Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

M.M.O. GEOCACHING GAME WITH MODULAR CONNECTIONS

Autor: Xavier Febrer Jordà

Director: Eloi Puertas Prats

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 25 de juny de 2015

Table of Contents

1. Introduction

M.M.O. Geocaching Game with Modular Connections - is a final grade project that pretends to create a game for Android devices, where the users participate in a massive multiplayer online experience in real time and located by GPS.

This project wants to be close to the new devices, services offered by other companies, libraries,... to have an end product in the modern market, using concepts like geocaching to narrow the distance between users using a synchronized and online service.

1.1 Project Overview

1.1.1 A Modern Approach

The evolution of the society with technology, produces more users connected to the Internet. The big difference between now and a couple of years ago, is that almost every user has more than one device combining with a very easy access to it, tying the basic needs of a sociable human.

For this, a little revolution in the computer world has happened, what was a static multimedia center, work station, communication rules,... has just retired drastically. With perspective, it can be seen that the appearance of the smartphones, as a heart of our essential operations. like communication and the giant support that services from other external companies available for everyone, made the desktop computers and laptops to adapt to those services, evolving to a level of portability never seen in a massive way in real life. Those devices adapted in a more usable physical form and the

software too, for example, the concept of cloud in the recent years, really pushed the synchronization of the data to a really easy usable way in words of a developer. It can be seen in tablets, laptops with the screen capable of the user touch, that those evolutions derived from the previous computer stages.

There are some big examples of the software evolution that is not directly developed for a desktop pc or laptop:

a. Clash of Clans is an MMO game only for mobile devices, where you control a city and other real people can attack you, you can create online clans,... The decisive thing is that is online, is one of the most played games in the world (so you will encounter a lot of people and fast) and in addition you synchronize the state of the game in the device you log in, meaning that the user will notice a significantly less waste of time.
b. Whatsapp is an application that has a lot of popularity with more than 500 Million users, puts all the people connected so easily, with 1€ of cost per year approximately and a 24 hour 365 days a year perfect service. This makes the users join directly, rely on it, learn and remember the usability and the company.
c. Google Play Services is a library for developers that provides the services of Google to the users, that way the users can rely on a great company that manages the email, the application data, all fast, secure and for free. On the other hand, the developers can rely on a server that allows to store and retrieve essential data and they do not have to maintain servers, which is really not very desirable if you can avoid it so easily.

The usage of the cloud services, makes them good for processing data, and the smartphones or other devices makes them a good client to receive it. This client-server architecture is used in this project, because it needs a center of intelligence to synchronize and process the data to harmonize the game.

If we go deeper, this project is divided in parts called modules and are easy to identificate: server, Android client, Android Wear support client and a MongoDB database. Other modules are planned and can be applied in the project: the module to maintain the world populated of resources, an A.I. as a client or as an internal part of the server, splitting the server into a request receiver and a request processor,...

The client of the project, needs to send location data to the server, any device that knows how to get the position in the world and can access to the server, normally via Internet, is able to run an application that can execute this project's protocol and play with the other people in the game.

The game result of this, is still in evolution, it adapts to the ideology before and at least has a clear idea of what is happening right now in the market and how to compete in it.

1.1.2 A Fast Introduction to Join The World

In the game, each user has a unique account that resides in the server. Here is where fantasy starts, in this account the user can create two characters, to play with them in the world and choose a civilization: belong to the Othura, they are a future imperial civilization and being a mage of their great magical legion is a respected position into their society, otherwise the Slako civilization are a biologically evolved human empire, that they know how to be strong when they are out there alone, specially their witches with their magic.

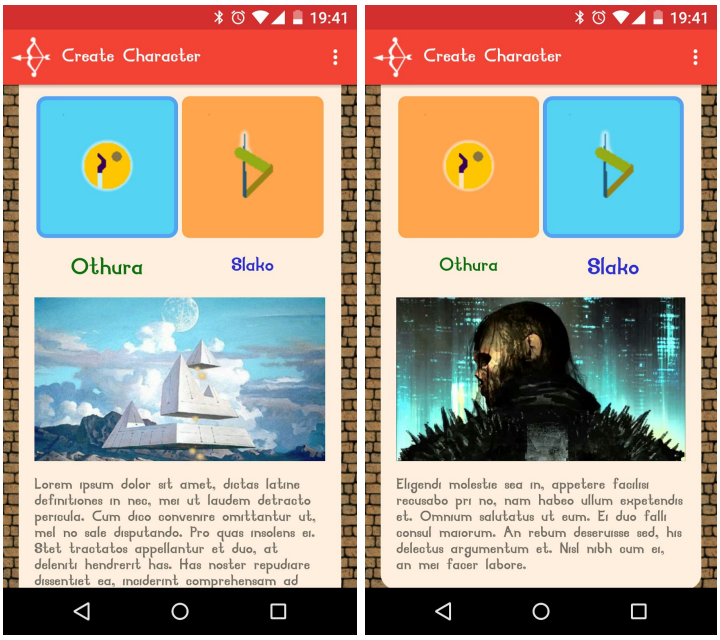

Fig.1 Civilization selection.

Once a character has been selected, the game centers the character selected to the user's location and pretends to override the reality of the world with a fantasy map combined: with the current real world map, with coins, keys, chests, portals, other real player characters,... in a real time and obviously online.

To sum up, the selected character of the player appears in the world as a fantasy character and the streets and cities are populated with objects and fantasy stuff. This way, the user can play where it goes and can explore the world and leave its mark capturing a portal for example.

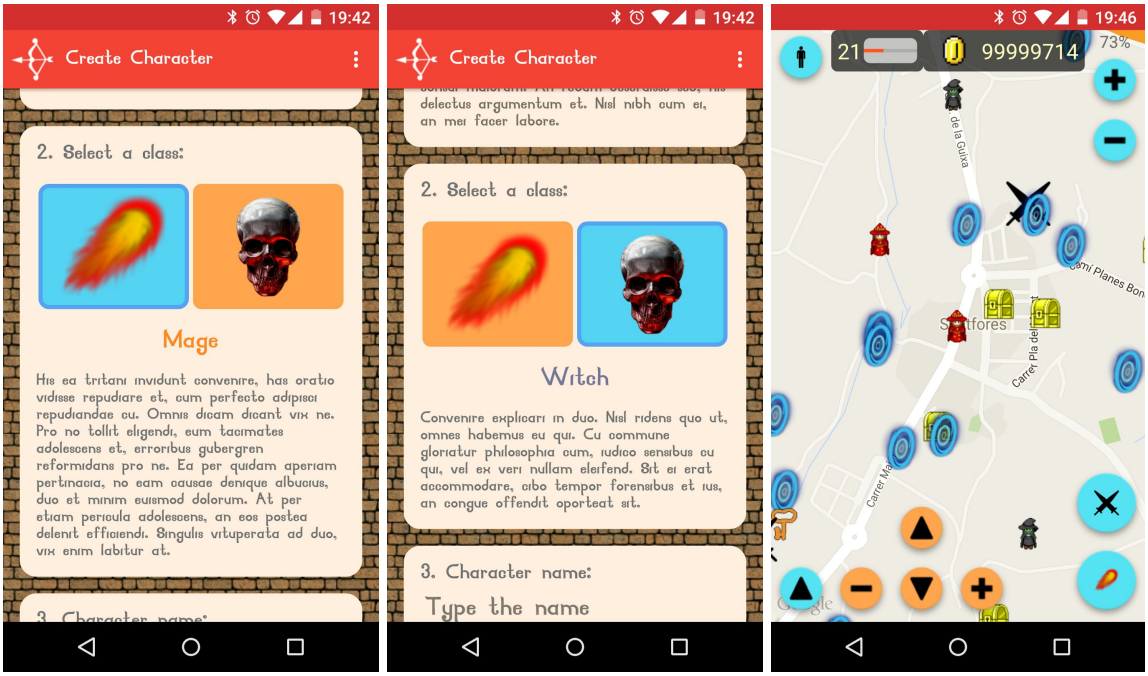

Fig.2 Mage and Witch.

1.2 Motivation, Target Audience and Status

Currently, while exploring the possibilities that smartphones bring us, to replace things that in the past were did in a different way and try to search for an improvement, in time, in values,... the mobile device is one of the most important sectors of the planet. The objective of this game is to innovate in new fields, having a clue of what other games and services in general do, which show a greatly positive performance.

In addition, this project is an skeleton or some kind of SDK, where some modules can be reused for other applications or they can be improved to increase the stability of the core, because the more stability of the server has in this project, the more fluent gameplay the users feel.

Games Clash of Clans & Candy Crush Saga	Social Facebook & Instagram	Communication Whatsapp & Facebook Messenger
Installs 100,000,000 - 500,000,000 Installs 100,000,000 - 500,000,000	Installs 500,000,000 - 1,000,000,000 Installs 1,000,000,000 - 5,000,000,000	Installs 1,000,000,000 - 5,000,000,000 Installs 1,000,000,000 - 5,000,000,000

Fig.3 Popularity of the cloud synchronized applications. Data from Google Play.

This game is oriented approximately from teenagers to 30 year old people, because the type of representation, visualization and gameplay, fits to the way of life of these people. This population is very active in the world of smartphones and in particular Android.

Currently, the state of the game is not final, neither it was thought to be in the future, because the game evolve with its module structure, to achieve the good level of stability for the audience. However, each revision or update can be a small release and one small release can be a big release, considering the things that can be presented to the users.

At the start of the project, a connection protocol was needed, and putting walls to the development and to the modules was needed to concatenate the parts.

1.3 Requirements

To play this game, some approximate requirements are needed:

- Android: 4.2 minimum.
- CPU: dual-core 2.5 GHz minimum or quad-core with 2.3 minimum recommended.
- RAM: 2GB minimum, 3GB-4GB recommended
- Storage: 50MB minimum, 100 recommended.
- GPS: needed.
- Accelerometer: needed.
- Magnetometer: needed.

If your device does not have this requirements the app is not assured to run well or even run.

You can attach any smartwatch on the market considering a minimum of 512MB of RAM and a Qualcomm Snapdragon 400 processor at least. The smartwatch is used to represent data, so it won't perform big procedures during the game. The required Android version of the smartwatch is 4.2 at least.

In the current version of the game, the battery and the internet connection is not optimized, as it is a pre-alpha release, bugs can make a misuse of those sources.

The next lines of code must go inside the AndroidManifest.xml and inside the <manifest><manifest/> tag:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

These lines of code are permissions granted to the developer when publishing the app. Without these permissions, the application will fail to execute. This way, when a user installs the application, can see the resources used by the application and can prevent malicious software to be installed in their device.

Mongo uses a server to provide a connection to the database, and that server can be configured by following these easy steps:
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

In this application, the MongoDB server listens to the port 20206.
After that the MongoDB Java library provides Objects maintain a connection to the server in a form of a facade to provide an easy usage.

The Java server of this application can be reached by connecting to the server ip and the hardcoded port value of 20202.

It is recommended to open the ports from 20200 to 20210 TCP/UDP to be able to connect the different parts of the MongoDB to the server, and the server to the client.

To be able to work on the project, it is needed the latest Android Studio distribution and the latest Eclipse with web plugin to open the respective Android Studio and Eclipse Web projects.

2. Planification and Objectives

To follow and adapt to the changing world this project, as said in the introduction, does not have a final release, it can have future periodic revisions and updates. Using the ideology of Kanban as methodology to develop this project, it is viable to change to Scrum or stay Kanban, it is compatible for the releases and for the development, because for the people involved and the time remaining made Kanban the best approach.

In the next sections, the procedures, tasks and decisions that were made in the past, will be explained.

2.1 Software, Hardware and more Considerations

2.1.1 Device and software needs

For each user of the game, an Android smartphone device or an Android tablet as a center element is used. Also an Android smartwatch is used to increase the game experience as an optional element, which its functionality is to show the context of the game and the data of the central device. It is needed to differentiate that Android Wear is an extension of Android and it depends on a smartphone or a tablet to complete its Android core functionality.

Thanks to the modular structure, the protocol of the game not only accept Android, but also other applications for other platforms can be added to the project easily, if the device agrees the requirements of the project, like an IPhone, Windows Phone or even laptops.

For the software part, the connection between the user and the server is made by web services. Currently it is used Java for all the client parts and the server, which uses a Jersey library to make easy the requests received in the server.

Once the hardware and software needs are found, it must start the iteration of the planification about the project.

2.2 First Approach

First of all, the first to think was that the project was implemented with a server, that managed the world and users connected to the world, with an Android smartphone. An Android smartwatch was considered from the beginning.

It was planified by parts and it was accorded that every two or three weeks, there will be a revision and the planification will be revisited according to the requirements, the look and feel of the current application, the remaining time,...

For this, the first revision was an application that regrouped approximately the next objectives:

- Game interface, menus,... all gamificated in some way, to show an ambience. It needs to be remarked that there is no need to contact to a server in this revision, the part of receiving the data to the server can be mocked.

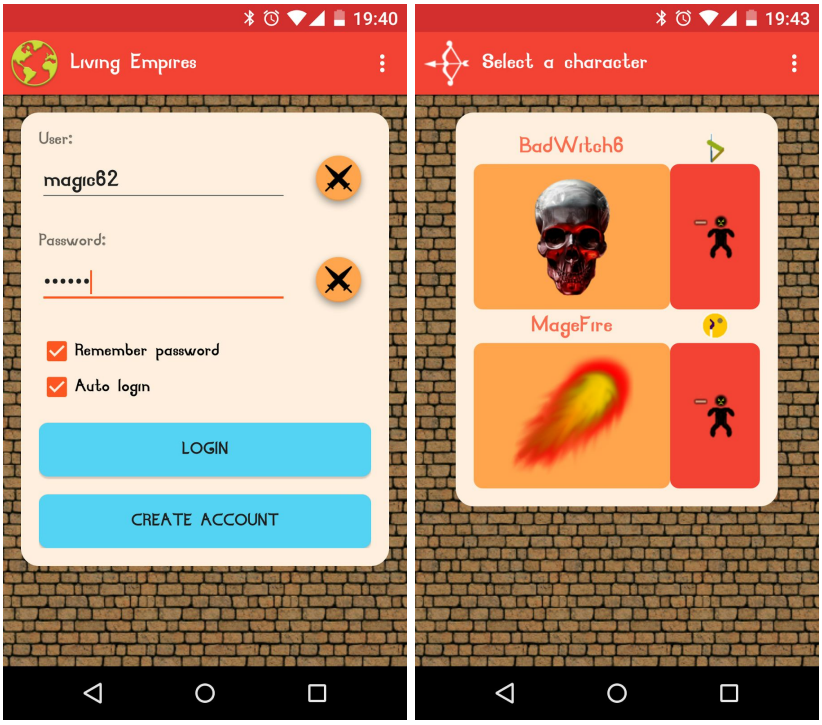

Fig.4 Gamificated Interface

- Having enabled the google maps service for the users, so the user can use the maps with a legally certificate.

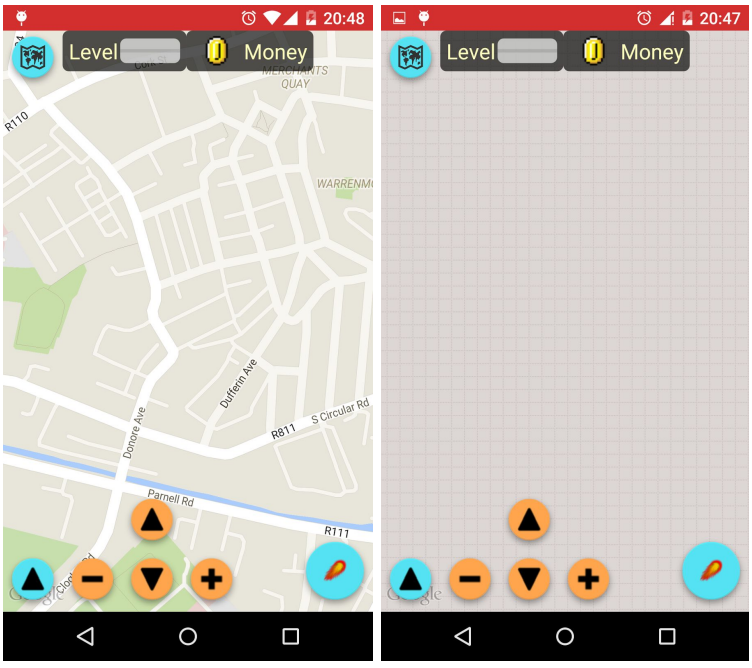

Fig.5 Difference between a licenced and an unlicenced Google Maps.

- Add support library dependencies to the interface like widgets and functionalities, to be able to provide a nice visual approach of the application and to not lose time on those aspects, is important because it can consume a lot of time creating those widgets from scratch, and a lot of libraries can provide you with free and good material.

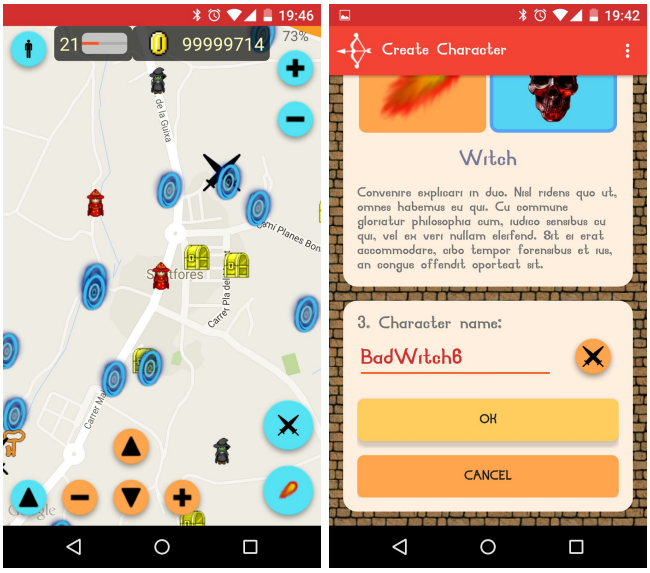

Fig.6 The fancy circle buttons in this screen are part from the FloatingActionButton library.

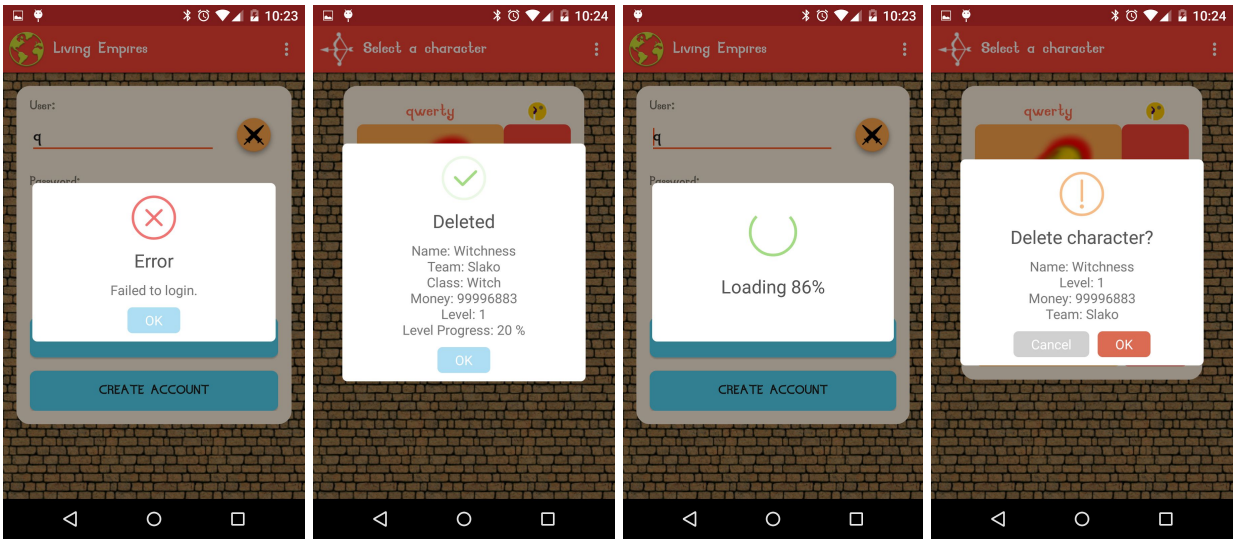

Fig.7 The fancy dialogs are part of the SweetAlert library.

- Learn and test the new technologies like Jersey, web services with Android.
- Before this revision it was planned to used a 3D map renderer, but google maps was finally considered to make things easier.
- Be able to demonstrate an application to the revision that shows these objectives.

2.3 Iterations

As the time passed, the menus, the look and feel were modified, but after the first release the server had to work. Developing the server and the Android client altogether, the account data was created to let the testers be able to login to the developing application.

Because of that, some packages contained different objects to make this communication easier: the request clients, the transfer objects if needed, and response objects from the server.

The game menu reached a state where it was alright to stop developing it, because it had the music and sounds required and the time needed to be spent on other things.

More at the end, when the application worked, an Android library to communicate with the Android Wear was introduced, to make things easier because it can send JSON strings to, like the server, that way the smartwatch receives a JSON string containing all the needed information.

One thing to remark is that the application as the development and the data structures grow, the network was not very well optimized and when playing with data internet, the game suffered from lag, but the optimization part does not fit until the final, if there is time.

2.4 Current approach

The current approach is playable. This means a user can connect to a server, create a character, join the world, grab coins, get keys and open chests, throw abilities to to world or players, capture portals, chat with other people,... and because of that, the main objective of this project is complete because it was planned from the beginning.

Other parts can be created in the future:

- The Gardener is a planned module for the server, that controls and repopulates the world, so the players can expect automatic generated resources like coins, keys, portals,... This module can be inside of the server, it can be outside of the server, it can have A.I. and because of that, the modular structure of the project can make things separated or not, it is the developer's choice.

- A.I. Bots can be added as a normal client to the server, they must be controlled for security reasons. As said before, this part can be a client like another player or they can be inside of the server to minimize the work of the server.

3. Development

3.1 Logic and Protocol

Each user application requires a connection to the server to synchronize the state. For this, the Android part and the server have Transfer Objects, that are converted to a JSON String and they are sent through the network. Thanks to JSON the object are easily reconverted to Object again.

Important note: The current Jersey version has a bug and the POST HTTP method is not recognized for the server. For this, the GET HTTP method is used for all the calls to the server. Fixing this thing into the server is changing the Annotations that Jersey provide, but for the requirements and the time remaining for this release, the GET HTTP methods are prefered because it is needed a working server. In the future they must be changed.

3.1.1 Client Requests

The requests of the client are JSON Strings, all the requests can be found below in a readable way:

Name	**Req. Get Account Data**
Description	Get the user account data
Method	GET
Endpoint	/account
Parameters	- String token - String id - String password
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Login**

Description	Login the user to the server, this way only one device can access using the token
Method	GET
Endpoint	/login
Parameters	- String id - String password
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Logout**
Description	Removes the token of the session and device
Method	GET
Endpoint	/logout
Parameters	- String token - String id - String password
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Register**
Description	Registers a user to the server
Method	GET

Endpoint	/register
Parameters	- String id - String password - String email
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Delete Game Character**
Description	Deletes a character from an account
Method	GET
Endpoint	/register
Parameters	- String token - String id - String password - int accountSlot
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Update email**
Description	Updates an email from an account
Method	GET
Endpoint	/update_email
Parameters	- String token - String id

	- String password - String newEmail
Response Object from Retrofit:	AccountInfoResponse

Name	**Select Game Character**
Description	Selects a Game Character to start playing the game
Method	GET
Endpoint	/select_game_character
Parameters	- String token - String id - String password - String accountSlot
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Add Game Character**
Description	Adds a Game Character to the account slot
Method	GET
Endpoint	/add_game_character
Parameters	- String token - String id - String password

	- String accountSlot - int gameCharacterTeam - int gameCharacterClass - String gameCharacterName - double gameCharacterLatitude - double gameCharacterLongitude - double gameCharacterMetersSee - double gameCharacterMetersTouch
Response Object from Retrofit:	AccountInfoResponse

Name	**Req. Move Game Character**
Description	Moves a character to the given position, if the character triggers an event moving, the response will contain the data.
Method	GET
Endpoint	/move_game_character
Parameters	- String token - String id - String password - double latitude - double longitude
Response Object from Retrofit:	MoveGameCharacterInfoResponse

Name	**Req. Throw Projectile**

Description	Throws a projectile to a given angle (clockwise)
Method	GET
Endpoint	/throw_projectile
Parameters	- String token - String id - String password - double angle - int type
Response Object from Retrofit:	OkResponse

Name	**Req. Add Chat Message**
Description	Add a chat message to global or team chat.
Method	GET
Endpoint	/add_chat_message
Parameters	- String token - String id - String password - String message - int type
Response Object from Retrofit:	OkResponse

Name	**Req. Get Chat Messages**
Description	Gets the chat messages from global or team chat.
Method	GET
Endpoint	/get_chat_messages
Parameters	- String token - String id - String password - int type
Response Object from Retrofit:	GetChatMessagesResponse

3.1.2 Server Responses

The responses of the server are JSON Strings, all the responses can be found below in a readable way:

Resp. Account - Response of a change or request of an account information:

- Response
    - ResponseResult
        - int code
        - String message
    - AccountInfoContent
        - AccountInfo
            - Account
            - GameCharacter

- GameCharacter

Resp. Movement Response of a movement request from a character:

- Response
  - ResponseResult
    - int code
    - String message
  - AccountInfoContent
    - MoveGameCharacterInfo (What the character sees when moving, the rewards from any interaction when moving,...)
      - List<Chest> (Chest objects)
      - List<Coin> (Chest objects)
      - List<Crown> (Chest objects)
      - List<AreaEffect> (Abilities)
      - GameCharacter (The character moved)
      - List<GameCharacter> (Other players)
      - List<Key> (Chest objects)
      - List<Portal> (Portal objects)
      - List<Reward> (Result reward from any action)

Resp. Ok Response from a projectile throw or any other confirmation or a chat message is added correctly:

- Response
  - ResponseResult
    - int code
    - String message
  - OkResponseContent

- msg

Resp. GetChatMessages Response from the request Req. GetChatMessages:

- Response
  - ResponseResult
    - int code
    - String message
  - GetChatMessagesContent
    - List<ChatMessage> chatMessages

3.1.3 Useful Libraries

3.1.3.1 GSON

The first example of Object to JSON String:

We have the class Bottle:

```
public class Bottle{
    private boolean filled;
    private String label;
    // Getters and Setters ...
}
```

And to get the JSON String of this class using GSON will be:

```
Bottle bottle = new Bottle();
bottle.setFilled(true);
bottle.setLabel("Pure Water");

Gson gson = new Gson();
```

String jsonBottle = gson.toJson(bottle);

The content of *jsonBottle* is:
{ filled:true, label:"Pure Water" }

The second example of Object to JSON String:

To convert a list to json is the same procedure:

```
public class BottleShelter{
        private List<Bottle> bottles;
        // Getters and Setters ...
}

BottleShelter bottleShelter = new BottleShelter();
List<Bottle> bottles = new ArrayLis<Bottle>();

Bottle bottle = new Bottle();
bottle.setFilled(true);
bottle.setLabel("Pure Water");
bottles.add(bottle);

bottle = new Bottle();
bottle.setFilled(false);
bottle.setLabel("Transparent Water");
bottles.add(bottle);
```

bottle = new Bottle();
bottle.setFilled(true);
bottle.setLabel("Water with salt");
bottles.add(bottle);
bottleShelter.setBottles(*bottles*);

*String jsonBottleShelter = new Gson().toJson(*bottleShelter*);*
And the result of *jsonBottleShelter* will be:
{
bottles:[
{ filled:true, label:"Pure Water" },
{ filled:false, label:"Transparent Water" },
{ filled:true, label:"Water with salt" }
]
}

To parse the JSON and get the Object back:

To get the Bottle back from the first example:

Gson gson = new Gson();
Bottle bottle = gson.fromJson(jsonBottle,Bottle.class);

To get the Bottle back from the second example:

Gson gson = new Gson();
List<Bottle> shelterBottles = gson.fromJson(jsonBottleShelter,List.class);
Those examples show the simplest Object - JSON conversion in Java.

3.1.3.2 Retrofit

These steps will show an example of how to create the simplest access to a service from an Android device using Retrofit:

1. Create a Java interface using Retrofit Annotations, for example:

```
public interface PersonServices{
        @GET("/person")
        PersonResponse account(
                @Query("male") boolean male,
                @Query("age") int age );
}
```

2. Create a Retrofit Adapter:

```
RestAdapter adapter = new RestAdapter.Builder()
   .setEndpoint(http://192.168.1.37:20202/com.edinbros.service/gameloc)
   .build();
```

3. Create the web service object accessor to gain access to the server and get the data (something like RMI):

```
PersonServices personServices = adapter.create(PersonServices.class);
```

4. Get the data from the server.

```
PersonResponse personResponse1 = personServices.account(false,28);
PersonResponse personResponse2 = personServices.account(true,21);
PersonResponse personResponse3 = personServices.account(false,59);
```

5. Note that the response is an object directly, this is because Retrofit uses GSON to automatically parse the server JSON String to an Object automatically.

This way, the usage of the JSON to call the server is fast and easy.

3.1.3.3 Jersey

The Jersey library is a library that recognize classes with the @Path(String) Annotation and when the web service runs, the library attaches these classes to the requests given by the clients. For example, once request is received correctly, the Jersey library looks if the request end (without the parameters) matches the *"/login"* word and after that, it gets the parameters and if the parameters match one method parameters, the method *service(@QueryParam("id")String id)* in this case is executed. The return value is the return response to the client.

In the project, the server response objects are the objects converted with GSON to a JSON String and the return value of the *service(@QueryParam("id")String id)* method below will be the JSON, so the server returns back a response and a client receives the JSON to make it easier overall.

```
@Path("/login")
public class SomeService {
        @GET
        @Produces(MediaType.APPLICATION_JSON)
        public String service(@QueryParam("id")String id) {
                return “...”;
        }
}
```

There are more annotations and options to configure, but to understand the simplicity of the project, the example above is perfect to understand its functionality.

If there is any problem with the requests from the client or any problem from the Jersey itself, it prints and shows the problem very easily in the log/console window.

3.2 Server Structure

The server has a module for the requests and the procedure to update the data:

- The data from the user is taken and a request parser (web service, Jersey) grabs the info and process it. This means passing the data request to the ResponseManger.
- In the ReponseManager, the data is passed to GameMongoDBUtils which cares about containing synchronized data of the world in the MongoDB database.
- Inside the GameMongoDBUtils, there are different classes used to manage each type of object of the game map called (Type)Manager (the type must be replaced for each object: CoinManager, GameCharacterManager,...) which coordinates and synchronizes the data to the database using the synchronized keyword in Java as needed.
- After the operation is completed, the data returns to the ResponseManager and it creates a Response that encapsulates:
    - The code of the operation, which is a number that contains if the operation is successful or not, in this release follows two of the HTTP code results, 200 OK and 400 for error.
    - A message, just a simple String.
    - And the content part that has the response data for the user. For example if the user requests a login and the request information is accepted for the server, the content will be information about the user account, otherwise it will be empty.
- On the other hand, for example the CoinManager extends MapObjectManager, which it has built in basic methods to insert, update, remove elements from the

MongoDB. For that, the architecture is cleaner and the CoinManager can implement only the Coin methods, this means that every object can have its manager and the lines of code to perform an operation is drastically reduced.

- Because every user request runs in a separate Thread, the database synchronization is key. The MapObjectManager in this case locks itself and performs a database operation. The following example shows how to synchronize a call to the database:

```
public String delete(String id){
        Document document = new Document();
        document.put(MapObject.KEY_ID,id);
        try{
        synchronized(this){
                DeleteResult deleteResult = getCollection().deleteMany(document);
                return deleteResult.getDeletedCount() > 0 ? id : null;
        }
        }catch(Exception e){
                e.printStackTrace();
        }
        return null;
}
```

3.3 MongoDB

MongoDB is a NoSQL database which makes JSON a best fit to save data. You can save an Object converted to JSON to into the database so easily within some steps. The decision to choose MongoDB is because there is no need to keep a robust database because the revisions make the project change and if we maintain a robust database schema, there won't be time to do other stuff.

First, we need to install MongoDB and create a database, it is trivial to follow this link: *http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/*

First, the connection to the database must be done using an IP and a Port:

```
MongoClient mongoClient = new MongoClient("localhost",20206)
```

After that, to get an access the objects stored in the database is needed to retrieve the collection object:

```
MongoDatabase db = mongoClient.getDatabase("databaseName");
MongoCollection<Document> collection = db.getCollection("coins");
```

The collection can perform insert, update, find items,... using a Bson request and parsing the Document from JSON String to the desired Object, in this case Coin:

```
String requestId = "asd8731dn0";
Document doc = collection.find(eq("id",requestId)).first();
Coin coin = null;
if(doc != null) coin = new Gson().fromJson(doc.toJson(),Coin.class);
```

The code above will retrieve the Coin with the requestId if found.

3.4 Android Client and Wear

The Android platform is used because it has a lot of support, popularity and mainly it is because it is an open and free to use project. Because of that, a lot companies use

this operative system as a base to contain the applications that can be installed on every Android device that agrees with the AndroidManifest.xml, normally.

Because of that popularity and because its support, for this project as mentioned before, it used a main central device (smartphone or a tablet) and a Android Wear optional companion capable of show the data of the game.

Basically the structure of the client is separated by Android Studio modules and these modules by packages.

The Smartphone module has separated packages including: the map objects, the map objects animations, the client request to the server, utilities.

The Android Wear device also known as smartwatch separates the packages too, but it is much simpler, the device has 3 screens: a main menu referencing the radar and the information about the user character. The radar and the information screens need the smartphone to be playing the game, otherwise it won't update any data.

Fig.8 Menu where the Android Wear app can be started

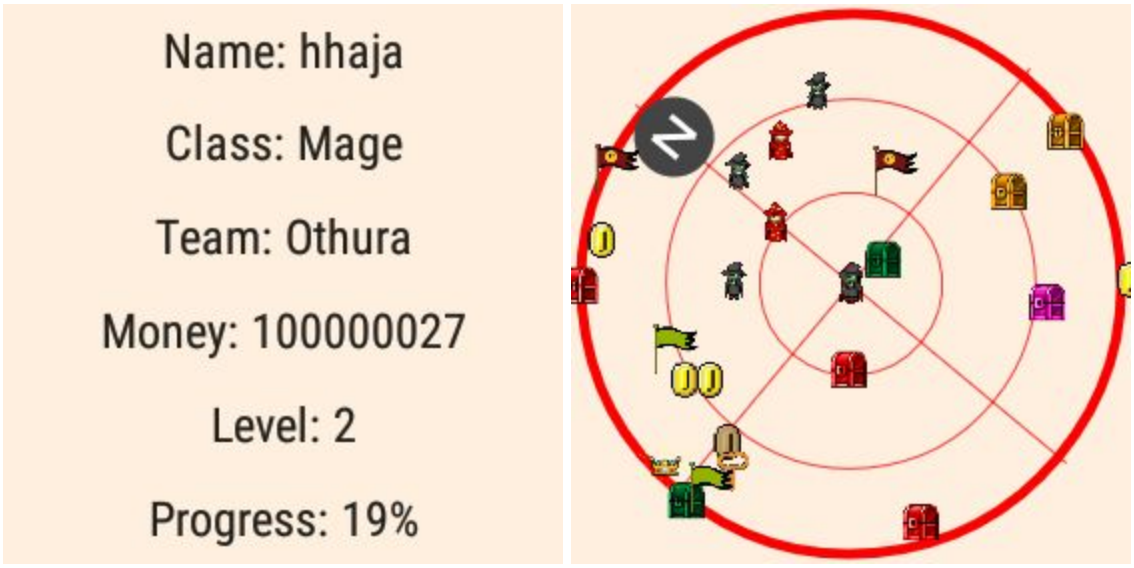

Fig.9 The left is the Information screen and the right is the Radar screen

4. Results and Tests

The procedure done below will identify some parts described in this document, to clarify and tie the concepts. The user grabs the Nexus 5 Android Smartphone, creates an account, creates a character, plays the game and disconnects.

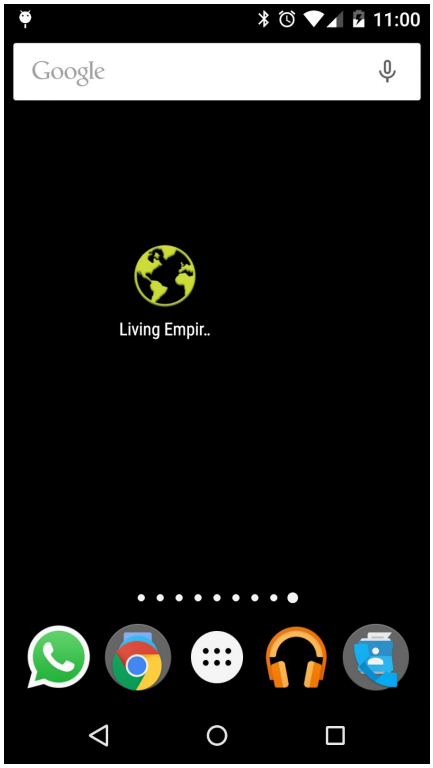

Fig.10 The user opens the application

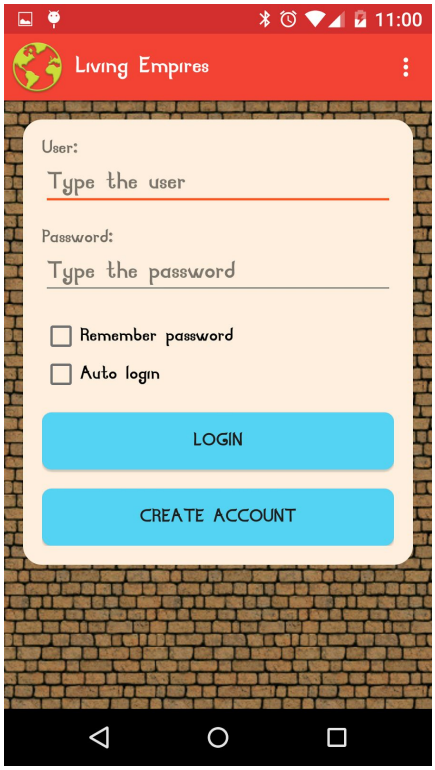

Fig.11 The user sees the login menu

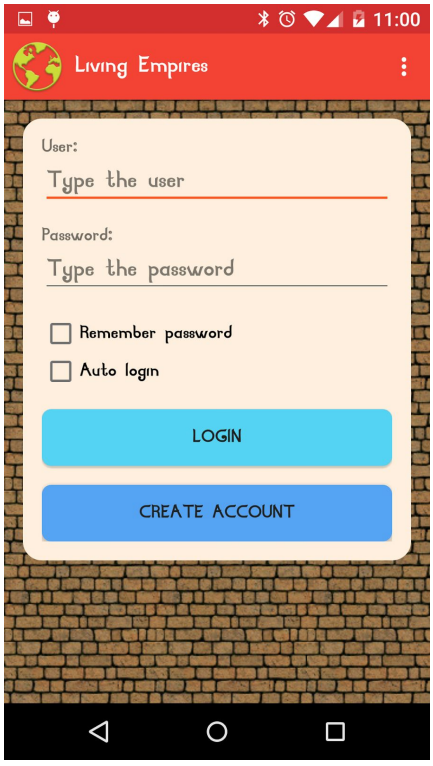

Fig.12 The user clicks Create Account

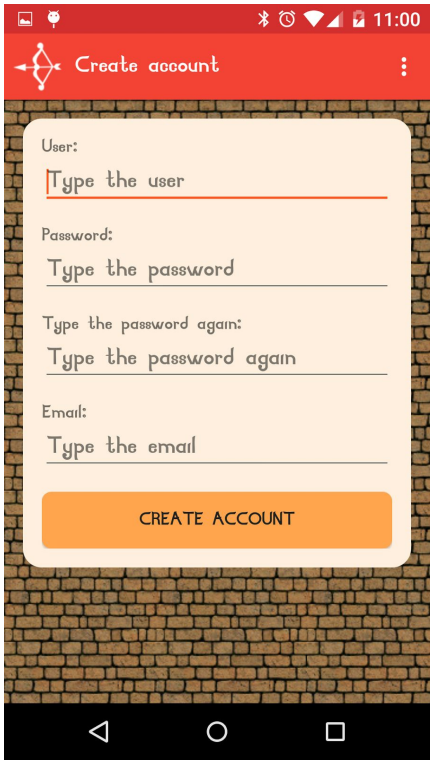

Fig.13 The user sees the Create Account Menu

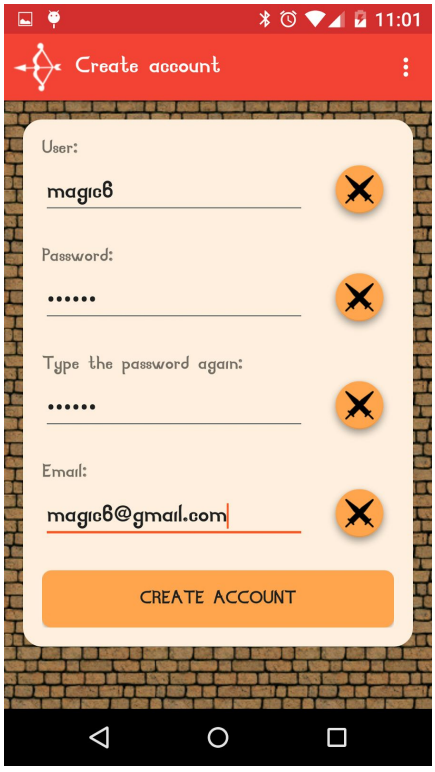

Fig.14 The user enters the data required

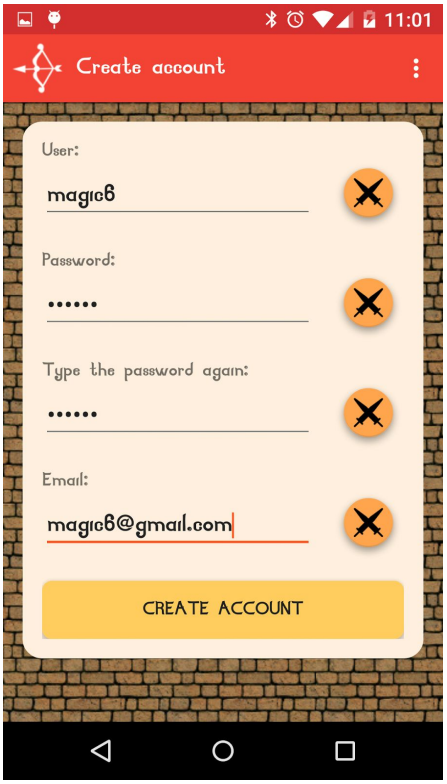

Fig.15 The user clicks Create Account

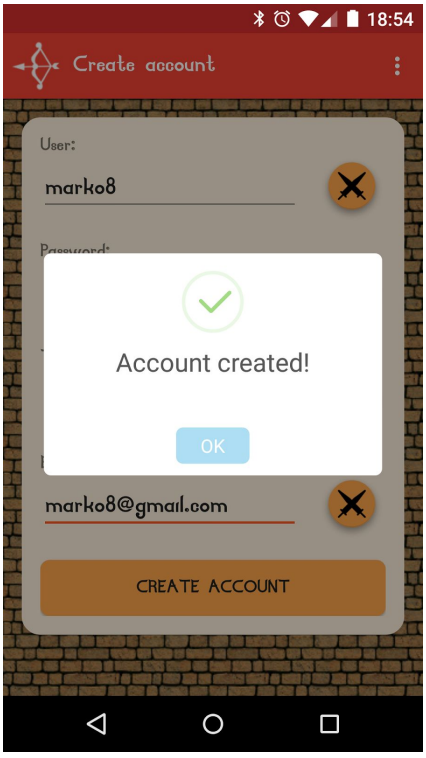

Fig.16 A confirmation dialog shows telling the user that the account is created

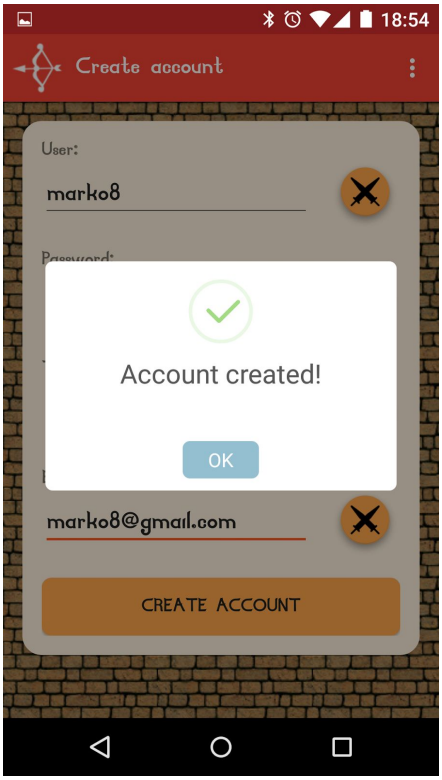

Fig.17 The user clicks OK

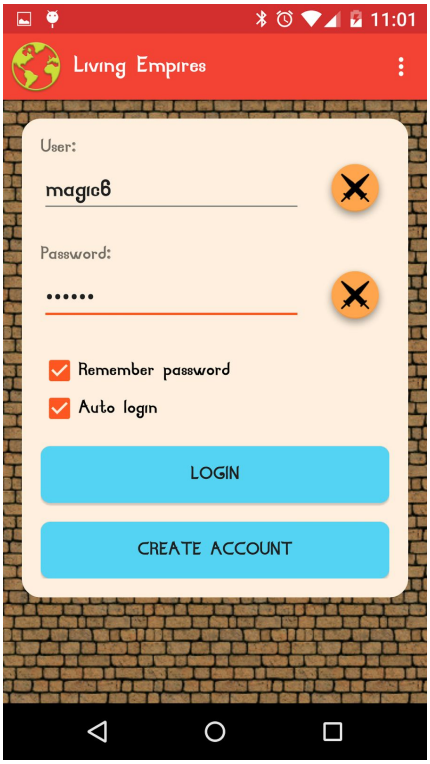

Fig.18 The user is redirected back to the login menu and fills the login data

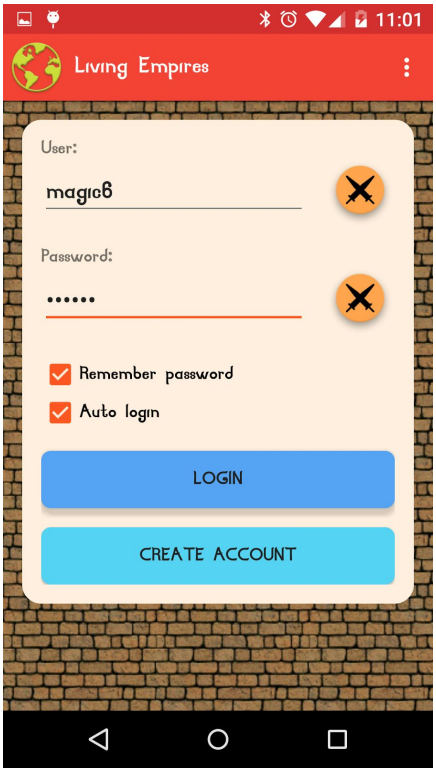

Fig.19 The user clicks Login

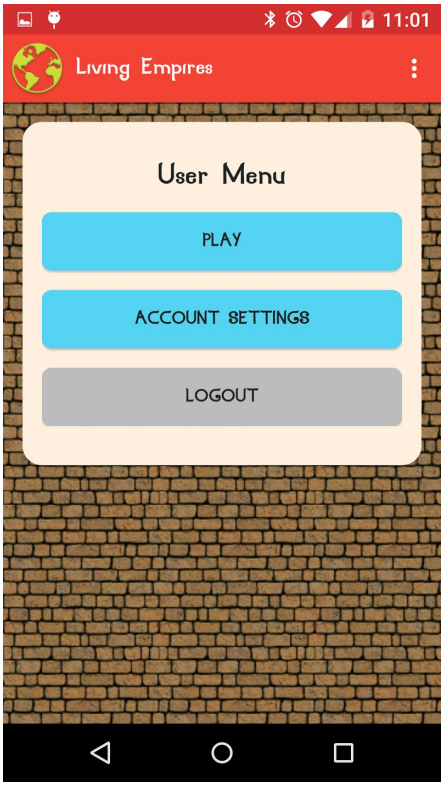

Fig.20 The user is redirected to the User Menu

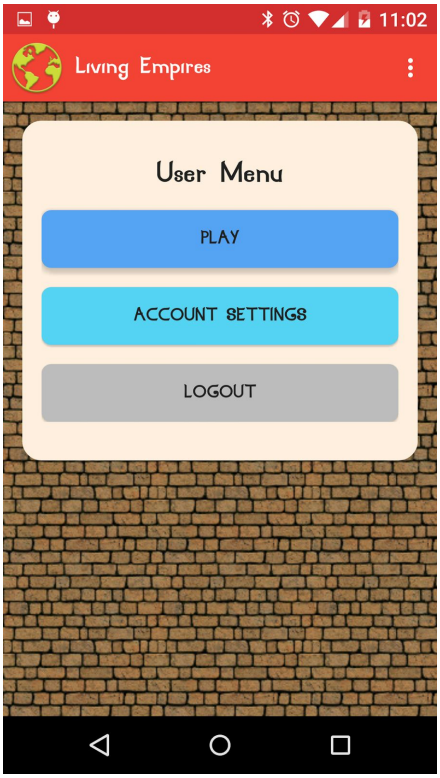

Fig.21 The user clicks Play

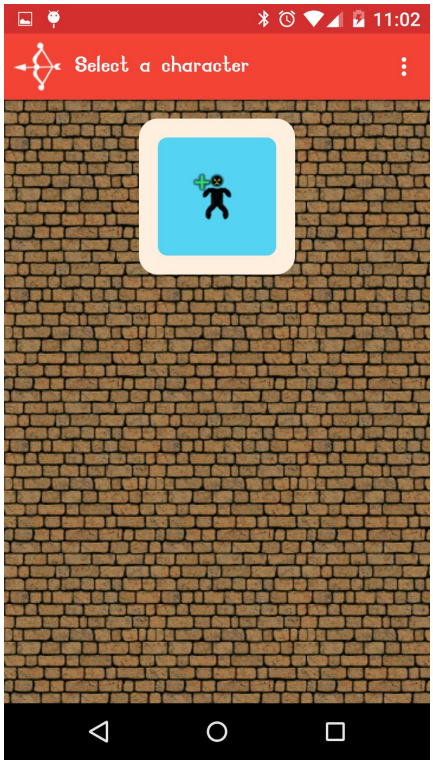

Fig.22 The Select Character Menu appears

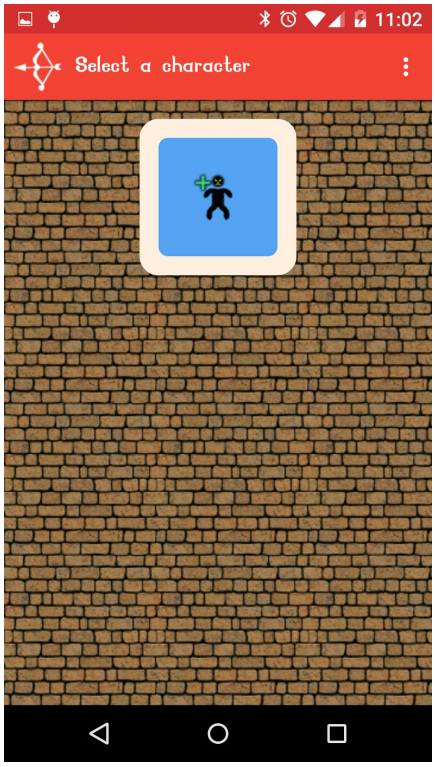

Fig.23 The user selects Create Character

Fig.24 The Create Character menu appears

Fig.25 The user selects a civilization

Fig.26 The user selects the type of character

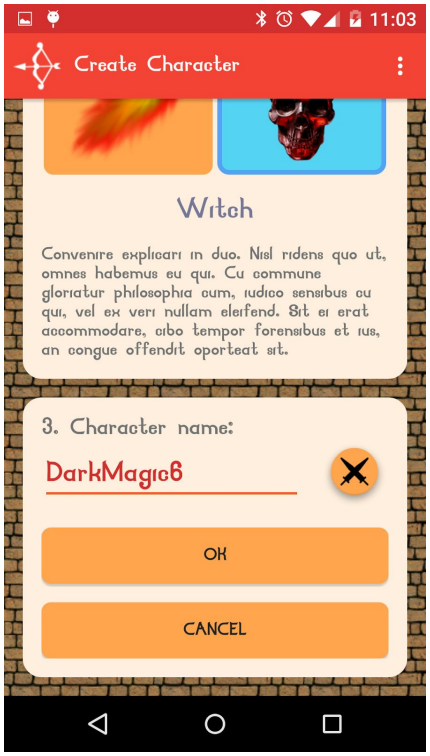

Fig.27 The user enters the name of the character

Fig.28 The user clicks OK

Fig.29 A confirmation dialog is shown

Fig.30 The user clicks OK

Fig.31 A dialog with a message showing that the character has been created

Fig.32 The user selects OK

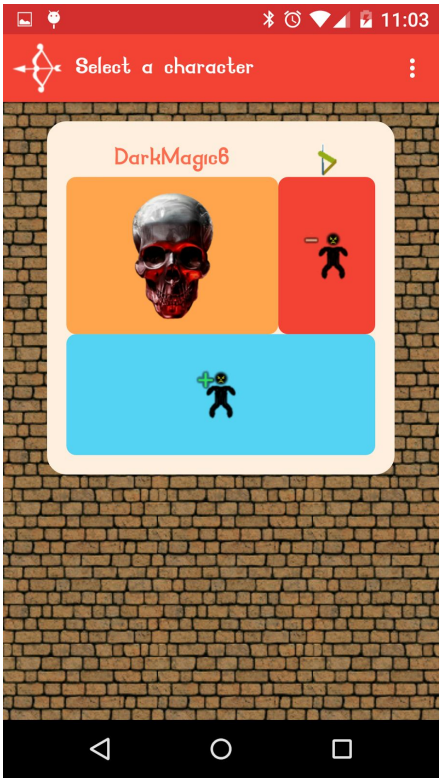

Fig.33 The Select Character menu reappears

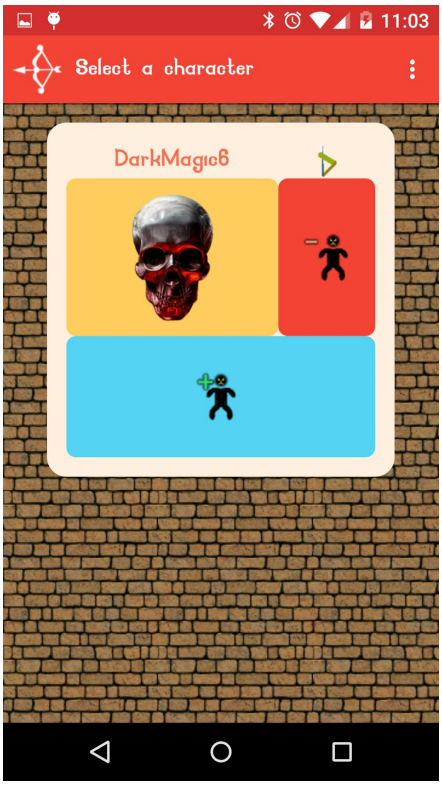

Fig.34 The users clicks the image of the character created before

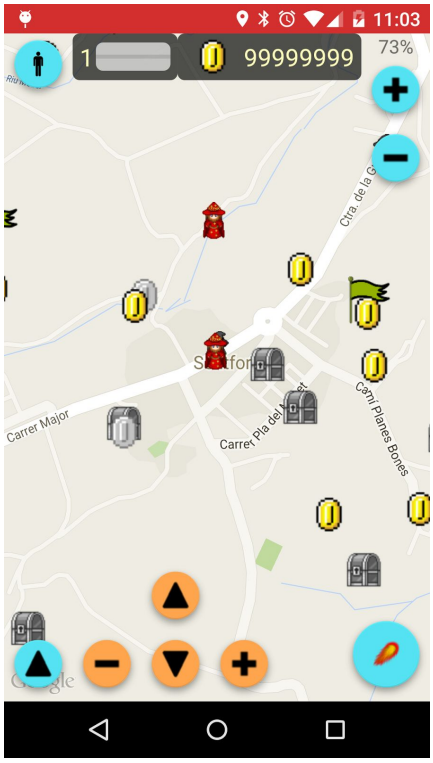

Fig.35 The user now plays the game. The application retrives the position every a few moments and puts the character to the screen.

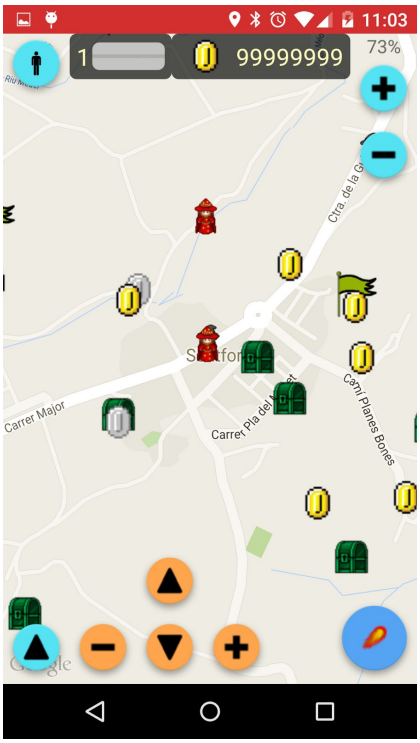

Fig.36 The user presses the button "Throw Projectile"

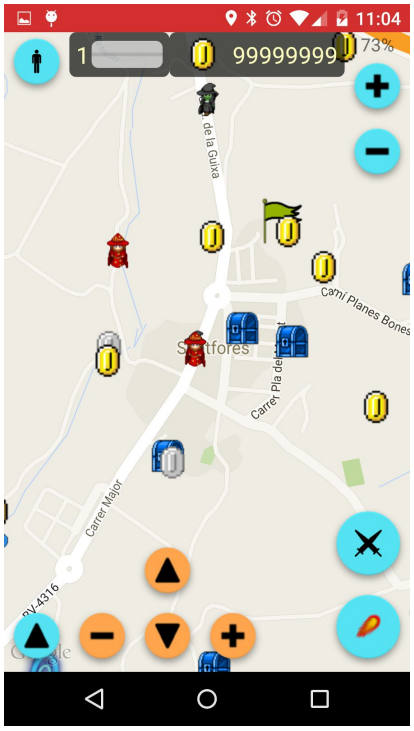

Fig.37 The map rotates according to the smartphone orientation, to aim where the projectile wants to target.

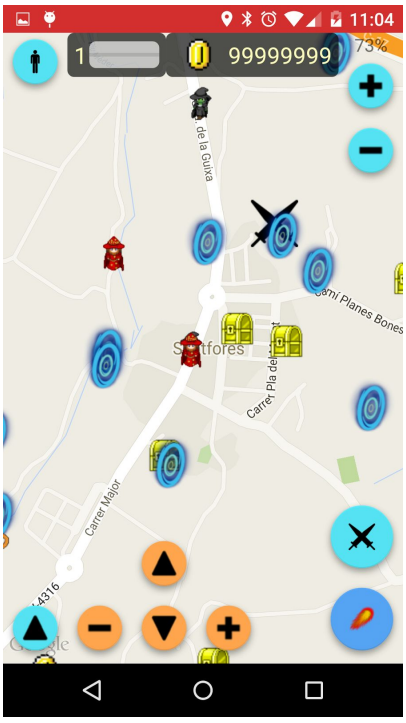

Fig.38 The user clicks the button "Confirm Throw Projectile"

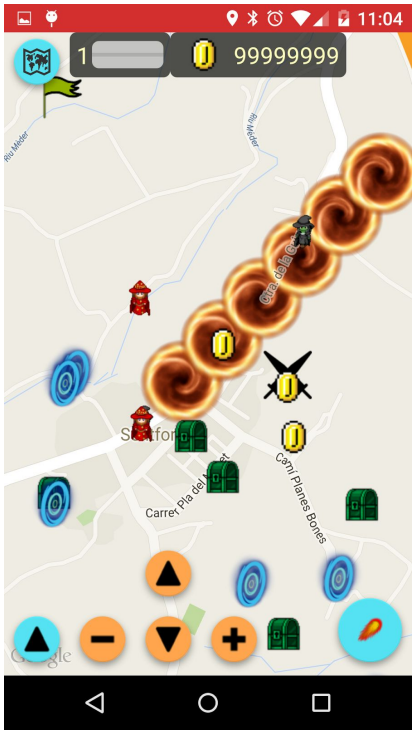

Fig.39 The projectile is thrown at the aimed direction and the screen orientation recovers to the North Pole orientation.

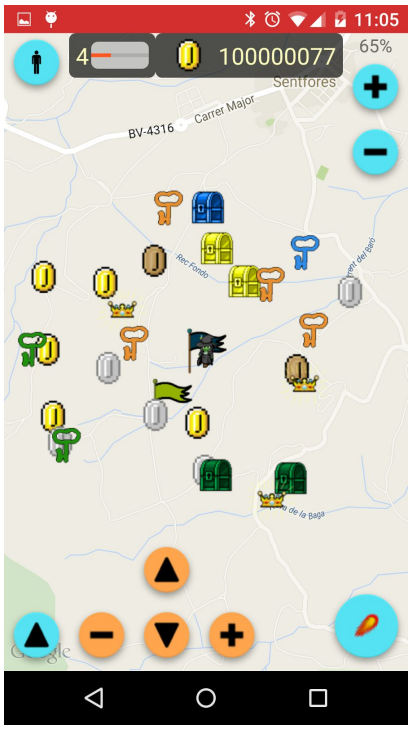

Fig.40 The user captures a portal and the level experience is rising

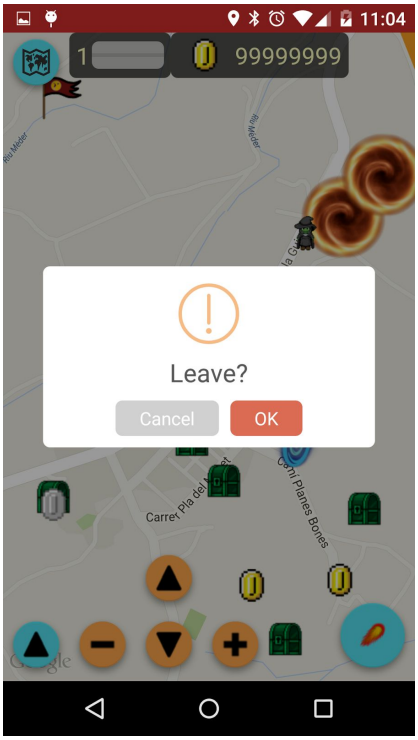

Fig.41 The user presses back and a confirmation dialog to leave the game appears

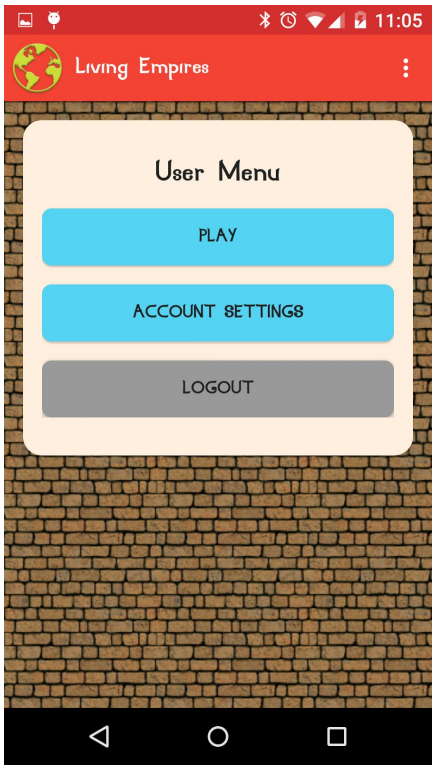

Fig.42 The user is redirected to the User menu and presses logout to disconnect from the server.

Note that this test uses version of the game is not the current release. Some little things changed since then.

5. Future Steps and Conclusions

Clearly, some of the planned future steps are:

- Improve the performance and the usability of this application, this means focusing on the internet connection and handle the speed of the connection correctly.

- Reduce the time that passes between the received data and it is painted to the screen. Some game frameworks can increase a lot the painting performance like LibGDX. The lag found in the application is due to the system of views of Android and its resolution on the screen and if the data is set a lot of times, the screen may freeze some milliseconds.

- The next step to program is the Gardener module. The main functionality of this module is to track the current playing users and add game items to their surroundings, that way the user won't feel alone in the world.

- If there are no users nearby the user may feel alone. If some A.I. is introduced as a module, it can play with the user and make things a lot more interesting, also but Adding an A.I. can affect the gameplay and make it worse, though.

- Some planned actions haven't been implemented yet because there was no time to implement them. For example when a character is capturing a portal, the other users from distance away, can help the character that is capturing the portal by spending money and gaining a temporally boost of capture.

- Another thing out of scope is that there is only two chat rooms for each character: one for each team and one for the global communication. The planned thing was that a character can send a message to another character, this was discarded when the time was not enough to program that.

- One of the planned things of this project is to play absolutely with the smartwatch but the time of learning the software, the evolving software of Google Play Services and that, there were more important things like the server or the client, made this thing not reachable for the moment.

Although the difficulties and the ambitious theme and structure of the project, it reached a level of independence from the server that is a lot appreciated, any user can login from the device they want, even uninstall and reinstall the application and they won't lose any data.

This approach might be what the smartphone users want because any person can do a lot of things in a day and having this interaction with the game (the server cares) does not bother that much.

One of the problems of this kind of applications is that it needs a lot of maintenance, because it is real-time, people interact with other people and maybe there are some problems found in the future and there are a lot of applications out there with this kind of cloud service, so the competence can be found very nearly.

6. Bibliography

[1] Freesound: Open Source Free Music, Creative Commons 0, Barcelona, Spain, www.freesound.org, 2015.

[2] Jose Vidal: REST Tutorial, www.youtube.com/watch?v=rzSKa8qiiMI, 2012.

[3] WebConcepts: REST API concepts, www.youtube.com/watch?v=7YcW25PHnAA, 2014.

[4] Java Brains: REST Web Services, www.youtube.com/watch?v=xkKcdK1u95s, 2014.

[5] Chris Broadfoot: Gradle Please, gradleplease.appspot.com/, 2015.

[6] Lars Vogel, Vogella: REST with Java (JAX-RS) using Jersey, Hamburg, Germany, www.vogella.com/tutorials/REST/article.html, 2012.

[7] Gontuseries: Web Services - How to create RESTful Web Services, www.youtube.com/watch?v=n_agJFIB9bw, 2012.

[8] Lars Vogel, Vogella, Android Sensors, Hamburg, Germany, http://www.vogella.com/tutorials/AndroidSensor/article.html, 2012.

[9] Lars Vogel, Vogella: Android Tutorials, Hamburg, Germany,

http://www.vogella.com/tutorials/android.html, 2015.

[10] Lars Vogel, Vogella: Eclipse IDE, Hamburg, Germany,
http://www.vogella.com/tutorials/eclipseide.html, 2015.

[11] Lars Vogel, Vogella: Eclipse, Hamburg, Germany,
http://www.vogella.com/tutorials/eclipse.html, 2015.

[12] Lars Vogel, Vogella: Web, Hamburg, Germany,
www.vogella.com/tutorials/web.html, 2015.

[13] Lars Vogel, Vogella: Apache Tomcat, Hamburg, Germany,
www.vogella.com/tutorials/ApacheTomcat/article.html, 2014.

[14] MongoDB: MongoDB Configuration,
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/, 2015.

[15] MongoDB: MongoDB Driver Quick Tour,
http://mongodb.github.io/mongo-java-driver/2.13/getting-started/quick-tour/, 2015.

[16] MongoDB, GitHub: Mongo Java Driver,
https://github.com/mongodb/mongo-java-driver, 2015.

[17] MongoDB: Java Mongo Driver,
http://docs.mongodb.org/ecosystem/drivers/java/, 2015.

[18] Android Developers: Samples,
http://developer.android.com/samples/index.html, 2015.

[19] Android Developers: Reference,
http://developer.android.com/reference/packages.html, 2015.

[20] Gimp: Gimp for Windows,
http://www.gimp.org/, 2015.

[21] OpenGameArt: Open Art for Developers,
http://opengameart.org/, 2015.