Location-aware Real Time Strategy Games
Master Thesis
Winter Semester 2012 - Summer Semester 2013

Vadim Costache

May 10, 2013

Contents

1 Introduction: Real Time Strategy games

The purpose of this paper is to describe the development of a new type of GPS-enabled mobile game, a Real Time Strategy / Shooter hybrid. The motivation for this came up during the search for GPS-enabled multiplayer game concepts.
Ever since the first operating system for handheld devices, the spreading of smartphones has intensifed every year.In 2012, the number of smartphones in the world has reached one billion, according to Bloomberg(1)

The rapid expansion of mobile computing presents new challenges and opportunities for both the user and the developer. The ease of use and the presence of touchscreens, GPS receivers, gyroscopes, accelerometers and, since recently, even barometers gives way to new approaches in developing games.

In this paper, we will define a number of computer game genres, classify some of the most popular location-aware games and game platforms and present the evolution of the concept and development of a game type prototype. This game of a genre that has a long legacy among computer games, but is not yet known to the location-aware mobile context. The new genre is that of 'Real Time Strategy'(RTS)(2) games. We will do that with a subgenre of RTS, called Real Time Tactics(RTT)(3). For reasons that will be presented throughout this paper, the type of game can be considered an RTT / Shooter hybrid.

1.1 What is Real Time Strategy?

RTS games are defined as real-time (continuous time) competitive games, in which several players fight against each other, either in a skirmish or team versus team. The purpose is to defeat the opponents by taking real-time decisions on managing resources and troops. The focus is, therefore, on three key elements: **managing resource gathering**, **building a base to provide troops** and **battle tactics**.(2)

This project will focus on a subgenre of RTS, Real Time Tactics(RTT) - which, instead of managing all three aspects of RTS, focuses on **battle tactics**.

Because it is essentially a simplified approach to RTS(3), RTT can provide a good proof of concept and at the same time simple and enjoyable playability. The advantage of having a RTT GPS-enabled game versus RTS is that it greatly reduces the play time, requires less skills and has the potential of being less stressful than RTS.

1.2 Real Time Strateg versus Turn Based Strategy

Turn Based Strategy games(4)(such as chess and board games, for example) allow the player to take his time and plan every move. Implicitly, the duration of the game is greater. This type of game is not in the scope of this project, yet it deserves mentioning, as there are many such games for mobile devices. Some notable implementations are the ones of Scotland Yard(Ravensburger GMBH), Catan(Catan GMBH) and Monopoly(Hasbro, Inc.).

In particular, there are two implementations for Scotland Yard upon which a comparison has been made on whether it is better to port a board game in a real-time(continuous-time) or turn-based mobile GPS-based game(5). These two implementations are Mobilis XHunt(turn based) and MisterX Mobile(real-time). The comparison has been made on 10 aspects: Fun, Smooth Progression, Dynamic Gameplay, Ease of play, Stressless Gameplay, Communication, Strategy, Clear Rules, Low Risk, Education. The conclusion was that there was no favorite between the two, but it has been concluded that these 10 aspects have different weights for the player and that Fun, Smooth Progression and Dynamic Gameplay have a higher individual weight to players than the rest.(5, p. 5)

Based on the knowledge gathered from the above-mentioned comparison, the aim of this project will be to maximize the interactivity and dynamics of the RTS game.

2 Location-Based Augmented Reality Games

Ever since the first operating system for handheld devices, the spreading of smartphones has intensified every year.

The rapid expansion of mobile computing presents new challenges and opportunities for both the user and the developer. The ease of use and the presence of touchscreens, GPS receivers, gyroscopes, accelerometers and, since recently, even barometers gives way to new approaches in developing games.

Although it is one of the oldest additions to the smartphone, the GPS-enabled smartphone is still not ubiquitous. As of now, each company's flagship and most of their mid-level smartphones are GPS-enabled. This makes way to the propagation of GPS gaming. Although the concept is old, very few attempts have been made in this direction and this branch of game development may be considered to still be in one of its early stages of maturity.

This paper is a proposal for extending the genre of mobile real-time multiplayer location-based games.

2.1 Location-based games

Location-based games take advantage of the mobile devices' built-in receivers for global positioning. They provide the user's location with an accuracy ranging from a few to a couple dozen meters. Because the most mobile devices in the world today rely on the Global Positioning System(only recently support for GLONASS has been added to smartphones), we can use the term GPS-games.

GPS-games came up long before this feature has become ubiquitous in mobile phones and tablets. One of the first widespread GPS-games is Geocaching. It is composed of two parts:

a. Placing physical caches at various locations that can be considered interesting or worth visiting and publishing their GPS positions (eg. on websites).

b. Searching for various caches by using a GPS device.

Along with the evolution of smartphones came that of the mobile games. GPS games come in a lot of flavors, from GPS-based tours, adventure and investigation games to various race games - single and multi player and massively multiplayer online games.

2.2 Types of GPS-based games

There is a number of GPS-enabled games and game authoring tools that are available for iOS and Android devices. The ones studied for this paper are : ARIS, Tourality, Wherigo, conTAGion, Shadow Cities, SCVNGR, Please Stay Calm, Parallel Mafia, Parallel Kingdom, Tripventure, Warfinger, Totem, Portal Hunt, aMazing, Ingress, MobileWar, Mister X Mobile, Mobilis XHunt, Own This World, MapAttack.

For better understanding the classification done below, we will first define each type of game:

a. **Adventure/Investigation Game** - Game in which the player plays the role of a character in a story. The primordial characteristics of this genre is that it is focused on immersion in the story, puzzle solving and investigation, rather than on physical skills. Also, the tendency of this genre is towards single player experience, though occasionally multiplayer is also implemented (eg. the ARIS-based game 'Mentira').

b. **Massively Multiplayer Online Game** - This type of game is designed to support large numbers of players (even in the number of millions in some cases) that play and interact in a persistent virtual world. This type of game allows both cooperative and competitive gameplay and is exclusively based on multiplayer. Subgenres include MMO Role Playing Games and MMO Shooters.

c. **Casual Game** - Analogous to the MMO, the Casual Game is targeted at mass at a mass audience and can incorporate any type of game type. The particularity of this genre is that it aims at having simple rules, simple gameplay and requiring no specialized skills.

d. **Racing Game** - It's a genre defining a broad range of games. In the case of computer games, it describes mostly motorized vehicle racing. In the mobile context, it mostly describes racing on foot against time or through a number of checkpoints.

e. **Shooter Game** - This one's a subgenre of action games. It focuses on first or third person experience, speed, aiming and reaction time. Usually the weapon is ranged, although close-combat weapons are included in most games.

We will now classify the games/platforms based on the genres they best fall in:

a. **Adventure/Investigation Games**

   (a) ARIS
   (b) Wherigo
   (c) Tripventure
   (d) Tidy City

b. **Massively Multiplayer Online Games**

   (a) Shadow Cities
   (b) Please Stay Calm
   (c) conTAGion
   (d) Parallel Mafia
   (e) Parallel Kingdom
   (f) Portal Hunt
   (g) Ingress

c. **Casual Games**

   (a) SCVNGR
   (b) Warfinger
   (c) aMazing
   (d) Own This World
   (e) MapAttack

d. **Racing Games**

   (a) Tourality

e. **Shooter Games**

   (a) MobileWar

A special category is represented by Mister X Mobile and Mobilis XHunt, which both bring a board game (Scotland Yard) to the mobile environment. While the former adapts the board game to real-time gameplay(placing it closer to the 'Multiplayer Racing Games', with elements from 'Real Time Strategy Games'), the latter falls in the definition of 'Turn Based Strategy' games.

3 Real Time Strategy Games

This proposal is for the research and development of GPS-enabled Real Time Strategy games. They are to be augmented reality games for single player or multiplayer competitive 'free for all'/'skirmish' and 'team versus team' games. This category of games offers opportunities to also enhance the experience for all the previously existing types of GPS-enabled mobile games.
For this project, the proposed games are :

a. **Territory Takeover**

b. **The War Game**

1. The **Territory Takeover** game is a multiplayer, team versus team competitive game. The players or game author define an area of play, which will be divided into multiple divisions. Each division will be marked by a 'flag' (a GPS marker). To capture the area division, a team must capture its flag. The game ends when all flags have been captured and the winner is the team with most captured flags. Each flag may be given a time that a player must spend next to it in order to capture it. Once a flag (and implicitly the territory) is captured, it remains so until the end of the game. The winner can be decided on flag counting or, alternatively, each flag may receive a number of points, according to the size of the territory marked by it and the difficulty of the terrain.

This game can be enhanced with the use of virtual tools or weapons. For the purpose of this project, the following tools/weapons have been considered :

a. The **Immobilizer** is an ability that can be used by each player to block an opponent from moving. The 'attacker' 'activates' the ability and a circle around him is drawn to show the range in which he can shoot. If an opponent enters the range area, the 'attacker' will select him on the map and shoot. The 'victim' will receive a notification that he is immobilized. A circle or rectangle will be defined around him and he will not be allowed to move outside of it for a given time, say 30 seconds or 1 minute. If he does, he gets disqualified and kicked out of the game. An alternate solution would be that the team loses points, for the case that this is the scoring methodology implemented.

b. **Demobilizer** is an ability that an immobilized player can use. For this project, it will only work on the person that uses the ability. The effect is that a person that is immobilized gets the waiting time halved.

Both the abilities have a common cooldown timer. That means that if a player immobilizes somebody and is immediately immoblized himself, he won't be able to use the demobilizer because of the cooldown following the usage of the immobilizer.

2. The War Game is inspired by Real Time Strategy Games and Airsoft. Two teams are formed. An area of play is delimited on the map. Each player can choose between a number of characters. For the purpose of this project, four characters are proposed: Defender, Marine, Sniper and Heavy Trooper. Each of the four characters has special abilities and characteristics :

a. The **Defender** has the ability to generate shields for short periods of time. Members of the team can hide behind those shields for defence. The Defender may also act as a Medic and heal or revive members of the team. He has low health, long ability cooldowns and a sidearm with short range, small damage and fast cooldowns .

b. The **Marine** has a weapon that can shoot a medium range with medium damage and fast cooldowns. He has medium health.

c. The **Sniper** has two weapons : the sniper rifle that can shoot at distant ranges and deal large damage to single targets and the sidearm, which is the same as the Defender's. His health is low, just like the Defender's. The sniper shot may penetrate the Defender's shield and cause reduced damage to one target.

d. The **Heavy Trooper** has three weapons : the bazooka, the sidearm and mines. The bazooka is a mid-range weapon with splash damage - it therefore can be fired against compact groups, such as the ones that might be hiding behind a shield. The bazooka cannot deal damage through the shield, but it may be shot next to it, causing damage from the side. The damage to each target varies from moderate to small, depending how far they are from the center of the 'projectile explosion'. The mines can be placed randomly on the map and their 'explosions' will not affect the members of the Heavy Trooper's team. Also they cannot be triggered by the members of his team. The damage dealt will be moderate, with splash damage, just like the bazooka projectile. The bazooka and the mines have long cooldowns, therefore the sidearm is added. The Heavy Trooper has high health.

This game has some advantages and drawbacks:
Advantages: It does not require specialized gear and setting, nor does it need long amounts of time to be played. It can be enjoyed with a bunch of friends on a sunny weekend afternoon.
Drawbacks: It highly depends on GPS accuracy. This issue may affect gameplay. It also does not feel as 'real' as real-life games, nor PC games.

4 Requirements

The two above-mentioned games will use a common framework that will enable multiplayer interaction for both 'free for all'/'skirmish' and 'team versus team' approaches. They will require a server to centralize player information such as GPS data and the virtual 'health' attribute. Because the games proposed are fast-paced, they require quick response times from the server, client and and the communication protocol between them.

4.1 Client

The following technologies have been considered for developing the frontend for the application : Android, XCode(for Apple iOS) and multiplatoform APIs such as PhoneGap/Cordova and Titanium. The multiplatform APIs offer the benefit of the 'code once, deploy everywhere' philosophy, at the cost of performance(6)(7)(8). In this case, the approach of using a platform-agnostic framework is disadvantageous. Therefore, the choice of native app development makes more sense for this situation.

The client will be developed with Android, due to its accessibility and versatility and the fact that it's ubiquitous and open source.

4.2 Server

The games that are to be implemented will actively rely on communication with a centralized server. This implies a constant Internet connection on each mobile device. Also, they require fast, low latency message exchanges. The server should be able to handle this for players in the numbers of a few dozens.

The server will be written using NodeJS(9). NodeJS is a javascript framework based on the Google V8 Engine that is designed for web server functionality. Yet, due to the fact that it is a framework and not a standalone server, one can take advantage of the 'Event Loop' functionality to write various other types of servers. This particular feat is beneficial to the purpose of this project. It is also faster than Apache for general purpose, fast message exchange scenarios(10).

4.3 Communication Protocols

During a game, active communication must be performed between the mobile devices and the server in order to multicast all player positions and intentions to all players. There are two aspects to this approach:

a. The game is created on the server, in a so-called 'Lobby' which all the players join.

b. The game is created on the server and once the game starts, all players send their positions to the server. The server is then responsible to multicasting the positions to all the players.

This is, of course, a subset of the tasks that must be performed by the server. There will be other information that must be exchanged periodically, such as the player's health and quantities of various other attributes that will be added during development and/or proposed during trying out the game. In the general idea of communication protocol usage, the list of choices has been narrowed down to three:

a. WebRTC - A protocol that is to be part of the HTML 5 standard. It will be based on the RTP(Reliable Transport Protocol), which is the base for VoIP protocols and is itself based on UDP. It promises to be a very fast standard protocol, appropriate for audio and video streaming and massive multiplayer games.(11) It is still in draft format and there are no official implementations for it. Implementing the protocol itself is outside the scope of this project.

b. WebSockets - A protocol that is part of the HTML 5 standard. It is a low latency TCP-based protocol that promisses to replace http in several types of web applications.(12)

c. RTP - The protocol on which VoIP and WebRTC are based. It is based on UDP and it is designed for real-time streaming of data.(13)

From these three protocols, a choice will be made between WebSockets and RTP. WebRTC is to be left as an option until its official release and implementation for both NodeJS and Android.

4.4 Game Elements

Both games share a number of common features:

a. a number of players that are visible on the map

b. a number of teams, represented by different colors(in the case of free-for-all(skirmish) play, the number of teams equals the number of players)

c. a number of items that each player can use

d. item properties, such as range and cooldown times

e. player health

The 'War Game' also introduces a number of characters to the game. Each character type has a different health value and different tools, with specific properties.

The server must provide a 'game lobby', to which the players connect and prepare for starting the game. This includes character choice for the 'War Game' and a 'Ready' button. When all players declare themselves to be 'ready', a countdown starts and then the game begins.

5 Implementation

Implementation will mean developing a server and a client application from scratch, covering all the functionality needed for the games to work according to the description in section 'Real Time Strategy Games'.

5.1 Schedule

This project, consisting of one server and one client application, is to be developed in four steps:

a. **Development** - During the first iteration of development, the most basic features of the game are to be implemented: basic server functionality that would allow the game to work, basic client functionality and the 'War Game' without all features.(2 months)

b. **Testing and Evaluation** - During this phase, the game and its functionality will be live-tested for feasibility and quality. New ideas will be sought and documented. Most importantly, player feedback will be gathered.(1 month)

c. **Development** - During the second iteration of development, the 'War Game' will be completed and, using its framework, the 'Territory Takeover' game will be implemented. Bugs will be removed and tweaks will be made to the framework and the game concepts to match the player feedback.(2 months)

d. **Evaluation and Completion** - During the second evaluation phase, both games will be tested for playability, player feedback will be gathered and the Dissertation Paper will be completed.(1 month)

5.2 Authoring Tool

As it is not the focus of this project, the Authoring Tool will comprise the most basic elements necessary for setting up the game(such as defining obstacles and probably the border of the gameplay area). Future work might include modifying and/or adding new character attributes, multiple obstacle types and probably extending the RTT games to full-fledged RTS.

6 Documentation on the Games

Extensive documentation has been found from research done on education-oriented GPS-games.(14)(15)(16)(17)(18)(19), describing concepts and approaches in developing platforms and games to this end(16), porting them to different locations(17), comparing them and discussing their functionality(19)(14)(15) or discussing their effect and usefulness(14). Unfortunately for the direction of this particular project, this documentation focuses exclusively on GPS-enabled Adventure Games.

The lack of documentation for the other games, except their websites has left trying them out one by one as the only option to understand their components and functionality. Additional information was retrieved from video descriptions, examples and reviews of the games.

During the preparation of this proposal paper no games or documentation have been found on GPS-enabled Real Time Strategy games. From the searches performed, no location-aware mobile games have been found to fall in the genre of RTS. However, one has been found in the genre of 'Shooter Games' - MobileWar.

The purpose of this project is to create a framework on which the GPS-enabled Real Time Strategy games 'Territory Takeover' and 'The War Game' will be implemented as proof of concept. This framework is to be extended during future work. The names of the games themselves are chosen only by their descriptive nature and are prone to change during the steps of development and evaluation.

7 The project : obstacles and changes

The two games are proposed with group teambuilding and recreation in mind. Both games require team strategy and cooperation. Territory Takeover allows for both team versus team and free for all gameplay, allowing for both small and large groups to play. The War Game is to be a fast paced game spanning a time interval in the range of a few tens of minutes. Although it is a RTS Game and not a Shooter Game, the 'War Game' can be seen as a more casual approach to Airsoft. Where it lacks the realism of Airsoft or the immersion of classical computer Real Time Strategy games, it gains in the intensity of real-life experience and teamwork, without requiring specialized equipment(Airsoft) or highly developed skills(computer Real Time Strategy games).

8 The development of the game

This section will follow the development of the application. The backend, frontend and mechanics of the game will be analyzed and presented in their evolution.

The game is composed of three parts :

a. **Mobile App Frontend** - A set of menus for setting up the game and a GUI on top of Google Maps for the game itself.

b. **Mobile App Backend** - The core of the app. This includes the modules for communicating with the server, for in-game protocols and mechanics.

c. **Server** - All players will communicate with each other through a central server that can only host one game at a time.

8.1 Initial plan

The original plan was to develop and Android and iOS app, using a development framework such as PhoneGap. This would assume Javascript / JQuery programming for the UI and application backend. The communication was planned to be established through Websockets. Therefore, for optimum performance, NodeJS was to be used on the server side. For the location presentation, the Google Maps API was to be used.

8.2 Unified vs. Native Frameworks

Once an idea of how the game should look like was crystalized, I started looking for unified frameworks for cross-platform mobile development. The first choice was PhoneGap, which I have used in the Agile Lab in the previous summer and was a bit familiar. The other frameworks that I considered were Titanium SDK, Sencha and Corona. The discussion on which one is better can be resumed to the conclusion that none of them is appropriate for the development of such a game. And here are the reasons :

a. The game is fast-paced and will require a fast backend as well as a frontend that is as fast as possible. The cross-platform frameworks essentially interface only some of the native functionality and display the app through a webview - requiring Javascript or a Javascript-based framework for creating the interfaces. I have tested some these frameworks on top of Phonegap, during the Agile Lab. The purpose was then to find the fastest solution for static menus and slow-paced UIs. Even then, the outcome has been disappointing. Qooxdoo, ExtJs and JQuery were tried out. Qooxdoo performs faster than the other two, but makes it very slow and tedious to develop a complex UI. Still, it is slow for the purpose and feels nonnative. After reading a few articles that compare Phonegap, Titanium, Sencha and Corona, I have concluded that with or without various advantages and disadvantages between them, they are all similar in performance - and therefore too slow for the development of this game.

b. Being a game prototype, all the details on which features of the phone or of the operating system will be used were unclear. I found it unwise to develop on a platform that only permits a number of features that are common to all platforms.

c. Developing native code can prove to be easier, as for example the Android development community is much larger than that of Phonegap developers. The support is both more intensive and extensive for native platforms.

From this point on, I gave up the unified frameworks and had to decide between iOS and Android development. This was an easy choice : Android develpment is free, while iOS development requires a paid developer account and XCode runing on MacOS exclusively. Therefore, I chose Android.

8.3 Communication Protocols

Initially, I have planned to use Websockets for the client-server communication. The reasoning behind it has two main arguments :

a. Websockets is an HTML5 protocol currently presented in draft by the IETF. This protocol provides reliable two-way communication between a server and a client and manages various complex issues or network features that come above TCP. This future standard is developed to replace HTTP and add functionality for technical aspects that HTTP was not covering. The reason for using Websockets for the client-server communication was that it promises to abstract a lot of protocol management issues, while offering speeds close to plain TCP. Also, for the game to run properly, UDP and its child protocols cannot be used - total reliability is required in the communication.

b. Websockets communication can be implemented in NodeJS, which is a framework well suited for fast-paced message exchanges and which has been proven more scalable than, for example, Apache Tomcat.

As I decided to develop the server in NodeJS, the first choice was to use the Socket.IO Websocket plugin. For the client side, I chose Autobahn for Android. The only alternative for Autobahn was not free. After developing a basic Websockets client with Autobahn for Android, I realized that they do not communicate. After a lot of searching, I found out that the protocol draft version that Socket.IO is using is different than that used in Autobahn, and unlike Autobahn, Socket.IO uses an HTTP handshake for establishing the connection. The Websocket-Node and ws NodeJS libraries for Websocket communication were tried out afterwards. Neither were compatible with Autobahn for Android. Then, I have switched to Python and after some technical difficulties both in the Autobahn libraries for Android and establishing the communication between client and server, I decided to give up Websockets and start off with pure TCP. Because writing code in two different languages feels slower, I have switched to Java.

8.4 Google Maps API V1

The first version of the app used Google Maps API V1. I chose it because it has more extensive developer support. Unfortunately, the Google Maps API and the Android Support Library cannot work together at the same time, because they need to subclass different Fragment classes.

8.4.1 Map Overlays

The Google Maps V1 API supports the use of Overlay objects to draw on top ofthe map. Most tutotials found online make use of the so-called ItemizedOverlay, which enables easy integration of multiple markers. I have attempted to use this, but here are my conclusions :

a. The ItemizedOverlay uses features that we may call 'magic' - the programmer is kept at arm's distance from understanding exactly what they do. The ItemizedOverlay object uses an ArrayList for storing the map markers as OverlayItem objects. It also needs a function that gives it the size of the ArrayList.

b. Then, by unknown means, the markers are redrawn onDraw. This has proven very difficult to work with. For starters, there is no control over the draw process. Then, the OverlayItem objects cannot be stored in another data structure, such as a HashMap - which I ended up using.

c. Because of its design, the ItemizedOverlay is fast, but useless for the purpose of this app. I chose to replace it with a custom Overlay.

As a replacement for the ItemizedOverlay, I have started to work on how to create a custom Overlay that would fit my needs. I realized how simple it is to modify the onDraw() method and control the display of the markers. This has also allowed me to change their Drawables(icons) dynamically, according to my needs. I have added functionality to change the Drawable for a selected marker or for the marker of a dead character.

A real challenge was to add proper onTap() functionality for the custom Overlay. The advantage of the already-given-up ItemizedOverlay was that it handled marker touch events. The new, custom overlay had no such thing implemented. What I have done was to get the pixel resolution of the screen, along with pixel density data from the system and consider a 0.2 inch radius around the center of the touch on the screen (this is what I have estimated to be a circle to describe the tip of an average human adult finger). That 0.2 inch radius in pixels has been translated in a radius, in meters, on the Google Map. All markers inside that radius were considered and the closest to the center of the touch was chosen. This made choosing a marker out of both crowded and loose situations relatively easy and natural. The tutorial for this will be added as an annex to the paper.

8.5 Google Maps API V2

The use of the Google Maps API V1 ended when I realized that it is not compatible with the Android backwards compatibility libraries - that enable the use of Fragments, for example, in Android versions prior to 3.0. The compatibility libraries require the use of a custom FragmentActivity and custom Fragment, FragmentManager, FragmentTransaction objects.

The statistics given by Google about the versions of Android in use were underrepresenting the situation that I faced when trying to test the app. Only one of my friends and acquaintances had Android 4.1. The rest carried mid-range phones with Android 2.2 - 2.3. I convinced two friends to let me upgrade their OS versions to 4.0, even if they were officially unsupported. This was tedious and risky. Other friends did not accept such a thing. This led me to realize that it is unfeasible to test the app, when it was supporting Android versions starting at 3.0.

With this new information in mind, I started the migration to Google Maps API V2, which, instead of requiring a MapActivity as wrapper, is using MapFragment. Therefore, I could use both the support library and the Maps V2 API, which is included in the Google Play Services libraries.

8.5.1 Maps API V1 vs. V2

The migration to V2 is very destructive and at first feels quite unnatural. The entire logic is changed. In V1, the map is rendered through a MapView object that can be manipulated in a more direct and intuitive manner - lower level access is both possible and needed. In contrast, the V2 map is accessed through a GoogleMap object, which is no longer subclassing View. Therefore, it cannot be manipulated in the same manner. Adding markers is done through the .addMarker() method of the GoogleMap object. The same applies to drawing circles. Now lists must be kept for all drawn objects, not for future redrawing, but for being able to remove or change them. The entire Fragment object, along with a lot of helper classes had to be rewritten completely. All methods helping out the onTouch event for the custom Overlay were not necessary anymore. Also many refresh workarounds were thrown away and in the process, the Activity's runOnUIThread() method was discovered.

8.6 Android development

I must also add the fact that learning how to program in Android was done while developing. This was often a trial-and-error process, covered with personal takes on the tutorials found mostly on StackOverflow.

9 Organization of the project

Before there was the project, there was a long search. I initialliy wanted to continue what was done one semester before, in a lab on agile software development, where 12 people developed a tour app using Phonegap. I wanted to add multiplayer functionality to the game, but simply seeing everybody on the map was to simple. Ideas arose from discussing this with my coordinator. Among them were ideas for including games and side activities. During the next three months, I have tried all the mobile games that used a GPS and that I could find on iOS. Most of them were also ported to Android. In the meantime, I have found out of the existence of Bluestacks, an Android emulator for Windows. A large number of games were tried out, but many gave a feeling of involvement and some gave the impression that they will require too much commitment from the player.
In the proposal for the project, I separated the 6 month time in 4 parts :

a. First phase of development - 2 months

b. First phase of testing - 1 month

c. Second phase of development - 2 months

d. Second phase of testing -1 month

This interval was also supposed to include the writing of this paper, in between the lines. This schedule has proven unfeasible - not because of the length of the development phases, but because of the very short length of the testing phases. Another big part of the development of this app has been played by the continuous search for the appropriate technologies and practices for the project - including my own organizing:

a. The exploration phase - Creating a UI test (Android), a server test (NodeJS, then Python, then Java), a client test(Android). There have also been some short UI tests and discussions along the way. This phase took around two-three weeks.

b. The initial development phase - Creating a functional app with the Maps API V1 and writing a full-blown server. This was the longest phase - taking around one month and a half.

c. The first testing phase - It lasted for half a day. Preparations for it took another few hours (upgrading the OS on two phones from Android 2.3 to 4.0).I brought a few friends, installed the app on all their phones and tried it out for 1 hour (in multiple attempts, with some crashes). After playing the game, the friends and I discussed the usability, mechanics and feeling in and out of the game, in a focus group approach. A friend and I took notes, while we all discussed various aspects of the app. Also the menus and a possible logo were discussed. I gathered so much input from them that only their proposals would change the face of the app entirely. Also this is when I realized that I must make the app compatible with Android 2.3 for the least (almost 50% of Android users have this version on their phones).

d. The second development phase - I have worked for another 3 weeks to completely construct a new UI according to the requirements from the testing phase and the switch from Google Maps API V1 to V2. I have added a crash logging and data usage logging system to the app.

e. The second testing phase lasted one day. I created a Facebook event and invited 10 people that brought Android phones. Most of them also had 3G sim cards. The game was finally field tested and I could get the opinions of the people on the UI, the functionality, the speed of the app.

f. The documentation phase is ongoing. This is the paper documenting the development of the app.

9.1 Kanban

For managing the development steps of the app, I have mainly used a simplified Kanban board. As is specific to Kanban, I have gradually proposed stories and fragmented them into tasks, where necessary. As I worked alone, all team aspects of this organizational system have been removed. The fields that have been kept are an 'Input Queue', two 'Development' slots, the 'Integration' slots and the 'Live slots'.

There is a significant physical difference between the board that I used, versus the one used in the 'Agile Lab' from the previous semester. The first used a whiteboard with slots defined by lines drawn with a marker, on which the stories and tasks were held with magnets. Although this has been highly efficient for the short duration of the lab, I don't condsider it useful for an endeavor of longer duration - stories and tasks and magnets are lost. The markers get misplaced and paper is often not found. For the development of the app, the Kanban board was made out of an actual wooden board, on which transparent plastic envelopes of two different sizes were placed as slots for the stories and tasks. Also, under the board, three more slots have been added : one for the papers for the stories and tasks, one for the utensils(markers and pens of various colors and scissors) and the last one for the finished stories and tasks that did were pushed outside the board. This gave better control over the location of everything needed and multiple stories could be placed in the same slot. For example, the two 'Input Queue' slots have proved to be not enough for the influx of ideas, and proposals for modification. Also, the 'Live' slots were not used for single stories, but for groups, as will be described below.

A personal touch has been added to the Kanban board : the 'Development' slots have only been used at the very beginning of the project. What I have noticed is that the developer must have a number of stories and tasks at hand, for orientation. I have personally lost track of what I was doing when the 'Development' tasks were at the appropriate locations on the board. As a result, I stopped using the two slots on the board and created a separate slot on the wall, next to my main monitor. Having all the stories that I was working on has proven much more useful and straightforward.

I have noticed that many stories are linked in various ways. I was writing the stories and grouping them according to their connectedness in various aspects - from similar functionality to work in the same few classes. According to these criteria, I have treated stories in groups, mentally separating them. For my future work on this project, I propose adding a few more slots on the wall, in front of the programmer. In each slot, the programmer can put together stories with common traits. Also, I recognized the importance of having the full story in front of you, even if you are working on only one task from it. Having the big picture at all times is just as important as dealing with the details. In the case of work within a team, I would have the story and tasks on the Kanban board, with the names of the developers dealing with each task, while each developer would have a copy of the story and the tasks from the board (with the names of the assignees included) in front of his own workstation. Therefore, a link between the people working at different tasks and between the tasks themselves would be permanently kept - and synchronization on overlapping tasks would be easier. Also due to the fact that I treated stories in bulks of related work, they have been added to the 'Live' slots based on these same criteria.

As a difference from the Kanban board used in the 'Agile Lab' from the previous semester, I propose that all completed stories and tasks have their own slot, in a visible place outside the board. This offers a good morale to all the developers who see the stack of solved stories thickening. Having a hold of all the stories also greatly helps write this paper.

9.2 Unit Testing

I considered unit testing for this project, but soon gave it up. This application is a conceptual prototype. A lot of trial and error was done during development, changes to some piece of functionality occured as often as a few times a day. I cannot not acknowledge the usefulness of unit testing in a full blown, large scale project. But in this case, it has proven to be a liability. I wrote test cases at the beginning, only to find frustration in the fact that the specifications for the functionality changed very quickly. Also, because this app has been more of a creative than technical endeavor, I had to keep my mind more on the ideas than on their implementation. Often, I have started with a 'quick and dirty' approach, tried out how I felt with the implementation, trying to put myself in the shoes of an unknowing player and only after this organized and modularized the code.

10 The game concept

The app that I developed is temporarily called People With Guns. It is a GPS- based RTS / Shooter hybrid concept protoype for Android. It can played by several people (The upper limit has not been established yet. Until now, the highest number of players in the game is 6.) that choose to join one of two opposing teams. The purpose of the game is to use 'weapons' and 'powerups' to defeat the opposing team. By 'defeat', I mean to 'virtually kill' all the opponents with the available tools. The current 'tools' are as follows :

The weapons :

a. Pistol

b. Rifle

c. Sniper Rifle

d. Knife

The 'powerups' :

a. Invisiblity Cloak

b. Shield

c. Painkillers / Heal

Each item used by a player has the following attributes :

a. Cooldown - The amount of time that has to pass until the weapon can be used again.

b. Duration - The amount of time during which a powerup is in effect

c. Damage - The amount of health points that are subtracted upon a hit (or added, in the case of the Painkillers) from the target's total available health points.

d. Range - The maximum distance within which a weapon can be fired.

Based on these attributes, we can make a differentiation between weapons and powerups. Until this point of the game development, weapons have damage and no duration. The only powerup that also has damage are the Painkillers - they deal negative damage to the target. The other two powerups, Invisibility Cloak and Shield, have a greater than zero 'duration' attribute, but no 'damage'.

Some of the attributes of the weapons and powerups will be modified, during a phase of balancing. I will therefore describe their conceptual construction :
The weapons :

a. Pistol - Weapon with low damage and medium fire rate. All the character types have it. It is the basic and least effective weapon of them all. The range is reduced.

b. Rifle - Weapon with low damage and high fire rate. The damage is higher than that of the pistol and the cooldown takes half as long. The range is reduced the same as that of the Pistol

c. Sniper Rifle - High damage weapon with very low fire rate. The range is far greater than that of the Rifle and Pistol.

d. Knife - The weapon that deals the highest damage of all. The cooldown is greater than that of the Sniper Rifle and the range is very small.

e. Invisiblity Cloak - Powerup that makes the player disappear from the map for a short while. While the player is invisible, he/she cannot be shot. The effect duration is short, while the cooldown is lengthy.

f. Shield - Powerup that reduces the damage taken to half, while it is in effect. The duration of this powerup is short and the cooldown is lenghty.

g. Painkillers / Heal - Powerup that functions like a weapon that deals negative damage to the player or his/her friends. It's effective immediately and requires a long cooldown time.

For more complexity in the game, a number of so-called 'character types' have been created, from which players can choose. Thus far, there are four character types implemented in the game: Marine, Medic, Sniper, Scout. Each of these has a different number of health points and different weapons. Because the actual health amount will vary during a phase of character balancing, I will simply mention the conceptual construction of the characters :

a. Marine - Has average health and two weapons: Pistol and Rifle. Represents the basic attack unit.

b. Medic - Support unit with Shield and Painkillers/Heal abilities. Has average health.

c. Sniper - Attack/ defense unit. Has a Sniper Rilfe for shooting at large distances. Has low health.

d. Scout - Attack unit specialized at sneaking up on the victim. Has the 'Invisibility Cloak' ability for disappearing from the map and the 'Knife' weapon for dealing large amounts of damage within a very small range. Has large health.

Another character has been created for testing purposes. It has been called the 'All Encompassing' and posesses all the weapons and skills presently existing in the game and very large health. This character has inadvertently opened a window for two more game types:

a. 'David versus Goliath' - This be a game of many players using regular character types versus a much smaller number of 'All Encompassing' characters.

b. 'Duel' - During the many small tests made to this app beyond the public ones, a new style of playing this game has emerged: in the absence of a large enough number of players, two can play in the 'Duel' mode : both use the 'All Encompassing' character type and, instead of running around, attempt to win the game by optimizing combinations of weapons and powerups. This game is generally played side by side, for at most a few minutes and has proved to be entertaining for the ones who tried it out.

11 Developing the app

The development of the app has started on the 14th of January 2013. I had an idea and no Android programming skills. Everything described from now on is a process of creativity and learning, intertwined together. But before the development came the idea, which took three months to concoct.

11.1 The exploration phase

The project has started as a proposal to add a multiplayer feature to a tour app. It was hazy and unstructured. The main point was that the tour app was targeting individual users. But then, what happens when a larger group tours together? At the current stage of development, the tour app was offering an individual experience within the group. So I proposed that the group experiences the tour together, somehow.

Simply adding the functionality to see all the others within the group on the map was not sufficient - it would just help if someone got lost or went astray. Otherwise, I concluded that the user experience would not be

improved in any significant way. Then came the idea of adding small games, hidden caches or quests and so on. The best idea that still had the tour app as a main platform was to add detours from the main track as bonuses. On those detours, the people would have to solve various riddles and small puzzles to get points and find out about hidden historical spots or interesting facs about the places they are visiting.

At this point, I contoured the following addon to the main app: the players would have a main tour path and, as they passed by certain waypoints, would be offered to go through a bonus/extra track within the area that they are visiting. If enough of them would agree to do this(by an in-app vote, for example), they would be presented with a new set of waypoints. These waypoints would belong to a number of categories: normal waypoints, waypoints where they would have to split, waypoints where they would have to be together and waypoints where the whole group would have to wait within a virtual circle, while one delegate(through vote) would find an item or solve a riddle.

I have come up with another scenario, during the research: In the case of the group waiting and one member being delegated to solve a riddle or find an item, a means of cooperation can be brought up: When the team votes and chooses the delegate, he/she gets a black screen(no map) and the rest of the team can see the goal on their phones. Through messages or VoIP, they would help the delegate navigate to the goal. Then, once the goal has been found or reached, the virtual circle would disappear and the whole team would be free to move. If anyone would exit the circle at some point, they would receive penalties and eventually get disconnected from the game.

Gradually, the ideas for multiplayer features went astray from the tour app, towards multiplayer GPS-based games. I have explored the idea of GPS-based puzzle / adventure games and encountered ARIS, a platform for creating such games. Then I discovered Tourality, and WarFinger GPS, which were the main sources of inspiration for my app and a few other games that didn't make it in the main concept, but are to be developed within the game as future work.

There has been a point when I simply tried out all the GPS-based mobile games that I could find, and evaluated them. At this point, I already had in mind the goals for this app: It should move the players to an outdoor environment and have them walk and run as the main activity, while using the game itself for an improved experience. Hide and seek and Tag were considered as a model of entertaining game to be played by a group. The games I found and already enumerated had, in my opinion, one of two major disadvantages:

a. Did not engage the users enough: Games such as Parallel Mafia, SCVNGR or Please Stay Calm do not motivate the player to move around much. They also offer a very dim user experience in areas with few or no players.

b. Engaged the users too much: Mobile users, even hardcore gamers, do not spend too much time playing on the phone. Rather, they would play on consoles or computers, for better immersion. No matter how good the game is, it is still displayed on a small phone screen(tablets are not considered in this paper). Games such as Ingress and Shadow Cities offer a better and more immersive story, but are still demanding of the player and request the full focus of the phone owner. This I considered to be a major downside, as per my gamer experience, when I get out of my home, I prefer to focus on what happens around me and only rarely have the time and will to play such a demanding game while on the go.

c. Did not motivate the players to move enough - Only Tourality does not possess this downside, as it its various game modes are specifically designed for running.

d. Are location-dependent - All adventure / puzzle games such as those developed with ARIS, most MMOs such as Parallel Mafia do require the player to be in certain places in order to progress through the game. This means that the player is put in one of two situations : He/She 1. has to get out of one's way in order to make any progress within the game, or 2. has to travel to remote locations in order to play the game in the first place. This might be interesting for some, but will certainly fail to capture the attention of a worker or student during weekdays.

The engaging problem can also be linked to a time problem. Games that are too engaging also require a lot of time to be played. My personal take on this is that the amount of time dedicated to the mobile game should be decided by the player and not by the game.

11.2 The initial development phase

The first development phase was mainly one of searching for the right tools to get the job done and testing their functionality.

The initial idea was to use Websockets for communication, a NodeJS-based server and, after some evaluation, native Android. The messages exchanged between server and clients would be in the JSON format. For this, the choices that I found viable have been Jackson and Google GSON and json-smart. After determining their speed and ease of use I have chosen Jackson, as besides its speed, it offers an easy way of serializing and deserializing JSON directly into a hierarchy of HashMaps - a method which I personally prefer for a testing phase. Mapping something yet unknown into POJOs would have been a much harder task on the long run.

Exploring the use of Websockets has proven unfruitful, as there have been dead ends :

a. The only freely available Websocket library for Android at the time of research was Autobahn. The Websocket libraries available for NodeJS were Socket.IO, Websocket-Node and ws. None of them worked with Autobahn for Android. I eventually found a forum post in which one of the developers of Autobahn stated the reasons for the incompatibility between Socket.IO and Autobahn for Android: First, the protocol implementations were based on different draft versions. Second, Socket.IO used an HTTP handshake for the connection - and that was not supported by Autobahn. The same issue applied to Websocket-Node and ws. A Python implementation of Autobahn has been tried for the server, but after a few unfruitful attempts, I decided on Java. In the case of Java, Autobahn for Android does not work. I have identified one of them: Autobahn subclasses a Handler object that is part of the Android SDK. Because of this and the realization that until Websockets receives its final version, I cannot fully rely on the protocol. And for the purpose of a prototype application where communication should not be overly complicated, I chose TCP.

b. Because of the Websocket issue, but also the subjectively perceived slow development pace with NodeJS (Libraries are not documented, autocomplete mostly does not work and there is no javadoc equivalent), I have decided to switch to Java. Python was also attempted in the meantime, but productivity was perceived as low for the same reasons as for NodeJS.

At the end of these attempts, the tools that I decided upon were native Android clients, Java server and TCP communication in between. What was left to decide upon was whether I would use JSON or XML. Based on some research and my need for this app, I chose JSON - it is easier to use and it takes less bytes to transmit the same data as it would with XML. I was planning to use simple data structures for the messages that were to be exchanged between server and clients.

JSON was chosen. But now the question remained: which Java libraries for processing JSON would I use? After some quick browsing, three names came on top: Jackson, Gson and smart-json. I did some searching for benchmarks and tried out Jackson and Gson. I chose Jackson.

Now the tools were readily available and I started developing the app. I split the development in three sub-phases :

a. A game UI that would add some mock players on the map and provide some usability insight.

b. The server

c. The whole app, based on the initial UI experience.

In developing the initial game UI, I added three buttons(one for generating a number of markers ('Generate Markers') on the map, one for further use('Check Info') and one for shooting ('Shoot')) and a spinner, on top of a MapView. The spinner served as a weapon selector. Once a weapon was selected (on launch, the first one in the list was automatically pre-selected), its range would be drawn on the map. I placed the three buttons in three corners of the screen- bottom-left, bottom-right, top-left. I placed the spinner to the right of the 'Shoot' button. The functionalities added were as followed :

a. 'Shoot' button - Mock method to display a Toast message stating if the shot was performed or not and the selected weapon.

b. 'Generate Markers' button - Mock method to randomly generate markers on the map (for this game's purposes).

c. 'Check Info' button - sporadic functionality to display a Toast message with various info on data structures in the back end.

d. Spinner - selecting weapons to 'shoot'.

I initially developed the server in NodeJS, then switched to Python and eventually settled with Java.

11.2.1 The structure of the server

a. A 'communication' module for dealing with adding and managing connections and messaging.

b. A 'messages' module for managing the incoming and outgoing messages for each connection.

c. A 'game' module for handling in-game data, such as keeping track of connected players, teams and game status.

11.2.2 The structure of the 'communication' module

a. The 'TcpServer' class that starts listening for incoming TCP connections on a given port.

b. A 'ConnectionManager' class that provides static synchronized methods for managing adding and removing connections, MessageSender, MessageReceiver and connection keepalive heartbeats.

11.2.3 The structure of the 'messages' module

a. A 'Messages' class that contains a number of inner classes for categorizing the different types of messages.

b. A 'MessageSender' class that deals with sending messages to a single client, but can also access all other instances of this class to multicast and/or broadcast messages to all the other clients, when needed.

c. A 'MessageReceiver' class that deals with receiving messages from a single client

d. A 'HeartbeatListener' class that listens for heartbeats from the connected clients and keeps the connections alive or closes them.

11.2.4 The structure of the 'game' module

a. A 'Player' class that stores information about the players (eg. name, 'profession', health points and position).

b. A 'GameManager' class that manages the addition, removal and management of players.

11.2.5 How the server works

Once started, the server listens on a port. When a remote client connects, the server adds an InetSocketAddress instance containing the IP address and port of the client. Then, a Player object with some default values is initialized and a random UUID is generated. The Player object is added in one of the HashMaps for the two teams - home team or away team - using the UUID as key. A MessageReceiver and a MessageSender are instantiated. The MessageReceiver, MessageSender, a Date object containing the moment of the connection and the newly generated UUID and added to HashMaps with the InetSocketAddress of the client as key. The server sends a configuration json containing the already connected players, the available professions and their attributes(title, health, weapons, description), the available weapons and their attributes(name, range, cooldown, duration, description and usage policy). From now on, whether a player remains connected to the server depends on the so-called heartbeats : The client will send periodic heartbeat messages to the server. The server will update the Date object for the last moment when a heartbeat was received. If a delay beyond a threshold comes up, the client is removed from the server.

The server now acts only as a relay - game state is held on the client side.

Because TCP does not allow multicasts and broadcasts and UDP was purposely avoided, methods for sending multicasts and broadcasts were conceived as follows: When a client sends a message that requires multicast/broadcast, the server accesses all the MessageSender objects, iterates through all of them and sends the given message to all of them. This applies to location updates, shot alerts and in-game messages.

Once a number of players (1 or more) have connected to the server, they may send 'Ready' messages to the server, stating that they are prepared to enter the game. The server will check on receipt on each 'Ready' message if all the players connected to the server are 'Ready'. If yes, a countdown timer will be started: A broadcast with the seconds left until the game starts will be sent to all the players. After the countdown, an 'Enter Game' message is broadcast to all the clients. Because the server holds the information on the positions of all the players, a condition and another message have been added: when the distance between the average positions of all the players in each team is at least some given distance(eg. 500m) and each team member is at most some other given distance(50m) from the average position of his/her team, 'Start Game' message is broadcast to all the clients. This will later be used to enable the weapons for all the players only when they satisfy these conditions.

There is no direct disconnect message between server and client. Disconnection is done exclusively based on the heartbeat interval.

11.2.6 How the client works

The client is an Android application that uses one Activity and multiple Fragments.
The client uses seven fragments for the UI: 'Main Menu', 'Info and Tutorials', 'Settings', 'Lobby','Lobby Settings', 'Loading' and 'Game':

a. 'Main Menu' : It is the entry point of the the application (it is the fragment loaded by the Activity after it is created). It features four buttons: 'Connect to server', 'Info and Tutorials', 'Settings' and 'Exit'. By pressing 'connect to server', the player attempts to join the game. If the connection is successful, he/she is brought into the 'Lobby'. 'Info and Tutorials' and 'Settings' lead to the homonymous 'screens'.

b. 'Info and Tutorials' is a fragment with a number of buttons and a ScrollView for displaying text and images. Here, the user will find instructions for the purpose how use of the app and detailed descriptions of the functionality of the 'Lobby' and 'Game' UIs.

c. 'Settings' now only contains two TextViews for changing the default IP address and port of the server to which the client can connect.

d. 'Loading' is an intermediary fragment that shows a loading widget for the duration of time during which the connection to the server is established and the client receives the config json from the server. Once this process is done, it automatically switches to the 'Lobbby' fragment.

e. 'Lobby' is the main game preparation fragment. It shows the lists of players in the two teams(nicknames, professions and 'Ready' status). The fragment features four buttons: 'Back', 'Change Name and Profession', 'Change Team' and 'Ready'. If the 'Back' button, the connection to the server is closed and the 'Main Menu' fragment is brought to the front. The 'Change Name and Profession' button brings upon the 'Lobby Settings' fragment. By pressing the 'Ready' button, the player toggles his/her 'Ready' status and sends a message containing this status to the server.

f. 'Lobby Settings' is the fragment in which the player can change his/her nickname and 'profession'. A textbox for the name, a spinner and a description textbox for the profession and an 'Ok' button are provided.

g. 'Game' is the most important fragment in this app: it provides the UI for gameplay. Due to changes in the use of the Maps API and tester feedback, it has gone through the most changes. At this point, the game screen looks as follows: for all the 'weapons' and 'powerups', buttons are aligned starting from the bottom left corner of the screen and extending them to the right. From the bottom right corner and going

up, on the vertical axis, three buttons and a TextView exist: the 'Previous Target', friend/foe toggle, 'Next Target' buttons and a TextView that shows the distance to the next selected player. If no player is selected, the presented text will be '0(Self)'. Otherwise, just the distance measurement is shown, with no text. In the top right corner, the default 'My Position' button is shown - with its specific icon. In the top left corner of the screen, a TextView with large text shows the remaining 'health points' of the player. Underneath this TextView lays a spinner for sending predefined messages to the team. The full functionality of the UI will be presented below.

The typical app use scenario goes as follows: The player enters the game and sees the 'Main Menu'. The app checks if Google Play Services are installed and if the GPS is turned on. If either of these conditions is not met, a dialog is presented: the player must choose to install Google Play Services and/or turn on the GPS or exit the game. Each dialog directs the user to the appropriate Google Play or Settings page - where the player only has to either click 'Install' or switch the 'Enable GPS' toggle to 'ON'.

Once there are no more requirements to be met in order to play the game, the player can click on 'Connect to Server', be briefly presented with the 'Loading' fragment and then enters the 'Lobby'. There, if the app was started for the first time, he/she can see the default nickname 'Player' and the default 'profession' - Marine. If the app was previously used, the player will see the last nickname and 'profession' used in previous runs. The server distributes the players according to team sizes, so the player has an equal chance to see oneself in either the 'Home' or 'Away' team. Then, the player can opt to change the team by using the 'Change Team' button. Also, the nickname and 'profession' can be changed by navigating to the 'Lobby Settings' screen. Once the 'Ok' button is pressed in the 'Lobby Settings' screen, the changes are saved in the app and sent to the server for broadcasting, and the player is returned to the 'Lobby' screen.

At the point where all the players have done setting up, they can send the 'Ready' signal. When all the players are ready (for testing purposes, one player present on the server is enough to start a game), the five second countdown is received from the server and the player is now facing the 'Game' screen.

In order to understand what can be done at this point, a detailed description of the 'Game' UI is necessary:

We can visualize the UI based on the screenshot presented in Figure 1. It presents the groups of UI elements on the screen. We shall now present all of them, separated into groups, by position (positioning also separates functionality groups, therefore we can also state that they are presented by related functionality). All the UI elements on the 'Game' UI are programmatically generated - and not from an XML file :

a. **The bottom of the screen, starting from the bottom-left corner**: The 'weapon'/'powerup' buttons are generated once the 'Game' fragment is loaded, based on the list of weapons specific to the player's 'profession'. They appear from left to right. If less or equal to three weapons are given, the buttons will be made slightly larger than otherwise - up to a maximum of six (the number of weapons featured in the test 'profession', 'All Encompassing'). Each button press shoots the 'weapon' or 'powerup', according to a so-called 'policy' provided in the Weapon object. The policies are as follows: 'self', 'friends', 'friends and self', 'enemies', 'friends and enemies', 'friends and enemies and self'- stating on which kinds of players one given 'weapon' or 'powerup' can be used. A button long press will draw the range of the 'weapon'/'powerup' on the map, with the player's position in the center.

b. **The bottom-right corner of the screen**: The player selection buttons and the distance indicator are generated on the vertical axis, starting from the bottom-right corner of the screen - in order, the 'previous target', friends/enemies toggle, 'next target' buttons, and on top a TextView showing the distance to the selected player or '0(Self)' if no player is selected(in translation, the 'self' or 'current player' is selected). The friends/enemies toggle is used for the obvious reason of quickly choosing between one of the two types of players besides the 'self'. The 'next target' button's functionality is to find the next closest player from the selected category. Let's say that the closest player is already selected. Then, on a press of the 'next target' button, the second closest player will be selected and so on. The reverse applies to the 'previous target' button - which selects the previous farthest player if a player is selected and the closest one otherwise. The distance indicator serves for the use of an experienced player who knows the weapon's distances by heart and does not want to the lengthy long click operation to get the weapon's distance shown on the map. The players may also be selected by simply tapping their respective markers on the

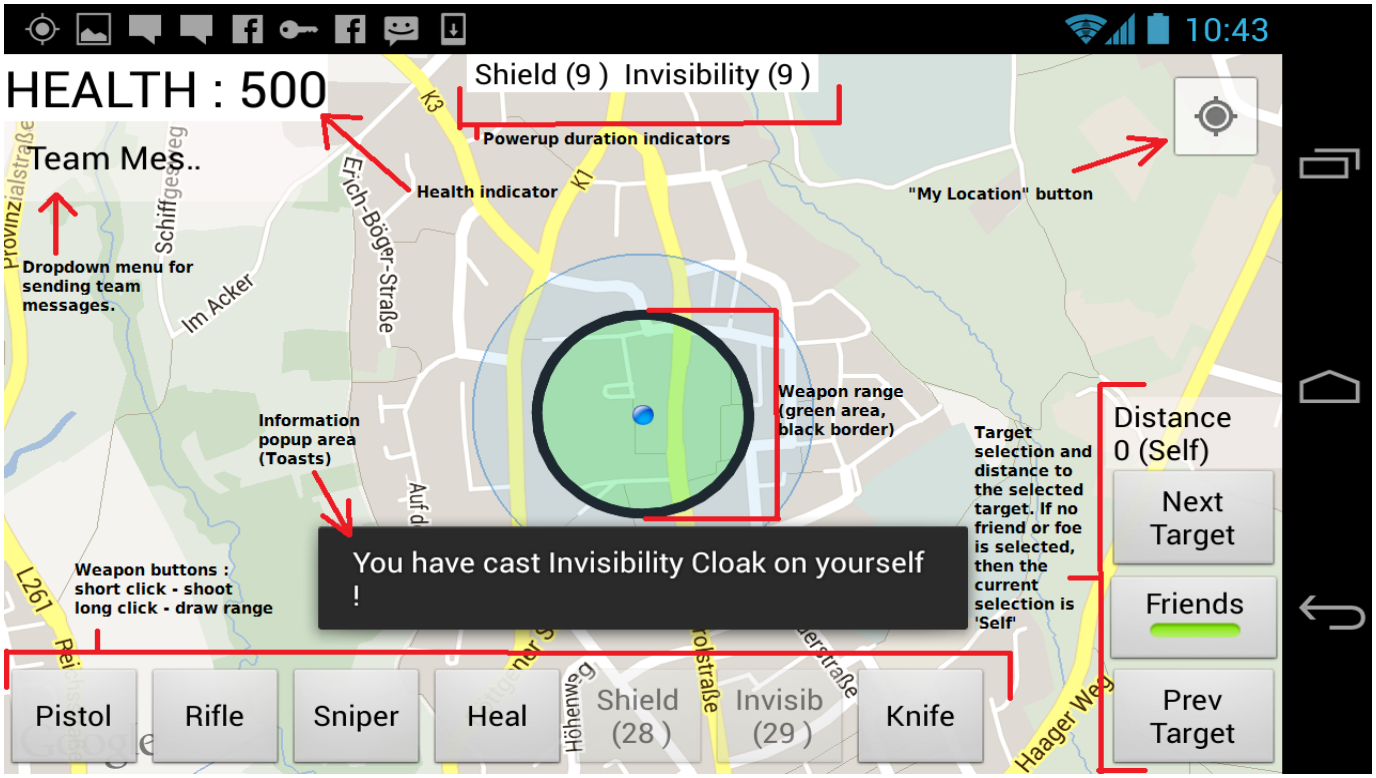

Figure 1: *The in-game UI*

map - but through testing it has been determined that in a more intense and dynamic situation, selection by clicking on the marker can become difficult and inaccurate. Both modes are supported now. One other aspect of player selection is the now-vital click in a random unoccupied area on the map for deselecting all players(and selecting oneself for, say, self-healing or activating powerups on oneself).

c. **The top-right corner of the screen**: The 'my position' button that is provided as an option through the Maps API. Clicking it will cause an animation to scroll through the map until the position of the player is centered.

d. **The top-left corner of the screen**: The health indicator is a TextView that presents in large text the remaining health points of the player. Underneath it lies a spinner with a list of messages that the player can send to his/her team. One can do so by clicking on the UI element representing the spinner, beneath the health indicator. After the first click, the spinner dialog appears and shows the list of messages. The player can send one of the messages in the list by clicking it. The dialog is canceled by clicking outside it.

e. **The area in between the two top corners of the screen**: This is the area where the powerup duration is dynamically shown during its effect. This can be seen when the player casts 'invisibility' or 'shield' on oneself or when a friend casts 'shield' on the player

f. **The area above the weapon buttons**: This is where the notifications appear, in the form of Toasts. By default, a toast has a given lifespan - but each time a new toast needs to be displayed, it first closes the toast currently on display, if it is the case.

Although they are not yet discretized, thus far we can distinguish three types of game that can be played with the current setup: the **default game**(which requires multiple players, but excludes the 'All Encompassing' game type), **duel**(two players choose the 'All Encompassing' profession - receiving the entire arsenal provided by the game - and try to take each other out by using various strategies of combining 'weapons' and 'powerups'), **David vs. Goliath**(a small number of players choose the 'All Encompassing' profession and play against a significantly larger number of players that use all the professions except 'All Encompassing').

Until now, the default game and the duel have been tried out - and they are fit for diffrenent situations: The default game can take up to 20 minutes and is played across distances varying from small to very large, depending on the mood and disposition for running of the players. The duel is performed usually by two players sitting next to each other and takes two-three minutes.

Thus far, no game end condition has been introduced - players who lose all health points are notified that the game is over for them through a dialog that gives them two options: quit the game or spectate. The second option implies that they are marked as 'dead' on the map, cannot be selected and their weapon buttons are removed from the screen. Still, they can watch the evolution of the game on their mobile devices. When all players of one team are marked as 'dead' (they have lost all their 'health points') and at least one player of the other team still has more than zero health points, it can be considered a win for the other team. The ending condition has not been implemented, because for this stage of development of the app it would be just a dialog stating the obvious 'You have won !' message. A more interesting feature may be added in the future, if imagination or user feedback provides it.

11.3 The first testing phase

11.4 The second develpment phase

11.5 The second testing phase

11.6 Documentation and development

12 The UI design

13 The server

14 Future work

15 ANNEX A - game and game platform websites

ARIS Games: http://arisgames.org/

Tourality - http://www.tourality.com/

Wherigo - http://www.wherigo.com/

conTAGion - http://www.2clams.com/

Shadow Cities - http://www.shadowcities.com/

SCVNGR - http://www.scvngr.com/

Please Stay Calm - http://pleasestaycalm.com/

Parallel Mafia - http://www.parallelmafia.com/

Parallel Kingdom - http://www.parallelkingdom.com/

Tripventure - http://www.tripventure.net/tripventure/

Warfinger GPS - http://www.warfingergps.com/

Totem - http://www.fit.fraunhofer.de/en/fb/cscw/projects/totem.html

Portal Hunt - http://www.totem-games.org/?q=portalhunt

aMazing - http://www.totem-games.org/?q=aMazing

Ingress - http://www.ingress.com/

Mobile War - http://www.mobilewar.org/index.php/en/

Mister X Mobile - http://qeevee.com/projects/misterx

Mobilis XHunt - https://github.com/danielschuster/mobilis/wiki/Mobilis-XHunt

Own This World - http://www.ownthisworld.com/

MapAttack - http://mapattack.org/

16 ANNEX B - game genre definitions

Real Time Strategy Game http://encyclopedia2.thefreedictionary.com/Real-time+strategy+game

> A type of video game in which players exercise strategy along the way, typically to conquer enemies and reach a final destination without being eradicated. For example, to win, players decide which routes to take, what needs to be done and how to do it.

Real Time Tactics Game http://encyclopedia.thefreedictionary.com/Real-time+tactics

> Real-time tactics or RTT is a subgenre of tactical wargames played in real-time simulating the considerations and circumstances of operational warfare and military tactics. It is differentiated from real-time strategy gameplay by the lack of resource micromanagement and base or unit building, as well as the greater importance of individual units and a focus on complex battlefield tactics.

Massively Multiplayer Online Game http://encyclopedia.thefreedictionary.com/Massively+Multiplayer+Online

> A massively multiplayer online game (also called MMO) is a multiplayer video game which is capable of supporting hundreds or thousands of players simultaneously. By necessity, they are played on the Internet, and feature at least one persistent world.

Adventure Game http://encyclopedia.thefreedictionary.com/adventure+game

> An adventure game is a computer-based game in which the player assumes the role of protagonist in an interactive story driven by exploration and puzzle-solving instead of physical challenge.[1] The genre's focus on story allows it to draw heavily from other narrative-based media such as literature and film, encompassing a wide variety of literary genres. Nearly all adventure games are designed for a single player, since this emphasis on story and character makes multi-player design difficult.

Casual Game http://encyclopedia.thefreedictionary.com/casual+game

> A casual game is a video game or online game targeted at or used by a mass audience of casual gamers. Casual games can have any type of gameplay, and fit in any genre. They are typically distinguished by their simple rules and lack of commitment required in contrast to more complex hardcore games.[1] They require no long-term time commitment or special skills to play[. . .]

Racing Game http://encyclopedia.thefreedictionary.com/Racing+game

> One of the more common uses of the term racing game is to describe a genre of computer and video games. Racing games are either in the first or third person perspective. They may be based on anything from real-world racing leagues to entirely fantastical settings, and feature any type of land, air, or sea vehicles. In general, they can be distributed along a spectrum anywhere between hardcore simulations, and simpler arcade racing games.

Shooter Game http://encyclopedia.thefreedictionary.com/Shooter+game

> Shooter games are a subgenre of action game, which often test the player's speed and reaction time. Because shooters make up the majority of action games, it is a fairly wide subgenre. It includes many subgenres that have the commonality of focusing "on the actions of the avatar using some sort of weapon. Usually this weapon is a gun, or some other long-range weapon".A common resource found in many shooter games is ammunition. Most commonly, the purpose of a shooter game is to shoot opponents and proceed through missions without dying yourself.

Turn Based Strategy Game http://encyclopedia.thefreedictionary.com/Turn+based+strategy

> A turn-based strategy (TBS) game is a strategy game (usually some type of wargame, especially a strategic-level wargame) where players take turns when playing. This is distinguished from real time strategy where all players play simultaneously.

Location Based Game (Location-enabled Game http://encyclopedia.thefreedictionary.com/location+based+game

> A location-based game (or location-enabled game) is one in which the game play somehow evolves and progresses via a player's location. Thus, location-based games almost always support some kind of localization technology, for example by using satellite positioning like GPS. "Urban gaming" or "Street Games" are typically multi-player location-based games played out on city streets and built up urban environments.

17 ANNEX C - Websockets vs. TCP speed

18 ANNEX D - Node.JS vs. Apache - performance and scaling

19 ANNEX E - JSON vs. XML

20 ANNEX F - Jackson vs. Gson vs. smart-json

21 ANNEX G - Reviews of the games mentioned in the paper

22 ANNEX H - Overlay tutorial

23 ANNEX I - Countdown buttons Tutorial

24 Bibliography

References

[1] J. Yang, "**Smartphones in Use Surpass 1 Billion, Will Double by 2015**," *Bloomberg.com*, October 2012.

[2] C. Janssen, "**Real-Time Strategy (RTS)**," *Techopdia.com*, 2010.

[3] J. Eldridge, "**What are Real-time tactics games? RTT games, definition and meaning**," *Examiner.com*, March 2012.

[4] S. Johnson, "**Analysis: Turn-Based Versus Real-Time**," *Gamasutra.com*, November 2009.

[5] Pascal Bihler, Holger Mügge, Daniel Schuster, Danny Kiefner, Robert Lübke, and Thomas Springer, "**Step by Step vs. Catch Me If You Can - On The Benefit of Rounds in Location-based Games**," tech. rep., Institute of Computer Science III Universität Bonn and Computer Networks Group Technische Universität Dresden, 2012.

[6] C. Warren, "**The Pros and Cons of Cross-Platform App Design**," *Mahsable.com*, February 2012.

[7] D. Hoang, "**Native vs. Cross-Platform Development: Which is the way to go?**," *Awesomegiant.com*, April 2012.

[8] D. Wehmeier, "**HTML5, Cross-Platform Compiled, Or Native: Choose Wisely In App Development**," *Mobiledevdesign.com*, October 2012.

[9] D. Software(dacris.com), "**Node.js: Microsofts Web Empire is Coming to an End**." http://www.dacris.com/blog/2011/08/31/node-js-microsofts-web-empire-is-coming-to-an-end/, August 2011.

[10] M. Zgadzaj, "**Benchmarking Node.js - basic performance tests against Apache + PHP**." http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php, 2010.

[11] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, and Anant Narayanan, "**WebRTC 1.0: Real-time Communication Between Browsers**," tech. rep., World Wide Web Consortium, august 2012.

[12] I. Fette and A. Melnikov, "**The WebSocket Protocol (RFC 6455)**," tech. rep., Internet Engineering Task Force (IETF), December 2011.

[13] H. Schulzrinne, S. Casner, and R. Frederick, V. Jacobson, "**Real Time Transport Protocol (RFC 3550)**," tech. rep., Network Working Group, July 2003.

[14] Kurt D. Squire, Mingfong Jan, James Matthews, Mark Wagler, John Martin, Ben Devane, and Chris Holden, "**Wherever you go, there you are: Place-based Augmented Reality Games for Learning**," tech. rep., University of Wisconsin-Madison and Academic ADL Colab, Madison WI, 2006.

[15] Eugenio J. Marchiori, Javier Torrente, Ángel del Blanco, Iván Martínez-Ortiz, and Baltasar Fernández-Manjón, "**Extending a Game Authoring Tool for Ubiquitous Education**," tech. rep., Department of Software Engineering and Artificial Intelligence, Complutense University, Madrid, 2010.

[16] D. J. Gagnon, "**ARIS - An open source platform for developing mobile learning experiences**," tech. rep., The University of Wisconsin - Madison, December 2010.

[17] Chinmay Barve, Sanjeet Hajarnis, Devika Karnik, and Mark O. Riedl, "**WeQuest: A Mobile Alternate Reality Gaming Platform and Intelligent End-User Authoring Tool**," tech. rep., School of Interactive Computing, Georgia Institute of Technology.

[18] Christopher L. Holden and Julie M. Sykes, "**Leveraging Mobile Games for Placebased Language Learning**," tech. rep., University of New Mexico, May 2010.

[19] J. M. Mathews, "**Using a studio-based pedagogy to engage students in the design of mobile-based media**," tech. rep., University of Wisconsin, 2010.

[20] Andreas Ritzberger and Stephan A. Drab, "**TeamTags: Domination An AGPS game for mobile phones**," tech. rep., Upper Austria University of Applied Sciences - Media Technology and Design and Upper Austria University of Applied Sciences Hagenberg - Mobile Computing, 2004.

[21] 3rd WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION (WPNC06), ***Field trial on GPS Accuracy in a medium size city: The influence of builtup***, 2006.