ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

“DESARROLLO SOFTWARE DE APLICACIÓN ANDROID DE JUEGOS COGNITIVOS PARA NIÑOS CON TDAH”

Nekane Bastero Huarte

Jose Javier Astrain Escola

Pamplona, 28-06-2013

Contenido

1. Introducción

El objetivo de este Proyecto Fin de Carrera ha sido colaborar con la empresa Job Accommodation, Ingeniería y Consultoría para la Discapacidad en el desarrollo software de una aplicación para Android que sirva como apoyo a terapeutas y pedagogos en su trabajo diario con niños con TDAH (Trastorno de Déficit de Atención e Hiperactividad).

La finalidad de esta aplicación ha sido recopilar varios juegos empleados habitualmente en la terapia con niños con TDAH, almacenar los resultados obtenidos por los niños y ofrecer una herramienta que permita estudiar su evolución.

El trastorno de TDAH se trata de un trastorno del comportamiento caracterizado por distracción moderada a grave, períodos de atención breve, inquietud motora, inestabilidad emocional y conductas impulsivas.

Las principales características de este trastorno son:

- Actividad excesiva e inapropiada en relación a la tarea propuesta.
- Poca atención mantenida.
- Dificultad para inhibir impulsos.
- Bajo rendimiento escolar y autoestima.

Un 67% de los niños diagnosticados continúan en su edad adulta, por tanto es importante el uso de terapias para mejorar en todo lo posible el estado del usuario.

1.1. Antecedentes

La aplicación ya estaba empezada cuando me asignaron el proyecto, el estado en el que se encontraba la aplicación era desastroso, todo se encontraba mezclado, los nombres de las clases y de las variables no eran nada intuitivas, los juegos no iban bien, la base de datos estaba creada (sólo algunas cosas) de manera local en cada dispositivo y por tanto los tutores o pedagogos no podían hacer las consultas pertinentes.

1.2. Formulación del Problema

El trastorno de TDAH se caracteriza por la distracción moderada a grave, períodos de atención breve, inquietud motora, inestabilidad emocional y conductas impulsivas. Habitualmente, los síntomas empeoran en las situaciones que exigen una atención o un esfuerzo mental sostenidos o que carecen de atractivo o novedad.
Este trastorno se identificó primero en la edad infantil. Sin embargo, se fue reconociendo su carácter crónico, ya que persiste y se manifiesta hasta después de la adolescencia. Los estudios de seguimiento a largo plazo han demostrado que entre el 60 % y el 75 % de los niños con TDAH continúa presentando los síntomas hasta que son adultos.

El diagnóstico es complejo y debe basarse en la evaluación clínica realizada por un médico experto en el reconocimiento y tratamiento del mismo. Dicha evaluación debe obtenerse tanto de la observación de la conducta del niño como de la información facilitada por los

padres, profesores, familiares y amigos. Dada la evidencia de la importante carga genética del TDAH, es conveniente realizar una historia médica detallada tanto personal como familiar.

Los usuarios con este tipo de trastorno suelen ser medicados con una combinación de medicamentos y psicoterapia. Hay diferentes tipos de medicamentos para el THDA que se pueden usar solos o combinados.

Los psicoestimulantes (también conocidos como estimulantes) son los fármacos que más comúnmente se utilizan para el TDAH. Aunque estos fármacos se denominan estimulantes, realmente tienen un efecto tranquilizante en las personas con este trastorno.

Estos fármacos abarcan:

- Anfetamina-dextroanfetamina (Adderall)
- Dexmetilfenidato (Focalin)
- Dextroanfetamina (Dexedrine, Dextrostat)
- Lisdexanfetamina (Vyvanse)
- Metilfenidato (Ritalina, Concerta, Metadate, Daytrana)

Se ha demostrado que los usuarios mejoran razonablemente con una serie de tareas en forma de juegos. Es muy importante realizar este tipo de tareas para la mejora de:

- Atención dividida y simultánea
- Atención focalizada y selectiva
- Control de la impulsividad
- Atención sostenida y vigilada
- Inhibición
- Planificación y organización
- Flexibilidad cognitiva
- Fluidez visual y verbal

Por tanto, en colaboración con pedagogos especializados en este tipo de trastorno, se propuso realizar una aplicación que facilite el uso de juegos que abarquen diferentes áreas del trastorno. Esto facilita a los tutores y pedagogos un seguimiento del usuario para evaluar si mejora o empeora.

Las ventajas que presenta este tipo de aplicación para los usuarios con TDAH son:

- Facilidad para jugar en cualquier parte.
- Pueden jugar cuando ellos quieran.
- Juegan sin ningún tipo de presión porque pueden realizar los ejercicios solos o acompañados.

Las ventajas de cara al tutor o pedagogos:

- Pueden consultar los datos en cualquier momento y por tanto ver la evolución a tiempo real sin necesidad de establecer constantemente citas en las que el usuario se puede llegar a cansar.

- Pueden valorar los datos obtenidos y establecer cambios o soluciones respecto a los resultados obtenidos.

1.3. Estado del arte

Se han tomado como referencia algunas aplicaciones ya creadas en Play Store. Se ha observado que muchas de ellas no tienen en cuenta los colores que utilizan, que tengan las imágenes adecuadas, menús fáciles de entender… En resumen, que sean aplicaciones para todo tipo de usuarios.
Muchas de ellas no tienen buenos controles de movimiento, no se muestran correctamente las puntuaciones obtenidas y sobretodo no hay un control de las mismas que pueda ayudar a personas con trastorno TDAH a mejorar realizando este tipo de ejercicios.

1.4. Descripción Propuesta

La aplicación que se va a desarrollar debe ser diferente al resto, debe destacar por ser una aplicación de fácil entendimiento para todo tipo de usuario en la que cada juego refuerce, sin que el usuario sea demasiado consciente, ciertos trastornos derivados del TDAH.
Los aspectos clave de este proyecto deben ser:

- Los menús deben de tener botones grandes y sin demasiados elementos a su alrededor que puedan distraer en exceso la atención.
- Los colores deben estar debidamente estudiados, ya que los colores juegan un papel fundamental en la concentración sobre la actividad que se está realizando.
- Las imágenes deben ser fácilmente reconocibles por cualquier tipo de usuario.
- Los datos deben de ser volcados en una base de datos para que los tutores o cualquier profesional puedan acceder a ellos y poder hacer un seguimiento del usuario.

Todas las características que los pedagogos añadan o cambien serán integradas al proyecto ya que ellos saben mejor que nadie como debe ser la estética de la aplicación para que resulte efectiva.

Hay que tener en cuenta que los expertos en cromoterapia recomiendan el color amarillo en tonos pasteles alternados con otros colores ya que favorece la concentración y desarrollo intelectual. Los colores frescos (azul, verde o combinación entre ambos) favorecen fijar la concentración debido a que transmiten un ambiente de tranquilidad y relajación.
No es recomendable el uso del color rojo en niños hiperactivos en situaciones donde es necesaria la concentración, como leer.
Los colores que debería tener la aplicación deberían regirse por la siguiente tabla:

Naranja Combina los efectos de los colores rojo y amarillo: Energía y alegría. Las tonalidades suaves expresan calidez, estimulan el apetito y la comunicación, mientras que las tonalidades más brillantes incitan la diversión y la alegría. Puede ser considerado para el cuarto de juego de los niños en combinación con colores neutros.

Azul Es un color muy importante para calmar a las personas, se trata de un color frío que produce paz y sueño. Es utilizado en tono pastel para relajar, para ambientar cuartos, camas, etc.

Amarillo Estimula la actividad mental. Se utiliza el color amarillo en niños con gran dispersión, poca concentración. Utilizado en tono pastel en escritorios, libros, útiles para promover actividad intelectual, en ambientes en donde trabajan niños con dificultades de aprendizaje o fatiga mental. También es un color que inspira energía y optimismo
Violeta Se trata de un color místico, especialmente importante en la meditación, la inspiración y la intuición. Estimula la parte superior del cerebro y el sistema nervioso, la creatividad, la inspiración, la estética, la habilidad artística y los ideales elevados.
Verde El verde hace que todo sea fluido, relajante. Produce armonía, poseyendo una influencia calmante sobre el sistema nervioso.
Celeste Tiene un poder sedante, relajante, analgésico y regenerador.

1.5. Metodología y Planificación

Lo primero que se ha hecho ha sido elegir en qué tipo de sistema se va a basar la aplicación. Se decidió utilizar el sistema basado en Linux, Android, ya que tienen infinidad de dispositivos en los que poder volcar la aplicación.

Una vez elegido el sistema operativo, en una reunión con los pedagogos y con el coordinador de proyecto, Patxi Fabo, se han decidido los tipos de juegos que se van a desarrollar. Los juegos se han elegido de acuerdo a las necesidades principales de los usuarios y los juegos que han creído oportuno los terapeutas desarrollar en primer lugar. Por tanto su creación ha sido decisión del grupo de pedagogos. Los 5 juegos elegidos para desarrollar en primer lugar han sido, Memory, Torres de Hanoi, Parejas, Stroop y Simón, de los cuales hay alguno que ya está empezado.

Las dos o tres primeras semanas se dedicarán a la adaptación y aprendizaje de la programación orientada al sistema operativo Android.

Seguidamente se realizará el diseño de cada juego en colaboración con los pedagogos especializados en este trastorno. Esta gran etapa puede durar unos 6 meses ya que se trata de 5 juegos, alguno de ellos con dificultades para entender lo que ya está creado.

Una vez desarrollados los juegos se diseñará y creará la base de datos en un tiempo de 1 mes o mes y medio. A la base de datos se accederá de manera remota con el servidor de la empresa, esta parte puede ser costosa ya que no hay apenas información concreta de cómo realizar este tipo de conexión. Además hay que integrar todas las consultas y conexiones en las clases de cada juego.

La fase de pruebas finales se realizará una vez estén realizados todos los juegos y la base de datos, para comprobar el correcto funcionamiento de la aplicación. Las pruebas también se irán desarrollando conforme se van desarrollando los diferentes juegos.

La última fase será la de mantenimiento, la cual no tiene una duración limitada en el tiempo ya que la aplicación necesitará actualizaciones y mejoras según las necesidades de los usuarios o de las exigencias de los pedagogos.

2. Análisis y Diseño

2.1. Requisitos

Los requisitos que han sido declarados a continuación han sido definidos por el coordinador del proyecto, Patxi Fabo, con la colaboración de pedagogos. Tras varias reuniones, se redactaron las necesidades fundamentales del proyecto y me las transmitieron una vez llegué a la empresa.

Requisitos funcionales, que expresan la capacidad de acción del sistema.
La aplicación deberá permitir la definición de diferentes perfiles de usuario dentro del sistema.

Tanto los menús como los juegos deben ser lo más sencillos posibles debido a que esta aplicación está orientada a un trastorno en el que los usuarios tienen déficit de atención, por tanto no se deben integrar muchas imágenes en los menús, muchos colores, en resumen, no sobrecargar la aplicación.

Los juegos no deben de ser excesivamente largos en el tiempo ya que los usuarios con este tipo de trastorno se caracterizan por perder el interés por las cosas con facilidad.

Los juegos elegidos deben ser atractivos para el usuario, ya que es necesario que juegue a todos los juegos propuestos.

La configuración de usuarios, como eliminación o modificación de usurarios, se hace en la parte del tutor de la aplicación.

Requisitos no funcionales, características del proceso de desarrollo:

Los soportes que se van a utilizar para la utilización de la aplicación van a ser dispositivos basados en el sistema operativo android, por tanto, una de las exigencias que se han impuesto al proyecto es la de utilizar programación basada en el lenguaje de programación java.

Requisitos Hardware

Como se ha dicho anteriormente se va a utilizar el sistema operativo Android, por lo que es necesario cualquier dispositivo basado en este sistema.

Las tablets o móviles utilizados deben de tener conexión a internet, puede realizarse la conexión mediante wifi o datos.

Se necesitará un ordenador (servidor) que contenga la base de datos para poder volcar los datos sobre él y un router que será el que redirige las peticiones al servidor.

Requisitos Software

En primer lugar habrá que definir el sistema operativo en el que la aplicación se desarrollará. Como se ha mencionado anteriormente el sistema operativo que se va a usar será Android, con una versión mínima de sistema 2.1 (Éclair).

Es necesario tener instalado un entorno de desarrollo, en este caso Eclipse Helios, para desarrollar la aplicación y posteriormente realizar las pruebas y el mantenimiento de la aplicación.

2.2. Estructura del Sistema

La estructura del sistema se define mediante el diagrama de la Figura 1. De esta manera se puede tener una idea clara de la utilización de la aplicación.

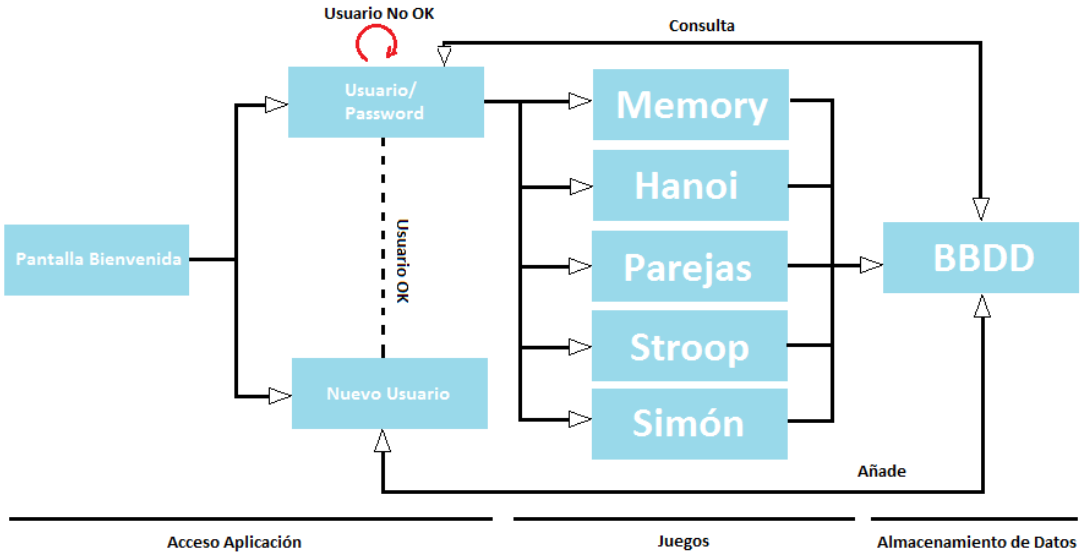


Figura 1: Estructura del sistema

En la pantalla de bienvenida está la opción de introducir el usuario y password de los usuarios ya registrados ó ingresar un nuevo usuario. Si al comprobar el usuario en la base de datos nos devuelve que no es correcto informa y vuelve a la misma pantalla de ingreso de usuario. Una vez accedemos al sistema como usuario registrado podemos acceder a los juegos.

Los juegos volcarán los datos en la base de datos siempre que se juegue. En el que caso de no disponer de internet en el dispositivo se podrá jugar, pero los datos se perderán. Este apartado se menciona en líneas futuras como mejora importante en la aplicación.

Se ha necesitado un servidor donde alojar la base de datos con conexión a internet. El servidor se encuentra alojado en la empresa Job Accommodation y lo tienen configurado de tal manera que tiene la suficiente seguridad para que no se pueda acceder de ninguna manera a él.

2.3. Diagramas casos de uso

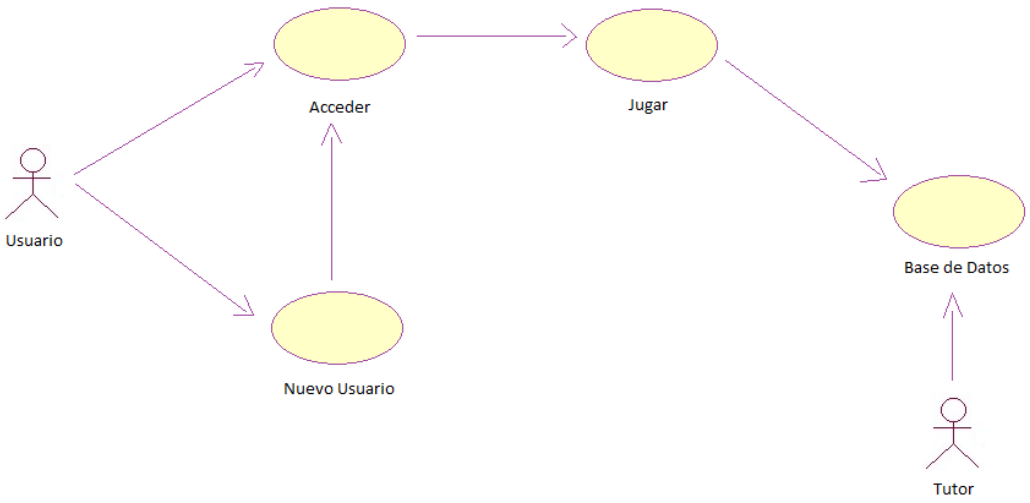


Figura 2: Casos de uso

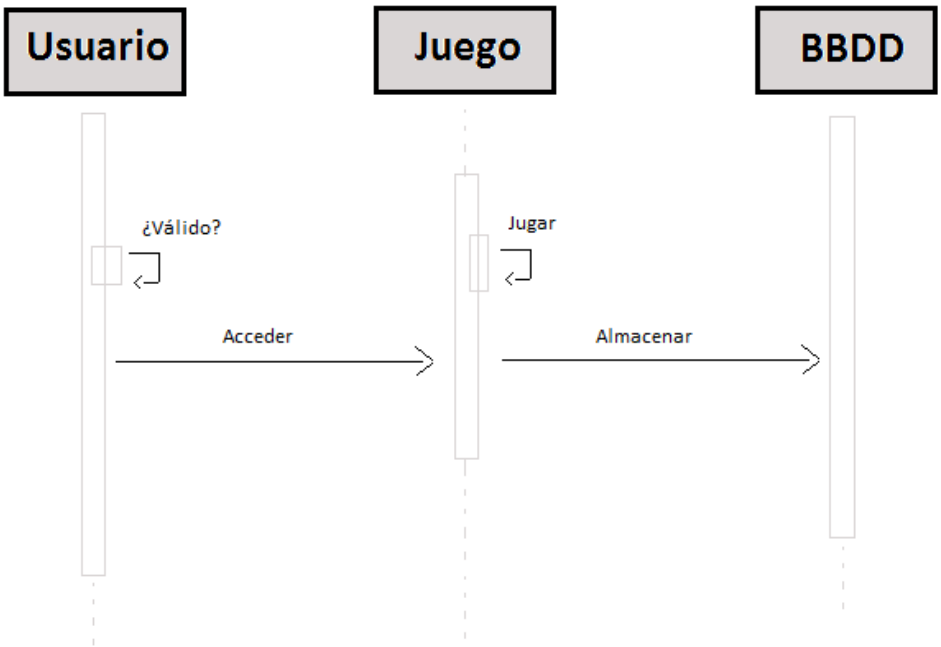


Figura 3: Diagrama de secuencia

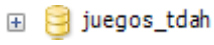
2.4. Base de Datos

La manera en la que se han guardado todos los datos ha sido mediante una conexión remota con la base de datos alojada en el servidor de la empresa.

Esto se ha desarrollado así para que tanto los tutores como pedagogos que quieran analizar los resultados de los usuarios lo hagan desde cualquier parte sin necesidad de tener cada dispositivo de cada usuario para la comprobación de los datos.

De esta manera además pueden consultar desde cualquier parte la base de datos e interactuar con ellos.

Se ha creado la base de datos dentro del servidor de la empresa. La base de datos se ha llamado juegos_tdah, la conexión se establece mediante la dirección IP pública del servidor y el puerto 3306. Se le ha añadido un usuario y contraseña para poder acceder a ella.



Se ha utilizado el sistema de gestión de bases de datos relacional MySQL. Se ha utilizado este sistema debido a que en la empresa siempre se ha trabajado con este sistema, además de que una de sus principales características es que es multiusuario y multiplataforma.

La conexión con las aplicaciones queda reflejada en el apartado 2.4. dónde se especifica la conexión entre la aplicación y la base de datos.

Se han creado 3 tablas para el almacenamiento de los datos. No han sido necesarias más tablas ya que los datos que se introducen son escasos. Los datos introducidos han sido metidos de manera sencilla para que una compañera en Madrid obtenga esos datos de manera fácil y trabaje con ellos para realizar la segunda parte de la aplicación.

La tabla usuarios tiene como clave principal el id_usuario que se autoincrementa de manera que no se pueda dar el caso de dos usuarios con el mismo id ya que daría un error.

Todos los campos, menos el diagnóstico y apellido2, son no nulos.

```
Table usuarios
==============
id_usuario, nombre, apellido1, apellido2, pass, nick, fecha_conex, fecha_registro, edad, diagnostico, tipo
--------------
id_usuario    INT
nombre        VARCHAR
apellido1     VARCHAR
apellido2     VARCHAR
pass          VARCHAR
nick          VARCHAR
fecha_conex   VARCHAR
fecha_registro VARCHAR
edad          INT
diagnostico   VARCHAR
tipo          VARCHAR
```

Las tablas partidas y juegos son muy sencillas. La clave principal en partidas es id_partida y la de juegos id_juego.

```
Table partidas
==============
id_partida, fecha_comienzo, hora_comienzo, toque, ok, finalizado, juego
--------------
id_partida    INT
fecha_comienzo VARCHAR
hora_comienzo VARCHAR
toque         VARCHAR
ok            VARCHAR
finalizado    VARCHAR
juego         VARCHAR
```

```
Table juegos
============
id_juego, nombre, punttotal, nvecesjug
------------
id_juego      VARCHAR
nombre        VARCHAR
punttotal     INT
nvecesjug     INT
```

Se ha realizado la comprobación de que la base de datos se ha creado correctamente. Para ello se ha accedido al servidor y se ha seleccionado la base de datos “juegos_tdah”, sacando por pantalla la información de la Figura 4.

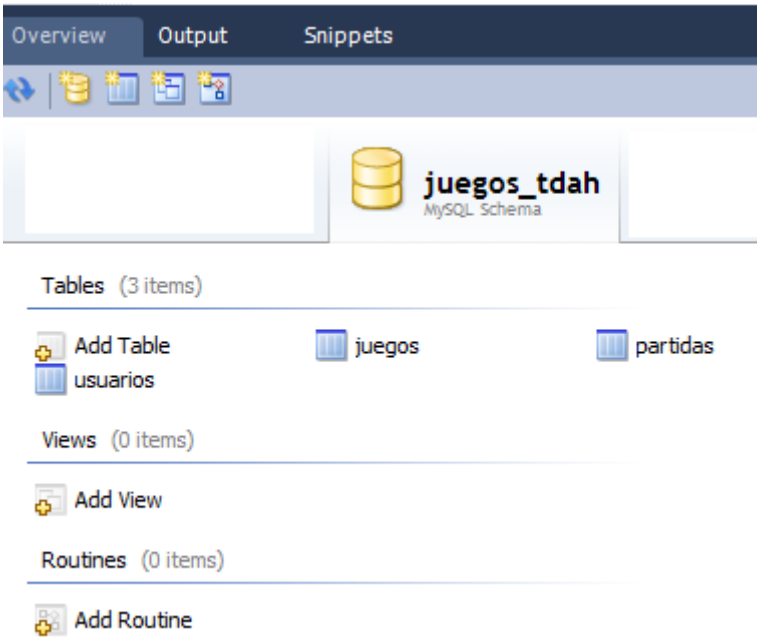


Figura 4: Base de datos creada

2.5. Conexión

La conexión con la base de datos se ha desarrollado de la manera siguiente:

Figura 5: Conexión con Base de Datos

Desde el dispositivo se hace una petición a la base de datos, el enlace entre ambos es internet.

De manera más específica se resume en la Figura 6.

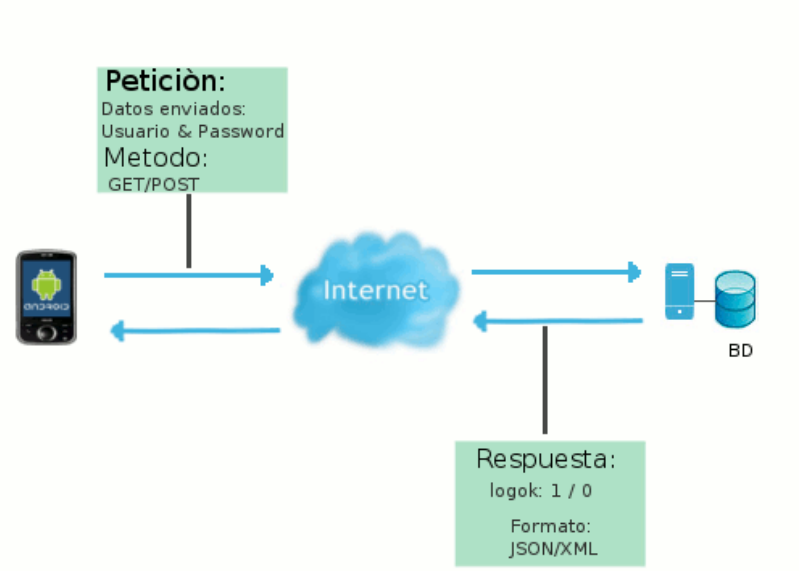


Figura 6: Conexión Detallada

Se hace una petición, en este caso mediante el método POST, incluyendo los datos a enviar. El envío se realiza con JSON. Mediante internet nos conectamos con el servidor donde se encuentra la base de datos dándonos la respuesta a la petición. Igualmente, la respuesta es devuelta con JSON.

Se ha utilizado JSON ya que se trata de un formato ligero de intercambio de datos y, además, no requiere del uso de XML.

Lo primero que se ha hecho es recoger los datos que la aplicación quiere que introduzcamos o consultemos en la base de datos. Esto se realiza en todas las clases necesarias.

```
Httppostaux post;
String IP_Server="83.173.132.215:8081";//IP DE NUESTRO PC
String URL_connect="http://"+IP_Server+"/addpartidas.php";
String URL_connect2="http://"+IP_Server+"/addpartidasinicio.php";
```

Aquí se especifica dónde tiene que realizar la conexión con una dirección IP y el puerto, en este caso se ha utilizado la IP pública del router de la empresa. Se ha utilizado el puerto 8081 ya que es un puerto que queda libre y no está prevista su utilización. La conexión url se hace llamando al php que interesa. En este caso se utilizan dos php por tanto se realizan dos conexiones.

Se ha necesitado una clase auxiliar (Httppostaux.java) para el envío de peticiones a nuestro sistema y del manejo de respuesta.

Esta clase auxiliar crea y realiza la conexión, convierte la respuesta y la devuelve.

```
public JSONArray getserverdata(ArrayList<NameValuePair> parameters, String urlwebserver ){
    httppostconnect(parameters,urlwebserver);
    if (is!=null){
        getpostresponse();
        return getjsonarray();
    }else{
        return null;
    }

}
```

Se conecta vía http y envía un post.

```
private void httppostconnect(ArrayList<NameValuePair> parametros, String urlwebserver){
try{
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost(urlwebserver);
httppost.setEntity(new UrlEncodedFormEntity(parametros));
//ejecuto peticion enviando datos por POST
HttpResponse response = httpclient.execute(httppost);
HttpEntity entity = response.getEntity();
is = entity.getContent();

}catch(Exception e){
Log.e("log_tag", "Error in http connection "+e.toString());
}

}
```

Creo la petición http. Se ejecuta la petición enviando los datos mediante el método post.

```
public void getpostresponse(){
 try{
 BufferedReader reader = new BufferedReader(new InputStreamReader(is,"iso-8859-1"),8);
 StringBuilder sb = new StringBuilder();
 String line = null;
 while ((line = reader.readLine()) != null) {
 sb.append(line + "\n");
 }
 is.close();
 result=sb.toString();
 Log.e("getpostresponse"," result= "+sb.toString());
 }catch(Exception e){
 Log.e("log_tag", "Error converting result "+e.toString());
 }
}

public JSONArray getjsonarray(){
 //parse json data
 try{
        JSONArray jArray = new JSONArray(result);
        return jArray;
 }
 catch(JSONException e){
 Log.e("log_tag", "Error parsing data "+e.toString());
 return null;
 }
}
```

Se pasa el resultado obtenido a String y se devuelve con JSON.

Volviendo a la clase desde donde se ha llamado a Httppostaux, una vez se ha inicializado la llamada hay que crearla.

```
post=new Httppostaux();
```

Se crea un ArrayList donde se van a introducir los valores que se quieren mandar.

```
ArrayList<NameValuePair> postparameters4send= new ArrayList<NameValuePair>();
postparameters4send.add(new BasicNameValuePair("hora_comienzo",salida));
postparameters4send.add(new BasicNameValuePair("juego","Parejas"));
postparameters4send.add(new BasicNameValuePair("fecha_comienzo",formatteDate));
JSONArray jdata=post.getserverdata(postparameters4send, URL_connect2);
```

En este caso en concreto se han introducido tres datos y se ha creado la conexión . La respuesta la guardamos en una variable JSONArray llamada jdata.

Este procedimiento se realiza en todas las clases que necesitemos. La llamada siempre es la misma, lo que hace que se haga una cosa u otra en la base de datos son los archivos php.

Una vez hecha la petición tenemos que configurar el router para que deje pasar nuestra llamada.

Se ha activado el puerto 8081 y se ha redirigido a la IP del servidor, 192.168.1.13. Una vez tenemos todas las peticiones redirigidas correctamente, se instala el paquete XAMP. Se podría utilizar otro paquete similar como WAMP o incluso Apache y MySQL por separado.

Se ha elegido el paquete XAMP por comodidad, ya que incluye Apache y MySQL.

Una vez instalado XAMP, cambiamos el puerto de acceso de Apache que por defecto tiene el 80. La manera de cambiar el puerto es accediendo a la carpeta de archivos de configuración de

apache, dentro de XAMP, y modificando el 80 por 8081. Guardamos, reiniciamos XAMP y se comprueba que el puerto ha sido modificado.

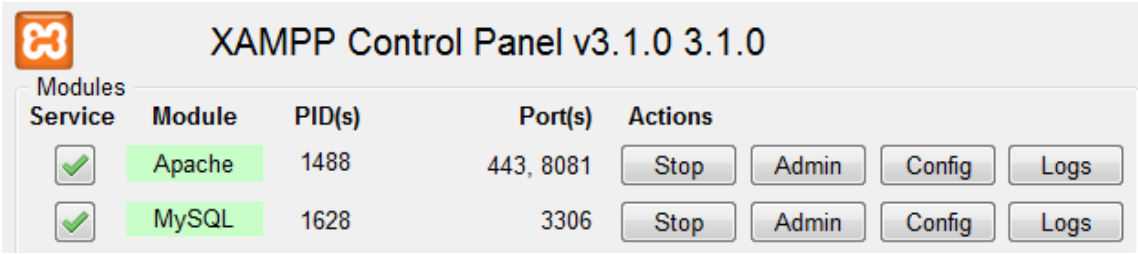


Figura 7: Panel de Control XAMP

Se ha creado una base de datos llamada juegos_tdah donde se han creado todas las tablas y relaciones necesarias (Figura 4).

Para que se pueda acceder a la base de datos se ha creado un usuario llamado “usuario” al que se le han concedido todos los privilegios para que se pueda acceder sin que genere ningún error.

Ahora se han introducido los documentos php en la carpeta htdocs de XAMP, para que este realice la conexión.

Hay 4 archivos php:

- Login.php
- Adduser.php
- Addpartidas.php
- Addpartidasinicio.php

El archivo Login.php se encarga de la comprobación del usuario y la contraseña. Si es correcto devuelve 1, y si el resultado es incorrecto devuelve 0.

Se reciben el usuario y contraseña mediante el método POST, se recogen los valores en dos variables y se pasan los datos para su comprobación. Una vez obtenemos el resultado lo devolvemos.

```
$nick = $_POST['usuario'];
$pass = $_POST['password'];

$db = new funciones_bd();

  if($db->login($nick,$pass)){

  $resultado[]=array("logstatus"=>"0");
  }else{
  $resultado[]=array("logstatus"=>"1");
  }

echo json_encode($resultado);
```

La función login mira si ese usuario y contraseña existe. Hace una consulta. Si el resultado coincide con los datos recibidos devolvemos true, si en cambio no coincide devolvemos false.

Si la sentencia SQL falla devolvemos un mensaje de error.

```
public function login($nick,$pass){

$result=mysql_query("SELECT COUNT(*) FROM usuarios WHERE nick='$nick' AND pass='$pass'");

  if (!$result) {

    // query failed
    echo "Query failed: " . mysql_error();

  }
  else {

    $count = mysql_fetch_row($result);

  }

  if ($count[0]==0){

    return true;

  }else{

    return false;

  }

}
```

Adduser.php añade un nuevo usuario en la base de datos, comprobando que no hay un usuario igual ya registrado en la base de datos.

Recogemos los datos. Se ha tenido en cuenta que hay valores que se necesitan en formato entero, por tanto, se ha hecho la conversión oportuna. Si el usuario ha sido introducido correctamente devuelve 1 y si no 0.

```php
$fechareg = $_POST['fecha_registro'];
$fechacon = $_POST['fecha_conexion'];
$nick = $_POST['nick'];
$pass = $_POST['password'];
$nombre = $_POST['nombre'];
$apellido1 = $_POST['apellido1'];
$apellido2 = $_POST['apellido2'];
$edad = $_POST['edad'];



$edadint = (int) $edad;

$db = new funciones_BD();

if($db->isuserexist($nick,$pass)){

echo(" Este usuario ya existe ingrese otro diferente!");
}else{

if($db->adduser($nombre,$apellido1,$apellido2,$pass,$nick,$fechacon,$fechareg,$edadint))
{
$resultado[]=array("logstatus"=>"1");
}else{
$resultado[]=array("logstatus"=>"0");
}
echo json_encode($resultado);
}
```

Con la función adduser agregamos los datos en la base de datos mediante un INSERT. Si se ha introducido correctamente devolvemos true y si no se han podido introducir los datos devolvemos false.

```php
public function adduser($nombre,$apellido1,$apellido2,$pass,$nick,$fechacon,$fechareg,$edadint) {


$result = mysql_query("INSERT INTO usuarios(nombre,apellido1,apellido2,pass,nick,fecha_conex,fecha_registro,edad,diagnostico,tipo) VALUES ('$nombre','$apellidc
    if ($result) {

        return true;

    }
    else {

        return false;

    }
}
```

Se ha comprobado si el usuario a registrar existe, para evitar que haya redundancia innecesaria en la base de datos. Hacemos una consulta sobre el usuario introducido y si hay coincidencias con la base de datos indica que existe el usuario, y en cambio, si no existe, lo podemos agregar sin problemas.

```php
public function isuserexist($nick) {

    $result = mysql_query("SELECT nick from usuarios WHERE nick = '$nick'");


    $num_rows = mysql_num_rows($result); //numero de filas retornadas


    if ($num_rows > 0) {

        // el usuario existe

        return true;

    } else {

        // no existe

        return false;

    }

}
```

Se introduce un usuario y se realiza una comprobación del correcto funcionamiento.

```sql
select * from usuarios;
```

Y se obtiene:

id_usuario	nombre	apellido1	apellido2	pass	nick	fecha_conex	fecha_registro	edad	diagnostico	tipo
1	Nekane	bastero	huarte	neka	nekabh	2013-03-15	2013-03-15	24		nino

Addpatidasinicio.php se encarga de meter la fecha, hora y nombre del juego una vez hemos accedido a una partida.

```php
$fecha_comienzo = $_POST['fecha_comienzo'];
$hora_comienzo = $_POST['hora_comienzo'];
$juego = $_POST['juego'];


$db = new funciones_BD();


if($db->addpart($fecha_comienzo,$hora_comienzo,$juego))
{
$resultado[]=array("logstatus"=>"1");
}else{
$resultado[]=array("logstatus"=>"0");
}
echo json_encode($resultado);
```

Se insertan las partidas iniciales en la base de datos. Si se ha introducido correctamente se devuelve true y si no se devuelve false.

```php
public function addpart($fecha_comienzo,$hora_comienzo,$juego) {

$result = mysql_query("INSERT INTO partidas(fecha_comienzo,hora_comienzo,juego) VALUES ('$fecha_comienzo','$hora_comienzo','$juego')");
    // check for successful store

    if ($result) {

        return true;


    } else {

        return false;
    }


}
```

Se introduce una partida para su comprobación.

```
select * from partidas;
```

Se obtiene.

id_partida	fecha_comienzo	hora_comienzo	toque	ok	finalizado	juego
978	2013-04-16	12:16:11	NULL	NULL	NULL	Parejas

En cambio, Addpartidas.php añade todas las partidas, no añade la fecha ni el juego, sino que se limita a introducir la hora en la que se realiza el movimiento, si ha sido correcto o no y si se ha terminado la partida.

```php
$toque= $_POST['toque'];
$ok= $_POST['ok'];
$finalizado= $_POST['finalizado'];

$db = new funciones_BD();


if($db->addpart($ok,$toque,$finalizado))
{
$resultado[]=array("logstatus"=>"1");
}else{
$resultado[]=array("logstatus"=>"0");
}
echo json_encode($resultado);
```

Con un INSERT se introducen los datos.

```php
public function addpart($ok,$toque,$finalizado) {


$result = mysql_query("INSERT INTO partidas(toque,ok,finalizado) VALUES ('$toque','$ok','$finalizado')");
    // check for successful store
    if ($result) {
        return true;
     } else {
         return false;
    }
}
```

Se ha insertado una partida desde la aplicación alojada en la tablet.

```
select * from partidas;
```

id_partida	fecha_comienzo	hora_comienzo	toque	ok	finalizado	juego
1013	NULL	NULL	8:33:35	bien	no	stroop6
1014	NULL	NULL	8:33:36		si	stroop6

Todos los archivos php necesitan de funciones que les permitan realizar la conexión con la base de datos. En todos son las mismas.

```php
class funciones_bd {
    private $db;
    // constructor
    function __construct() {
        // connecting to database
        $this->db = new connectdb();
        $this->db->connect();
     }
     // destructor
    function __destruct() {
     }
```

```php
class connectdb {

    // constructor

    function __construct() {

}

    // destructor
    function __destruct() {

        // $this->close();

    }

    // Connecting to database

    public function connect() {

        // connecting to mysql

        $con = mysql_connect("83.173.132.215", "usuario", "usuario");

        // selecting database

        mysql_select_db("juegos_tdah");

         // return database handler

        return $con;

    }

     // Closing database connection

    public function close() {

        mysql_close();

    }
```

La más importante es la función que tiene los datos para realizar la conexión, connect().

Se le ha pasado la dirección IP pública del servidor de la empresa, y el usuario y contraseña.

Se facilita el nombre de la base de datos creada para que sepa a qué base de datos debe conectarse.

Se ha incluido una función muy importante que es la de cierre de la base de datos, close(), que se encarga de cerrar la conexión.

2.6. Adaptación pantalla

El objetivo de esta aplicación es que pueda ser utilizada en los diferentes dispositivos Android que tenemos actualmente en el mercado.
Esto conlleva que tengamos diferentes tamaños en los que adaptar nuestra aplicación, ya que si no lo hacemos de esta manera, al utilizar dispositivos grandes como 10'' o muy pequeños 3,5'' se descuadrarán tanto los botones, como el texto, etc. Esto nos puede causar tanto problemas estéticos como de usabilidad (que se nos oculten botones, textos…).

Los tamaños que podemos tener de pantallas son:

xlarge→Pantallas de al menos 960dpx720dp (10'')

large→ Pantallas de al menos 640dpx480dp (7'')

normal→ Pantallas de al menos 470dpx320dp (5'')

small→ Pantallas de al menos 426dpx320dp (3.5'')

Tras consultar varias formas de adaptar la aplicación, la elegida para realizar aplicaciones que se autoajuste al terminal ha sido la siguiente:

Lo primero creamos la siguiente estructura de directorios para los "layout":

res/layout/	Layout normal ("default")
res/layout-small/	Layout pequeño, pantallas pequeñas
res/layout-large/	Layout grande, pantallas grandes
res/layout-xlarge/	Layout muy grande, pantallas extragrandes

En el caso de no crear algún tamaño de pantalla especifico (no en nuestro caso), cogería por defecto el Layout normal, el de por defecto.

Una vez creada la estructura de carpetas, crearemos el layout(XML) y lo copiaremos a todas las carpetas. Una vez hecho esto tenemos que hacer las modificaciones de tamaño necesarias para cada dispositivo, es decir, por ejemplo en el caso de querer ver la aplicación en un móvil de 3.7'' tendremos que ir a la carpeta res/layout-small/ y modificar los xml para adaptarlos al dispositivo, en este caso de 3.7''. Una vez hecho esto para cada tipo de carpeta, el dispositivo elegirá donde debe coger el xml según su tamaño.

De la misma manera, para adaptar las imágenes habría que crear la siguiente estructura y copiar a cada una las imágenes adecuadas:

res/drawable-xhdpi/my_icon.png	//Para densidades pequeñas
res/drawable-mdpi/my_icon.png	//Para densidades medias
res/drawable-hdpi/my_icon.png	//Para densidades grandes

3. Implementación

3.1. Entorno de desarrollo

Se ha utilizado Eclipse para el desarrollo de la aplicación ya que posee un conjunto de herramientas de código abierto multiplataforma que se adapta perfectamente a las necesidades requeridas para realizar este proyecto.

Para poder volcar los resultados en la tablet facilitada por la empresa se ha tenido que instalar un driver específico. Se ha buscado el driver para la tablet bq Edison, en concreto el usb-driver, y se ha descargado e instalado.

Una vez instalado se puede volcar la aplicación en la tableta sin problemas.

3.2. Herramientas y lenguaje

El sistema operativo empleado para el desarrollo de la aplicación es Android 2.1. El lenguaje empleado ha sido Java. Además de ser el lenguaje más extendido a la hora de programar en Android, se ha elegido principalmente debido a la independencia de la plataforma.
Además la especificación J2ME está desarrollada específicamente para el mercado de dispositivos electrónicos, por tanto, se adapta a las necesidades del proyecto.

Se ha necesitado un ordenador donde se pueda instalar el software Eclipse Helios para el desarrollo de la aplicación. Un dispositivo basado en Android, en mi caso se ha utilizado la Tablet BQ Edison.

3.3. Problemas

El mayor problema ha surgido en el entendimiento del código inicial de algunos juegos que ya estaban creados. Los juegos no funcionaban bien ninguno y la introducción a la base de datos era caótica. La parte de la base de datos tenía el principal problema que estaba creada de manera local, en el dispositivo, y esto no era el objetivo de la empresa, además sólo se introducían algunos datos, que no eran los que se necesitaban realmente.
El principal problema de la parte de los juegos aparte de tener comentarios en varios idiomas era que los nombres de las clases no eran nada intuitivos con lo que hacía la clase. Esto hizo que se perdiera bastante tiempo en entender lo que hacía cada clase. Tampoco era claro el código que había, ya que estaba desordenado y dispuesto de malas maneras.

La conexión con la base de datos fue bastante costosa de hacer, no hay apenas información de cómo realizar la conexión y se ha estado más de una semana intentando conseguir la conexión. Tras buscar información sobre este tipo de conexión durante días, se encontró la solución mediante la conexión vía Http, utilizando el método Post, el cual, con JSON, realiza la consulta con la base de datos.

3.4. Desarrollo y pruebas

3.4.1. Pantalla Inicial (PantallaBienvenida.java)

En la pantalla inicial (Figura 8) se realiza el acceso a la aplicación. Para acceder a la aplicación se debe insertar en el campo Usuario y Contraseña nuestros campos como usuario registrado. Cuando pulsemos el botón “Entrar” hará una conexión con la base de datos para comprobar si existe el usuario y si la contraseña es la correcta.

Figura 8: Menú Bienvenida

Como se puede apreciar en la Figura 9, mientras se comprueba que el usuario y contraseña son correctos aparece un diálogo en el que pone “Autentificando…”.

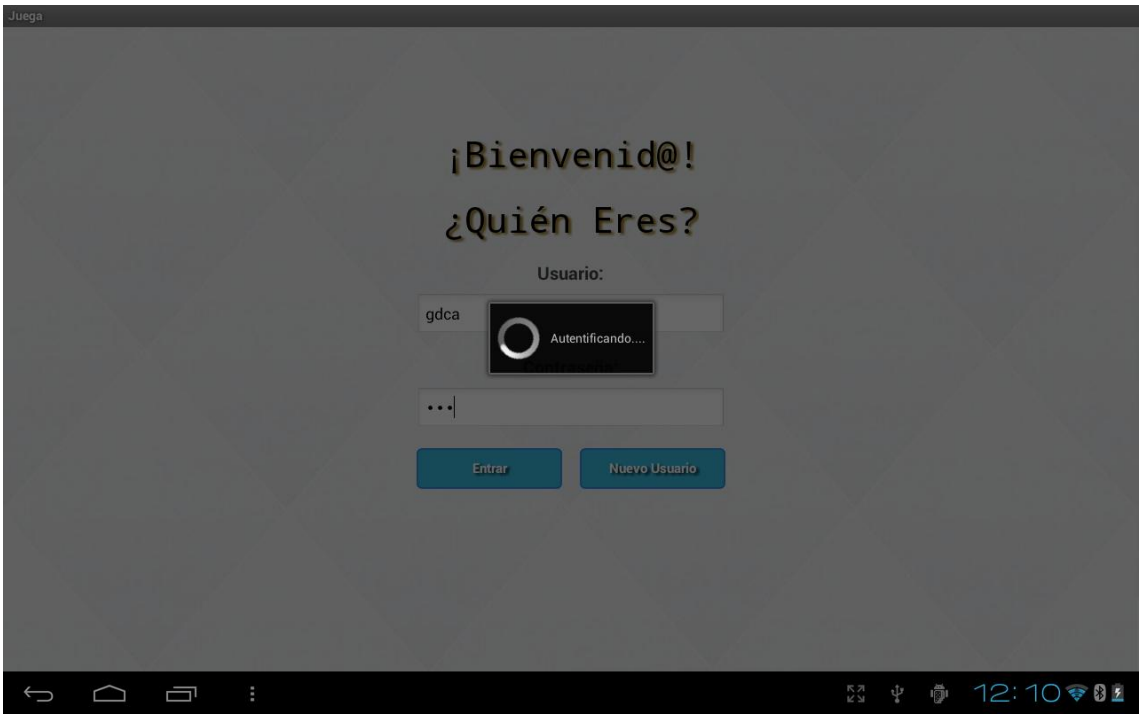


Figura 9: Autentificando

Pueden darse varios casos en esta primera pantalla:

- Si se han introducido los datos correctamente, se accederá a la pantalla principal Menu_juegoActivity.java (Figura 11).
- Si se ha introducido el usuario o contraseña de manera incorrecta volverá a la pantalla PantallaPrincipal.java mostrándose el siguiente mensaje:

  "Error: Nombre de usuario o contraseña incorrectos"

- Si por otro lado se deja uno de los dos campos en blanco, como en el caso anterior, volverá a la pantalla principal (PantallaPrincipal.java) y mostrará el mismo mensaje que en el caso en el que hayamos introducido el usuario o contraseña de manera incorrecta.

También existe la opción de crear un nuevo usuario. Cuando se pulse la opción de nuevo usuario accederemos a la pantalla nuevo usuario patallaNuevoUsuario.java.

Código

En cuanto a la parte gráfica (el XML), se trata de un LinearLayout que contiene el texto, los recuadros y los dos botones. Lo más destacable es que a todos los botones de la aplicación se les ha establecido un estilo, que está definido de la manea siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true">
        <shape>
        <gradient android:angle="270" android:startColor="@color/azulpulsar" android:endColor="@color/azulpulsar"></gradient>
            <stroke android:width="2dp" android:color="@color/azulborde"></stroke>
            <corners android:radius="2dp"></corners>
            <padding android:top="10dp" android:right="10dp" android:bottom="10dp" android:left="10dp"></padding>
        </shape>


    </item>

    <item>
        <shape>
            <gradient android:angle="270" android:startColor="@color/azulb" android:endColor="@color/azulb"></gradient>
            <stroke android:width="2dp" android:color="@color/azulborde"></stroke>
            <corners android:radius="5dp"></corners>
            <padding android:top="10dp" android:right="10dp" android:bottom="10dp" android:left="10dp"></padding>
            </shape>
    </item>
</selector>
```

En este estilo se definen las características de ambos botones distinguiendo si el botón está pulsado o sin pulsar.

El primer ítem es para cuando el botón ha sido seleccionado, por tanto hay que indicar:

android:state_pressed="*true*".

La única diferencia entre uno y otro después de haber definido cuál va a encargarse del botón pulsado son los colores, uno llama a un color y el otro a otra tonalidad de color. Esto se ha hecho para que se vea claro si el botón ha sido pulsado.

Lo más relevante en la parte java de esta pantalla es la conexión con la base de datos para la comprobación de los datos (explicación detallada en la parte de conexión con la base de datos) y las características que se les da a cada elemento del XML.

Una vez introducido el usuario y contraseña se ha incluido un diálogo que especifica que se está autentificando el usuario mientras que en segundo plano verifica si el usuario y contraseña son correctas.

```
        class asynclogin extends AsyncTask< String, String, String > {

         String user,pass;
            protected void onPreExecute() {
             //para el progress dialog
                pDialog = new ProgressDialog(PantallaBienvenida.this);
                pDialog.setMessage("Autentificando....");
                pDialog.setIndeterminate(false);
                pDialog.setCancelable(false);
                pDialog.show();
            }

    protected String doInBackground(String... params) {
    //obtnemos user y pass
    user=params[0];
    pass=params[1];

    //enviamos y recibimos y analizamos los datos en segundo plano.
         if (loginstatus(user,pass)==true){
         return "ok"; //login valido
         }else{
         return "err"; //login invalido
         }

    }

    /*Una vez terminado doInBackground segun lo que halla ocurrido
    pasamos a la sig. activity o mostramos error*/
            protected void onPostExecute(String result) {
               pDialog.dismiss();//ocultamos progess dialog.

         Log.e("onPostExecute=",""+result);

         if (result.equals("ok")){
            Intent b = new Intent(PantallaBienvenida.this,Menu_juegoActivity.class);
            startActivity(b);

          }else{
           err_login();
          }

              }

      }
}
```

3.4.2. Pantalla Inicial (PantallaBienvenida.java)

En esta pantalla se ha dado la posibilidad de crear un usuario nuevo para poder acceder a la aplicación. Todos los datos son recogidos y almacenados en la base de datos de manera automática en el momento en el que se pulse el botón de aceptar, si no se da ninguna situación de error.

Figura 10: Nuevo Usuario

Está pantalla tendrá control de situaciones de error como por ejemplo:

- Dejar algún campo vacio.
- Repetición de contraseña, que sea igual en ambos casos.
- Introducción incorrecta de la edad (inserción de caracteres, edades mayores a 120 años…)

Este control de errores no se ha realizado en la clase de nuevo usuario sino que se ha realizado al hacer la conexión con la base de datos, en el archivo php de creación de un nuevo usuario (adduser.php).

Una vez se han introducido de manera correcta los datos en la base de datos se accede al menú principal de la aplicación (Menu_juegoActivity.java).

La parte de código java es muy similar a la de Menu_juegoActivity.java, ya que la comprobación de errores, como se ha dicho anteriormente se realiza en el php.

3.4.3. Pantalla Principal (Menu_juegoActivity.java)

En esta pantalla es donde se encuentran todos los juegos a los que se puede acceder.
Se han creado 5 juegos diferentes:

- Simón: Refuerza la memoria visual y sonora.
- Memory: Refuerza la memoria visual.
- Torres de Hanoi: Refuerza la planificación y organización
- Stroop: Refuerza la atención focalizada y selectiva
- Parejas: Refuerza la memoria visual.

Cada juego ha sido representado por un botón diferente.

Cada botón tiene una imagen asociada al juego al que ha de representar. También existe la opción de salir de la aplicación, dándole al botón "Salir". Una vez pulsado el botón se redirige al principio, a la introducción del usuario y contraseña.

Figura 11: Menú Juegos

Código

En el apartado java se ha dado funcionalidad a los botones, según el juego que se elija se irá a un lugar o a otro.

```
@Override
public void onClick(View v) {
    // TODO Auto-generated method stub

    switch(v.getId()){
    case R.id.b_simon:
        Intent a = new Intent(this,juego.packages.SimonDice_principal.class);
        startActivity(a);
        break;
    case R.id.b_memory:
         Intent b = new Intent(this,juego.packages.Memory_MainActivity.class);
         startActivity(b);
        break;

    case R.id.b_hanoi:

     Intent c = new Intent(this, juego.packages.HanoigrossiActivity.class);
         startActivity(c);
    break;
            case R.id.b_stroop:
        Intent d = new Intent(this,juego.packages.StroopActivity.class);
        startActivity(d);
        break;
    case R.id.b_corsi:
        Intent h= new Intent(this,juego.packages.Corsi.class);
        startActivity(h);
        break;

    case R.id.b_math:
        Intent e = new Intent(this,juego.packages.Parejas2_prin.class);
        startActivity(e);
        break;
```

En la parte de xml se han dado forma y características a los botones.

3.4.4. Juego Memory

Este juego consiste en un mosaico de cartas que se sitúan boca abajo, y que se deben ir descubriendo por parejas para localizar las parejas de cartas iguales.

Las cartas desde el inicio se encuentran boca abajo sin que se pueda ver la imagen que se encuentra tras de ella, una vez se pulse una carta del tablero esta mostrará la imagen de un animal, en este momento se elegirá otra carta intentando que contenga el mismo animal de la carta que ha sido ya volteada.
En este juego se asegura que siempre van a poderse hacer todas las parejas.

Se ha creado un menú donde el usuario puede elegir entre jugar sólo (Un jugador) o con otra persona (Dos jugadores).

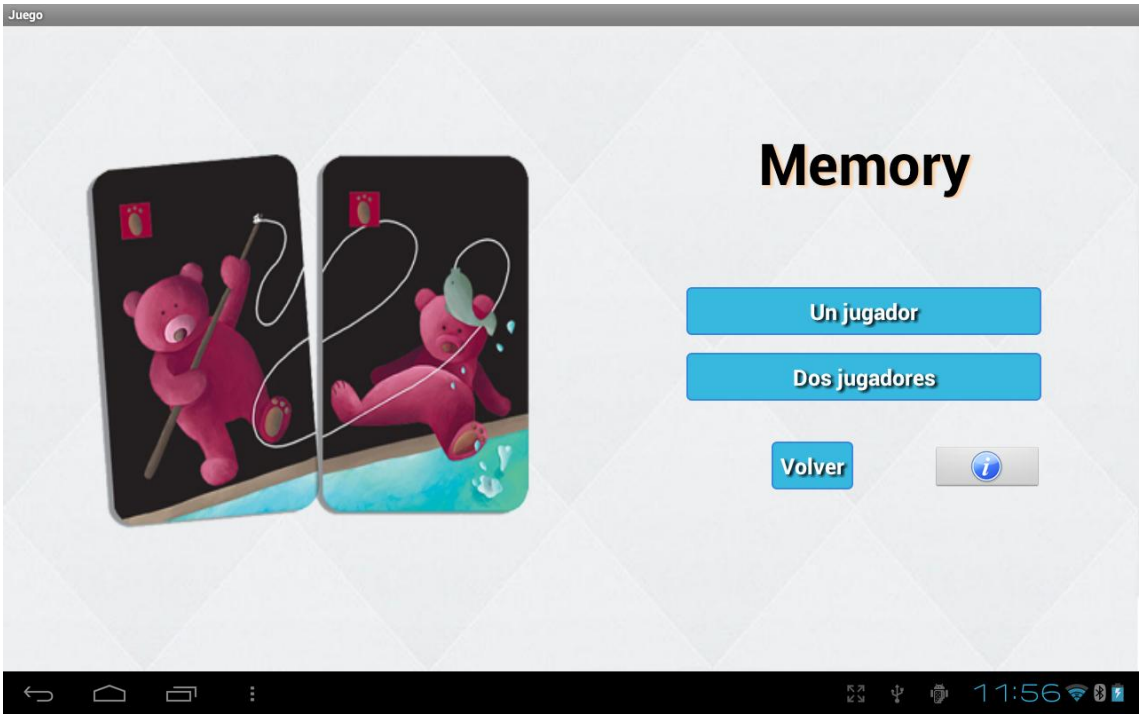


Figura 12: Menú Memory

El botón de Información muestra un bocadillo (ver Figura 13) donde se ofrecen las instrucciones del juego.

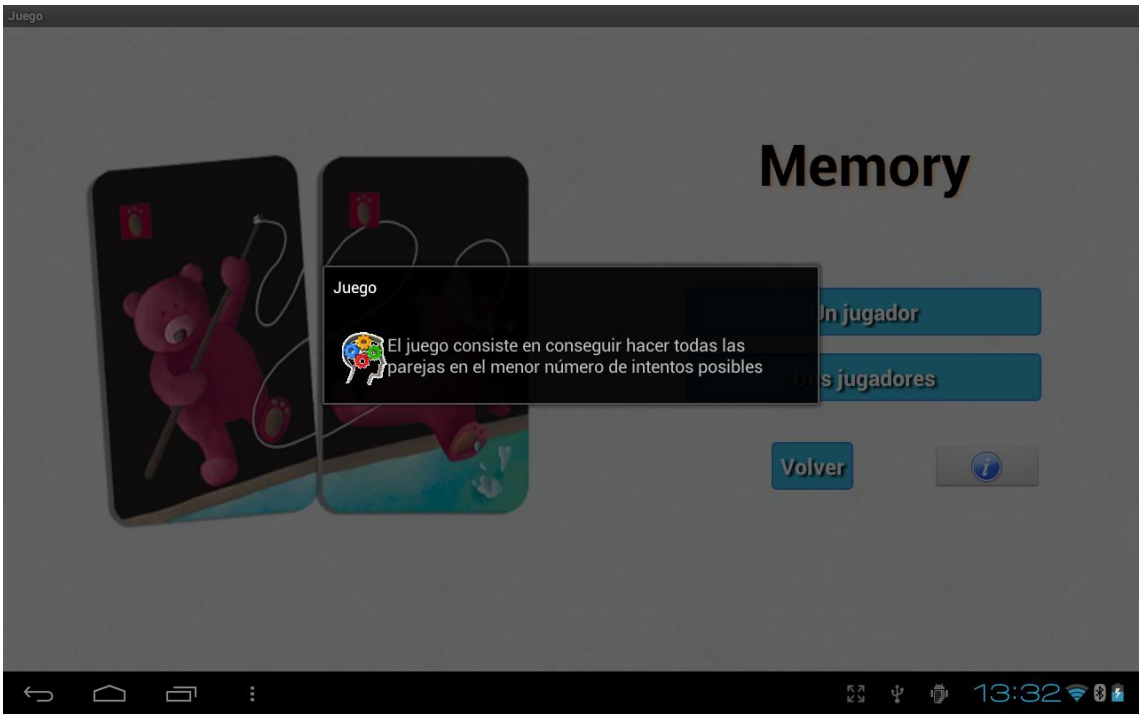


Figura 12: Memory información

Si se ha elegido la opción de dos jugadores, iremos directamente al juego y se podrá comenzar a emparejar las cartas (Figura 15).

En cambio, si se elige el modo “Un jugador” saldrán varias opciones según el número de cartas que se desee que aparezcan (ver Figura 14):

- 4x4
- 4x5
- 4x6
- 6x6

Figura 14: Menú 1 jugador

Una vez elegido el modo de juego se accederá al tablero con las cartas, este ejemplo es el de mayor dificultad en el modo de "Un jugador", 6x6.

1º Se tiene el tablero en su estado inicial, nada más entrar. El número de intentos es igual a 0.

Figura 15: Panel principal

2º Se levantan dos cartas. Pueden darse dos opciones que se ilustran en las figuras 16 y 17:

- Que las cartas sean diferentes y por tanto volverán a darse la vuelta y quedar con la imagen del interrogante.

Figura 16: Intento pareja

- Que las cartas sean iguales y por tanto se eliminen ambas cartas.

Figura 17: Eliminación pareja

En ambas opciones el marcador aumentará en una unidad cada vez que se haga un intento.

Código

Las pantallas donde se escoge el modo de juego son simples ya que es donde se han declarado los botones y se les ha dado funcionalidad.

Lo más destacable es el botón de información que da las instrucciones del juego, que ha sido creado con un imageButton.

```
    public void onClick(View v) {

        switch(v.getId()){
        case R.id.salir:
            Intent p = new Intent("juego.packages.Menu_juegoActivity");
            startActivity(p);
            break;
        case R.id.imageButtonIcon:
            Intent i = new Intent("juego.packages.Ficheroacercade");
            startActivity(i);
            break;
        }}
```

La funcionalidad del botón se ha creado llamando al fichero "Ficheroacercade" donde cargamos un layout con la apariencia que se pretende que tenga dicha información.

```
package juego.packages;

import android.app.Activity;


public class Ficheroacercade extends Activity {

    /** Called when the activity is first created. */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);


    }
}
```

El layout que cargamos es muy simple, se le ha dado el texto a mostrar y las características de este.

```
<?xml version="1.0" encoding="utf-8"?>
 <TextView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/TextView01"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="20dip"
  android:textSize="20dp"
  android:text="El juego consiste en conseguir hacer todas las parejas en el menor número de intentos posibles">
</TextView>
```

El archivo en el cual se desarrolla el juego "Memory_1jugador" y "Memory_2jugadores" tiene casi todo en común, las principales características en ambos son:

- Todas las pantallas tienen la opción de volver al menú de inicio del juego con la integración de un botón.
- En el caso de un jugador la elección del nivel se ha controlado con un switch donde se han mostrado las cuatro opciones de nivel para acceder a cada uno de ellos.

```
private void initializeBoard() {
    GridView gv = (GridView)findViewById(R.id.gridview2);
    switch(dificultad){
    case 1:
        gv.setNumColumns(4);
        break;
    case 2:
        gv.setNumColumns(5);
        break;
    case 3:
        gv.setNumColumns(6);
        break;
    case 4:
        gv.setNumColumns(8);
        break;
    }
```

- Se ha creado una clase “Memory_CardAdapter.java” en la cual se han declarado todas las imágenes. Estas imágenes se han introducido en un array de cartas, al cual se accede para obtener las cartas que se van a mostrar en el tablero.
  Para poder sacar las imágenes de manera aleatoria y así no coincidan unas partidas con otras se ha utilizado la función random que escoge aleatoriamente las imágenes. En el caso de “Un jugador”, utilizando un switch, se declaran los arrays concretos de cada nivel dependiendo el número de cartas de estos (Memory_CardAdapter2.java).
- Con la función getCard() se ha hecho un recorrido de las imágenes y se han devuelto para integrarlas en el tablero.
- En el caso de la cartas que tienen el símbolo del interrogante  , se ha declarado y asignado una imagen, la función getCardBack()es la que la devuelve. Con la función random se han podido escoger las imágenes de manera aleatoria.

```
private class cardController{
    private Integer[] cards;
    /**
     * @uml.property  name="cardBack"
     */
    private Integer cardBack = R.drawable.icon;
    int index;
    public cardController( ) {
        index = 0;
        cards  = new Integer[]  {
                 R.drawable.card1,  R.drawable.card2,   R.drawable.card3,   R.drawable.card10,
                 R.drawable.card13, R.drawable.card14,  R.drawable.card5,   R.drawable.card6,
                 R.drawable.card7,  R.drawable.card17,  R.drawable.card1,   R.drawable.card2,
                 R.drawable.card11, R.drawable.card3,   R.drawable.card13,  R.drawable.card14,
                 R.drawable.card10, R.drawable.card5,   R.drawable.card7,   R.drawable.card9,
                 R.drawable.card12, R.drawable.card6,   R.drawable.card4,   R.drawable.card8,
                 R.drawable.card17, R.drawable.card12,  R.drawable.card9,   R.drawable.card16,
                 R.drawable.card4,  R.drawable.card11,  R.drawable.card8,   R.drawable.card16,
        };
        Random rgen = new Random();
        for (int i=0; i<cards.length; i++) {
            int randomPosition = rgen.nextInt(cards.length);
            int temp = cards[i];
            cards[i] = cards[randomPosition];
            cards[randomPosition] = temp;
        }
    }
    public  Integer getCard() {
        if(index<cards.length){
            Integer result = cards[index];
            index++;
            return result;
        }
        else{
            Integer result = cards[0];
            index=1;
            return result;
        }
    }
    /**
     * @return
     * @uml.property  name="cardBack"
     */
    public Integer getCardBack(){
        return cardBack;
    }
    public int getnumCards(){
        return cards.length;
    }
}
```

La comprobación de las cartas se realiza con el método “NewMovementTask()” el cual comprueba todos los posibles resultados. Una vez encuentra un resultado se dirige a la función que corresponde para realizar lo oportuno.

```
public void newMovement(View v){
    actualizar();
    CheckTurntask task = new CheckTurntask();
    task.execute(v);
}

public Integer newMovementTask(View v){
    Integer result = -1;
    turn.selectedCards.add(v);
    if(selectedCards.size() == 2){
        if(checkSuccess()){
            if(checkEnd()){
                result = 2;
            }
            else{
                result = 1;
            }
        }
        else{
            result = 0;
        }

    }
    return result;

}
```

```
private boolean checkEnd() {
    GridView gridview = (GridView) findViewById(R.id.gridview2);
    return gridview.getCount() == (deletedCards+2);
}

private boolean checkSuccess() {
    return(!sameCard(selectedCards) && sameImage(selectedCards));

}
private void actualizar()
{
    String aux = String.valueOf(intentos);

    tv.setText(aux);
}

private boolean sameCard(ArrayList<View> selectedCards) {
    return selectedCards.get(0).equals(selectedCards.get(1));
}
```

Unos de los métodos más importantes son los siguientes:

- deleteCards: Elimina las cartas que son iguales, por tanto cuando la pareja es correcta.

```
private void deleteCards(ArrayList<View> selectedCards2) {
    ((Memory_CardView)selectedCards.get(0)).deleteCard();
    ((Memory_CardView)selectedCards.get(1)).deleteCard();
    deletedCards += 2;
    if(musicON)soundPool.play(idRemoveSound, 1, 1, 1, 0, 1);

}
```

- flipCards: Vuelve a dar la vuelta a las cartas, cuando la pareja no coincide.

```
private void flipCards() {
    ((Memory_CardView)selectedCards.get(0)).flipView();
    ((Memory_CardView)selectedCards.get(1)).flipView();
}
```

- sameImage: Realiza la comprobación de las cartas, si se tratan de cartas iguales o diferentes.

```
private boolean sameImage(ArrayList<View> selectedCards) {
    Integer id1 = ((Memory_CardView)selectedCards.get(0)).getImageID();
    Integer id2 = ((Memory_CardView)selectedCards.get(1)).getImageID();
    return id1.equals(id2);
}
```

Cuando se ha elegido la opción de dos jugadores, el turno de cada uno y la puntuación de ambos se controlan con las funciones TurnController y nextTurn.

```
public TurnController() {
    Random dice = new Random();
    pos = dice.nextInt(numPlayers-1);
    setCurrentPlayer(pos);
    selectedCards = new ArrayList<View>();
    deletedCards = 0;
}

public void nextTurn(){
    pos = (pos+1)%numPlayers;
    setCurrentPlayer(pos);
    selectedCards.clear();
    Toast t = Toast.makeText(getApplicationContext(),R.string.next_turn,Toast.LENGTH_SHORT);
    t.setGravity(Gravity.BOTTOM, 0, 0);
    t.show();
}

public void setCurrentPlayer(int pos)
{
    currentPlayer = players[pos];
    updatePlayerView(pos);
}
```

Por último se tiene que comprobar la puntuación, el número de intentos que ha realizado el usuario hasta finalizar el panel. Como se ha terminado la partida, el valor de la variable result será 2 y por tanto mostrará la pantalla de victoria al usuario y almacenará en la base de datos el número de intentos realizados.

La variable intentos se va a incrementar en todos los casos. Va a llamar a las funciones para introducir los datos a la base de datos según el caso. Cuando se han eliminado todas las cartas saltará un layout que va a informar de que se ha superado con éxito la prueba.

```
@Override
protected void onPostExecute(Integer result){
    if(result.equals(0)){
        synchronized (this) {
            try {
                wait(400);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            flipCards();
            intentos++;
            bbddmalno();
            nextTurn();
        }
    }
    if(result.equals(1)){
        deleteCards(selectedCards);
        intentos++;
        bbddbienno();
        actualizar();
        //updateScore();
        initializeMovements();
    }

    if(result.equals(2)){
        deleteCards(selectedCards);
        intentos++;
        bbddfinsi();
        actualizar();
        LayoutInflater inflater = getLayoutInflater();
        View layoutView = inflater.inflate(R.layout.memory_victoria, null);
        Toast to = new Toast(getApplicationContext());
        to.setDuration(Toast.LENGTH_SHORT);
        to.setView(layoutView);
        to.show();
    }
}
```

La inserción de los datos en la base de datos se explicará en el apartado sobre la inserción en base de datos.

Se ha realizado una prueba en este juego que nos ha hecho modificar algún aspecto. Se tenían una serie de cartas con fondos verdes que probando con un usuario sin ningún problema visual se distinguían de manera adecuada. Se hizo la prueba con dos personas que tienen daltonismo

y el resultado no ha sido igual de satisfactorio, por tanto se ha tomado la decisión de cambiar las cartas por animales que contentan fondo blanco.

En la Figura 18 se muestran el tipo de cartas que ocasionaron problemas.

Figura 18: Cartas con problemas

Se volvió a repetir la prueba con personas que veían sin ninguna dificultad y con personas con daltonismo y los resultados han resultado ser mucho mejores. Así que se han dejado las imágenes de animales con fondo blanco, las que aparecen en la Figura 16.

3.4.5. Juego Torres de Hanoi

Se tienen tres ejes o palos verticales. En el primero de ellos hay varios discos, dependiendo de la dificultad habrá más o menos, quedando los otros dos libres. Los discos están dispuestos de manera que el radio de estos esté en orden decreciente. El jugador debe mover todos los discos a otro de los ejes vacios, colocándolos en el mismo orden del que partían. Para ello puede ayudarse de los dos ejes restantes que se encuentran vacios.
Hay que intentar hacerlo de manera que realicemos el menor número de movimientos posibles.

La única norma que hay que cumplir es que nunca puede haber un disco más grande sobre otro más pequeño.

En este caso la pantalla principal de acceso al juego es diferente a las del resto de la aplicación. Hay un menú en forma de lista con todos los niveles posibles. Esto se ha puesto así debido a que hay muchos niveles.

Figura 19: Menú Torres de Hanoi

Un nivel puede ser superado siempre que se hayan conseguido pasar todos los discos de manera correcta. Si el nivel se ha hecho en el menor número de movimientos posibles, la estrella se pondrá de color naranja. En cambio, si se ha superado el nivel pero se ha utilizado un mayor número de movimientos, se habrá resuelto de manera correcta pero no se iluminará la estrella.

Una vez se ha elegido el nivel, se ha incorporado un efecto de rotación para acceder al juego.

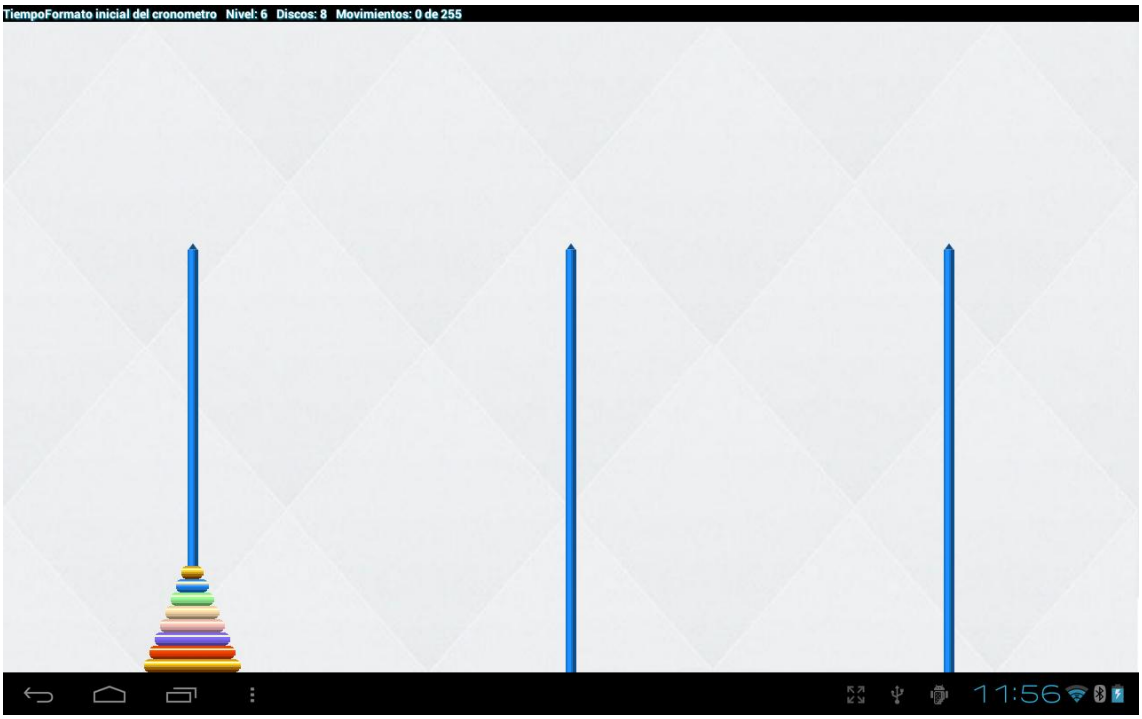


Figura 20: Estado Inicial

Se ha puesto un fondo que no interfiera demasiado con los colores de los discos, se ha podido aprovechar el fondo que da estética a toda la aplicación. Las barras de color azul se han pensado para que contrasten fuertemente con el fondo y se distingan con facilidad.

Una vez se ha elegido un nivel, en la parte superior de la pantalla se da información sobre el nivel que se ha elegido, el número de discos y el número de movimientos mínimos que se pueden realizar con su correspondiente contador de movimientos.

El contador se incrementará cada vez que realicemos un movimiento.

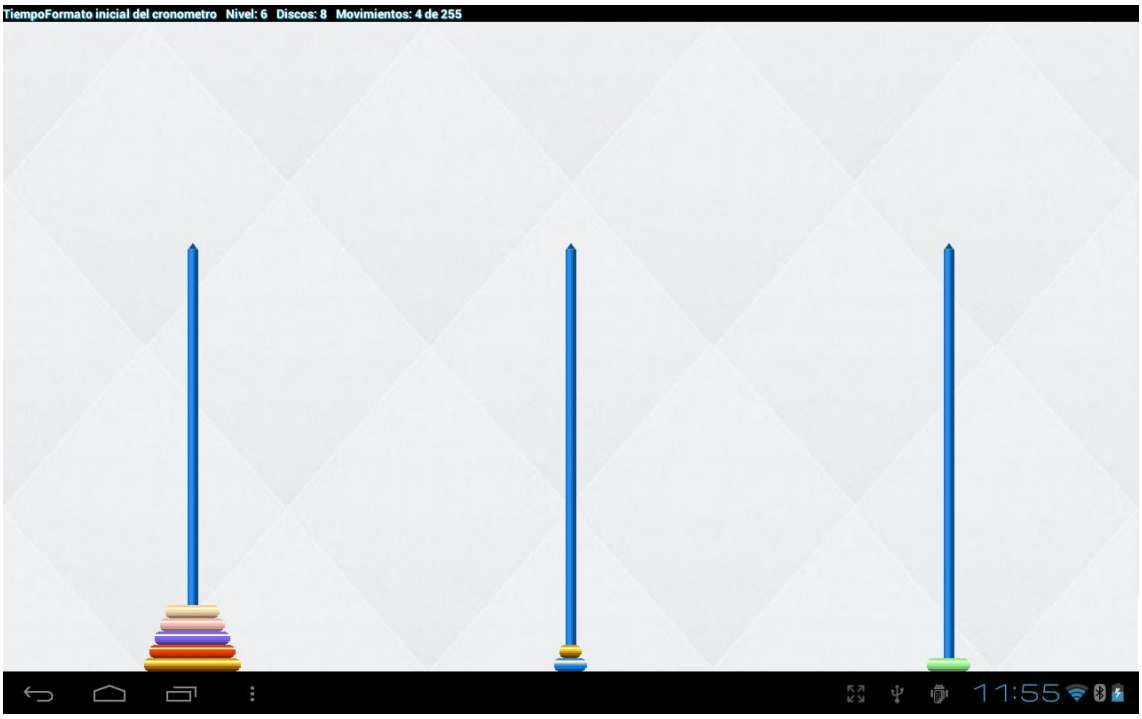


Figura 21: Movimientos Realizados

El movimiento de los discos es realmente bueno, tiene muy buena sensibilidad al movimiento.

No coge discos que no debe y sigue perfectamente la línea marcada con el dedo.

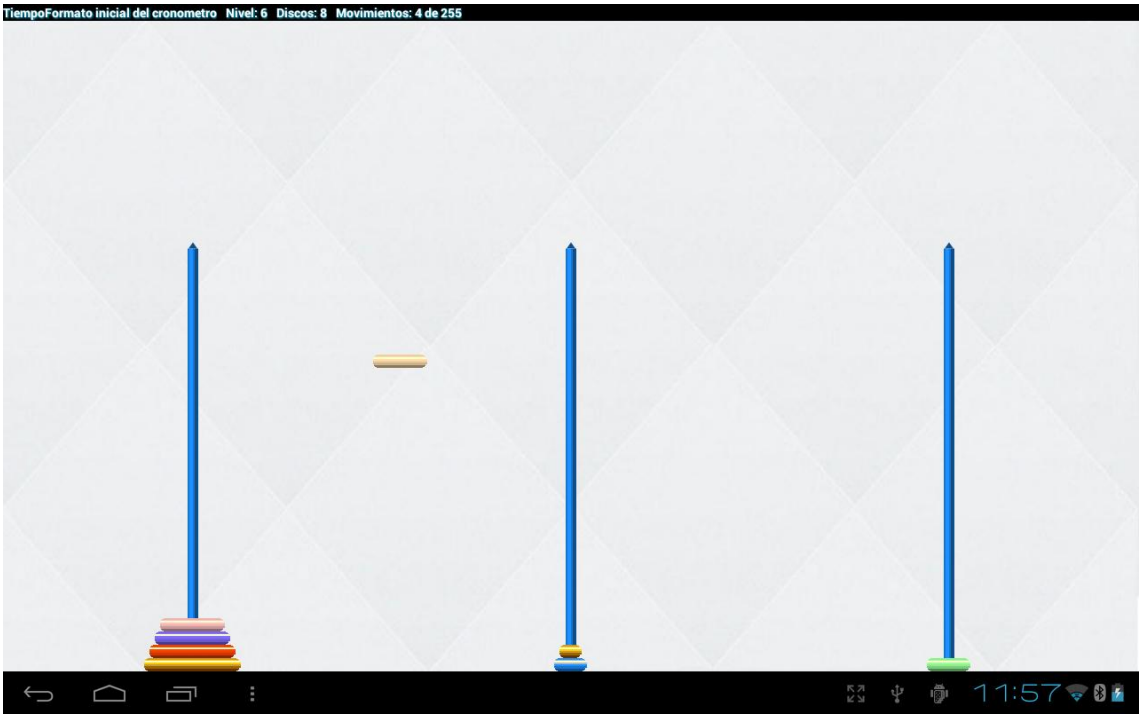


Figura 22: Movimiento disco

Código

La creación del menú de niveles se ha creado con ListView dentro del layout maintorres.xml.

```
<ListView android:id="@+id/fases"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"/>
```

El efecto de rotación se ha conseguido utilizando varias clases. Se muestra una pequeña parte del código donde se especifica el centro de la pantalla para realizar la rotación desde ese eje, la duración que va a tener y el tipo de efecto. La rotación se realiza dos veces, cuando seleccionamos el menú y cuando va a entrar al juego.

```
private void applyRotation(int position, float start, float end) {
    // Find the center of the container
    final float centerX = mContainer.getWidth() / 2.0f;
    final float centerY = mContainer.getHeight() / 2.0f;

    // Create a new 3D rotation with the supplied parameter
    // The animation listener is used to trigger the next animation
    final Rotate3dAnimation rotation =
        new Rotate3dAnimation(start, end, centerX, centerY, 310.0f, true);

    rotation.setDuration(500);
    rotation.setFillAfter(true);
    rotation.setInterpolator(new AccelerateInterpolator());
    rotation.setAnimationListener(new DisplayNextView(position));
    //iniciar animacao
    mContainer.startAnimation(rotation);
}
```

La función de rotación estaba ya creada cuando se asignó el proyecto, pero en un futuro se va a eliminar, por tanto no se le ha dado mucha importancia.

Una vez se ha seleccionado la fase que se quiere jugar, se cogen los datos del nivel para poder mostrarlos en la parte superior de la partida.

```
@Override public void actionNovaFase(IHanoigrossiBoard board, Fase fase) {
    //Iniciamos el conteo
    this.cromometro.stop();
    this.cromometro.setBase(SystemClock.elapsedRealtime());

    //configurar el nuevo estado dos objetos.
    this.hanoigrossiview.setBoard(board);
    //numero de discos
    this.textViewDiscos.setText(getString(R.string.discos) + ": " + fase.getDisco());
    //numero do nivel.
    this.textViewNivel.setText(getString(R.string.nivel)+": " + fase.getNivel());

    //numero de movimento.
    this.textViewMovi.setText(getString(R.string.movimentos) + ": " + hanoigrossi.getBoard().getMoveCount() +
            " " + getString(R.string.de) +  " " + hanoigrossi.getBoard().getMinMoves());

    //iniciar contagem.
    this.cromometro.start();
}
```

Cada vez que un disco es movido se incrementa el contador de movimientos.

```
@Override public void actionMoverDisco(boolean sucesso, int min, int move) {
    //numero de movimento.
    this.textViewMovi.setText(getString(R.string.movimentos) + ": " + move +
            " " + getString(R.string.de) +  " " + min);
}
```

Siempre que se acceda a un nuevo nivel o se acabe una partida, se llama a la función de acceso a la base de datos. La función recoge los datos y los envía.

En este caso hay varios métodos, ya que uno se encarga de meter los datos cuando hemos accedido al juego, otro cuando se ha terminado y el último para introducir todos los movimientos realizados por el usuario.

Se va a mostrar el caso en el que accedemos al juego, en el que se introduce fecha, hora y juego.

```
public void bbddmovini(){
    //Calculamos la hora
    Calendar calendario = new GregorianCalendar();
     hora =calendario.get(Calendar.HOUR_OF_DAY);
     minutos = calendario.get(Calendar.MINUTE);
     segundos = calendario.get(Calendar.SECOND);
     String salida = hora + ":" + minutos + ":" + segundos;
    //Calculamos la fecha actual con formato yyyy-MM-dd
     java.util.Date date = new java.util.Date();
     java.sql.Date sqlDate = new java.sql.Date(date.getTime());
     SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");
     String formatteDate = df.format(sqlDate); //La fecha esta en formatteDate
    ArrayList<NameValuePair> postparameters4send= new ArrayList<NameValuePair>();
    postparameters4send.add(new BasicNameValuePair("hora_comienzo",salida));
    postparameters4send.add(new BasicNameValuePair("juego","hanoi"));
    postparameters4send.add(new BasicNameValuePair("fecha_comienzo",formatteDate));
    JSONArray jdata=post.getserverdata(postparameters4send, URL_connect2);
}
```

Coge los datos de la hora de comienzo de la partida, el juego del que se trata y la fecha.

Los datos los manda mediante JSON a la base de datos. JSON es un formato ligero de intercambio de datos que además es independiente del lenguaje de programación. Está constituido por dos estructuras. Una colección de pares de nombre/valor, objeto, y una lista ordenada de valores, listas.

Explicación en detalle de la conexión en el apartado especifico a conexión con la base de datos de manera remota.

Se han declarado los componentes principales del juego como las tres barras y los discos. Se ha especificado el número mínimo y máximo de discos que puede haber.

```
public class Hanoigrossi implements IHanoiAcao {
    /** Primera barra **/
    public static final int PEG1 = 0;

    /** Segunda barra **/
    public static final int PEG2 = 1;

    /** Tercera barra **/
    public static final int PEG3 = 2;

    /** Numero minimo de discos **/
    public static final int MINDISCS = 3;

    /** Numero maximo de discos **/
    public static final int MAXDISCS = 12;
```

Se ha colocado el panel de forma que los discos estén en el primer palo.

```
   public Hanoigrossi(Context ctx, HanoiGrossiView view ) {
       super();

       this.ctx = ctx;
       this.view = view;
       this.board = new HanoigrossiBoard(MINDISCS, PEG1, MINDISCS);
       this.board.setHanoiAcao(this);
       this.controleFase = Utils.getInstaciaControleHanoi(ctx);
       this.audio = Utils.getInstaciaControleAudio(ctx);
   }
```

```
@Override public void actionMoverDisco(final boolean sucesso, final int min, final int move) {

    Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            if(sucesso) {
                audio.playMoverDisco();
            } else {
                audio.playMoverDiscoFalha();
            }

            int haste;
            if(sucesso && (haste = board.verificarPassoCorreto()) != -1) {
                audio.playBonus();
                Utils.msgVibracall(ctx, 100);
                view.anicacaoDiscoHaste(haste);
            }

            if(game != null) {
                game.actionMoverDisco(sucesso, min, move);
            }

            if(controleFase.getFaseAtual().isMovimentos()) {
                if(min < move) {
                    audio.playGameOver();
                    if(game != null) {
                        game.actionGameOver(min, move);
                    }
                }
            }

           if(board.isObjetivoConcluido()) {
               audio.playConclusaoFase();

               final Fase fase = controleFase.getFaseAtual();
               controleFase.proximaFase();

               if(game != null) {
                   game.actionObjetivoConcluido(min == move, fase);
               }
           }
       }
```

Dentro de la clase run se ha llamado a los sonidos, se ha comprobado si el paso es correcto y, de ser así, se ha dibujado la animación.

Se ha comprobado la fase que requiere y el número de movimientos necesarios, por tanto, se ha mirado si se han excedido los movimientos mínimos requeridos o no. Si se da el caso en el que se han realizado más pasos que movimientos mínimos se llamará a GameOver.

En cambio si se ha cumplido el objetivo, se desbloquea la próxima fase y se pasa la fase para poder marcar la estrella.

Todos los sonidos son declarados en AudioPlay.java. No se ha creado ninguna opción en la que se pueda configurar el sonido por si algún usuario tiene discapacidad auditiva. Esto es un punto que se va a tener en cuenta para las líneas futuras de la aplicación.

```
soundPool.unload(soundPoolMap.get(SOUND_FIBRA));
soundPool.unload(soundPoolMap.get(SOUND_CLICK));
soundPool.unload(soundPoolMap.get(SOUND_FLIP));
soundPool.unload(soundPoolMap.get(SOUND_VICTORY));
soundPool.unload(soundPoolMap.get(SOUND_BONUS));
soundPool.unload(soundPoolMap.get(SOUND_GAMEOVER));
```

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    int action = event.getAction();
    float x = event.getX();
    float y = event.getY();

    switch (action) {
    case (MotionEvent.ACTION_DOWN):
        sourcePeg = pixelToPeg(x, y);
    if (board.isStartPeg(sourcePeg)) {
        sourceDisc = board.getTopDisc(sourcePeg);
        if(sourceDisc.getNum() > 0) {
            board.discoSilencionado(sourceDisc);
            redesenharView(sourceDisc, x, y);
        }
    }
    break;

    case (MotionEvent.ACTION_UP):
        targetPeg = pixelToPeg(x, y);
    if (sourceDisc != null && board.moveDisc(sourceDisc, sourcePeg, targetPeg)) { }
    redesenharView(null, 0, 0);
    sourceDisc = null;
    break;

    case (MotionEvent.ACTION_MOVE):
        redesenharView(sourceDisc, x, y);
    break;
    }
    return true;
}
```

Se han evaluado los posibles eventos que pueden ocurrir con el manejo de la pantalla:

- Si se ha pulsado la pantalla, se ha utilizado ACTION_DOWN.
- Si se ha dejado de pulsar la pantalla, se ha utilizado ACTION_UP.
- Si se está desplazando el dedo sobre la pantalla, se ha utilizado ACTION_MOVE.

Cuando se ha pulsado la pantalla guardamos las coordenadas del lugar dónde se ha pulsado. Se hace lo mismo cuando ha dejado de pulsarse y se calcula el movimiento.

```
private int pixelToPeg(float x, float y) {
    double peg = (double) x / pegSpace;

    if (peg < 2) {
        return Hanoigrossi.PEG1;

    } else if (peg < 4) {
        return Hanoigrossi.PEG2;

    } else {
        return Hanoigrossi.PEG3;
    }

}
```

Se definen las áreas que dispone cada barra. Esto se ha hecho ya que no sólo está la opción de seleccionar el disco poniendo el dedo sobre él, sino que si se coloca el dedo en la zona definida de la barra cogerá de manera automática el disco.

Se ha utilizado la clase Canvas para implementar un comportamiento personalizado.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    pegSpace = getWidth() / 6;
    onDrawHaste(canvas, pegSpace);
    onDrawHaste(canvas, 3 * pegSpace);
    onDrawHaste(canvas, 5 * pegSpace);

    for (int p = 0; p <= 2; p++) {
        Disco[] disc = board.getDisco(p);

        for (int d = 0; d < disc.length; d++) {
            //recuperar disco.
            final Disco dc = disc[d];
            int width = dc.getDiscWidth();
            float xTem = (2 * p +1) * pegSpace - width/2;
            float yTem = getHeight()-(d + 1) * DISCHEIGHT;
            onDrawDisco(canvas, xTem, yTem, dc);
        }
    }

    if (dragged != null) {
        onDrawDisco(canvas, x, y, dragged);
    }
}
```

Primero se ha calculado la simetría entre las varillas. Luego se han establecido los centros de las tres varillas.

Una vez se ha seleccionado el disco, se han calculado las posiciones de “x” e “y” y se ha dibujado el disco. SI el disco arrastrado no es nulo se dibuja.

```
protected void onDrawHaste(Canvas canvas, float posX) {
    float x = posX;
    x -= 5;
    int h = getHeight();
    float ha = (float) (getHeight() - (getHeight() * 0.65));

    Paint paint = new Paint();
    paint.setStyle(Paint.Style.FILL);
    Cor.AZUL_MAIS_ESCURO.setCor(paint);
    paint.setAntiAlias(true);

    Path path = new Path();
    path.moveTo(0, -7);
    path.lineTo(5, 0);
    path.lineTo(-5, 0);
    path.close();
    path.offset(posX+1, ha);
    canvas.drawPath(path, paint);

    //Tonalidades de las barras
    Cor.AZUL_ESCURO.setCor(paint);
    canvas.drawRect(x,   ha, x+2, ha +(h-ha), paint);//2 pixel
    Cor.AZUL_CLARO.setCor(paint);
    canvas.drawRect(x+2, ha, x+6, ha + (h-ha), paint);//4 pixels
    Cor.AZUL_ESCURO.setCor(paint);
    canvas.drawRect(x+6, ha, x+8, ha + (h-ha), paint);//2 pixels
    Cor.AZUL_MAIS_ESCURO.setCor(paint);
    canvas.drawRect(x+8, ha, x+12, ha + (h-ha), paint);//4 pixels
}
```

La función onDrawHaste() se ha creado para dar forma a las barras verticales. Se ha calculado la altura del panel y las barras se han puesto al 65% para que no queden excesivamente altas.

Las barras se han pintado de varias tonalidades de azul para que aporte una sensación de barras con forma redondeada.

Los discos se han definido de manera similar, también se les han dado varias tonalidades para que dé sensación de redondez, incluso se les ha introducido un tono muy claro con el objetivo de que dé sensación de brillo.

```
private float fixX(float x, int width) {
    if (x < 0) x = 0;
    if (x > getWidth() - width) x = getWidth() - width;
    return x;
}

private float fixY(float y, int height) {
    if (y < height) y = 0;
    if (y > getHeight() - height) y = getHeight() - height;
    return y;
}
```

Se han calculado las coordenadas del disco arrastrado.

Cuando se realiza una torre correctamente, aunque no hayamos utilizado todos los discos, esta se pone de color naranja.

En la clase HanoigrossiBoard.java tenemos especificaciones de los discos y movimientos que se pueden realizar.

```
private Stack<Disco>[] hastes;
/** Numero de hastes(baras) **/
private static final int PEGS = 3;
/** Tamaño de los discos **/
static final int DISC_SIZES[][] = {
    {68,18}, {76,16}, {84,14}, {92,13}, {100,12},
    {108,12}, {112,11}, {116,10}, {120,9}, {124,9}
};
```

Se le da tamaño a los discos que van a tener.

Vista principal del juego una vez entramos.

```
public HanoigrossiBoard(int discs, int peg1, int min) {
    super();

    hastes = new Stack[PEGS];
    for (int i = 0; i < PEGS; i++) {
        hastes[i] = new Stack<Disco>();
    }

    discos = new Disco[discs];
    for (int i = discs-1; i >= 0; i--) {
        Disco disco = new Disco(i+1);
        disco.setDiscWidth(DISC_SIZES[discs - min][0] -(DISC_SIZES[discs - min][1] * (discs-1-i)));

        discos[i] = disco;
    }

    for (int i = discs-1; i >= 0; i--) {
        hastes[peg1].push(discos[i]);
    }

    moveCount = 0;
    minMoves = Utils.moviMinimo(discs);
}
```

Se han creado las barras y los discos. Se han colocado todos los discos en la primera barra y se han calculado los movimientos mínimos.

```
@Override
public boolean moveDisc(Disco disco, int hasteOrigem, int hasteDestino) {
    boolean movi  = false;

    if (hasteOrigem >= 0 && hasteDestino >= 0 && disco != null) {
        if (hasteOrigem != hasteDestino &&
                (isHasteVazia(hasteDestino) || hastes[hasteDestino].peek().getNum() > disco.getNum())) {
            hastes[hasteDestino].push(disco);
            moveCount++;
            movi = true;

        } else {
            hastes[hasteOrigem].push(disco);
        }
    }

    //Existe alguna accion
    if(acao != null) {
        //notifique
        acao.actionMoverDisco(movi, minMoves, moveCount);
    }

    return movi;
}
```

Se ha analizado el movimiento del disco. Siempre que el movimiento no es nulo se ha comprobado si se puede colocar el disco en el destino, que se trata de un disco más pequeño que el que hay en la barra de destino. Si es un movimiento correcto se aumentará en contador y se señalará como movimiento igual a true.

En cambio, si el disco está mal colocado, volverá a la posición anterior.

```
public boolean isbjetivoConcluido() {
    if((hastes[PEGS-1].empty() || hastes[PEGS-2].empty()) && hastes[PEGS-3].empty()) {
        return true;
    }

    return false;
}
```

Para la comprobación de que el juego ha terminado de manera correcta se ha comprobado que la primera barra se encuentra vacía, y que una de las otras dos barras también lo esté. No se ha comprobado ninguna situación en el que la colocación de los discos esté mal ya que ya se ha realizado anteriormente. Siempre que se quiere poner un disco mayor encima de otro menor, vuelve a la posición en la que se encontraba.

```
public int verificarPassoCorreto() {
    if((hastes[PEGS-1].empty() || hastes[PEGS-2].empty()) && !hastes[PEGS-3].empty()) {
        if(!hastes[PEGS-1].empty() && hastes[PEGS-1].size() > 2) {
            if(isDiscoSequencia(hastes[PEGS-1])) {
                return PEGS-1;
            }

        } else if(!hastes[PEGS-2].empty() && hastes[PEGS-2].size() > 2) {
            if(isDiscoSequencia(hastes[PEGS-2])) {
                return PEGS-2;
            }

        }
    }

    return -1;
}
```

Se comprueba el estado en el que pueden estar las barras.

Se ha creado una función para seleccionar el disco.

```
@Override
public void selecionarDisco(int haste, boolean select) {
    Disco[] discos = getDisco(haste);

    int size = discos.length;
    for (int i = 0; i < size; i++) {
        discos[i].setSelecionado(select);
    }
}
```

3.4.6. Juego Parejas

El juego consiste en realizar todas las parejas que se puedan en el menor número de movimientos posible. A diferencia del juego memory aquí no hay parejas siempre, hay cartas que no tendrán parejas. Esto hace que el usuario que utiliza el juego tenga que estar más atento ya que puede que marque una carta que no tenga la pareja y aumente el número de intentos innecesariamente.

Como en los demás casos, se ha creado un menú principal para poder acceder al juego.

Figura 23: Menú Parejas

Las cartas siempre se muestran, nunca son tapadas. Se han utilizado los mismos animales que el juego Memory. También incorpora un botón que permite la vuelta al menú principal del juego.

Se ha incorporado un contador con el número de intentos que se realizan. También se lleva la cuenta del número de parejas creadas correctamente.

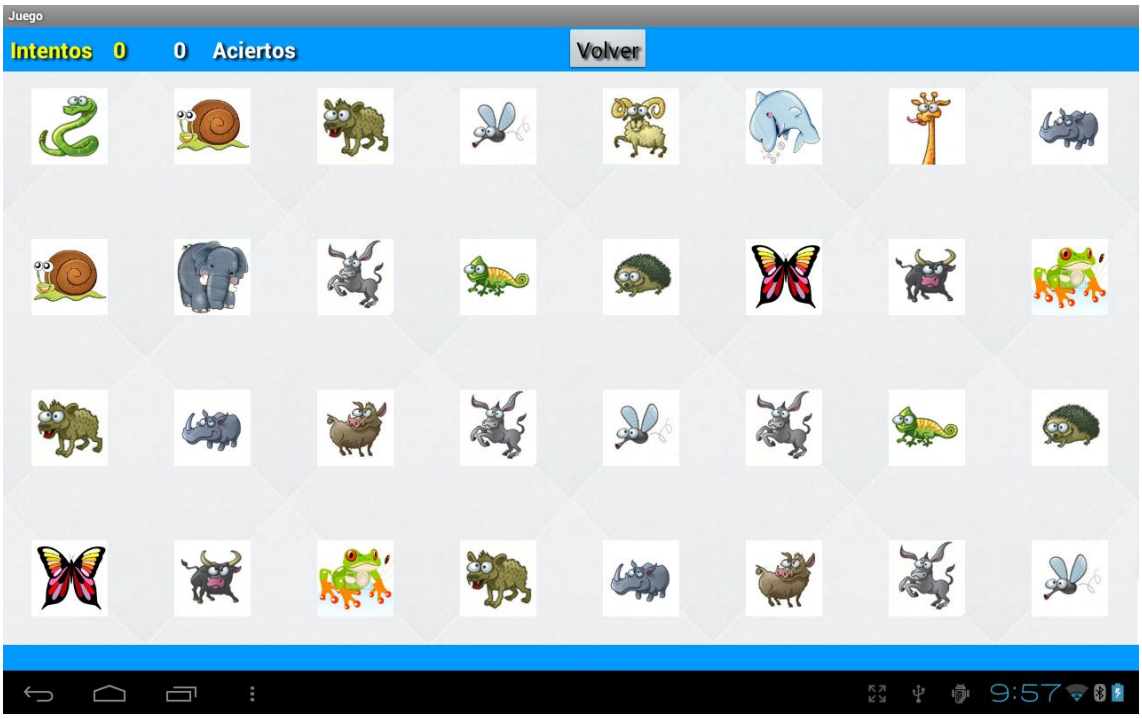


Figura 24: Panel inicial

El menú de información.

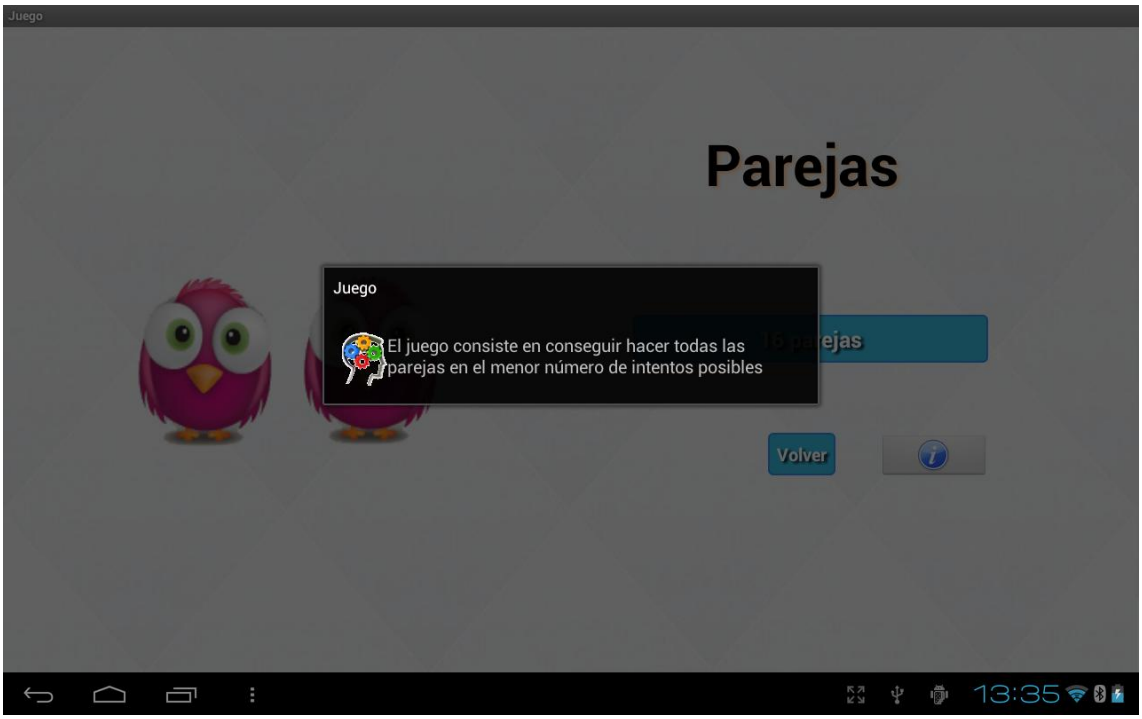


Figura 25: Instrucciones

Código

El código es muy similar al juego memory. El primer cambio es que no se ha necesitado que haya una carta tapando el animal, así que la imagen de interrogante no se ha mostrado y la

función que hace el cambio ha sido eliminada, por tanto ahora se ven todas las imágenes de los animales.

El juego, de manera interna, se divide en 4 niveles. El primer nivel es el tablero inicial. Todas las cartas se encuentran a la vista en ese momento. A la hora de hacer la primera pareja de manera correcta se cargará el segundo nivel, este segundo nivel está compuesto por las mismas cartas que el primer nivel. Este segundo nivel solamente se cargará en las cartas que se han eliminado, rellenado el hueco que dejarían. El tercer nivel es igual que el segundo pero en este caso se activará cuando hayamos hecho una pareja con alguna carta que ya estaba en el segundo nivel. El cuarto nivel hace simplemente que las cartas desaparezcan.

```
private class cardController{
    private Integer[] cards;
    private Integer[] cards2;
    private Integer[] cards3;
    /**
     * @uml.property
     */
    int index;
    public cardController( ) {
        index = 0;
        cards  = new Integer[]  {
                 R.drawable.card1,  R.drawable.card2,   R.drawable.card3,   R.drawable.card10,
                 R.drawable.card13, R.drawable.card22,  R.drawable.card21,  R.drawable.card19,
                 R.drawable.card7,  R.drawable.card17,  R.drawable.card1,   R.drawable.card2,
                 R.drawable.card11, R.drawable.card3,   R.drawable.card13,  R.drawable.card14,
                 R.drawable.card10, R.drawable.card18,  R.drawable.card7,   R.drawable.card9,
                 R.drawable.card12, R.drawable.card6,   R.drawable.card4,   R.drawable.card8,
                 R.drawable.card17, R.drawable.card12,  R.drawable.card9,   R.drawable.card16,
                 R.drawable.card4,  R.drawable.card11,  R.drawable.card20,  R.drawable.card16,
        };
        cards2 = new Integer[]  {
             R.drawable.card1,  R.drawable.card2,   R.drawable.card3,   R.drawable.card10,
             R.drawable.card13, R.drawable.card22,  R.drawable.card21,  R.drawable.card19,
             R.drawable.card7,  R.drawable.card17,  R.drawable.card11,  R.drawable.card13,
             R.drawable.card14, R.drawable.card18,  R.drawable.card9,   R.drawable.card12,
             R.drawable.card6,  R.drawable.card4,   R.drawable.card8,   R.drawable.card5,
             R.drawable.card20, R.drawable.card16,
       };
        cards3 = new Integer[]  {
                R.drawable.card1,   R.drawable.card2,   R.drawable.card3,   R.drawable.card10,
                 R.drawable.card13, R.drawable.card22,  R.drawable.card21,  R.drawable.card19,
                 R.drawable.card7,  R.drawable.card17,  R.drawable.card11,  R.drawable.card13,
                 R.drawable.card14, R.drawable.card18,  R.drawable.card9,   R.drawable.card12,
                 R.drawable.card6,  R.drawable.card4,   R.drawable.card8,   R.drawable.card5,
                 R.drawable.card20, R.drawable.card16,
          };
        Random rgen = new Random();
        for (int i=0; i<cards.length; i++) {
            int randomPosition = rgen.nextInt(cards.length);
            int temp = cards[i];
            cards[i] = cards[randomPosition];
            cards[randomPosition] = temp;
        }
    }
```

Se han declarado los arrays que van a contener las cartas. Se ha creado una función Random donde va a coger las cartas de manera aleatoria.

```
public  Integer getCard() {
    if(index<cards.length){
        Integer result = cards[index];
        index++;
        return result;
    }
    else{
        Integer result = cards[0];
        index=1;
        return result;
    }
}

public  Integer getCard2() {
    Random rgen = new Random();
    for (int i=0; i<cards2.length; i++) {
        int randomPosition = rgen.nextInt(cards2.length);
        int temp = cards2[i];
        cards2[i] = cards[randomPosition];
        cards2[randomPosition] = temp;
    }
    if(index<cards2.length){
        Integer result = cards2[index];
        index++;
        return result;
    }
    else{
        Integer result = cards2[0];
        index=1;
        return result;
    }
}
```

Con los métodos getCard(), getCard2() y getCard3() se ha obtenido la imagen a mostrar y se devuelve. Se accede a la tabla de imágenes y se elige una de manera aleatoria. La función getCard3() es idéntica a getCard2() pero en vez de acceder a la tabla de imágenes cards2 se accede a cards3.

Se ha modificado la clase SameImage del Memory para que pueda hacer la comparación entre los diferentes niveles de manera correcta.

```
private boolean sameImage(ArrayList<View> selectedCards) {

    Integer id1 = ((Parejas2_CardView)selectedCards.get(0)).getImageID();
    Integer id2 = ((Parejas2_CardView)selectedCards.get(1)).getImageID();
    Integer id3 = ((Parejas2_CardView)selectedCards.get(0)).getImageID2();
    Integer id4 = ((Parejas2_CardView)selectedCards.get(1)).getImageID2();
    Integer id5 = ((Parejas2_CardView)selectedCards.get(0)).getImageID3();
    Integer id6 = ((Parejas2_CardView)selectedCards.get(1)).getImageID3();
    if (id1.equals(id2)){
        return true;
    }
    else if (id1.equals(id3)){
        return true;
    }
    else if (id1.equals(id4)){
        return true;
    }
    else if (id1.equals(id5)){
        return true;
    }
    else if (id1.equals(id6)){
        return true;
    }
    else if (id2.equals(id3)){
        return true;
    }
    else if (id2.equals(id4)){
        return true;
    }
    else if (id2.equals(id5)){
        return true;
    }
    else if (id2.equals(id6)){
        return true;
    }
    else if (id3.equals(id4)){
        return true;
    }
    else if (id3.equals(id5)){
        return true;
    }
    else if (id3.equals(id6)){
        return true;
    }
    else if (id4.equals(id5)){
        return true;
    }
    else if (id4.equals(id6)){
        return true;
    }
    else if (id5.equals(id6)){
        return true;
    }
    else {return false;}
}
```

Se han declarado los diferentes niveles y se ha hecho la comparación de cada uno de ellos para ver si las cartas coinciden o no coinciden. Si la carta es la misma, se ha devuelto true y si la carta es diferente, se ha tratado con el else y se ha devuelto false.

```
@Override
public void onPostExecute(Integer result){
    //Si son cartas diferentes, que no haga nada
    if(result.equals(0)){
         intentos++;
         actualizar();
         synchronized (this) {
             try {
                    wait(100);
             } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
             }
             //flipCards();
             bbddmalno();
         }

                        }
    //Si son dos cartas iguales, las eliminamos
    if(result.equals(1)){
        deleteCards(selectedCards);
        intentos++;
        bbddbienno();
        actualizar();
        contok++;
        actualizar2();
        //updateScore();
        initializeMovements();
    }

   //Ultimas dos cartas
   if(result.equals(2)){
       deleteCards(selectedCards);
       intentos++;
       bbddfinsi();
       actualizar();
       contok++;
       actualizar2();
       //updateScore();
       LayoutInflater inflater = getLayoutInflater();
       View layoutView = inflater.inflate(R.layout.memory_victoria, null);
       Toast to = new Toast(getApplicationContext());
       to.setDuration(Toast.LENGTH_SHORT);
       to.setView(layoutView);
       to.show();
       //end();
   }
}
```

Cuando se hacen dos parejas erróneamente se deja igual y solamente incrementa el contador de intentos.

Si se ha realizado una pareja de manera correcta se eliminan las cartas y se cargan otras de manera aleatoria. Se incrementan tanto el contador de intentos como el de aciertos. En el caso de que se haya hecho la última pareja o se haya llegado a hacer el número de parejas previsto se carga una pantalla que nos informa de la que hemos ganado.

Todos los intentos son guardados en la base de datos, tanto si se ha equivocado como si no.

```
Httppostaux post;
String IP_Server="83.173.132.215:8081";
String URL_connect="http://"+IP_Server+"/addpartidas.php";
String URL_connect2="http://"+IP_Server+"/addpartidasinicio.php";
```

Se han declarado los datos necesarios para la conexión. Vamos a acceder a dos php diferentes, el que añade las partidas iniciales y las que introducen los datos de la partida, por tanto hemos declarado ambas conexiones.

```
Calendar calendario = new GregorianCalendar();
 hora =calendario.get(Calendar.HOUR_OF_DAY);
 minutos = calendario.get(Calendar.MINUTE);
 segundos = calendario.get(Calendar.SECOND);
 String salida = hora + ":" + minutos + ":" + segundos;
//Calculamos la fecha actual con formato yyyy-MM-dd
 java.util.Date date = new java.util.Date();
 java.sql.Date sqlDate = new java.sql.Date(date.getTime());
 SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");
 String formatteDate = df.format(sqlDate); //La fecha esta en formatteDate
ArrayList<NameValuePair> postparameters4send= new ArrayList<NameValuePair>();
postparameters4send.add(new BasicNameValuePair("hora_comienzo",salida));
postparameters4send.add(new BasicNameValuePair("juego","Parejas"));
postparameters4send.add(new BasicNameValuePair("fecha_comienzo",formatteDate));
JSONArray jdata=post.getserverdata(postparameters4send, URL_connect2);
```

Se ha calculado la fecha y la hora actuales ya que son datos esenciales para que los datos tengan una cronología en el tiempo para los pedagogos. Se han agregado los datos que queremos introducir en la base de datos a la lista. Posteriormente se realiza el intercambio de datos.

Las llamadas a los métodos bbddfinsi(), bbddsi() y bbddno() son para incluir los datos en la base de datos.

```
public void actualizarView(){
    if(status == 1) {this.setImageResource(imageID);
    status++;}
    else if(status == 0) this.setImageResource(imageID);
    else if(status == 2) this.setImageResource(imageID2);
    //Aqui creo el numero de niveles de cartas
    else { c++;
        if(c==1)this.setImageResource(imageID3);
        else if(c==2)this.setImageResource(imageID2);
        else if(c==3)this.setVisibility(INVISIBLE);
        else {}}
}
```

La comparación de los niveles se ha realizado aquí. Cuando se llega a que “c” es 3, querrá decir que ya se han superado los 3 niveles de cartas por tanto se ha puesto la carta invisible.

3.4.7. Juego Stroop

El juego se basa en una interferencia semántica en el tiempo de reacción de una tarea, cuando una palabra como "azul", "verde" ó "rojo" está escrita con una tinta de un color diferente del color expresado por su significado semántico, se produce un retraso en el procesamiento del color de la palabra, lo que aumenta el tiempo de reacción y favorece los errores.

El juego tiene tres opciones:

- 4 Colores: Se tendrán 4 colores que rotan aleatoriamente.
- 6 Colores: Se tendrán 6 colores que rotan aleatoriamente.
- Tiempo: Hay que acertar el mayor número de colores en un minuto, se muestra el tiempo restante a cada momento.

Se ha creado una pantalla principal en la cual se elige el modo de juego que queremos.

Figura 26: Menú Stroop

Se dispone también como en el resto de los juegos de un botón informativo donde se explican las instrucciones del juego.

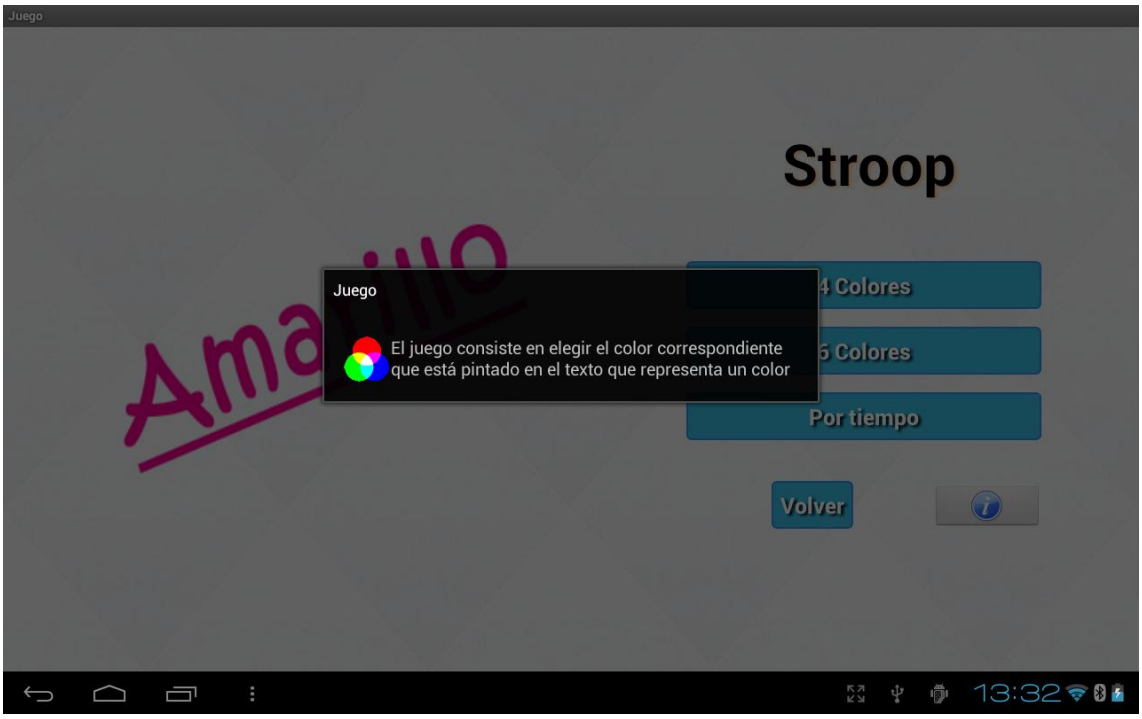


Figura 27: Instrucciones

Una vez elegido el modo de juego se accede a jugar.

- 4 Colores

Figura 28: 4 Colores

- 6 Colores

Figura 29: 6 Colores

- Tiempo

Figura 30: Tiempo

Donde en la parte superior izquierda de la pantalla se encuentra el cronómetro con el tiempo restante.

En los tres modos se van a obtener los datos de número de aciertos, número de fallos y número de jugadas realizadas en total.

Código

La pantalla principal del juego (StroopActivity.java) es muy sencilla, lo único que hace es cargar el layout creado y dar funcionalidad a los botones.

```
public class StroopActivity extends Activity  {


public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.strooptipojuego);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        final Button btncuatro = (Button)findViewById(R.id.btncuatro);
        final Button btnseis = (Button)findViewById(R.id.btnseis);
        final Button btntiempo = (Button)findViewById(R.id.btnTiempo);
        final Button btnsalir = (Button)findViewById(R.id.salir);
        final Button btninfo = (Button)findViewById(R.id.imageButtonIcon);

        btncuatro.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
            {
                Intent i = new Intent(getBaseContext(), StroopPlay4.class);
                 startActivity(i);

                }
        });
```

Las clases StroopPlay4.java, StroopPlay6 y StroopTiempo.java son muy similares, lo único que los diferencia es el número de colores que se han cogido aleatoriamente.

Se va a mostrar con el caso StroopPlay4.java:

Se han creado 4 funciones para la implementación del juego.

```
ramdom();
getReferenceOfViews();
contador();
tiempo();
```

Random se encarga de obtener el color que va a mostrar de manera aleatoria.

```
public void ramdom()
{
    int x = new Random().nextInt(4);
    int y = new Random().nextInt(4);


//azul

    if (x == 0)
    {
        final TextView txtcolor = (TextView)findViewById(R.id.txtcolor);
        String texto = txtcolor.getText().toString();
        texto = "azul";
        txtcolor.setText(texto); //palabra que va mostrar
        // palabra en color rojo
        if (y == 0)
        {
            txtcolor.setTextColor(Color.RED);
        }

        else if (y == 1)
        {
            ramdom();
        }
        //palabra en color verde
        else if (y == 2)
        {
            txtcolor.setTextColor(Color.GREEN);
        }
        //palabra en color amarillo
        else if(y == 3)
        {

            txtcolor.setTextColor(Color.parseColor("#FFFF00"));
        }

    }
```

Donde “x” es un random de las palabras e “y” es un random de los colores de las palabras.

La variable “x” puede tomar valores del 0 al 3, donde el 0 es el color azul, el 1 es el rojo, el 2 el amarillo y el 3 es el color verde.

La variable “y” toma los mismo valores que “x” con la diferencia de que se le asignan los colores de diferente manera. También hay que tener en cuenta que si ha salido la palabra del mismo color al que representa se tendrá que volver a realizar la llamada a la función random.

Se trata del mismo proceso en los demás colores.

El método getReferenceOfViews es la encargada de mostrar el tiempo.

```
private void getReferenceOfViews()
{
    textViewShowTime = (TextView) findViewById(R.id.txttiempo);

}
```

La función contador es la encargada de llevar la cuenta y mostrar la puntuación que se va creando conforme discurre la partida.

```
public void contador()
{
final TextView txtContador = (TextView)findViewById(R.id.txtContador);
String texto = txtContador.getText().toString();
texto = " "+"Aciertos" +" "+ aciertos +". "+ "Fallos" +" "+ fallos +". "+"Número de jugadas" +" "+ nu
txtContador.setText(texto);
}
/*
* contador 1: cuenta los aciertos de esa partida
*/
public void contador1() //contador de aciertos
{
aciertos = aciertos +1;

}
/*
* contador 2: cuenta el numero de fallos en la partida
*/
public void contador2() //contador de fallos
{
fallos = fallos +1;

}
/*
* jugadas en el numero de jugadas realizas en la partida.
*/
public void jugadas()
{
numJugadas = aciertos + fallos;
}
```

Cuando ocurre un evento de acierto o fallo, se incrementa en una unidad el contador oportuno. Respecto al número de jugadas, siempre se trata de la suma de aciertos y fallos.

La función de tiempo se encarga de llevar la cuenta del tiempo en que se desarrolla el juego.

```
public void tiempo()
{

totalTimeCountInMilliseconds = 0;
totalTimeCountInMilliseconds = 5 * 1000 ;


        countDownTimer = new CountDownTimer (totalTimeCountInMilliseconds, 500)
        {
            @ Override
       public void onTick ( long leftTimeInMilliseconds)
      {

            long seconds = leftTimeInMilliseconds / 1000 ;
            textViewShowTime.setText (String.format ("% 02d" , seconds / 60) + ":" + String.format ("% 02d" ,  seconds % 60 ));

      }
            public void onFinish ()
            {
                    contador2();
                    bbddfinsi();
                        jugadas();
                        contador();
                        alerta();
                        ramdom();
                        tiempo();
                }

        }.start();
    }
```

Se ha creado un formato del TextView para mostrar el formato de manera que sea de fácil lectura.

Se llama al método onFinish cuando la cuenta atrás llega a cero, para que pare la aplicación.

Para realizar la comprobación de si el botón pulsado ha sido el correcto o no, se hace una comparación del botón con el color del texto.

Si es correcto, aumenta el contador de aciertos, llama a la base de datos para introducir los datos, etc. En el caso de no ser correcto llama al contador de fallos.

```
btnverde.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0){
        //si los colores son iguales realizara la siguentse acciones
        if (btnverde.getCurrentTextColor()== txtcolor.getCurrentTextColor())
        {       cancel();
                contador1(); //Aciertos
                bbddbienno();
                jugadas();
                contador();

                alerta();
                ramdom();
                tiempo();
        }
        // si los colores son distintos realizara la siguiente acción
        else if (btnverde.getCurrentTextColor()!= txtcolor.getCurrentTextColor())
        {
                cancel();
                contador2(); //Fallos
                bbddmalno();
                jugadas();
                contador();

                alerta();
                ramdom();
                tiempo();
        }
    }
});
```

Esto se realiza con todos los colores, en este caso 4.

La llamada a alerta se trata de un método para informar de que el juego ha finalizado, y por tanto nos va a informar del resultado obtenido con una alerta del tipo setMessage.

Este tipo de alerta permite incorporar botones, en este caso el de volver a jugar o la opción de salir. Si pulsamos el botón “Jugar” se pondrán todos los contadores a cero y volverá a comenzar la partida, en cambio, si se pulsa la opción de salir se irá al menú de principal del juego Stroop (StroopActivity.java).

```
public void alerta()
{

if (numJugadas == 10 && aciertos >= fallos  )
  {
     AlertDialog.Builder Alerta = new AlertDialog.Builder(this);
      Alerta.setMessage("Has Ganado:"+" "+"aciertos:"+" "+ aciertos+" "+"fallos:" +fallos);

      Alerta.setPositiveButton("Jugar", new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialogo, int con)
                    {
                    numJugadas = 0;
                    fallos = 0;
                    aciertos = 0;
                    ramdom();
                    tiempo();
                    }
                });
                //Segundo botón:
                Alerta.setNegativeButton("Salir", new   DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialogo, int con)
                    {
                        Intent a = new Intent(getBaseContext(),Menu_juegoActivity.class);
                        startActivity(a);
                    }
                });
                //Se muestra la alerta.
                Alerta.show();
                }
```

```
else if (numJugadas == 10 && aciertos < fallos)
{
    {
        AlertDialog.Builder Alerta = new AlertDialog.Builder(this);
        Alerta.setMessage("Has perdido."+ "+"aciertos:"+" "+ aciertos+" "+"fallos:"+" "+fallos);
        Alerta.setPositiveButton("Jugar", new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialogo, int con)
                    {
                        //Inicialización de la lista.
                    numJugadas = 0;
                    fallos = 0;
                    aciertos = 0;
                    ramdom();
                    tiempo();
                    }
                });
                //Segundo botón:
                Alerta.setNegativeButton("Salir", new   DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialogo, int con)
                    {
                        Intent a = new Intent(getBaseContext(),StroopActivity.class);
                        startActivity(a);
                    }
                });
                Alerta.show();
      }
   }
```

3.4.8. Juego Simón

El juego consiste en repetir una secuencia visual y sonora que se le propone al usuario. Para realizar esa secuencia de sonidos y luces, al usuario se le muestra la "interpretación automática" de la misma y se espera que la repita actuando sobre los botones.
Si se realiza con éxito, la complejidad de la secuencia se incrementa añadiendo una nota más a la misma secuencia y se repite el ciclo.

Consta de un menú principal donde el usuario puede elegir el tipo de dificultad. Se han añadido dos tipos de dificulatades:

- Normal: Toda la secuencia de colores se repite entera cada vez que se añade un color nuevo.
- Hell: La secuencia no es repetida, por tanto una vez se haya realizado el movimiento anterior se añade un nuevo color sin la repetición de la secuencia que se tenía anteriormente.

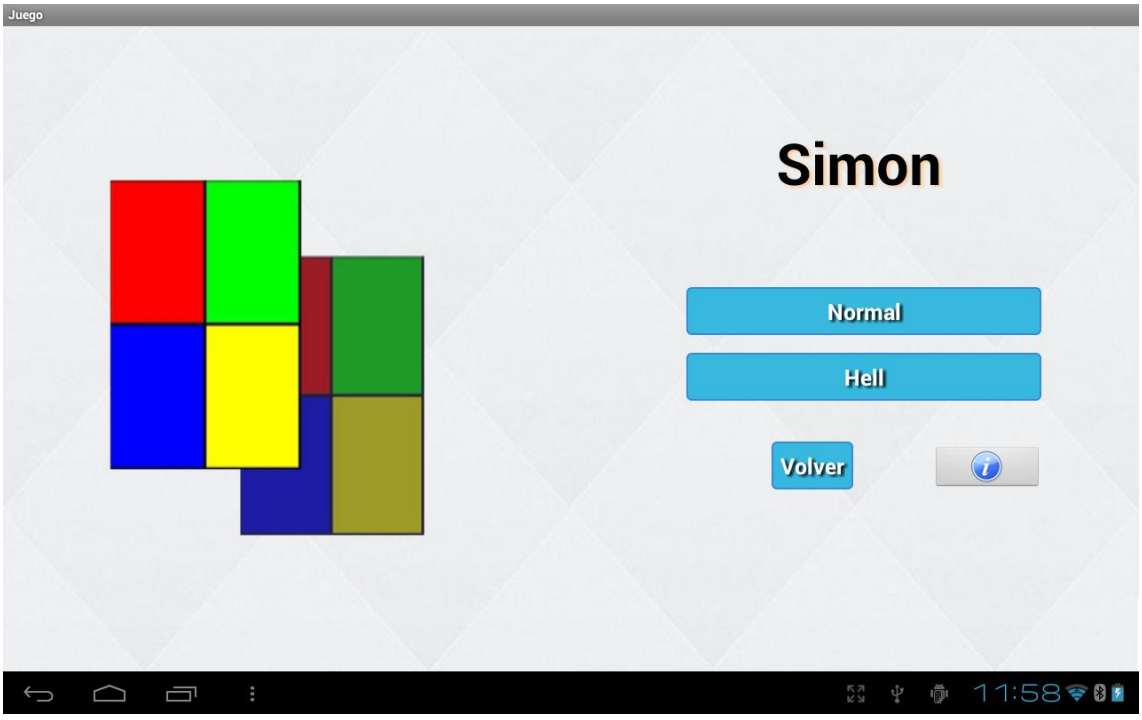


Figura 31: Menú Simón

Como en los anteriores casos se ha añadido un botón informativo del juego.

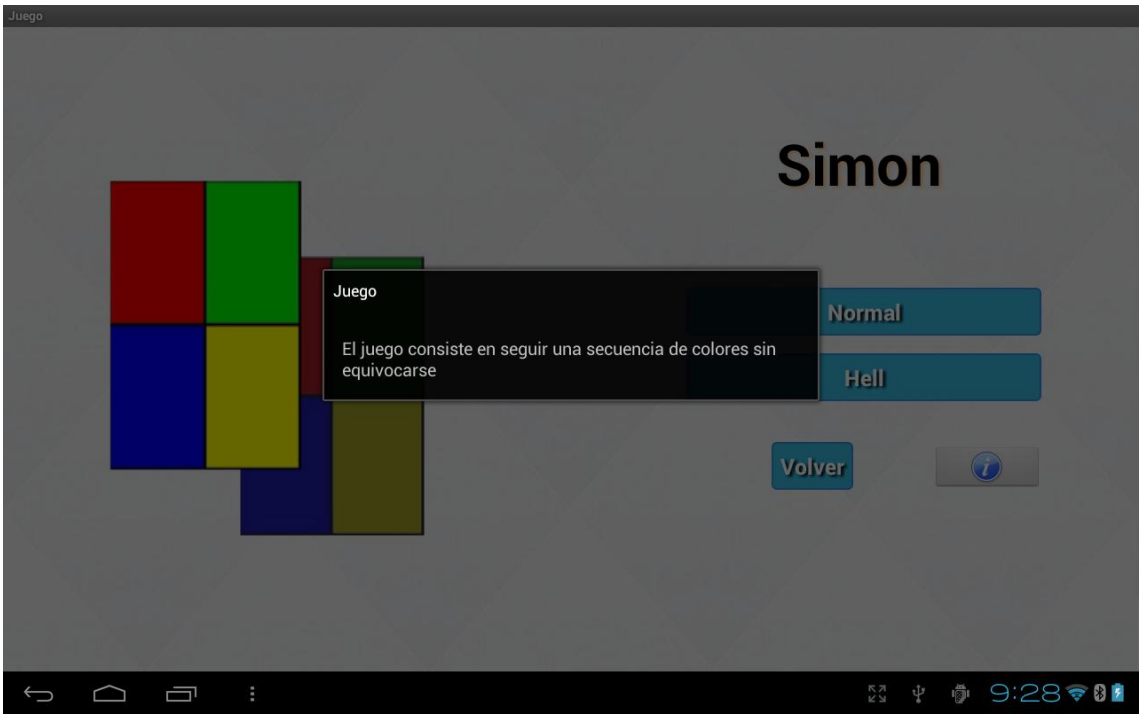


Figura 32: Instrucciones

Una vez se ha elegido la dificultad, el usuario tiene la posibilidad de elegir el tamaño del tablero, si es de tamaño 2x2 (4 colores) o de 3x3 (9 colores).

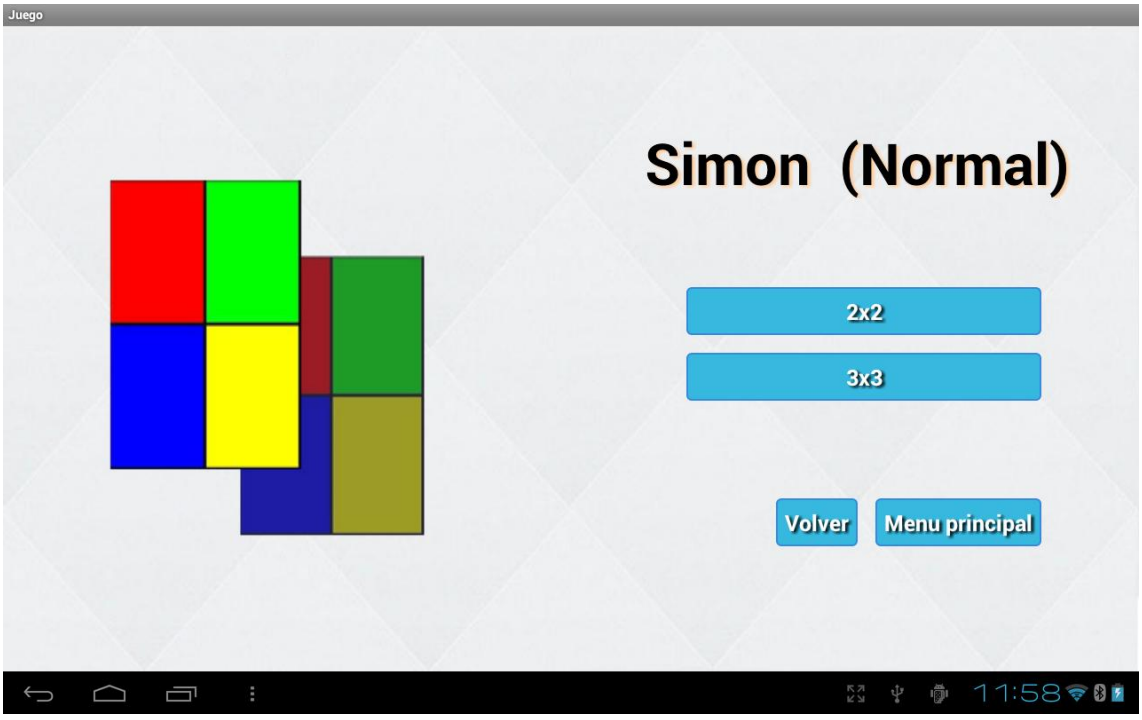


Figura 33: Menú Simón 2 (Normal)

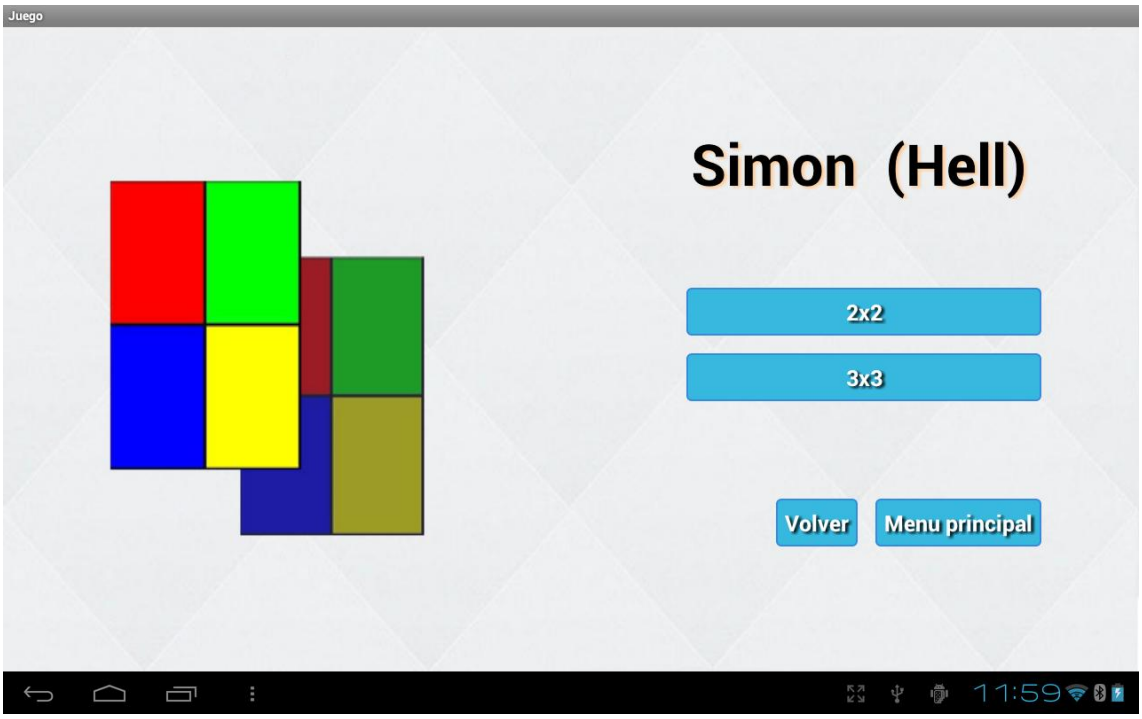


Figura 34: Menú Simón 2 (Hell)

Una vez se ha elegido el modo y la dificultad del juego se accede al mismo.

Se ha incluido un botón para que el usuario decida cuándo empezar la partida y un cuadro informativo de cómo se juega, con la posibilidad de entrar al juego o salir de él.

Figura 35: 4 Inicial

Una vez se ha accedido, se tiene el tablero con los colores en una tonalidad apagada y se reproduce un sonido. El sonido va acompañado de un color intenso y con brillo, que es el que debemos pulsar. Cada color tiene un sonido asociado, así resulta más sencillo la memorización.

Figura 36: Inicial y Pulsado

Código

El juego básicamente funciona con hilos. En el momento que se tiene que cargar un color se llama al hilo correspondiente y se muestra la imagen con el color resaltado.

Los menús de elección de niveles no tienen ningún tipo de dificultad, se declaran los botones y se les da funcionalidad.

```
public class SimonDice_DifHell extends Activity{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.simon_dificultadnuevo);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        final Button btn2 = (Button)findViewById(R.id.btn1);
        final Button btn3 = (Button)findViewById(R.id.btn2);
        final Button btnvolver = (Button)findViewById(R.id.btnvolver);
        final Button btnmp = (Button)findViewById(R.id.btnmp);

        btn2.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
            {
                    Intent i = new Intent("juego.packages.SimonDice_Inter2x2Hell");
                    startActivity(i);
                }
        });

        btn3.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
            {
                  Intent i = new Intent("juego.packages.SimonDice_Inter3x3Hell");
                  startActivity(i);
                }
        });
```

Los sonidos han sido almacenados con un número representativo a cada nota, por tanto, se realiza una consulta para obtener los sonidos con ese código y se coge el primero. Se realiza la comparación para saber si se trata de la nota que se buscaba, si coincide se carga el sonido de la nota, si no coincide se carga un sonido vacio.

```
String[] args2 = new String[] {"2"};
Cursor c2 = db.rawQuery("SELECT texto FROM Sonidos WHERE codigo= ?" , args2);
c2.moveToFirst();
String dos = c2.getString(0);

if(dos.equals("do")){
doss = sp.load(this.getContext(),R.raw.dos,1);}
else{
doss = sp.load(this.getContext(),R.raw.vacio,1);}

String[] args3 = new String[] {"3"};
Cursor c3 = db.rawQuery("SELECT texto FROM Sonidos WHERE codigo= ?" , args3);
c3.moveToFirst();
String re = c3.getString(0);

if(re.equals("re")){
reso = sp.load(this.getContext(),R.raw.re,1);}
else{
reso = sp.load(this.getContext(),R.raw.vacio,1);}
```

Esto se repite con todas las notas y sonidos de error de igual manera.

El panel informativo inicial de acceso al juego se ha declarado con alertas a las que se les han incluido los botones.

```
public void empezamos()
{

    AlertDialog alerta2;
alerta2 = new AlertDialog.Builder(this.getContext()).create();
alerta2.setTitle("Instrucciones ");
alerta2.setMessage("El juego consiste en seguir la combinación de colores y sonidos propuesta"
    + " por el móvil, cada vez se irá añadiendo una combinación mas, "
    + "el juego finalizará cuando no consigas recordarlos todos en el orden propuesto.");
alerta2.setButton("Jugar", new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialogo, int
con) {
sonidos();
contadorColores = 0;
contadorJugada = 0;

contadorFinal = 0;
contadorMostrar = 0;
lista = new ArrayList<String>();
Generador();
});
alerta2.setButton2("Salir", new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialogo, int
con) {
Intent f = new Intent("juego.packages.SimonDice_principal");
getContext().startActivity(f);
});
alerta2.show();
}
```

Aquí es donde se ha aprovechado a poner todos los contadores a cero. También se ha creado una lista donde se van a almacenar las cadena de colores generadas.

```
public void Generador()
{
String es = "";
contadorFinal= contadorFinal + 1;
contadorJugada=contadorFinal;
int x = (int)( Math.random() * 4);
switch(x)
{
case 0:
es = "rojo";
break;
case 1:
es = "azul";
break;
case 2:
es = "verde";
break;
case 3:
es = "amarillo";
break;
}
lista.add(es);
Mostrador();
}
```

En el String declarado a cero es donde se va a tener provisionalmente el color seleccionado aleatoriamente. Cuando se genera un nuevo color aumentamos el contadorFinal en una unidad, para saber cuántos colores se han mostrado.

El contador de la jugada pasa a valer lo mismo que el contador final, es decir, lleva contados el número de colores mostrados hasta el momento. Esto servirá para saber el número de veces que el jugador tiene que tocar la pantalla, en el momento en el que llegue a cero será cuando se tiene que añadir un nuevo color o repetir la secuencia añadiendo un color nuevo.

La elección del color aleatorio lo hacemos mediante un random de cuatro números. En el caso de 9 colores el random, lógicamente, será de 9. Una vez obtenemos el color en la variable “x”, con un swich vemos el color correspondiente que se debe añadir y se añade a la lista.

```
public void Mostrador()
{
Thread t = new Thread() {
public void run() {
    miManejador.post(pasoMostrador);
}

};

t.start();
```

En el método mostrador se ha creado un hilo secundario para mostrar los diferentes colores, la creación de este hilo es necesaria para que cuando se tenga que mostrar el mismo color dos

veces seguidas se vea dos veces, si no, sólo se vería una debido a que el refresco de las imágenes en android sólo se puede hacer a la salida de los métodos.

```
if(lista.size() > contadorColores ){
    s = lista.get(contadorColores);

    if(s == "rojo")
    {
        Thread t=new Thread("rojo");
        try { Thread.sleep(300); /**Tiempo entre la iluminacion de un color y el siguiente*/
        t.join();
        }catch (InterruptedException e) {
        e.printStackTrace(
        );
        }
        hiloRojoGenerado();
    }
    if(s == "azul")
    {   Thread t= new Thread("azul");
        try { Thread.sleep(300); /**Tiempo entre la iluminacion de un color y el siguiente*/
        t.join();
        }catch (InterruptedException e) {
        e.printStackTrace(
        );
        }
        hiloAzulGenerado();
    }
```

Dependiendo del color obtenido se ha llamado a los métodos que se encargan de manejar los diferentes hilos para cada color. Se ha definido el tiempo entre la iluminación de un color y otro.

```
public boolean isListo() { boolean  listo = true; return listo; }
protected void hiloRojoGenerado()
{
Thread t = new Thread()
{
public void run() {

miManejador.post(refrescarRojo);
}
};
t.start(); try{t.join();}catch (InterruptedException e){e.printStackTrace();}
try { Thread.sleep(200);
}catch (InterruptedException e) {
e.printStackTrace(
);
}
}
final Runnable refrescarRojo = new Runnable() {
public void run() {
if(doss != 0){
sp.play(doss,1,1,0,0,1);}
setBackgroundDrawable(fondorojo);

try { Thread.sleep(200); /**Tiempo entre la iluminacion de un color y el siguiente*/
}catch (InterruptedException e) {
e.printStackTrace(
);
}
contadorColores++; Mostrador();
}
};
```

El manejo de estos hilos es igual para los cuatro colores.

El manejador es el que se encarga de ejecutar refrescarRojoGenerado. En el método run () de refrescarRojo es donde se realiza la inicialización del sonido correspondiente al rojo.

Se cambia el color de fondo cargando el fondo en el que el color rojo queda resaltado entre los anteriores.

Por último, aumentamos en una unidad el contador y volvemos a llamar al método mostrador.

```
public boolean onTouchEvent(MotionEvent event) {
float x = event.getX();
float y = event.getY();

String ColorValido;
//ROJO
//En el caso de que la x sea menor que 160 y la y sea menor que 240 nos encontramos en el área del rojo.
if(x<232 && y<364)
{
ColorValido = lista.get(contadorMostrar);
if(ColorValido == "rojo")
{

hiloRojoPulsado();
contadorJugada--;
contadorMostrar++;
}

else
{
fin();
}
}
```

Este método se ha sobrescrito de la clase View. Es el que se encarga de capturar eventos ocurridos en la pantalla mediante el contacto con ella. Este método se ejecuta cuando se toca la pantalla. Se obtienen las coordenadas “x” e “y” de la pulsación de la pantalla.

Se declara el String que va a recoger el color correspondiente de la lista.
Declaramos el área donde se encuentra el color rojo, en este caso cuando la “x” es menor de 160 y la “y” menor de 240 (arriba a la izquierda).
Por último, realizamos la comprobación de que realmente se ha pulsado el color rojo. Se ha lanzado el hilo que permitirá el cambio de color y la reproducción del sonido. Se ha disminuido el contador de la jugada para saber cuándo se acaba la lista de colores y aumentado el contador mostrar para que la próxima vez que se pulse la pantalla obtenga el color siguiente de la lista.
En el caso de que el color pulsado no se corresponda con el que en ese momento tocaba en la lista, se realiza una llamada al método fin(), por tanto se habrá acabado la partida.

La única diferencia entre la dificultad “Normal” y dificultad “Hell” es que la segunda no repite la secuencia.

```
public void fin()
{
        if(err != 0){
        sp.play(err,1,1,0,0,1);}

int puntuacion;
if(contadorFinal==1)
puntuacion=0;
else
puntuacion = contadorFinal;
AlertDialog alerta;
alerta = new
AlertDialog.Builder(this.getContext()).create();
alerta.setTitle("Fin del juego");
alerta.setMessage("Te has equivocado! \nHas obtenido una puntuacion de: "+ puntuacion +" puntos. \n¿Deseas volver a jugar?");
alerta.setButton("SI", new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialogo,
int con) {
    sonidos();
    contadorColores = 0;
    contadorJugada = 0;
    contadorFinal = 0;
    contadorMostrar = 0;
    lista = new ArrayList<String>();
    //Llamada al método generador de combinaciones.
    Generador();
}
});
alerta.setButton2("NO, salir del juego", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialogo,
int con) {
    Intent f = new Intent("juego.packages.SimonDice_principal");
    getContext().startActivity(f);
    }
});
alerta.show();
}
```

El método fin () es el que da la posibilidad de salir del juego. Se ha mostrado la puntuación obtenida mediante una alerta donde se han incluido dos botones en los que se puede salir del juego o volver a jugar. Si el jugador decide volver a jugar, se lleva a cabo la re inicialización de los contadores y de la lista, posteriormente comienza el juego de nuevo. En cambio, si el jugador ya no quiere jugar más, saldrá del juego e irá al menú principal del Simón.

3.4.9. Pruebas

Según se ha ido desarrollando la aplicación se han ido realizando pruebas tanto de caja negra como de caja blanca.

- Pruebas de especificación o caja negra: Verifican el comportamiento externo de la aplicación sin que el usuario sepa lo que hay por detrás.

  Se han probado todos los juegos, con el que más pruebas se han hecho ha sido con el "Memory" y con el "Parejas" ya que se han probado distintos tipos de animales para ver cuáles son los que se ven mejor.

  En el juego "Hanoi" se han probado todos los niveles, también se ha probado que la estrella se ilumina correctamente si se realiza la partida en el menor número de movimientos posibles.

  La aplicación a sido probada por varios compañeros de la empresa llegando a crearse pequeños piques sobre quién conseguía hacer una puntuación más alta en el juego Simón por ejemplo.

  También, como he mencionado en el desarrollo, la aplicación fue probada por varios usuarios con daltonismo. No se obtuvieron buenos resultados, había colores que

confundían o no veían claramente, por tanto con su opinión y sus consejos se mejoró el aspecto de la aplicación.
El juego que más se vio afectado por esta prueba de daltonismo fue el juego memory, que al principio tenía unas cartas con un fondo verde que dificultaba la distinción del animal en usuarios con daltonismo. Las cartas fueron cambiadas a cartas con un fondo blanco para que resultase mucho más sencilla la distinción del animal.

Con el juego Simón también hubo que ajustar determinados parámetros debido a que el color se "iluminaba" muy poco y en ocasiones no se podía distinguir el color que había que pulsar. Una vez cambiado se vio en las pruebas la mejoría notable de los resultados.

- Pruebas de estructura o caja blanca: Verifican la implementación interna de la unidad.

  Se han emulado todas las posibles situaciones que pueden generarse en la aplicación para comprobar la correcta resolución de las mismas. Se han evaluado situaciones de error tanto en los juegos como en la conexión con la base de datos en la introducción y consulta de datos.

  Si no hay una conexión a internet disponible en el dispositivo, no se podrá acceder a la aplicación. Esto es debido a que la autentificación tanto del usuario y su contraseña lo hace vía internet, por tanto nos dará un error si no tenemos dicha conexión. Se trata de una mejora importante que es comentada en el apartado 4.2. Líneas futuras.

4. Conclusiones y líneas futuras

4.1. Conclusiones

Los conocimientos que he adquirido durante los 8 meses que ha durado este proyecto han sido notables. He sido capaz de crear varios juegos y he manipulado el servidor de la empresa, lo cual me ha hecho aprender mucho.

He aprendido a trabajar mejor en equipo. Me he enfrentado a reuniones con pedagogos en las que he tenido que exponer soluciones y puntos de vista alternativos. He tenido que solucionar los problemas que iban surgiendo lo más eficientemente y rápidamente posible sin permitir que se me fuese de las manos.

Se tomó una mala opción al comienzo del proyecto que fue el hecho de corregir todo a partir de lo que ya tenía la empresa creado del proyecto. Esta decisión fue errónea ya que habría sido mucho más sencillo y sobretodo mucho menos costoso, haber comenzado la aplicación desde cero. Se perdió mucho tiempo en el entendimiento del código que había.

4.2. Líneas futuras

El objetivo es seguir con la aplicación adelante, que no sea un proyecto que se queda en la nada. Se quieren añadir más juegos como el Test de caras que mejora la atención sostenida y vigilada, el Go-no-Go que trabaja la inhibición o el juego Wisconsin que mejora la flexibilidad cognitiva.

La manera de volcar los datos en la base de datos es un punto claro a modificar. Una mejora significativa sería que en vez de volcar los datos en la base de datos instantáneamente, se hiciese una copia en el dispositivo y este, una vez al día, los volcase todos de una vez a la base de datos. Esto evita pérdida de datos y el no tener que estar obligado a que el dispositivo este constantemente conectado a internet.

Un problema importante es que la autentificación tanto de usuario como de contraseña se realiza vía internet, por tanto si el dispositivo por alguna razón no dispone de internet, no se podrá utilizar la aplicación.

Sería interesante que los juegos se mostrasen dependiendo del usuario. Es muy probable que los usuarios no tengan igual de dificultad en unas áreas que en otras, por tanto, cuando un usuario accede a la aplicación solamente aparezcan aquellos juegos recomendados especialmente para él. Se podría crear una interfaz para el tutor que decida qué juegos son realmente valiosos para el usuario y que los integre o elimine según su criterio y evolución con el tratamiento.

Por último la mejora del aspecto de la aplicación, realización de mayor número de pruebas para asegurarse de que la aplicación es sencilla para cualquier tipo de usuario.

5. Bibliografía

Programación Android

http://www.sgoliver.net/blog/?page_id=3011

Programación Java

http://www.publispain.com/supertutoriales/diseno/java/cursos/1/java.pdf

Bases de datos

http://blog.netrunners.es/usar-nuestra-propia-base-de-datos-sqlite-en-android/

Libros

ZECHNER, M. (2011). *Beginning Android Games*. New York: Apress.

JUEGOS COGNITIVOS PARA NIÑOS CON TDAH

Nekane Bastero Huarte

¿Qué es el trastorno de TDAH?

- Trastorno del comportamiento caracterizado por distracción moderada a grave, períodos de atención breve, inquietud motora, inestabilidad emocional y conductas impulsivas.

- Las principales características de este trastorno son:
  - Actividad excesiva e inapropiada en relación a la tarea propuesta.
  - Poca atención mantenida.
  - Dificultad para inhibir impulsos.
  - Bajo rendimiento escolar y autoestima.

- Un 67% de los niños diagnosticados continúan en su edad adulta.

Antecedentes

- La aplicación ya estaba empezada cuando me asignaron el proyecto, el estado en el que se encontraba la aplicación era desastroso:
  - Mezclado
  - Los nombres de las clases y de las variables no eran nada intuitivas
  - Los juegos no iban bien
  - La base de datos estaba creada (sólo algunas cosas) de manera local en cada dispositivo y por tanto los tutores o pedagogos no podían hacer las consultas pertinentes.

Acceso a la Aplicación

Pantalla Bienvenida

Si se tiene un usuario ya creado

Pantalla Nuevo Usuario

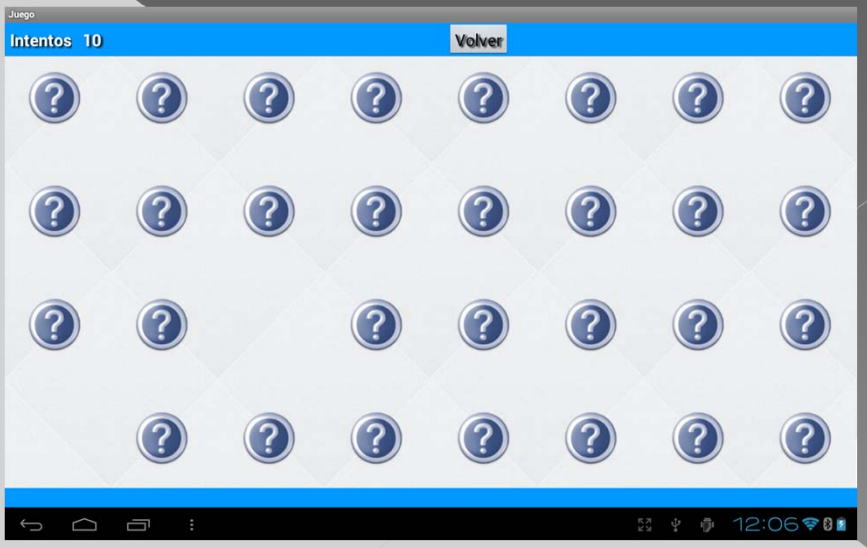
Si no se tiene el usuario creado

Menú Principal

- **Memory:** Refuerza la memoria visual.
- **Torres de Hanoi:** Refuerza la planificación y organización
- **Parejas:** Refuerza la memoria visual.
- **Stroop:** Refuerza la atención focalizada y selectiva
- **Simón:** Refuerza la memoria visual y sonora.

Memory

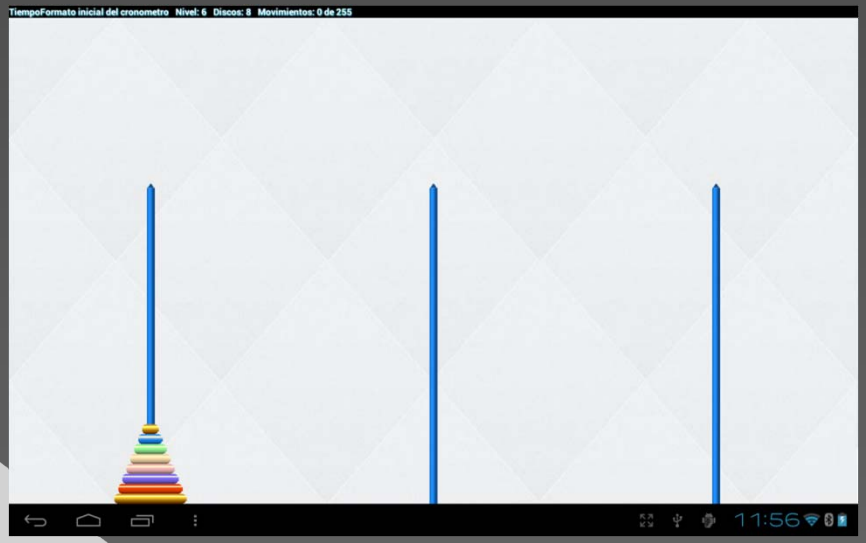
Mosaico de cartas que se sitúan boca abajo, y que se deben ir descubriendo por parejas para localizar las parejas de cartas iguales.

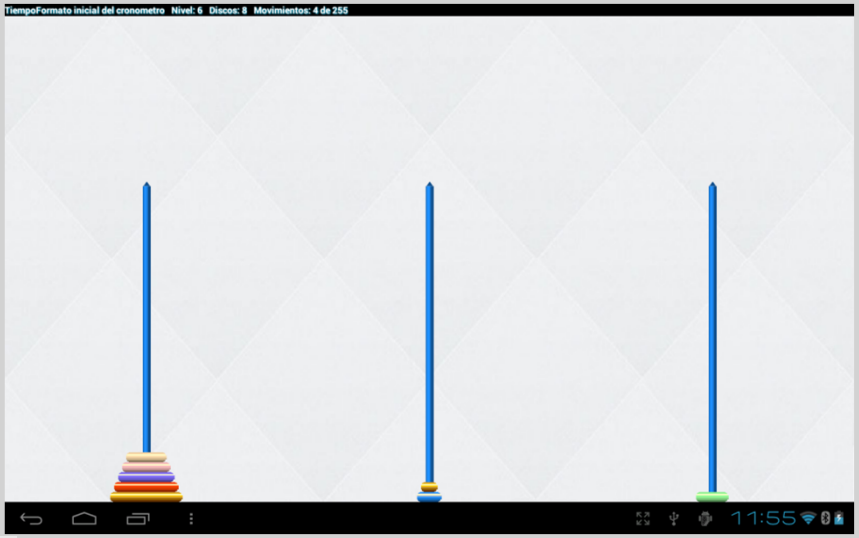


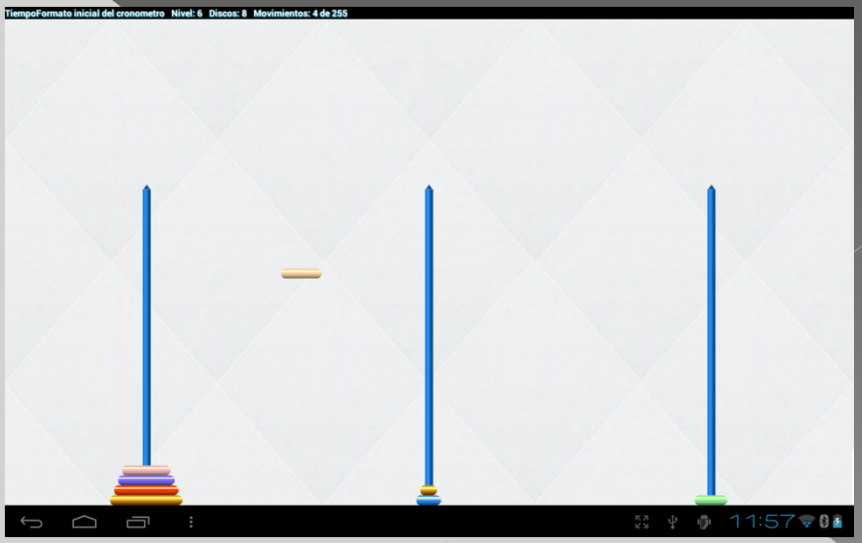
Torres de Hanoi

- El jugador debe mover todos los discos a otro de los ejes vacios, colocándolos en el mismo orden del que partían. Para ello puede ayudarse de los dos ejes restantes que se encuentran vacios.

- Nunca puede haber un disco más grande sobre otro más pequeño

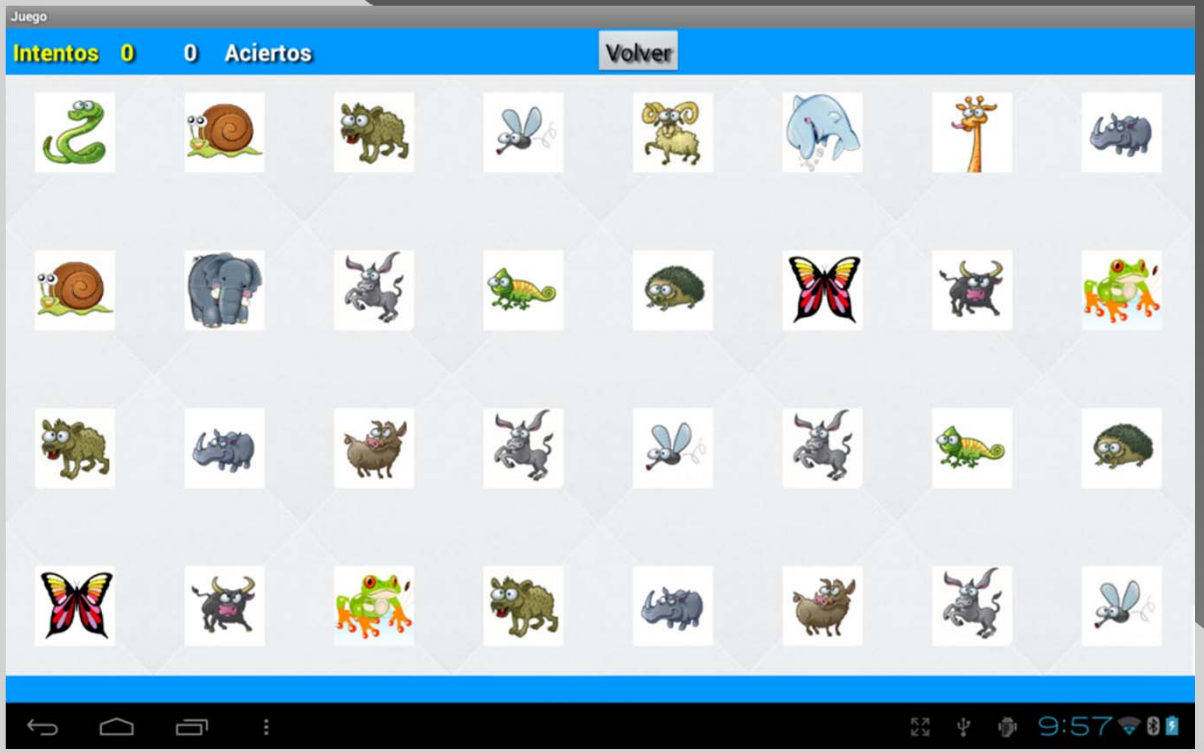






Parejas

- Realizar todas las parejas que se puedan en el menor número de movimientos posible.

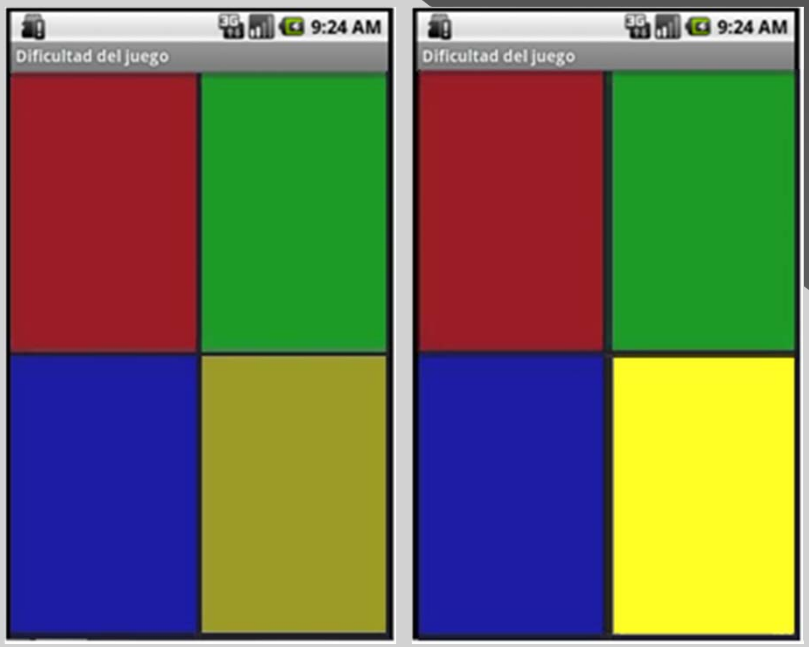


Stroop

El juego consiste en pulsar el botón que simboliza el color del que está pintada la palabra.

Simón

Dificultad 2x2



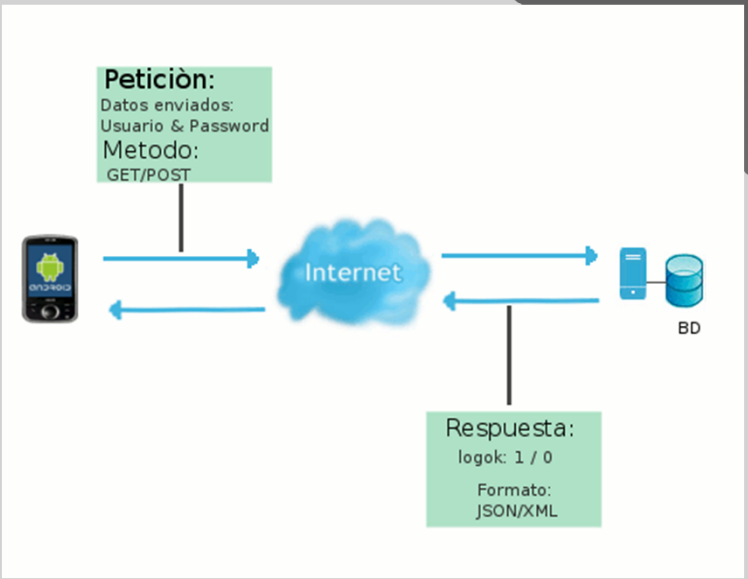
Dificultad 3x3

Repetir una secuencia visual y sonora que se le propone al usuario.

Para realizar esa secuencia de sonidos y luces, al usuario se le muestra la “interpretación automática” de la misma y se espera que la repita actuando sobre los botones.

Conexión Base de Datos

- Se hace una petición, en este caso mediante el método POST, incluyendo los datos a enviar en un array. El envío se realiza con JSON.
- Mediante internet nos conectamos con el servidor donde se encuentra la base de datos dándonos la respuesta a la petición.
- La puerta de enlace entre el servidor y la base de datos son 4 archivos php.
- Se ha utilizado un puerto libre del router de la empresa.
- Igualmente, la respuesta es devuelta con JSON.



Líneas Futuras

- Se quieren añadir más juegos como el Test de caras que mejora la atención sostenida y vigilada, el Go-no-Go que trabaja la inhibición o el juego Wisconsin que mejora la flexibilidad cognitiva.

- Una mejora significativa sería que en vez de volcar los datos en la base de datos instantáneamente, se hiciese una copia en el dispositivo y este, una vez al día, los volcase todos de una vez a la base de datos. Esto evita pérdida de datos y el no tener que estar obligado a que el dispositivo este constantemente conectado a internet.

- Los juegos se mostrasen dependiendo del usuario.

Agradecimientos

JOB^CCOMMODATION

upna
Universidad
Pública de Navarra
Nafarroako
Unibertsitate Publikoa