UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
ELECTRONICS AND COMPUTERS FIELD

DESIGN AND DEVELOPMENT OF A MOBILE CITY GAME

CITYSCRABBLE

DIPLOMA THESIS OF

MARCO ANTONIO MUÑOZ ALONSO

STUDENT OF THE UNIVERSIDAD DE VALLADOLID

SUPERVISOR: PROF. N. AVOURIS

PATRAS JULY 2012

CERTIFICATION

It is certified that diploma thesis with the title:

DESIGN AND DEVELOPMENT OF A LOCATION AWARE GAME BASED ON THE ANDROID PLATFORM

Of the sudent of the University of Valladolid

Muñoz Alonso Marco Antonio

(Surnames) (Name)

Was presented in public at the department of Electrical and Computer Engineering of the University of Patras, Greece on July 13.

The supervisor The head of the electronics and computers division

First of all, especial thanks to Christos Sintoris for his support and guide every day during the last six months. I learned a lot with him and without his effort and dedication have not been possible to complete the project.

Thanks also to all the people that were working with me in department. In Particular, Ruben Llorente and Hector Fadrique, Erasmus students from University of Valladolid for all the time that we spent together working in the lab.

I would like to thank all my Greek friends and Erasmus people, and most specialty to the people who attended the evaluation session and therefore have worked directly with the project.

I want also to thank Professor Nikolaos Avouris (my Erasmus coordinator in Patra) for bring me this interesting project.

Finally I would like thank my family, especially to my parents and my brother, for his motivation every day from Spain.

Contens

Abstract

The broad objective of the thesis is make implementation of CityScrabble, a network multiplayer game, for mobile devices based on the Android platform, using the programming language Java for Android.

The specific objectives of this study were outlined as follows:

a) To improve the communications protocol.
b) To make the design, interface and the user interaction better.
c) To correct errors in the last game.
d) To implement new features.
e) Evaluation session

The idea to develop this game begins in a previous game that was developed by Diego Rodriguez Puerto, a student from University of Valladolid.

He implemented the game using a Web service based on REST architecture, so the main aim of my project was replace the existing protocol and implement XMPP. It is an open and extensible protocol based on XML, originally designed for instant messaging. This is the protocol selected for Google messaging service Google Talk, Facebook and Myspace and other for chat. For this, I have integrated a Smack library that allows connecting, sending and receiving messages between the players' devices and XMPP server.

The most important reason why it was decided to change the protocol of the game is that if we have more than one scenario (more than a group of People playing CityScrabble at the same moment), we need to create another chat room in the server in order to differentiate between different kinds of games that are being run on the same server.

In order to improve the design, as the last interface was adequate and worked property, some parts have been modified to optimize the game and make it more intuitive. A login screen, a progress dialog, a new toggle button and toasts have been integrated to correct errors in the system and improving the Diego Rodriguez version.

Others objectives were also correct errors in the last application and implement new features. Some of these features have been proposed is the method of unlocking the game elements using GPS integrated in Android devices, a new login design that allows create and join games, different options depending of the scenario and program a real communications server.

After completing the application, we have prepared an evaluation session in a real scenario with a group of people in order in order to see if the end user like the application has been designed and consider proposals to improve it in the future.

CHAPTER 1: Design

We focused the design work with three overarching goals for our core apps and the system at large. As you design apps to work with Android, consider these goals:

- Beauty is more than skin deep. Android apps are sleek and aesthetically pleasing on multiple levels. Transitions are fast and clear; layout and typography are crisp and meaningful. App icons are works of art in their own right. Just like a well-made tool, your app should strive to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful. A beautiful surface, a carefully-placed animation, or a well-timed sound effect is a joy to experience. Subtle effects contribute to a feeling of effortlessness and a sense that a powerful force is at hand.
- Android apps make life easier and are easy to understand. When people use your app for the first time, they should intuitively grasp the most important features. The design work doesn't stop at the first use, though. Android apps remove on going chores like file management and syncing. Simple tasks never require complex procedures, and complex tasks are tailored to the human hand and mind. People of all ages and cultures feel firmly in control, and are never overwhelmed by too many choices or irrelevant flash.
- It's not enough to make an app that is easy to use. Android apps empower people to try new things and to use apps in inventive new ways. Android lets people combine applications into new workflows through multitasking, notifications, and sharing across apps. At the same time, your app should feel personal, giving people access to superb technology with clarity and grace.
- Allow people to directly touch and manipulate objects in your app. It reduces the cognitive effort needed to perform a task while making it more emotionally satisfying.
- People love to add personal touches because it helps them feel at home and in control. Provide sensible, beautiful defaults, but also consider fun, optional customizations that don't hinder primary tasks.
- Learn peoples' preferences over time. Rather than asking them to make the same choices over and over, place previous choices within easy reach.[i]

Application Architecture

As mobile devices become more common, it becomes imperative to understand how this environment poses unique application architecture challenges.

We start by describing some of the general concepts and terms behind client-server architectures and follow this by describing clients and servers and the connectivity between them. We then present several interesting architectural patterns and describe why they are useful as general mobile application architecture solutions. Finally, we discuss some of the tenets behind good architectural design and the considerations you need to be aware of when designing mobile applications.

Application architectures are often modelled to highlight or illustrate the overall layout of the software (e.g., application code and platform) and hardware (e.g., client, server, and network devices). While there are many possible combinations of software and hardware, application architectures often fall into a series of recognizable patterns.

Application architectures are commonly modelled in terms of a client-server architecture wherein one or more client devices requests information from a server device. The server typically responds with the requested information.[ii]

The following photo shows an illustration with CityScrabble logical architecture:

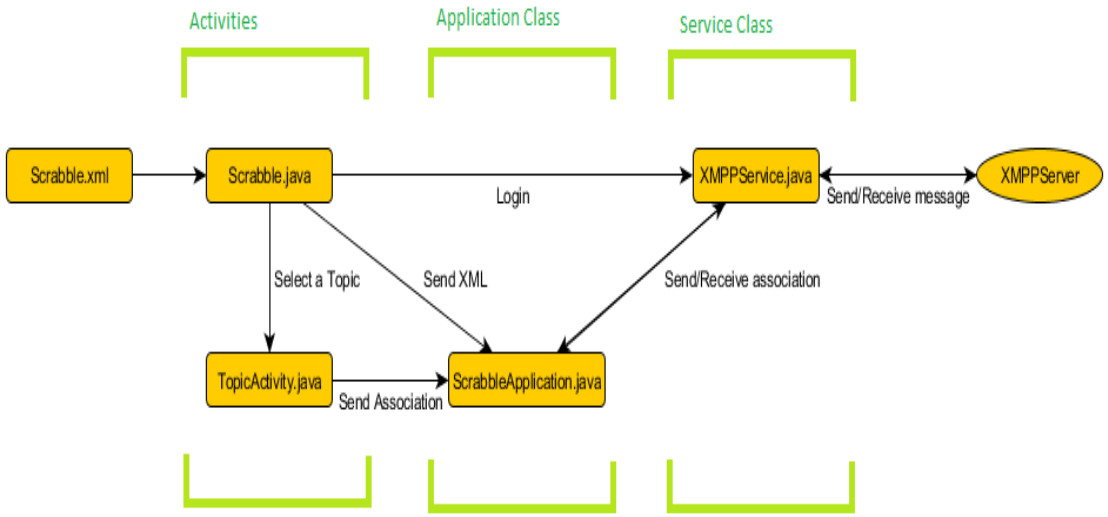


Figure 1. CityScrabble Architecture

Then, I will describe the main functionality of all parts of the application architecture:

Activities

There are two activities in the application Scrabble Activity and Topic Activity:

> **Scrabble Activity:** It start reading Scrabble.xml and send all of this data to the Application Class. In this activity the player can log in and send the username and the password to XMPP Service. Then, he has to select a Topic in order to start playing.
>
> **Topic Activity:** Starts when the player selects a topic from the previous activity. Display information about the Topic with their respective keys. Here the player has to unlock an exhibit in order to make a correct association.

Application Class

Java class that makes the following tasks:

- Store all the important data about the game:
  - Topics, keys and exhibits.
  - Player information (Id and username)
  - Score
- Communications between the service and activities through Intents and BroadcastReceivers.
- Update the score

XMPP Service

Receive username and password from Scrabble Activity, do the connection with the server and wait messages from the server (rest of the players) and the activities.

XMPP Server

Server processes incoming messages from mobile devices and sends a broadcast message to the rest of the players. It also responds to the player who made the request.

Working example

1. The application is launched and after the loading screen, Scrabble activity read the XML file and sends this information to the Application class.
2. At this point the player has to log in into the game. The user name and password are sent to the service to connect with the XMPP server
3. The player select a Topic and goes to the second Activity (Topic Activity)

4. When the player makes an association (Key – Exhibit), Topic Activity sends a broadcast intent (Id, Key and exhibit data) to Application class.
5. Application class transfers it to the XMPP service that sends a message to the XMPP Server.
6. The Server sends a broadcast message to the rest of the players that are playing and responds to the player that made the association.
7. The message arrives from the service and Application Class will be responsible for updating the score (if it is a correct association) and to inform the player about the request that he did.

Interface

In this section I will discuss about the previous Interface and the changes that I did in order to improve it. The main objective of the project was not completely change the interface because previously it worked correctly but I made some small changes in it.

First design

In *CityScrabble v1*, we could see the nine images buttons of the topics, which were on the Relative Layout, for a *GridView* that contains these images. This task is done on the class *TopicTileSet*. Due to this improvement it got a better screen structure and a better load time. In addition, the images now are joined in the same widget, getting a better management.In a similar task, the linear layout that grouped the different components of the state informing to the player of the game was replaced with for a *Scorebar*. *Scorebar* is a class that extends the View class and will enable many more design options due to implementation of the *OnDraw* method.

Figure 2.Previus Design

For the second screen we had a new UI after to implement all the necessary features of the game, such as communications, management and updating of the game status.

The design of the topic screen contains the following elements:

- A *ScoreBar* that shows the color of the player, the name of the player and the score
- A small thumbnail of the topic with a title and the description of the topic

- A radio group of radio buttons that represents the hints
- The text of the hint that has been selected on the radio group
- A Toggle button that is used to link the hints with the exhibits
- A Gallery that shows the exhibits
- A text that describes the exhibit selected

When the game starts and the player has not decided yet the exhibits with the camera of the mobile using the QR codes, the exhibits will appear empties and the button that let to use the camera, will be available and visible.

Final design

Topic Screen

As the first activity was attractive, adequate and works properly, I just added a button Login that allows the user to connect to game server. The login Dialog consists of two text boxes where the player enters the username and password. «OK» button sends the data to the server and the cancel button cancels the request.

We can see in the following screenshot:

Figure 3. Login Screen

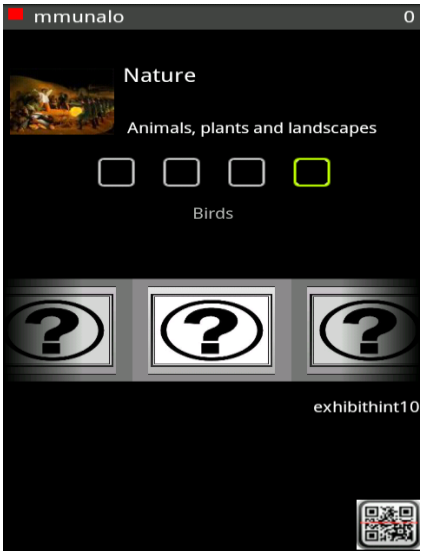


Figure 4. Topic Activity Screen with QR button

Topic Activity Screen

In the Topic Activity, changes in the interface were made in the locker, and buttons that open the scanner (QR button) and the geographical situation (GEO button) depending on the type of exhibit that the player selected:

QR Button

Standard button was replaced by a more aesthetic.Image Button with the form of a QR scanner.

GEO Button

Although the position detector that works by GPS has not been implemented in the application, I made ready the button allows the user to unlock the exhibits of this type.

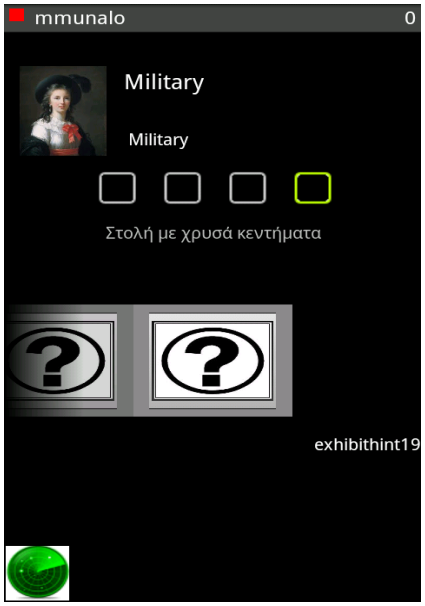


Figure 5. Topic Activity Screen with GEO button

Lock Button

The lock implemented in the previous application works properly but we decided it was a little bit confusing for the player to interact with it so we decided to change the icons and add colors in order to enhance the gaming experience.

Now the player can intuit better when the association is available, or may revoke an association that he made before:

CHAPTER 2: The protocol

XMPP

In computing a protocol is a set of rules for two machines are capable of understanding. This gives us the possibility that two (or more) machines (or programs within a machine) to share information.

Extensible Messaging and Presence Protocol, better known as XMPP (Extensible Messaging Protocol and communication of presence) is an open and extensible protocol based on XML, originally designed for instant messaging.

With the XMPP protocol is established a platform for the exchange of XML data that can be used in instant messaging applications. The characteristics in terms of adaptability and simplicity of XML are inherited in this manner by the XMPP protocol.

Unlike proprietary protocols and message exchange, XMPP is documented and are encouraged to use in any project. There are free servers and clients that can be used free of charge.

As I said begore, change the communications protocol of messages between the mobile device and the server was the primary objective of my project. The previous protocol (REST) has been replaced by XMPP.

History

Jeremie Miller began the Jabber project in 1998. His first major software release came in May 2000. The main product of the project was *jabberd*, an XMPP server.

This initial XMPP protocol created the basis for the XMPP, published as RFC 3920. He has often been considered a competitor of simple, based on the SIP protocol as standard protocol for instant messaging and presence notification.

Jabber Software Foundation was renamed XMPP Standards Foundation on 15 January 2007.
Since 2005, there were about six XMPP server implementations, written in different programming languages.

At first, Android included support for XMPP whose first implementation was provided by the Smack library, developed by Jive Software. In later versions of the SDK was limited to an IM service *GTalkService* moving to call for that in the latest

versions (0.9 and 1.0) were eliminated its functionality. This measure was because Rich Cannings (Google security researcher) discovered several vulnerabilities that could reveal information about users. In order to solve the problems of safety certificates Google's servers and to make submissions on XMPP is must apply a patch to the Smack library.

First of all, I think that it is important to describe the architecture. I will explain why we chose XMPP, and I will explain how to get the libraries and then talk about the procedure to send and receive messages using the Smack library.

Why XMPP?

The most important reason why it was decided to change the protocol of the game is that if we have more than one scenario (more than a group of People playing CityScrabble at the same moment), we need to create another *ChatRoom* in the server in order to differentiate between different kinds of games that are being run on the same server. For example, there are two groups of people playing in the city of Patras and another group playing in Valladolid.

Also, as all protocols have its advantages and disadvantages:

Advantages

Decentralization: The architecture of the XMPP network is similar to e-mail, you can set up your own XMPP server, without any central server.

Open Standards: The Internet Engineering Task Force has formalized a protocol XMPP instant messaging technology standard and specifications have been published as RFC 3920 and RFC 3921. The development of this technology is not tied to any particular company and does not require payment of licenses.

History: XMPP technologies have been in use since 1998. There are multiple implementations of the XMPP standards for clients, servers, components and libraries, with the support of companies like Sun Microsystems and Google.

Security: XMPP servers can be isolated from the public XMPP, and have robust safety systems to support the use of encryption systems, the XMPP Standards Foundation offers XMPP server administrators xmpp.net certification authority offering free digital certificates.

Flexibility: Can be made on functionality as XMPP, to maintain interoperability, common extensions are managed by the XMPP Software Foundation.

Disadvantages

Presence data overload: Typically about 70% of the traffic between servers are occurrence data, and about 60% of these transmissions are currently being studied redundancy. New protocols can be used to solve this problem.

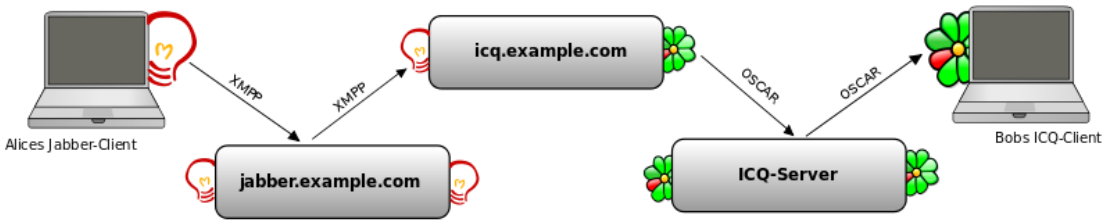
Scalability: XMPP also suffers the same problem of redundancy in services and subscription *ChatRoom*.

No binary data: XMPP is encoded as a single long XML document, making it impossible to deliver unmodified binary data. However, file transfers have been solved using other protocols such as HTTP. If unavoidable, XMPP can also transfer all the data encoded.

How works XMPP?

One of the original design goals of the early open-source community was enabling users to connect to multiple instant messaging systems (especially non-XMPP systems) through a single client application. This was done though entities called transports or gateways to other instant messaging protocols, but also to protocols such as SMS or email. Unlike multi-protocol clients, XMPP provides this access at the server level by communicating via special gateway services running alongside an XMPP server. Any user can "register" with one of these gateways by providing the information needed to log on to that network, and can then communicate with users of that network as though they were XMPP users. Thus, such gateways function as client proxies (the gateway authenticates on the user's behalf on the non-XMPP service). As a result, any client that fully supports XMPP can access any network with a gateway without extra code in the client, and without the need for the client to have direct access to the Internet. However, the client proxy model may violate terms of service on the protocol used (although such terms of service are not legally enforceable in several countries) and also requires the user to send their IM username and password to the third-party site that operates the transport (which may raise privacy and security concerns).

Another type of gateway is a server-to-server gateway, which enables a non-XMPP server deployment to connect to native XMPP servers using the built in interdomain federation features of XMPP. Such server-to-server gateways are offered by several enterprise IM software products.[iii]



Smack

Smack is an Open Source XMPP (Jabber) client library for instant messaging and presence. A pure Java library, it can be embedded into your applications to create anything from a full XMPP client to simple XMPP integrations such as sending notification messages and presence-enabling devices.

- Extremely simple to use, yet powerful API. Sending a text message to a user can be accomplished in only a few lines of code:

```
Connection connection = new XMPPConnection("servername");
connection.connect();
connection.login("user", "password");
Chat chat = connection.getChatManager().createChat("receiver@domain",
new MessageListener() {

    public void processMessage(Chat chat, Message message) {
        System.out.println("Received message: " + message);
    }
});
chat.sendMessage("");
```

- Doesn't force you to code at the packet level, as other libraries do. Smack provides intelligent higher level constructs such as the Chat and Roster classes, which let you program more efficiently.
- Does not require that you're familiar with the XMPP XML format, or even that you're familiar with XML.
- Provides easy machine to machine communication. Smack lets you set any number of properties on each message, including properties that are Java objects.
- Open Source under the Apache License, which means you can incorporate Smack into your commercial or non-commercial applications.

Working with Smack libraries

At first tests were made with Smack API 2.3.3 and received errors running the program. So, I decided to change the .jar files by downloading a new library from the following website:

http://beem-project.com/projects/beem

Beem provides a full featured and easy to use a Jabber/XMPP client on Android. It is under constant development. Releases are usually frequent and driven by user contributions, such as bug reports and patches.

Debugging the application with this library, I could properly handle the connection to the server, send and receive messages so I decided to work with it.

Creating a Connection

The *org.jivesoftware.smack.Connection* class manages your connection to an XMPP server. The default implementation is the *org.jivesoftware.smack.XMPPConnection* class. Two constructors are mainly used. The first, *XMPPConnection*(String) takes the server name you'd like to connect to as an argument. All default connection settings will be used:

- A DNS SRV lookup will be performed to find the exact address and port (typically 5222) that the server resides at.
- The maximum security possible will be negotiated with the server, including TLS encryption, but the connection will fall back to lower security settings if necessary.
- The XMPP resource name "Smack" will be used for the connection.

Alternatively, you can use the *XMPPServer(ConnectionConfiguration)* constructor to specify advanced connection settings. Some of these settings include:

- Manually specify the server address and port of the server rather than using a DNS SRV lookup.
- Enable connection compression.
- Customize security settings, such as flagging the connection to require TLS encryption in order to connect.
- Specify a custom connection resource name such as "Work" or "Home". Every connection by a user to a server must have a unique resource name. For the user "jsmith@example.com", the full address with resource might be

"jsmith@example.com/Smack". With unique resource names, a user can be logged into the server from multiple locations at once, or using multiple devices. The presence priority value used with each resource will determine which particular connection receives messages to the bare address ("jsmith@example.com" in our example).

Connect and disconnect

```
// Create the configuration for this new connection
ConnectionConfiguration config = new ConnectionConfiguration("jabber.org",
5222);
config.setCompressionEnabled(true);
config.setSASLAuthenticationEnabled(true);

Connection connection = new XMPPConnection(config);
// Connect to the server
connection.connect();
// Log into the server
connection.login("username", "password", "SomeResource");
....
// Disconnect from the server
connection.disconnect();
```

By default Smack will try to reconnect the connection in case it was abruptly disconnected. Use *ConnectionConfiguration#setReconnectionAllowed*(boolean) to turn on/off this feature. The reconnection manager will try to immediately reconnect to the server and increase the delay between attempts as successive reconnections keep failing.
In case you want to force a reconnection while the reconnection manager is waiting for the next reconnection, you can just use *Connection#connect()* and a new attempt will be made. If the manual attempt also failed then the reconnection manager will still continue the reconnection job.

Messaging using Chats

Sending messages back and forth is at the core of instant messaging. Although individual messages can be sent and received as packets, it's generally easier to treat the string of messages as a chat using *theorg.jivesoftware.smack.Chat* class.

Receiving a message

A chat creates a new thread of messages (using a thread ID) between two users. The following code snippet demonstrates how to create a new Chat with a user and then send them a text message:

```
// Assume we've created a Connection name "connection".
ChatManager chatmanager = connection.getChatManager();
Chat newChat = chatmanager.createChat("jsmith@jivesoftware.com", new
MessageListener() {
```

```
    public void processMessage(Chat chat, Message message) {
        System.out.println("Received message: " + message);
    }
});

try {
    newChat.sendMessage("Howdy!");
}
catch (XMPPException e) {
    System.out.println("Error Delivering block");
}
```

As we can see in the code, jsmith@jivesoftware.com is the user that we are listening in order to receive chat messages from him.

Sending a message

The *Chat.sendMessage*(String) method is a convenience method that creates a Message object, sets the body using the String parameter, then sends the message. In the case that you wish to set additional values on a Message before sending it, use the *Chat.createMessage*() and *Chat.sendMessage*(Message) methods, as in the following code snippet:

```
Message newMessage = new Message();
newMessage.setBody("Howdy!");
message.setProperty("favoriteColor", "red");
newChat.sendMessage(newMessage);
```

"Howdy!" is the chat message that we are sending. The remaining lines are not necessary. Just to apply attributes such as color.

You'll also notice in the example above that we specified a *MessageListener* when creating a chat. The listener is notified any time a new message arrives from the other user in the chat. The following code snippet uses the listener as a parrot-bot -- it echoes back everything the other user types.

```
// Assume a MessageListener we've setup with a chat.

    public void processMessage(Chat chat, Message message) {
        // Send back the same text the other user sent us.
        chat.sendMessage(message.getBody());
    }
```

Incoming Chat

When chats are prompted by another user, the setup is slightly different since you are receiving a chat message first. Instead of explicitly creating a chat to send messages, you need to register to handle newly created instances when the *ChatManager* creates them.

The *ChatManager* will already find a matching chat (by thread id) and if none exists,

then it will create a new one that does match. To get this new chat, you have to register to be notified when it happens. You can register a message listener to receive all future messages as part of this handler.

```
// Assume we've created a Connection name "connection".
ChatManager chatmanager = connection.getChatManager().addChatListener(
    new ChatManagerListener() {
        @Override
        public void chatCreated(Chat chat, boolean createdLocally)
        {
            if (!createdLocally)
                chat.addMessageListener(new MyNewMessageListener());;
        }
    });
```

Packet Properties

Smack provides a flexible framework for processing incoming packets using two constructs:

- *org.jivesoftware.smack.PacketCollector* -- a class that lets you synchronously wait for new packets.
- *org.jivesoftware.smack.PacketListener* -- an interface for asynchronously notifying you of incoming packets.

A packet listener is used for event style programming, while a packet collector has a result queue of packets that you can do polling and blocking operations on. So, a packet listener is useful when you want to take some action whenever a packet happens to come in, while a packet collector is useful when you want to wait for a specific packet to arrive. Packet collectors and listeners can be created using an Connection instance.

The *org.jivesoftware.smack.filter.PacketFilter* interface determines which specific packets will be delivered to a *PacketCollector* or *PacketListener*. Many pre-defined filters can be found in *theorg.jivesoftware.smack.filter* package.

The following code snippet demonstrates registering both a packet collector and a packet listener:

```
Message message = chat.createMessage();
// Add a Color object as a property.
message.setProperty("favoriteColor", new Color(0, 0, 255));
// Add an int as a property.
message.setProperty("favoriteNumber", 4);
chat.sendMessage(message);
```

Getting those same properties would use the following code:

sing objects as property values is a very powerful and easy way to exchange data. However, you should keep the following in mind:

- Packet extensions are the more standard way to add extra data to XMPP stanzas. Using properties may be more convenient in some cases, however, since Smack will do the work of handling the XML.
- When you send a Java object as a property, only clients running Java will be able to interpret the data. So, consider using a series of primitive values to transfer data instead.
- Objects sent as property values must implement Serialiable. Additionally, both the sender and receiver must have identical versions of the class, and a serialization exception will occur when de-serializing the object.
- Serialized objects can potentially be quite large, which will use more bandwidth and server resources.

XML Format

The current XML format used to send property data is not a standard, so will likely not be recognized by clients not using Smack. The XML looks like the following (comments added for clarity):

```
<!-- All properties are in a x block. -->
<properties xmlns="http://www.jivesoftware.com/xmlns/xmpp/properties">
    <!-- First, a property named "prop1" that's an integer. -->
    <property>
        <name>prop1</name>
        <value type="integer">123</value>
    <property>
    <!-- Next, a Java object that's been serialized and then converted
         from binary data to base-64 encoded text. -->
    <property>
        <name>blah2</name>
        <value type="java-object">adf612fna9nab</value>
    <property>
</properties>
```

Multi User Chat

Although Multi Chat is not implemented yet in our application, we will see how it works in order to create different games in the future.

Create a new Room

It allowed users may create new rooms. It is the procurement that the players will be following in order to create a game.

There are two types of rooms that you can create. Instant rooms which are available for immediate access and are automatically created based on some default configuration and reserved rooms which are manually configured by the room creator before anyone is allowed to enter.

In order to create a room you will need to first create an instance of *MultiUserChat*. The room name passed to the constructor will be the name of the room to create. The next step is to send create (String nickname) to the *MultiUserChat* instance where nickname is the nickname to use when joining the room.

Depending on the type of room that you want to create you will have to use different configuration forms. In order to create an Instant room just send *sendConfigurationForm*(Form form) where form is an empty form. But if you want to create a reserved room then you should first get the room's configuration form, complete the form and finally send it back to the server.

Example

```
// Create a MultiUserChat using a Connection for a room
      MultiUserChat muc = new MultiUserChat(conn1,
"myroom@conference.jabber.org");

      // Create the room
      muc.create("testbot");

      // Send an empty room configuration form which indicates that we want
      // an instant room
      muc.sendConfigurationForm(new Form(Form.TYPE_SUBMIT));
```

In this example we can see how to create a reserved room. The form is completed with default values:

```
  // Create a MultiUserChat using a Connection for a room
      MultiUserChat muc = new
      MultiUserChat(conn1,"myroom@conference.jabber.org");

      // Create the room
      muc.create("testbot");

      // Get the the room's configuration form
      Form form = muc.getConfigurationForm();
      // Create a new form to submit based on the original form
      Form submitForm = form.createAnswerForm();
      // Add default answers to the form to submit
      for (Iterator fields = form.getFields(); fields.hasNext();) {
          FormField field = (FormField) fields.next();
      if (!FormField.TYPE_HIDDEN.equals(field.getType()) &&
      field.getVariable() != null) {
              // Sets the default value as the answer
              submitForm.setDefaultAnswer(field.getVariable());
          }
      }
      // Sets the new owner of the room
      List owners = new ArrayList();
      owners.add("johndoe@jabber.org");
      submitForm.setAnswer("muc#roomconfig_roomowners", owners);
      // Send the completed form (with default values) to the server to
      configure the room
      muc.sendConfigurationForm(submitForm);
```

Join a room

This feature will allow us to join a game that has been created previously. Your usual first step in order to send messages to a room is to join the room. Multi User Chat

allows specifying several parameters while joining a room. Basically you can control the amount of history to receive after joining the room as well as provide your nickname within the room and a password if the room is password protected.

In order to join a room you will need to first create an instance of *MultiUserChat.* The room name passed to the constructor will be the name of the room to join. The next step is to send join(...) to the *MultiUserChat* instance. But first you will have to decide which join message to send. If you want to just join the room without a password and without specifying the amount of history to receive then you could use join(String nickname) where nickname if your nickname in the room. In case the room requires a password in order to join you could then use join (String nickname, String password).

And finally, the most complete way to join a room is to send join (String nickname, String password) where nickname is your nickname in the room, password is your password to join the room, history is an object that specifies the amount of history to receive and timeout is the milliseconds to wait for a response from the server.[iv]

In this example we can see how to join a room with a given nickname:

```
// Create a MultiUserChat using a Connection for a room
      MultiUserChat muc2 = new MultiUserChat(conn1,
"myroom@conference.jabber.org");

      // User2 joins the new room
      // The room service will decide the amount of history to send
      muc2.join("testbot2");
```

In this example we can see how to join a room with a given nickname and password:

```
// Create a MultiUserChat using a Connection for a room
      MultiUserChat muc2 = new MultiUserChat(conn1,
"myroom@conference.jabber.org");

      // User2 joins the new room using a password
      // The room service will decide the amount of history to send
      muc2.join("testbot2", "password");
```

In this example we can see how to join a room with a given nickname specifying the amount of history to receive:

```
// Create a MultiUserChat using a Connection for a room
      MultiUserChat muc2 = new MultiUserChat(conn1,
"myroom@conference.jabber.org");

      // User2 joins the new room using a password and specifying
      // the amount of history to receive. In this example we are
requesting the last 5 messages.
      DiscussionHistory history = new DiscussionHistory();
      history.setMaxStanzas(5);
      muc2.join("testbot2", "password", history,
SmackConfiguration.getPacketReplyTimeout());
```

CHAPTER 3: First Prototype

Introduction

In this chapter, I will describe the first Scrabble project that I did in order to test the XMPP protocol and also to learn how Android works. At first was the final application but then we thought better to work on the project that was done. So I took the things that worked property for this application and I implemented it in the original CityScrable project.

Architecture

The architecture that I programmed for this game was very similar to the last one:

App.java (Application Class): Same function as what we have already explained above: Store important data about the game, Intents and Broadcast Receivers.

Also, it processed the messages coming from the service and make changes in the game. The most important difference between the two is that I used an array to monitor changes in the associations that had been made. This is not a correct way to do this. You must use a class that stores all changes to topics made the new application.

Service.java (Service): Communications with the XMPP server. As this is the same code that was used later, I will explain this operation in the next chapter.

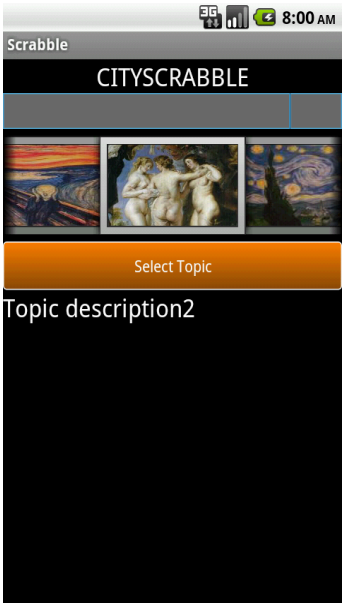


Figure 6. MainActivity

MainActivity.java (First Activity): Contains a gallery showing the Topics and a description of each item below it. We decided not import to the new project because it was not too attractive. Also, it did not show the flags with the colors of the players who had made a correct association.

SecondActivity.java (Second Activity): Shows a list of elements that has the topic that we chose in the previous screen. There is a button that allows the player to scan an exhibit

Figure 7. SecondActivity

XML structure

As I began to design the application from nothing, I created a new XML file than that we already had previously. In the picture we see as an exhibit and a hint were in the previous design:

```
<topic>
        <title>Topic7</title>
        <image>Img7</image>
        <description>Topic description7</description>
        <hint0 id="29" info="info7/hint0">topic7/hint0</hint0>
        <hint1 id="30" info="info7/hint1">topic7/hint1</hint1>
        <hint2 id="31" info="info7/hint2">topic7/hint2</hint2>
        <hint3 id="32" info="info7/hint3">topic7/hint3</hint3>
</topic>
<exhibit>
        <title>Exhibit0</title>
        <image>http://school.discoveryeducation.com/clipart/images/art-color.gif</image>
        <qr>QR00</qr>
        <description>Exhibit description0</description>
</exhibit>
    <exhibit>
        <title>Exhibit1</title>
        <image>Img1</image>
        <qr>QR01</qr>
        <description>Exhibit description1</description>
</exhibit>
```

We had to constantly call whenever we wanted to make changes on the game screen or display data by using the following lines:

InputStream is = getResources().openRawResource(R.raw.list);

It was a problem, and the only way to fix it was to create a java class that store all the information and send it to the application in order to be always available and not have to constantly check the xml file.

Scanning an Exhibit

In contradistinction to the final game, I integrated Barcode Scanner directly into the application through a project that served as a library. Therefore it was not necessary to ask the user to install it (if he had not done before) in order to launch the game. The scanner managed by attached project is called *Zxing* and all information is available in the Appendix.

The code that calls the scanner is as follows:

```
Intent intent = new Intent("com.google.zxing.client.android.SCAN");
intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
startActivityForResult(intent, 0);
```

As we can see is an Intent to the other project that works as a library assistant.

In order to collect the values we use the following method:

```
public void onActivityResult(int requestCode, int resultCode, Intent
intent) {
       if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
   contents = intent.getStringExtra("SCAN_RESULT");
   String format = intent.getStringExtra("SCAN_RESULT_FORMAT");
   // Handle successful scan
        } else if (resultCode == RESULT_CANCELED) {
   // Handle cancel
        }
}
```

Now we just have to search the value contents variable with the value of the all QR codes that we have in the xml file:

```
for (int k=0;k<exhibitlist.size();k++)
{
int result =
contents.compareToIgnoreCase(exhibitlist.get(k).getQr());
if (result == 0)
{
       urlImage = exhibitlist.get(k).getImage();
       exText.setText(exhibitlist.get(k).getDescripcion());
}
}
```

If we find the code, the application shows the description and the image of the exhibit and allows the player to select an association.

Processing incoming messages

Another part that I would like to explain my old application is processing messages coming from the server to the device:

```
public class ReceiverService  extends BroadcastReceiver {
      @Override
      public void onReceive(Context context, Intent intent) {

      String men = intent.getStringExtra("mensaje");
      if (men.indexOf("@")>0 )
      {
            String tokens[] = men.split("@");
            int player = Integer.parseInt (tokens[0]);
            int hint = Integer.parseInt (tokens[1]);
            Intent intentToA = new
            Intent(project.SecondActivity.MESSAGE_FROM_SERVER);
            switch (player) {
            case 0:
                  Toast.makeText
                  (application, "Wrong Association",
                  Toast.LENGTH_LONG).show();
                  break;
```

```
                case 1:
                        Toast.makeText
                        (application, "Correct Association!",
                        Toast.LENGTH_LONG).show();
                        Score = Score + 5;
                        ArrayHint[hint] = player;
                        intentToA.putExtra("score", Score);
                        intentToA.putExtra("ArrayHint", ArrayHint);
                        break;

                default:
                        Toast.makeText
                        (application, "Another player made a good
                        Association", Toast.LENGTH_LONG).show();
                        ArrayHint[hint] = player;
                        intentToA.putExtra("ArrayHint", ArrayHint);
                        break;
                }
                sendBroadcast(intentToA);
                }


        else {
                        Toast.makeText
                        (application, "Incorrect message from server",
                        Toast.LENGTH_LONG).show();
                }
        }
```

It was a Broadcast Receiver that picks up the message that arrives from the Service and depending on the player to make an association, through a Switch we perform the appropriate action and inform the player by Toast.

CHAPTER 4: Development and implementation

In this chapter, I am going to describe the most important details of the final application. I emphasize the changes I made in order to improve the previous game.

First of all, I will summarize the elements involved in the game, then I will explain how to play CityScrabble and finally I come in to explain technical details on the code in Java and XML.

Elements

There are three types of elements that perform an important role in the game and with which the player can interact using the interface. They are Topics, Keys and Exhibits.

Topics: There are eight main themes in the first screen that the player must choose in order to find the hints and score points. When a player selects one of them, he passes to the second screen that contains all the information and a list of keys about it. Exhibits also appear but these do not depend on selected topic because we will always have the same list of them (locked or unlocked) regardless of the topic we have chosen previously.

Keys: one of the changes that I did was rename the name of the old Hints. Now they are called Keys because we believe that this was more intuitive and correct because the real hint is not shown on these items, it is related with the exhibit that we must find in the scenario.

It ought to show to the player the information in a simple, intuitive and clear way, in order to provide that if the player just has a quick look at the screen, he can obtains all the necessary information.

Exhibits: They are the elements that we have to find in the scenario and unlock their using QR codes or GPS and then the application allows to make an association with each Key that the actual Topic has. I have modified the XML file and now we can see a hint «exhibit hint» that shows how to find it in rather than the accurate description is displayed only once unlocked.

How to play

The aim of the ScrabbleCity application is to engage groups of people to find a list of exhibits, following hints along a location or scenario. If the player finds an exhibit, first of all, he unlocks it and then he has to make an association with a key.

The location can be from a city to a museum, interacting through mobile devices. The people need to be divided in different groups (teams). Each team receives one name and one color that identify them.

The Scrabble game has two main screens that correspond to the two activities of the application, Scrabble and TopicActivity. Scrabble activity shows nine images with the topics about which the game will develop. In the second one shows all information about the chosen topic, as well as the different keys and exhibits to answer and the button to link them.

The player has to follow the next step in order to make a good association (Key – Exhibit):

1.-When the game starts, the player must be log into the game with his username and password because without this, he cannot send data to the game server.

2.-After that, the player has to interact with the topic selection screen. Each topic icon has four flags signaling availability of its keys. Here the game has just begun and the flags are empty; that means the topics are available for scoring points. Later this flags will show the keys which have been answered and the team which has answered them. When the player knows about which topic he wants to play, he has to push over the chosen image topic and the topic screen will be displayed.

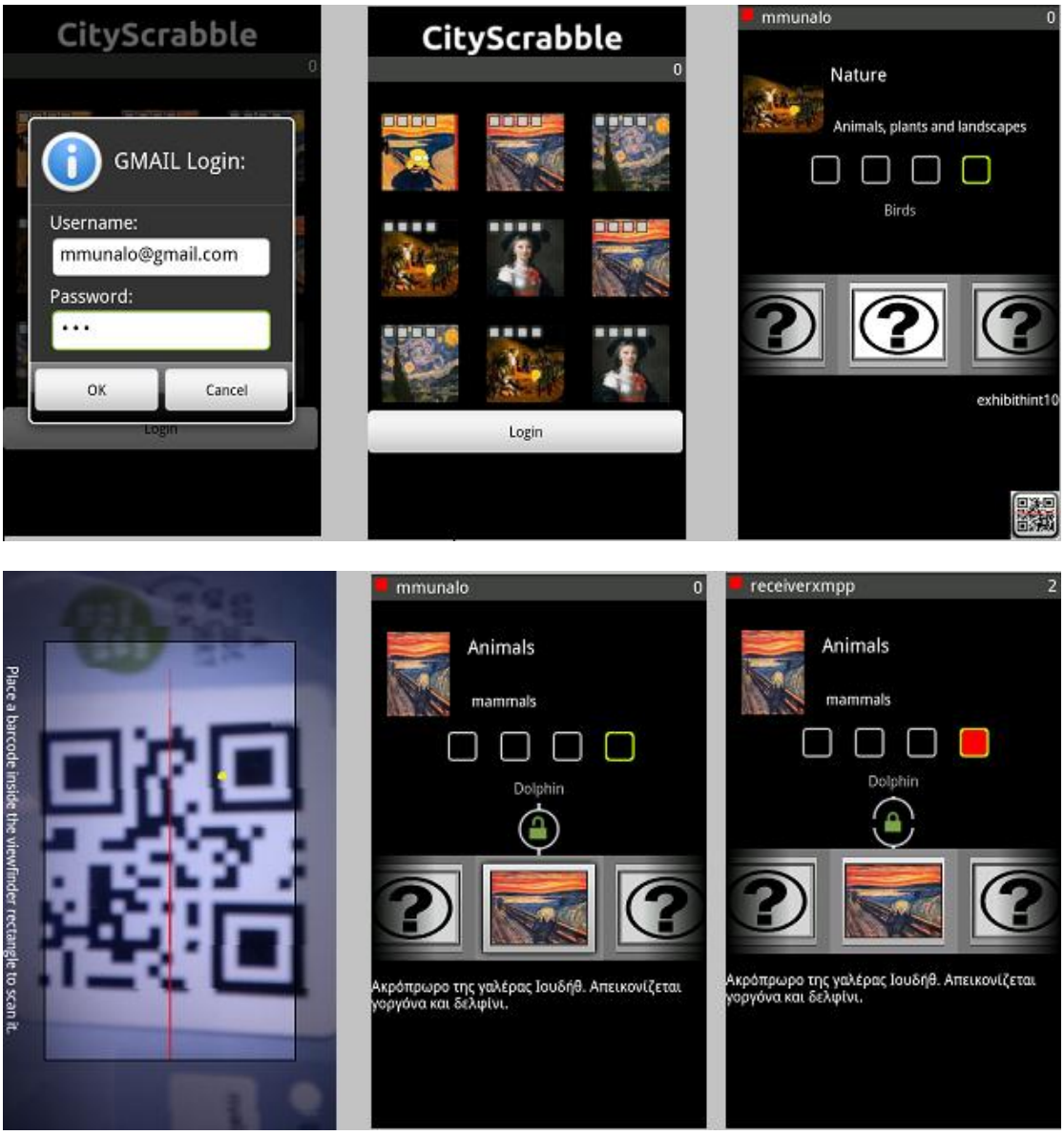
3. - As we said earlier the main goal of the game is to move around the scenario and find the exhibits in order to unlock them using various methods available. (QR codes, GPS).

4. - Once this is done, we have the exhibit available to be selected by an association with the key topic that we deem appropriate.

5. - Between the list of hints and the gallery of exhibits appears the link button. This button is used to link the key with the exhibit when the player thinks that the join is correct. The player can also strategically join a pair key-exhibit, doing that the pair is not available for other players, but with this the player will not get score. There will be a maximum number of incorrect answers that the player can have linked at the same time.

6. - The player will be able to unlink his pairs. To do this, he has to select the key, the exhibit and press on the link button. If the pair that the player has unlinked was

correct, the value of this hint will be subtracted of the score. But if the pair unlinked was incorrect, nothing is subtracted.



In these photos you can see all the points that I explained before:

1) The login screen,

2) The main screen with a list of topics,

3) The TopicActivity screen with all the exhibits locked

4) An exhibit has been unlocked

5) The Barcode Scanner ready to capture a QR code

6) Team red made a correct association and won 2 points.

When a player makes an association, the application is displayed a progress dialog until the server receives this association and send a request back to the player's device. This is a new feature that we made in order to solve problems that I will explain later.

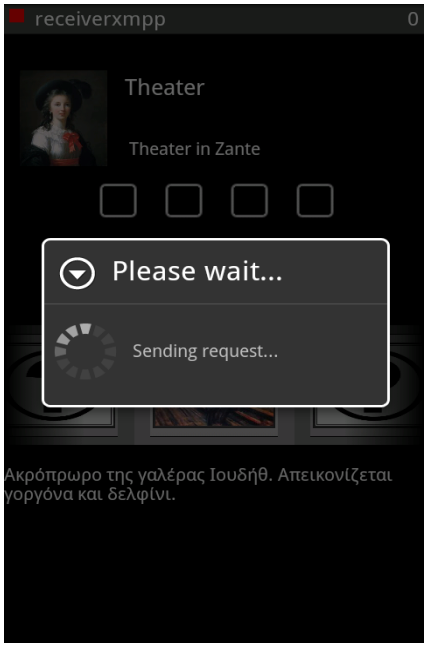


Figure8. Progress Dialog

If one key has been answered for other player, automatically the flag of the hint will be drawn with the color of the player and the link button will be disabled for the player.

The player can continue looking for exhibit matching at the current topic or at any time return to the topic selection screen.

When the player returns to the topic screen, it is visible that the other players have been also busy securing hints. This gives an overview of the other player´s activity, but doesn´t provide any information about their score. They might as well have created meaningless associations earning no points at all.

The flags of the topic selection screen are updated instantly when an association has been made. So, one can watch the screen and get an idea about the level of activity in each topic as the time elapses.

New Features

The most important feature is that we have introduced is that now the player has the risk of losing points if you make an incorrect association. Before it, he could make all the pairs that he never lost any point.

Another important improvement is that when we scanned an Exhibit, the result being sought throughout the list. Now only check the selected exhibit is equal to the scan, if not so it cannot be unlocked.

Flow Chart

The image below shows a flowchart of the most important processes following the implementation depending on the choice of the player, internet connection or if he has previously installed barcode scanner or not.

Orange boxes have been used to represent the screens, green boxes for processes that depend on the player's choice, orange square boxes are background application processes and blue boxes represent action consequences displayed generally by Toast:

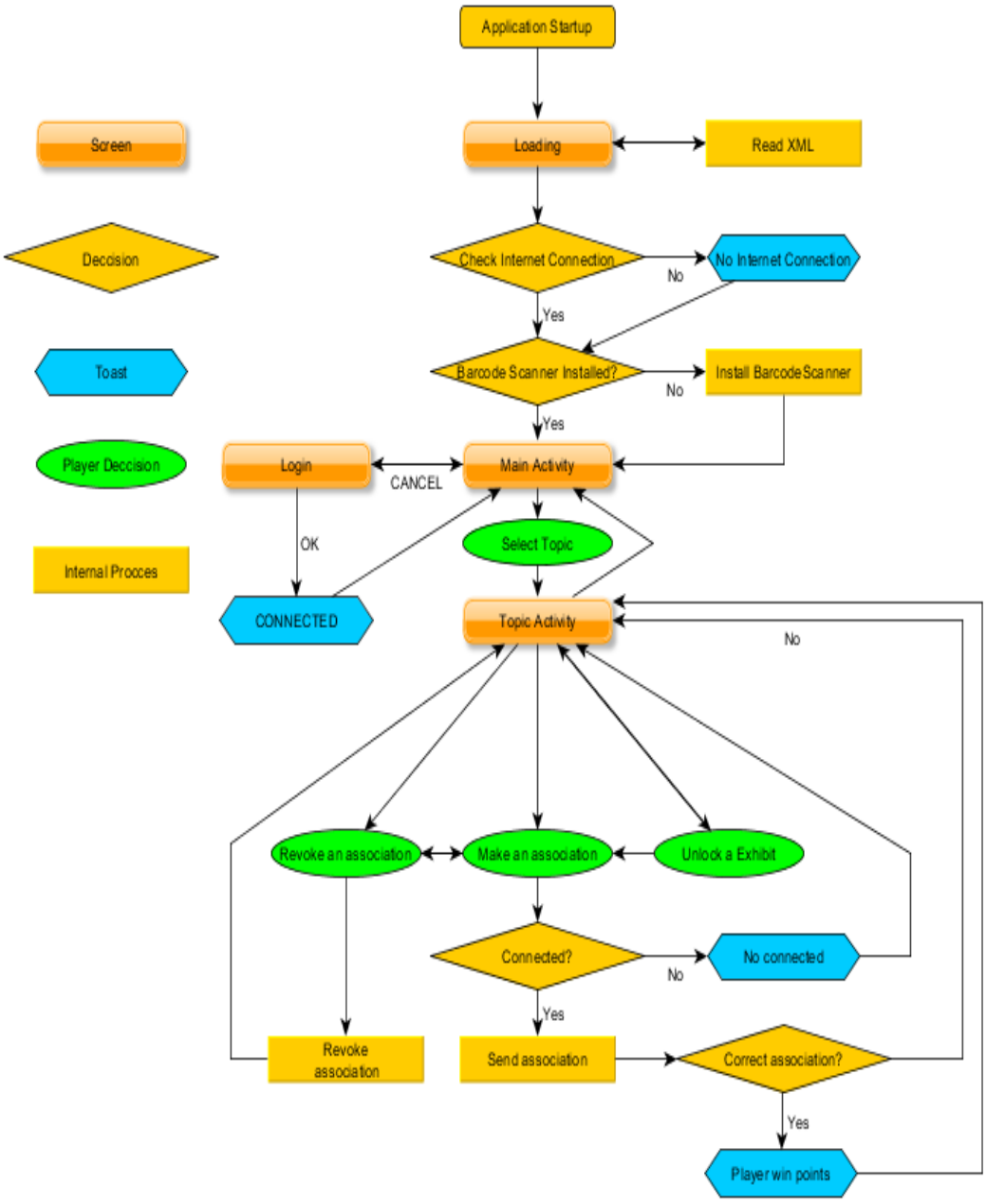


Figure 9. Application Flowchart

The Java Code

This project has been focused basically in the development and implementation of the application. Most of the time spent in the project has been used for this task. Due to this, it has been decided to present a detailed report of the features that have been designed and implemented in the application.

As the main aim was to improve the application, I will describe the most important parts of the code about I worked and I will not go into details of the parts that were already designed.

XMPP Service

The last REST Service has been replaced by XMPP Service. That means that the code has been modified in order to send and receive messages from XMPP server:

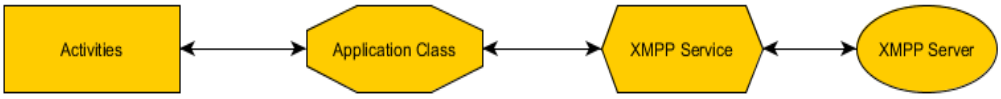


Figure 10. Communications Structure

First of all is to send a connection request to the server as explained in Chapter two. This is to use an Intent to launch the service from the main activity, and after the service attempts to connect. It depends on the username and passwords are correct and if we have internet connection available.

To do this, we need to call the Service from the first activity using an Intent and send the data directly there without going through the Application Class.

```
	protected void startXMLService() {
//		// Start XMPP Service
		Intent serviceIntent;
		serviceIntent = new Intent(context, XMPPClientService.class);
		serviceIntent.putExtra("userAccess", userAccess);
		serviceIntent.putExtra("passAccess", passAccess);
		serviceIntent.putExtra("myPlayerId", app.MyPlayerId);
		startService(serviceIntent);


	}
```

In CityScrabble v1 this method was called from the loading task, but as a Login screen has been integrated, now it has to be released after having the player their username and password.

Now we have everything that we need to connect to the server and wait for messages arriving from it using the code described in the Chapter 2.

Sending a message

When the played press the link button in order to make an association, inside the Button Listener there is a piece of code that check if the association is correct:

```
if (exhibitId == currentKey.getKeyExhibitId()) {
                    isCorrectPair = true;
}
```

That we can see, there are two variables that contain the exhibit Id and the key Id. The code checks if this value is the same and puts the Boolean in *true* status.

The Application Class will store all the data from the activities and it will send a message with the Service that contains the Player Id, Exhibit Id and Key Id:

```
private void sendAssociationToServer(int keyId, int exhibitId) {
      Intent intentNewAssociation = new
      Intent(MYPLAYER_HAS_MADE_A_NEW_ASSOCIATION);
      intentNewAssociation.putExtra("keyId", keyId);
      intentNewAssociation.putExtra("exhibitId", exhibitId);
      intentNewAssociation.putExtra("playerId", MyPlayerId);
      sendBroadcast(intentNewAssociation);
}
```

The way to do it is by sending a broadcast request and receiving the message through a Broadcast Receivers. It is a different way if we just want to exchange data between activities. All information of how this works can be found in the Index.

The following code shows a method in the Service that allows sending data to the XMPP server:

```
private void _sendToServer(int playerId, int keyId, int exhibitId) {

      String player = String.valueOf(playerId);
      String key = String.valueOf(keyId);
      String exhibit = String.valueOf(exhibitId);
      String message = ("S"+"@"+player+"@"+key+"@"+exhibit);
      try {
            newChat.sendMessage(message);
      } catch (XMPPException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
}
```

Before sending this, we must convert the integers that we received. The way to do this is with the method *valueOf* and sends using tokens (@) in order to put together in a String.

Emulating a server for debugging: Google Talk

Many platforms as Facebook and Google are using XMPP protocol in order for chat communications between users. As there is not a real server available to handle the messages, we can use one of these platforms like a Mechanical Turk. For more information about how this system works you can revise the Appendix.

In order to test (debugging) our application we were using Google Talk. It emulated the communications server responding all request coming from the devices.

Until now, we have managed to send a message from the Service to the XMPP Server containing information: Player Id, Key Id and Exhibit Id.

For example, the player with Id = 86 selects key Id =84 with exhibit Id =71.

The *google.talk* account we are using as the Turk receives the following:

S@86@84@71

At that moment, we use the chat in order to send a broadcast message to all the players that are playing at the same moment. (We see all the players connected to the chat room)

Now, everybody knows that player Id= 86 made an association (Key-Exhibit) and proceeds to change the color of the flag. If it is necessary, score is also update (correct association).

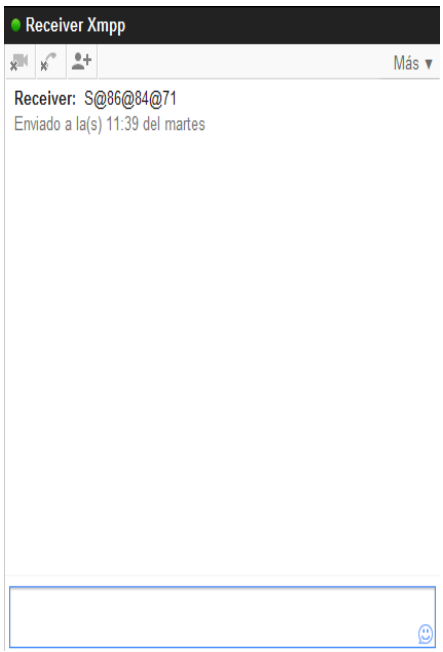


Figure 11.Google Talk

If the player wants to revoke an association, the procedure is similar.

R@86@84@71 (player Id=86 wants to revoke Key Id = 84 and exhibit Id = 71)

As we have seen the server distinguishes between an association(S) and a revoke (R), depending on the type of message.

Processing a message from the server

Messages from the server arrive at the Service that sends to the Application Class where they are processed to make changes to the game: change flags color or update the score if the association was correct. Broadcast Receiver collected the message. It was a String, so we will need to convert it back to Integer:

```
private class NewGameUpdateEventReceiver extends BroadcastReceiver {
      @Override
      public void onReceive(Context context, Intent intent) {

            String type = intent.getStringExtra("type");
            int keyId = intent.getIntExtra("keyId", -1);
            int exhibitId = intent.getIntExtra("exhibitId", -1);
            int playerId = intent.getIntExtra("playerId", -1);
```

Below it is a comparison to see if we have a message such association or a revoke message:

```
            int result = type.compareToIgnoreCase("S");
```

It checks the ID of player who has made the association and if the player that owns the device was trying to do the same, informing the player that another player has the same association before him.

```
            if (result == 0)
            {
            if(playerId != MyPlayerId && keyId == mykey &&
            exhibitId == myexhibit )
                  {
            Toast.makeText(getApplicationContext(),
            "Another player has made the same association before
            you", Toast.LENGTH_LONG).show();
            }
```

If that does not happen, is because the association is correct and it will update the score adding points. Depending on the type of association that the player made, he will win more or fewer points. This is determined by the value of variables *newScore* and *newScore2*:

```
            if(newScore == true && playerId == MyPlayerId)
            {
            Score = (Score) + keyValue;
            Toast.makeText(getApplicationContext(),
            "Correct association"
            Toast.LENGTH_LONG).show();
            Toast.makeText(getApplicationContext(),"You win "
            +keyValue+ "points", Toast.LENGTH_LONG).show();
            newScore = false;
            }
            if (newScore2 == true && playerId == MyPlayerId)
            {
            Score = (Score) + keyValue2;
            Toast.makeText(getApplicationContext(),"Correct
            association!", Toast.LENGTH_LONG).show();
            Toast.makeText(getApplicationContext(),"You earn "
            +keyValue2+ "points", Toast.LENGTH_LONG).show();
            newScore2 = false;
```

At this point we need to update the list of topics to change the color of the flag corresponding to each player:

```
                    topicsList.updateAssociation( playerId,
                    playerList.getPlayerByPlayerId(
                    playerId).getColor(), keyId, exhibitId);
                    }
```

If the message is of type Revoke, it will remove the color of the flag depending on the player that requested it:

```
                    else
                    {
                    topicsList.removeAssociation(keyId);
                    if (playerId == MyPlayerId )
                    {
                    Toast.makeText(getApplicationContext(),"Revoked
                    asscociation", Toast.LENGTH_LONG).show();
                    }
                    else
                    {
                    Toast.makeText(getApplicationContext(),"Another played
                    revoked an asscociation", Toast.LENGTH_LONG).show();
                    }
```

Finally there is only send a broadcast to the activities in order to update the screens:

```
                    }
                    Intent intentNewAssociationReceivedFromServer = new
                    Intent(OTHERPLAYER_HAS_MADE_A_NEW_ASSOCIATION);
                    AssociationReceivedFromServer.putExtra("playerId",
                    playerId);
                    sendBroadcast(intentNewAssociationReceivedFromServer);
                    }
                }
        }
```

Login

In order to connect to communications server, each player must enter your username and password through a Dialog that we designed to perform this function. When the player presses the login button the application automatically picks up the Android operating system user name of the main account that is synchronized to the device. This function is performed using the following code:

```
public String get_google_account(){
  Account[] accounts =
  AccountManager.get(this).getAccountsByType("com.google");

  for (Account account : accounts) {
    String possibleEmail = account.name;
    return possibleEmail;
  }
return null;
```

It is a method that returns a String with the player's email address that will be assigned to him in the game. If he does not want to play with this user name he has the option to modify it through a Text View.

Dialogs

In the application, there are two types of dialogs: a Prompt Dialog which collects the user name and password and a Progress Dialog that is launched when a player makes an association and waits a response from the server before disappearing. Information about how to operate the dialogs is available in the Appendix.

Prompt Dialog

As already it is explained above, the Prompt Dialog has been used as a login screen asking for the email and password. By default the system collects the primary email account that is already installed on Android operating system. However, for privacy and security, android will never leave the password to access the device but if we can use the email in our applications. We just have to add a permission line in Android Manifest file:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

For this we need to schedule a layout that contains two Text View elements (username and password) and two Buttons (Ok and Cancel). The layout has a standard design but we will show the Java's code about this dialogue in order to explain it:

First, the code calls to *alert_dialog_text_entry.xml* the XML file that contains everything we have said previously.

```
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case DIALOG_TEXT_ENTRY:
      LayoutInflater factory = LayoutInflater.from(this);
       final View textEntryView =
       factory.inflate(R.layout.alert_dialog_text_entry, null);
       BoxUser      = (EditText)textEntryView.findViewById(R.id.nomUsuario);
       BoxPass      =
       (EditText)textEntryView.findViewById(R.id.passUsuario);
        BoxUser.setText(get_google_account());
         return new AlertDialog.Builder(Scrabble.this)
       .setIcon(R.drawable.info)
       .setTitle(R.string.alert_dialog_text_entry)
       .setView(textEntryView)
       .setPositiveButton(R.string.alert_dialog_ok, new
       DialogInterface.OnClickListener() {
```

It collects the username and password from the Text Views:

```
public void onClick(DialogInterface dialog, int whichButton) {

                    try {
                    userAccess = BoxUser.getText().toString();
                    passAccess   = BoxPass.getText().toString();
```

The following piece of code split the username into the player's email:

```
String tokens[] = userAccess.split("@");
app.ScorebarName = tokens[0];
```

With all of this data, the XMPP service will be lunched and the Login button will be removed in the screen. Automatically if the username and password are incorrect, the Service will send a broadcast to the Scrabble Activity and dialogue will be launched again.

```
startXMLService();
login.setVisibility(View.INVISIBLE);
      } catch (Exception e) {
            String menssage = e.getMessage();
            showMenssage("ERROR: " + message);
      }
}
})
```

If the user presses the cancel button, it will display a message through toast indicating that you must register before start playing:

```
.setNegativeButton(R.string.alert_dialog_cancel, new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int
whichButton)
{
String txtCancel = "You must to login if you want to
play";
showMessage(txtCancel);
login.setVisibility(View.VISIBLE);
}
})
.create();
}
return null;
```

Progress Dialog

The idea of integrating a Progress Dialog in the application was because when a player sends the server an association, perhaps the message will be lost and it never reaches the server. So it has established a maximum time for the Progress Dialog and if this timer expires the system displays a message using a toast that tells the player that the message is lost.

The other reason why we implemented this is if because two players make the same association at the same time. Then the system has to decide which of the two messages reach before the server. The first made the association and the second to arrive will be notified by Toast that I made the association after his opponent.

Using a Handle, we can show the Dialog's message:

```
private void runDialog(final int seconds)
	{
			final Handler handle = new Handler();
		progressDialog = ProgressDialog.show(this, "Please wait...",
	"Sending request...");
```

Then the task is started until the timer is equal to the integer value of the variable seconds, and if the Progress Dialog has not been stopped in another part of the code through progressDialog.dismiss(),it will throw two Toasts that inform the player that his association is lost in the way to the server.

```
		thread = new Thread(new Runnable(){
			public void run(){
				try {
					Thread.sleep(seconds * 1000);
					handle.post(proceso);
					progressDialog.dismiss();
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		 });thread.start();
	}
	final Runnable proceso = new Runnable(){
		public void run(){
			Toast.makeText(TopicActivity.this,"Request lost.",
			Toast.LENGTH_SHORT).show();
			Toast.makeText(TopicActivity.this,"Check your Internet
			connection",
			Toast.LENGTH_SHORT).show();
		}
	};
```

Toast

As can be seen in the Appendix, Toast shows instant messages that appear on screen to inform the player about changes in the game.

The idea of introducing this in the application emerged because in the CityScrabble v1 when making changes such as for example that an opponent made an association is important that all players that are playing CityScrable at the same moment know it.

Toast is a simple code that can easily show what we want:

```
Toast.makeText(getApplicationContext(),"Message here",
		Toast.LENGTH_LONG).show();
```

The parameter getApplicationContext() is the context of the application and allows displaying the message regardless of which activity (It is shown throughout the application)

Toast has been used to inform all players when they made a correct or wrong association, an association has been revoked or the player won points.

New Toggle Button

As it stated in the section that we talked about the interface, it has changed the button layout to make it more intuitive for the players.

1. The player locks the locked, and four options can occur:

    - The join is correct and the pair hasn`t been answered before. We lock the lock for the player, and the lock must be disabled for the other players. The flag is drawn with the color of the player, and finally the score is added.
    - The join is correct but the pair has been answered before for the player. The lock will be open, the score will be subtracted and the color of the flag will be transparent. Also the pair will be available for the other player. Now the state is open. We can see that the lock is green that inform better to the player that it is available.
    - The join is incorrect and the hint hasn`t been answered before for the player. The lock must be closed and disabled for the other players. Also the flag will be drawn.
    - The join is incorrect and the hint has been answered before for the player. The lock must be open and the color of the flag will be transparent. Also the pair will be available for the other player. Now the state is open.

If we want to change the design of the lock we have to modify the file *toggle.xml* contended in the drawable folder of the project:

```
<item
    android:state_checked="true"
    android:state_focused="true"
    android:drawable="@drawable/lock_closed_green" />
<item
    android:state_checked="true"
    android:state_focused="false"
    android:drawable="@drawable/lock_closed_green" />
<item
    android:state_checked="false"
    android:state_pressed="true"
    android:drawable="@drawable/lock_open_green" />
<item
    android:state_checked="false"
    android:state_focused="true"
    android:drawable="@drawable/lock_open_green" />
```

Figure 12. toggle.xml

As always, this file will be controlled using Java so we can change the button state to our choice.

The XML Code

To build an Android application we have to program in Java and XML and both programming languages have to work together. When we are planning will often see inconsistencies between the two languages so it is very important to design the application knowing well the two languages in order to avoid errors.

XML is a language very similar to HTML. Android use it to program the Android Manifest file, layouts or to work with parsers.

The Android Manifest file registers the Application Class, Services and Activities that make up the overall application. It also contains the permission lines and the minimum version of Android that is needed to launch the application.

As we shall know, it also used to create layouts that make the user interface and graphic design.

I would like to emphasize in the parsers. In our particular application, they are very important because they collect information from the XML file that contains all the information about the game (topics, exhibits and keys), and transfers them to the application through Java in order to have them always available.

There have been no changes in the method of parsing the data, but we have made changes to fix a bug that it did not let us play Cityscrabble correctly. We will discuss this bug is in the next section.

Changes in the XML code

As already mentioned above, it has changed the element name Hint to Key in the Java's code and in the XML's code as well. The exhibits have been adapted in order to support the GPS tracking feature in upcoming updates.

The following screenshots will appreciate the changes in the XML between CityScrabble v1 and the new one:

```
<WoA id="34">
    <title>Military</title>
    <description>Military</description>
    <image>military.jpg</image>
    <keys>
        <key key_id="77" exhibit_id="272" value="2" exhibit_id2="0" value2="0">
            <key_text>Red uniform</key_text>

        </key>
        <key key_id="78" exhibit_id="108" value="1" exhibit_id2="265" value2="1">
            <key_text>Στολή με παράσημα</key_text>

        </key>
        <key key_id="80" exhibit_id="270" value="2" exhibit_id2="0" value2="0">
            <key_text>Ορκομωσία οπλαρχηγού</key_text>

        </key>
        <key key_id="79" exhibit_id="142" value="2" exhibit_id2="0" value2="0">
</WoA>
```

Figure 13. Key XML Design

As we can see, in the previous version of the file we had five Hint's tags and in the new one there are four. Previously the second Hint was repeated and that caused problems. This change solved the bug discussed below. To collect this we must also make changes to the java code but I have not considered necessary to mention it.

Below are shown the changes into the Exhibit's tags where we added two new tags: *Collecttype* and *Collectdata*:

- **Collectype** : Contains information about what kind of exhibit is(QR or GPS)
- **Collectdata**: Data that will be used to unlock the Exhibit. In the case of a QR exhibit, this contains a QR code. In the case of GPS type it contains the coordinates of the exhibit.

It has also been added *Exhibithint* tag that show a hint to the player before unlock it.

```
<Exhibit id="166">
    <title>The lion of Venice</title>
    <exhibithint>exhibithint15</exhibithint>
    <description>ο ανάγλυφο λιοντάρι, κατασκευασμένο από πέ
    <collecttype>qr</collecttype>
    <collectdata>84B1E4E2</collectdata>
        <alternates>
            <alternate woaId="38">ο ανάγλυφο λιοντάρι, κατα
        </alternates>
    <image>cuadro1.jpg</image>
</Exhibit>
```

Figure 14. Exhibit new design

```
</Exhibit>
<Exhibit id="71">
    <title>The head prow of the "Judith" galleon</title>
    <description>Ακρόπρωρο της γαλέρας Ιουδήθ. Απεικονίζεται γο
    <rfidserial>84B17E42</rfidserial>
        <alternates>
            <alternate woaId="38">Ακρόπρωρο της γαλέρας Ιουδήθ.
        </alternates>
    <image>0a57cb53ba59c46fc4b692527a38a87c78d84028.jpg</image>
</Exhibit>
```

Figure 15. Exhibit old design

Problems fixed

In this section, we will describe all the bugs that we fixed in order to improve the application and with the final objective of program an application that will work property. Errors were found in the many procedures. There was one in the communication process when two players tried to do the same association at the same moment, another when the player wants to unblock a specific exhibit, another one with the scores that does not worked property before and the most important we that we fixed about the Keys because we cannot play the game that we will explain later.

Double association problem

As it explained before, we used a Progress Dialog to inform the player what is happening with the association that he just sent to the server. The real XMPP servers are fast and can process a response within a few milliseconds. But be can also have problems because our Internet connection is bad and slowly and It is important to inform the player the status of his message: If the server has been correctly received or not.

To do this we will launch a dialogue at the time at the same time we sent the message. When the player made an association, the locker status will change the status (opened to closed), the method *associateKeyToExhibit* will be called, and also the Progress Dialog will be launch:

```
//The player made an association...
linkButton.setChecked(true);
associateKeyToExhibit(currentTopicId, exhibitId,
currentKey.getkeyId(), isCorrectPair, isCorrectPair2);
runDialog(Xs);
```

When a broadcast-type message that comes back from the server arrives to the Activities, it is the time to stop the Progress Dialog:

```
public void onReceive(Context context, Intent intent) {

      topicKeys = topic.getTopicKeys();
      keysView.updateKeyList(topicKeys, app.MyPlayerId);
      if (intent.getIntExtra("playerId", -1) !=
      app.MyPlayerId)
      {
      setLinkButtonState(LinkButtonStates.INVISIBLE);
      }



      if (scoreBar != null && progressDialog !=null &&
      thread != null)
      {
      scoreBar.updateScore(app.Score);
      progressDialog.dismiss();
```

```
                thread.interrupt();
```

If dialogue is not interrupted for a period of time, runs its task and inform the player that his request has been lost.

Unlocking a single Exhibit using QR codes

The application was programmed such if the player wanted to unlock an Exhibit using the scanner, the application does not distinguished between them and the exhibit did not care that we selected before using the scanner because if any of the exhibits from our list coincided with the code we were scanning is unlocked. This is not correct, because the player can only unlock the exhibit that is currently selected.

In order to solve it we made a comparison between the current exhibit and the name of the content that the QR code contains:

```
if (app.exhibitList.isCodeExists(contents)) {
      //Here we make a comparation between currentExhibit and the
      collecdata tag
      if(contents.compareToIgnoreCase
      (currentExhibit.getcollectdata()) == 0)
      {
      app.exhibitList.markExhibitAsFound(contents);
      exhibitGalleryAdapter =
      new ExhibitGalleryAdapter(this, app.exhibitList);
      exhibitGallery.setAdapter(exhibitGalleryAdapter);
```

If the comparison of the Strings was correct we will inform the player that the Exhibit has been found:

```
      Toast.makeText(TopicActivity.this,"Exhibit found",
      Toast.LENGTH_SHORT).show();
      checkLinkButton(exhibitId, 1);
      exhibitGalleryAdapter.notifyDataSetChanged();
      }
```

In the other case, a Toast show the following:

```
      else
      {
      Toast.makeText(TopicActivity.this,"This code does not match this
      exhibit", Toast.LENGTH_SHORT).show();
      }
```

Not internet connection problem

The elements about the game are collect from Internet. We have to put all the data in an external folder, because later maybe we need to modify our choice without having to make internal changes in the application. It is also very important to have access to the Internet to exchange messages with the server.

When we do not have Internet access, the program collects all data from *raw* folder that is installed with the application. It will download all the game elements (hints, keys, descriptions) from this XML file but not the images that will always need the Internet to display it.

The piece code then checks if we have internet connection is the following:

```
public static boolean checkConex(Context ctx) {
	boolean AvaliableConnection = false;
	ConnectivityManager connec = (ConnectivityManager)
	ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
	// Available networks in the device (wifi, gprs...)
	NetworkInfo[] networks = connec.getAllNetworkInfo();
	for (int i = 0; i < 2; i++) {
	if (networks[i].getState() == NetworkInfo.State.CONNECTED) {
		AvaliableConnection = true;
		}
	}
	if (!AvaliableConnection) {
		msgError =
		ctx.getString(R.string.No_Internet_connection);
		Toast.makeText(ctx, msgError, Toast.LENGTH_LONG).show();
	}
	return AvaliableConnection;
}
```

This will be check if there are networks available (wifi,gprs...). If it is available the data will be collect from Internet, if we have the Wi-Fi turn of, it will take from the raw file into the application.

The problem is when we have networks available but we are not connected in this moment. If this happen, the application will launch a fatal error and it will never start.

Scores

When I started to improve the application, the score method did not work property because we updated the score before the association arrives to the server. It is not correct because if we have a problem with the communication, the message will never be received in the server. The correct method is update the score after receive it in the device.

If the Id of the player is equal at the message that we received and the Boolean *newScore* has been change to true because the pair was correct, the following will happen:

```
if(newScore == true && playerId == MyPlayerId)
{
Score = (Score) + keyValue;
Toast.makeText(getApplicationContext(),"Correct association!",
Toast.LENGTH_LONG).show();
Toast.makeText(getApplicationContext(),"You win " +keyValue+
"points", Toast.LENGTH_LONG).show();
newScore = false;
}
```

If we made a false association will lose points:

```
if(newScore == false && newScore2 == false && MyPlayerId == playerId)
{
Score = (Score) - keyValue;
Toast.makeText(getApplicationContext(),"Wrong
association!",Toast.LENGTH_LONG).show();
Toast.makeText(getApplicationContext(),"Sorry.You lost  " +keyValue+
"points", Toast.LENGTH_LONG).show();
}
```

Green squares bug

It is the biggest problem we found in Diego Rodriguez' version and it not let us play correctly. In order to solve it we changed the *scrabble.xml* file and a part of the Java's code. As we discussed in the section about XML in the previous chapter, in some Topics we had more than four tag because in some of them, more than one Exhibit may be associated with the same key, and there was a problem with the parser trying to collect the Key's data into the tags. So, we removed it and changed the format and now we have the following:

<key key_id="59" exhibit_id="230" value="5" exhibit_id2="223" value2="3">

As we can see, now one tag contains the two exhibits with which the player can make a correct pair.

About the Java's code we changed the following in *TopicActivity.java:*

```
if (exhibitId == currentKey.getKeyExhibitId()) {
      isCorrectPair = true;
}
else if (exhibitId == currentKey.getKeyExhibitId2()) {
      isCorrectPair2 = true
}
else {
      isCorrectPair = false;
      isCorrectPair2 = false;
```

The code make a comparison between the Exhibit Id and the current Key Id, if one of them corresponds to the key ID of a Boolean variable will be *true* v and the system now know that we made a correct pair.

CHAPTER 5: Evaluation Session

Before starting

The evaluation session was done on July the 6th of 2012 at the HCI laboratory of the University of Patras. It included a survey that contains two kinds of questions: general questions about Android, applications and videogames and another group about CityScrabble. In the last point of the survey we asked to the responders their opinion about future improvements.

The survey results generally have been successful and people have positively assessed the application and without being influenced by the rest because there were varieties of different answers.

The number of surveys was finally 10 and they were from different countries (Spain, Greece, Turkey and Pakistan) and the age range were between 21 and 29.

Before to test the application at the HCI laboratory, the *scrabble.xml* file that contains all the information about the game was changed because CityScrabble v1 was in Greek and the responders were from many countries. So, all the information was changed to English.

The prototype that was used contained Topics about nine cities in Europe (Athens, Thessaloniki, Patras, Valladolid, Madrid, Barcelona, London, Paris and Rome). In each city there were four Keys about famous places, monuments, squares, rivers that we can find there and finally a list of forty Exhibits with images, hints and descriptions in order that the player could make pairs. It is not the correct usage that should be given to the game in the future, that meant that the game must contain information about only a city and the player has to move with the hints looking Exhibits, unlocking them and making correct associations. But as the only test we were going to perform in the laboratory we seemed wise to make this series of changes to make the proper application to respondents.

First of all, it was a demonstration of how to unlock Exhibit through QR codes for all attendees to the evaluation. After this it made a small change in the program and all the Exhibits were unlocked from the beginning in order to make the game faster because they were not in a real scenario and players had not to find items along a city.

As the real server is not ready, we prepared Google Talk for the exchange of messages between devices respondents, reporting that everything was transparent to the player and only they had to evaluate the response of the mobile application.

The devices used were HTC desire. All information about them can be seen in the appendix. The respondents formed groups of two or three people at most and the game was scheduled for the first team to reach 20 points was the winner.

Below are the instructions to supply each user before start playing.

Instructions

1. - Launch the game: Scrabble

2. - Wait while the application loads the game data

3. - Login in the game using one of these accounts:

- receiverxmpp@gmail.com
- receiverxmpp2@gmail.com

Both have the password: *kalimera7*

4. - Select one of the Topics that are cities.

5. - Find one of the Exhibits below and try to make a correct association with the Keys that each city has using the lock. If the pair was correct you will win points.

Note: If you made a bad association you can revoke it pressing the Key that was wrong and then touching the lock.

Survey

General Questions

1. Have you got an Android device? What model??

_____

_____

2. Have you ever used applications like this? What kind?

_____

_____

3. Do you like to play videogames? Where? When?

_____

_____

4. When you're playing a game. Do you prefer a single player game or multiplayer?

_____

_____

5. Do you know about QR codes?

_____

_____

Questions about the game

	1=Poor	2=Fair	3=Good	4=Excellent	5=Not sure
The Game, in general					
The user Interface					
Pictures in the gallery					
The descriptions and hints					
Ease of play					
Application response					
QR Scanner response					
The locker button					
The login					
Scan button					

Improvement

We would like to know your opinion about we have to improve in the future:

_____

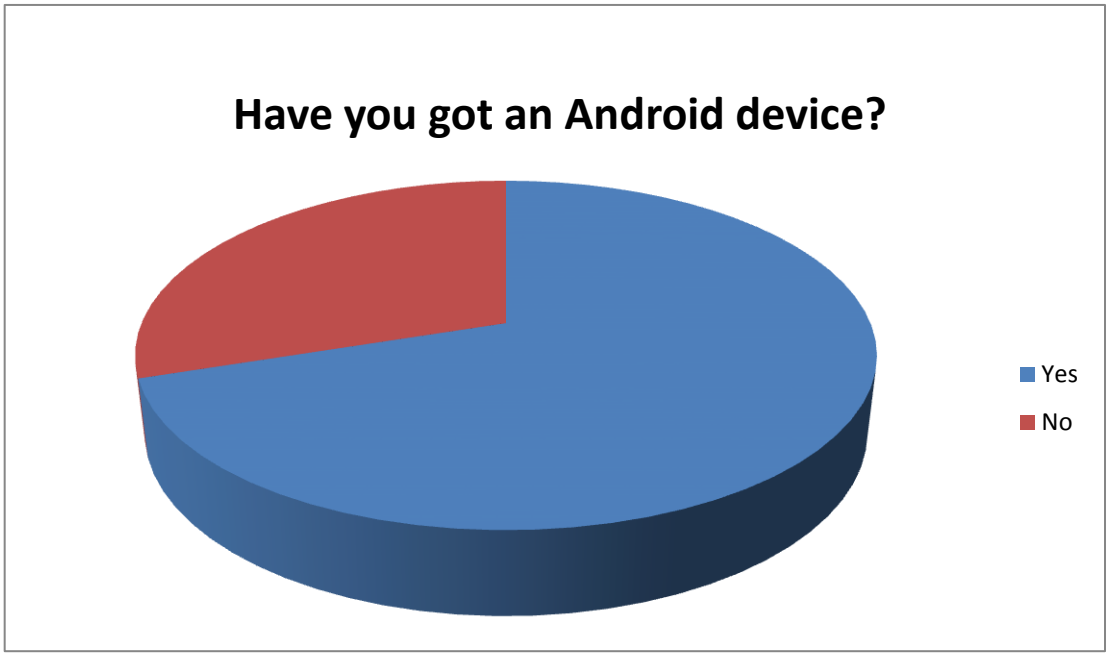
_____

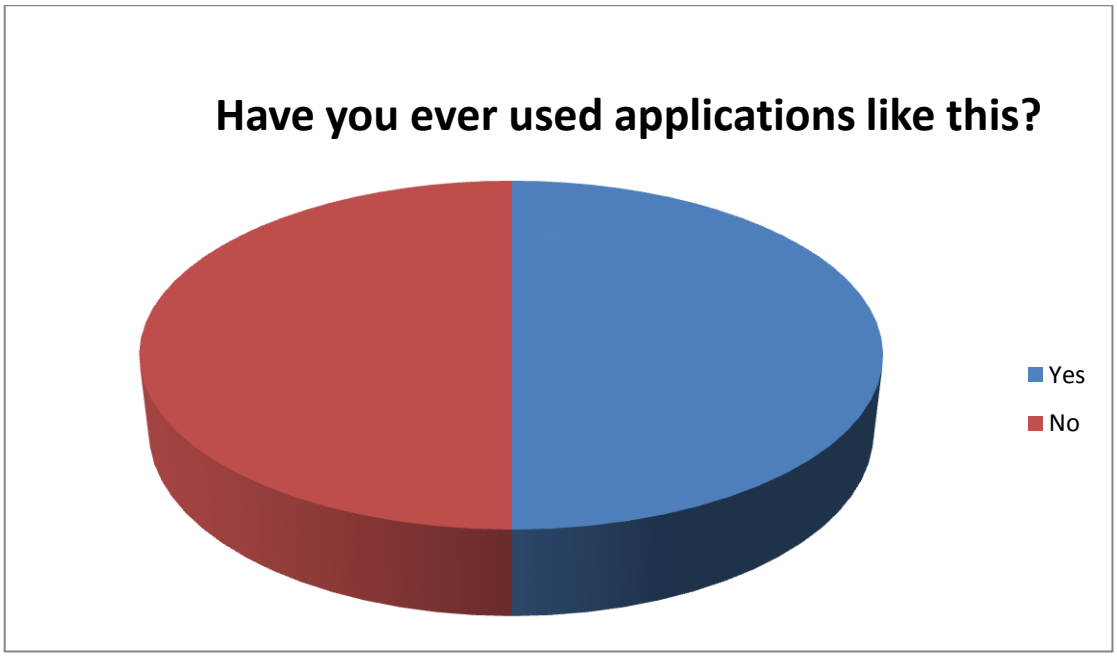
_____

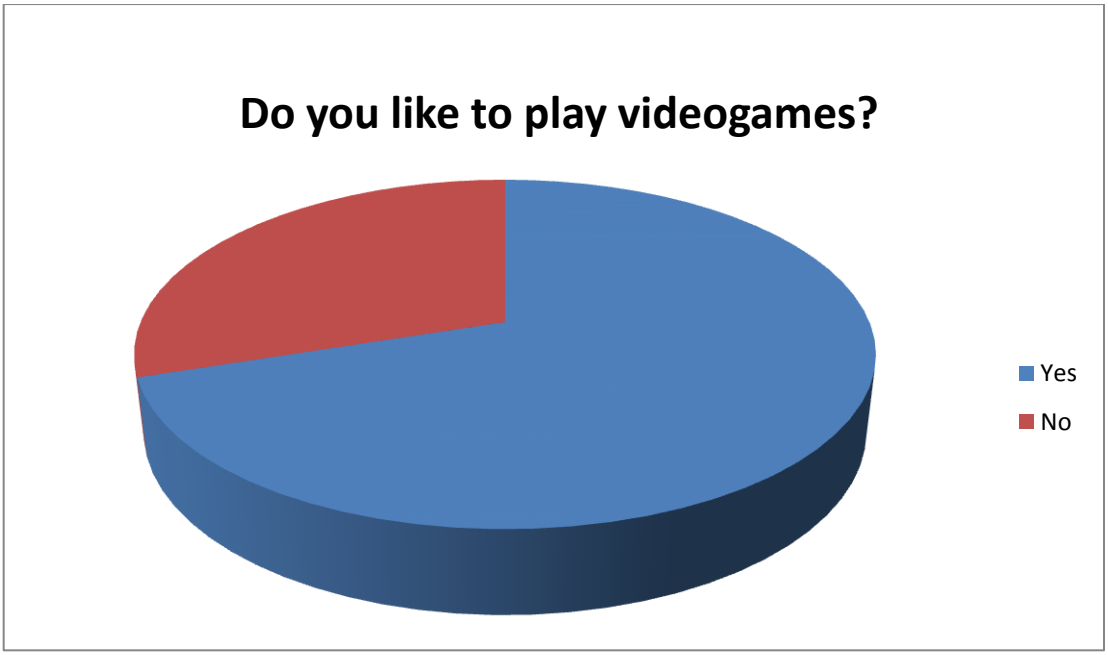
_____

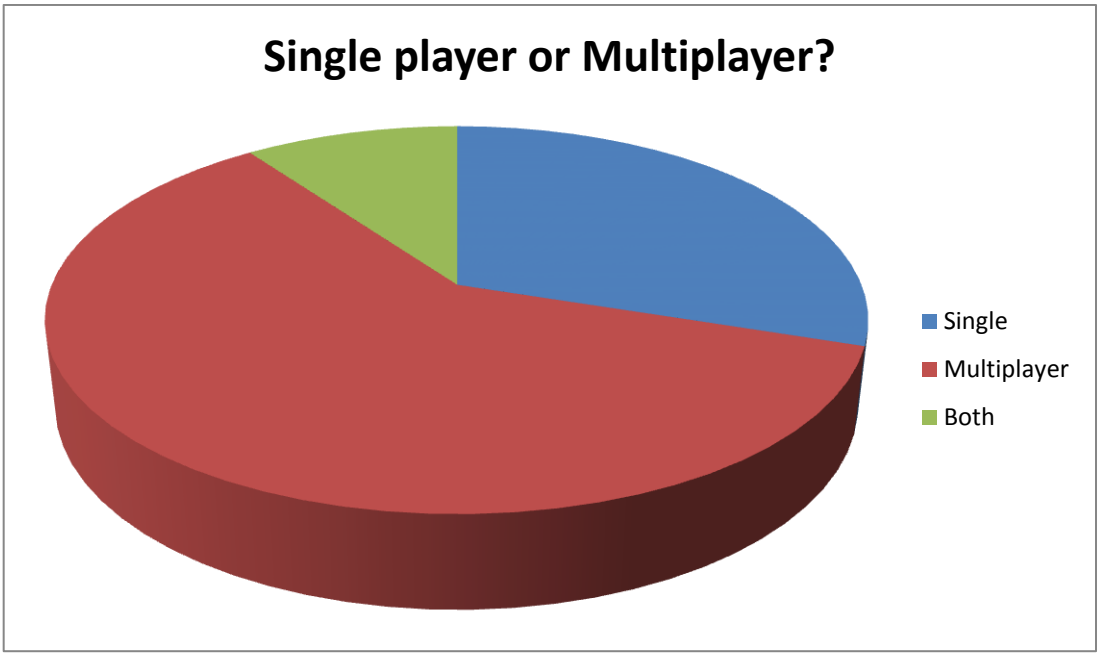
The results

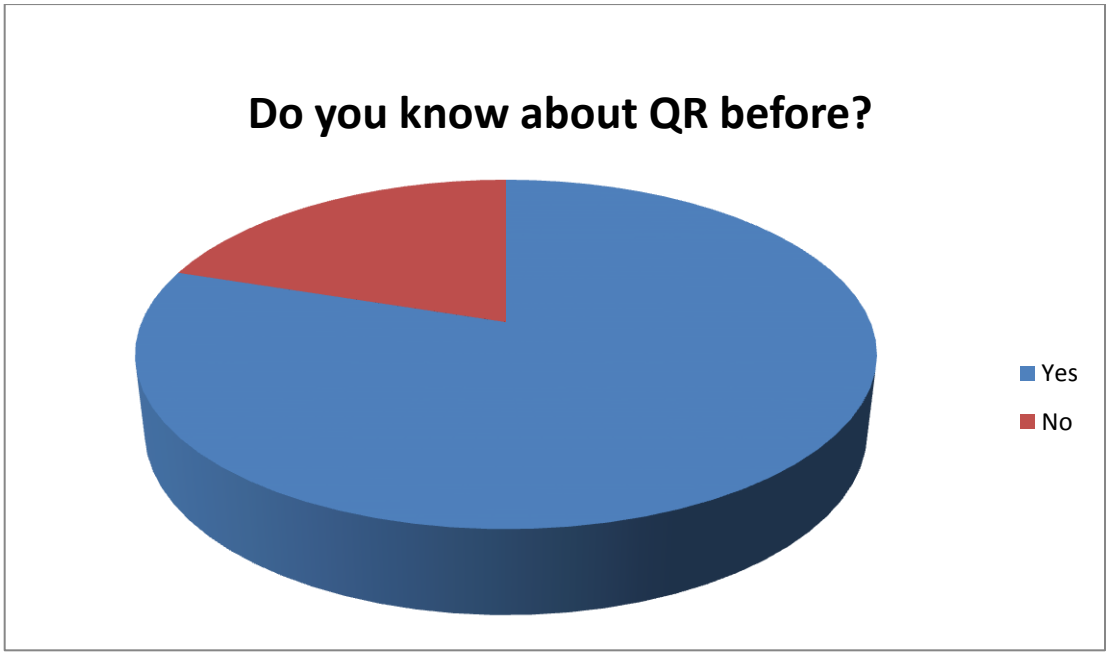
General Questions





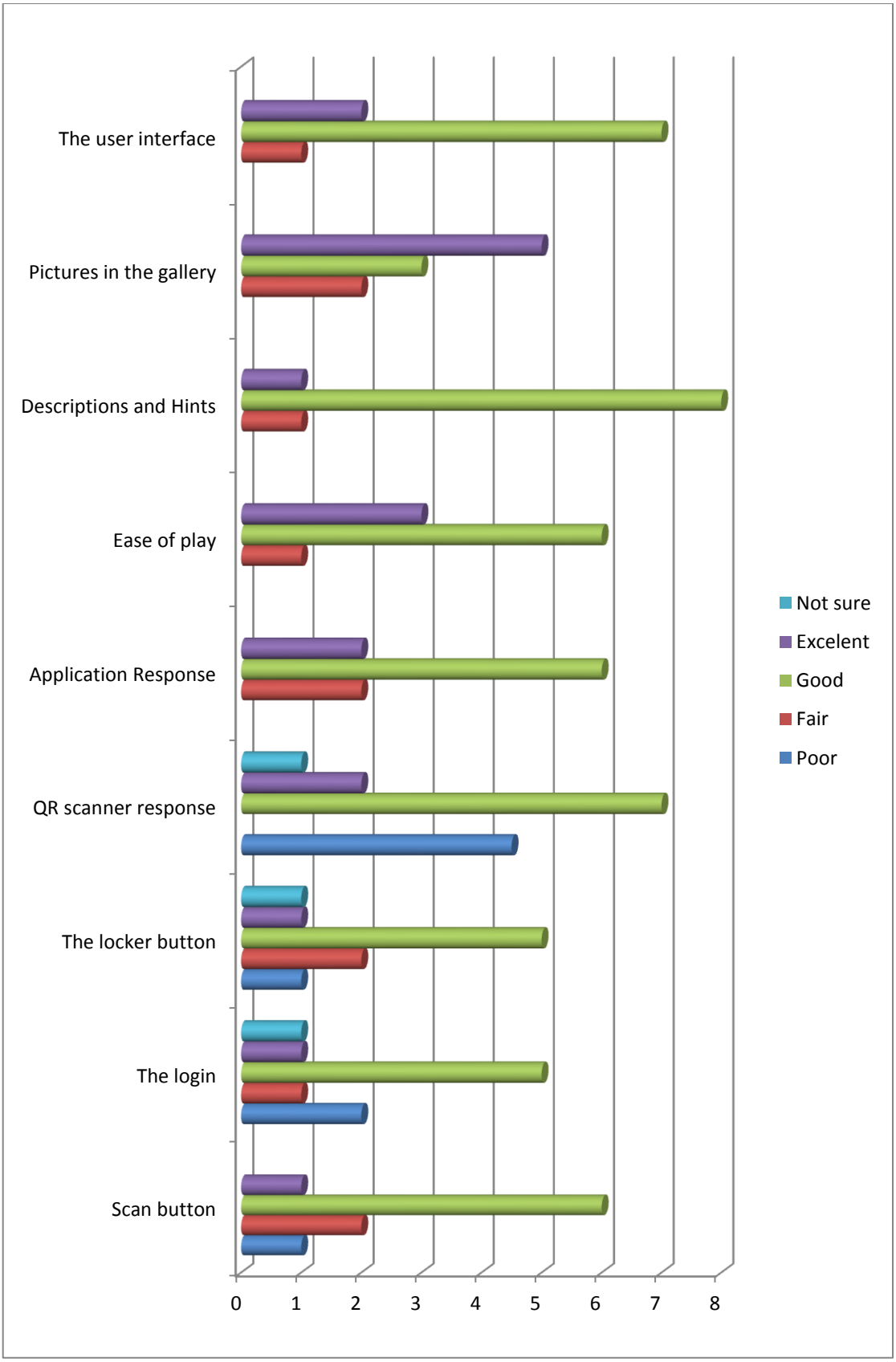

Do you like to play videogames?
Yes
No


Single player or Multiplayer?
Single
Multiplayer
Both



Questions about the game:

Here the most important statistic that collects the most important assessment of the respondents about the game:


The user interface
Pictures in the gallery
Descriptions and Hints
Ease of play
Application Response
QR scanner response
The locker button
The login
Scan button
0
1
2
3
4
5
6
7
8
Not sure
Excelent
Good
Fair
Poor

And finally, a list of the most important improvements proposed respondents:

- Bigger pictures and buttons.
- Find another interface design with other screens and more visual effects.
- The lock button

With these results we can analyze what the next step in the application development could be to improve the interface design changing the gallery style and the buttons. That means that maybe the future work would be focus to give a general change to the application and not change the style of lock, look for a different way of asking the user to iterate in order to unlock the game elements.

CHAPTER 6: Future work

This chapter is to summarize the proposals for improving implementation in the future. Some of these features have been proposed is the method of unlocking the game elements using GPS integrated in Android devices, a new login design that allows create and join games, different options depending of the scenario and program a real communications server.

GPS

The idea is to add a different type of exhibit besides the already have to be unlocked by QR codes.

Most Android devices allow to determine the current geolocation. This can be done via a GPS (Global Positioning System) device, via cell tower triangulation or via wifi networks.

Android contains the `android.location` package which provides the API to determine the current geo position.

The `LocationProvider` class is the superclass of the different location providers which deliver the information about the current location. This information is stored in the `Location` class.

The Android device might have several `LocationProvider` available and you can select which one you want to use. In most cases you have the followng `LocationProvider` available.

LocationProvider	Description
network	Uses the mobile network or WI-Fi to determine the best location. Might have a higher precision in closed rooms then GPS.
gps	Use the GPS receiver in the Android device to determine the best location via satellites. Usually better precision then network.
passive	Allows to participate in location of updates of other components to save energy

The `Geocoder` class allows to determine the geo-coordinates (longitude, laditude) for a given address and possible addresses for given geo-coordinates.

This process is known as forward and reverse geocoding.

The *MyLocationOverlay* class allows to display the current geolocation and allows to enable a compass.

The following code snippet shows how you can register *myLocationOverlay*:

```
// This goes into the onCreate method

myLocationOverlay = new MyLocationOverlay(this, mapView);

mapView.getOverlays().add(myLocationOverlay);


myLocationOverlay.runOnFirstFix(new Runnable() {

        public void run() {


        mapView.getController().animateTo(myLocationOverlay.getMyLoca
tion());

                }

});
```

Using this method we can collect the device's position and then we will use it in order to make a comparison with the *Collecdata* tag contained in the scrabble.xml file.

If the two strings are equal, the Geo-type exhibit will be unlocked.[v]

New Login screen design

The current login screen, as we know, allows the player to register in the XMPP server using his username and password. If in the future we want to implement a complex real server that will allows to create more than one game at the same time we have to improve this screen and add new options.

This new screen would include the option to create a new game or join an existing one created by another player. This function can be managed by Multi User Rooms as we explained in the XMPP Chapter.

Another important option that could be implemented is the option to choose more than one scenario to play. As we know the application collect the elements of the game

(Topics, Exhibits, Keys and their images) from the file we have to parse scrabble.xml. If we generate more than one file like this can give the player the option to choose between the many scenario options available. That is, we will have City Scrabble available in many cities.

Real Server

As the main aim of the project was to design one mobile application, we have not implemented yet a real server and Google Talk has been used for debugging.

In the future, we will need to program a server that receives messages from a particular player and send a broadcast message to the other players in order to know all the changes in the game as we have emulated.

Prosody

In order to program a real server, we are thinking that Prosody is the more adequate for our application.

Prosody is a modern flexible communications server for *Jabber/XMPP* written in *Lua*. It aims to be easy to set up and configure, and light on resources. For developers it aims to be easy to extend and give a flexible system on which to rapidly develop added functionality, or prototype new protocols.

It is licensed under the permissive *MIT/X11 license*.

Installing Prosody is quite straightforward in most cases. We have packages for the major operating systems to make it even easier.

Installing and configuring Prosody

Installing Prosody is quite straightforward in most cases. We have packages for the major operating systems to make it even easier. It is available in Windows, Linux and Mac OSX.

The configuration is divided into two parts. The first part is known as the “global” section. All settings here apply to the whole server, and all virtual hosts.

The second half of the file is a series of *VirtualHost* and Component definitions. Settings under each *VirtualHost* or Component line apply only to that host.

The only thing you are required to configure now is the hosts/domains you wish Prosody to serve.

A host in Prosody is a domain on which user accounts can be created. For example if you want your users to have addresses like john.smith@example.com then you need to add a host “example.com”.

Adding a virtual host to the server is as easy as adding a line to the configuration file under the global settings. For example.org, one would add:

```
VirtualHost "example.org"
```

All options under this heading will apply only to this host until another *VirtualHost* or Component entry, so be sure to add it in the right place after all the global options.

The name “virtual” host is used in configuration to avoid confusion with the actual physical host that Prosody is installed on. A single Prosody instance can serve many domains, each one defined as a *VirtualHost* entry in Prosody's configuration. Conversely a server that hosts a single domain would have just one *VirtualHost* entry.

Now you have your server configured and serving your domain you need to create some user accounts. The multiple ways of creating accounts into your Prosody server are described on our page 'Creating accounts'.

Components are extra services your server can provide, usually on subdomains of the main server.

They provide functionality such as *ChatRooms*, and transports/gateways to other networks and protocols.

Prosody has a number of built-in components; an example is the MUC (Multi-User Conference) component for running *ChatRooms*.

```
Component "conference.example.org" "muc"
```

This example sets up a MUC *ChatRoom* service at “conference.example.org”, which you can then join rooms on using your client.

Prosody also supports external server-independent components if they support XEP-0114. You can get more help on our page 'Configuring components', including how to add external components and other component options.[vi]

Conclusions

After the work developed, the first thing to mention in this chapter is that the prototype of the game is running satisfactorily. The evaluation that we made had good results and survey results were generally positive.

The idea of implementing this application, started with a previous design, based on REST Architecture. After the study, it has shown that the new XMPP protocol operates more correctly than the last one. The results confirm that the new protocol is more efficient and allows the programmer to more efficiently manage communication using multiple user rooms in the XMPP server.

After debugging the application we can probe that using the communications between devices and the XMPP Service to manage the sessions of all the players and other main features are implemented working property.

To implement the XMPP Service it has been used the Smack libraries. Many applications based on Android using this type of libraries, so we decided to work with them because since the first test, the communication system was good.

In addition, after multiple testing of the game, we detected some errors in CityScrabble v1 and most of them were corrected in order to design an application that works property.

About upcoming implements, the idea of the game is extend the location or scenario of the game not only to cities or museums, the location will be able to anywhere. The creation of new locations should be quick and easy. The application only needs a XML file with a particular structure to adapt to a new location. One of the future works that might be realized would be the creation of a tool to do this. Other interesting features could be integrating a GPS in order to program another way to unlock the exhibits by geolocation.

Maybe also a better login interface with more options that allows the player to create or join games would be useful.

Finally, the next step in the development of CityScrabble is to implement a XMPP server for communications between devices in order to create a commercial game ready for use.

APPENDIX

History of Android

Every year, more and more people have smart phones or other devices based on Android, a platform with hundreds of thousands of applications available to only a few clicks. So many developers have turned their sights on Android, a huge potential market that is growing.

The technological revolution in recent years has become possible with the wireless mobile devices. Therefore also this technology is used for creating video games. It must take into account the special characteristics of this technology in the design of applications for these devices.

The Open Handset Alliance led by Google, is the creator of Android. The first reports of developing applications for the Android platform were confusing. Among the topics discussed included errors, lack of documentation, infrastructure, inadequate quality control, lack of tracking system problems.

In December 2007, Adam MacBeth, founder of mobile MergeLab said *"There is no functionality, is poorly documented or not working and certainly not ready for release."* Nevertheless, the Android-driven applications began to appear the week of its appearance. The first published application was the snake game. The Android Development Phone is a device with the SIM and hardware unlocked, aimed at advanced developers. As consumer devices can be used to test and use normal applications, some programmers prefer unlocked devices without contract.

On 12 November 2007 was launched a test version of Android SDK. On July 15, 2008, the team Android Development Contest accidentally sent an email to all participants announcing a new version of the restricted area SDK downloads. The mail should be directed only to the winners of the first round. The fact that Google will provide a new version of SDK developers and not some other, through a private agreement, caused great discontent in the community of Android developers.

On 19 August release of version 0.9 Beta of Android SDK was released with enhanced API extended improved tools and an updated design of the main screen. There is detailed instructions update for those still using older versions. According to the release notes, included mostly bug fixes, although small changes incorporated. It included several changes to the API for version 0.9. Since then, many versions have been distributed.

In April 2011 there were over 200,000 applications accounted for Android, with about 3 billion downloads. The Android platform has grown to be one of the preferred developers for mobile platforms. A June 2011 study indicates that 67% of developers used mobile platform at the time of publication of the study.[vii]

Android

Android is a mobile operating system based on Linux and was designed for use on mobile devices like smart phones, tablets, Google TV and other devices. It is developed by the Open Handset Alliance, which is led by Google. This system usually handles applications such as Market or its update, Google Play Store.

It was initially developed by Android Inc., a firm acquired by Google in 2005.It is the main product of the Open Handset Alliance, a consortium of manufacturers and developers of hardware, software and other operators. Units of smart phones sold Android located in the first place in the United States, in the second and third quarters of 2010.

Worldwide reached a market share of 50.9 % during the fourth quarter of 2011, more than double the second operating system (iOS iPhone) with more volume.

It has a large community of developers writing applications to extend the functionality of the devices. At this date, there are about 400,000 applications (of which two thirds are free) available for official App Store Android: Google Play, regardless of other applications for Android unofficial stores, such as the Amazon's App Store or App Store apps Samsung Google Samsung.

Google Play is the online application store run by Google, although there is the possibility of external software. The programs are written in the programming language Java. However, there is an operating system free from malware, but most of it is downloaded from third party sites.

Android was announced 5th November 2007 along with the creation of the Open Handset Alliance, a consortium of 78 hardware companies, software and telecommunications dedicated to developing open standards for mobile devices.

Google released most of the code of Android under the Apache License, a license free and open source. It is the reason that many anonymous programmers work with this operating system.

The Android operating system structure consists of applications running in a Java framework for applications on the core Java libraries in a virtual machine Dalvik compiled at runtime. The libraries written in C language include a GUI manager (surface manager), a framework OpenCore a SQLite relational database, a programming interface API OpenGL ES 2.0 3D graphics, a rendering engine WebKit, a graphics engine SGL , SSL, and a standard library of C Bionic.

The operating system is composed of 12 million lines of code, including 3 million lines of XML, 2.8 million lines of C language, 2.1 million lines of Java and 1.75 million lines of C + +.[viii]

Update History

Android has seen numerous updates since its initial release. These updates typically base operating system bugs fix and add new features. Generally each update of the Android OS is developed under a code name of an element.

Android has been criticized many times by fragmentation suffered by their terminals not being supported with regular updates by the various manufacturers. It was believed that this situation would change after a Google update which stated that manufacturers undertake to apply updates at least 18 months after its release, but that the end never materialized and the project was canceled.[ix]

The code names are in alphabetical order.

The following lines describe the most important features of each version of android:

1.0	• Released on 23rd September 2008
1.1	• Released on 9th February 2009
1.5 (Cupcake) **Based on Linux kernel 2.6.27**	• Released on 30th April 2009 ▪ Ability to record and playback through the camcorder mode. ▪ Ability to upload videos to YouTube directly from your phone ▪ A new keyboard with predictive text ▪ Support for Bluetooth A2DP and AVRCP ▪ Automatic connection capability for connecting to Bluetooth headset at a distance ▪ New widgets and folders can be placed on the home screens ▪ Animated screen transitions
1.6 (Donut) **Based on Linux kernel 2.6.29**	▪ Released on 15th September 2009 ▪ The gallery now allows users to select multiple photos to eliminate ▪ Voice Search-date, with faster response and better integration with native applications, including the ability to make contacts ▪ Update support for CDMA / EVDO, 802.1x, VPN and text-to-

	speech ▪ Support for WVGA screen resolutions ▪ Framework of actions and development tool ▪ Free Navigation turn-by-turn Google
2.0 / 2.1 (Eclair) Based on Linux kernel 2.6.29	▪ Released on 26th October 2009 ▪ Optimized hardware speed ▪ Support for more screen sizes and resolutions ▪ New user interface and browser support for HTML5 ▪ 3.1.2 Improvements in Google Maps ▪ Integrated support for camera flash ▪ Digital zoom. ▪ Bluetooth 2.1 SDK **2.0.1** was released on 3rd December 2009 SDK **2.1** was released on 12th January 2010.
2.2 (Froyo) Based on Linux kernel 2.6.32	▪ Released on 20th October 2009 ▪ Overall system optimization Android, memory and performance ▪ Integration of the V8 JavaScript engine in Google Chrome Browser ▪ Enhanced application launcher with shortcuts to your applications and Browser ▪ Market Update with Automatic Updates ▪ Functionality of Wi-Fi hotspot and tethering via USB ▪ Turns off data traffic across the network operator ▪ Support for numeric and alphanumeric passwords ▪ Support for application installation on memory card ▪ Support for Adobe Flash 10 ▪ Support for display of high number of points per inch
2.3 (Gingerbread) Based on Linux kernel 2.6.35.7	▪ Released on 6th December 2010 ▪ Support for extra large screens and higher resolutions WXGA ▪ Native support for SIP VoIP ▪ Support for video playback and decoding WebM/VP8 AAC ▪ New audio effects like reverb, EQ, headphone virtualization and bass boost ▪ Features cut, copy and paste available throughout the system ▪ Multi-touch keypad redesigned

	▪ Improved support for native code development ▪ Improvements in data entry, audio and graphics for game developers ▪ Change file system YAFFS to ext462
3.0 / 3.1 / 3.2 (Honeycomb)	• Better support for tablets • 3D desktop widgets redesigned • Improved multitasking system • Improvements in the default web browser, on which is tabbed browsing, forms, sync bookmarks with Google Chrome and private browsing • Support for video chat using Google Talk • Better support for Wi-Fi networks and save separate settings for each SSID • Adds support for a wide variety of peripherals and accessories with USB keyboards • The widgets can be resized manually without limiting the number of frames that have each desktop • Adds optional support for resizing correctly initially created for mobile applications that look good on tablets

Android 4.0

Android 4.0 (Ice Cream Sandwich) is the latest version of the Android platform for phones, tablets, and more. It builds on the things people like most about Android: easy multitasking, rich notifications, customizable home screens, resizable widgets, and deep interactivity — and adds powerful new ways of communicating and sharing.

The new interface has eliminated all physical buttons on the screen, including digital 3 buttons at the bottom used to navigate the platform. The desktop has also been renovated, combining what is already seen in Gingerbread and Honeycomb, and enabling the creation of folders by simply dragging the icon.

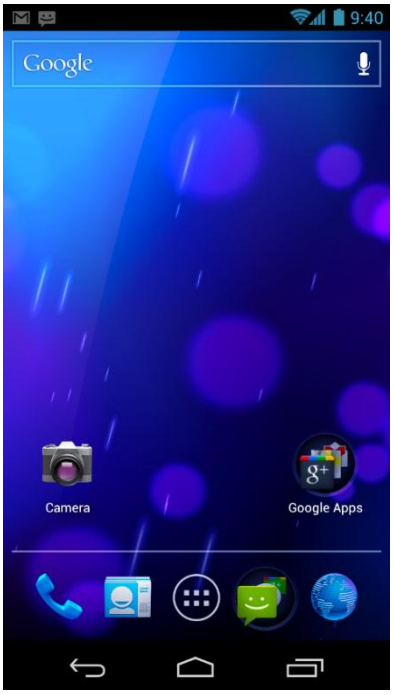


Figure 16.Android 4.0 Layaout

In terms of accessibility apps, Ice Cream Sandwich includes widgets that will allow we put into the place you want. This type of accessibility is the same as we saw in the tablets that integrate Honeycomb.

Another feature in the Android 4.0 operating system is the question of release. It can directly access any application and device notifications do not need to unlock, just as had been necessary until now. Ice Cream Sandwich offers the advantage of facial release. Just bring your face to the terminal, it unlocks automatically.

The task manager has also been renovated. A slider will show us that we have open apps, and most recently used, adding the ability to close them by dragging them.

The camera has been improved in terms of settings and effects, and other applications like Gmail, contacts or calendar also experience significant improvements over their predecessors.

Others important features are the following:

- This Version unifies the use on any device, both phones, tablets, televisions, netbooks, etc...
- Clean and modern interface called "Holo" with a new font called "Roboto", like Honeycomb.
- Option to use the virtual buttons in the user interface, instead of the capacitive touch buttons.
- The hardware acceleration, which means that the interface can be managed and drawn by the GPU and significantly increasing its speed, responsiveness and obviously the user experience.

- Improved multitasking, Honeycomb style. Adding the possibility of completing a task simply by moving off the list.
- Added a manager of Internet data traffic. The environment allows you to set alerts when you reach a certain amount of use and deactivation of data when moving from its limit.
- The widgets are in a new tab box application, contained in a list similar to applications in the main menu. So the option to be added by a long press on an empty area of the desktop is gone.
- The spelling of text has been redesigned and upgraded, offering the option of playing in a word for us to see a list of editing options similar words and suggestions.
- The notifications have the possibility to exclude non-significant and display the notification bar with the device locked.
- The screenshot, just press the volume down button and power button.
- The camera application has been a good facelift, with new utilities such as the ability to make panoramas automatically.
- Voice Recognition User
- New phone application with the functionality of visual voicemail lets you forward or back voice messages.
- Facial recognition, which would be able to change the view
- Folders are much easier to create, with drag and drop style[x]

HTC Desire

Figure17.HTC Desire

The mobile device used to test the application has been the **HTC Desire** that is a smartphone developed by the HTC Corporation, announced on 16 February 2010 and released in Europe and Australia in the second quarter of the same year. The HTC Desire runs the Android operating system, version 2.2 "Froyo". Android, version 2.3 "Gingerbread" update coming in May or June 2011.

List of specifications:

Processor: 1 GHz

Operating System: Android 2.2 (FroYo) with HTC Sense

Memory: Internal Memory = 1.5 GB / RAM = 768 MB / microSD Card Reader

Dimensions: size = 123 × 68 × 11.8 mm (4.84 × 2.67 × 0.46 inches) / Weight = 164 grams (5.78 ounces) with battery

Display: 4.3 "WVGA screen with capacitive multitouch technology, LCD with 262,000 colors

Connectivity: HSPA / WCDMA (900/2100 MHz) GSM / GPRS / EDGE (850/900/1800/1900 MHz)

Connections: Bluetooth ® 2.1, A2DP, FTP, OPP, PBAP and Wi-Fi ®: IEEE 802.11 b / g / n, 3.5 mm headphone jack Stereo standard micro-USB connector

Camera: A color, 8-megapixel with autofocus and dual LED flash, video recording at 720p HD

Power: Lithium Ion Battery replaceable and rechargeable 1230 mAh AC power adapter: 100 ~ 240V AC, 50/60 Hz

Sensors: movement, proximity and ambient light[xi]

Installing the SDK

You should have already downloaded the Android SDK. Now you need to set up your development environment.

The SDK you've downloaded is not the complete SDK environment. It includes only the core SDK tools, which you can use to download the rest of the SDK packages.

Getting started on Windows

Your download package is an executable file that starts an installer. The installer checks your machine for required tools, such as the proper Java SE Development Kit (JDK) and installs it if necessary. The installer then saves the Android SDK Tools into a default location (or you can specify the location).

Make a note of the name and location of the SDK directory on your system—you will need to refer to the SDK directory later, when setting up the ADT plugin and when using the SDK tools from the command line.

Once the tools are installed, the installer offers to start the Android SDK Manager.

Adding Platforms and Packages

The Android SDK separates different parts of the SDK into separately downloadable packages. The SDK starter package that you've installed includes only the SDK Tools. To develop an Android app, you also need to download at least one Android platform and the latest SDK Platform-tools.

You can update and install SDK packages at any time using the Android SDK Manager.

If you've used the Windows installer to install the SDK tools, you should already have the Android SDK Manager open. Otherwise, you can launch the Android SDK Manager in one of the following ways:

- On Windows, double-click the **SDK Manager.exe** file at the root of the Android SDK directory.
- On Mac or Linux, open a terminal and navigate to the **tools/** directory in the Android SDK, then execute **android sdk**.

When you open the Android SDK Manager, it automatically selects a set of recommended packages. Simply click **Install** to install the recommended packages. The Android SDK Manager installs the selected packages into your Android SDK environment. The following sections describe some of the available SDK packages and more about which ones we recommend you install.

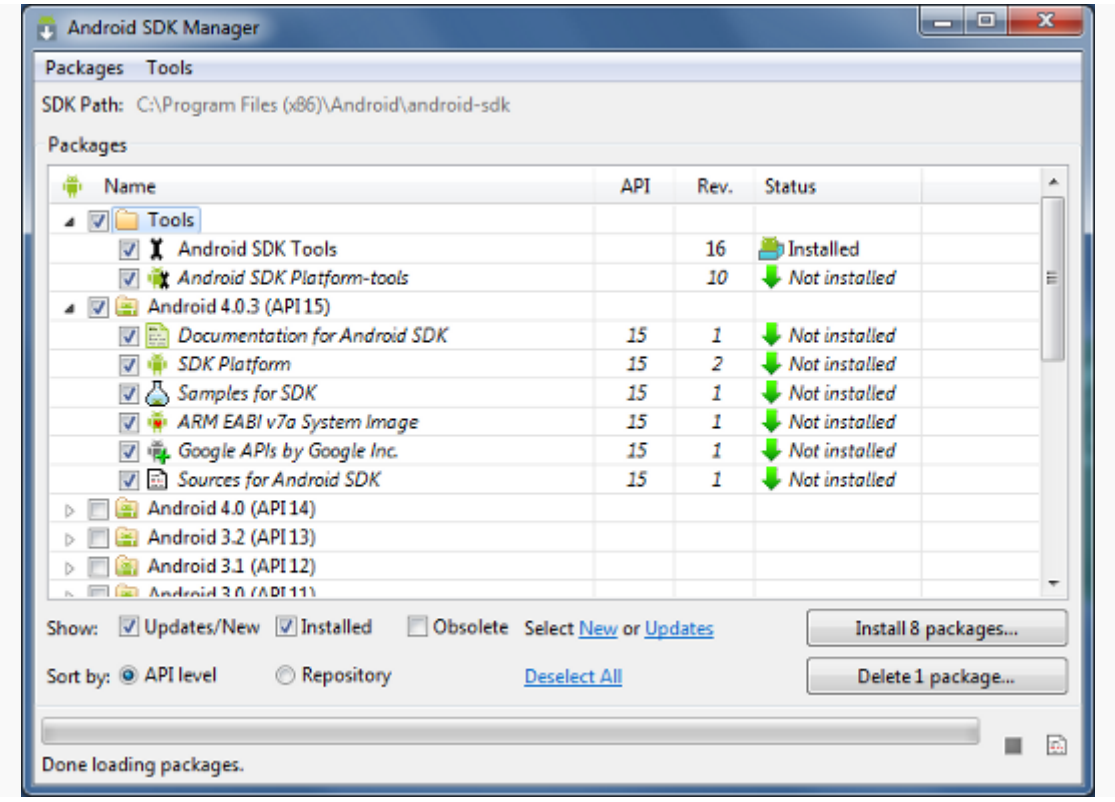


Figure 18 ***SDK Manager***

Installing the Eclipse Plugin

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT). This plugin is designed to give you a powerful, integrated environment in which to develop Android apps. It extends the capabilites of Eclipse to let you quickly set up new Android projects, build an app UI, debug your app, and export signed (or unsigned) app packages (APKs) for distribution.

If you will be developing in Eclipse with the ADT Plugin, first make sure that you have a suitable version of Eclipse installed on your computer as described by the system requirements.

If you need to install Eclipse, you can download it from http://www.eclipse.org/downloads/. We recommend the "Eclipse Classic" version. Otherwise, you should use a Java or RCP version of Eclipse.

Note: If you prefer to work in a different IDE, you do not need to install Eclipse or ADT. Instead, you can directly use the SDK tools to build and debug your application.

Download the ADT Plugin

1. Start Eclipse, then select **Help** > **Install New Software...**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*:

```
https://dl-ssl.google.com/android/eclipse/
```

4. Click **OK**

   Note: If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**.

   Note: If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
8. When the installation completes, restart Eclipse.

Configure the ADT Plugin

After you've installed ADT and restarted Eclipse, you must specify the location of your Android SDK directory:

1. Select **Window** > **Preferences...** to open the Preferences panel (on Mac OS X, select **Eclipse** > **Preferences**).
2. Select **Android** from the left panel.

   You may see a dialog asking whether you want to send usage statistics to Google. If so, make your choice and click**Proceed**.
3. For the *SDK Location* in the main panel, click **Browse...** and locate your downloaded Android SDK directory (such as **android-sdk-windows**).
4. Click **Apply**, then **OK**.

Updating the ADT Plugin

From time to time, a new revision of the ADT Plugin becomes available, with new features and bug fixes. Generally, when a new revision of ADT is available, you should update to it as soon as convenient.

In some cases, a new revision of ADT will have a dependency on a specific revision of the Android SDK Tools. If such dependencies exist, you will need to update the SDK Tools package of the SDK after installing the new revision of ADT. To update the SDK Tools package, use the Android SDK Manager.

To determine the version currently installed, open the Eclipse Installed Software window using **Help** >**Software Updates** and refer to the version listed for "Android Development Tools".

Follow the steps below to check whether an update is available and, if so, to install it.

1. Select **Help** > **Check for Updates**.

   If there are no updates available, a dialog will say so and you're done.
2. If there are updates available, select Android DDMS, Android Development Tools, and Android Hierarchy Viewer, then click **Next**.
3. In the Update Details dialog, click **Next**.
4. Read and accept the license agreement and then click **Finish**. This will download and install the latest version of Android DDMS and Android Development Tools.
5. Restart Eclipse.[xii]

Creating an Android Project

An Android project contains all the files that comprise the source code for your Android app. The Android SDK tools make it easy to start a new Android project with a set of default project directories and files.

Create a Project with Eclipse

1. In Eclipse, select **File > New > Project**. The resulting dialog should have a folder labeled *Android*.
2. Open the *Android* folder, select *Android Project* and click **Next**.
3. Enter a project name (such as "MyFirstApp") and click **Next**.
4. Select a build target. This is the platform version against which you will compile your app.

   We recommend that you select the latest version possible. You can still build your app to support older versions, but setting the build target to the latest version allows you to easily optimize your app for a great user experience on the latest Android-powered devices.

   Click **Next**.

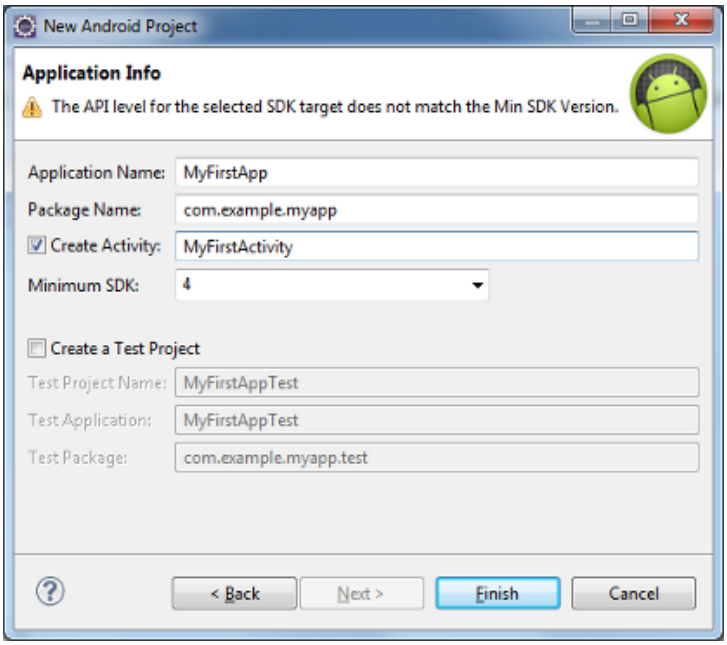


Figure 19New Android Project

5. Specify other app details, such as the:
   - ***Application Name***: The app name that appears to the user. Enter "My First App".
   - ***Package Name***: The package namespace for your app (following the same rules as packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. For this reason, it's important that you use a standard domain-style package name that's appropriate to your company or publisher entity. For your first app, you can use something like "com.example.myapp." However, you cannot publish your app using the "com.example" namespace.
   - ***Create Activity***: This is the class name for the primary user activity in your app (an activity represents a single screen in your app). Enter "MyFirstActivity".
   - ***Minimum SDK***: Select *4 (Android 1.6)*.

Because this version is lower than the build target selected for the app, a warning appears, but that's alright. You simply need to be sure that you don't use any APIs that require an API level greater than the minimum SDK version without first using some code to verify the device's system version (you'll see this in some other classes).

Click **Finish**.

Your Android project is now set up with some default files and you're ready to begin building the app.

Running Your App

How you run your app depends on two things: whether you have a real Android-powered device and whether you're using Eclipse. This lesson shows you how to install and run your app on a real device and on the Android emulator, and in both cases with either Eclipse or the command line tools.

Before you run your app, you should be aware of a few directories and files in the Android project:

`AndroidManifest.xml`

This manifest file describes the fundamental characteristics of the app and defines each of its components. You'll learn about various declarations in this file as you read more training classes.

`src/`

Directory for your app's main source files. By default, it includes an **Activity** class that runs when your app is launched using the app icon.

`res/`

Contains several sub-directories for app resources. Here are just a few:

`drawable-hdpi/`

Directory for drawable objects (such as bitmaps) that are designed for high-density (hdpi) screens. Other drawable directories contain assets designed for other screen densities.

`layout/`

Directory for files that define your app's user interface.

values/

Directory for other various XML files that contain a collection of resources, such as string and color definitions.

When you build and run the default Android project, the default **Activity** class in the **src/** directory starts and loads a layout file from the **layout/** directory, which includes a "Hello World" message. Not real exciting, but it's important that you understand how to build and run your app before adding real functionality to the app.

Run on a Real Device

You need to:

1. Plug in your Android-powered device to your machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device.
2. Ensure that **USB debugging** is enabled in the device Settings (open Settings and navitage to **Applications** > **Development** on most devices, or select **Developer options** on Android 4.0 and higher).

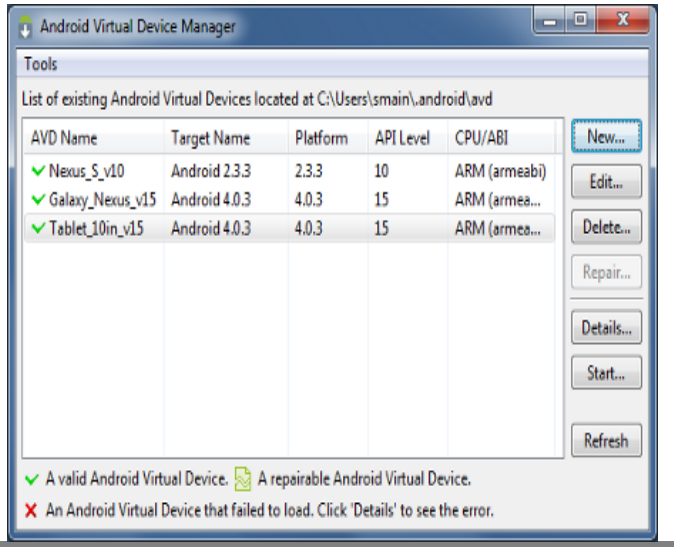
To run the app from Eclipse, open one of your project's files and click **Run** from the toolbar. Eclipse installs the app on your connected device and starts it.

Run on the Emulator

You need to first create an Android Virtual Device (AVD). An AVD is a device configuration for the Android emulator that allows you to model different device configurations.

To create an AVD:

1. Launch the Android Virtual Device Manager:
   a. In Eclipse, select **Window** > **AVD Manager**, or click the*AVD Manager* icon in the Eclipse toolbar.
   b. From the command line, change directories to**<sdk>/tools/** and execute:
2. In the *Android Virtual Device Device Manager* panel, click **New**.



3. Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default).
4. Click **Create AVD**.
5. Select the new AVD from the *Android Virtual Device Manager* and click **Start**.
6. After the emulator boots up, unlock the emulator screen.

To run the app from Eclipse, open one of your project's files and click **Run** from the toolbar. Eclipse installs the app on your AVD and starts it.[xiii]

Android Develop

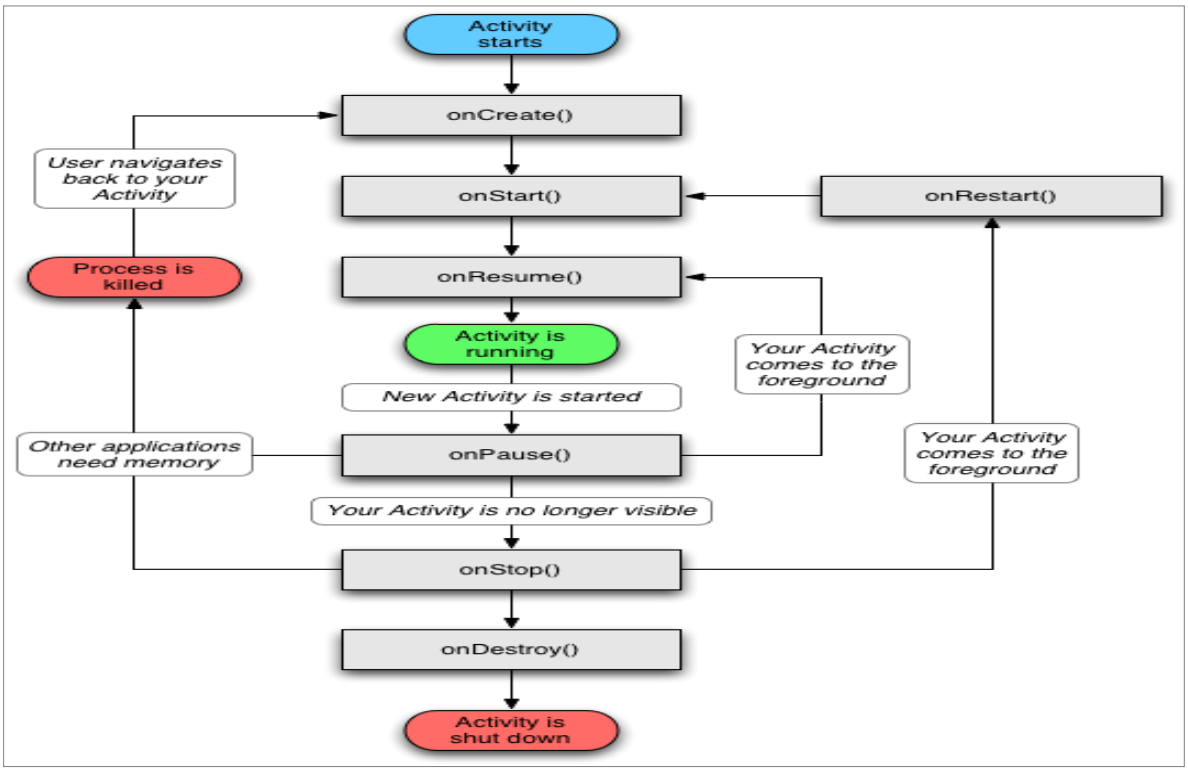
Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface.

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.

If we want to create a class that extends Activity must implement at least the function onCreate (), which is executed when the activity is created. By default this function receives a package (bundle), which contains the state of the activity if it has been suspended.

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle *callback* methods. There are *callback* methods that an activity might receive, for example when the system is creating it, stopping it, resuming it, or destroying it. Each *callback* provides you the opportunity to perform specific work that's appropriate to that state change.



Managing the Activity Lifecycle

Managing the lifecycle of your activities by implementing *callback* methods is interesting to developing a strong and flexible application. The lifecycle of an activity is directly affected by its association with other activities, its task and back stack.

Method	Description	Killable after?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see Saving Activity State, later). Always followed by onStart().	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()	No	onStart()
onStart()	Called just before the activity becomes visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()
onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.	**Yes**	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.	**Yes**	onRestart() or onDestroy()

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish, or simply killing its process. When the activity is opened again (after being finished or killed), it must be created all over.

Intents

Intents are asynchronous messages which allow Android components to request functionality from other components of the Android system. For example an Activity can send an Intent to the Android system which starts another Activity.

Therefore Intents allow combining loosely coupled components to perform certain tasks.

They can be used to signal to the Android system that a certain event has occurred. Other components in Android can register to this event and will get notified.

Intents are instances of the *android.content.Intent* class.

When they are sending to the Android system, depending on how the Intent was constructed the Android system will run a receiver determination and determine what to do.

Intent can also contain data. This data can be used by the receiving component. For example your application can calls via Intent a browser component. As data is it may send the URL to the browser component.

Calling Activities

If you send an Intent to the Android system, Android requires that you tell it to which type of component your Intent should be send.

To start an Activity use the method startActivity(Intent). This method is defined on the Context object and available in every Activity object.

After writing a single activity, there comes a need to transition to another activity to perform another task either with or without information from the first activity.

Do not forget to declare any new activity in the AndroidManifest.xml with permission.

The following shows an explicit Intent. If that Intent is correctly sent to the Android system, it will start the associated class:

```
Intent intent = new Intent(this, ActivityTwo.class);

startActivity(intent)
```

Launch a second "Activity" and pass parameters

We can see the difference with the above concept is that we call the class putExtra Intent. It has two parameters; the first data indicate the name and the second data value:

```
Intent intent = new Intent(this, ActivityTwo.class);

intent.putExtra("Value1", This value one for ActivityTwo );

intent.putExtra("Value2", This value two ActivityTwo);

startActivity(intent);
```

In ActivityTwo.class we have to collect the data sent by the first activity. For example, Value1 was a String and Value2 was an Integer in the last activity:

```
String Value1 = intent.getStringExtra("Value1");

int Value2 = intent.getStringExtra("Value2");
```

Launch a Service using Intents

The correct way to lunch a Service from an Activity is following this code:

```
Intent serviceIntent;

serviceIntent = new Intent(context, XMPPClientService.class);

startService(serviceIntent);
```

If we need to send some variable to the service:

```
Intent serviceIntent;

serviceIntent = new Intent(context, XMPPClientService.class);

serviceIntent.putExtra("userAccess", userAccess);

serviceIntent.putExtra("passAccess", passAccess);

serviceIntent.putExtra("myPlayerId", MyPlayerId);

startService(serviceIntent);
```

Do not forget to declare any new activity in the AndroidManifest.xml with permission:

```
<service

android:name="scrabble.project.services.XMPPClientService">

</service>
```

Defining Intent Filters

IntentFilters are typically defined via the "AndroidManifest.xml" file. For *BroadcastReceiver* it is also possible to define them in coding. An *IntentFilters* is defined by its category, action and data filters. It can also contain additional metadata.

The following will register an Activity for the Intent which is triggered when someone wants to open a webpage.

```
<activity
android:name=".BrowserActivitiy" android:label="@string/app_name">
  <intent-filter>
     <action android:name="android.intent.action.VIEW" />
     <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

Another way to do this is to use the java class IntentFilter:

```
IntentFilter newFilter = new
IntentFilter(ApplicationClass.INTENT_NAME);
NewReceiver newReceiver = new NewReceiver();
registerReceiver(newReceiver, newFilter);
```

Layouts

Layout is the architecture for the user interface in an Activity. It defines the layout structure and holds all the elements that appear to the user. You can declare your layout in two ways:

- **Declare UI elements in XML**: Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**: Your application can create View and *ViewGroup* objects (and manipulate their properties)

Using Android's XML vocabulary, you can quickly design layouts and the screen elements they contain, in the same way you create web pages in HTML.

In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods.

For example, here's an XML layout that uses a vertical *LinearLayout* to hold a *TextView* and a Button:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

              android:layout_width="fill_parent"

              android:layout_height="fill_parent"

              android:orientation="vertical" >

    <TextView android:id="@+id/text"

              android:layout_width="wrap_content"

              android:layout_height="wrap_content"

              android:text="Hello, I am a TextView" />

    <Button android:id="@+id/button"

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:text="Hello, I am a Button" />

</LinearLayout>
```

Load the XML Resource

When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.*layout_file_name* For example, if your XML layout is saved as main_layout.xml, you would load it for your Activity like so:

```
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main_layout);

}
```

The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched.

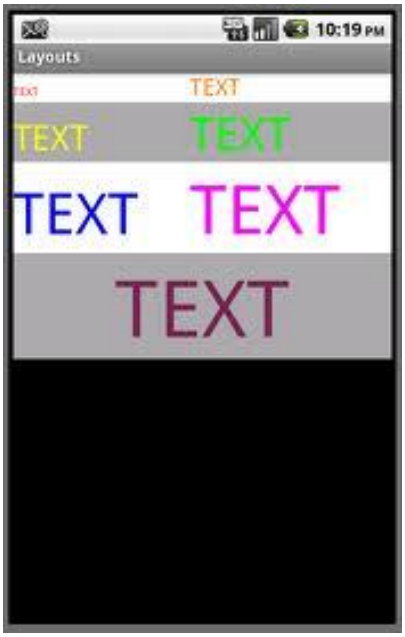


Figure20. Layout example 1

Figure21.Layaout example2

Application object

We know there is an Application class in the Android api and according to the class name it's used for global settings or running entrance.

In the Android reference it describes the Application class:

"Base class for those who need to maintain global application state. You can provide your own implementation by specifying its name in your AndroidManifest.xml's tag, which will cause that class to be instantiated for you when the process for your application/package is created."

You need to extend the *android.app.Application* class and override the *OnCreate* method. This method is invoked by the Android runtime once when your application is started. So this is a good point to do application initialization and set some global settings like shared preferences, etc.

So you can create your own subclass of Application like this:

public class MyApplication extends Application {

```
  //Application wide instance variables
      //Preferable to expose them via getter/setter methods
@Override
  public void onCreate() {
    super.onCreate();
    //Do Application initialization over here
        }
        //Appplication wide methods
}
```

You can then introduce public methods in your Application Class. These public methods can then be called from anywhere in the Android application:

And specify its name in your AndroidManifest.xml's tag:

```
<application android:icon="@drawable/icon" android:label="@string/app_name"
android:name="MyApplication">
```

Sometimes you want to store data, like global variables which need to be accessed from multiple Activities - sometimes everywhere within the application. In this case, the Application object will help you.

For example, if you want to get the basic authentication data ,you can implement the methods for authentication data in the application object.

After this,you can get the username and password in any of the activities like this:

```
MyApplication mApplication = (MyApplication)getApplicationContext();
String username = mApplication.getUsername();
String password = mApplication.getPassword();
```

Having an Application wide controller in your application is a commonly used design with significant benefits. Instead of rolling out your own custom implementation,

Android provides a simple mechanism to introduce it in your applications, letting you focus on your Application wide logic while it takes care of its lifecycle methods.

Services

What is a service?

A service is an application that runs automatically, without user interaction. It is an important task to develop other applications or the system.

Services on Android are not independent, are in the same process as the application that consumes them.

A service can essentially take two forms:

Started: A service is "started" when an application component starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

Bound: A service is "bound" when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

To create a service, you must create a subclass of Service or one of its existing subclasses). In your implementation, you need to override some *callback* methods that handle key aspects of the service lifecycle and provide a mechanism for components to bind to the service,

Declaring a service in the manifest

Like activities (and other components), you must declare all services in your application's manifest file.

To declare your service, add a <service> element as a child of the <application> element. For example:

```
<manifest ... >

  ...

  <application ... >

      <service android:name=".ExampleService" />

      ...

  </application>

</manifest>
```

Broadcast Receiver

A broadcast receiver is a class that extends *BroadcastReceiver* and is registered as a receiver in an Android application either through AndroidManifest.xml file or using the code. *BroadcastReceiver* class will be able to receive the method Intents *sendBroadcast* (). It also defines the method *OnReceive* (), where our object is valid broadcast receiver, after the end of its execution, the system will consider that the object is no longer active, so it no longer possible to perform some other task asynchronously.

Typically, a Broadcast Receiver is used to display notifications of events that occur in our mobile phone such as the discovery of a wireless network or battery depletion.

With a *BroadcastReceiver* we can capture events like:

```
Event android.provider.Telephony.SMS_RECEIVED received message.
Event android.intent.action.PHONE_STATE calls.
Event android.intent.action.AIRPLANE_MODE flight mode.
Event android.intent.action.BATTERY_LOW low battery.
Event android.intent.action.BOOT_COMPLETED operating system boot.
Event android.intent.action.SCREEN_OFF screen lock.
Event android.intent.action.SCREEN_ON unlock screen.
Event start android.bluetooth.intent.action.DISCOVERY_STARTED Bluetooth
scanner.
Event android.bluetooth.intent.action.ENABLED Bluetooth enabled.
```

There are two classes of broadcasts that can be received:

- **Normal broadcasts** (sent with *Context.sendBroadcast*) are completely asynchronous. All receivers of the broadcast are run in an undefined order, often at the same time. This is more efficient, but means that receivers cannot use the result or abort APIs included here.

- **Ordered broadcasts** (sent with *Context.sendOrderedBroadcast*) are delivered to one receiver at a time. As each receiver executes in turn, it can propagate a result to the next receiver, or it can completely abort the broadcast so that it won't be passed to other receivers. The order receivers run in can be controlled with the *android:priority* attribute of the matching intent-filter; receivers with the same priority will be run in an arbitrary order.

Even in the case of normal broadcasts, the system may in some situations revert to delivering the broadcast one receiver at a time. In particular, for receivers that may require the creation of a process, only one will be run at a time to avoid overloading the system with new processes. In this situation, however, the non-ordered semantics hold: these receivers still cannot return results or abort their broadcast.

Note that, although the Intent class is used for sending and receiving these broadcasts, the Intent broadcast mechanism here is completely separate from Intents that are used to start Activities with *Context.startActivity*(). There is no way for a *BroadcastReceiver* to see or capture Intents used with *startActivity*(); likewise, when you broadcast an Intent, you will never find or start an Activity. These two operations are semantically very different: starting an Activity with an Intent is a foreground operation that modifies what the user is currently interacting with; broadcasting an Intent is a background operation that the user is not normally aware of.

An example of a *BroadcastReceiver* class would be the following code:

```
public class ReceiverService  extends BroadcastReceiver {

            @Override

            public void onReceive(Context context, Intent intent) {


            String username = intent.getStringExtra("username");

            String password = intent.getStringExtra("password");

            }

}
```

The *BroadcastReceiver* class is launched as a component through a manifest's <receiver> tag).

Dialogs and Toast messages

The Android Dialog class (*android.app.Dialog*) is the base class for all types of dialog controls that you can use within your Activity classes. Dialogs live within the

lifecycle of your Activity (*android.app.Activity*). They pop up in the foreground, blocking your Activity screen, to catch the user's attention for a variety of reasons.

Dialogs are useful when you want to:

- Inform the user of some event or progress (e.g. "You have mail!" Or "Downloading Message 1 of 200,000")
- Force the user to confirm an action (e.g. "Are you sure you want to delete all your contacts? Really sure?")
- Prompt the user for further information and collect it (e.g. "Please enter your username and password.")

Some developers also use Toast messages (*android.widget.Toast*) for sending simple notifications or messages to the user. A Toast message displays over your Activity screen for a few seconds and then disappears automatically. The user has no chance to interact with a Toast message. We like to think that the dividing line between when to use a Toast over a Dialog is as follows: if the user is being informed of non-essential information, use a Toast, but when the information being presented is vital, use a Dialog. We use Toasts as very lightweight, informational notifications. Any information you want to ensure the user acknowledges should be displayed using a Dialog that requires their active participation to dismiss.[xiv]

A Toast message example is shown below:

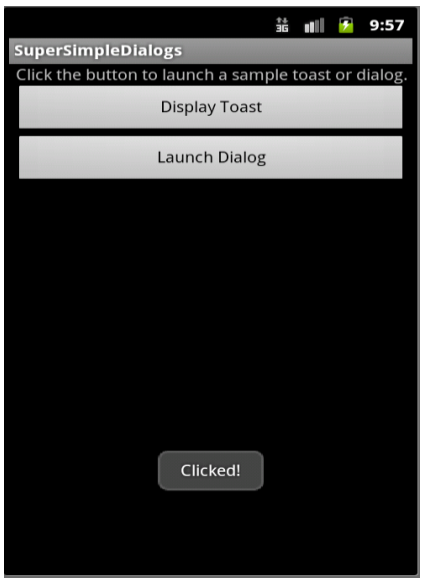


Figure 20. Toast example

Parsers

XML parser on Android

The Android platform is an open source mobile development platform. It gives you access to all aspects of the mobile device that it runs on, from low level graphics, to hardware like the camera on a phone. With so many things possible using Android, you might wonder why you need to bother with XML. It is not that working with XML is so interesting; it is working with the things that it enables. XML is commonly used as a data format on the Internet. If you want to access data from the Internet, chances are that the data will be in the form of XML. If you want to send data to a Web service, you might also need to send XML. In short, if your Android application will leverage the Internet, then you will probably need to work with XML. Luckily, you have a lot of options available for working with XML on Android.

One of the greatest strengths of the Android platform is that it leverages the Java programming language. The Android SDK does not quite offer everything available to your standard Java Runtime Environment (JRE,) but it supports a very significant fraction of it. The Java platform has supported many different ways to work with XML for quite some time, and most of Java's XML-related APIs are fully supported on Android. For example, Java's Simple API for XML (SAX) and the Document Object Model (DOM) are both available on Android. Both of these APIs have been part of Java technology for many years. The newer Streaming API for XML (StAX) is not available in Android. However, Android provides a functionally equivalent library. Finally, the Java XML Binding API is also not available in Android. This API could surely be implemented in Android. However, it tends to be a heavyweight API, with many instances of many different classes often needed to represent an XML document. Thus, it is less than ideal for a constrained environment such as the handheld devices that Android is designed to run on. In the following sections, you will take a simple source of XML available on the Internet, and see how to parse it within an Android application using the various APIs mentioned above. First, look at the essential parts of the simple application that will use XML from the Internet.

Working with DOM

DOM parsing on Android is fully supported. It works exactly as it works in Java code that you would run on a desktop machine or a server. The following code shows a DOM-based implementation of the parser interface.

```
public class DomFeedParser extends BaseFeedParser {

    protected DomFeedParser(String feedUrl) {
        super(feedUrl);
    }

    public List<Message> parse() {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        List<Message> messages = new ArrayList<Message>();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document dom = builder.parse(this.getInputStream());
            Element root = dom.getDocumentElement();
            NodeList items = root.getElementsByTagName(ITEM);
            for (int i=0;i<items.getLength();i++){
                Message message = new Message();
                Node item = items.item(i);
                NodeList properties = item.getChildNodes();
                for (int j=0;j<properties.getLength();j++){
                    Node property = properties.item(j);
                    String name = property.getNodeName();
                    if (name.equalsIgnoreCase(TITLE)){
                        message.setTitle(property.getFirstChild().getNodeValue());
                    } else if (name.equalsIgnoreCase(LINK)){
                        message.setLink(property.getFirstChild().getNodeValue());
                    } else if (name.equalsIgnoreCase(DESCRIPTION)){
                        StringBuilder text = new StringBuilder();
                        NodeList chars = property.getChildNodes();
                        for (int k=0;k<chars.getLength();k++){
                            text.append(chars.item(k).getNodeValue());
                        }
                        message.setDescription(text.toString());
                    } else if (name.equalsIgnoreCase(PUB_DATE)){
                        message.setDate(property.getFirstChild().getNodeValue());
                    }
                }
                messages.add(message);
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return messages;
    }
}
```

Figure 21. DOM example

The DOM parser reads the entire XML document into memory and then allows you to use the DOM APIs to transverse the XML tree, retrieving the data that you want. This is very straightforward code, and, in some ways, simpler than the SAX-based implementations. However, DOM generally consumes more memory as everything is read into memory first. This can be a problem on mobile devices that run Android, but it can be satisfactory in certain use cases where the size of the XML document will never be very large. One might imply that the developers of Android guessed that SAX parsing would be much more common on Android applications, hence the extra utilities provided for it. One other type of XML parser is available to you on Android, and that is the pull parser.[xv]

DOM Methods

There are two types of Java's code depending on the type of data that we want to parse.

```
public List<Hint> parseHint(InputStream is, String s) {
      // get the root elememt
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
  List<Hint> listHints = new ArrayList<Hint>();
  try{
  DocumentBuilder builder = factory.newDocumentBuilder();
      Document dom = builder.parse(is);
      // get the root elememt
      Element docEle = dom.getDocumentElement();
      // get a nodelist of <players> elements
      NodeList nl = docEle.getElementsByTagName(s);

      if (nl != null && nl.getLength() > 0) {
            for (int i = 0; i < nl.getLength(); i++) {
                  Element elementHint = (Element) nl.item(i);
                  // b. get the player element
                  Hint hint = getHintFromXml(elementHint);
                  // c. add it to list
                  listHints.add(hint);
                  }
            }
  }
       catch (Exception ex)
     {
     }
  return listHints;
}


private Hint getHintFromXml(Element elementPlayer) {
      int id = Integer.parseInt(elementPlayer.getAttribute("id"));
      String info = elementPlayer.getAttribute("info");
      // Object with id and info data that we parsed
      Hint newHint = new Hint(id,info);
      return newHint;
}
```

This code allows us to collect data within the tag.

The following method performs the same function but collecting the data between tags:

```
public List<Topic> parse(InputStream is, String s)
    {
            DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
        List<Topic> topiclist = new ArrayList<Topic>();
        try
        {
            DocumentBuilder builder = factory.newDocumentBuilder();
```

```
            Document dom = builder.parse(is);
            // get the root elememt
            Element docEle = dom.getDocumentElement();

            // get a nodelist of <players> elements
            NodeList nl = docEle.getElementsByTagName(s);

           for (int i=0; i<nl.getLength(); i++)
           {
               Topic topic = new Topic();
               Node item = nl.item(i);
               NodeList datosNoticia = item.getChildNodes();
               for (int j=0; j<datosNoticia.getLength(); j++)
               {
                   Node dato = datosNoticia.item(j);
                   String etiqueta = dato.getNodeName();

                   if (etiqueta.equals("title")) {
                         String texto = obtenerTexto(dato);
                         topic.setTitulo(texto);
                   }
                   else if (etiqueta.equals("description"))
                   {
                       String texto = obtenerTexto(dato);
                       topic.setDescripcion(texto);
                   }
                   //...
               }
               topiclist.add(topic);
           }
       }


      catch (Exception ex)
       {
            Log.e(_TAG, ex.toString());
       }
       return topiclist;
   }
```

These two methods are very important and we can use one or the other depending on where we want to collect information from the XML file.

Smack Libraries

Downloading and adding libraries

Now, I will explain how to get Smack library and the process of integrating it in our game:

1. First of all, we have to download the Smack API. I found the API version 2.3.3 (latest version) in the following webpage:

   http://www.igniterealtime.org/projects/smack/

2. We will add jar files in our Eclipse project: Properties -> Java Build Patch -> Add JARS, and then select the files location as we can see in the following screenshots:

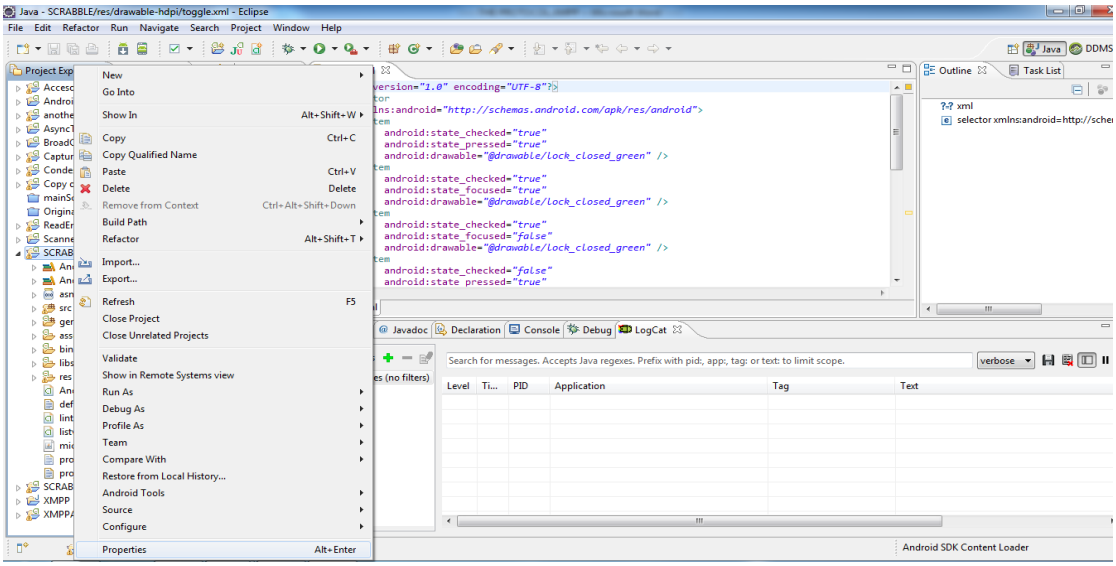


Figure 22. Step 1

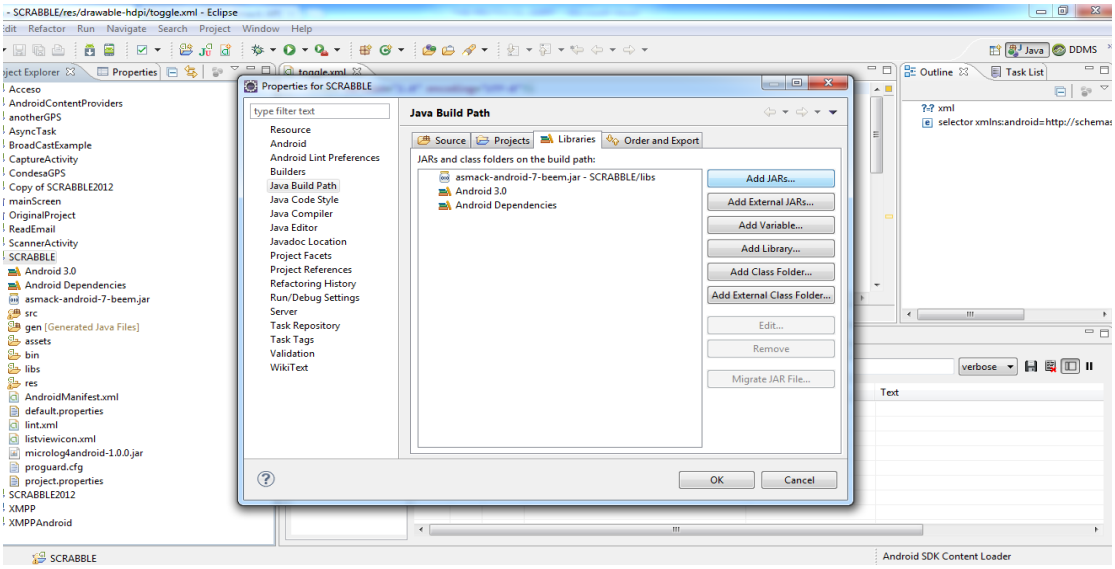


Figure 23. Step 2

JAR Files and Requirements

Smack is meant to be easily embedded into any existing JDK 1.5 or later Java application. It has no external dependencies (except for the Jingle voice chat functionality) and is optimized to be as small as possible. The library ships as several JAR files to provide more flexibility over which features applications require:

- *smack.jar* -- provides core XMPP functionality and is the only required library. All XMPP features that are part of the XMPP RFCs are included.
- *smackx.jar* -- support for many of the extensions (XEPs) defined by the XMPP Standards Foundation, including multi-user chat, file transfer, user search, etc.
- *smackx-debug.jar* -- an enhanced GUI debugger for protocol traffic. It will automatically be used when found in the classpath and when debugging is enabled.

The Mechanical Turk

What it is?

The Turk was a famous farce posing as a chess-playing automaton. It was built and revealed by Wolfgang von Kempelen in 1769. It was shaped like a log cabin four feet long and 60 cm deep and 90 high, with a mannequin dressed in a tunic and turban sitting on it. The cabinet had doors that once opened showed clockwork and when it was activated was able to play a game of chess against a human player to a high standard. As well as perform the knight's tour, a puzzle that requires the player to move a knight to occupy every square of a chessboard exactly once. However, the cabin was well-considered an optical illusion that allowed a chess master to hide inside and operate the mannequin. Consequently, the Turk won most of the games.

While the Turk was a scam and not a real robot, later it was real chess attempts by robots, such as "The Chess Player."

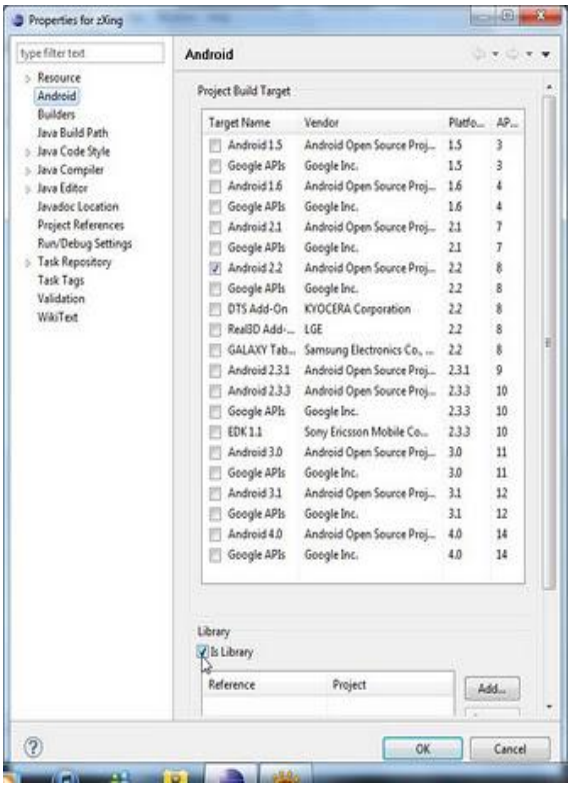
Zxing and QR Codes

ZXing

ZXing is an open-source, multi-format 1D/2D barcode image processing library implemented in *Java*. This library focuses on using the built-in camera on mobile phones to photograph and decode barcodes on the device, without communicating with a server.

Downloading and Installing Zxing

Grab the latest stable distribution from http://code.google.com /p/zxing/downloads/list. After that, extract the contents of the .zip file in the root directory of the hard disk.



Until version 1.7, the programmer had to compile the libraries in order to create the *core.jar* file using Apache Ant.

In the last version, Zxing 2.0, *core.jar* file is compiled and ready for use.

If we want to Integrate ZXing source lib into the Android Code project through Eclipse we have to do the following:

In *Eclipse* right-click the Project Folder → *Properties* → *Java Build Path* → Libraries → Add External JARs →Navigate to the *ZXing* folder and open the core directory. Select *core.jar*.

Finally, it is necessary to correct a few errors in the translations and the AndroidManifest.xml file. After that, it is possible to compile, and the developer will have a working standalone barcode scanner application based on the *ZXing* source.

As we did in Chapter 3, in addition to the *core.jar* file, in the archive Zxing2.0.zip that we downloaded before, we find a folder called */android*. If we download this, it is another project that works as a library and if we integrate in our project, it is not necessary to ask the user to install Barcode Scanner because it is fully integrated into the application:

QR Codes

What is a QR Code?

At its most basic, a QR Code is a barcode on steroids. They're used for encoding information in two-dimensional space, like in the pages of magazines, in advertisements and even on TV and Web sites. They were originally used to track auto parts, but have become popular for much broader, often commercial purposes.

They come to us from Japan where they are very common. QR is short for Quick Response (they can be read quickly by a cell phone). They are used to take a piece of information from a transitory media and put it in to your cell phone. You may soon see QR Codes in a magazine advert, on a billboard, a web page or even on someone's t-shirt. Once it is in your cell phone, it may give you details about that business (allowing users to search for nearby locations), or details about the person wearing the t-shirt, show you a URL which you can click to see a trailer for a movie, or it may give you a coupon which you can use in a local outlet.

The reason why they are more useful than a standard barcode is that they can store (and digitally present) much more data, including URL links, geo coordinates, and text. The other key feature of QR Codes is that instead of requiring a chunky hand-held scanner to scan them, many modern cell phones can scan them.

How is it different than a barcode?

Whereas a barcode encodes data in only the horizontal plane (as scanners read the width and distance between the vertical lines), QR codes encode data both horizontally and vertically in a grid of tiny squares. This allows for much more data to be encoded in a smaller space. Barcodes, then, though ubiquitous, are good for little more than identifying products and objects. Specially programmed scanners can read barcodes, and match them to product names, prices and inventory, but that's about it. QR codes, on the other hand, can actually embed that information in the code itself, and, when read with the proper software, can trigger actions like launching a website or downloading a file. Additionally, QR codes can be read from any angle, while barcodes must be aligned properly.

So what exactly can I do with QR codes?

QR codes are tailor-made for quickly and easily linking to content on smartphones. Simple uses include magazine advertisements that link to websites.

Android uses QR codes to link directly to apps in the Android Marketplace and provide links to information from the World Heritage Foundation and guide visitors to nearby shops or parking locations via Google Maps. In turn, Google has been using QR codes to promote local businesses (and itself) with the Google Places business directory, which includes reviews, contact info, and, if the business so wishes, coupons.

How can I use them?

There are a number of apps in the iPhone App Store that can read QR Codes, including the free *QRReader*. Most Android phones and BlackBerries are able to read the codes right out of the box, as can newer Nokia handsets. Windows Mobile users can download *QuickMarks*. All you need to do is launch the appropriate app, and point your phone's camera at the QR code you want to scan.

QR codes are only bound to become more common in the coming months and years. We're increasingly reliant on our mobile devices, and typing out URLs or other data on their tiny keyboards is still not very efficient. These squares of elaborately arranged boxes are a shortcut around that problem, can easily be integrated with various services, and incorporate geo-location data.

Encoding

The format information records two things: the error correction level and the mask pattern used for the symbol. Masking is used to break up patterns in the data area that might confuse a scanner, such as large blank areas or misleading features that look like the locator marks. The mask patterns are defined on a 6×6 grid that is repeated as necessary to cover the whole symbol. Modules corresponding to the dark areas of the mask are inverted. The format information is protected from errors with a BCH code, and two complete copies are included in each QR symbol.

The message data is placed from right to left in a zigzag pattern, as shown below. In larger symbols, this is complicated by the presence of the alignment patterns and the use of multiple interleaved error-correction blocks.

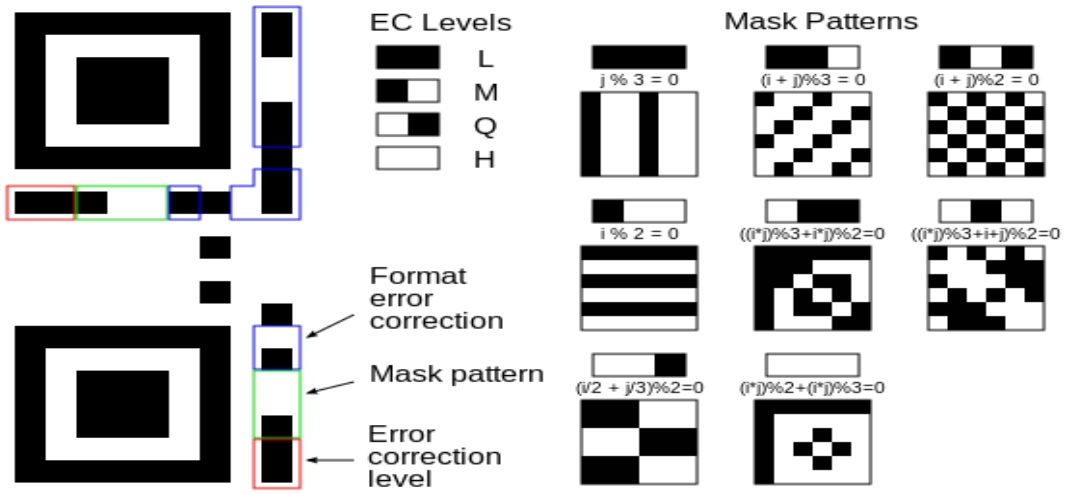


Figure 24. Encoding

Storage

The amount of data that can be stored in the QR Code symbol depends on the datatype (*mode*, or input character set), version (1,...,40, indicating the overall dimensions of the symbol), and error correction level. The maximum storage capacities occur for 40-L symbols (version 40, error correction level L), and are as follows (where *character* refers to individual values of the input *mode/datatype*, as indicated)

- **Numeric only:** Max. 7,089 characters (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- **Alphanumeric:** Max. 4,296 characters (0–9, A–Z [upper-case only], space, $, %, *, +, -, /, :)
- **Binary/byte:** Max. 2,953 characters (8-bit bytes) (23624 bits)
- **Kanji/Kana:** Max. 1,817 characters

Version 1 (21×21). Content: "Ver1"

Version 2 (25×25). Content: "Version 2"

Version 3 (29×29). Content: "Version 3 QR Code"

Version 4 (33×33). Content: "Version 4 QR Code, up to 50 char"

Version 10 (57×57). Content: "Version 10 QR code, up to 174 char at H level, with 57x57 modules and plenty of error correction to go around. Note that there are additional tracking boxes"

Version 40 (177×177). Content: "Version 40 QR Code can contain up to 1852 chars. ..." (a total of 1,264 characters of ordinary/ASCII text [taken from an early version of this Wikipedia article])

Used program to read QR codes: Barcode Scanner

Nowadays, you can find dozens of readers capable of reading and interpreting these codes in Internet. Everything depends on the operating system of your mobile phone and little more.

I have used Barcode Scanner. This is a reader of polyvalent QR codes. Beside reading and to interpret this standard, the program is capable to capture other formats of bar codes and to throw searches of references in Internet.

With Barcode Scanner you will be able to read bar codes using the camera of your telephone and to find in the web the corresponding products.

To use Barcode Scanner you have to focus the bar code with your camera. If the code is recognizable, in a few seconds you should have the reference of the concrete product.

Barcode Scanner is very useful to find information of, for example, books or DVDs and later to buy them in Internet. In addition, Android has developed an intelligent system with which you will be able to accede to the Android Market across the codes QR that have his programs.

Finally, Barcode Scanner is also suitable for supermarkets and incorporates the classic option of sonorous notice or for vibration whenever it recognizes a product.

References

[i] Android Developers : Design http://developer.android.com/design/index.html

[ii] InformIT: Introduction to Mobile Architectures
http://www.informit.com/articles/article.aspx?p=336262

[iii] Extensible Messaging and Presence Protocol :
http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol

[iv] Smack Documentation : http://www.igniterealtime.org/builds/smack/docs/latest/documentation/

[v] Largs Vogel : http://www.vogella.com/articles/AndroidLocationAPI/article.html

[vi] Prosody : http://prosody.im/

[vii] Android Software Development : http://en.wikipedia.org/wiki/Android_software_development

[viii] Android: http://en.wikipedia.org/wiki/Android_(operating_system)

[ix] Android (Spanish) http://es.wikipedia.org/wiki/Android

[x] Introducing Android 4.0 http://www.android.com/about/ice-cream-sandwich/

[xi] HTC Desire Full Phone Specifications http://www.fonearena.com/htc-desire_1078.html

[xii] Android Developers: Installing SDK http://developer.android.com/sdk/installing/index.html

[xiii] Android Developers : Building your First App
http://developer.android.com/training/basics/firstapp/index.html

[xiv] Android Developers : Reference http://developer.android.com/reference/packages.html

[xv] IMB Developers Work: http://www.ibm.com/developerworks/library/x-android/