4.29

En este método se puede realizar una acción al pulsar, otra al deslizar el dedo por la pantalla y finalmente al finalizar la pulsación. Todas estas distinciones son necesarias para tratar el movimiento de las monedas. En primer lugar analizamos si hemos pulsado en una de ellas (Figura 4.29 en rojo), después si la movemos, reasignamos su posición (Figura 4.29 azul) y al soltarla volvemos a indicar que su estado es “no pulsada” ()Figura 4.29 verde).

Esta clase contiene también un método onDraw(), en el cual vamos analizando la situación de los objetos mostrados. Estos objetos tienen cada uno su propio método draw() que será llamado aquí. También se comprobará si está sonando la voz del dependiente para invocar un método que contiene, que permitirá animar la imagen, como veremos más adelante. En general este método es como el mostrado anteriormente, aunque algo más completo por contener mayor cantidad de elementos a mostrar.

El método cambiarprecio() (Figura 4.30) se encarga de calcular cada vez que se introduce una moneda, si el pago ha sido correcto. En caso afirmativo mostrará la imagen de acierto y generará el audio correspondiente, mientras que en el caso contrario mostrará la imagen de error acompañada de una locución de error.

```
public void cambiarprecio() {
    // Vamos a recalcular el precio cada vez que sucede que la moneda entra
    // en la caja

    if ((moneda.pedircambio() == true) && (m1 > 0)) {
        int nuevoprecio = obj.getprecio();
        nuevoprecio = nuevoprecio - moneda.getvalor();
        obj.setprecio(nuevoprecio);
        if (nuevoprecio == 0) {
            muy.setmostrar(75);
            numaciertos();
            yuhu();
        } else if (nuevoprecio < 0) {
            malo.setmostrar(60);
            mal();
        }
        m1--;
        Log.d(TAG, "valor m=" + m1 + " , pedircambio= " + dos.pedircambio()
                + " pedir= " + moneda.pedircambio());
    }
```

Figura 4.30

Para crear los objetos en todas las clases necesarias, se ha empleado una estructura como la mostrada en la Figura 4.31. En algunos casos puede ser más compleja en función de la aleatoriedad que deseemos otorgar a la generación de los objetos, pero la estructura es igual para todas las actividades.

```
public void cargarobj() {
    // Cada vez que queramos cargar otro objeto simplemente debemos llamar a
    // este metodo y nos lo creará

    // Con el directorio de la aplicacion debemos buscar las imagenes
    f = getResources().getIdentifier(aleatorio.calc(), "drawable",
            directorio);
    if (f == aux) {
        f = getResources().getIdentifier(aleatorio.calc(), "drawable",
                directorio);
    }

    // Cargamos las fotos de forma aleatoria y su precio
    obj = new Objeto(BitmapFactory.decodeResource(getResources(), f),
            aleatorio.getprecio(), aleatorio.getetiqueta());
```

Figura 4.31

Es necesario indicar, que todas las clases de tipo SurfaceView han seguido este tipo de estructura, siendo similares sus métodos. Por ejemplo, el método onTouchEvent() (Figura 4.32) ha sufrido una modificación, ya que queremos detectar si se pulsa en el botón "si" o "no", pero el método sigue siendo el mismo.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    // Vamos a comprobar donde pulsamos
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        if (pulsado) {
            if ((alto > event.getY()) && (event.getY() > alto * 14 / 16)) {
                // Nos situamos en la parte de los botones
                if (event.getX() > ancho - si.getWidth()) {
                    // Estamos pulsando el Si
                    Sipulsado();
                } else if (event.getX() < no.getWidth()) {
                    // Estamos pulsando el No
                    Nopulsado();
                } else {
                    // estamos pulsando en el centro nose si haré algo aquí
                }
            }
        }
    }
    return true;
}
```

Figura 4.32

Por este motivo, solo resta mostrar los métodos realmente diferentes o que, siendo el mismo, realicen alguna labro fundamental de la aplicación.

El método getResizedBitmap() (Figura 4.33) es el encargado de redimensionar una imagen.

```
public Bitmap getResizedBitmap(Bitmap bm, int newHeight, int newWidth) {
    int width = bm.getWidth();
    int height = bm.getHeight();
    float scaleWidth = ((float) newWidth) / width;
    float scaleHeight = ((float) newHeight) / height;
    // Creamos una matriz para modificar la imagen
    Matrix matrix = new Matrix();
    // reescalamos el bitmap
    matrix.postScale(scaleWidth, scaleHeight);
    // recreamos el bitmap
    Bitmap resizedBitmap = Bitmap.createBitmap(bm, 0, 0, width, height,
            matrix, false);
    return resizedBitmap;
}
```

Figura 4.33

```
public void Sipulsado() {
    if (moneda.getvalor() >= obj.getprecio()) {
        GeneraMeLLega.muybien();
        aciertos++;
        cargarobj();
        if (aciertos % 3 == 0) {
            try {
                Thread.sleep(1750);
                cogermonedas();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        numaciertos();
    } else {
        GeneraMeLLega.no();
    }
}
```

```
public void Nopulsado() {
    if (moneda.getvalor() < obj.getprecio()) {
        GeneraMeLLega.muybien();
        aciertos++;
        cargarobj();
        if (aciertos % 3 == 0) {
            try {
                Thread.sleep(1750);
                cogermonedas();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        numaciertos();
    } else {
        GeneraMeLLega.no();
    }
}
```

Figura 4.34

En la Figura 4.34 se muestra en función del botón pulsado, cómo la aplicación generará un nuevo objeto, acumulará el acierto, esperará un tiempo y reproducirá un sonido.

A continuación, se analizará las clases que pertenecen al tema 5, que es el más dinámico y completo.

En primer lugar debemos tener en cuenta la suma del dinero ahorrado al superar los niveles, ya que en función de este, mostraremos las monedas o billetes un mensaje indicando que no disponemos de dinero y que debemos jugar a los temas anteriores.

Además de los métodos que implementan el resto de clases que generan los temas, aquí incluimos uno (Figura 4.35) que modifica el valor del total acumulado dentro de preferencias. Este método será llamado cuando salgamos del tema mediante el método onPause() (Figura 4.36), para que no sea necesario rescribir permanentemente dicho valor.

```
public void aumentarAhorrado(int nuevasuma) {
    SharedPreferences.Editor editor = pref.edit();
    editor.putInt("Ahorrado", nuevasuma);
    editor.commit();
}
```

Figura 4.35

```
@Override
protected void onPause() {
    // Aqui cambiaremos la cantidad de dinero acumulada
    System.out.println("La suma dentro de la Actividad es = "
            + mg.getSuma());
    aumentarAhorrado(mg.getSuma());
    super.onPause();
    unavez = 1;
    pararaudio();
}
```

Figura 4.36

Por otro lado, la clase GameView, que hereda de SurfaceView, es diferente debido a que realiza más animaciones tiene más complejidad que el resto.

Lo primero que realiza esta clase es el cálculo de todo el dinero acumulado y calcula, mediante el algoritmo de las monedas mencionado anteriormente, qué monedas debe cargar de manera dinámica (Figura 4.37). También carga seis objetos de manera aleatoria y los sitúa en la maquina expendedora cada uno en una posición (Figura 4.38).

```
private void generarMonedas() {
    // Este método creará las monedas necesarias
    for (int i = 11; i >= 0; i--) {
        if (cantidadMonedas[i] > 0) {
            if (i == 0) {
                while (cantidadMonedas[i] > 0) {
                    monedas.add(new MonedasMaquina(BitmapFactory
                            .decodeResource(getResources(),
                                    R.drawable.cincocents), 0,
                            cantidadMonedas[i], ancho, alto, 5));
                    cantidadMonedas[i] = cantidadMonedas[i] - 1;
                }
            } else if (i == 1) {
                while (cantidadMonedas[i] > 0) {
                    monedas.add(new MonedasMaquina(BitmapFactory
                            .decodeResource(getResources(),
                                    R.drawable.diezcents), 1,
                            cantidadMonedas[i], ancho, alto, 10));
                    cantidadMonedas[i] = cantidadMonedas[i] - 1;
                }
            } else if (i == 2) {
                while (cantidadMonedas[i] > 0) {
                    monedas.add(new MonedasMaquina(BitmapFactory
                            .decodeResource(getResources(),
                                    R.drawable.veintecents), 2,
                            cantidadMonedas[i], ancho, alto, 20));
                    cantidadMonedas[i] = cantidadMonedas[i] - 1;
                }
```

Figura 4.37

```
aleatorio1 = new Aleatoriedad();
aleatorio2 = new AleatorioBilletes();
obj0 = cargarobjeto();
obj0.setX(ancho * 2 / 10);
obj0.setY(alto * 1 / 16);
obj1 = cargarobjeto();
obj1.setX(ancho * 45 / 100);
obj1.setY(alto * 1 / 16);
obj2 = cargarobjeto();
obj2.setX(ancho * 2 / 10);
obj2.setY(alto * 2 / 10);
obj3 = cargarobjeto();
obj3.setX(ancho * 45 / 100);
obj3.setY(alto * 2 / 10);
obj4 = cargarobjeto();
obj4.setX(ancho * 2 / 10);
obj4.setY(alto * 33 / 100);
obj5 = cargarobjeto();
obj5.setX(ancho * 45 / 100);
obj5.setY(alto * 33 / 100);
obj0.setverprecio(false);
obj1.setverprecio(false);
obj2.setverprecio(false);
obj3.setverprecio(false);
obj4.setverprecio(false);
obj5.setverprecio(false);
```

Figura 4.38

Esta clase contiene el método onDraw() más complejo de todos los implementados en la aplicación. Este método consta de dos partes.

La primera (Figura 4.39) se encarga de cargar los objetos, mostrar la iluminación si un objeto está seleccionado, animar la máquina y mostrar el dinero disponible.

La segunda parte (Figura 4.40 y Figura 4.41) realiza una animación creando desde el centro de la pantalla un cuadrado emergente y finalmente mostrando el objeto comprado y los cambios recibidos si se introduce una cantidad de dinero mayor al importe del objeto.

```
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawARGB(255, 130, 200, 255);
    if (!cambio) {
        if (GeneraTienda.bien.isPlaying() || GeneraTienda.mal.isPlaying()
                || GeneraTienda.saludos.isPlaying()) {
            permitirmovimiento = false;
        } else {
            permitirmovimiento = true;
        }
        if (seleccionado != 10) {
            ma.draw(canvas, true);
        } else {
            obj0.setverprecio(false);
            obj1.setverprecio(false);
            obj2.setverprecio(false);
            obj3.setverprecio(false);
            obj4.setverprecio(false);
            obj5.setverprecio(false);
        }
        dst.set(0, 0, canvas.getWidth(), canvas.getHeight() * 10 / 16);
        canvas.drawBitmap(maquina, null, dst, null);
        marcar(canvas);
        obj0.dibujarMaquina(canvas);
        obj1.dibujarMaquina(canvas);
        obj2.dibujarMaquina(canvas);
        obj3.dibujarMaquina(canvas);
        obj4.dibujarMaquina(canvas);
        obj5.dibujarMaquina(canvas);
        if (suma == 0) {
            dst.set(canvas.getWidth() / 2 - juega.getWidth() / 2,
                    canvas.getHeight() * 10 / 16, canvas.getWidth() / 2
                            + juega.getWidth() / 2, canvas.getHeight());
            canvas.drawBitmap(juega, null, dst, null);
        }
        for (MonedasMaquina m : monedas) {
            m.draw(canvas, true);
        }
        landa = 0;
```

Figura 4.39

```
} else {
    if (GeneraTienda.bien.isPlaying() || GeneraTienda.mal.isPlaying()
            || GeneraTienda.saludos.isPlaying()) {
        permitirmovimiento = false;
    } else {
        permitirmovimiento = true;
    }
    dst.set(0, 0, canvas.getWidth(), canvas.getHeight() * 10 / 16);
    canvas.drawBitmap(maquina, null, dst, null);
    obj0.dibujarMaquina(canvas);
    obj1.dibujarMaquina(canvas);
    obj2.dibujarMaquina(canvas);
    obj3.dibujarMaquina(canvas);
    obj4.dibujarMaquina(canvas);
    obj5.dibujarMaquina(canvas);
    ma.draw(canvas, false);
    crearTodo(canvas);
}
```

Figura 4.40

```
private void crearTodo(Canvas canvas) {
    // Este método será llamado cuando paguemos correctamente un producto y
    // nos mostrará el producto y los cambios
    permitirmovimiento = false;
    //Creamos la animacion emergente
    if (landa < canvas.getWidth() / 2) {
        landa += 5;
        dstaux.set(canvas.getWidth() / 2 - landa, canvas.getHeight() / 2
                - landa, canvas.getWidth() / 2 + landa, canvas.getHeight()
                / 2 + landa);
        canvas.drawBitmap(logro, null, dstaux, null);
    } else {
        //Si ya disponemos de la superficie mostramos el objeto y los cambios
        landa = canvas.getWidth() / 2;
        dstaux.set(canvas.getWidth() / 2 - landa, canvas.getHeight() / 2
                - landa - canvas.getHeight() * 2 / 16, canvas.getWidth()
                / 2 + landa, canvas.getHeight());
        canvas.drawBitmap(logro, null, dstaux, null);
        am.draw(canvas);
        dst.set(canvas.getWidth() * 3 / 10, canvas.getHeight() / 2 - landa
                - canvas.getHeight() * 2 / 16, canvas.getWidth() * 7 / 10,
                canvas.getHeight() * 15 / 100);
        canvas.drawBitmap(objetocomprado, null, dst, null);
        dst.set(canvas.getWidth() * 4 / 10, canvas.getHeight() * 3 / 16,
                canvas.getWidth() * 6 / 10, canvas.getHeight() * 5 / 16);
        canvas.drawBitmap(auxiliar.getBmp(), null, dst, null);

        dst.set(canvas.getWidth() * 2 / 10, canvas.getHeight() * 8 / 16,
                canvas.getWidth() * 8 / 10, canvas.getHeight() * 10 / 16);
        canvas.drawBitmap(letracambios, null, dst, null);

        // aqui ira la llamada a las monedas de los cambios
        for (MonedasMaquina ma : monedascambio) {
            ma.cambiosdraw(canvas);
        }

        dst.set(canvas.getWidth() * 3 / 10, canvas.getHeight() * 6 / 16,
                canvas.getWidth() * 7 / 10, canvas.getHeight() * 8 / 16);
        canvas.drawBitmap(bmpcontinuar, null, dst, null);
        continuar = 0;
    }
}
```

Figura 4.41

Además, en esta clase realizamos un tratamiento más complejo dentro del método onTouchEvent() porque es necesario detectar la moneda o billete a mover. Hasta ahora como disponíamos de una sola moneda o billete de cada tipo en pantalla, y todos ellos bien distribuidos por la misma, no era necesario detectar si se pulsa una u otra, porque no había problema con ello. En este nivel, encontramos que podemos disponer varias monedas o billetes del mismo valor, que además estarán colocados uno sobre otro. Esto ocasionaba problema ya que al seleccionar una moneda o billete se activaba en todos ellos su estado de pulsado, lo que ocasionaba que se movieran todos a la vez.

Para evitar este problema se realizó el código correspondiente a la figura 4.42, en el cual se puede observar que se recorre toda la lista de monedas buscando la

posición que corresponde con la moneda que he pulsado. En cuanto obtengo dicha posición, cambio el valor de pulsado del resto de las monedas, logrando así que solo una se mueva. Lo mismo sucede con los billetes.

```
if (event.getAction() == MotionEvent.ACTION_MOVE) {
    if (bug == 0) {
        // for (MonedasMaquina m : monedas) {
        for (int i = monedas.size() - 1; i >= 0; i--) {
            if (monedas.get(i).isTouched()) {
                MonedasMaquina m = monedas.get(i);
                // for(int i=monedas.size()-1; i>=0 ;i--){
                // //Si una moneda ya ha sido pulsada el resto las
                // tengo
                // que dejar sin pulsar
                // if(monedas.get(i)!=m){
                // monedas.get(i).setTouched(false);
                // }
                // }
                for (MonedasMaquina mr : monedas) {
                    if (mr == m) {

                    } else {
                        mr.setTouched(false);
                    }
                }
                m.setX(((int) event.getX()) - ancho * 3 / 24);
                m.setY(((int) event.getY()) - alto * 2 / 32);
```

Figura 4.42

Después se debe detectar si la moneda ha sido arrastrada hasta la ranura de la máquina. Esto se hace comprobando la posición de la moneda o billete en la pantalla. Para no tener problemas con eliminar de la lista de monedas y billetes, todos aquellos que no deban ser eliminados, es necesario sincronizar el hilo de la aplicación (Figura 4.43) para que se calculen las monedas a mostrar de manera ordenada.

```
synchronized (holder) {
    for (int i = monedas.size() - 1; i >= 0; i--) {
        MonedasMaquina mr = monedas.get(i);
        if (mr.isCollition(event.getX(), event.getY(), ancho,
                alto)) {
            monedas.remove(mr);
            break;
        }
    }
}
```

Figura 4.43

En caso de introducir correctamente la cantidad de dinero a pagar, se recalculará los cambios a mostrar de la misma manera descrita anteriormente. De este modo se podrán mostrar los cambios dentro de la segunda parte del método onDraw().

Tras mostrar los cambios, si el usuario pulsa en continuar se realizará un ajuste de la cantidad de dinero disponible. Esto ha sido dos motivos, el primero relacionado con la implementación y el segundo con la representación del dinero en pantalla.

El problema con la implementación surgió por el manejo de la lista de monedas. En un principio se eliminaba cada moneda que se introducía en la máquina, pero al ser una situación irreal se modificó para que solo se contabilizara si la cantidad disponible era mayor que el precio del producto. Más adelante se implementó la adición de las monedas generadas como cambios a las monedas disponibles eliminando previamente la moneda introducida, pero no siempre se obtenían resultados correctos, ya que en función del momento en que se llamase al método draw() de las monedas o billetes no se mostraban correctamente. Finalmente se diseño el método recalcular() (Figura 4.44) en el cual se sincroniza el hilo de ejecución y se realiza un recorrido comparativo entre las monedas disponibles y los cambios obtenidos. También se vuelve a calcular la cantidad disponible total con la finalidad de conocer si se dispone de dinero suficiente para el pago del siguiente artículo seleccionado.

```
private void recalcular() {
    // Aqui volveremos a calcular el dinero a mostrar
    int sumaux = this.suma;
    suma=0;
    sumaux = sumaux - auxiliar.getprecio();
    System.out.println("La suma dentro de GameView es = " + this.suma);
    synchronized (holder) {
        for (MonedasMaquina mc : monedascambio) {
            for (MonedasMaquina m : monedas) {
                if (m.getTipo() == mc.getTipo()) {
                    m.setCantidad(m.getCantidad() + 1);
                }
            }
            monedas.add(mc);
        }
    }
    synchronized (holder) {
        for (MonedasMaquina m : monedas) {
            suma=m.getValor()+suma;
        }
    }
    monedascambio.clear();
}
```

Figura 4.44

El problema relacionado con la distribución del dinero en pantalla fue detectado en las pruebas, de las que hablaremos más adelante. Este problema era que no se distinguía la cantidad de monedas del mismo valor disponibles, es decir, se mostraba únicamente una moneda, quedando el resto ocultas bajo esta, provocando la sensación de que se disponía de menos dinero. Para solucionar este problema, ha sido necesario mejorar el método draw() llamado en las monedas y billetes, que en función de la cantidad del mismo tipo, se situase en una posición un poco desplazada y mostrando parte de la silueta de la moneda o billete. De esta manera se crea la sensación visual de que las monedas se encuentran apiladas. Esta solución la veremos al analizar las monedas y billetes.

A continuación se muestra la estructura común a todos los hilos que son creados y ejecutados en cada clase de tipo SurfaceView. Tomamos como ejemplo MainThread.

Este hilo se encargará de contabilizar los ticks del sistema para lograr que el refresco de pantalla sea uniforme en todos los dispositivos. Gracias a ellos se logra estabilidad dentro de los temas, así como uniformidad en la representación gráfica.

```
@Override
public void run() {
    long ticksPS = 1000 / FPS;
    long startTime;
    long sleepTime;
    while (running) {
        Canvas c = null;
        startTime = System.currentTimeMillis();
        try {
            c = gamePanel.getHolder().lockCanvas();
            synchronized (gamePanel.getHolder()) {
                gamePanel.onDraw(c);
            }
        } catch (Exception e) {

        } finally {
            if (c != null) {
                gamePanel.getHolder().unlockCanvasAndPost(c);
            }
        }
        sleepTime = ticksPS - (System.currentTimeMillis() - startTime);
        try {
            if (sleepTime > 0)
                sleep(sleepTime);
            else
                sleep(10);
        } catch (Exception e) {
        }
    }
}
```

Figura 4.45

En la Figura 4.45 se observa el método principal de cualquier hilo, su método run(). En él se puede apreciar que se controla en todo momento el tiempo interno del sistema y se sincroniza con nuestro panel de juego. En caso de producirse un desajuste entre tiempos, lo que se hace es bloquear el canvas, que se encarga de mostrar los ítems en pantalla, hasta que de nuevo vuelve a estabilizarse. Si tenemos en cuenta que estos cálculos tienen un tiempo de procesado muy pequeño, se logra transmitir al usuario una sensación de fluidez. Esto es muy importante para que el usuario perciba un correcto funcionamiento, sin retrasos en los refrescos de pantalla ni errores inesperados.

Ahora vamos a ver como dotar de animación al dependiente y al niño. Ambas clases siguen la misma estructura, de modo que emplearemos la clase NinoAnimado para analizar sus métodos.

En primer lugar, al crear una instancia a esta clase, se debe definir un bitmap, el cual contendrá representadas varias posiciones, que estarán distribuidas en filas y columnas a las que se accederán cada vez que se refresque la pantalla para simular el movimiento. En la imagen 4.46 se puede apreciar estas variables y el numero de fotogramas (FPS) por segundo necesarios para el cambio entre posiciones.

```
public class NinoAnimado {

    private static final int columnas = 4;
    private static final int filas = 2;
    private Bitmap bitmap;
    private int currentFrame = 0;
    private int valor = 0;
    private int anchobmp;
    private int altobmp;
    private int srcx;
    private int srcy;
    Rect rectangulo, dibujo;
    private long frameTicker; //mide el tiempo desde la ultima actualización de frame
    private int framePeriod; //los milisegundos entre cada frame

    public NinoAnimado(Bitmap bitmap, int fps) {
        this.bitmap = bitmap;
        this.anchobmp = bitmap.getWidth() / columnas;
        this.altobmp = bitmap.getHeight() / filas;
        rectangulo = new Rect();
        dibujo = new Rect();
        framePeriod = 2000 / fps;
        frameTicker = 01;

    }
```

Figura 4.46

Una vez se disponen de estos valores, es necesario dibujar el objeto animado y con cada actualización modificar su estado. En el método update() (Figura 4.47) se comprueba si ha transcurrido el tiempo necesario entre cada movimiento y de ser así, se recalcula una nueva columna, que representa la imagen a mostrar.

```
public void update(long gameTime) {
    // con esto lo que logramos es movernos por las diferentes figuras
    // dentro del sprite
    if(gameTime > frameTicker + framePeriod){
        frameTicker = gameTime;
        if(valor==0){
            valor = 1;
        }else{
            valor = 0;
            currentFrame = ++currentFrame % columnas;
        }
    }
}
```

Figura 4.47

Además, mediante el método draw() (Figura 4.48) se realiza el llamamiento a update() y se ajusta la imagen obtenida tras el cálculo, al área definida. Es decir, se reajustan todas las posiciones a cada pantalla y además se intercambian para lograr la sensación de movimiento.

```
public void draw(Canvas canvas) {
    update(System.currentTimeMillis());
    srcx = currentFrame * anchobmp;
    srcy = valor * altobmp;
    dibujo.set(srcx, srcy, srcx + anchobmp,srcy+ altobmp);
    rectangulo.set(canvas.getWidth() * 2 / 10, canvas.getHeight() * 7 / 16,
            canvas.getWidth() * 8 / 10, canvas.getHeight());
    canvas.drawBitmap(bitmap, dibujo, rectangulo, null);
}
```

Figura 4.48

Ahora analizaremos la clase MuyBien, que es esquemáticamente idéntica a la clase Mal. En esta clase se lleva el control de un contador, que representa el número de refrescos de pantalla que se debe mostrar (Figura 4.49), en caso de indicarle que lo haga.

```
public void draw(Canvas canvas) {
    if (mostrar > 1) {
        canvas.drawBitmap(mb, (canvas.getWidth() / 2) - mb.getWidth() / 2,
                (canvas.getHeight() / 2), null);
    }
    mostrar--;
    if(mostrar==-1000){
        mostrar=1;
    }
}
```

Figura 4.49

Para finalizar con la implementación de las clases, queda por analizar la clase Moneda y Objeto. Moneda y Billete comparten estructura de modo que analizaremos Moneda.

De esta clase cabe destacar su método handleActionDown() (Figura 4.50) que se encarga de detectar si se ha pulsado en una moneda o billete, así como la modificación realizada en el método draw() para poder representar de una manera más clara la situación en la que se tiene mas de una moneda o billete del mismo valor (Figura 4.51). Se trata de un cambio en el que detectar la cantidad de monedas del mismo tipo y dibujar cada una desplazada una pequeña distancia respecto de la anterior.

```
public void handleActionDown(int eventX, int eventY) {
    if (eventX >= (x - bitmap.getWidth() / 2)
            && (eventX <= (x + bitmap.getWidth() / 2))) {
        if (eventY >= (y - bitmap.getHeight() / 2)
                && (eventY <= (y + bitmap.getHeight() / 2))){
            // En caso de pulsar en medio de la imagen el objeto es
            // pulsado
            setTouched(true);
        } else {
            setTouched(false);
        }
    }
}
```

Figura 4.50

```
public void cambiosdraw(Canvas canvas) {
    if (cantidad > 0) {
        dst.set(x, y, x + ancho * 2 / 10, y + alto * 2 / 16);
        canvas.drawBitmap(bp, null, dst, null);
        if (cantidad > 1) {
            if (this.tipo != 8) {
                dst.set(x - ancho * 2 / 10, y, x, y + alto * 2 / 16);
                canvas.drawBitmap(bp, null, dst, null);
            } else {
                dst.set(x - ancho * 2 / 12, y, x, y + alto * 2 / 16);
                canvas.drawText("X2", x, y, rojo);
            }
        }
    }
}
```

Figura 4.51

Finalmente, veremos cómo creamos los objetos. Para ello es necesario ver la clase Objeto y Aleatorio de manera conjunta.

La clase Aleatorio se encarga de proporcionar todos los valores necesarios para invocar la clase Objeto. Esta clase mediante un random, unos valores definidos dentro de la misma, que contienen el nombre de la imagen del objeto, su precio, y la etiqueta para mostrar en la pantalla de juego. Todo este proceso se puede apreciar en la Figura 4.52. Serán necesarios métodos sencillos que no se muestran por su simplicidad en dicha imagen para devolver los valores obtenidos.

```
public String calc() {
    Random rnd = new Random();
    val = rnd.nextInt(10);
    switch (val) {
    case 0:
        foto = "balon";
        precio = 300;
        etiqueta = "3,00";
        break;
    case 1:
        foto = "helado";
        precio = 150;
        etiqueta = "1,50";
        break;
    case 2:
        foto = "reloj";
        precio = 350;
        etiqueta = "3,50";
        break;
    case 3:
        foto = "pulpo";
        precio = 10;
        etiqueta = "0,10";
        break;
    case 4:
        foto = "palomitas";
        precio = 250;
        etiqueta = "2,50";
        break;
    case 5:
        foto = "jumpers";
        precio = 50;
```

Figura 4.52

Una vez obtenidos todos los datos necesarios, dentro de cada juego se creará el objeto con los valores obtenidos de la clase Aleatorio. Para mostrar los objetos en las diversas pantallas, variando sus posiciones y tamaños en función de si se trata del tema 1 y 3 o los temas 2 y 4, es necesario un método onDraw() con dos posibilidades. El código acorde al primer caso se señala en la Figura 4.53 mediante el recuadro de color rojo, mientras que para el segundo caso es de color azul.

```
public void draw(Canvas canvas) {
    if ((x == 0) && (y == 0)) {
        Rect dst = new Rect();
        dst.set(canvas.getWidth() * 1 / 10, canvas.getHeight() * 3 / 16,
                canvas.getWidth() * 4 / 10, canvas.getHeight() * 6 / 16);
        canvas.drawBitmap(bp, null, dst, null);
        if (verprecio) {
            canvas.drawText("" + this.etiqueta + "",
                    (canvas.getWidth() * 37 / 100),
                    (canvas.getHeight() * 7 / 16) + 12, pintura);
        }
    } else {
        if (verprecio) {
            Rect dst = new Rect();
            dst.set(x, y, x + 120, y + 150);
            canvas.drawBitmap(bp, null, dst, null);
            canvas.drawText("" + this.etiqueta + "", x + bp.getWidth() / 6,
                    375, pintura);
        }else{
            Rect dst = new Rect();
            dst.set(canvas.getWidth()*3/10,canvas.getHeight()*1/16,canvas.getWidth()*7/10,canvas.getHeight()*5/16);
            canvas.drawBitmap(bp, null,dst, null);
        }
    }

}
```

Figura 4.53

5.-Pruebas

En este apartado se van a describir las diferentes pruebas que se han llevado a cambo durante la realización de este proyecto. Estará distribuido por fases, debido a que durante el desarrollo de la aplicación, se ha probado en varias ocasiones para conocer las opiniones del usuario y realizar las mejoras pertinentes. En todos los casos nos centraremos en el apartado funcional principalmente, aunque haremos alguna excepción si han sido necesarios cambios en el apartado gráfico con el fin de mejorar la funcionalidad general de la aplicación. Todas las pruebas han sido realizadas con la ayuda de mi prima, mujer de 18 años con Síndrome de Down, bajo la supervisión de mis tíos y primo, que han aportado nuevas ideas en función de sus experiencias en la educación de mi prima.

5.1 Primera Fase

En esta fase se disponía únicamente de un nivel, concretamente el desarrollo básico del tema 1 y las pruebas fueron realizadas por mi prima. La distribución de las monedas en pantalla no era la adecuada, ya que al estar todas en una sola fila, el usuario tenía dificultades para seleccionar cada una de ellas. De modo que se optó por realizar dos filas de monedas, constando cada una de ellas de tres monedas. También se observó que la tendencia general del usuario es seleccionar las monedas correctas, en lugar de arrastrarlas, por ello se optó por crear el tutorial explicativo. Además se pudo observar que en ocasiones el refresco de pantalla no era lo suficientemente fluido como para transmitir una correcta sensación de funcionamiento al usuario, ya que en ocasiones las monedas seleccionadas dejaban de moverse correctamente.

Por otro lado, uno de los comentarios más constructivos que se realizaron tras estas pruebas, dio lugar a los temas 2 y 4, debido a que el usuario estableció como nuevo requisito un nivel en el que analizar el valor de las monedas y calcular si es posible comprar un artículo. También el método de arrastrar monedas, a pesar de no ser el más intuitivo, es el que mejor sensación causa debido a la sensación de libertad de movimientos que produce.

5.2 Segunda Fase

Para esta fase ya se disponía de una pantalla de tutorial y de una correcta organización de las monedas y billetes en pantalla. Estos cambios fueron valorados positivamente por el cliente, de modo que sentó la base de la distribución de las monedas y billetes en pantalla.

También se disponía del nivel que simulaba la tienda. Para la fecha de realización de las pruebas se disponía únicamente de su apartado visual, y su implementación todavía no estaba concluida, sin embargo se apreció cierta falta de

interés por parte del usuario objetivo por este nivel. Ésta es la principal razón por la que finalmente esta actividad fue descartada de la aplicación, debido a la falta de interés del usuario por el mismo, así como un difícil desarrollo tanto gráfico como en la implementación de la actividad.

Finalmente, ante el más que probable incumplimiento de un requisito fundamental, como la *jugabilidad* y sencillez en los niveles, se eliminó.

Por otro lado, a partir de esta fase se comienza a diseñar la aplicación de manera que sea compatible con varias resoluciones de pantalla. Para comprobar esta funcionalidad se ha dispuesto de varios dispositivos. En la fase final se mostrará el resultado de la aplicación en dichos dispositivos.

5.3 Tercera Fase

Las pruebas de esta fase se realizaron para mostrar al usuario, en este caso el tutor, los nuevos niveles incorporados, en este caso los temas 2 y 4.

Se pudo comprobar una correcta implementación de los mismos, así como una representación correcta de la idea sugerida por el usuario.

Me gustaría destacar un comentario realizado por el tutor durante las pruebas de estos temas que decía: “Me parece sencillo visualmente pero muy eficaz, además la melodía es pegadiza y a los niños les gusta eso”.

5.4 Cuarta Fase

En este nivel todas las pruebas fueron realizadas por mí, quedando las últimas pruebas realizadas para la última fase. En este nivel se comenzó añadiendo movimiento a las imágenes y un nuevo tema.

Pude comprobar que el último tema es el mejor implementado, debido a la fluidez que tiene en ejecución, así como un gran dinamismo.

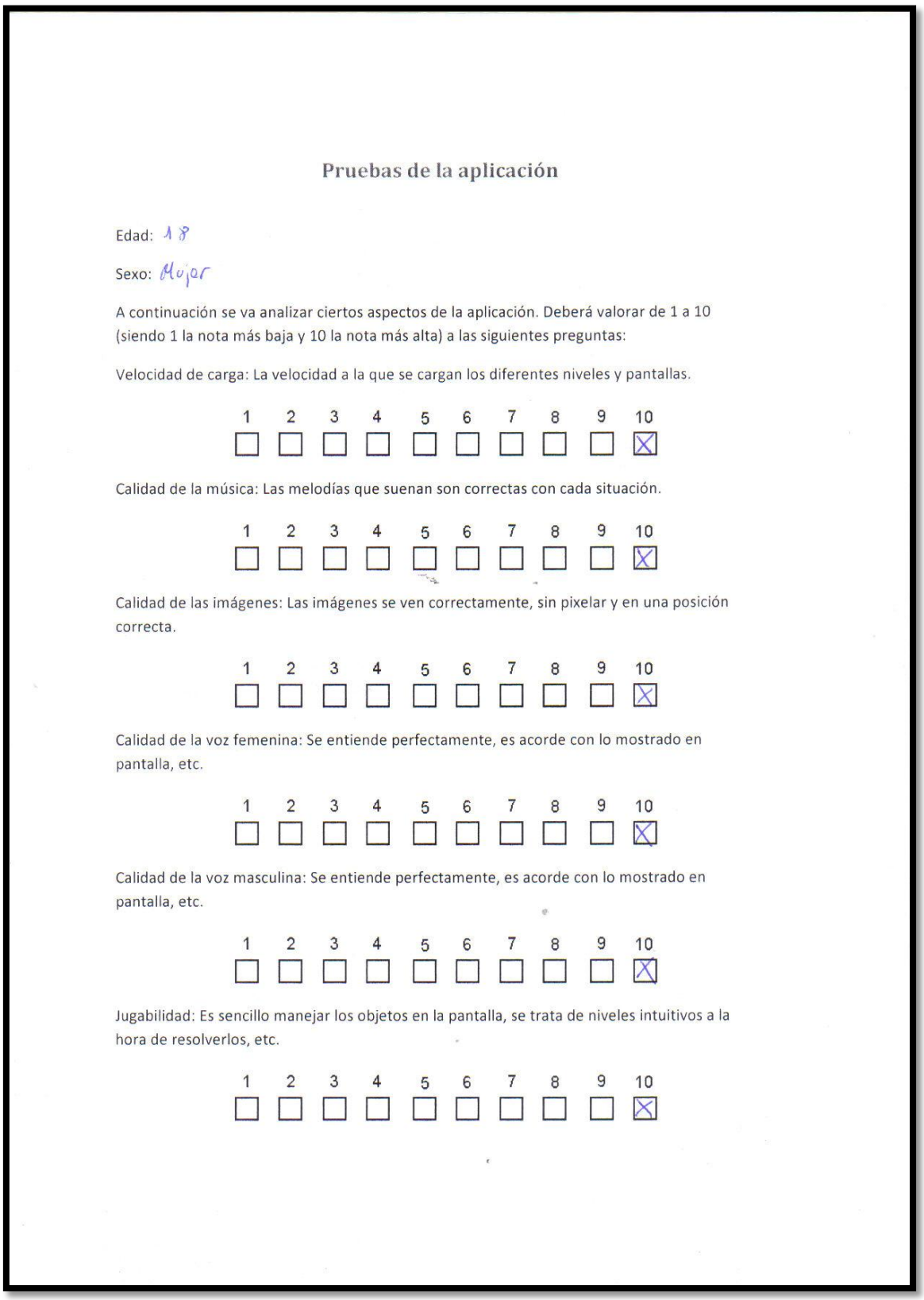
Se realizaron diversas comprobaciones de los cálculos que realiza la aplicación para mostrar en pantalla las monedas o billetes necesarios. En este caso se pudo comprobar que en ocasiones se mostraban momentáneamente de manera correcta, pero al pulsar en pantalla y producirse un refresco de la misma, había monedas o billetes que se eliminaban. Esto se debía a la sincronización del hilo de ejecución de la actividad con el del sistema, de modo que se solucionó como se muestra en el apartado de implementación.

5.5 Fase final

Para esta fase la aplicación ya disponía de todos los niveles y animaciones mostrados en el apartado de diseño gráfico. Sin embargo, fueron necesarias pequeñas modificaciones en función de las opiniones de los usuarios.

Los usuarios encargados de evaluar la aplicación fueron mi prima, mujer de 18 años con Síndrome de Down, mi primo, varón de 27 años muy aficionado a los videojuegos y mi tío, varón de 55 años que nunca ha empleado ningún dispositivo electrónico relacionado con los videojuegos. Además se tomaron en gran consideración las opiniones de mis padres y hermano y de mi pareja, quienes se encargaron de mejorar pequeños aspectos del apartado gráfico, como la consonancia de colores empleados a lo largo de la aplicación o que alguna de las pantallas causara sensación de saturación de las mismas. A pesar de ser cambios menores, tienen una gran importancia debido a que de ellos también depende la sensación transmitida al usuario cuando emplea la aplicación.

A continuación, se muestran en las Figuras 5.1, 5.2, 5.3, 5.4, 5.5 y 5.6, un informe realizado por los usuarios que evaluaron la aplicación, mencionados anteriormente. En dichas imágenes, se puede apreciar también sus opiniones acerca de la aplicación, posibles mejoras y posibles nuevos temas para añadir en líneas futuras.



Pruebas de la aplicación

Edad: 18

Sexo: Mujer

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

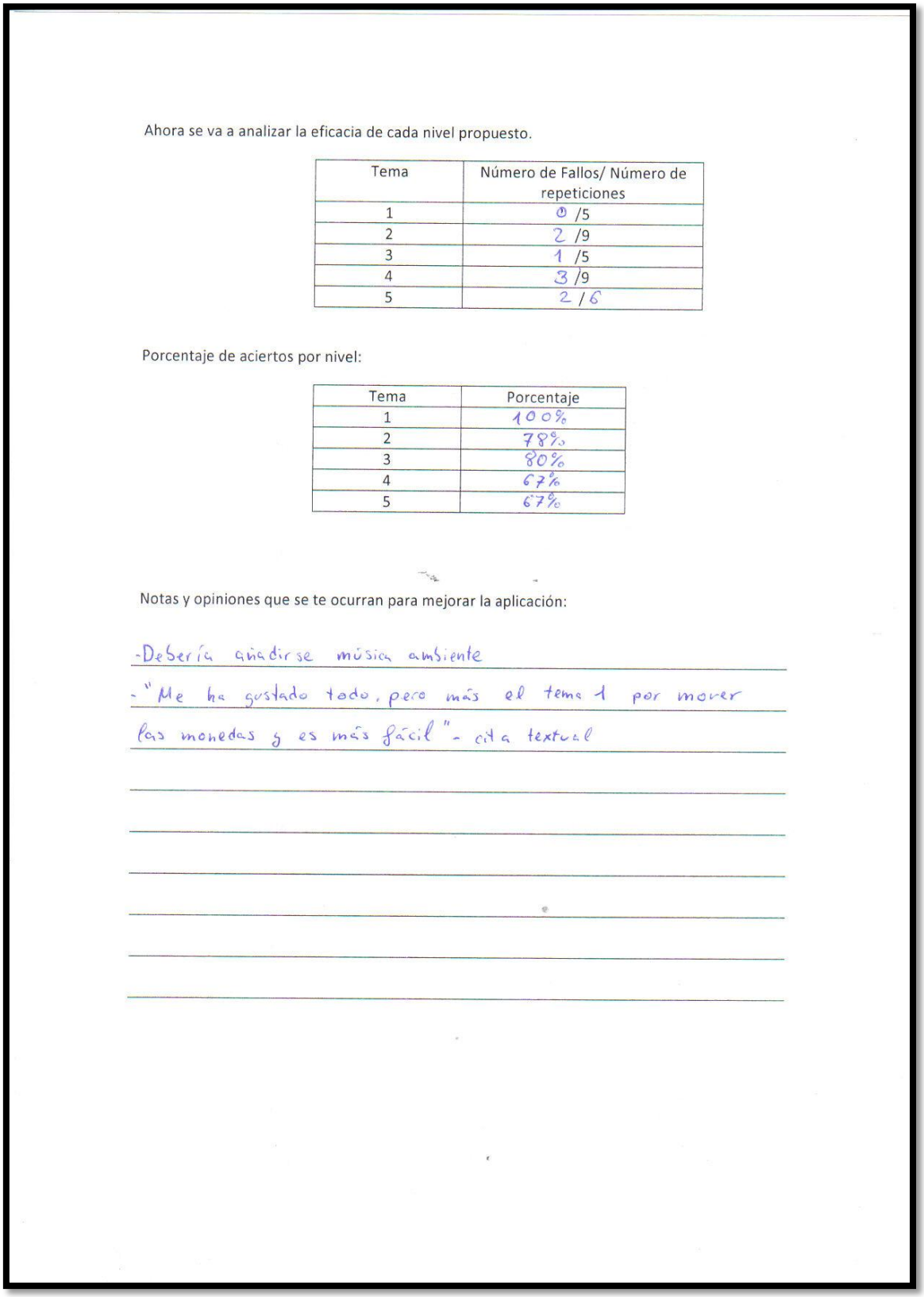
Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Figura 5.1



Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	2 /9
3	1 /5
4	3 /9
5	2 /6

Porcentaje de aciertos por nivel:

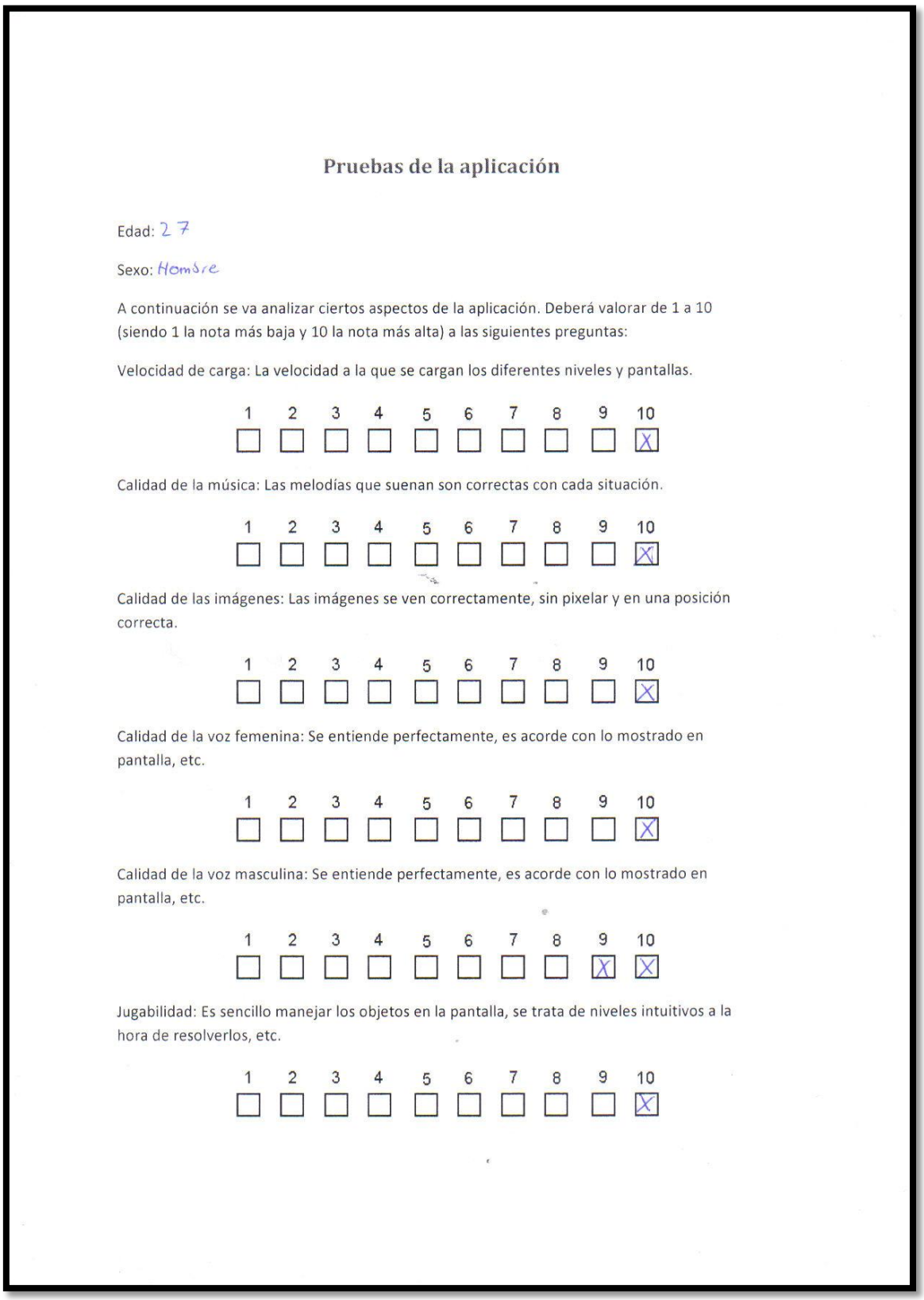
Tema	Porcentaje
1	100%
2	78%
3	80%
4	67%
5	67%

Notas y opiniones que se te ocurran para mejorar la aplicación:

-Debería añadirse música ambiente

-"Me ha gustado todo, pero más el tema 1 por mover las monedas y es más fácil" - cita textual

Figura 5.2



Pruebas de la aplicación

Edad: 27

Sexo: Hombre

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☒	☒

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Figura 5.3

Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	0 /9
3	0 /5
4	0 /9
5	0 /3

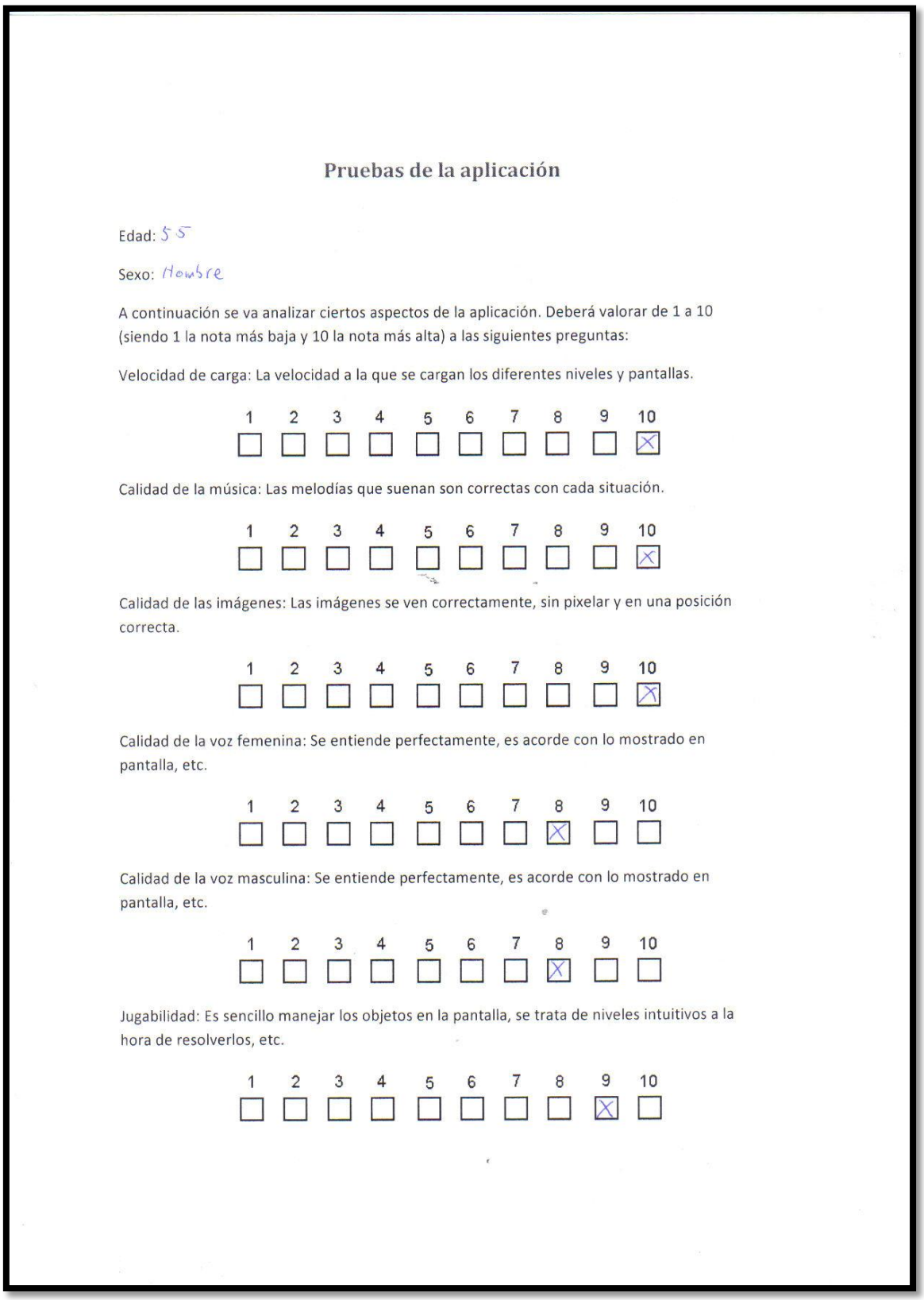
Porcentaje de aciertos por nivel:

Tema	Porcentaje
1	100%
2	100%
3	100%
4	100%
5	100%

Notas y opiniones que se te ocurran para mejorar la aplicación:

Hacer un nivel que tenga menos monedas y preguntar si lo que devuelve está bien.

Figura 5.4



Pruebas de la aplicación

Edad: 55

Sexo: Hombre

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☒	☐	☐

Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☒	☐	☐

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☒	☐

Figura 5.5

Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	0 /9
3	0 /5
4	0 /9
5	0 /

Porcentaje de aciertos por nivel:

Tema	Porcentaje
1	100%
2	100%
3	100%
4	100%
5	100%

Notas y opiniones que se te ocurran para mejorar la aplicación:

Poner las monedas más separadas en el último nivel
Tener el volumen del audio

Figura 5.6

A continuación, como se ha mencionado antes, vamos a mostrar la aplicación en funcionamiento en diversos dispositivos.

Estos dispositivos han sido, un Samsung Galaxy SII (GT I-9100), un Sony Xperia P, un Sony Xperia U, una Tablet Asus Nexus 7 y una Samsung galaxy Tab 10.1. Dichos dispositivos montan diversas versiones de Android. En orden sería Android 4.0.3, Android 4.0.4, Android 2.3.6, Android 4.1.2 y Android 3.0.

Las resoluciones de estas pantallas varían mucho entre ellas. Sus resoluciones en orden serían: 800x480, 960x540, 854x480, 1280x800 y 1280x800.

A continuación se muestra la pantalla principal en el Galaxy SII, Xperia P y Nexus 7 (Figura 5.7 y Figura 5.8). La Figura 5.9 muestra la pantalla de temas en dichos terminales.

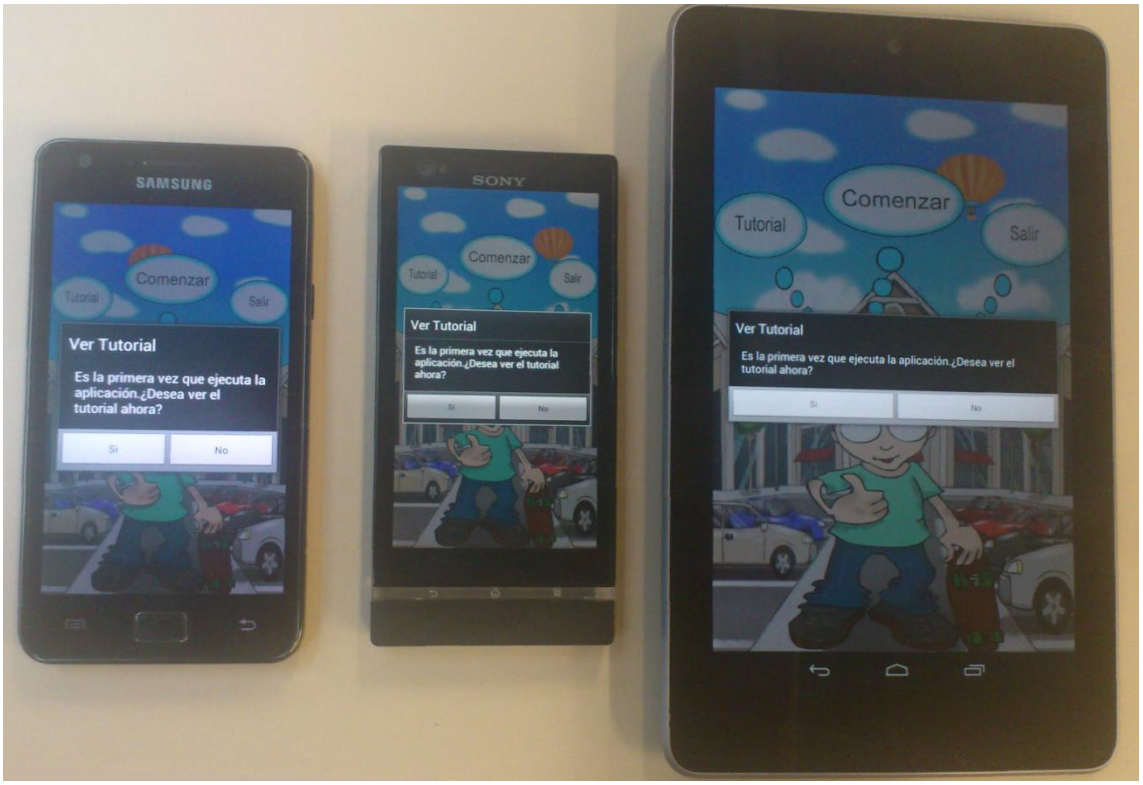


Figura 5.7

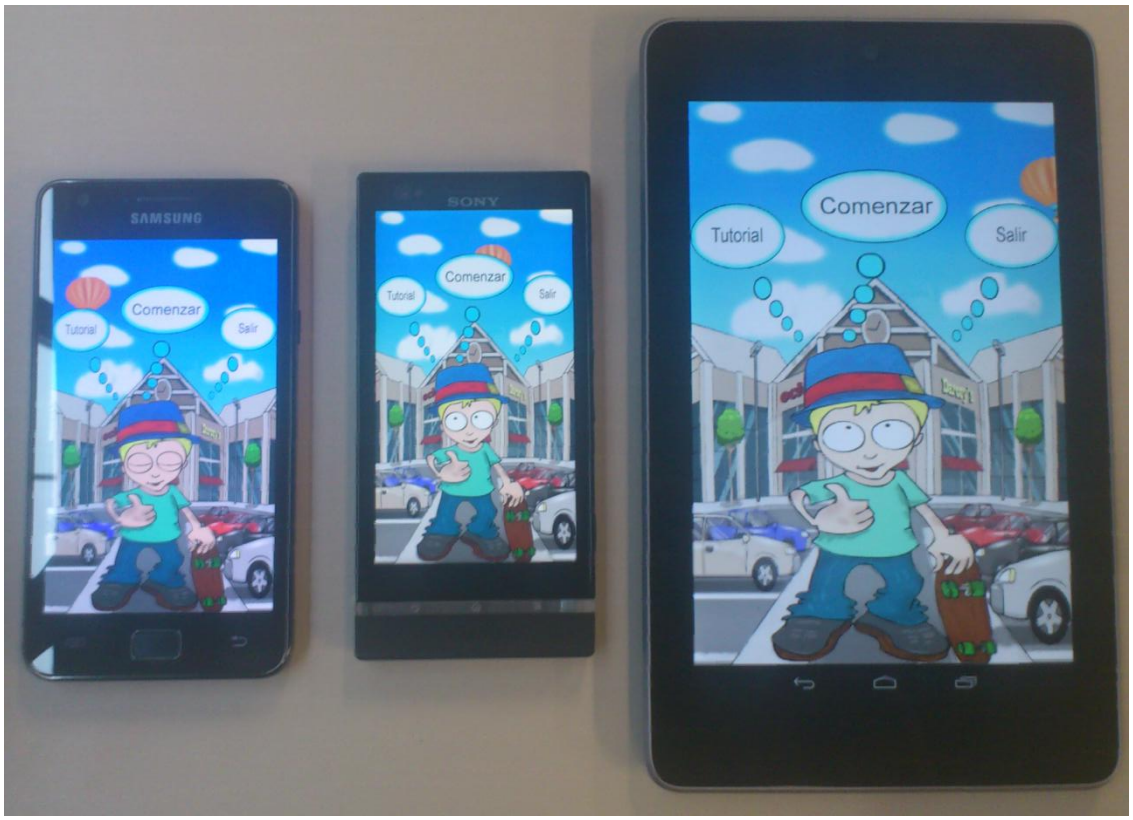


Figura 5.8

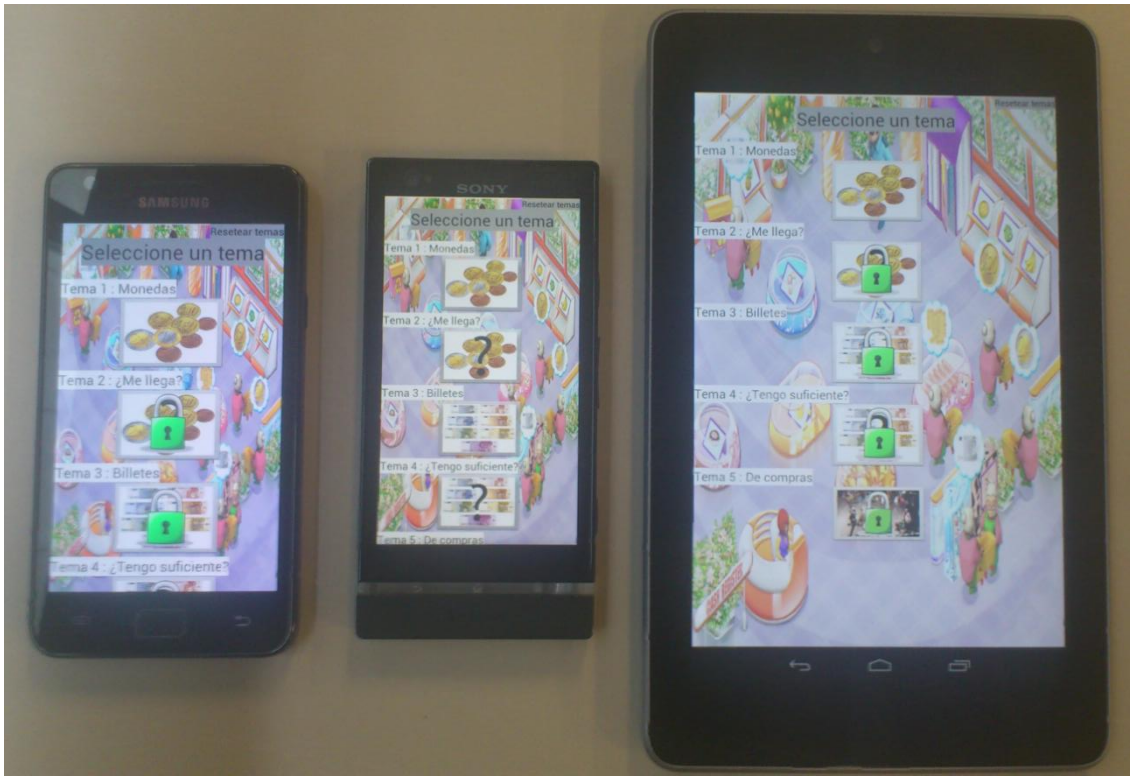


Figura 5.9

Ahora se van a mostrar diferentes imágenes (Figuras 5.10, 5.11 y 5.12) de los diferentes temas.

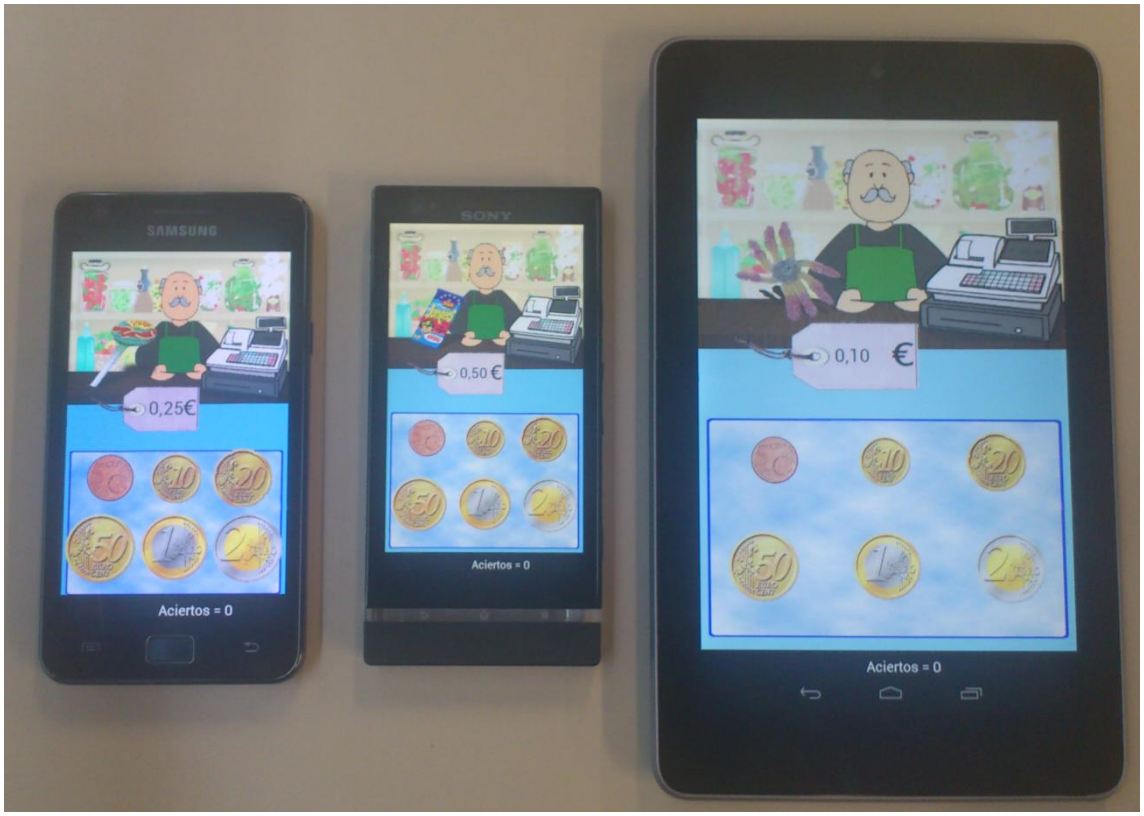


Figura 5.10

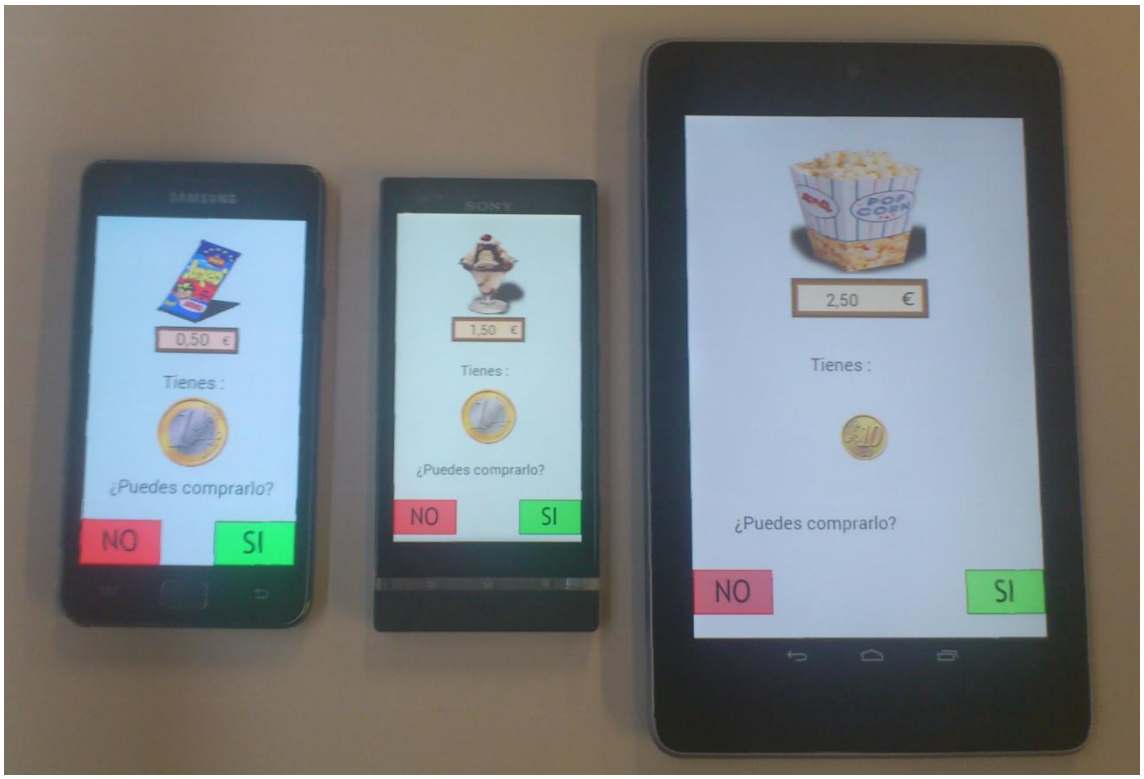


Figura 5.11

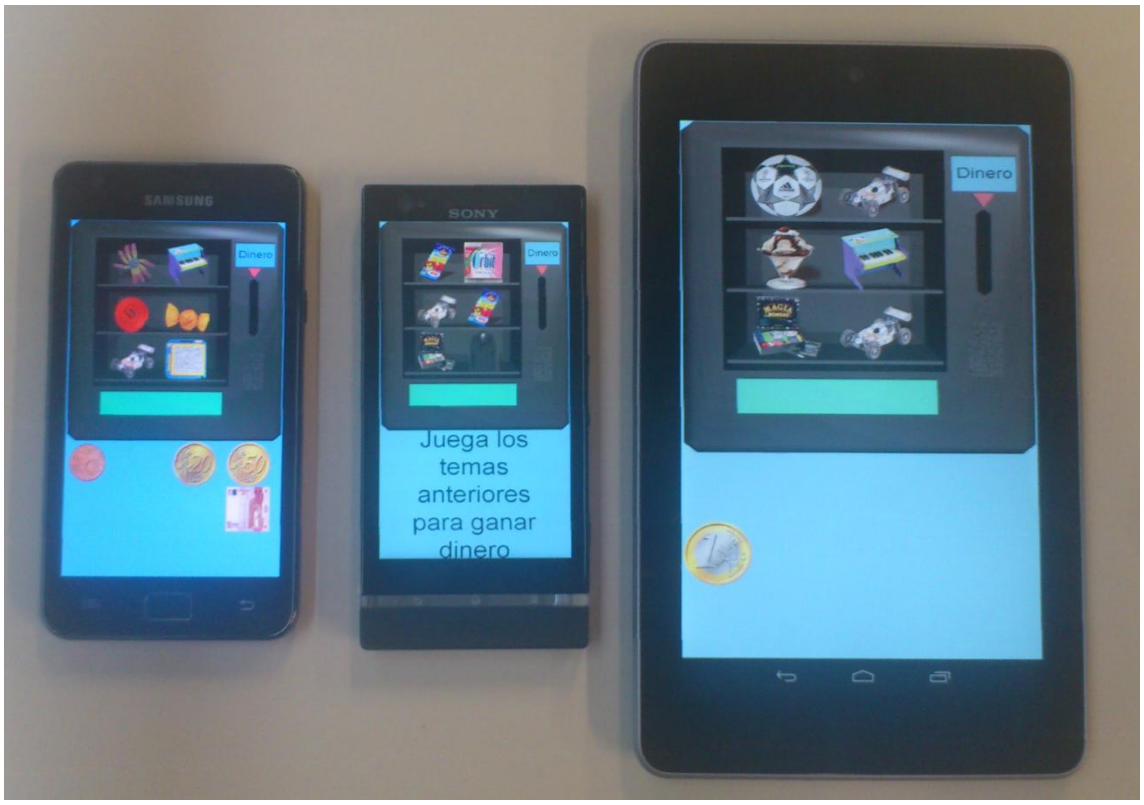


Figura 5.12

El resto de terminales no han estado disponibles para la imagen final.

6.- Presupuesto

Se va a intentar realizar una aproximación de un posible presupuesto por realizar esta aplicación. Para llevar a cabo esta sección, se ha consultado con el tutor posibles planteamientos para abordarlo. Al final me he decantado por realizar una aproximación por horas.

Esta aproximación permite representar una situación en la que el programador se puede encontrar si realiza trabajos por encargo. Dada esta situación debemos plantearnos el número de horas necesarias para realizar las diferentes etapas del proyecto.

En primer lugar se contempla el número de horas necesario para analizar los requisitos propuestos por el cliente. Esto, a su vez, conlleva reuniones entre ambas partes y esquemas iniciales que faciliten la comunicación y entendimiento entre ellos. Para esta labor se han empleado aproximadamente 5 días a 2-3 horas por día.
A estos valores se les debe multiplicar por el precio medio por hora que cobra un consultor de software. Buscando en internet diferentes fuentes, se puede decir que se cobra un precio medio entre 50 y 70 € la hora, de modo que por ser nuestro primer proyecto cobraremos 50€/h.

Haciendo los cálculos:

5 * 2.5 * 50 = 625€

Después se debe tener en cuenta el desarrollo realizado. En este aspecto debemos considerar tanto el trabajo gráfico, ya que se trata de un juego, como el trabajo de implementar la aplicación.

Para diseñar y realizar las diferentes imágenes a mostrar a lo largo de la aplicación, cumpliendo con las ideas del cliente, ha sido necesario realizar bocetos y aproximaciones de las mismas. En nuestro caso el propio desarrollador ha realizado ambas tareas, pero en un proyecto de mayor tamaño, sería necesario contratar a un diseñador gráfico para esta labor, lo que sumaría más horas de trabajo al tener que reunirse para evaluar los trabajos y poner ideas en común.

En mi caso, se han empleado 5 días trabajando de media 8 horas, para diseñar y realizar los gráficos de la aplicación. Para conocer el sueldo de un programador gráfico he encontrado información del colegio de diseñadores gráficos [11]. En dicha página se indica que se debe diferenciar el precio por hora del diseño y el presupuesto base. La hora de diseño se encuentra a un precio de 100$, que con el cambio a € serían de unos 78€. Si por ser nuestra primera aplicación y tratarse de unos gráficos sencillos de realizar si se tiene experiencia, lo justo sería cobrar entre 60 y 80€, de modo que los cálculos quedarían:

5 * 8 * 60 = 2400€

Ahora que da la fase de implementación. Aquí se puede evaluar de dos maneras, por horas o por líneas de código fuente efectivo generadas. Se realizará una aproximación como se ve en la carrera de la segunda manera.

Para realizar esta aproximación disponemos del modelo COCOMO, que permite evaluar el esfuerzo realizado. Para ello se deben tener en cuenta la cantidad de líneas de código fuente empleado en la aplicación. Tenemos 6 KSLOC, de los cuales código no eficiente es un 20%, es decir 4.8 KSLOC de código eficiente entre las diferentes clases. Si calculamos el esfuerzo realizado tenemos:

E = 2.4 * 4.8 ^ 1.05 = 12.6 personas/mes

El tiempo necesario para realizar este proyecto será:

T = 2.5 * 12.6 ^ 0.38 = 6,5 meses

El siguiente paso sería calcular cuántas personas hacen falta para desarrollar el proyecto. En mi caso he sido yo solo y al tratarse de un proyecto fin de carrera le he dedicado muchas horas no lectivas, pero no implica que este proyecto no se pueda llevar a cabo mediante una empresa, de modo que me dispongo a calcular el coste simulando una empresa. El número de personar necesarias será:

Nº = 13.28/6.6 = 1.94 >> 2 personas.

Por tanto, consideraremos que son necesarias 2 personas trabajando a jornada completa durante 6 meses y medio, aproximadamente. Si dichas personas cobran la hora a 40 € obtenemos un coste de proyecto de:

Coste = 35 días * 8 horas * 2 personas * 40 €/h = 22400 €.

Finalmente, se deben incluir gastos indirectos. Para ello se ha consultado un artículo de la universidad de Cádiz [12], donde gracias a las tablas proporcionadas, se puede calcular que los costes indirectos supondrían un plus del 30% del coste total. De modo que lo calcularemos más adelante.

Por último, se debe considerar el tiempo empleado en las pruebas. Dicho tiempo ha sido sido de 8 horas, si sumamos todos los intentos y reuniones. Si se cobra a 50 € tenemos:

Coste = 50 * 8 = 400 €

Realizamos el cálculo total de la aplicación:

625 + 4800 + 22400 + 400 = 25825 €

A este precio se le debe añadir los impuestos indirectos, es decir un 30%, de modo que tendremos que sumarle 7747,5 €, obteniendo un total de:

Total presupuesto de la aplicación: 33.572,50 €

7.- Conclusiones y líneas futuras

Para terminar la memoria del proyecto, se van a recopilar en este apartado las conclusiones y líneas futuras del mismo.

Lo primero que me gustaría señalar es que tras la finalización de la aplicación, gracias a las pruebas realizadas, he podido comprobar que la aplicación diseñada no solo puede ser empleada por los niños con Síndrome de Down, sino por todas aquellas personas que no estén familiarizadas con los euros o que se inicien en el uso de los mismos.

También he de mencionar en este apartado, que desarrollar la aplicación de manera que sea compatible con el mayor número de dispositivos posibles ha sido más complicado que si la hubiéramos diseñado de manera estática para un solo dispositivo. Sin embargo, estoy muy orgulloso del resultado final, especialmente cuando se tiene la posibilidad de probar la aplicación en diversos terminales de manera satisfactoria.

Por otro lado, se ha sufrido la famosa fragmentación que tiene Android, ya que cuando comencé a investigar y a aprender cómo desarrollar aplicaciones en Android, la versión del sistema que se encontraba en el mercado era la 2.3, mientras que al término del proyecto ya está disponible en diversos terminales la versión 4.2. Este es un gran problema actual, ya que el software y hardware avanzan más rápido que el soporte del mismo. Con esto quiero hacer una pequeña crítica social, ya que para mantener todas las aplicaciones actualizadas y estables, supone un enorme esfuerzo y gasto económico.

Además, como ya se ha comentado en algún caso, los desarrolladores nos encontramos con problemas en la implementación de algunos métodos dentro del propio sistema operativo para el que se programa. Esto no solo sucede con Android, ya que el sistema que supone una competencia directa para éste, iOS, también está comenzando a sufrir estos mismos problemas, debido a que la sociedad de consumo en la que nos encontramos.

En cuanto a la simplicidad de la aplicación, realmente se ha logrado transmitir este objetivo al usuario. Como anécdota relacionada con este aspecto, me gustaría indicar que uno de los usuarios que realizó la prueba, concretamente el de mayor edad, nunca había jugado a ningún videojuego ni nada por el estilo, sino más bien todo lo contrario ya que cumple con el perfil de hombre que está en desacuerdo con esta tecnología. Sin embargo, al ver a otros usuarios manejar la aplicación, decidió probarla y sus opiniones han sido de gran valía.

Como experiencia personal, realmente he disfrutado mucho realizando este proyecto. Siempre he estado interesado en el mundo tecnológico y desde que comencé a investigar acerca de Android, me ha fascinado todo lo relacionado con el mismo. También me gustaría probar suerte con esta aplicación, y algunas otras ideas que tengo en mente, en su distribución en Google Play. En mi opinión, es un mercado en auge que puede abrirme las puertas y acogerme en un futuro no muy lejano

Por otro lado, la aplicación se podría seguir desarrollando, añadiendo nuevos niveles y funcionalidades. Algunas de ellas se describen a continuación.

En primer lugar, se podría implementar la posibilidad de que el usuario pudiera introducir sus propias fotos de productos y sus propios precios. Esto no se ha implementado por falta de tiempo, pero se basaría en crear una pequeña base de datos en la que almacenar los directorios de las fotos. Sería sencillo de implementar debido a los métodos que están implementados, ya que se encargarán de ajustar dinámicamente cualquier imagen a la resolución correcta en función de cada terminal.

Otra de las mejoras que se podrían introducir, sería disponer de un hilo musical activo durante toda la aplicación. Esta funcionalidad es totalmente secundaria, pero produciría en el usuario mayor comodidad durante el uso de la misma.

Los usuarios también sugirieron que se podría añadir un nivel en el que se disponga de menor número de monedas o billetes, y se tuviera que calcular e indicar si los cambios recibidos tras el pago de un artículo son correctos o no. Este nivel no sería difícil de implementar ya que se dispone del algoritmo de cálculo de cambios, pero sería necesario realizar algún diseño gráfico nuevo para que resultase intuitivo y sencillo de jugar.

Además, si retomamos lo expuesto en el documento que representa el anteproyecto, la idea inicial fue crear una aplicación para ayudar a los niños con Síndrome de Down a resolver problemas cotidianos, como poner la mesa o distinguir objetos, y no únicamente enseñarles las monedas. Sin embargo no ha sido posible el desarrollo de una aplicación de dicho calibre por falta de tiempo, por lo tanto, ésta podría ser una clara línea de desarrollo futuro.

También será necesario mejorar el tratamiento de imágenes para ciertos terminales. Es decir, a pesar de haber probado la aplicación con distintos dispositivos, con el objetivo de probar la redimensión de imágenes y la compatibilidad entre versiones, la aplicación actual solo dispone de imágenes en alta resolución, por lo que ocupa un espacio en memoria considerable. Además no es posible instalarla en una memoria externa, sino que únicamente se puede instalar dentro de la memoria del teléfono. Esto supone un problema ya que muchos terminales del mercado pueden no disponer de gran capacidad de memoria a pesar de ser compatibles con la aplicación. El trabajo a realizar en este caso será modificar las imágenes para que su almacenamiento ocupe menos espacio.

Por último, me gustaría indicar que mi aplicación cumple con las expectativas puestas en ella y supone una clara alternativa a las aplicaciones destinadas a la enseñanza del euro. En comparación con las aplicaciones introducidas en el apartado de estado del arte, mi aplicación es diferente en modo de juego, en los niveles y en el dinamismo logrado. Además, en caso de ser distribuida, se realizarán dos versiones. En una de ellas se podrá encontrar publicidad no intrusiva dentro de la aplicación, que aportarán beneficios por descargas y por visualización de publicidad, y otra de pago en la cual la publicidad será eliminada. Esta versión me gustaría distribuirla con un precio acorde a la oferta y demanda del mercado de aplicaciones, y teniendo en cuenta que las disponibles en iOS cuestan 1.79€ y 2.69€ respectivamente, pero siendo acorde a la dinámica de Android considero 0.99€ su precio justo de venta.

8.- Bibliografía y referencias

Para obtener las bases necesarias para el proyecto ha sido necesaria la realización de diversos tutoriales. Entre ellos cabe destacar:

Serie de videos para desarrollar una aplicación Android, por Travis en el canal de thenewboston. Con ellos se realizó una toma de contacto con Android y cómo comenzar a desarrollar una aplicación.
http://www.youtube.com/watch?v=SUOWNXGRc6g&list=EC2F07DBCDCC01493A&feature=plcp

Para comenzar a diseñar un juego para Android, se empleó un pequeño tutorial de edu4java:
http://www.edu4java.com/androidgame.html

También fue necesaria la consulta de todos los API's y métodos recomendados por Google para desarrollar aplicaciones Android, todos ellos de su página de desarrolladores oficial:
http://developer.android.com/index.html

Además, se consultaron diversos libros de diferentes niveles de experiencia de programación en Android, concretamente:

- Beginning Android por Mark L. Murphy Ed. APRESS
- Android Essentials por Chris Haseman Ed. APRESS
- Pro Android por Sayed Y Hashimi y Satya Komatinemi Ed. Apress

Por último, se va a proceder a detallar las referencias indicadas a lo largo de la memoria.

[1] Special-Needs Collection Books, ADHD, Autism, Down Syndrome, www.woodbinehouse.com
[14 de noviembre de 2012]

[2] Androiduniverse.net, http://www.androiduniverse.net/informe-android-ics-alcanza-el-15-y-jelly-bean-aparece/
[7 de noviembre de 2012]

[3] Kids Numbers and Math Lite – Aplicaciones de Android en Google Play, https://play.google.com/store/apps/details?id=zok.android.numbers&feature=search_result#?t=W251bGwsMSwyLDEsInpvay5hbmRyb2lkLm51bWJlcnMiXQ..
[14 de noviembre de 2012]

[4] Aprende a contar 123 Gratis – Aplicaciones de Android en Google Play, https://play.google.com/store/apps/details?id=com.giggleup.TC1AFree&feature=search_result#?t=W251bGwsMSwyLDEsImNvbS5naWdnbGV1cC5UQzFBRnJlZSJd
[14 de noviembre de 2012]

[5] Euro€: monedas y matemáticas para iPhone, iPod touch y iPad en la App Store de iTunes, https://itunes.apple.com/es/app/euro-monedas-y-matematicas/id504737910?mt=8
[14 de noviembre de 2012]

[6] Monedas en la Selva – aprende a calcular monedas (para iPad) para iPad en la App Store de iTunes, https://itunes.apple.com/es/app/monedas-en-la-selva-aprende-a-calcular-monedas-para-ipad/id380864501
[14 de noviembre de 2012]

[7] Revista ICONO 14 #6 CREATIVIDAD, http://www.icono14.net/revista/num6/articulo%20JULIA%20GONZALEZ3.htm
[14 de noviembre de 2012]

[8] Eclipse IDE for Java EE Developers | Eclipse Packages, http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr1
[14 de noviembre de 2012]

[9] Pausing and Resuming an Activity | Android Developers, http://developer.android.com/training/basics/activity-lifecycle/pausing.html
[14 de noviembre de 2012]

[10] Algoritmos Voraces (Greedy), http://www.slideshare.net/luzenith_g/algoritmos-voraces-greedy
[14 de noviembre de 2012]

[11] Tarifario de Precios de Diseño Gráfico – Colegio de Diseñadores Gráficos de Misiones, http://www.cdgm.org.ar/tarifario/index.html
[14 de noviembre de 2012]

[12] Universidad de Cádiz, http://www2.uca.es/serv/asuntos_econo/catalogo_servicio/vi_8_calculo_costes_indirectos.htm
[14 de noviembre de 2012]

Aplicación de aprendizaje de tareas cotidianas en personas con discapacidad mediante dispositivos móviles

Asier Alejo Siles

Índice

Problema

- El problema que incentivó el proyecto fue la **dependencia** que tienen los niños con Síndrome de Down para desempeñar tareas cotidianas como puede ser realizar la compra de un producto.

Objetivos: Metas principales

- El proyecto se realizó con 2 objetivos fundamentales:
  - **Ayudar** a las personas con Síndrome de Down y **enseñarles** las monedas y billetes de euro.
  - **Aprender** y **demostrarme** a mi mismo la capacidad de crear un juego para Android.

Objetivos a lograr con el proyecto

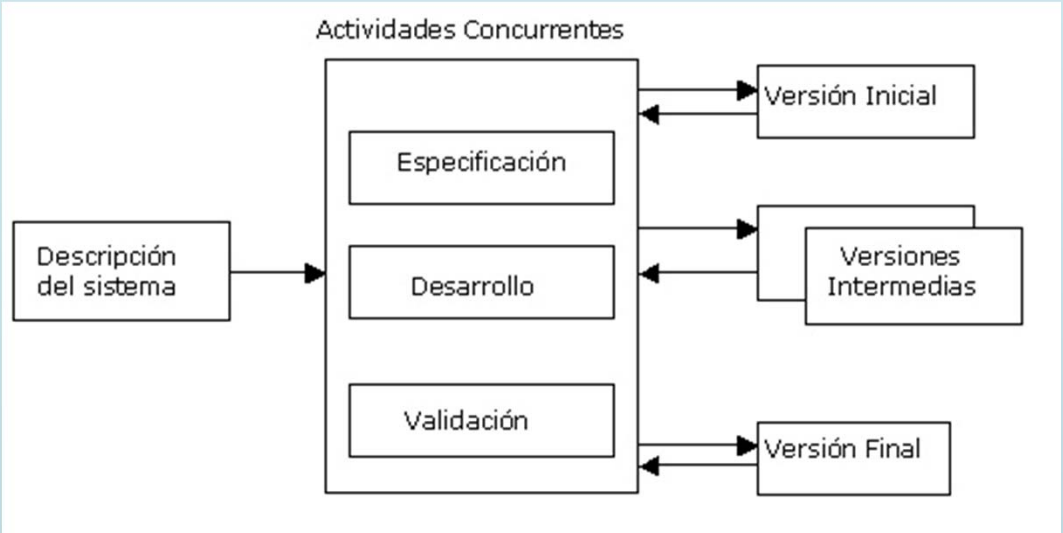
- **Conocimiento e identificación** de las distintas monedas y billetes de euro.
- Aprendizaje del **valor** de las monedas y billetes para comprar un producto.
- Práctica de **pago de precio exacto** en una **simulación** de compra en una tienda.
- Entrenamiento visual de los cambios **recibidos** tras el pago de un producto.
- **Crear** una aplicación en Android y **comercializarla** en un futuro.
- Realizar una toma de contacto con el **desarrollo** de juegos en dispositivos móviles.

Aplicación: Características

- La aplicación desarrollada tiene las siguientes características:
  - Sigue un diseño de juego.
  - Es sencilla de usar.
  - Intuitiva.
  - Dinámica.
  - Funciona en un amplio rango de versiones de Android.
  - Se puede emplear un gran número de terminales.

Desarrollo: Metodología

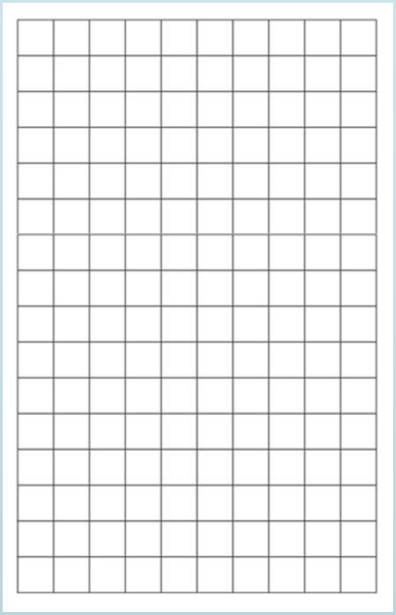
- La aplicación ha sido desarrollada siguiendo el modelo iterativo incremental.



- El mejor método para un proyecto pequeño, de una sola persona en este caso, y en el que se tiene muy en cuenta la opinión del usuario.

Desarrollo: Interfaz gráfica

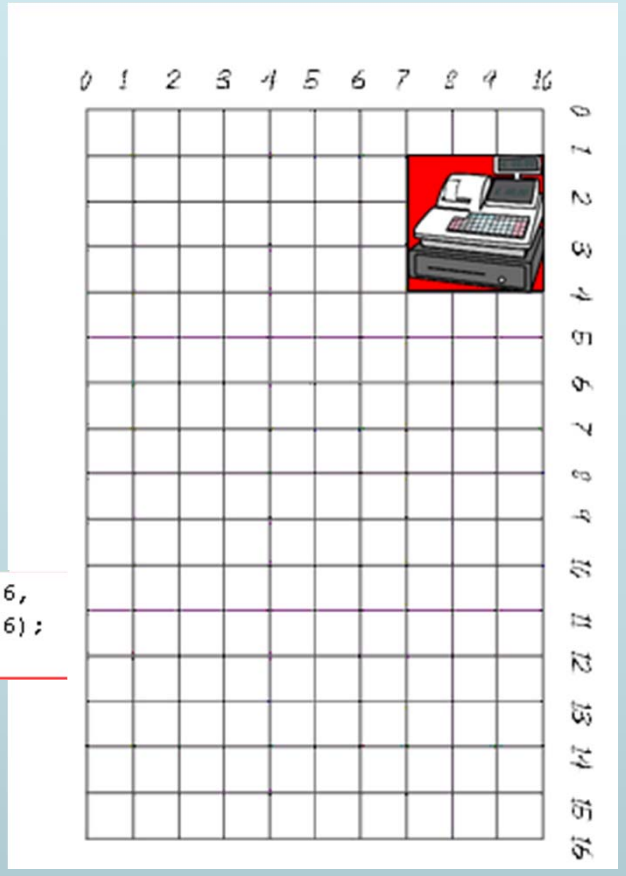
- **Ajuste dinámico** necesario para múltiples resoluciones de pantalla.



```
WindowManager wm = (WindowManager) context
        .getSystemService(Context.WINDOW_SERVICE);
Display display = wm.getDefaultDisplay();

ancho = display.getWidth();
alto = display.getHeight();
```

Desarrollo: Interfaz gráfica

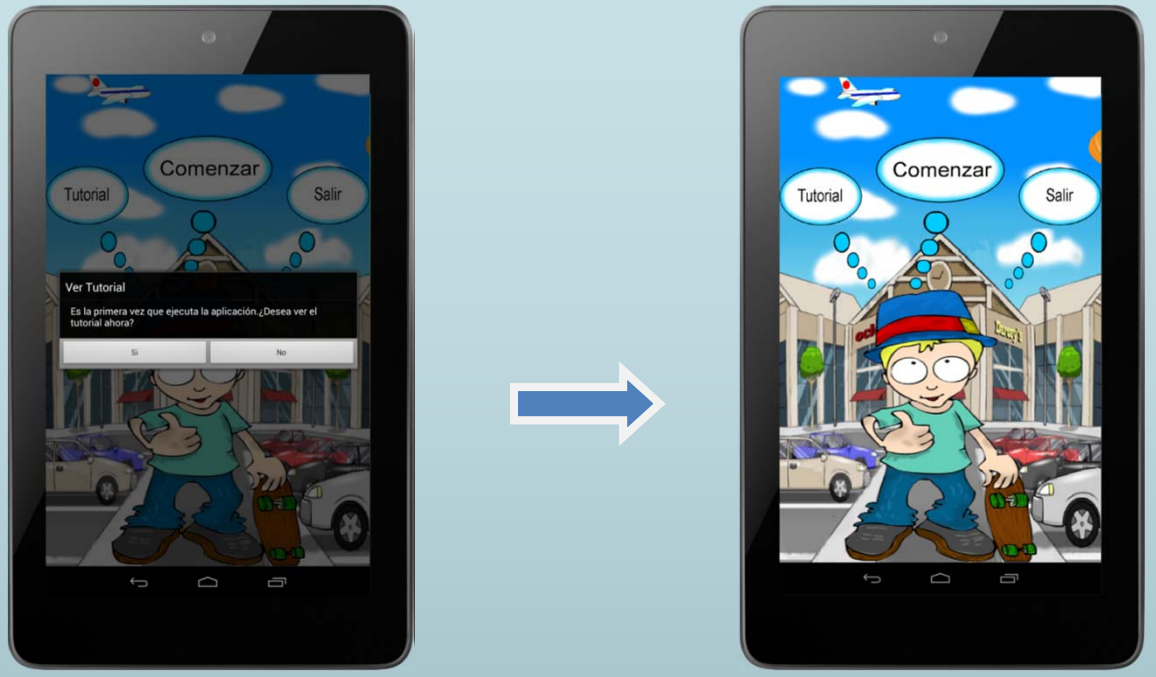



```
dst.set(canvas.getWidth() * 3 / 10, canvas.getHeight() * 2 / 16,
        canvas.getWidth() * 7 / 10, canvas.getHeight() * 7 / 16);
canvas.drawBitmap(comenzar, null, dst, null);
```

Desarrollo: Interfaz gráfica

- Diferentes niveles de la aplicación.

- Pantalla de inicio:



Desarrollo: Interfaz gráfica

- El tutorial:

Desarrollo: Interfaz gráfica

- Pantalla de selección de temas:

Desarrollo: Interfaz gráfica

- Pantalla del tema 1 y homóloga del 3:

Desarrollo: Interfaz gráfica

- Pantalla de premio:

Desarrollo: Interfaz gráfica

- Pantalla del tema 2 y homologa del 4

Desarrollo: Interfaz gráfica

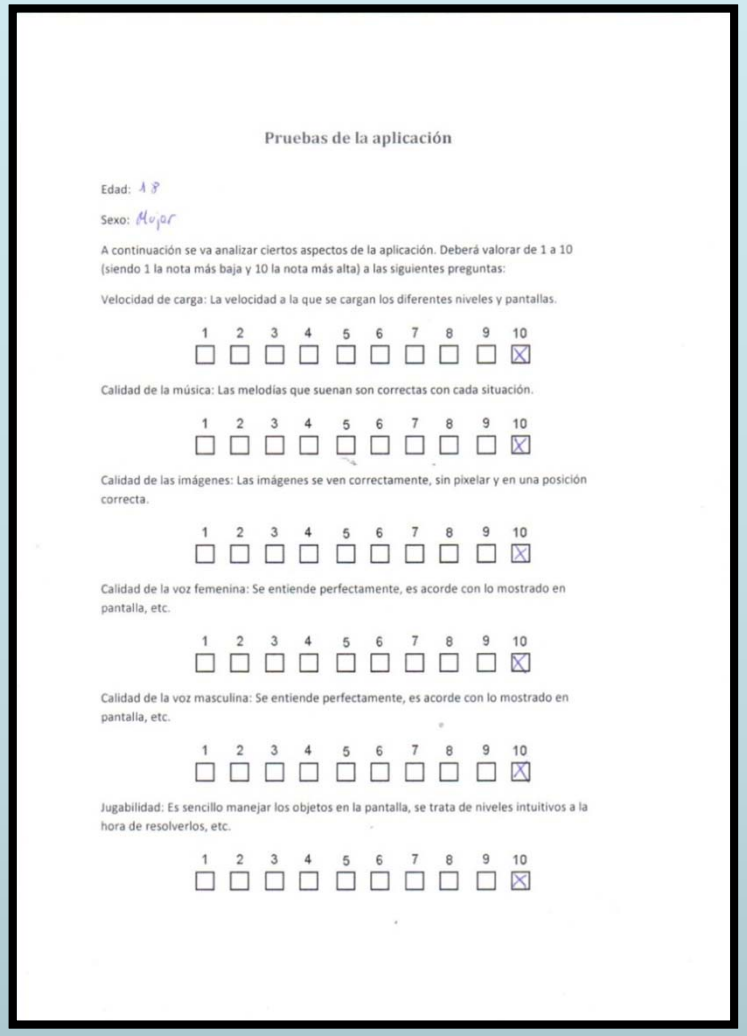
- Tema 5:

Pruebas

- Para realizar las pruebas finales de la aplicación se ha contado con la ayuda de una usuaria de 18 años con Síndrome de Down, de un usuario de 27 años usuario habitual de videojuegos y otro de 55 años, el cual nunca ha jugado a un videojuego.

- También se probó en diversos dispositivos con resoluciones distintas.

Pruebas



Pruebas de la aplicación

Edad: 18

Sexo: Mujer

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

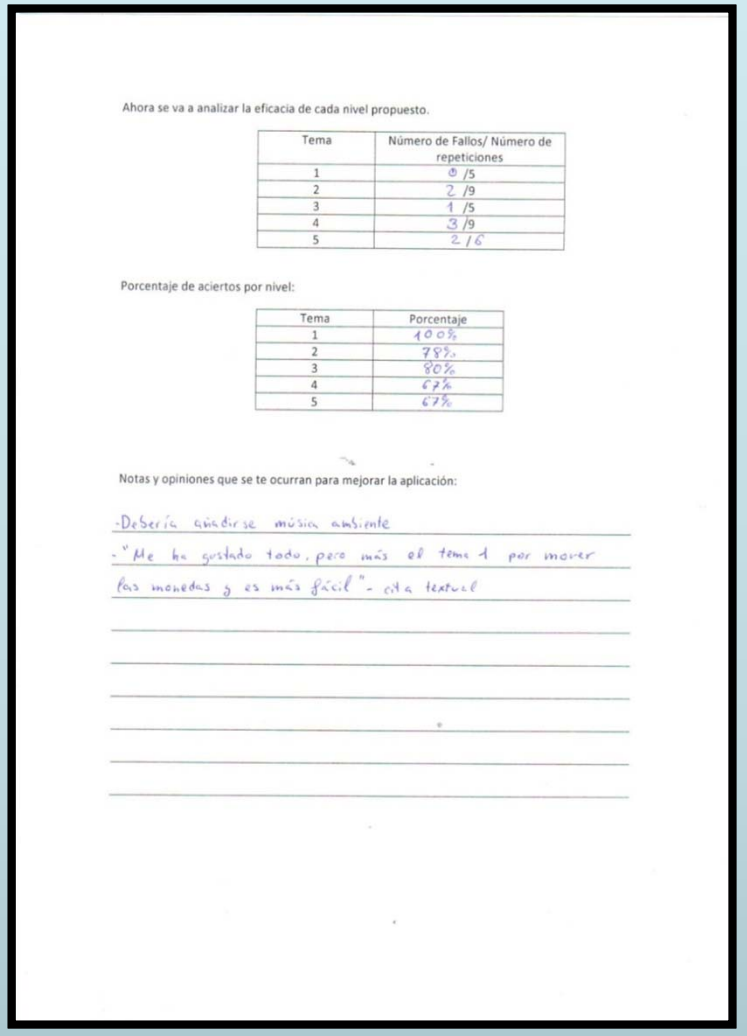
1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒



Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	2 /9
3	1 /5
4	3 /9
5	2 /6

Porcentaje de aciertos por nivel:

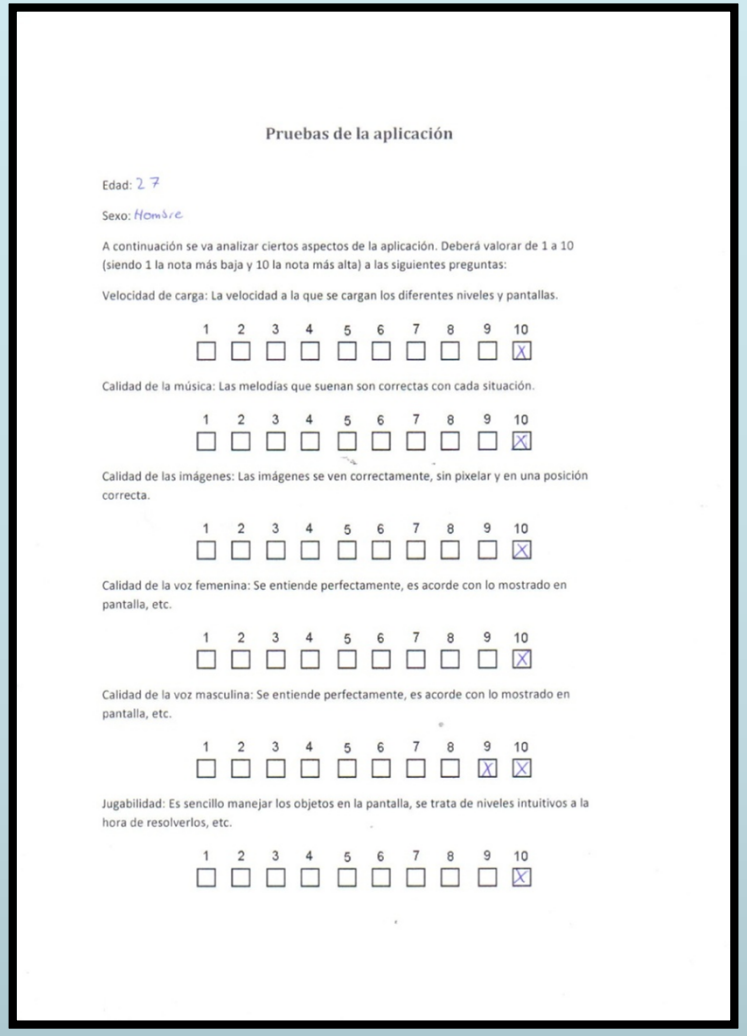
Tema	Porcentaje
1	100%
2	78%
3	80%
4	67%
5	67%

Notas y opiniones que se te ocurran para mejorar la aplicación:

-Debería añadirse música ambiente

-"Me ha gustado todo, pero más el tema 1 por mover las monedas y es más fácil" - cita textual

Pruebas



Pruebas de la aplicación

Edad: 27

Sexo: Hombre

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

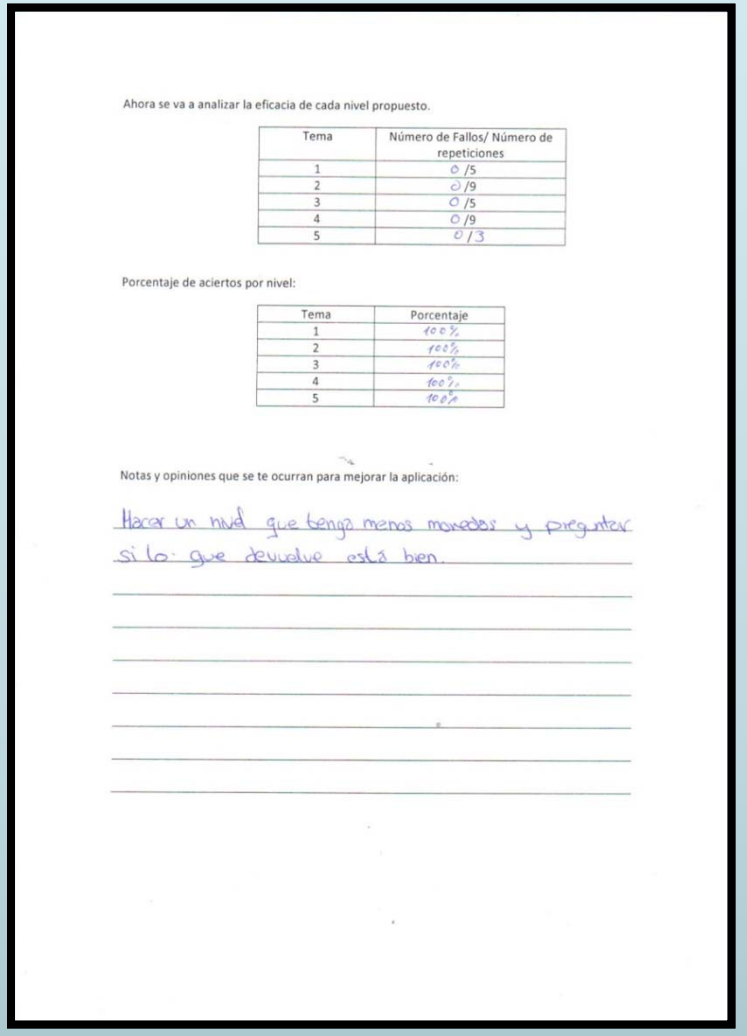
1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ ☒

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒



Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	0 /9
3	0 /5
4	0 /9
5	0 /3

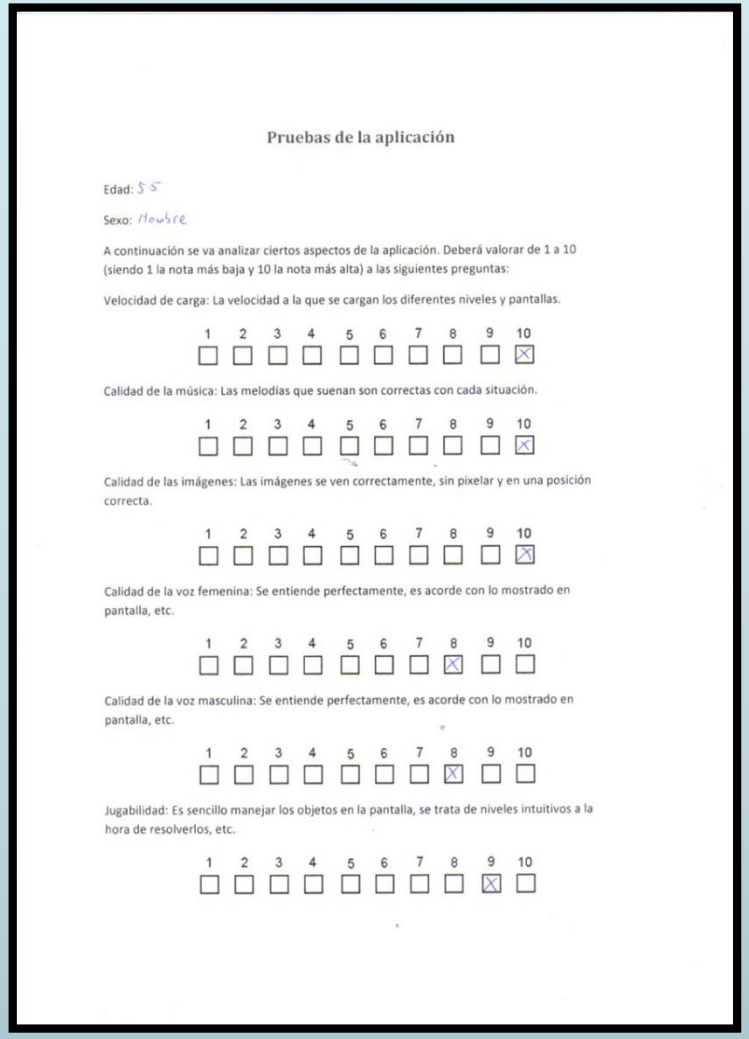
Porcentaje de aciertos por nivel:

Tema	Porcentaje
1	100%
2	100%
3	100%
4	100%
5	100%

Notas y opiniones que se te ocurran para mejorar la aplicación:

Hacer un nivel que tenga menos monedas y preguntar si lo que devuelve está bien.

Pruebas



Pruebas de la aplicación

Edad: 55

Sexo: Hombre

A continuación se va analizar ciertos aspectos de la aplicación. Deberá valorar de 1 a 10 (siendo 1 la nota más baja y 10 la nota más alta) a las siguientes preguntas:

Velocidad de carga: La velocidad a la que se cargan los diferentes niveles y pantallas.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la música: Las melodías que suenan son correctas con cada situación.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de las imágenes: Las imágenes se ven correctamente, sin pixelar y en una posición correcta.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒

Calidad de la voz femenina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

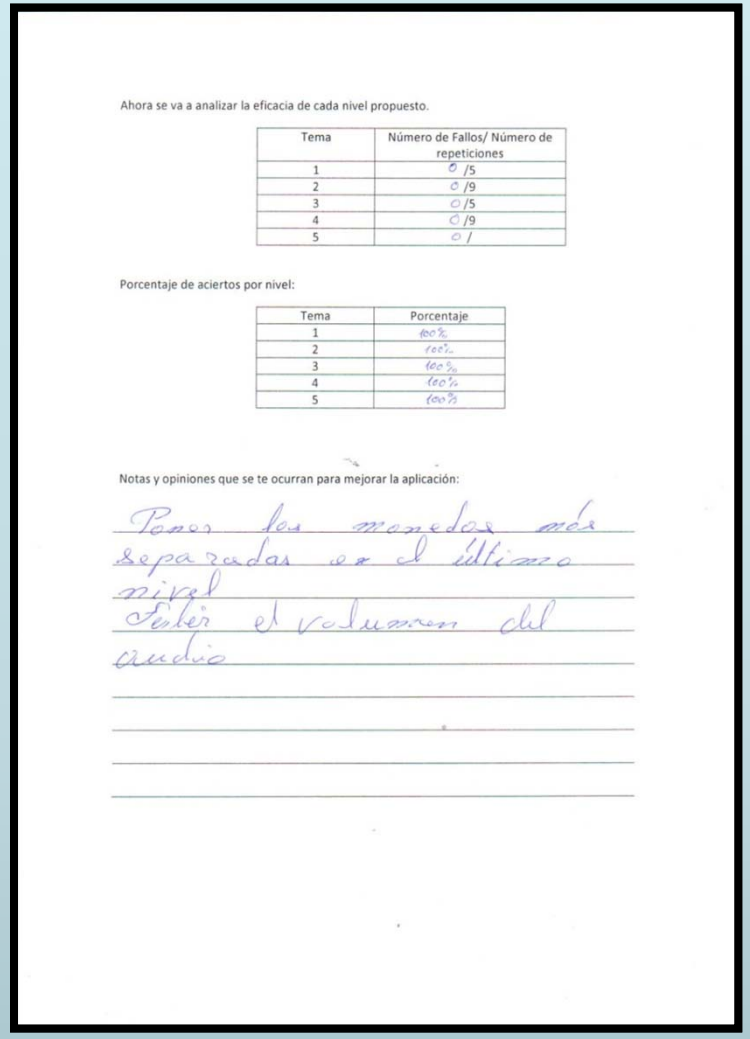
1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ ☐ ☐

Calidad de la voz masculina: Se entiende perfectamente, es acorde con lo mostrado en pantalla, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ ☐ ☐

Jugabilidad: Es sencillo manejar los objetos en la pantalla, se trata de niveles intuitivos a la hora de resolverlos, etc.

1 2 3 4 5 6 7 8 9 10
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ ☐



Ahora se va a analizar la eficacia de cada nivel propuesto.

Tema	Número de Fallos/ Número de repeticiones
1	0 /5
2	0 /9
3	0 /5
4	0 /9
5	0 /

Porcentaje de aciertos por nivel:

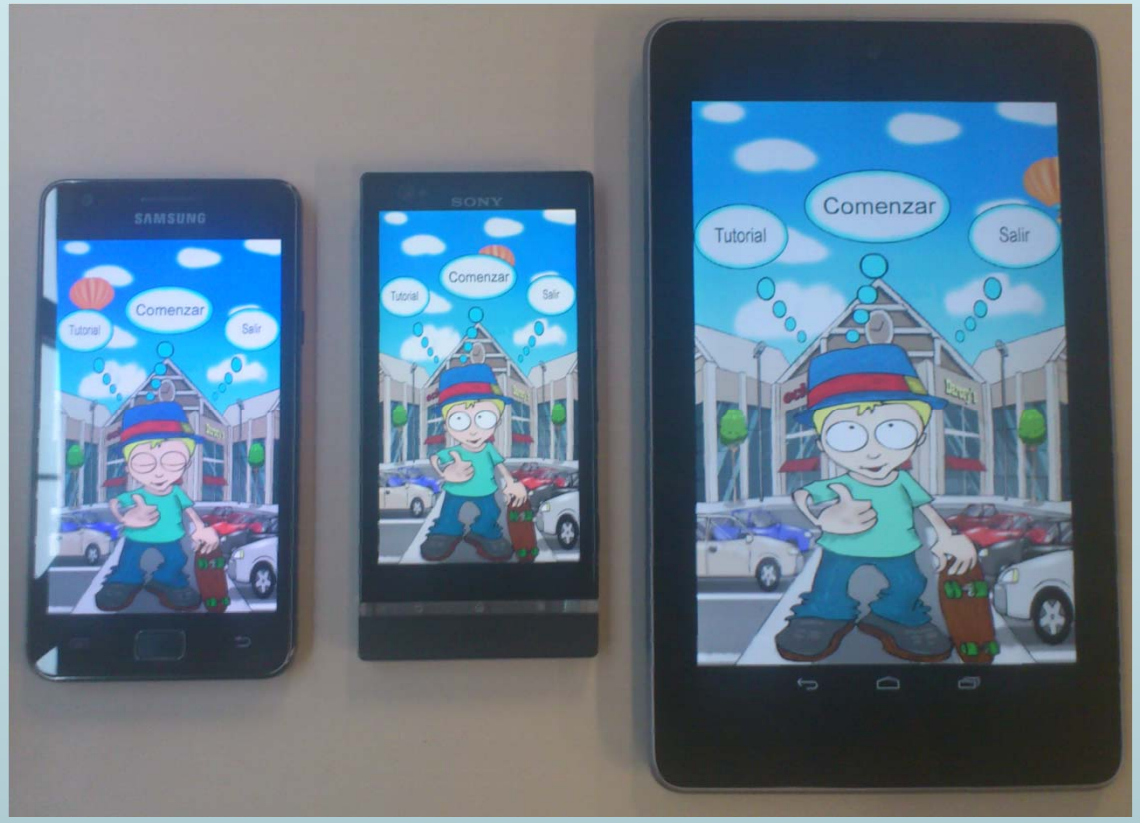
Tema	Porcentaje
1	100%
2	100%
3	100%
4	100%
5	100%

Notas y opiniones que se te ocurran para mejorar la aplicación:

Poner las monedas más separadas en el último nivel
Subir el volumen del audio

Pruebas

- La aplicación corriendo en varios dispositivos:



Presupuesto

- El cálculo realizado será la base para un posible encargo de una aplicación de éstas características, con 5 niveles distintos de juego, imágenes dinámicas, etc.

- Para su desarrollo serán necesarios:
  - Un analista.
  - Un diseñador gráfico.
  - Dos desarrolladores.

Presupuesto

- El coste de desarrollar la aplicación es de 33.572,50€ impuesto indirectos incluidos.

- Se distribuirán 2 versiones:
  - Gratuita con publicidad.
  - De pago sin publicidad a 0,99€.

- Se estima que tras 50.000 descargas la aplicación quedará amortizada.

Conclusiones

- Se comprueba que la aplicación puede ser **útil** para personas con Síndrome de Down y para cualquier persona que se inicia en el uso del euro.

- También se ha logrado **transmitir la simplicidad** de la aplicación al usuario.

- Por otro lado, se han sufrido los famosos problemas de **fragmentación** de Android, sin embargo la aplicación es compatible con versiones desde la 2.2 hasta 4.1 y funciona en un gran número de dispositivos gracias al reajuste de las imágenes en función de la resolución de sus pantallas.

- Por contraposición, he disfrutado enormemente de este sistema operativo que me ha brindado la **posibilidad de desarrollar un juego** de estas características y de adquirir una base de conocimientos para crear mis propias aplicaciones para el mismo.

Líneas futuras

- La aplicación se podría seguir desarrollando, añadiéndole funcionalidades como puede ser la posibilidad de que el usuario introduzca sus propias imágenes y precios a los productos, con la finalidad de otorgarle mayor realismo a la aplicación, siempre que se sigan cumpliendo los requisitos.

- Finalmente, la aplicación supone una clara alternativa a distintas aplicaciones de esta temática disponibles en los diversos market, tanto de Android como de su competidor más directo, iOS.

Aplicación de aprendizaje de tareas cotidianas en personas con discapacidad mediante dispositivos móviles

Asier Alejo Siles