Design and Implementation for an Android based Massively Multiplayer Online Augmented Reality Game

Entwurf und Implementierung für ein Android basiertes Massively Multiplayer Online Augmented Reality Game
Master-Thesis von Denis Lapiner
Februar 2014

TECHNISCHE
UNIVERSITÄT
DARMSTADT

DVS

Design and Implementation for an Android based Massively Multiplayer Online Augmented Reality Game
Entwurf und Implementierung für ein Android basiertes Massively Multiplayer Online Augmented Reality Game

Vorgelegte Master-Thesis von Denis Lapiner

1. Gutachten: Prof. Alejandro Buchmann
2. Gutachten: Max Lehn

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 28th February 2014

(Denis Lapiner)

Abstract

This Master's Thesis proposes and implements a game design for an Android based massively multiplayer online augmented reality game. The proposed game design is based on an analysis of other popular augmented reality games such as Ingress, a game created by Google Inc. The game of this thesis motivates the players to meet in the real world and move in groups to certain points of interest like historic monuments. The spatial locality of players, which comes up when players move in groups, makes the game interesting for the research on mobile ad hoc networks (MANETs). The provided implementation of the game contains a centralized communication component that logs the network communication between players and therefore, is usable for data collection of network traffic in MANETs. The collected data allows to make a case study on the advantages of using a MANET infrastructure for augmented reality games instead of a client-server based approach. The evaluation of a game session with up to 16 players during two hours reveals that 50% of the network messages could have been distributed locally over technologies such as Wi-Fi Direct without using the internet. Moreover, the increased use of energy saving local communication instead of power consuming cell based communication would lengthen the battery lifetime of smart phones used to play the game. Additionally, the reduced bandwidth use of the internet connection allows players charged on a volume basis to save money. The implementation of the game allows to easily exchange the networking components. This way, in a future work the networking component used for close range communication could be replaced by a real MANET solution.

Kurzfassung

Im Rahmen dieser Masterarbeit wurde ein Spiel-Design für ein Android-basiertes Massively Multiplayer Online-Augmented-Reality-Spiel entworfen und implementiert. Das vorgeschlagene Spiel-Design basiert auf einer Analyse anderer populärer Augmented-Reality Spiele, wie z.B. Ingress, ein von Google Inc erstelltes Spiel. Das Spiel dieser Arbeit motiviert Spieler sich in der realen Welt zu treffen und in Gruppen bestimmte Punkte von Interesse wie zum Beispiel historische Monumente zu besuchen. Die räumliche Lokalität der Spieler, welche entsteht wenn Spieler sich in Gruppen bewegen, macht das Spiel interessant für die Erforschung von mobilen ad hoc Netzwerken (MANET). Die bereitgestellte Implementierung des Spiels beinhaltet eine zentralisierte Kommunikationskomponente, welche die Netzwerkkommunikation zwischen Spielern aufzeichnet und sich daher gut für die Datenerfassung des Netzwerkverkehrs in MANET-Strukturen eignet. Die gesammelten Daten ermöglichen es, eine Fallstudie über die Vorteile der Verwendung einer MANET-Infrastruktur für Augmented-Reality-Spiele anstatt eines Client-Server-Ansatzes durchzuführen. Die Auswertung einer zwei Stunden langen Spielrunde mit bis zu 16 Spielern zeigt, dass 50% der Netzwerk-Nachrichten lokal über Technologien wie Wi-Fi Direct hätten zugestellt werden können, ohne das Internet zu nutzen. Der verstärkte Einsatz von energiesparender lokaler Kommunikation anstelle der leistungshungrigeren Zellen basierten Kommunikation würde die Batterielebenszeit von den zum Spielen benutzten Smartphones verlängern. Darüber hinaus könnte die reduzierte Bandbreitennutzung der kostenpflichtigen Internet-Verbindung es den Spielern erlauben, Geld zu sparen. Die Implementierung des Spiels erlaubt es, Netzwerkkomponenten leicht auszutauschen. So ist es möglich, die Netzwerkkomponente die für die Kommunikation auf kurzer Entfernung genutzt wird, in einer zukünftigen Arbeit mit einer echten MANET Implementierung auszutauschen.

Contents

1 Introduction

This thesis proposes a game design for an Android based massively multiplayer online augmented reality game usable for data collection of network traffic in mobile peer-to-peer networks also called mobile ad hoc networks (MANETs)[16]. Possible application areas and already existent MANET applications are presented. Related games are also presented and their similarities and differences to the data collection game of this thesis are discussed. The collected data gathered by the game of this thesis can be used to evaluate the performance or to simulate network technologies used in MANETs. An example implementation of the game design is provided. During this thesis data was collected and evaluated concerning the advantages of direct communication between smart phones. It is shown that in the played game session it would have been possible to save up to 50% of the traffic that was sent to the internet over cell based communication, if direct communication technologies such as Wi-Fi Direct[14] would have been used. Finally, a suitable network organisation that could efficiently handle the network traffic of the game is proposed.

This section explains in which way the game created in this thesis contributes to the scientific research on mobile peer-to-peer networks especially those that can be built between mobile phones. Besides, it is explained why multiplayer augmented reality games for smart phones are highly suitable for evaluation of such mobile peer-to-peer networks. This thesis was made in close cooperation with MAKI the collaborative research centre at TU Darmstadt, which is presented in more detail in this section. Finally, a glossary with terms and abbreviations used in the writings below is included.

1.1 Motivation

This section motivates the decision to create an Android based massively multiplayer online augmented reality game (MMOARG). Massively multiplayer online games (MMOGs) are very popular. A good example to prove this is World of Warcraft (WoW), being released in 2004, having brought in yearly more than one billion dollar and holding a record for the most popular massively multiplayer online role play game (MMORPG) in the Guinness Book of Records, the game has now still approximately 8 million players[1][8][9][10][18]. However, commercial games are still server based, which results in high operation costs and bandwidth problems especially for mobile networks. Furthermore, a server based approach does not provide an economical solution for hotspots (areas of interest) such as virtual cities, which produce much more traffic than other places. The amount of messages that the server has to distribute grows exponentially with the amount of players that need to see each other, because every message sent by a player has to reach every other player in vision range. To cope with a big amount of such dynamic data server arrays must be used and that makes the server approach even more expensive. Additionally, the amount of data produced by such a hotspot can easily exceed the bandwidth of a mobile network. All this makes it interesting to evaluate the performance of peer-to-peer (p2p) networks for MMOGs. Peer-to-peer networks cost nothing for the game publisher, since they are based on computers of the players. This cost reduction on the game publisher side could also positively affect the fees that players have to pay for subscriptions they need to play a MMOG game such as WoW. Additionally, for mobile MMOGs played on mobile devices such as smart phones the p2p communication could be run locally over costless technologies such as Wi-Fi or Bluetooth. This way, players could reduce the costs that result from high bandwidth use of their internet connection. All this explains why it is interesting to collect network load data of an Android based MMOG and use the data for example in a simulation of network traffic for studies about the application of peer-to-peer networks for mobile MMOGs[27][37] similar to the space shooter Planet π4[25][26][36] done for PC.

To illustrate why the augmented reality (AR) aspect makes the case study even more interesting one needs to take a look at the collaborative research centre MAKI[4][5][6] at TU Darmstadt[7]. MAKI is the abbreviation for the German name „Multi-Mechanismen-Adaption für das künftige Internet“, which translates to English as Multi-Mechanisms-Adaption for the future Internet. Its challenge is to cope with the growing dynamics and range of variation in communication networks with increasing mobility. Furthermore MAKI deals with the continuously increasing quality expectations and the diversity of existing and future protocols. One of many technologies researched within the scope of the collaborative research centre MAKI are ad hoc Wi-Fi networks. Ad hoc Wi-Fi networks could improve the performance of mobile networks such as public communication networks for cellular phones especially for smart phones. The load of the public cellular networks could be distributed more equally via routing of information between smart phones. For example, if one transmitter station is overloaded, messages from within the range of this station could find their way over a sub network consisting of smart phones to a less loaded neighbour transmitter station, which would reduce the load of the bottleneck station. This would decrease the response time of messages, which would be lost due to high concurrency and would need to be resent again. In addition, the traffic at the transmitter station can be reduced when the needed information is stored or buffered within a smaller network of interconnected smart phones instead of being polled repeatedly from the station by each single device. Another advantage of direct communication between smart phones is the reduced latency, which is crucial for games. While usually a message would be sent to the internet over cell based communication and be forwarded to the receiver again over cell based communication in a direct communication there would be no unnecessary data transmission. Instead the sender could deliver the message directly to the receiver for example using Wi-Fi Direct[14] if the receiver is in range. Considering the fact that the mean time of cell based communications starts at approximately 100 ms and goes up to half a second[34], the latency would be reduced drastically using Wi-Fi Direct. Furthermore, short range communication technologies such as Wi-Fi and Bluetooth require much less power than cell based communication[15], which increases the battery life time of mobile devices. However, the simple traffic of a MMOG with its points of interest, such as virtual cities, where more users generate more traffic would be not enough to generate valuable traffic for simulation of an ad hoc Wi-Fi network. Now also the spatial spreading of the users in the real world becomes interesting. If for example the users of the prior mentioned virtual city would also be in the same real city, the traffic, which the users produce, could be distributed among them using an ad hoc Wi-Fi network. Local information like seeing each other or position updates would not necessarily have to be sent to a server in the internet. Only global information like high scores or persistent information like in game money holdings would need to be distributed globally. This is where the augmented reality idea came up. Even the biggest enterprises like Google Inc. research in this area. Two good examples of Google's research are Google Glasses[2] and the game Ingress[3]. The popularity of both projects is very high and proves the interest of people for this topic. In our context AR comes in handy, since it brings people spatially together. In a multiplayer online augmented reality game, where augmented reality is used to motivate people to move in groups and visit the same places the degree of local traffic between players in close range is very high. All this explains that such a game is a good use case for the application of a mobile ad hoc Wi-Fi network.

Having said that it is clear that an Android massively multiplayer online augmented reality game (MMOARG) could not only collect data for simulation and evaluation of peer-to-peer networks, but would also provide a typical application for an ad hoc Wi-Fi network which could be evaluated or developed in the context of MAKI. Furthermore, it would enrich the game world with an AR game and allow studies on user behaviour in such games.

1.2 Glossary

This section explains abbreviations and uncommon terms used in this thesis.

AOI area of interest. In the context of this work AOI is always the area near some subject, for example the AOI of a player is the area in which other players are visible.

App abbreviation for application, often used in the scope of smart phone software.

Apk file apk is the extension of Android application archives. Similar to the exe files on Windows, where exe is the extension for executable files.

AR abbreviation for augmented reality. One speaks of AR when the reality is enhanced with computer based elements such as sound, visuals or GPS data.

DotA abbreviation for Defense of the Ancients a multiplayer strategy game. DotA was originally developed as a map for the game Warcraft 3, but with DotA 2 it became a self-contained game.

Gameplay describes all possible interactions between user and game software. Gameplay does not consider graphics and sound.

GPS abbreviation for Global Positioning System realized by navigation satellites. However, in this thesis GPS is not only used to describe satellite localisation, but also sometimes stands for the combination of GPS and other location services.

HP abbreviation for health points.

HUD head-up-display, shows the most important information to the player.

ID abbreviation for identifier.

Instance an instance of a program is its representation in the computer's memory.

Latency the time between the user input and its effect in a network game.

LoL abbreviation for League of Legends, a multiplayer strategy game.

MAKI abbreviation for the collaborative research centre called Multi-Mechanisms-Adaption for the future Internet.

MANET abbreviation for mobile ad hoc network.

MMOG abbreviation for massively multiplayer online game.

MVC abbreviation for the model view controller software design pattern.

OS operating system e.g. Windows, Linux, MacOS, Android, iOS, etc.

P2P abbreviation for peer-to-peer.

PC abbreviation for personal computer.

POI abbreviation for point of interest.

Pub/Sub abbreviation for publish and subscribe a peer-to-peer network organisation paradigm.

QoS abbreviation for quality of service. Used to describe the quality of a network communication service from the perspective of its user.

RPG abbreviation for role-playing game. Such games allow players to be in a role of a character and to meet decisions for their character. Mostly the character can be modified and evolved.

Space shooter a computer game which is played in the outer space, the players have usually to fight each other with their spaceships.

Spawn to spawn means to create something in the game world. For example a battle unit can be spawned in a game.

UID abbreviation for unique identifier.

URI abbreviation for uniform resource identifier. For example the path to a file in a file system is its URI.

URL abbreviation for uniform resource locator. For example the link to a home page (without a destination file) is the URL of the page.

WoW abbreviation for World of Warcraft a very successful massively multiplayer online game.

2 State of the Art

One could ask why already collected data of location traces and other sensors cannot be used instead of collecting data again. Indeed, collected data with much bigger scope and amount than everything that can be collected during this thesis already exists. For example a well-known data collection campaign took place in Lausanne a Swiss city. During this campaign nearly 170 participants equipped with a Nokia N95 phone were observed for a period of one year[23]. Evaluating the data of this campaign would also allow to calculate how much close range communication could have been possible considering the location traces of the participants, but still the data would not be relevant for games. Most of the participants probably do not know each other and therefore they seldom come together to the same place. In an augmented reality game hotspots and points of interest exist, that are visited often by different players. Therefore, the data collected by the game of this thesis highly differs from data of everyday life, since in the movement of the participants cooperative and social patterns vital for games exist. Those patterns are not included in any data collection of everyday life.

This section will first present possible use and existing applications for MANET. It will be clarified, that MANET for games is not widely researched, despite the fact that it is applied in the industry. Later on existing mobile augmented reality games will be presented and their relevance for this thesis explained. Finally, the aspects of the presented AR games that need to be included in the game of this work will be discussed.

2.1 Mobile Ad Hoc Networks

According to [24] mobile ad hoc networks (MANETs) can be rooted back to the 1970's where they were researched by military agencies. Already in the 1980's it was possible to use ad hoc networks to operate an ATM over a wireless, Satellite Communication Network. In the 1990's a functioning group in the Internet Engineering Task Force has started to work on standardized routing protocols for MANET[24]. Nowadays, the most popular mobile ad hoc network standard is the IEEE 802.11 better known as WLAN (Wireless Local Area Network).

The possibilities for application of MANETs can be found in various papers. MANETs can be used for military, rescue operations, data and device networks, internet sharing and sensor networks[24]. Existing MANET applications are used to reduce the installation costs of metropolitan broadband networks, to improve public transportation services by sharing the locations of buses and providing better estimates on their time schedule, to guarantee connectivity between police vehicles without an internet connection[40]. Another example for the use of MANET is 7DS[33] a p2p data dissemination and sharing system that helps clients with intermittent internet connection allowing them to ask their neighbours for cached info or internet access[19]. In [20] games are even mentioned with one line, saying that MANET can be applied for entertainment applications for example multi-user games. However, MANET for games is widely used for example the PSP (Play Station Portable) could be used already in 2005 to play games offline in an interconnected WLAN network[1]. Nevertheless, even though single papers can be found[28][32][35], MANETs for games are not widely researched. All of this makes it even more interesting to develop the game of this thesis in order to evaluate the advantages brought by MANET to an MMOARG.

2.2 Mobile Augmented Reality Games

In this section a few mobile AR games considered to be important are presented. Please take into account that there are many more games of that kind and introducing all of them would by far exceed the scope of this work.

[1] A list of games that are playable on PSP over Wi-Fi url: `http://www.portable-playstation.com/psp-wifi-games.htm`

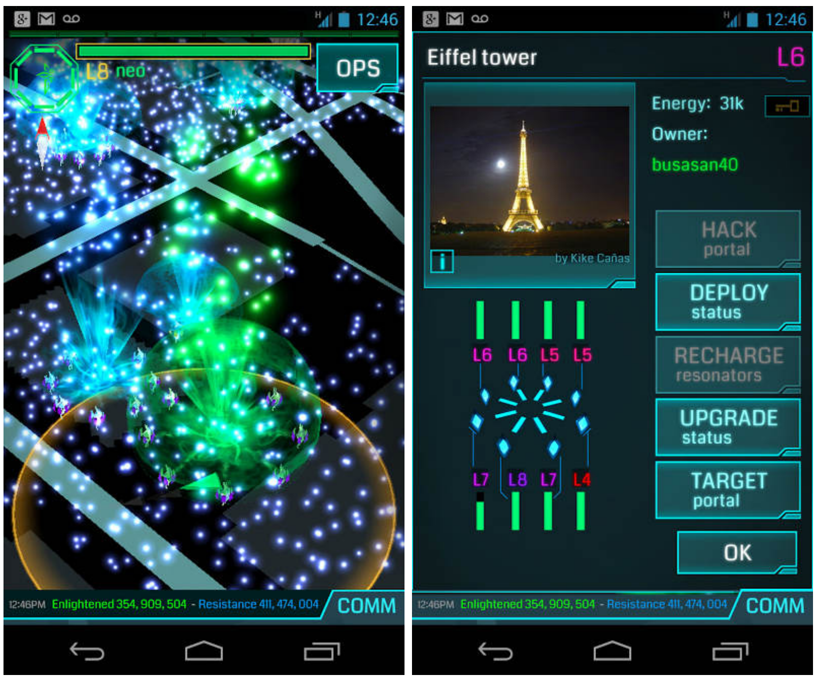


Figure 1: Screenshots of Ingress. The left image shows the player and multiple portals of different factions. The right picture shows the portal menu.
Source: `http://play.google.com/store/apps/details?id=com.nianticproject.ingress`

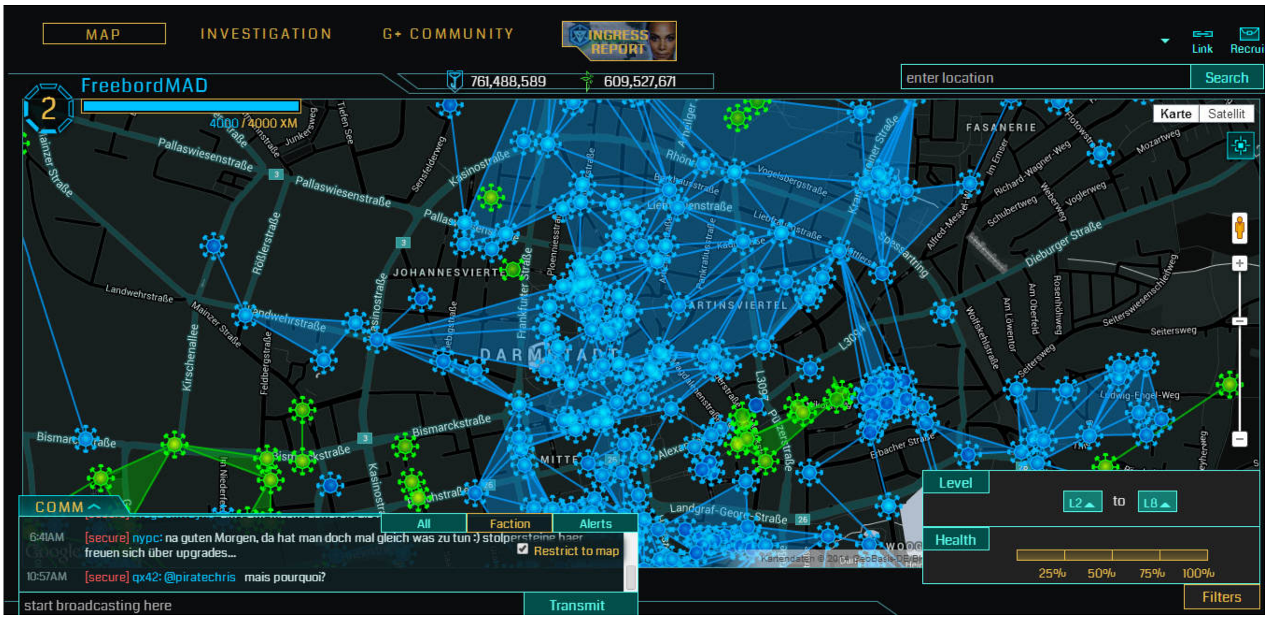
2.2.1 Ingress

Ingress is an AR game for Android made by Google Inc. When this thesis started Ingress was still in the beta phase and a beta key was needed to play the game, but already back than it had over half a million downloads in Google Play. By today it was installed more than one million times[2]. The gameplay is straight forward and very simple. In Ingress there are two factions competing each other. The player has to visit certain real world places, where so called portal are located. Mostly these portals are at historic monuments or buildings or just interesting places reported by many players. Visiting a portal gives the user items and portal keys. With the received items the user can either defend and reinforce the portals of his faction or attack and weaken the portals of the other faction. With the portal keys the player can link portals of the own faction, so that the links build triangles. The goal of the game is to cover as much space with linked triangles as possible for the own faction. Ingress is a never-ending game there are no rounds that can be won it is just a global score which is computed over the size and the population density of the covered area of each faction.

The connection between the real world where the places are located which the user has to visit and the virtual world with portals takes place on a custom Google Maps like screen. The left screenshot of **Figure 1** shows the player standing in front of a portal. In order to interact with a portal the player has to come in range of 35 meters to the GPS coordinates of the portal. When he is in range he can interact with the portal. The portal menu is shown in the right picture of **Figure 1**.

Google provides a website with the so called Intel Map[3]. It shows the portals and their links in the whole world. This map is outside of the game app, which reveals only few hundred meters around the player's GPS position on the screen. **Figure 2** shows a screen shot of the Intel Map with portals and links.

[2] The numbers are taken from Ingress page in Google Play Store.
Play Store url: `http://play.google.com/store/apps/details?id=com.nianticproject.ingress`
[3] Ingress Intel Map web page url: `http://www.ingress.com/intel`



6:41AM [secure] nypc: na guten Morgen, da hat man doch mal gleich was zu tun :) stolpersteine haer freuen sich über upgrades...

10:57AM [secure] qx42: @piratechris mais pourquoi?

Figure 2: Screenshot of the Ingress Intel Map web page

Ingress was not only selected because it motivates people to move in the real world and even spend money for gasoline for their cars or tickets for public transport in order to visit some portals in the surrounding cities, but also because it brings people together. Almost every big city in the world has already an Ingress group on Facebook or Google+. People arrange appointments and play together in their cities[4].

2.2.2 Life is Magic

Life is magic is a location based dungeon crawler for iOS and Android. Similar to Ingress the action takes place on a Google Maps like map. While the map excerpt is bound to the players real world coordinates, the user does not need to come close to real world position of the objects shown on the map like in Ingress, instead he can interact with all visible objects on the map. Such interaction can be shopping for equipment and new spells or collecting treasures. Further the player can build his own dungeon and raid dungeons all over the world which were created by other players. Life is Magic has over hundred thousand downloads on Android[5] and presumably the same amount on iOS, which results in proximity of half a million downloads. **Figure 3** shows a map excerpt with different objects and figure **Figure 4** shows a fighting scene inside a dungeon.

Life is Magic is a related game of this work, because of its content generation technique which is mostly based on the users. Letting the user generate content saves a lot of work time and the amount of needed (pre)processing.

2.2.3 Parallel Kingdom

Parallel Kingdom is a WoW like Google Maps based game for Android and iOS, since this game does not need the users location coordinates it can be played on PC and Mac with a simulator e.g. BlueStacks[6]. In contrary to Ingress and Life is Magic, Parallel Kingdom does not require real world movement. The game

[4] Ingress in a TV reportage: 'ARD Morgenmagazin'
YouTube url: `http://www.youtube.com/watch?v=fCiTOYd0uGs`

[5] The numbers are taken from Life is Magic page in Google Play Store. Unfortunately, the game was shut down in the last days of this thesis. The page in the store was removed and there is no evidence left for the named numbers.

[6] BlueStacks home page url: `http://www.bluestacks.com`

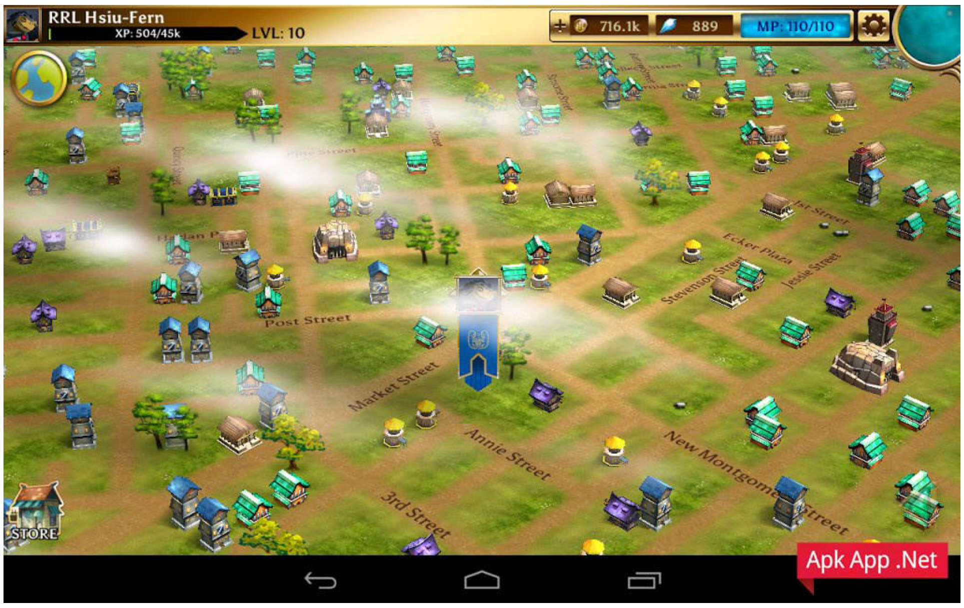


Figure 3: Screenshot with a map in Life is Magic.
Source: `http://apkapp.net/rpg-dlya-android/817-life-is-magic.html`

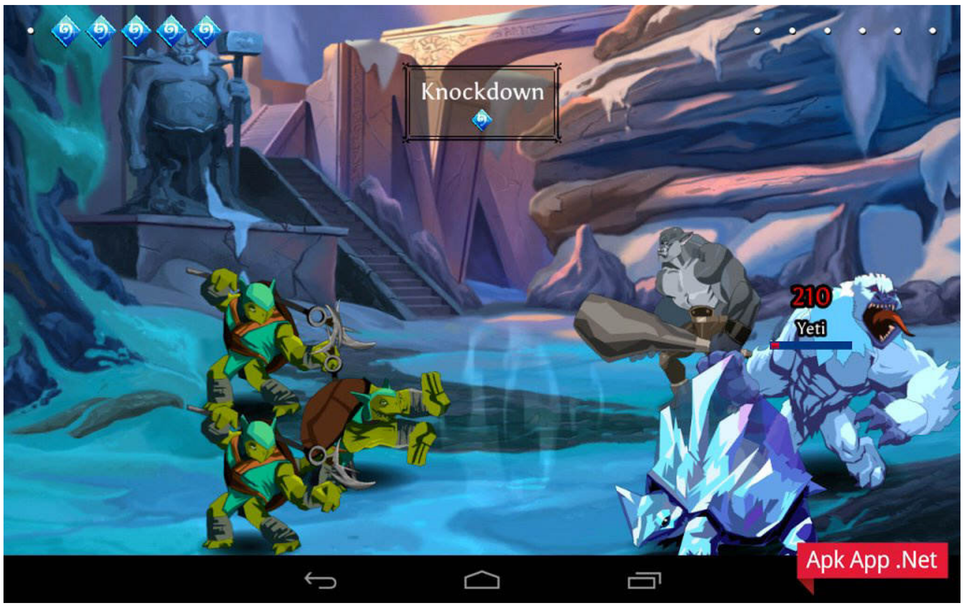


Figure 4: Screenshot showing a fight in a dungeon of Life is Magic.
Source: `http://apkapp.net/rpg-dlya-android/817-life-is-magic.html`

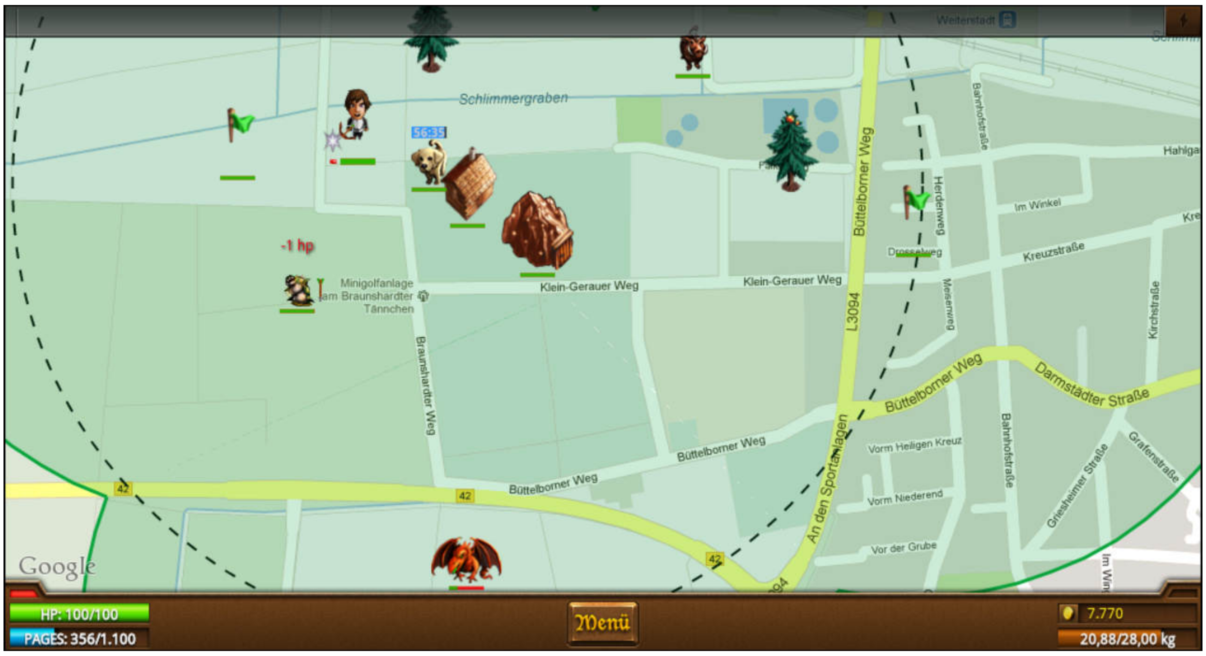


Figure 5: Screenshot of Parallel Kingdom

simply uses Google Maps as playground without further augmented reality features, however over one million downloads in Google Play Store[7] prove that even that little connection to the real world attracts several millions of players! In Parallel Kingdom the user collects items and resources, hunts monsters and claims areas for his kingdom. The multiplayer is realized through the possibility of attacking and conquering territories of other players. **Figure 5** shows a screenshot of Parallel Kingdom. Similar to Life is Magic Parallel Kingdom lets the users generate their own content through territory wars.

2.2.4 CodeRunner

CodeRunner is an iOS game, which completely differs from all the games that were introduced so far. In this game the player has to solve puzzles made by other players. First the player has to move to a location. Then he is requested to find the password on his location, it could be something like the second word in the second paragraph of a memorial plaque. After entering this word the player can proceed to the next step of the puzzle. When the player has completely solved a puzzle he is asked to rate it. This game is kind of a geocaching game incorporated into a nicely presented spy story. This game is related to this work, since it is a location based game with user generated content. Furthermore it motivates the users to move in the real world.

2.2.5 Camera Based AR Games

There are many games similar to the games presented in the preceding section, but there are even more augmented reality games for mobile phones, which use the build in camera. The most classic game type of that kind is a typical camera shooter. It projects small objects around the position of the player and shows them on the camera when the device points to the right direction. The player aims on the targets by focusing the camera on a certain spot and shoots when ready. For example DroidShooting (**Figure 6**) for Android, which has over half a million downloads[8], is a good example of such a game. In this game the player has to shoot small robots, which look like the robot from the official Android logo.

[7] Parallel Kingdom Play Store url: `http://play.google.com/store/apps/details?id=com.silvermoon.client`

[8] The numbers are taken from DroidShooting page in Google Play Store.
Play Store url: `http://play.google.com/store/apps/details?id=jp.co.questcom.droidshooting`

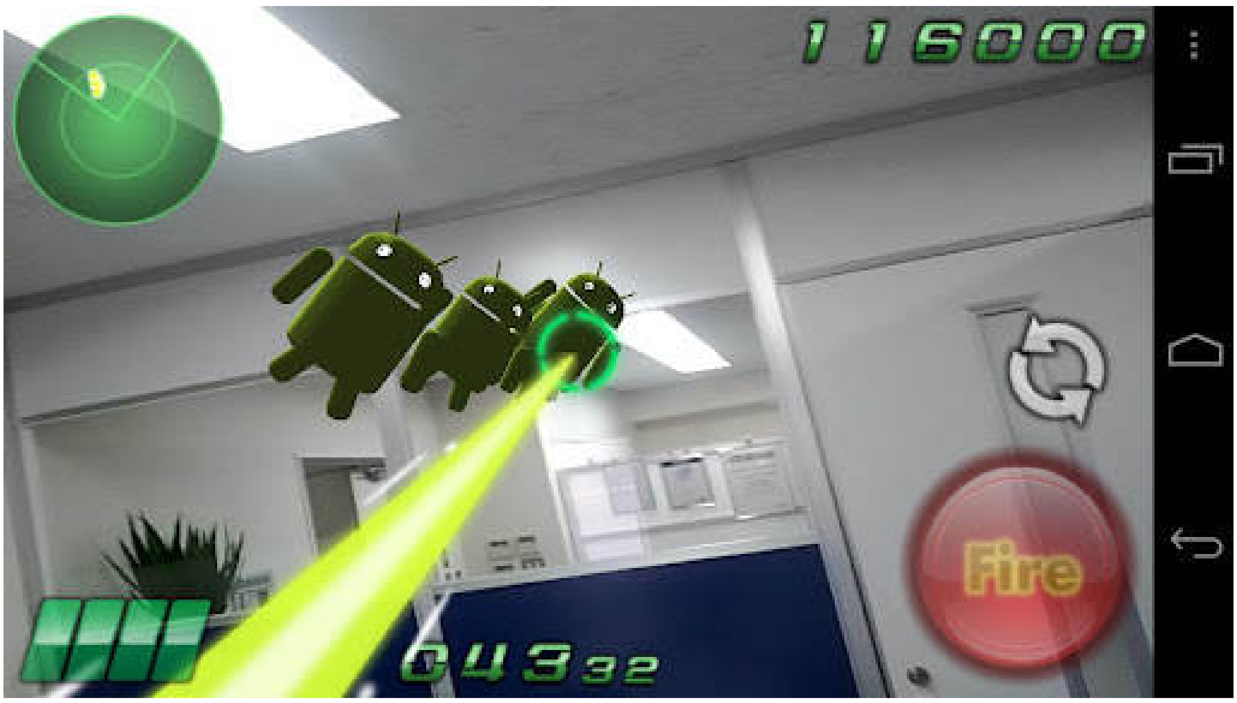


Figure 6: Screenshot of DroidShooting.
Source: `http://play.google.com/store/apps/details?id=jp.co.questcom.droidshooting`

Another example for a camera based AR game is AR Defender for iOS. This is a typical tower defence like game, but it projects the game into the real world via the device's camera. The player has to print out a simple paper marker, which the app recognizes **Figure 7**. Besides, in the newest version of the game it is possible to play together with up to 4 players on the same projected game area **Figure 8**. However, this game cannot be considered as relevant, since the game built in this work does not use the camera and more importantly there is no MMOARG based on camera yet.

2.2.6 AR Game Aspects

This section will briefly summarize, which augmented reality aspects extracted from the games named above will be included into this work's game. **Figure 9** shows an overview of the introduced games and aspects.

Strong Multiplayer

Ingress has a very strong multiplayer aspect. Doubling the amount of players walking together to a portal also doubles the power of the group. Even level one players can destroy portals of high level if they work in a big group. Actually exactly this little feature is the reason for the game's success and the amount of groups in social networks devoted to Ingress. Transferred to an AR game strong multiplayer stands for meeting people in the real world and if the people do not meet than they are at least aware of each other. Ingress brings large groups of people to meet together. New friendships are made within these groups. Strong multiplayer in an AR game has one more positive effect on the game, the players are even more committed to convince their friends to play the game, since in contrast to a usual multiplayer game where the players all over the world can play together, the players of an AR game need other players in close proximity. This is one of the most important reasons why strong multiplayer will be included in the game of this work. This AR game aspect allows to increase the viral spreading effect, which is crucially important for a small no budget game, which cannot compete with other games made by enterprises like Google Inc. Besides, the task of gathering data in order to allow traffic simulation in

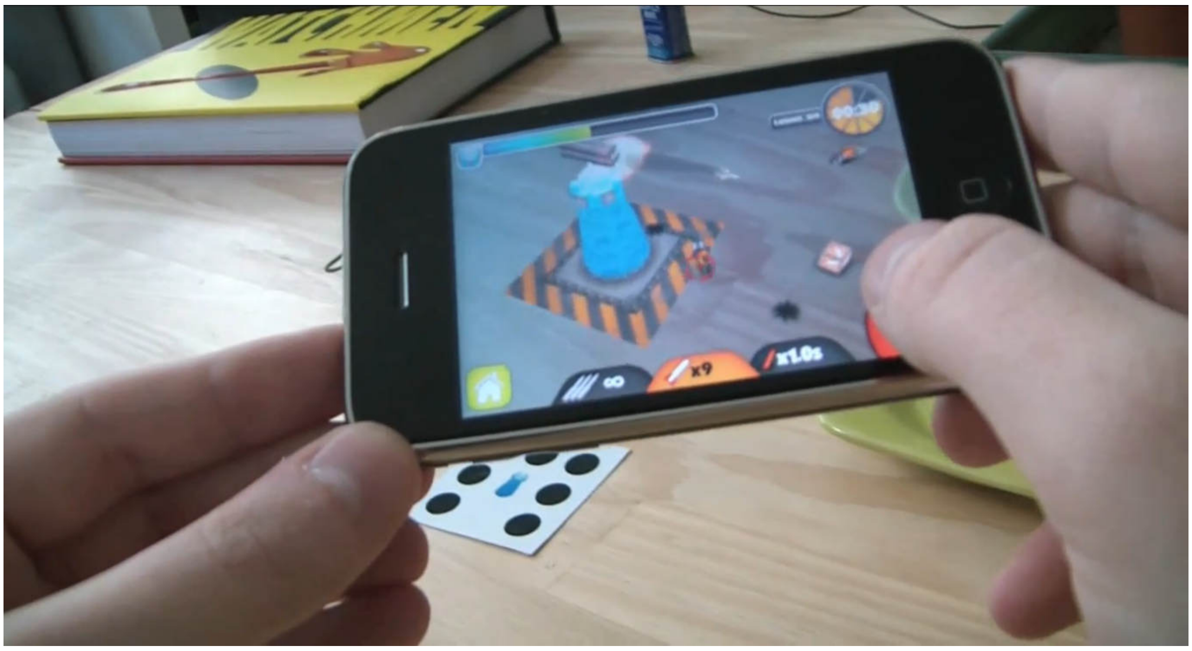


Figure 7: AR Defender 1 projected over paper marker onto table.
Source: `http://www.youtube.com/watch?v=rB5xUStsUs4#!`

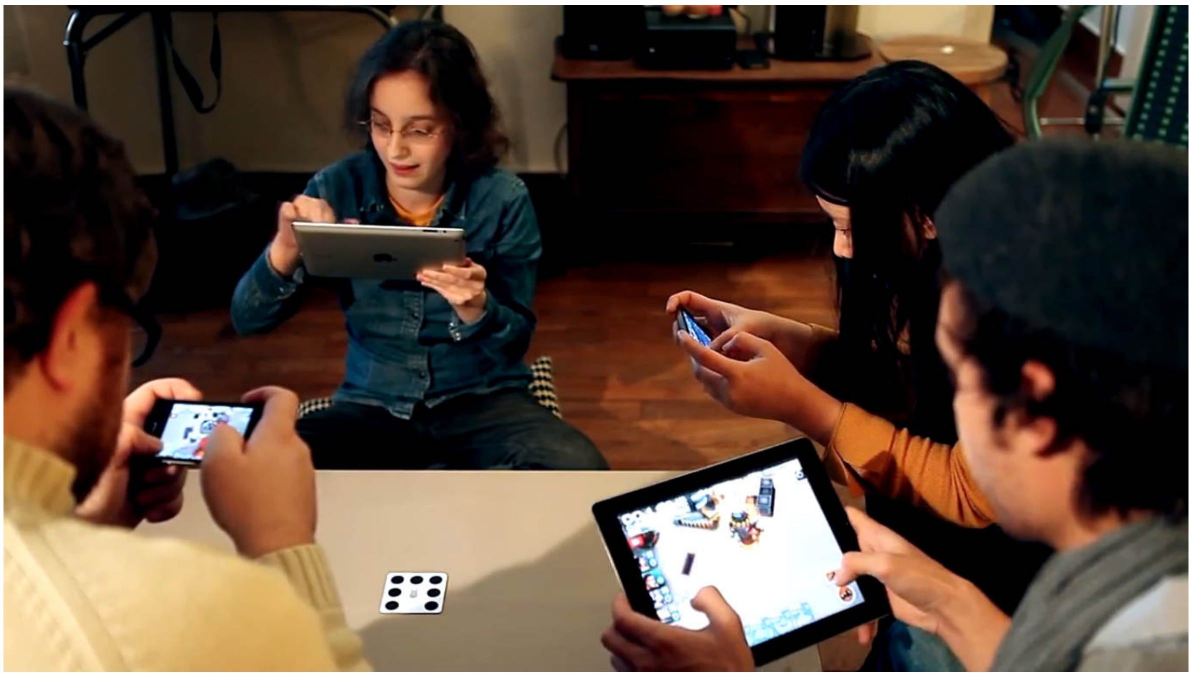
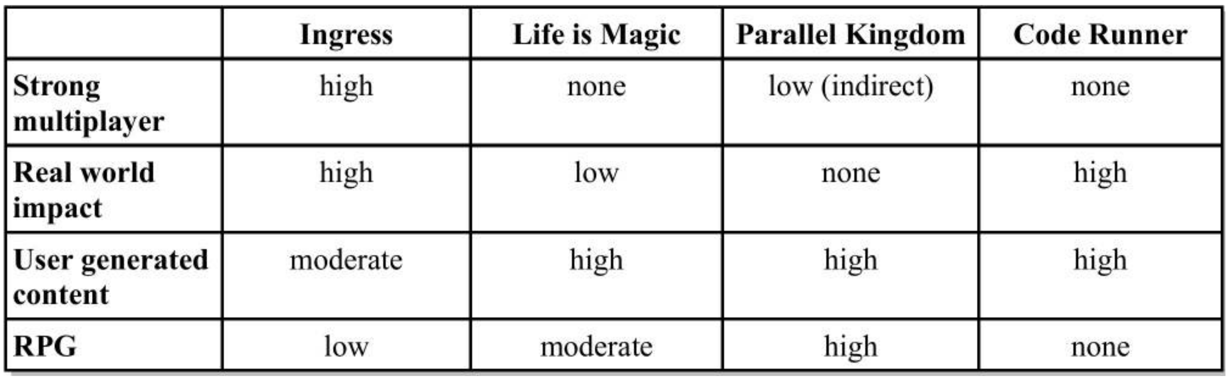


Figure 8: Four people playing AR Defender 2 on a table.
Source: `http://www.youtube.com/watch?v=0HQWZcHZLho`



	Ingress	Life is Magic	Parallel Kingdom	Code Runner
Strong multiplayer	high	none	low (indirect)	none
Real world impact	high	low	none	high
User generated content	moderate	high	high	high
RPG	low	moderate	high	none

Figure 9: Table showing the presence of an aspect (rows) in a certain game (columns)

networks is obviously requiring massive multiplayer. Furthermore, the ad hoc Wi-Fi network capabilities could be evaluated more precisely.

Real World Impact

Ingress and CodeRunner both games have a huge real world impact. These games make people walk around and gain awareness of their environment. Furthermore, Ingress has an educative effect, since most portals are placed at historic places and the players are forced to visit them and probably learn something about the places. The demand for such games is growing, which can be seen in the amount of geocaching games and Ingress's popularity. This novelty of projecting games into the real world or even the everyday life could also be very helpful for a little game to become successful being one of the first games of its kind.

User Generated Content

Life is Magic, Parallel Kingdom and CodeRunner all these games have location based user generated content. Even Ingress allows users to create new portals, even though the default portals, which Google generated using historic databases would make the game playable. CodeRunner makes users even rate the content created by other players. Without the user generated content it would be impossible to fill the whole world with unique personalized content or it would be extremely expensive. As to the fact that dedicated content generation would be unaffordable, user generated content will be incorporated into the game. Furthermore, the data from user based content generation could be helpful for simulation purposes such as evaluation of load balancers and searching for critical load capacity of networks. Also the MAKI research would have a solid evaluation database for tests of new network technologies.

RPG

Life is Magic, Parallel Kingdom and Ingress have many role play game (RPG) elements. All of these games have the element of progression, since players can level up and improve skills and abilities. These games have also an exploration gameplay part. In all three games the players collect items and resources. Parallel Kingdom has also a very high personalisation factor. The player can decide in which direction he skills his character. RPG games are very popular, from beginning from mid-70th[12] they have never disappeared from the market. These games satisfy the human need for personalisation and progression. The game developed in this work shall also use the approved gameplay of an RPG game to a certain amount.

3 Game Design

The following sections motivate the game type decision and point out similarities and differences to other existing games of the chosen genre. Hereafter, the mechanics of the game are described and their purpose is explained. The writings in this passage do not cover technical details and are intended to be understood by readers without technical background.

3.1 Concepts

This section first motivates and explains the game type decision and then sums up the basic concepts of the game.

The game of this work requires a unique gameplay, since it cannot compete with games made by enterprise corporations like Google or game studios. The inspiration for such unique gameplay is Anomaly War Zone Earth. Anomaly is a tower offence game and it is the first successful game of its kind. It was released in 2011 and won the Apple Design Awards 2011[13]. The game was published on all popular platforms such as Windows, Mac OS, Play Station 3, Xbox 360, Android and iOS. In Anomaly the player controls a group of units that try to reach a certain destination. The path which they have to move on is protected by stationary towers which shoot at the units as soon as they are in range. The player can decide the root which his units take to reach the destination. Furthermore, the player can reinforce his troops by dropping items. For example he can heal up his troops or he can hide them for a short time from towers. The number of items is limited and the levels can be accomplished only with smart usage of all items. Anomaly has a very new gameplay that has not yet been copied by other games. It can be compared to League of Legends (LoL) and Warcraft 3 Defence of the Ancients (DotA). In LoL and DotA the player controls a hero, which helps many endlessly spawning units to destroy the enemy's base. In contrast to LoL and DotA the unit number is limited and the player himself cannot attack towers in Anomaly. While LoL and DotA allow to improve the hero which is the most important part of the game in Anomaly the units, which are in focus of the game, can be upgraded. Another big difference is that there is always only one attacking and one defending party in the game, while in LoL and DotA every player can either defend his towers or attack enemy towers. Furthermore, there is no AR game based on a tower offence gameplay.

Tower offence and defence game types suit very well for our purposes. They allow to cover all important augmented reality game aspects mentioned before. Letting the players build their own tower defence levels, which would be played by other players satisfies the user generated content aspect. The RPG capabilities are very high. A level system can be introduced. The higher the level, the stronger are the player's troops and the towers he can place in his levels. Exploration aspects of RPG games may be realized through an open world, which could contain special items, game money or units. Strong multiplayer is easily realizable by allowing multiple players to play the same map simultaneously. The players would use the combined troop forces, so that even players with a low level can accomplish a high level map if they join forces. Also the real world interaction opportunities of a tower offence game are huge. The players need to move in the real world and they need to build groups in order to join force and grow their power.

3.2 Gameplay Combat

This section will describe the combat related game mechanics. The combat part of the game satisfies the tower offence game type. In the game the player can create troops on his position and move them towards a target point, while defence towers shoot at them. The target points are energy sources which give the player various advantages if he captures them. The currencies of the game are energy and power. While energy is recharged over time, power depends on the player's level and is rather static. Energy is the currency to pay for interactions and power is the currency to unlock game content.

3.2.1 Troops

First of all the player has to define a position on the map at which his troops will enter the game. This position is called spawn point. The player has to move to the GPS position of the desired spawn point. A valid spawn point must be out of range of all towers and energy sources. When the player's GPS position is a valid spawn point location then the app allows choosing this position to be the spawn point. Once the player has selected a spawn point his troops will be placed around this position. Spawning units cost a certain amount of energy depending on the type and count of the units. If the player has spawned units on the map then all his movements are recorded in a location trace. When the last unit is destroyed the location trace is also destroyed. The location trace is visualized with a flashing line. Once the spawn point is selected and the troops are spawned they will follow the player. They can move on the GPS trace of the player. Every unit has its maximum velocity, therefore if the player moves too fast his troops will need some time to follow the path he has travelled. This mitigates the advantages of players riding a bike or driving a car, which is undesired since it could be very dangerous for them and their environment. If the player closes a cycle in his location trace then the last position on the trace before the cycle and the end of the cycle are connected, all other points on the cycle are removed. A unit positioned on a cycle in the player's path, which is now closed and removed, moves to the origin of the cycle, which is the position where the cycle was closed. Troops can neither collide with each other nor with the towers. Hence, it can happen that two units are drawn on the same position. The first iteration of the game did not allow units to intrude in the graphical representation of each other, which should make the game look more realistic. However, many problems arose from this for example a big unit would block and slow down small units. Another problem was that many units could not move through a narrow serpentine road, because the units after a turn would collide with the units before it. Therefore, a collision less world was the right way to go. **Figure 10** shows screenshots with troops and towers in action. Another little detail of the iterative design is that the unit move in front of the player. The reason for this is that in the first iterations of the game the test players had the feeling that the units are too slow. The feeling arose when the players reached the energy source, but still had to wait for their units. With the units moving in front they always reach the source before the player does. The units move with a random offset to the player, this way the units move with a little distance to each other so that they all are visible, since otherwise they would all be on the same spot.

The movement of the units could have been done different, for example it would be possible to let the player first fully define a GPS trace and then let the units spawn and move on it. This way he could do some further interactions, while his troops are moving, such as using items or spawning reinforcements. In addition, this way it would be possible to let the troops move faster and let the game look more dynamic. However, this would make the game more static in the real world. The player would stay for a long time on the same position, while his troops are moving. The feeling of playing an interactive game would be destroyed and even the feeling of watching a video without having any chance to intervene in the happenings could come up.

3.2.2 Towers

The next important part of the combat game are the towers. Once a player's unit is in range of a tower it will start shooting at the unit. It will stop shooting as soon as the unit is out of range. A shot fired while the unit was in range will always hit even if the unit leaves the tower's range during the weapon animation. The player's marker which indicates his GPS position cannot be attacked by towers. Units can be destroyed by towers. Each unit has a certain amount of health points. These health points are reduced every time when a tower shoots a unit. A unit is destroyed when its health points fall below one. The health points of a unit are visualized with a bar over the unit. This bar is green when the unit is not damaged and completely red when the unit has no health points left, see **Figure 10** for more details. The death is visualized with an animation and the unit is removed from the map. The death animation must clearly show that a unit is lost not only to give feedback to the player, but also to emphasise that

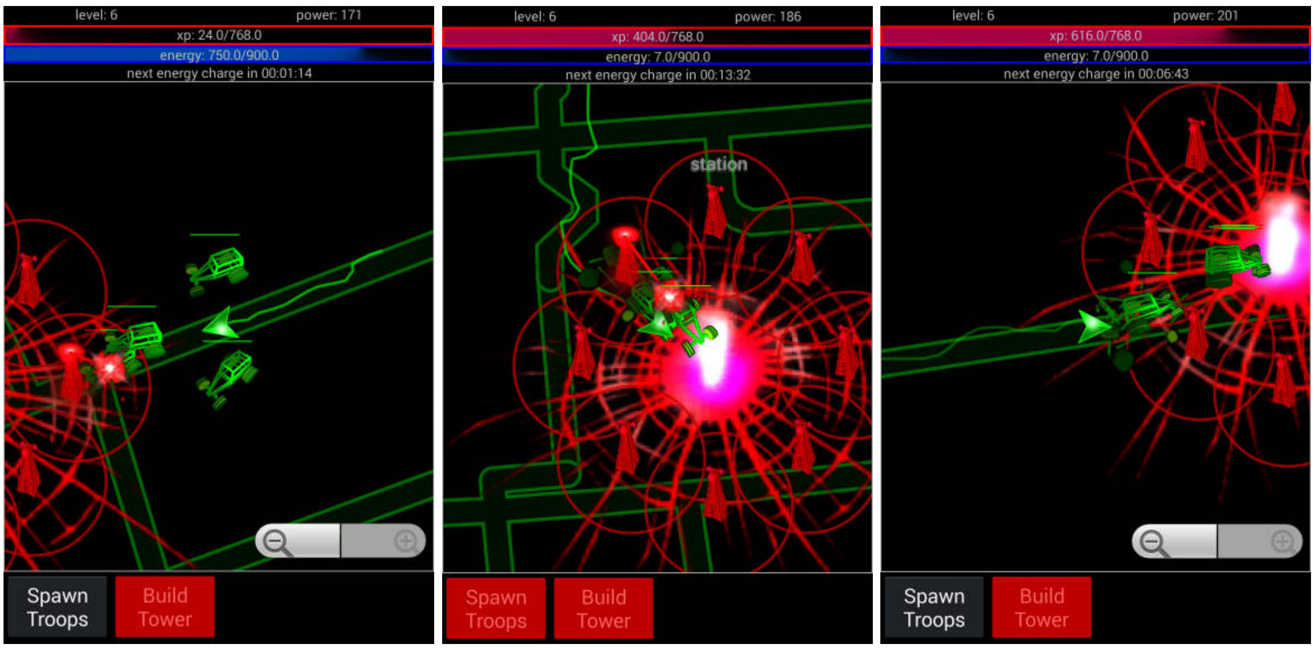


Figure 10: Screenshots showing units that follow the player to an energy source. In the first two pictures a tower shoots a unit. In the second and third image the remains of a destroyed unit are visible.

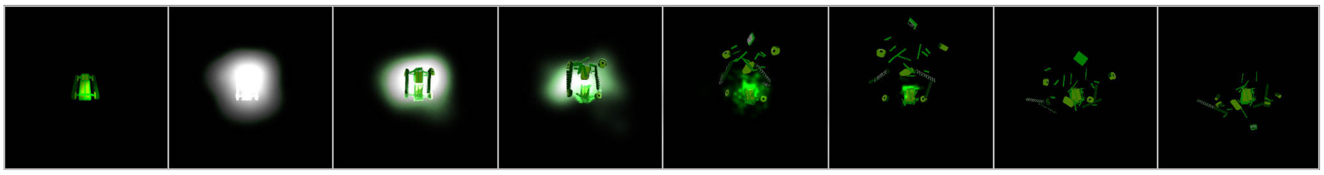


Figure 11: Some frames of the explode animation of a unit.

something bad has happened and the player should try to avoid this situation next time. Furthermore, the animation must be very long, because the player is moving in the real world and the game cannot expect to have his attention at all time. This is why the remains of a dead unit must stay on the screen for a while. **Figure 11** displays some frames of the death animation of a unit. The animation starts with a very bright white explosion, which stands in a singular contrast to the usual dark game visualisation and therefore, attracts player's attention.

3.2.3 Capture Condition

The player has captured the energy source if at least one of his units survives long enough to follow the player to the source's position. Once a player captures the energy source his units are recycled. Owned energy sources cannot be captured multiple times, they also cannot be used to recycle spawned troops. What advantages it brings to let as many units survive as possible and recycle them is explained later in this chapter. The process of capturing an energy source is illustrated in **Figure 12**. Energy sources captured by other players shortly before the player captures them cannot be captured either, since they are in safe mode. The safe mode prevents race conditions, such as players with high latency capturing targets that are already captured by other players. It also solves the conflict of multiple hostile player capturing one target simultaneously since the first player wins.

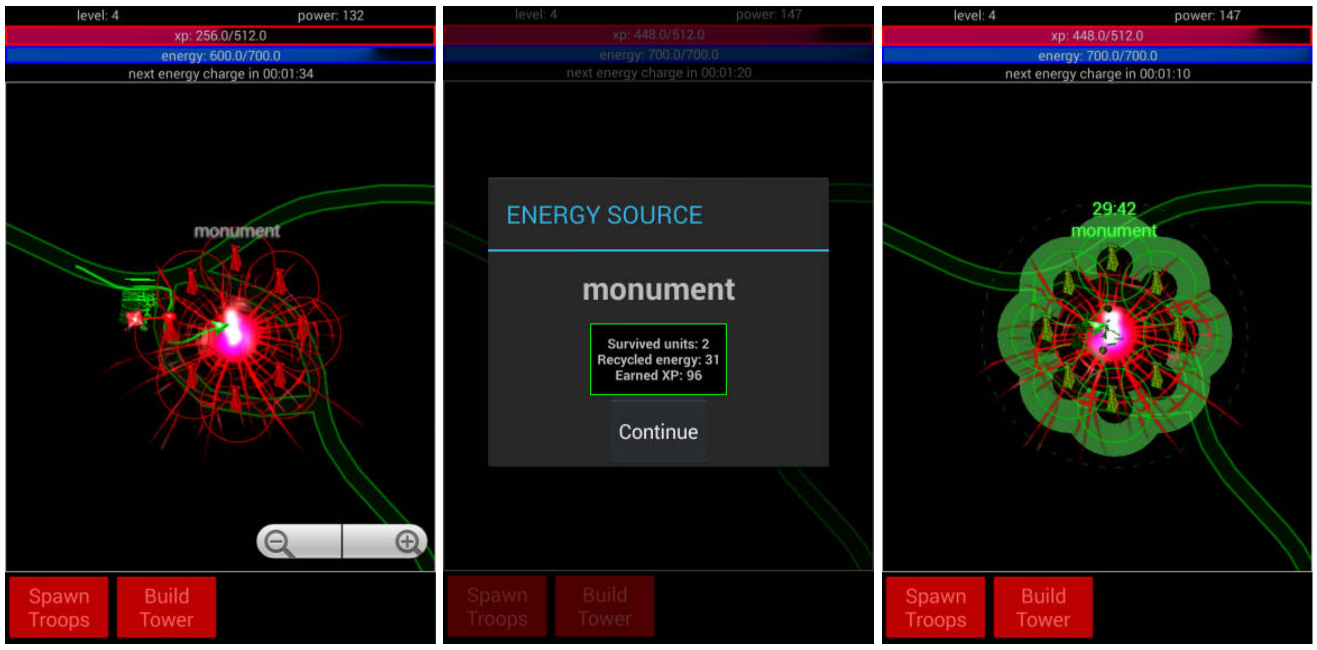


Figure 12: Screenshots of an energy source being captured. The first picture shows units approaching the source. The second picture shows the capture dialog. The third picture shows the captured energy source with the timed recharge count down and the maximal tower build range.

3.3 Gameplay Strategy

Below, the game mechanics of the strategic part of the game will be explained. Strategy gameplay does not only provide the room for the player to customize the world, but also allows him to use his environment in a smart way. He could block easy access paths in the real world with many towers in the game world so that the energy source will become hard to reach in the physical world since the normal paths cannot be used any more like it is exemplified in **Figure 13**. Furthermore, the build-up gameplay satisfies the human desire to create things and improve them. Finally, the most important strategic part of the game is the social gameplay. Players can group up and benefit from the combat power of the other players.

3.3.1 Building Towers

Towers can be built for captured energy sources. A valid position for a new tower is the opposite of a valid spawn position, thus towers can only be built in range of other towers or the energy source. Newly built towers are not removed if another player captures the target. Assuring this, softens the negative feedback of having lost an energy source, since everything that the player has built up can be captured back. Building a tower uses a certain amount of the player's energy. In addition, implied through the tower count limitation described below, building a tower requires a certain power level.

If the count of towers around a target would be unlimited, then infinitely hard levels would be possible, so that even players with the highest level would have no chance to capture the targets. Besides, more problems would arise. On the one hand the tower density would be an issue, since it would be possible to create an infinite amount of towers on the same location, which would make a comprehensible visualisation of such a place impossible. On the other hand the tower spread would bring trouble, because different energy sources could have their towers on overlapping regions. In this case the towers of the one source would protect the other energy source. The problem gets even worse with the time, since the world would only consist of towers and the players would not be able to move with their troops. Surely also here it would be impossible for low level players to capture any target.

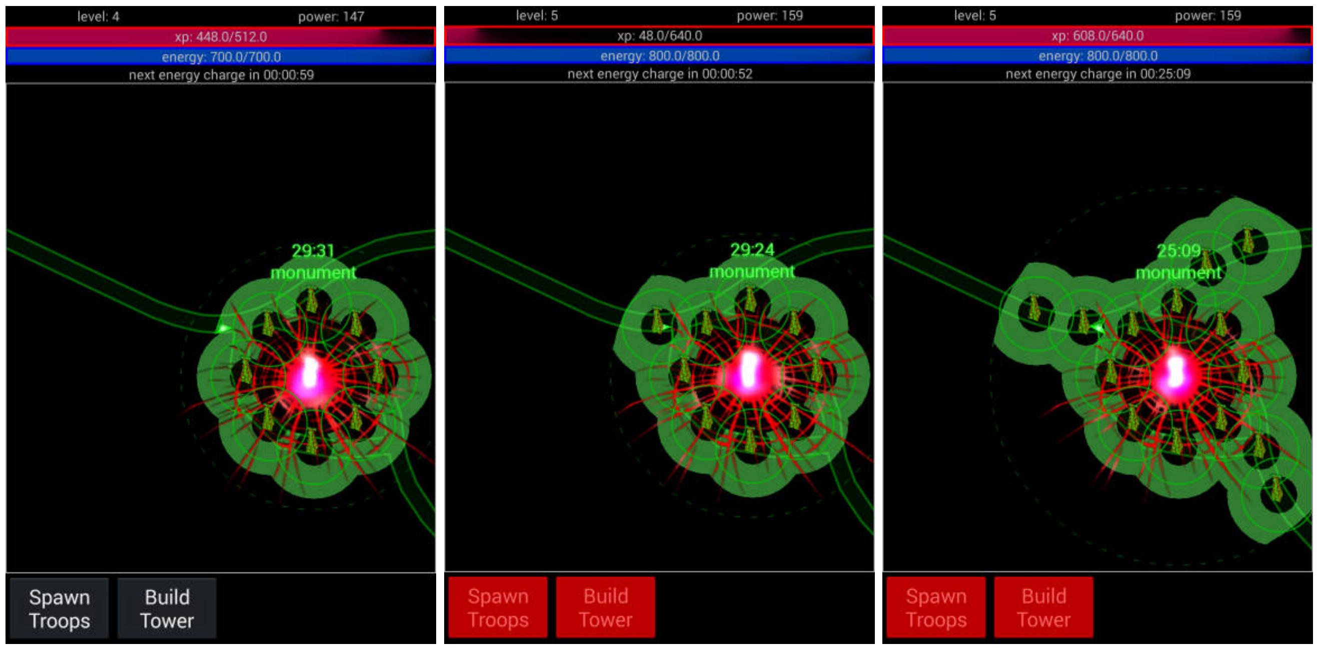


Figure 13: Screenshots demonstrating the process of building towers to protect a captured energy source. In the right picture all main access roads are blocked.

Having said that, it becomes obvious that three issues need limitations those are the maximal tower count, the tower density and the tower spread of an energy source. To limit the tower count players are only allowed to build a new tower if the overall count of towers for the energy source is below a certain amount, which depends on the player's power. This basically means that only players of a higher level will be able to create new towers. This mechanic is a further motivation for the players to achieve a higher level and seek for ways to increase their power. The limitation of tower density is simply achieved through a minimal tower clearance that needs to be complied with when building a new tower. This distance is much less than the range of a tower. Finally, the tower spread is limited through a maximal range to the energy source, which must be less than the double of the average distance to all built towers. Additionally, it is forbidden to build a tower in the range of any tower of another energy source or in the range of the source itself.

In the description above again decisions have been met. Such as creating an open world where all energy sources and their towers are placed. It would also be possible to create levels, which must be activated. This way more energy sources could be placed in the world, because they could share the space for their towers since only one tower level would be visualized and played at once. Furthermore, it would be possible to create bigger levels, since there would be nothing to interleave with. However, the size of the levels was reduced after every play session of the game, because users simply did not want to walk far distances. It turned out to be more fun to have many smaller levels in a city than having only very few, but very big levels. Despite the fact that the distance which the players walked during such a session was always almost the same it felt more rewarding to capture more sources in the same time. Having said that, dividing the world in separate levels becomes less important and considering the additional effort, which separated levels would need, the open world design was the one to choose.

3.3.2 Building Groups

If two players are close enough to each other then they will be asked if they want to group up with the other player. The group is build when both agree. During this process the players can click on a button to read more about the group mechanics. The group size is unlimited. When a group of players meets a

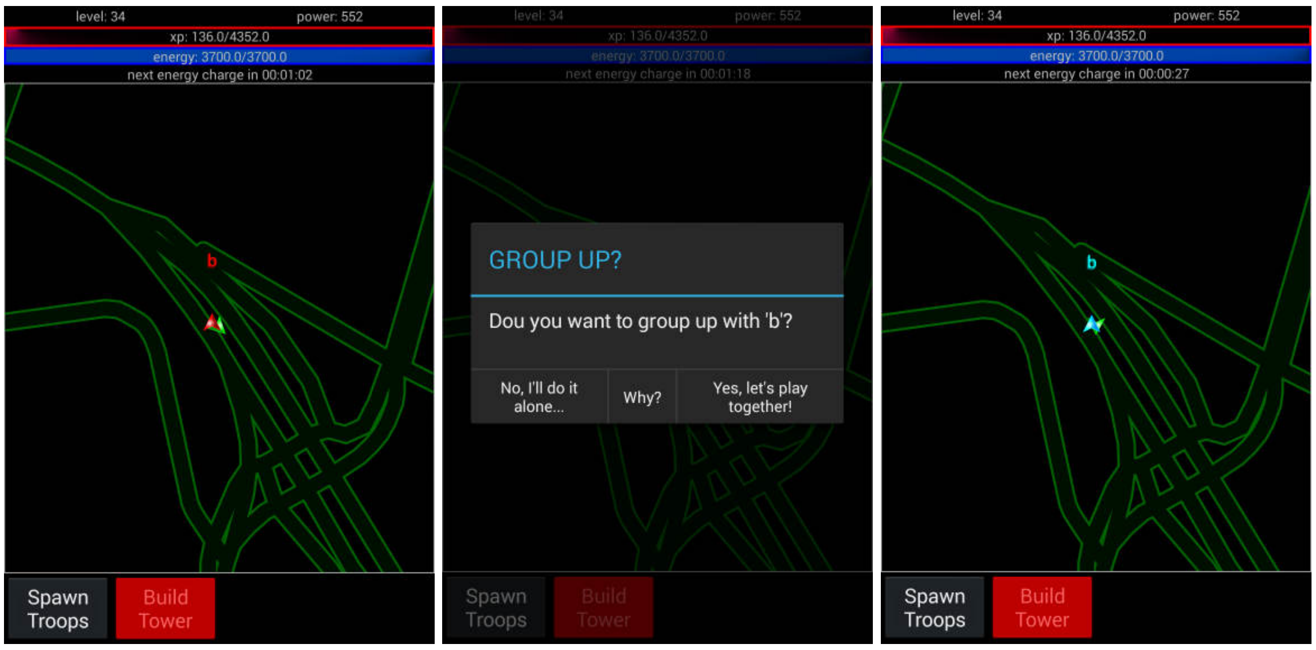
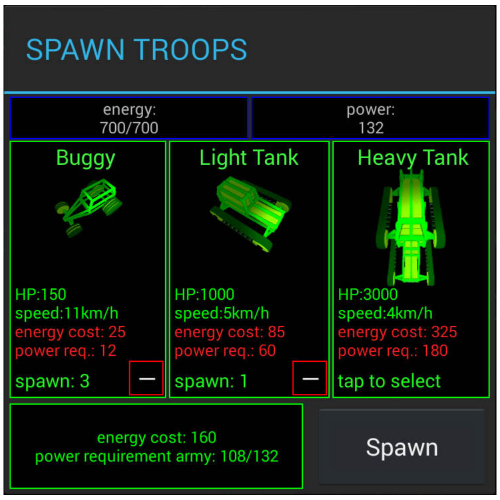

level: 34
power: 552
xp: 136.0/4352.0
energy: 3700.0/3700.0
next energy charge in 00:01:02
b
Spawn Troops
Build Tower
GROUP UP?
Dou you want to group up with 'b'?
No, I'll do it alone...
Why?
Yes, let's play together!
next energy charge in 00:01:18
next energy charge in 00:00:27

Figure 14: Screenshots of player 'a' and player 'b' going through the group up process. The left picture shows the players coming close. Being in the group up range bough players see the group up dialog displayed in the middle image. The right screenshot shows the view of player 'a' where 'b' is not displayed red anymore, but has a cyan colour.

single player and at least one player from the group accepts the group invite then the single player joins the whole group. Similar mechanics are applied when two groups encounter. If at least two players from different groups accept the group up then the groups are joined. The group up process is visualized in **Figure 14**. A player does not leave the group if his device goes to standby or his game crashes during a time period of ten minutes. The time out resets as soon as the player reconnects. The time out is the only way for a player to leave the group. Forcing this allows groups to split up and work in different locations simultaneously. This comes handy if the group consist of many high level players that can capture targets on their own, this way they can capture more targets in less time and the social component of meeting is met during group building. Furthermore, it is impossible to make any fraud like exiting the group on the last mile and capturing an energy source only for oneself. The group profits from any combat action of any player.

3.3.3 Group Advantages

All players within a group become the owners of an energy source that gets captured by any player of the group. Therefore all players get the bonuses which the energy source generates including the timed recharge with the full amount. Any player that was or still is in the group and now owns an energy source that has multiple owners can build new towers to protect the property of the group. Later group building does not provide ownership of energy sources that are owned by some players of the group. Hence, it is possible for two players in the same group to see an energy source as owned for one player and as owned by enemies for the other player. The player that not yet has captured the energy source must capture it alone or with other players of the group that have not yet gained the ownership on it. All this has the advantage that players have to meet at the same time again and again, since otherwise they would steal the ownership from each other if the group is not complete. This is the reason why there are no permanent groups like guilds, clans or gangs where the players would not need to meet each other after the first encounter. Furthermore, playing together allows gaining more XP and getting a higher


SPAWN TROOPS

energy: 700/700
power: 132

Buggy
HP:150
speed:11km/h
energy cost: 25
power req.: 12
spawn: 3

Light Tank
HP:1000
speed:5km/h
energy cost: 85
power req.: 60
spawn: 1

Heavy Tank
HP:3000
speed:4km/h
energy cost: 325
power req.: 180
tap to select

energy cost: 160
power requirement army: 108/132

Spawn

Figure 15: Spawn troops dialog with three unit types.

level very quick. Every player of the group that was in the range of an energy source's towers and had some troops alive will get XP for assisting when it is captured. There is a certain time limit between the first player being close to the source and the second player capturing it. If more time has passed then the first player will get no assist XP.

3.3.4 Spawn Units

There are three different unit types that can be spawned by the player. These three types allow the player to adapt his attack strategy to the placing of an energy source's towers. The units differ in their speed, size, amount of health point, energy consumption, power requirement and visualisation. The first type is fast and small with lowest health points amount, but also has the lowest energy consumption and power requirement. This is the unit type that every player can afford in the beginning of the game. The second unit type is slower and bigger than the first one, but has much more health points. The player needs to increase his initial power to afford this unit, also the energy consumption is higher. The third unit type's attributes are calculated in a similar way, but all differences to the first unit type are increased in comparison to those of the second unit type. Consequences are that players will want to spawn the later unit types, however these will be more expensive. The player will need to estimate the amount and the types of units that he has to spawn in best possible way, so that he does not spend too much energy. **Figure 15** shows a dialog that allows the player to choose between the three unit types.

3.3.5 Spawn Limitations

Again if the number of units that a player can spawn would be unlimited, then a player with some energy sources could spawn an infinite amount of units. This could be done because of the timed energy charges from the energy sources. A player could spawn troops, wait for the next energy charge and spawn more troops. Players should not do that, because on the one hand it is very boring to wait for the energy to recharge and on the other hand there would be no difference in the strength of the players, but a difference in the time that they need to spawn an army with a certain strength. Therefore, already spawned troop need a certain portion of the power that the player has. In order to spawn a new unit not only the power requirements of this unit has to be met, but the army's power usage including the new unit must be covered by the player's power reserves. The following formula turned out to be the right one to calculate the army's power requirement:

$$\sum_{n=0}^{\text{unittypescount}} \text{unitpower}_\text{n} 2^{\text{unitcount}_\text{n} - 1}$$

where $unittypescount$ is the count of different unit types in the currently spawned army, $unitpower_n$ is the power requirement of one unit of type n and $unitcount_n$ is the number of units of type n. Following to this formula the power requirement to spawn one unit of a certain type is the power requirement of the unit. The second unit spawned doubles the power requirement, which is still affordable. However the third unit already quadruplicates the power requirement compared to only one unit, while the fourth unit would octuple the power requirement. This way the number of spawned units for each type will be somewhere between one and four. This is needed to prevent players from spawning thousands of cheap units and be able to win the game only using the first unit type. Now the player has to rely on stronger unit types, because his troop resources are very limited. Also the combination of different unit types is favoured by this formula, since spawning two units of each type requires exactly the sum of the power requirements of each unit and not more.

3.3.6 Possible Extensions

The strategic game mechanics described above resulted from various game test sessions. However, they can still be evolved and extended. For example it could be a good idea to extend the described behaviour with the possibility to move, to upgrade and to build different towers. Moving towers will become interesting when the energy source has reached a certain amount of towers that can be managed only by very highly leveled players. They would need to rearrange towers, so that they do not collide with other energy sources and meet the tower spread requirements described above. Upgrading towers and having different types would allow more strategy, customization and variety in the world. Also upgrading an energy source and making it generate more energy or additional power could be interesting. Besides, players would have a positive cooperation, since they would capture the energy sources from each other (which is negative), but they would spend their resources to improve the energy source cooperatively, so that they will fight for the source, but get an advantage from energy sources which are fought by many players, since they will be highly upgraded. Another possible extension would be to introduce exploration like gameplay. To unlock new troop types the player would need to find and visit certain places, so that he would be motivated to use the game in every new location he passes by and also explore his own home city. Units could also have different upgrades, which could be found in such places. Furthermore, attacking units could be introduces. They could be able to destroy towers. On the one hand this would create more motivation for low level players, since even if they cannot capture an overprotected source, they could at least damage its defences or capture the source after several days. On the other hand it would be not possible for a place to become static after all tower capacities are used up.

3.4 Energy Source Generation

In the following the algorithms and mechanics used to generate the energy sources are explained. The locations at which the game takes place should be interesting for the players. Motivating the players to mark interesting locations where they would like to see an energy source placed would be the best way to solve this task. However, new players at new locations would not find any energy sources to play. Google's game Ingress has a great solution for that: first define an initial amount of energy sources (portals in Ingress) using for example databases over historical places and then continuously allow players to mark locations and create energy sources at locations that were marked frequently. Unfortunately the amount of players during the time of this thesis will not be high enough to sufficiently use a majority vote system for new energy sources and therefore this mechanic will remain a future work point of this thesis. For the initialisation of energy sources in this thesis the tags contained in the open street map data will be used. For example historical or tourist attractions can be selected, if there are no such places found train stations can be used as fall back.

While the open street map tags are parsed, every suitable location will be prioritized by attraction, so that a tourist sightseeing spot will be more important than a train station. Whenever a map tile is in a certain range of a player e.g. 5km the energy sources on this tile will be created if they were not created already. To do so, the list of potential energy spots will be iterated, starting with the most important entries. If the entry is not within a certain range e.g. 500m of any other already created energy source, then it will become a new source.

If the initial energy sources would stay unprotected then it would be too easy to capture them. The players would capture one source after another and get bored quickly because the game would be unchallenging. Besides, in order to build towers to protect their energy sources the players would need more resources than they have. Therefore the initial sources need a protection. A certain amount of initial towers protecting a new energy source will be created.

3.5 RPG

Below the role play game (RPG) elements of the game are explained. They are needed to provide a long term motivation for the player and let him discover new content slowly so that he does not get bored of playing the same game over and over again.

3.5.1 Level

A classic RPG level system is included. The player gets experience points for most of his actions. Those are spawning own units, capturing an energy source, building a tower and joining or creating a group.

With the increasing level the amount of experience points needed to level up also increases. Hence, the player advances in his level quickly in the beginning of his game, but the further he gets in the game, the slower he will increase his level. Higher level will unlock features of the game (e.g. new and more powerful units or more towers).

The level system serves two main purposes. On the one hand the player discovers the game slowly, so that he has time to learn the game mechanics and is not overchallenged with the amount of opportunities he has and on the other hand the long term motivation to continue playing the game will be assured through continuous discovery of new content and new opportunities to interact with the game.

3.5.2 Power and Energy

Power and energy are the two currencies of the game.

The game will take place in a world where almost each action will reduce the current energy reserves of the player. The amount of energy needed to spawn a unit depends on its type and power, while towers have a fixed price. Each player has a certain maximal capacity for energy that can be stored. If the limit is reached, but the player gets further energy charges the overhead energy that exceeds the player's capacity is lost. The maximal capacity is increased with each level and the current charge is refilled to maximum every time the player gets a level up. The energy reserves are recharged after a certain amount of time by a certain small value, however this small value is not enough to play the game extensively. It ensures that no player is caught in a situation where he cannot earn any energy and is caught in a closed circle situation where he needs to do actions to recharge, but these actions would consume energy so that the player cannot do any actions, since his energy reserves are empty. Moreover, the timed recharge is increased with the amount of owned energy sources, which is one of the motivators for the player to capture as many energy sources as possible. Each source generates energy every 30 minutes, the generation countdown is visible for owned sources see **Figure 12**. Therefore, a player that owns many sources can use more expensive and better units and build more towers, since he will have plants of energy reserves to spend. Energy is instantly recharged when an energy source is captured by the value, which would be applied for this source in the timed recharge. If the player is highly motivated and has used up his energy then he has a last opportunity to collect charges at certain places. **Figure 16** shows the player in front of such a place. Such a place can be visited only once per twelve hours. The location of such places is generated in exactly the same way like it is described in the **Energy Source Generation** section, however it is ensured that the selection tags do not collide with the tags used for energy sources

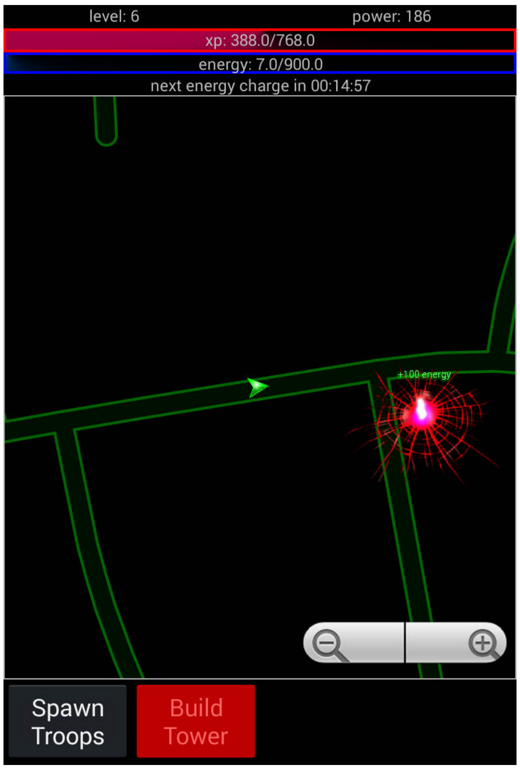


Figure 16: The player has almost no energy left and stands in front of a free energy source which can be collected once per twelve hours.

(e.g. restaurants could be used, since the player could be hungry after his long game session). While the players see those places on the same location the visit period is calculated for each player. This allows a group of players to collect the same charges so that running in a group has no drawbacks.

The second currency of the game is power. Units and towers are unlocked with a certain power, therefore power defines which troops and how many towers can be created. Power is not consumed by actions. The base amount of power is defined over the level of the player. However, power can be additionally increased through owned energy sources. The more sources the player owns the more additional power he will have. All together power and level up are used to unlock new content. Players that have played the game for a long time will unlock the content with their base power, even if they are not that successful. Players that have not played for a long time, but are very successful are rewarded with temporal unlocks of content, which need more power than the base power of the successful player and are unlocked only as long as the player holds enough energy sources.

To shortly summarize what power and energy do, it can be said that energy limits the amount of actions which a player can do and motivates him to think about what he does and makes the game more challenging. Power in contrary unlocks new content and ensures the long term motivation to play the game and immerse deeper and deeper into the game's world.

4 Implementation

This section documents the code written during this thesis. It is designed to give the reader an overview of the different implemented systems that are needed to make the game work. The writings below, illustrate what steps are required to run the game, which servers need to be started and which client changes are required to allow the game to connect to these servers. Below important code is explained, which is meant to be used as a starting point for changes, extensions or integration in different environments. In this chapter first some basic data type definitions are made. These will be used later to explain certain design decisions. Subsequently, the system design and its software architecture will be introduced. Then the decision to use Android with mapsforge[11] is explained. After that, the Android activities of the game and their transitions are introduced. Afterwards, the client implementation and the use of the Model-View-Controller (MVC) pattern is described. Then the need for the two different servers is explained and their implementation discussed. Thereafter, the data types of the game are enumerated, classified and explained. Finally, the reusable components and the components which might be changed during further development of the game will be labelled.

4.1 Data Type Definitions

In the following the data properties and the resulting data types are described and named. The influence of these on the game's network interface is explained later in this work. The definitions below will make it easier to understand certain system design decisions. This section is based on the game design described in the earlier chapter, therefore no technical knowledge is required to understand the text below.

4.1.1 Properties of Data

Data can have different properties and require different quality of service (QoS). Below the properties and QoS requirements used to categorize the network data generated by the game are briefly introduced.

Life Cycle: Persistent vs. Temporal

On the one hand, there is data that needs to be saved in a persistent way (e.g. player's name or energy resources), and on the other hand, there is data that is only relevant to currently interested entities (e.g. position updates of surrounding players). First off all, persistent data requires **availability** to all users, it must be possible to poll for such data at any time and from any location. While temporal data is a one shot event, persistent data must be **changeable**. The player might want to change his name or spend energy for a unit. All queries for persistent data must return the same value and reflect the same changes in a **consistent** way. Take into account that persistence does not imply that data is changed with low latency, but that data once it has changed, even if the change needs some time, is consistently changed in the whole network.

Propagation: Real-Time or Indispensable

While real-time data must be sent with lowest possible **latency**. Indispensable data requires **durability** and cannot be lost, it must reach its destination(s). Real-time data can be lost in worst case, but should not. For example position updates are sent periodically it is better to skip one than to resent an outdated position update. In contrary, a level up is indispensable and must be reflected even if it takes longer or some intermediate nodes of the network crash.

Area of Interest (AOI)

There is geographical position related data, which can be used with **range** queries or subscriptions. Every entry that supports the range functionality of the AOI data type has a geographical position represented with the values of longitude and latitude.

4.1.2 Data Types

This section enumerates the network data classes of the game. Every used data type can be classified to exactly one data class. The network data types used in the game will be enumerated and classified later in the **System Design** chapter.

Temporal, Real-Time, AOI Data

An example for this data type are position and state updates on the player and his troops (e.g. position, spawned units, captured targets). Such updates are distributed within an area of interest (AOI), therefore they are only interesting for players within a certain range. Furthermore, these messages are real-time, they are sent periodically with a high frequency. Loosing such a message has no severe consequences, but delivering it with a long delay can harm the player's game experience. Position and state updates are temporal, they cannot be queried or polled.

Temporal, Indispensable, AOI Data

There are certain events similar to position and state updates that are indispensable. For example if a player's unit dies and position updates for this unit are not sent any more, the information about the death of this unit must reach all players that could be interested in it. If it is lost other players would not remove the unit from their local game state.

Persistent, Indispensable Data

An example of persistent and indispensable data is the player's profile data. It holds information that describes the player and his current progress in the game (e.g. name, level, energy, power, owned energy sources). This data is persistent, its state must be consistent for all players and other users can query this data without the player being online or resending his data periodically. Furthermore, profile data must be changeable in a consistent way, the player can change his data at any time and this change must be reflected to all subsequent queries. The answers to queries and the update operations on this data must be also indispensable. For example an update of a player's energy resources cannot be lost. Player profile data does not use the AOI feature and is available independent from geographical position.

Persistent, Indispensable, AOI Data

World data that represents the towers or any other interactive object (e.g. energy sources, towers) must be also persistent and indispensable. In addition, such data is always bound to a geographical position and allows AOI ranging. World data can be changed, for example a tower can be build. World data queries or updates are indispensable. For example a query for all towers within hundred meters must always return all towers within this range.

Remaining Data Types

The remaining data types are not produced by the game. There is no temporal data that is sent to the whole network. Temporal data is always ranged within a certain AOI. There is also no persistent real-time data, because the chance of data loss allowed in real-time data leads to a contradiction with the consistency of persistent data.

4.2 System Design

The game consists of an Android client application, a Java p2p-network simulation server and a php/MySQL server for persistent data. To reflect data bound to an AOI with a temporal life cycle the event based p2p-paradigm of publish and subscribe (pub/sub) is a well-known solution. Despite the fact that the AOI related communication of the game is pub/sub based and could have been implemented with a p2p-network a server based simulating solution was chosen. It allows precise logging of all communication between the clients, which in turn allows to calculate statistics from the logged data.

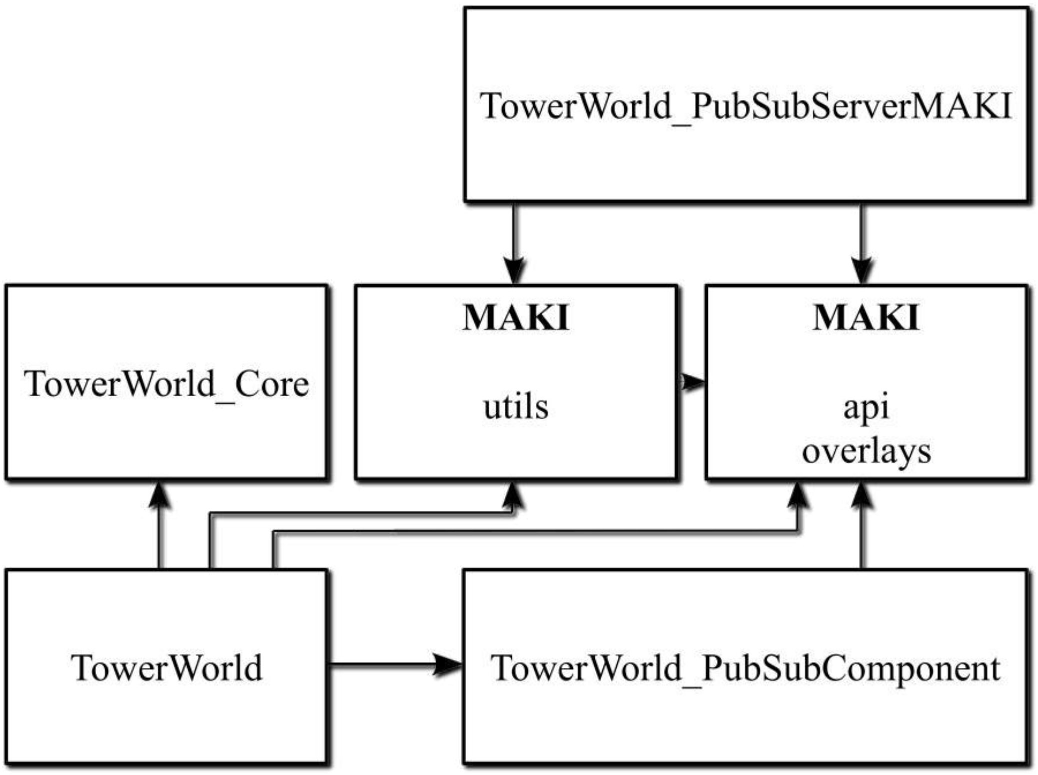


Figure 17: Project dependency graph.

This section starts with an overview over the first two systems. The php/MySQL server implementation will be explained later. The separation into four projects and eight packages is illustrated. Finally, the content of the packages and their purpose is explained in detail. However, not all classes will be explained below. Instead only those that might be an interesting starting point for any change to this work will be explained in detail. The code includes Javadoc comments for every class of the game.

To ensure that components can be simply reused, easily maintained and have no unnecessary dependencies to other components the Java part of the game is divided into four projects. In addition, this setup guarantees that no unnecessary dependency exists or can be made during future development. **Figure 17** illustrates the project dependencies including the MAKI projects, which provided the interfaces for the publish-subscribe functionality and a second pub/sub network simulation implementation.

TowerWorld_Core

This project implements the game logic of the client. It uses only the standard Java library and has no dependencies to other projects. Hence, TowerWorld_Core can be used on any platform that supports Java. The core implementation of the game can be run on a desktop PC with Linux, Windows or Mac OS for simulation purposes. The functionality of this project includes all logic that happens in the update loop, which is executed every frame such as movement of units, towers and weapons, energy sources, player positioning, unit spawning and tower building. Also all balancing values e.g. unit cost and HP or damage of weapons are included in this project. In summary, this projects provides model and controller functionality of the MVC pattern used in the client. Besides, TowerWorld_Core defines interfaces for visualisation, storage and communication of the game data. Instances of these interfaces must be passed during instantiation of the core. This way implementations of these interfaces can be replaced, which can be useful in a simulation scenario where one machine handles many game clients and handles all communication locally. In this scenario different implementations for storage and communication could be used. Also the visualisation could be removed or minimized to be only console output, since it is not required for simulation. However, for the core logic there would be no difference, since it does not need to know the exact implementation of the interfaces.

TowerWorld

The TowerWorld project in contrary to TowerWorld_Core references all other projects that are required in the client. First of all it references TowerWorld_Core. The core logic is instantiated and updated from this project. The TowerWorld project configures the combination of used components such as databases and networks. It passes these components to the core hidden behind general interfaces that are defined in the core. This way the core project does not know the exact implementation of the databases or networks and they can be replaced. Currently the game can run with one of two different implementations of the p2p-pub/sub-network simulation server. The one, that is included in the utils project provided by MAKI or the one, that was created during this thesis. In order to be able to connect to the MAKI server this project references the utils project and to use the other implementation TowerWorld_PubSubComponent is required. The api and overlays projects from MAKI are required for both communication implementations. Moreover, the TowerWorld project implements all Android specific features, such as activities, dialogs and localisation. For the visualisation this project uses the mapsforge library. Mapsforge is also used to generate points of interest e.g. energy sources and energy rechargers.

TowerWorld_PubSubComponent

TowerWorld_PubSubComponent is an independent centralized-publish-subscribe-network implementation based on the MAKI interfaces defined in the api and the overlays projects. It is not designed to be used for the game only and could be used for any other purpose. Conversely the game client does not need exactly this implementation and there is no game code included. This project includes p2p-server and p2p-node related code.

TowerWorld_PubSubServerMAKI

TowerWorld_PubSubServerMAKI puts together the components defined in the MAKI projects to run as a centralized-publish-subscribe-network server. This project includes only the server logic. Like TowerWorld_PubSubComponent this project has no game related code and could be used for any other purpose.

The contents of the projects above will now be explained in more detail. **Figure 18** gives an overview of the main packages included in the projects and their dependencies. Packages inside the MAKI projects will not be discussed in detail, since they were not made in this work.

game.mapsforge

This package provides the view component of the MVC pattern. To allow communication from core to view the event handling is implemented in this package. The game's visualisation is realized with the free map library mapsforge and the game.mapsforge package is the only one that references it. The mapsforge library is not only used to draw the map and the overlay with the game objects, but it also provides the data for points of interest. After reading the map file it is possible to search the data for tags like restaurants or historical places. This data is used to generate the energy sources and energy rechargers. In addition, the Android activity of the game is contained in this package. The reason for this is that it references the mapsforge library. In order to make game.mapsforge the only package that references the mapsforge library it was necessary to include the game activity in it, instead of the game.android package. This package is not reusable in another scenario it is designed to work only with mapsforge and the game.core classes.

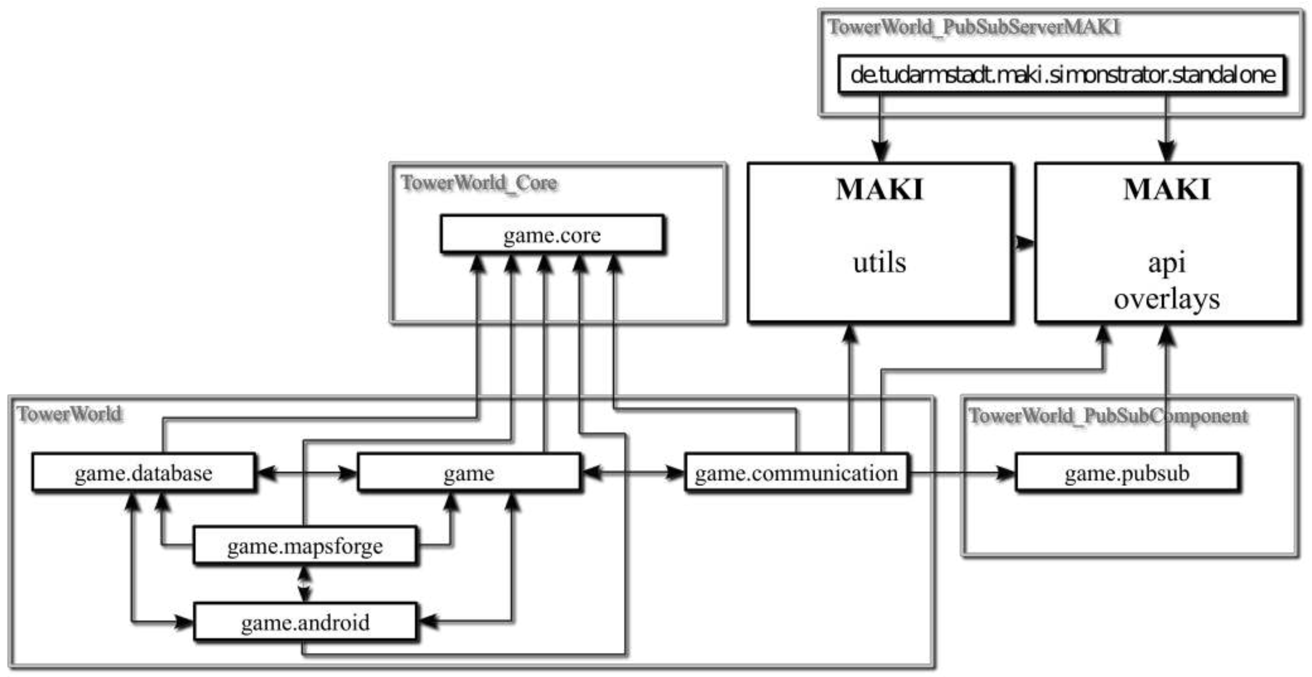


Figure 18: Package dependency graph also showing the projects.

game.android

All Android related classes e.g. activities and dialogs are contained in this package. This package draws the HUD showing the current energy level, XP, power, etc. using Android view interface. Also the local storage interface of the game is implemented in this package. All local data is stored in Android build in sql database. Another reusable component contained in game.android is the REST interface and its implementation in the HTTPDatabase class. It allows to make http get and post requests.

game.communication

The communication package contains only the AOI publish subscribe related mechanics. This package serves the above mentioned temporal, real-time, AOI data class. It provides two different implementations for the communication interface of the game. On the one hand it uses the implementation based on MAKI interfaces and classes, that can communicate with the centralized pub/sub-server from the TowerWorld_PubSubServerMAKI project and on the other hand it uses the classes from the TowerWorld_PubSubComponent project to use the other server. This package is the only package that references the MAKI utils and the TowerWorld_PubSubComponent projects.

game.database

The database package in contrast to game.communication contains only the database mechanics, which are currently implemented with the php/MySQL server. Persistent, indispensable is the data class served by this package. The database interface of the game is implemented in this package. The implementation requires an instance of the ISimpleREST interface, which can be replaced. Currently the above mentioned HTTPDatabase from the Android package is used. The implementation expects that the server on the other side of the ISimpleREST interface has certain php files and reacts in a certain way like it is implemented in the current php/MySQL server. This expected server behaviour is explained later.

game

The game package serves two purposes, on the one hand it declares global constants in the ConfigNet class to give different network implementations a common base instead of having each of them their own constants. Also the maintainability benefits from the ConfigNet class, since all network constants are located at the same place and can be easily found and changed. On the other hand the game package provides the system configuration by assigning the certain implementations to their respective interfaces in the ModuleConfigurator class. It allows to exchange components on a central location, which can be easily found. The ModuleConfigurator class hides the implementations from other game classes behind interfaces, this way components can be exchanged without any change in any other class than ModuleConfigurator. This way exchanging a network component is as easy as commenting one line in and one line out. The ModuleConfigurator provides commented sample code to use one of the two different network communication implementations. All replaceable components are listed later in this work.

game.core, game.pubsub, de.tudarmstadt.maki.simonstrator.standalone

The three projects of these three packages consist only of their dedicated package. Therefore, there is no need to describe these packages in more detail, since they are already well described in their respective project description. **Figure 18** shows to which project each of the packages belongs.

4.3 Platform and Library Choices

Nowadays almost everybody has a smart phone and many people that have one play games on it. The success of Google's game Ingress is showing us that people are indeed willing to play a game for hours and have no worries about their phone batteries getting empty. The technical advances of smart phones give us the unique chance to bring people together in the real world, few years ago nobody could imagine people walking outside with such computationally powerful and multi-sensory devices. The range of suitable mobile phone platforms is wide. However, in this work Android is used, because in the collaborative research centre MAKI at the TU-Darmstadt suitable software, such as peer-to-peer network implementation, was already in development for Android. Furthermore, the University provided free Android smart phones for this project. Nevertheless, it would also be feasible to use another platform such as iOS or Windows Phone. Mapsforge was chosen to be the open street map library of this work. Mapsforge in contrast to many other libraries allows to customize the map style by changing not only the colours, but also by defining which map information is drawn at all. The biggest competitor was surely Google Maps, however in Google Maps it is impossible to make such deep map style customizations. Mapsforge is a free library under Lesser General Public License version 3 (LGPL3)[11]. Besides, the data format, which was developed for mapsforge is very often used by other libraries on mobile devices, because it is designed for use on devices with limited resources. Mapsforge has no navigation features and is a straight forward short library, which fully satisfies the requirements of this thesis.

4.4 Android Activity Flow

Activities in Android can be compared to windows in desktop operation systems like Linux or Windows. Each activity represents a screen shown to the user. Only one activity can be in foreground at a certain time. The game consists of five Android activities. These activities and their transitions are visualized in **Figure 19**. The main activity of the game, which is started first by the OS is GameMainMenuActivity. However, the first activity that the player sees on the first start of the game is GameMapSelectionActivity. The user has to select his region and download a map file to play the game. The activity graph in **Figure 19** does not show the technical activity flow, but the one experienced by the user. The map selection activity is marked with a big arrow in the activity flow graph, because this is the first entry point of user interaction. There is no connection between the menu and the map selection, because it is impossible for the user to open the map selection from the menu. From the technical side the

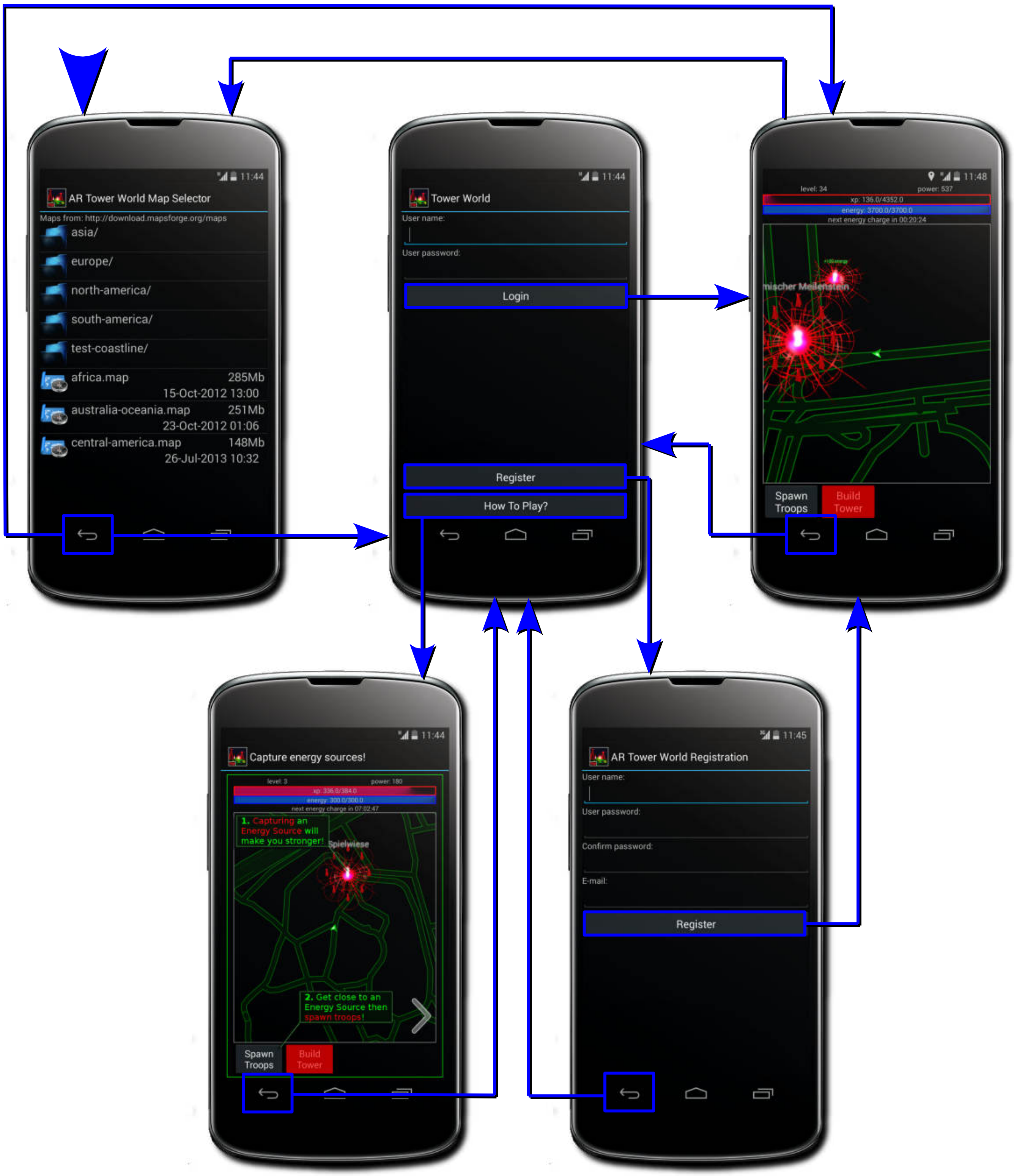

AR Tower World Map Selector
Maps from: http://download.mapsforge.org/maps
asia/
europe/
north-america/
south-america/
test-coastline/
africa.map 285Mb 15-Oct-2012 13:00
australia-oceania.map 251Mb 23-Oct-2012 01:06
central-america.map 148Mb 26-Jul-2013 10:32
Tower World
User name:
User password:
Login
Register
How To Play?
level: 34
power: 537
next energy charge in 00:20:24
Spawn Troops
Build Tower
Capture energy sources!
1. Capturing an Energy Source will make you stronger!
2. Get close to an Energy Source then spawn troops!
Spawn Troops
Build Tower
AR Tower World Registration
User name:
User password:
Confirm password:
E-mail:
Register

Figure 19: Activity flow graph showing the Android activities of the game: GameMapSelectionActivity(upper left), GameMainMenuActivity(middle), GameActivity(upper right), GameHowToActivity(bottom left), GameRegistrationActivity(bottom right).

menu activity starts the map activity and waits for a result code indicating that a map was selected if there is no map file selected yet. However, the GameMapSelectionActivity can also be opened from the GameActivity if the player enters an area outside of the currently active map file. In this case a dialog will be triggered and lead the player to the map selection. As soon as the GameMapSelectionActivity closes the previously opened activity will start again, therefore there are two activities that can be entered after touching the back button in the map selection.

On the first start it will be likely that the player will not take the direct way over the log in button to start the GameActivity, but will first need to register. Having started the GameRegistrationActivity by tapping the right button of the menu the player can start the GameActivity after filling the registration fields. Again technically the registration activity is closed in order to prevent it from unnecessarily running in background. It will return a result code parsed by the menu activity, if the player has registered then the menu activity will start the game. Finally, there is the GameHowToActivity which contains some images explaining the game. It can be started with the right button in the menu. When GameHowToActivity is closed it always returns to the menu.

4.5 Client

Most of the game logic is calculated by clients where each client tries to calculate the world and interactions with it only for itself and its troops. However, in some cases this is not trivial for example when two units from different clients are in the range of a tower. Which unit will be targeted by the tower? How is it secured that the tower is not calculated by both clients and shoots two different units. To solve this, the clients publish the tower shots, which they have calculated via the AOI communication. This way when a client starts to evaluate the reaction of a tower on a unit being in range, it is possible to check first if there is an active weapon already created by another client for this tower. This way clients dynamically decide which towers are calculated by which clients. Depending on the timing when the units of a client got into range of a tower, the first client that got into range will be the one to control the tower. Nevertheless, the database server also includes some logic. The reason for this and the server mechanics will be explained in detail in the next section. As mentioned above the publish-subscribe server has no game logic at all.

The implementation of the client uses the MVC pattern. **Figure 20** shows a MVC graph of the client that shows the responsible packages and their projects. As said before, the core project contains the controller and the model while the TowerWorld project contains the view. The communication from the controller to the view is event based. This way the view can be exchanged for example with a view that ignores almost all events and has no visualization in a simulation with many clients on one desktop PC. All events are contained in the game.core.event package. The view and the controller directly reference and access the model. The view instantiates and runs the main loop of the controller. The model is used only to store the game's state. It either knows the controller neither the view. The controller consists of two packages the game.core package explained above and the game.core.logic package, which contains all the game logic. The whole game model including all data types is contained in the game.core.model package. However, the data that is sent over the network contained in the game.core.network.remoteData is also part of the model. These four sub packages will be explained in detail later.

4.6 Servers

As already mentioned the game has two servers. A database server implemented in php working with a MySQL database and a publish-subscribe-peer-to-peer-network simulation server. For the later server two different implementations are provided in this work. The two different server types are required in order to cope with the network data generated by the game. One the one hand there is persistent data that cannot be stored in a pub/sub-network and requires a server with a database and on the other hand there is temporal data that does not need to be stored at all, but forwarded to all subscribers via a publish-subscribe-network. Obviously, it would have been possible to use one server that would

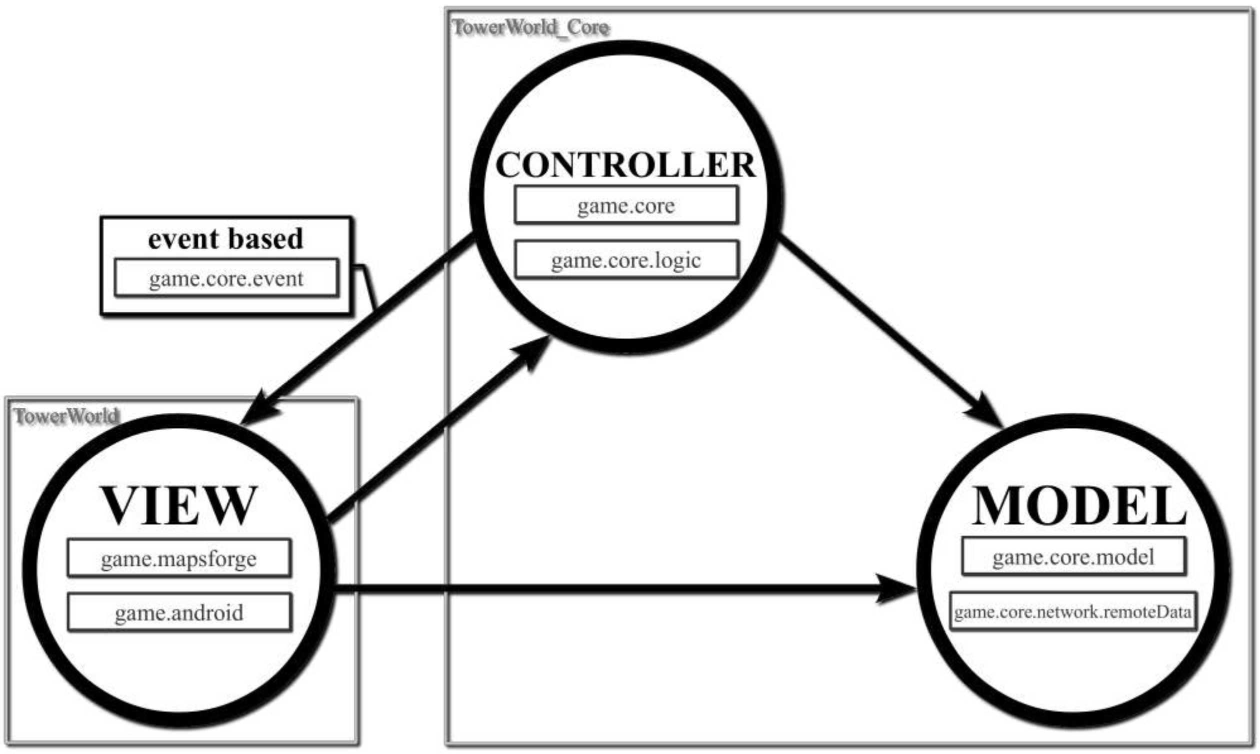


Figure 20: Model-View-Controller (MVC) pattern of the client including projects and packages.

selectively store or forward data depending on its type. However, this would be a very rigid solution, since the server would need to be very smart and hard to replace. In the current solution it is possible to replace one of the servers and keep the other one. The database server can be replaced by another implementation using another language or database type without changing the simulation server. In addition, it is possible to use different simulation server implementations and the step to use a real p2p-network would not require much change. As explained earlier the centralized implementation was required in order to log the network traffic. In a real p2p-network logging would be hard to implement. There would be no guarantee that messages would be delivered to all receivers, so that they could be logged.

4.6.1 Simulation Server

Both server implementations create logs and statistics on the network traffic of the game. The simulation server implemented in the TowerWorld_PubSubComponent project creates a log file containing each operation that the server has done. The log file is created in the same folder, where the jar file of the server is placed. Besides, the simulation server creates an error log file at the location where the jar was started from. There is a php script (server_control.php) included in this work that allows to remotely control the simulation server. It supports the following commands:
.../server_control.php?pw=[Key] will print the server's status. [Key] is a password that must be passed in order to prevent misuse.
.../server_control.php?pw=[Key]&cmd=start will start the server if it was not running before. The error log will be created at the location of the script (.../error_log.txt)
.../server_control.php?pw=[Key]&cmd=stop will stop the server if it is running.
The error log is easy to access and can be accessed by every person knowing the url. However, this is not desired for the normal log file, since personal information could be revealed. The players gave their consent only for scientific use of the data and therefore, it must be saved securely and hidden from the

public. Furthermore, the IP-addresses of the players are anonymized. Every new client gets a unique identification number (id). Every time the IP-address or port of the client is changed the client gets a new id. This can happen when the client's mobile device switches between networks e.g. from Wi-Fi to GPRS. The id has nothing to do with the IP-address and there is no way to decode the id back to the IP.

All possible lines of the server log are explained below. First of all, the server logs when it was started and at which port it can be reached. For example:

```
2014.01.09 13:01:47 — Centralized Pub Sub Game Server —
2014.01.09 13:01:47 started on port 14665
```

The next log line could be a client connecting to the server. The client id and the number of clients connected to the server will be printed:

```
2014.01.09 13:02:17 New client '24' connected clients '7'
```

After connecting to the server the client will usually subscribe for ranged updates in its area. The client id, the subscription and the number of subscriptions of this client including the current one will be printed. In the subscription the client's player name and the ranges for longitude and latitude are included. In the following log line the client '24' with the player name 'a' subscribes to ranged updates topic for longitude greater or equal than '8.679855', but less or equal than '8.789856' and latitude greater or equal than '50.131596' and less or equal than '50.241596':

```
2014.01.09 13:02:21 Subscribe '24' to 'topic - STRING - RangedUpdates filter: Filter: NEQ PlayerUID - STRING - a|GTE Lo - FLOAT - 8.679855|LTE Lo - FLOAT - 8.789856|GTE La - FLOAT - 50.131596|LTE La - FLOAT - 50.241596|' subscriptions count '1'
```

A client can also unsubscribe his prior subscription. This happens when the player has moved far enough and needs to update the coordinates of his AOI. In this case the client first unsubscribes from the old AOI and sends a new subscription afterwards. Besides, when the game is closed on the client then the client will also unsubscribe from ranged updates.

```
2014.01.09 13:04:34 Unsubscribe '24' from 'topic - STRING - RangedUpdates filter: Filter: NEQ PlayerUID - STRING - a|GTE Lo - FLOAT - 8.679855|LTE Lo - FLOAT - 8.789856|GTE La - FLOAT - 50.131596|LTE La - FLOAT - 50.241596|' subscriptions count '0'
```

The most valuable log entries describe the network traffic between the clients. The following few lines record the client '24' with the player name 'a' sending a ranged update message to '2' other clients, while '3' clients are connected to the server. The length of the message is 32 byte. The receiver names are 'wispli' and 'FreebordMAD' with the client ids '1' and '2'. The exact location of client 'a' is longitude '8.734843' and latitude '50.186604'. For the receivers the subscription, which includes a's location is printed. These three lines alone cannot be used to calculate the distance between those clients, since the subscriptions tell only a range instead of a certain location. To do it the receiver's location must be read and buffered from the latest message sent by one of the receivers. In most cases such a buffered location will not be older than two seconds, which is accurate enough concerning that players move with usual walk speed.

```
2014.01.09 13:02:25 Forwarding notification from ’24’ to ’3’ clients ’topic - STRING - RangedUpdates attr:PlayerUID - STRING - a|Lo - FLOAT - 8.734843|La - FLOAT - 50.186604| payload length 32’
2014.01.09 13:02:25 Forward notification to ’1’ with subscription ’topic - STRING - RangedUpdates filter: Filter: NEQ PlayerUID - STRING - wispli|GTE Lo - FLOAT - 8.679817|LTE Lo - FLOAT - 8.789818|GTE La - FLOAT - 50.131664|LTE La - FLOAT - 50.241665|’
2014.01.09 13:02:25 Forward notification to ’2’ with subscription ’topic - STRING - RangedUpdates filter: Filter: NEQ PlayerUID - STRING - FreebordMAD|GTE Lo - FLOAT - 8.679827|LTE Lo - FLOAT - 8.789827|GTE La - FLOAT - 50.131687|LTE La - FLOAT - 50.241688|’
2014.01.09 13:02:25 Forwarded notification to ’2’ receivers ’topic - STRING - RangedUpdates attr:PlayerUID - STRING - a|Lo - FLOAT - 8.734843|La - FLOAT - 50.186604| payload length 32’
```

One of the last lines of a usual log file will tell that a client is disconnected. The following line shows that the client ’24’ has disconnected and ’6’ clients are still connected to the server.

```
2014.01.09 13:04:34 Disconnected client ’24’ connected clients ’6’
```

The log file will also contain errors. For example if a client walks away from a Wi-Fi hotspot and starts to use an EDGE connection instead, the last connection which the server had to the client over the Wi-Fi hotspot will time out after a while. The server will print a Java socket exception and its stack trace before removing this client. Such an exception will be preceded by ’ERROR: ’ and the line before the exception line will in most cases tell if the exception occurred while receiving or while sending a message. The usual prior explained client disconnected log line will also be printed.

The simulation server is capable of recording messages generated by the game. This way, it is possible to record a whole game session and play it back at any time. The recorded data can also be modified for example in order to simulate the impact of different technologies on latency. The record and playback functions can be controlled via the command line if the IS_CMD_CONTROLLED constant in the StartServer class is set to true. If the server needs to record all messages the IS_RECORDING constant in the same class needs to be set to true. The recorded data is stored in a csv file at the location of the jar file. The columns of the csv file are: time, client id, player uid, subscription topic, longitude, latitude and payload encoded as a base 64 string.

4.6.2 Database Server

The database server is based on MySQL. It consists of six data and four relation tables. **Figure 21** gives an overview of the database design. All tables have the ’tw_’ prefix and the relational tables start with ’tw_rel_’. The tw_accounts table stores the account data of each player, which includes the login data, the current game state and data needed for energy generation. Tables tw_tiles and tw_groups are used to generate ids and to guarantee that ids stay unique. However, they can be used in future to give groups and tiles further attributes. The energy sources are saved in tw_targets. In addition to game data saved for a target also energy generation time and capture time are stored. In tw_towers the data of each tower is saved and a foreign key to the parent energy source set. Finally, the tw_statistics table is used to record every user action storing the action’s type, time and data for every account.

Before the database server statistics can be discussed the server implementation must be explained. The php scripts are based on meekrodb[38]. Meekrodb is an open source MySQL wrapper library under LGPL3[38]. The database server consists of some php files that are expected by the client. These expectations arise in the game.database.RemoteDatabase class, which can be replaced with another implementation of the games remote database interface. For example it could be required to use another implementation of the database in a simulation where php and MySQL are not desired. The only two tasks of the server, which cannot be easily handled by the clients, are energy generation and group han-

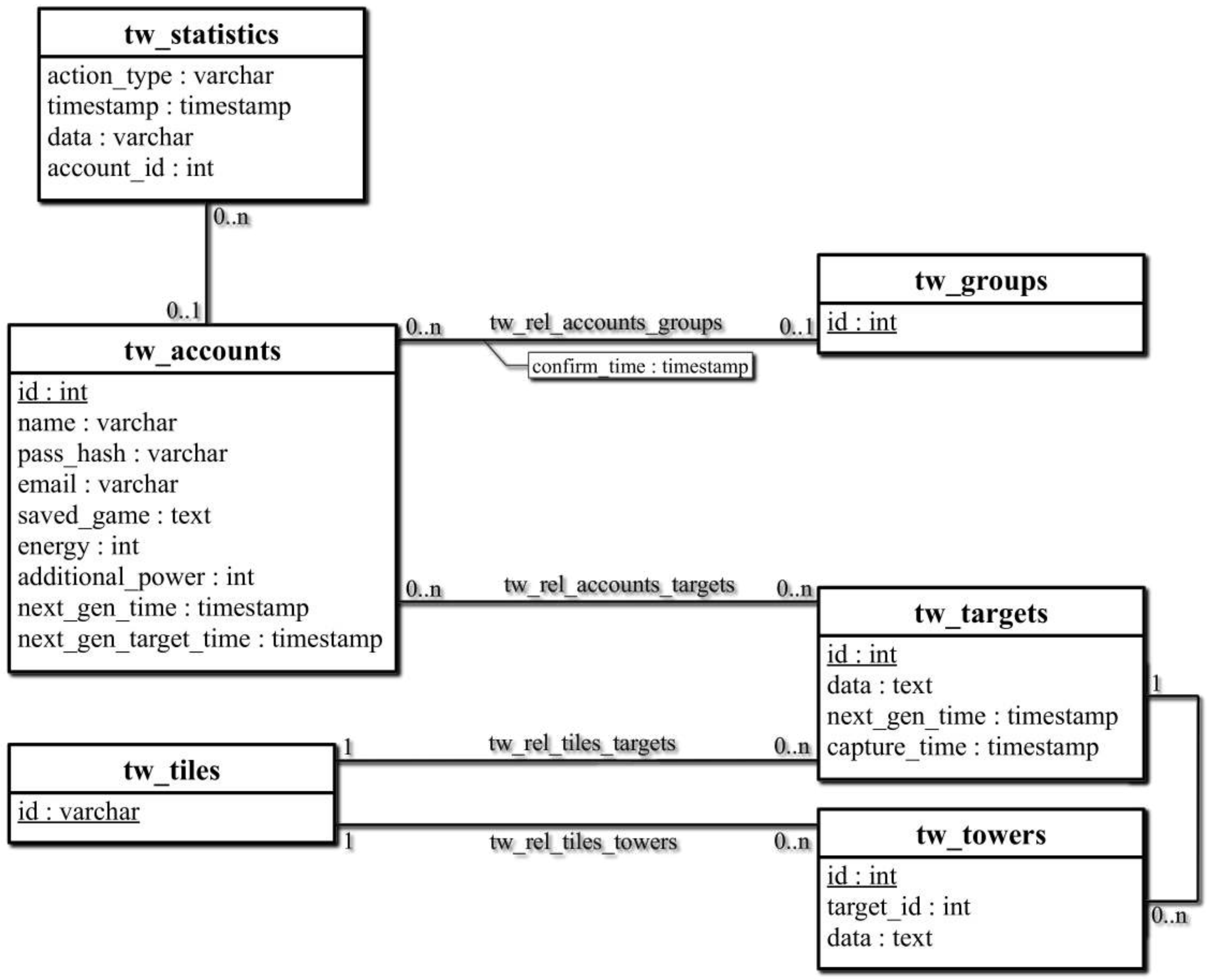


Figure 21: MySQL database table overview.

dling. Since most of the logic is handled on the clients the scripts are very short and with the included comments they are self-explaining. They will be listed and briefly explained below. In the following, also the tables that are managed by the each script are discussed. All scripts that require a player name and a password call authentication functions that read the tw_accounts table. Scripts that need to get the group state of the player call functions that can insert entries into tw_groups and read tw_rel_accounts_groups table.

account_[create,save,load].php

There are three scripts to manage accounts they change and read only the tw_accounts table. All three scripts require the player's name and the hash of the password to be passed as parameters. To create an account additionally an email address must be provided. Loading and saving of the account means passing a certain string on save and getting exactly the same string back on load. The server does not know what this string means or contains. The only account data that the server understands is energy and power. The result of the create and save scripts is the 'ok' code on success. The load account script returns the 'ok' code followed by the prior saved game string, the current energy amount, the additional power generated by the owned towers, current server time and the soonest energy generation time. The server time is needed to synchronize the player's time with the server time, since the player could be in another time zone or simply have the wrong time set in his device. The soonest energy generation time, which can be the default periodical energy generation or any target energy generation, is needed in order to allow the client to ask the server to calculate the energy generation. Only the server can do this, because the client does not know all owned energy sources. Even if the client would know all energy sources he would need to know all owners and be able to change their energy balance, which is technically inefficient, since the amount of data that would need to be transferred from the server to the client would be very high.

change_energy.php

Alters tw_accounts only. To change the energy, name and password hash must be passed. In addition, the relative value of the energy change must be passed. It can be negative if the player has spawned troops or positive if the player has reached an active energy recharger. The server trusts the clients and simply adds the given energy change to the players energy balance. The absolute amount of energy is never sent from client to server, because the energy balance could have already changed on the server. For example if a client logs in and some other client captures his energy source, it can happen that the first client gets some last energy from its source. Now the energy amount is desynchronised between the client that has the energy amount that was right during login and the server that now has the updated energy amount. Besides, the order of requests is not guaranteed it could happen that the first energy change comes last and the second first. If the energy changes would be passed by absolute value the result would be different depending on the order. Hence, it is better to send only the energy changes not the absolute value. The change_energy script returns the 'ok' code on success.

generate_energy.php

This script has to access the most tables of the server and this is the reason why it has to be done on the server side, otherwise the clients would need to pull huge information amounts. All target related tables are read, tw_accounts and tw_targets are altered. Energy is generated periodically and in addition every owned energy source generates energy following its own timer. The server knows the generation amounts and the timings. The client has received its soonest energy generation time during login when account_load.php was called. The client has to call this script when the given generation time is reached. After authentication with player name and password hash the default energy generation will be handled. If energy was generated the energy balance is increased and the next default energy generation time is calculated. The server knows only the energy of the client, but not the client's energy capacity. The generated energy is simply added to player's balance, if the balance is higher than the energy capacity the client has to call the change energy script and reduce it to the right value. The complicated part of the script is the target energy generation. Since targets can have multiple owners the energy must be generated for all players. The script iterates over all owned energy sources and checks if any of them have generated energy. The next energy generation timestamps are updated if energy was generated. The generated energy is added to every owner account. While iterating over all targets the next target energy generation time for this player was calculated and is now updated in the database. The result of the generate_energy script is as usual the 'ok' followed by the generated energy amount, the next energy generation time and the tile ids of energy sources that have generated energy. The tile ids are required for the player to know which targets need to be updated, since the game must visualize the updated energy generation countdown for each target.

capture_target.php

Similar to the generate_energy script energy generation must be calculated, since it can be triggered when a target gets captured. All tables that are used in the generation script are accessed, additionally the group tables are read and can be altered. To capture a target (energy source) the client has to pass the player name, password hash and the target id to this script. The client must have a valid group id. Technically it is not the client that captures the energy source, but its group. To guarantee that a client has a valid group id the ping script must have been called at least once before a target can be captured, which should be the usual case. Another failure reason is the safe mode of targets, which means that another player has captured the target shortly before. Additionally, an owned target cannot be captured again. If the target can be successfully captured the old owner accounts must be updated. First of all, energy generation must be handled, because if the old owners are offline they could not have called the generate_energy script themselves. After that, the additional power of the old owners, which was provided by the energy source, must be removed. Finally, the ownership of the prior owners can be removed. Now the additional power generated by the source and the instant energy recharge can be applied to all members of the capturing group. Same is true for the ownership, since every member of the group now also owns this target. After updating the energy generation and the safe mode timing of the target this script returns the 'ok' code.

build_tower.php

This script reads and inserts values into tw_rel_tiles_targets and tw_towers. To build a tower the player's name and password, the target's id and the new tower id must be passed. Additionally, tower data similar to the saved game string must be passed. The server does not know what tower data is and simply stores the string. If the target id is found in the database and the tower id does not already exist the tower can be stored. Since the ids of targets and towers depend on their coordinates it is impossible to build the same tower twice. This script returns 'ok' if the tower is built successfully.

group_join.php

Here tw_rel_accounts_groups is managed. In order to join a group one of the clients have to call the group_join script. In the current implementation if two clients want to be in the same group, the second client in a lexical comparison of player names will call this script. The name and password of the calling client (first client) must be passed. The name of the other client (second client) must also be given to this script. Both players must have a valid group id, which basically means that both have called the ping script within a certain interval before this script was executed. Now all players with the same group id as the id of the second player will get the group id of the first player. This way a single player can join another single player and build a group of two. A single player will also join a group of players. Two groups will be merged into one group. 'ok' is returned if a group is joined.

group_load.php

This script will return ’ok’ followed by all group member names if the passed player name and password is valid and the player has a valid group id.

ping.php

The ping script implements the timeout functionality for groups. If a player does not call ping for a certain time period he is removed from the group. To execute a ping the client must pass the players name and password. If the player had no group id assigned he will get a new one. Otherwise the last ping time of this player is updated in the database. It is stored in the tw_rel_accounts_groups relation table see **Figure 21**. If the difference of this last ping time and the current time is bigger than a certain time interval and some script tries to access the group id of the player, then the player will be removed from the group. Also this script returns ’ok’ on success.

initialize_tile.php

This script alters almost all tile, target and tower related tables. Since the server has no logic except for energy generation and group handling it is impossible to know the initial targets and towers for it. Hence, the clients have to provide the initial energy sources and their protecting towers by calling this script. The world is divided into non overlapping tiles, however the server does not know how the tile id, the target id or the tower id is calculated. The server simply stores what the clients upload to it. If the tile was not initialized already the ’ok’ is returned.

tile_load.php

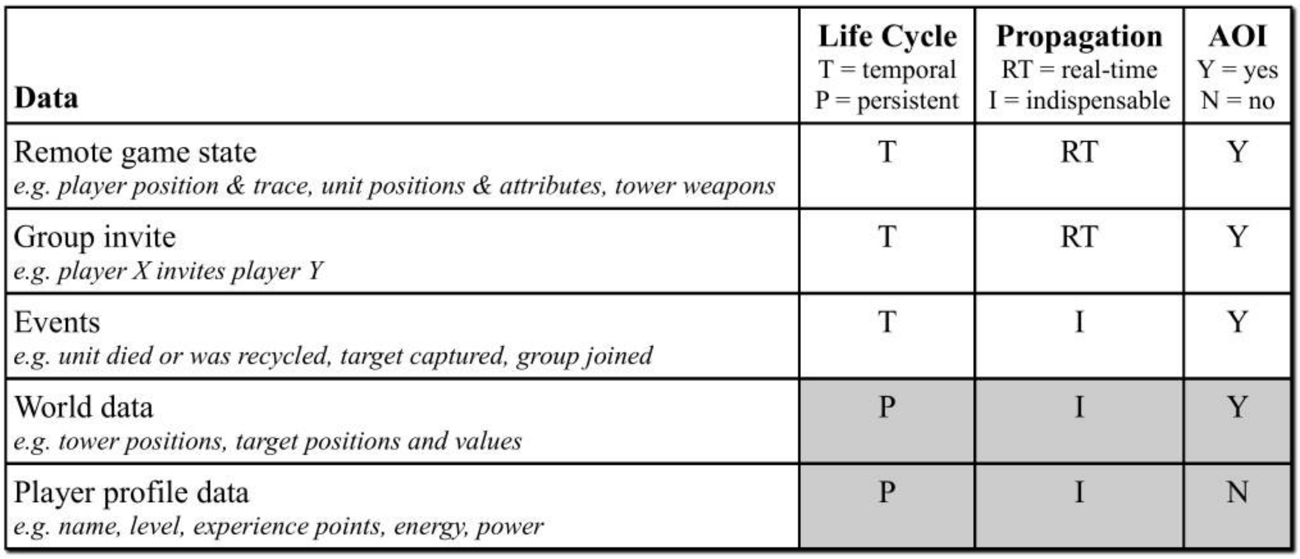
This script simply reads the tile, target and tower tables and prints the contents of a tile in a certain format that can be read by the client. There is no authentication needed for this script, only the tile id has to be passed.

config.php

This script is designed to be included and used by other scripts. It stores general functions that are used in all scripts, such as authentication with a player name and a password. Methods to fetch the player’s group or assign a new one if needed are also included. Table names and balancing values such us energy generation amounts and timings are also included in the config script.

All of the scripts above except config.php insert an entry into the statistics table for each successful action. Possible action types can be TILE_LOAD, INIT_TILE, LOAD_GROUP, GROUP_CONFIRM, JOIN_GROUP, GENERATE_ENERGY, CHANGE_ENERGY, CAPTURE_TARGET, BUILD_TOWER, ACCOUNT_SAVE, ACCOUNT_LOAD, ACCOUNT_CREATE. The time of the entry is always the server time of the event. Except for TILE_LOAD, which does not require the user to login, the account id of the player is stored for every action. However, in this work this data is not evaluated. Instead only the log of the pub/sub-server is investigated in more detail.

In order to make cheating more difficult the server-client communication is protected with a token mechanism. Every message sent from the server or from the client includes a token. This token is the SHA-256 hash of all passed parameters concatenated with a secret value. This secret value is hard coded on server and client. This way the post parameters of an HTTP request cannot be simply modified. However, obviously this system is not secure, since the secret can be found when the game’s code is decompiled. Nevertheless, it needs time and incentive to decompile code and most people would not take the trouble just to cheat in the game. In case of cheating this security system can simply be extended, for example the secret could be passed during login. In this case the secret value would exist only in memory and would be harder to find.



Data	**Life Cycle**
	T = temporal
	P = persistent
		Propagation
		RT = real-time
		I = indispensable
			AOI
			Y = yes
			N = no

Remote game state *e.g. player position & trace, unit positions & attributes, tower weapons*	T	RT	Y
Group invite *e.g. player X invites player Y*	T	RT	Y
Events *e.g. unit died or was recycled, target captured, group joined*	T	I	Y
World data *e.g. tower positions, target positions and values*	P	I	Y
Player profile data *e.g. name, level, experience points, energy, power*	P	I	N

Figure 22: Table showing data types and their properties. They grey coloured types are served by the database server, while the other data types are distributed over pub/sub.

Another mechanism used by the server that needs to be explained is versioning of scripts. If the scripts or the database are fundamentally changed and the old client version must be updated it is possible to create a new folder named for example 'v5' and put the newest version of the scripts inside. The old version in the old folder needs to contain only the account_load script, which should be replaced by a script that always throws an error saying that the client needs to be updated. This message will be displayed to the client when he tries to login with an outdated client version. The new client version must change the server address to include the 'v5' version. In the current client implementation the HTTP_DATABASE_SERVER_ROOT constant in the ConfigNet would need to be adjusted.

4.7 Data Type Realisations

Figure 22 shows all data types of the game and their properties. The grey coloured fields in the table show data exchanged with the database server all the other data is handled by the pub/sub-network. In the current implementation the pub/sub-network is simulated with a server, however later in the scope of MAKI the simulation server can be replaced with a real p2p-network.

Remote game state data fully describes the game state of one client. It contains the location and the trace of the player. Also all units their location, direction, speed, size and type are included. The current active weapons of towers are also included. If a client connects to the pub/sub-network and receives some remote game states of the surrounding players he can fully initialize, draw and handle the remote players. There is no need to announce each other. The other players will have all the data needed as soon as they receive the first remote game state of the new player. However, the remote game state contains more data and not only the current game state. Usually, a pub/sub-network is realized with a p2p-network. In such a network messages can be lost or delayed, which makes it complicated to realize events with real-time requirements. The table shows a data type called events, however there is no explicit data type representing any event, instead events are realized in an implicit manner. For example the event of a unit X being destroyed is not send as a message saying unit X is destroyed, since the message could be lost or one of the clients could be in standby and simply miss this event. Instead the unit is simply removed from the remote game state. If a client receives a remote game state from the player that owned unit X and unit X is not included in the list any more, then the client considers the unit to be dead and plays the explode animation. This works even if the client was in standby for long time and the unit X is already dead for a while or if some remote game state updates from the player of unit

X are lost. Furthermore, in some cases the death animation should not be played for example if a unit was recycled. To handle this, the remote game state contains a list of unit ids that were recycled. This list is announced during a long time interval in many successive remote game state messages. If a unit disappears from the list, but its id is included in the recycled list then no animation needs to be played and the unit's icon simply disappears. In addition, the remote game state includes a time stamp and a tile id of the last modified tile. A tile can be modified if one of its targets is captured or a tower is built on this tile. This modification event should force all other players to synchronize their game state and reload this tile from the database. Including the last modified tile id and the modification time stamp allows all players even if they miss many messages to check if their data is up to date or needs to be reloaded. Having said that, it is obvious that the group invite data is not send once like an event, but repeated until a time out is reached or the invited player has joined the group. While broadcasting the invite the client periodically checks the group members in the database server. This way the client knows when the invited player has accepted the invitation. All this explains that there is no explicit data of the **Temporal, Indispensable, AOI Data** data class, but all such information is implicitly realized in other messages.

The data types used to communicate with the database server are simple and straight forward. All such data is persistent and indispensable the only difference in the data properties is that there is AOI data and global data. On the one hand the world data is always needed only for the surrounding area of the player and on the other hand the player's profile data is not bound to a location and is requested in the same way from different places. However, there is no difference on the server for AOI data and global data, the only AOI processing made on the server is the optimized SQL query that fetches only the tiles from the database, which the player needs to know.

4.8 Replaceable Components

Replaceable components allow to change the implementation, but keep the functionality. Simulation on one machine or platform portability are examples where components must be replaced, but functionality kept. In a simulation rendering is not required and respective components can be replaced to generate console output only. Furthermore, network components would need to be replaced in order to simulate them locally. In order to make the game run on a desktop PC all Android related components must be replaced. Having read the System Design section, it is obvious that some packages and all their components can be fully replaced such as mapsforge, communication, database or pubsub packages. Nevertheless, the game itself does not change when replaceable components are exchanged.

game.core.network.IRemoteCommunication

This interface defines the pub/sub-based communication mechanics of the game. It does not actually allow to subscribe or publish to a certain topic, however it is designed to be used in a network with published and received messages that can be lost, reordered or delayed. The underlying implementation can use the given player uid, location and range to find the target receiver group and find the senders, which the game is interested in. This interface handles the temporal area of interest related data of the game. The ModuleConfigurator class instantiates the used implementation of IRemoteCommunication, which is passed to the constructor of the Game class, the controller of the MVC pattern.

The implementation of this interface is provided in the abstract class CommunicationBase contained in the game.communication package. This class is based on MAKI interfaces and allows connecting to a pub/sub network via the MAKI PubSubComponent interface that must be created in the constructor of the derived class. The whole communication happens in one topic its name is defined in the TOPIC_RANGED_UPDATES constant. CommunicationBase subscribes to the topic of the game. The subscription is limited via attributes to the area which the user is interested in. Besides, the subscription filters messages with the player's name, which makes sure that sent messages do not come back to the sender. Every time the player changes its AOI the old subscription is unsubscribed and a new subscription is generated. This implementation serializes and deserializes instances of RemoteDataGameState class defined in game.core.network.remoteData package. The group invitation is realized through the message attribute mechanics of the MAKI interface. All used attribute names can be found in constants with the ATTRIBUTE_NAME_ prefix. This implementation does not need to be replaced if a real p2p-network instead of a simulation server is used. To use a real p2p-network implementation a class must derive from CommunicationBase and initialize the desired PubSubComponent interface implementation in the constructor. The last line of the constructor should call the create method of the CommunicationBase class. This class only needs to be replaced if the use of MAKI interfaces is not desired.

MAKICommunication and ServerBasedCommunication derive from the implementing class. MAKICommunication uses the centralized server implementation from the MAKI utils project. ServerBasedCommunication uses the pub/sub server implementation written in this thesis. These are the replaceable components of IRemoteCommunication. These classes are meant to be substituted by a real adaptable p2p-implementation in the scope of MAKI in a future work.

game.core.network.IRemoteDatabase

The IRemoteDatabase interface defines the database server communication of the game. Persistent and indispensable data types are implemented with this interface. No requirements are declared concerning the communication technology. The database server does not need to be php and MySQL based like it is implemented in this thesis. This interface could also be realized with a distributed p2p-database. IRemoteDatabase is rather based on basic game functions like login in, load group members, capture target, get tile data, build tower, etc. than simple database operations. Hence, this interface expects the database server to be able to handle some logic, especially the energy generation and the group mechanics, because as explained in the servers section it is impractical to implement those on the client. If it is desired to replace the php/MySQL server implementation provided with this thesis this class needs to be replaced. In order to keep all functionality of the game the new implementation must behave in exact the same way the server does now. The implementation of this interface must be set in the ModuleConfigurator class and will be passed to the Game class constructor in the GameMainOverlay class.

RemoteDatabase is the current implementation of this interface. The RemoteDatabase class requires an ISimpleREST interface implementation which it gets from the ModuleConfigurator. The reason for this additional interface is to allow the RemoteDatabase class to be used on another platform than Android by replacing the implementation of the ISimpleREST interface. ISimpleREST is explained in the next paragraph. RemoteDatabase requires php scripts of the current database server implementation. They are enumerated and explained in the database server section.

game.android.http.ISimpleREST

The ISimpleREST interface is as simple as its name says. It consists only of two methods get and post. Get fetches a resource only by its URI, which must include all parameters. Post additionally allows to add a list of name value pairs to the request. HTTPDatabase implements this interface and allows to make simple HTTP get and post requests. Besides, the above explained token cheat preventing mechanics are implemented in the post method of the HTTPDatabase. All values of the passed post parameters are concatenated. The secret is appended and the result is hashed with SHA-256. The hash is the token and is always appended as the last parameter to the post header. Having said that, it is obvious that the order of parameters passed to the post method must be the same as expected in the php scripts, otherwise the token would differ from the token calculated on the server. Similar to the first two mentioned interfaces ISimpleREST must be assigned in the ModuleConfigurator. The HTTPDatabase can be replaced by another implementation. For example if Android is not the desired platform it must be replaced. Another scenario is the use of different servers for different php scripts, for example a separate login server and many different servers for different countries in the world.

game.core.ILocalDatabase

The energy rechargers are generated separately for each player. This allows to keep them fully local and make the clients handle them on their own. However, the data needs to be stored locally on the client. ILocalDatabase defines two functions to store and read the energy recharger timings. The GameMainOverlay class implements this interface and passes a reference of its instance to the constructor of the Game class. It uses the SQLdb class, which stores the data in the Android sql database, available for every Android app. The implementation can be replaced for simulation purposes. For example to allow different clients simulated on the same not Android based machine to use different local databases.

game.core.ITileImplementationAdapter

This interface hides the map implementation. The game expects the world to be divided into tiles and this interface defines four simple methods to interact with tiles without knowledge of their size and the underlying map implementation. This way the currently used mapsforge library could be replaced by Google Maps. The first two functions allow to get a tile id for certain coordinates and to get all tile ids that are in the player's vision range. Also the initial energy rechargers, targets and towers are expected to be generated in the implementation of this interface. The current implementation can be found in the MapsforgeTileAdapter class in the game.mapsforge package. It is instantiated in the MainGameOverlay class and passed to the Game class constructor.

game.core.event.IGameEventHandler

The event handling between the controller and the view in the MVC pattern is declared in this interface. Some events announce game objects being created or destroyed. This allows to handle their visualisation such as showing and hiding icons. Other events are used to show certain information to the user such as having captured an energy source or a network error being caught. The only event type that requires user interaction is asking the player if he wants to group up with another player that comes into group range. The implementation of this interface can be replaced for simulation purposes e.g. console simulation without visualisation or if visualisation is needed on another platform than Android.

The current implementing class is GameMainOverlay. It asynchronously creates and hides views. It shows information to the player in dialogs with a single 'ok' button. The group up event is handled also by a dialog that allows to accept or decline the group up. In addition, the group up dialog provides some information on the group mechanics of the game.

game.core.ITime

Finally the last replaceable interface is ITime. It defines the local timings that are used for unit movement and tower weapons. Also the speed of most animations depends on this time. Furthermore, most periodic update times such as assist time limit or remote player disconnect time out use this timing. The used implementation is assigned in the ModuleConfigurator. The GameMainOverlay passes the selected implementation to the static Time class, which can be accessed everywhere in the game. This interface's implementation can be replaced for simulation purposes for example to synchronize all simulated clients on certain timing. The current implementation of this interface is the TimeDefault class. It supports a time scale that can be applied. The time scale is defined in the constant TIME_SCALE of the ConfigGame class. It is very useful in debug situations and demonstrations.

4.9 Components Open For Future Change

In order to continue the development of the game, components must be changed and functionality replaced or evolved. Such changes can be e.g. changing server addresses, changing balancing values (energy generation timings, HPs, weapon damage), adding new unit or tower types and creating their visualisation. In the following enumerated components expect to be changed. They are designed in a way that allows changes to happen without harming the systems and without creating unexpected outcomes. For each component also an example for a supported change is provided.

4.9.1 Config Files

The game contains many values that need to be tested and balanced and accessed from various components. These values might change during further development or new constants may need to be declared. It starts with basic game values like the amount of health points of a unit. Besides, also network values must be adjusted for example the frequency of periodic game state updates needs to be as low as possible to save bandwidth, but high enough to prevent artefacts like jumping units. Finally, global known constants like a port number of the server or result codes of activities must be known at different places. Storing these values at the place where they are used involves different risks. On the one hand maintainability of such values is getting worse and worse with the number of values and the number of code pieces that are spread over the whole project. Finding a specific value takes more and more time. By changing one value the balance with another value that is written in another class could be broken e.g. when changing the damage of a tower weapon the health points of different units might need to be adjusted. On the other hand values that are used by different components would exist multiple times in the code. If for example the server implementation would change its well-known port number all code pieces in the client using this port number would need to be found and changed otherwise the client would not work anymore. To improve maintainability and prevent problems arising through change five configuration classes are provided.

game.ConfigGame

The biggest and the most important configuration class is game.ConfigGame. It stores all basic game constants for units, towers, targets, energy rechargers, group mechanics, network periods and time outs, message texts and role play game values. The only class that contains game balancing values except ConfigGame is the TroopsSpawner class that defines the attributes of different units. The TroopsSpawner class will be explained in more detail later.

Finding the right balance of the values in this file took at least 15% of the time of this thesis. They were adjusted in various game sessions. One of the biggest challenges of this thesis was to find the right distance for the tower ranges. Every time the range was changed, many things had to be adjusted such as other constants like unit speed and size, but also the visualization had to be adapted. Having started with a value around 50 meters the tower range was reduced further and further with every game session, because the participants felt that it was too much running for too little impact in the game. However, on some point the tower range was reduced below 10 meters and many problems arose from the inaccuracy of the GPS localisation. After long testing and balancing the value of 12.5 meters seemed to perform best.

game.ConfigNet

The game.ConfigNet configuration file defines all required server addresses, ports and also encryption values like the prior mentioned secret for token calculation used in the database server communication. However, one of the constants in this class is also defined in another class. The simulation server port is defined once in ConfigNet and once in ConfigServer. It seemed to be the right way to prevent the server project from referencing the client project only to get the right port number.

game.mapsforge.ConfigGUI

ConfigGUI contained in the game.mapsforge package is responsible for visualisation constants related to mapsforge. For example the maximal and the start map zoom levels are calculated dependent on the resolution of the used device. The zoom levels at which certain parts of the GUI are hidden such as names of energy sources and their towers are also defined in ConfigGUI. Also early clipping constants are included. Early clipping improves performance by hiding objects when their centre is outside the screen and the nearest screen border is further away than the mentioned constants.

game.android.ConfigAndroid

ConfigAndroid in the game.android package defines files and folders used to store map files and the map skin on the Android device's SD-card. In addition, it contains result codes used by different activities for example RESULT_CODE_MAP_SELECTED is returned by GameMapSelectionActivity when the user has successfully downloaded and selected a map file and RESULT_CODE_MAP_NOT_SELECTED is returned when the map selection activity was closed, but no map was selected. The GameActivity class parses these result codes and needs to know their exact value.

de.tudarmstadt.maki.simonstrator.standalone.ConfigServer

As mentioned above the server port is defined in ConfigServer and in ConfigNet. Besides, the MAKI simulation server implementation needs to know the display name of the network interface, which should be used for communication. This name is also defined in ConfigServer.

Handy Debug Constants

There are few very important constants that make the debugging of the game easier. For example the TIME_SCALE in ConfigGame combined with IS_LOCATION_CONTROLLED in ConfigGUI allows to quickly test game features with fast moving units and a faked location that can be changed by swiping over the screen. Furthermore, IS_DEBUG_VIEW_TILE and is IS_DEBUG_POI contained in ConfigGUI allow to see tile borders and open street map tags contained in the map file. Especially IS_DEBUG_POI is handy if more POIs need to be generated and further map tags must be added to the POI selection set.

4.9.2 Game Logic (MVC-Controller)

The Game class in the game.core package is the heart of the controller component in terms of the MVC pattern. This class itself contains almost no logic code at all. It rather holds and organizes classes that implement logic. However, this class is the main entry point of the game.core package. The update function of the Game class is the main loop of the game and is executed on every frame. This function propagates the update event in a certain order to all the instances of classes that implement parts of the logic. These classes are contained in the game.core.logic package. **Figure 23** gives an overview of the logic classes and their dependencies in a class diagram. This class diagram should explain the basic structure of the logic related code, therefore certain details are skipped in order to keep the figure comprehensive. All classes have private members only, the private methods are hidden, hence members and functions in the diagram are all public and their visibility is not explicitly shown. Complex collection data types like hash maps and lists are simplified and the parameter lists of functions are hidden. Since multiplicities of all associations are 1 they are also omitted. Below all classes are enumerated and shortly explained.

LogicAsyncOp

The logic classes have references among themselves. For example the LogicAsyncOp class is referenced by the most other logic classes. This class is responsible for asynchronous processing of database operations. It stores the IAsyncOperation instances returned in methods of IRemoteDatabase. As soon as such an operation is executed LogicAsyncOp calls a given callback function, that needs to be executed when the communication to the database server is done. This functionality is required in most parts of the logic and can also be used outside the core project. For example when troops are spawned the energy needs to be reduced and only if the energy could have been reduced successfully on the server the troops can be spawned on the client.

LogicPlayer

This class is also referenced by others. It handles the player related functionality, such as a level up after adding experience points, which can happen in different parts of the logic. For example XP can be awarded for troop spawning, target capturing or having assisted in a target capture. All these XP gen-

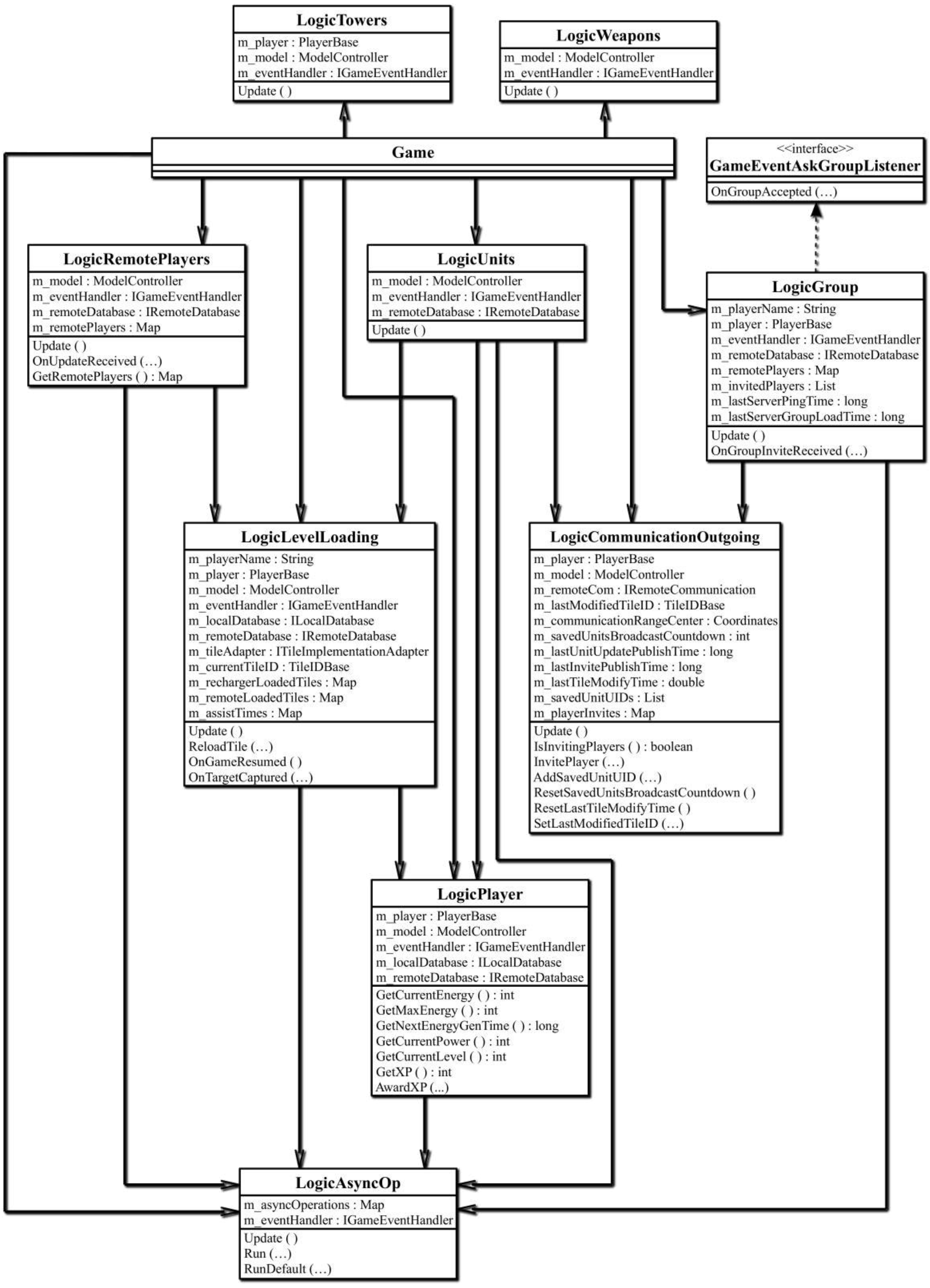

LogicTowers
m_player : PlayerBase
m_model : ModelController
m_eventHandler : IGameEventHandler
Update ( )
LogicWeapons
m_model : ModelController
m_eventHandler : IGameEventHandler
Update ( )
Game
<<interface>>
GameEventAskGroupListener
OnGroupAccepted (…)
LogicRemotePlayers
m_model : ModelController
m_eventHandler : IGameEventHandler
m_remoteDatabase : IRemoteDatabase
m_remotePlayers : Map
Update ( )
OnUpdateReceived (…)
GetRemotePlayers ( ) : Map
LogicUnits
m_model : ModelController
m_eventHandler : IGameEventHandler
m_remoteDatabase : IRemoteDatabase
Update ( )
LogicGroup
m_playerName : String
m_player : PlayerBase
m_eventHandler : IGameEventHandler
m_remoteDatabase : IRemoteDatabase
m_remotePlayers : Map
m_invitedPlayers : List
m_lastServerPingTime : long
m_lastServerGroupLoadTime : long
Update ( )
OnGroupInviteReceived (…)
LogicLevelLoading
m_playerName : String
m_player : PlayerBase
m_model : ModelController
m_eventHandler : IGameEventHandler
m_localDatabase : ILocalDatabase
m_remoteDatabase : IRemoteDatabase
m_tileAdapter : ITileImplementationAdapter
m_currentTileID : TileIDBase
m_rechargerLoadedTiles : Map
m_remoteLoadedTiles : Map
m_assistTimes : Map
Update ( )
ReloadTile (…)
OnGameResumed ( )
OnTargetCaptured (…)
LogicCommunicationOutgoing
m_player : PlayerBase
m_model : ModelController
m_remoteCom : IRemoteCommunication
m_lastModifiedTileID : TileIDBase
m_communicationRangeCenter : Coordinates
m_savedUnitsBroadcastCountdown : int
m_lastUnitUpdatePublishTime : long
m_lastInvitePublishTime : long
m_lastTileModifyTime : double
m_savedUnitUIDs : List
m_playerInvites : Map
Update ( )
IsInvitingPlayers ( ) : boolean
InvitePlayer (…)
AddSavedUnitUID (…)
ResetSavedUnitsBroadcastCountdown ( )
ResetLastTileModifyTime ( )
SetLastModifiedTileID (…)
LogicPlayer
m_player : PlayerBase
m_model : ModelController
m_eventHandler : IGameEventHandler
m_localDatabase : ILocalDatabase
m_remoteDatabase : IRemoteDatabase
GetCurrentEnergy ( ) : int
GetMaxEnergy ( ) : int
GetNextEnergyGenTime ( ) : long
GetCurrentPower ( ) : int
GetCurrentLevel ( ) : int
GetXP ( ) : int
AwardXP (...)
LogicAsyncOp
m_asyncOperations : Map
m_eventHandler : IGameEventHandler
Update ( )
Run (…)
RunDefault (…)

Figure 23: Class diagram of the game.core.logic package. The multiplicity of all associations is 1. Constructors and private methods are hidden. Parameter lists and set data types are simplified.

erating events happen in different logic classes. In its update function LogicPlayer handles the removal of the player's trace when all units are destroyed, communication to the database server concerning the energy generation, energy rechargers logic and finally calculations required to determine if the player has assisted to capture an energy source.

LogicCommunicationOutgoing

This class handles the publishing of game data in the AOI communication implemented in the IRemoteCommunication interface. In its update function it handles the players communication range, the periodic invite messages of the group up process and the periodic game state updates sent to all nearby players. Besides, LogicCommunicationOutgoing provides functionality used by different classes for example storing and publishing the last modified tile id and time. Also specific functionality like sending invites used by LogicGroup and publishing the ids of saved troops used by LogicUnits is contained in this class.

LogicLevelLoading

LogicLevelLoading allows to request tiles to be reloaded and loads visible tiles by itself. If a tile is not saved on the database server the initial tile content is generated and uploaded to the database. This class also handles the assist XP generation. If a tile is reloaded and an energy source previously not owned by the player now belongs to the player he could have assisted in capturing this energy source. Furthermore, this class reloads all visible tiles when the device comes back from standby. Other classes use this class to reload tiles for example when an energy source is captured its tile must be reloaded to synchronise the ownership state of the source with the server. Another example is the reloading of tiles that are marked as changed by other players.

LogicGroup

This class implements the group up process. Depending on the lexical order of the player's name and the name of the other player this class either waits for a message of the other player that he also wants to group up and then informs the database server, or it waits for the own player to accept the group up and then sends a message to the other player. This way only one player communicates with the database server. In its update loop LogicGroup periodically pings the database server and refreshes the list with the group member names. Besides, the group up events and thus the group up dialog is triggered in this class.

Others logic classes

LogicRemotePlayers processes incoming game state updates from other players. It stores and updates the remote players and their units. **LogicUnits** updates and removes the units and checks if a target is captured. **LogicTowers** updates the towers and allows them to shoot the units and thereby generate weapons, which are then handled by **LogicWeapons**.

The component structure of the game logic not only improves the maintainability of the code, but also allows to easily extend the logic with new classes. If for example the group up logic needs to be changed, then the change only affects one small and clear class (LogicGroup) or in worst case two classes (LogicGroup, LogicCommunicationOutgoing) are affected, if also the communication concerning groups must be changed. If a new functionality is required such as a new object e.g. a troop shop then its logic can be encapsulated in a new class inside the game.core.logic package.

As already mentioned earlier in this thesis the communication between the controller and the view is event based. This way the controller does not know the exact implementation and types of its view, thus it is independent of the view. All event types are placed in the game.core.event package. If a new event type is required for example in the case of the troop shop above, it can be simply added to the package without changing existing code of all other event types and their handling in the view. This way similar to the GameEventAskGroup, which indicates the player that he can group up with someone and triggers a group up pop up dialog in the view, a new event could trigger a shop menu and pass callbacks to the view. The view would fire the right callbacks depending on player's actions like it is already implemented in the group up accept callback of the GameEventAskGroup.

Build and spawn logic

There are two classes not contained in the logic package with logic code. These are the TowerBuilder and the TroopsSpawner classes in the game.core package. They contain the definition of valid build and spawn points. Additionally, the TroopsSpawner class defines the attributes of the different unit types.

4.9.3 Game Model (MVC-Model)

The model related classes of the game are located in the game.core.model package. The model contains not only object data and states, but also the behaviour of each object type. The ModelController class in the game.core package is a container with the most important model types such as energy rechargers, units, towers, weapons and targets. The ModelController class allows to add, search and destroy objects. In addition, sorting towers by parent targets this container allows to do fast lookups for towers in range. The classes and enumerations contained in the model package are summarized below.

Basic Data Types

The game has its own class to represent geographical coordinates of locations. This way the core project can be independent of any map or localisation library. The **Coordinates** class stores the latitude and longitude of a location and provides simple methods to compare coordinates and convert them to meters. Besides, to keep the core independent of other libraries and projects an abstraction for tiles is needed, it is declared in the abstract class **TileIDBase**. Projects using the core must derive from this abstract class to implement a certain tile representation like it is done in the game.mapsforge.TileIDMapsforge class. The enumerations **EPlayerStatus** and **ETargetStatus** describe the group and ownership status for players and targets. **EUnitType** enumerates all unit types.

Objects

The model package provides various basic object implementations. All of them are designed in a way that allows to create derived classes, hence all of them have the 'Base' postfix. For example **UnitBase** currently does not implement shooting, but if in future some of the units will be able to shoot at towers a new class for example ShootingUnit can be derived from **UnitBase**. All needed members are protected and will be accessible in the derived class. The parent class of all objects is **ObjectModelBase**. Its purpose is to provide each object with a unique object id (UID). The only directly deriving class is **WeaponBase**. In contrast to all other objects a weapon is not bound to a certain location, but depends on the locations of two other objects. A weapon exists for a certain time period, when the time has passed the hit callback assigned in the constructor is fired. Since all other objects are bound to a certain location they all have an abstract parent class called **LocatedObjectBase**. This class declares a GetCoordinates method and provides an algorithm to convert coordinates to a unique id. The child classes **EnergyRechargerBase**, **PlayerBase**, **TargetBase**, **TowerBase**, **UnitBase** represent their objects. While the first three classes are used only for data storage also providing access methods to their data, the latter two contain behaviour logic like towers shooting at units in range or units following the player and capturing targets.

MovementTrace

The MovementTrace class stores and manages the player's location trace. This class also removes cycles in the trace. MovementTrace derives from the Observable class of the Java's standard library. Observers are notified every time the trace changes. The index of the truncated coordinate is passed in the case of a cycle being closed. The UnitBase class implements the Observer interface and reacts respectively to changes of the observed trace.

RemotePlayer

The RemotePlayer class represents another player in the vision range of the local player. All data about the other player is contained in this class, such as position and location trace, troops and currently active weapons. The remote player has its own ModelController, which shares the targets and towers with the local ModelController in the Game class. RemotePlayer moves the remote units in its update function.

Remote Data

In order to communicate the remote game state of a RemotePlayer data must be serialized and deserialized. The game.core.network.remoteData package contains classes that store and handle data serialization. All remote data types implement the game.core.model.ISerializableData interface, which allows to serialize and deserialize data to and from a byte stream. Targets, units and towers provide methods to get remote data in order to send an update to other players and also to apply received remote data to update the state of remote objects locally. Each object has its own remote data type e.g. RemoteDataTarget, RemoteDataTower, RemoteDataUnit, RemoteDataWeapon, RemoteDataPlayerObject. The RemoteDataGameState class is a container with the whole status of a RemotePlayer including units, weapons and player data. It also supports serialization by serializing all its data. The RemoteDataTile class stores the data for targets and towers of a tile.

The model can be easily extended with further object types. For example a troop shop would derive from LocatedObjectBase. If the troop shops would be calculated locally like energy generators, no remote data would be needed. Otherwise a new remote data type would be required to save the troop shop data on the database server or send ranged updates for it.

4.9.4 Visualisation (MVC-View)

The MVC-view component of the game is based on mapsforge. In contrast to Google Maps, mapsforge allows to draw the map in the dark green style of the game. The theme of the map is defined in render_theme.xml file contained in the Android apk file of the game. All other objects like units, towers, targets and the player itself are drawn in an overlay on top of the map. The render order of objects in this overlay is defined in the enumeration game.mapsforge.EOverlayLayer. The order of the entries in this enumeration can be changed. If for example the player should be drawn on top of towers then the PLAYER entry must be moved below the TOWER entry. First of all the game.mapsforge.view package will be explained and its classes listed. This package contains the single views of the objects described above in the model section. Besides, some debugging views and view managers are included in this package.

IBaseView, BaseView, ShootingBaseView

These three views build the basis for all other views. IBaseView allows one view to be drawn in multiple layers. For example the target view is drawn in three different layers. First it draws the valid build range mask, which is used later by towers to draw the valid build area for every tower. In a later layer the target icon itself is drawn. When all other icons are drawn the draw function of the TargetView class is called the third time with the TARGET_TEXT layer as parameter and the view prints the name of the target and its owner information on the overlay. Besides, IBaseView defines a Destroy function, which is called for every view before all references to it are removed. This way the view can remove its references and free resources when it is not needed any more. The abstract class BaseView implements the IBaseView interface. The BaseView forces all views to define the layers in which they should be drawn by requesting an array with layers in its constructor. This way the child classes can define the layers in one line calling the super constructor. ShootingBaseView serves the same purpose, but additionally it defines abstract methods to get weapon offsets. This way the point at the tower icon from which the laser starts can be defined and passed to the WeaponView. The code that passes on these offset values does not need to know if the shooting view is a tower or maybe after further development a unit, since it can use the base class ShootingBaseView. Hence, if further objects that can shoot and create WeaponViews need to be added to the game they should derive from ShootingBaseView. In this case applying the offsets will be done automatically. All other views should derive from BaseView.

TargetView, TowerView

These two views are responsible for the visualisation of energy sources and towers. All together they use six different layers, where two layers are used to draw the bright green areas visible on **Figure 13**. These areas show the locations at which a new tower can be built. To describe the creation process of this green area the following colour notation is used (A,R,G,B) where A is the alpha channel and R,G,B are the red, green, blue colour channels. The values for each colour channel are between 0 and 255. The colour is drawn additively so if first (1,1,0,0) and then (1,0,1,0) are draw the result is (2,1,1,0). In the beginning of every frame the colour of the whole overlay is reset to (0,0,0,0), which represents transparent black colour. The first two layers are drawn only if the player owns the target and its towers.

In the first layer VALID_BUILD_RANGE_MASK both the targets and the towers draw a mask without complying any specific order. When the mask layer is fully drawn the green build area can only be drawn on those parts of the overlay whose colour is (1,0,1,0). The targets draw a filled circle with the radius of the valid build range with the colour (1,0,1,0). This valid build range limits the maximal tower spread. To make sure that no tower can be build too close to the target another small circle with (1,1,0,0) colour inside the build range circle is drawn. The towers do the same to assure a minimal clearance between towers and draw a little circle around themselves with (1,1,0,0). The colour of the resulting mask can be (1,0,1,0), which symbolizes valid build area in the mask. Other possible colours are (X,X,1,0) or (X,X,0,0) where X is a number between 1 and 255, which highlight areas where towers cannot be build, because they are too close to another tower or the target itself. Additionally, there is still the (0,0,0,0) blank colour, which means that this spot is too far away from a target.

The second layer VALID_BUILD_RANGE is drawn only by the owned towers. Again a filled circle with the radius of the maximal build range between towers, which is a little bigger than the tower range, is drawn. This circle is only drawn on top of pixels with the (1,0,1,0) colour and has the light green colour visible in **Figure 13**. This way the circles have a hole inside, which visualises the minimal tower clearance. Also the target itself is not highlighted for building. The mask is not removed after this process, since it cannot be seen by the players because it is too dark and drawn almost 100% transparently.

In the next layer TARGET the target icon and a dashed outline around the maximal build range area are drawn. After that in the TOWER_RANGE layer the tower range is drawn in green or red depending on the target ownership. The next layer is the TOWER layer, which draws the tower icon. Finally, the target name and either the countdown or the names of the target owners are printed above the energy source in the TARGET_TEXT layer.

MovementTraceView

The movement trace view observes the MovementTrace instance in the model. Everytime the trace is updated arrays with the latitude and longitude of the trace points are created. The trace is drawn red for enemies, blue for group members and green for the current player. The flashing of the trace is realized through increasing and decreasing of the alpha component of the used colour.

FXView

The FXViews draws an icon on a given coordinate for a given time period. Currently it is only used to draw the death animation icon of the units. When a UnitView is destroyed it checks if the unit is dead. If the unit was not saved, but destroyed than the UnitView creates an FXView with the die animation using

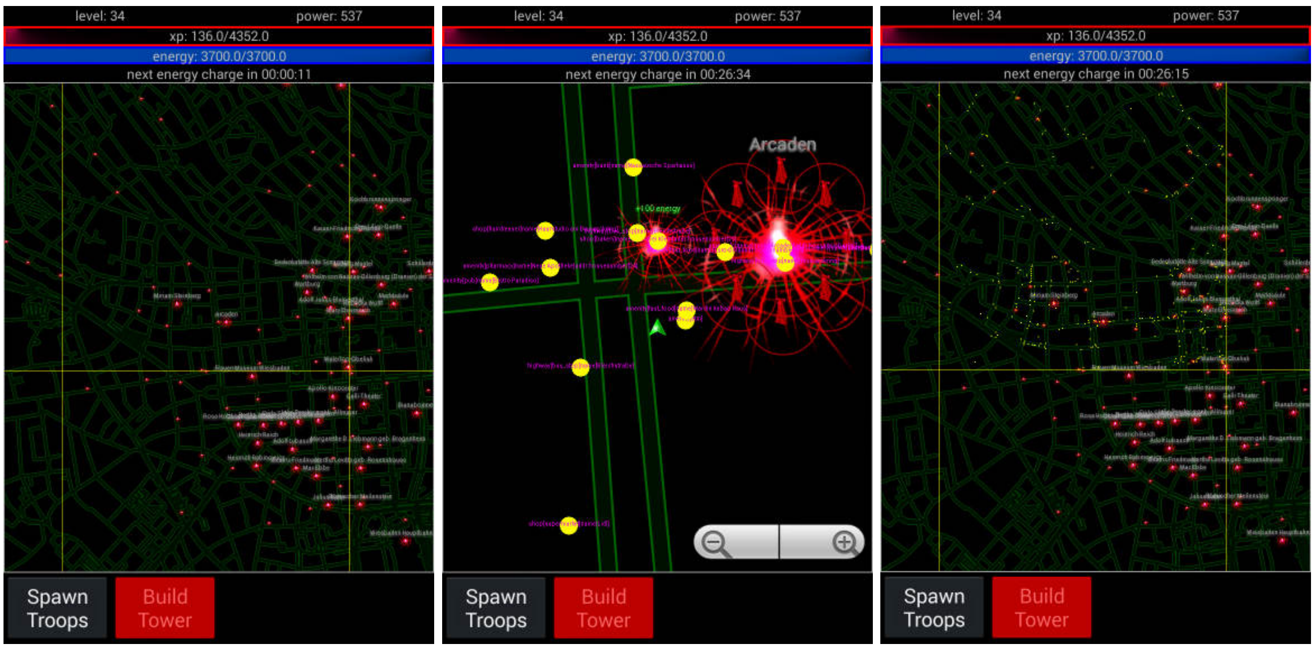


Figure 24: Screenshots of the game with enabled debug views. The left image shows a debug view with visualized tile borders. The middle picture shows a debug view showing map meta information used to generate POIs. The right picture shows both views combined.

the FXManager class. The FXView allows to pass a custom bitmap paint this way the death animation can be coloured green, blue or red depending on the owner of the unit. The FXView is time limited, when the effect time has passed the view is destroyed.

Other object views

There are still four further object views left the EnergyRechargerView, PlayerView, UnitView and the WeaponView. However, all of them are straight forward. The energy recharger view simply draws its icon if energy is available at its location. The weapon view simply draws a line representing the laser and an animated icon on the hit point. The player and unit views draw their icon after adjusting its direction.

Views representing non model data

Not all views have to represent model data. For example the GUILoadingView is responsible for showing a loading message in the right upper corner if the communication with the database server is pending for longer than a certain amount of frames (currently 15 frames). Besides, there are debugging views for example DebugTileBorderView and DebugPOIView. The first view simply draws the borders of the tiles and the second view draws meta information saved in the map files for the current tile. **Figure 24** shows the game with the debug views enabled.

ViewFactory and FXManager

The game.mapsforge.view package contains two classes helping to manage the single views. On the one hand there is the ViewFactory, which has a reference to every single existing view. It can create a view for each object of the model that has a visualisation. Besides, it can search or destroy already created views given their controlling object. On the other hand there is the FXManager, which is responsible for showing instances of the FXView class. It also removes finished effects.

To summarize, the game.mapsforge.view contains all single view implementations and their managing classes. This is the place where new views can be added. For example if the game mechanics will be enriched with troop shops in the future, than a new view called TroopShopView will be created and it will derive from the BaseView class. Another example could be a new debug view or a new HUD view similar to the view which displays the loading message in the upper right map corner.

4.9.5 The Main Overlay and Skins

Having introduced the single view classes and their low level management in the previous section, this section will explain the high level processes and handling of the views. The main class controlling all view components is the GameMainOverlay from the game.mapsforge package. This class implements the Overlay interface of the mapsforge library. It has a draw method, which is called every frame passing the canvas of the overlay to be drawn on top of the map. This function updates the game logic and the Android related GUI. Then all views are drawn in their respective layers. The GameMainOverlay instantiates the Game class and passes all the replaceable components that it gets from the ModuleConfigurator to the game constructor. It handles game events such as showing and hiding objects and showing dialogs to the user. The GameMainOverlay defines mechanics that allow to load all icons asynchronously. This prevents the game from freezing when many new objects must be loaded. Besides, this class is responsible for the player's location and handling of the GPS and screen swipe based control modes. In addition, this class creates the debug views introduced in the last section. It reads constants from the configuration file ConfigGUI like IS_DEBUG_VIEW_TILE or IS_DEBUG_POI and creates the respective views in its constructor. This way enabling debugging is one line in a configuration file. Furthermore, the debug code of a certain debug feature is encapsulated in a single view class. If further debugging views are required than a new constant should be added to ConfigGUI and the GameMainOverlay should create the new view with the requested debugging functionality.

In the previous section no details on drawing an icon were given. The reason for this is that the single view classes do not have much knowledge on what they are drawing. For example the FXView simply calls a draw method without knowing if the icon that it draws is directed or animated. Which in case of the unit die animation is both true. The game.maopsforge.skin package contains mechanics to hide such details from the views. ISkinDrawable is the interface that hides all details of what is being drawn from the views. It has a GetDrawable function that returns the Android Drawable class. The simplest implementation of ISkinDrawable is SkinIcon, which simply returns a Drawable that was passed in its constructor. This way for example static bitmaps like those of the towers can be drawn. However, some icons must be rotated before they can be drawn for example the player arrow. SkinIconDirectedRotated allows to handle this task, but since the single views should not know the implementing class another interface ISkinIconDirected is required. It simply defines setters and getters for the icons direction. If the views would reference the SkinIconDirectedRotated class directly it would be impossible to exchange the directed icon with a directed and animated icon. However, only 2D icons like the arrow can be simply rotated. A 3D icon like a unit cannot be rotated. If a unit icon would be rotated by 180 degrees then it would look like it is upside down in comparison to a tower. To solve this challenge the unit icon is rendered 32 times and rotated by 11.25° between every frame. The SkinIconDirectedFramed also implementing the ISkinIconDirected interface selects the best fitting frame of these 32 frames for the current direction. To emphasize the necessity of hiding the interface once more, the player icon could now simply be replaced by a SkinIconDirectedFramed instead of the SkinIconDirectedRotated implementation without any change in the PlayerView. In the current implementation one example for the use of the SkinIconDirectedFramed are the units. If a unit is hit then a hit animation of the laser weapon needs to be drawn, which requires one more drawable type the SkinAnimation. It works very similar to the framed icon, but this time the index of the returned icon is time dependent and recalculated every time when the GetDrawable function of the ISkinDrawable interface is called. Finally, one more class is required to draw the die animation of a unit, since it is animated and rotated. SkinAnimationDirected solves this task by selecting the right frame dependent on time and then rotating it to fit the direction.

Hence, 3d directed animations need to be drawn from a bird eye perspective with an angle as close as possible to 90°, otherwise the animation could look like it is upside down. One more detail needs to be explained here, ISkinDrawable contains more methods for example the scale of an icon. Also parameters can be set and retrieved, which is required for example in tower icons which need to define where on the image the laser can start from. All these methods are implemented in SkinDrawableBase and all above named classes derive from it.

The bridge between the views and the drawable is established by the Skin class also from the game.mapsforge.skin package. An instance of it is passed to every view by the ViewFactory. The single views call the GetMarkerIcon method of the skin passing an enumeration value from EIconType, which describes all known icons. The skin returns an ISkinDrawable, which can be one of the types explained before. This way the views do not know what they draw and the type of their icons can be exchanged at any time in the Skin class. Besides, the skin defines all parameters of the icons such as scale or weapon offsets. Additionally, also all paint types and styles are defined such as the paint for the range of a tower or the paint for the name of a target.

The methods required to draw the ISkinDrawable instances onto the canvas of the overlay are contained in the game.mapsforge.HelperGraphics class. To minimize the amount of code in the views all possible drawing methods are placed in this class. It can draw lines, circles, texts and of course ISkinDrawable taking all its parameters into account. Furthermore, it provides various intersection test methods. They are needed to early skip the drawing of objects that are not on the screen and therefore save performance.

To summarize, the GameMainOverlay class is the right place to extend the game with further debug or HUD views. The Skin class allows to easily change the icons of certain views without changing their code. Furthermore, it allows to change all line and text styles of the game. The game.mapsforge.skin can be extended with further icon types for example a directed animation, which is rotated using frames could be needed in the future. Finally, the HelperGraphics handles all the rendering and is the right place to add new rendering related functions or change the behaviour of currently existing rendering processes.

4.9.6 POI Generation

Figure 24 shows that the mapsforge open street map file format comes with meta informations containing information on different places. The POISearcher class in the game.mapsforge package provides methods to search for certain places using the meta informations from the map. Its subclass SearchTag allows to define the characteristics of places that are interesting for the game. Such a search tag consists of a key and a list of values for this key. Some examples for the searched keys used to find suitable places for energy sources are 'historic' and 'amenity'. While the historic tag does not have a list of allowed values and therefore all historic places are found and returned, the amenity key requires certain values such as 'fountain', 'clock' or 'watering place'. Usually only places that have a name are found however the SearchTag class allows to make exceptions. For example in the energy rechargers search tags the 'amenity' key with the 'bench' value does not require a name, since most benches do not have a name. In this case not only historical or in another way important benches that have a name are searched, but simply any bench. The results that were found without a name get their value, in this case 'bench', as the returned name. In addition, the SearchTag class also stores a priority this way some locations are more likely to be used as a POI than others. For example in the target search historical places and fountains are highly prioritized in comparison to the fall back solution of using railway stations. One more important parameter that can be set in a search using the POISearcher is the zoom level. The higher the zoom level, the more tags can be found. On the one hand a very high zoom level increases the search time and implicates the risk of finding too small not relevant places and on the other hand a too low zoom level would skip to many places that are too small to be included in this zoom level, due to that the overall result number would be too low. The full definition of the search tags used to find energy sources and energy rechargers can be found in the MapsforgeTileAdapter class. It is recommended to use the DebugPOIView view to find new tags or change the existing tag selection.

5 Evaluation

This section evaluates the game design and the collected data created in this thesis. First the collected data is introduced and the metrics that can be calculated from it explained. The amount of data that could have been distributed locally instead of using the internet connection of the mobile devices will be discussed. Further possible use of the collected data in simulation scenarios will be shown. Also the problems that came up during the evaluation of the data will be discussed and their solution explained. Finally, the success of the selected game design, but also its shortcomings will be discussed.

5.1 Collected Data

One of the contributions of this work is its collected data and the possibility of further data collection. This section points out the significance of the collected data and also the significance of the data that can be collected in the future. It reports on the play session in the Herrngarten, a municipal park in the German city Darmstadt. The properties and metrics of the collected data are introduced. Thereafter, statistics over the collected data concerning the advantage of using direct communication between players are presented. Finally, further evaluation and use of the collected data will be proposed.

5.1.1 Game Session

The data was collected for more than two hours. While the peak number of participant reached 16 the average number of players was 12. During this game session 50.714 messages were published. These messages were distributed among all players, since all of them played in the vision range of the game. Therefore, the p2p simulation server has forwarded 622.701 messages to the players in order to distribute the published data. **Figure 25** shows the visualisation of the game session between the 55th and the 120th minute. The positions of the players (red crosses) are shown for 12 time slots. It needs to be borne in mind that players have often switched their phone into standby during long walk sessions, which explains that the diagrams show less crosses than actually players were playing. In the beginning of the game the players were instructed to play in two groups. This way, on the one hand the groups could group up within the game and play cooperatively within their group and on the other hand each group had an enemy to fight, which was the other group. In the beginning of the session the groups moved in opposite directions and started to capture energy sources and increase their armies. In the middle of the session the groups started to capture targets previously captured by the other group. In the end of the session the members of the different groups periodically met at the same locations, while the one group was trying to capture a source the other group was building towers in the way of the attackers. During the game the groups split up in smaller groups and later merged again. The smaller groups were sent out to capture unprotected sources or to build defend towers on different main access roads of the captured energy sources. All in all, it can be said that during the data collection different participant generated diversified location traces inside an area measuring 1 km in length and 300 meters in width.

5.1.2 Analysis

Using the collected data statistics can be calculated on the amount of data that could have been sent via direct communication, such as Wi-Fi Direct or Bluetooth, instead of accessing the internet over global communication channels like GPRS. As already explained in the motivation section not using GPRS or other global connection channels can significantly decrease the energy consumption and thus increase battery lifetime[15]. Moreover, costs can be reduced by reducing the data usage exploiting free communication channels instead of using a fee charged internet connection. Furthermore, the latency can be drastically reduced and consequently, the gaming experience improved. **Figure 26** shows the amount of data that could have been sent via local direct communication depending on the maximal direct communication range. In this graph the direct communication is assumed to flawlessly work within the range

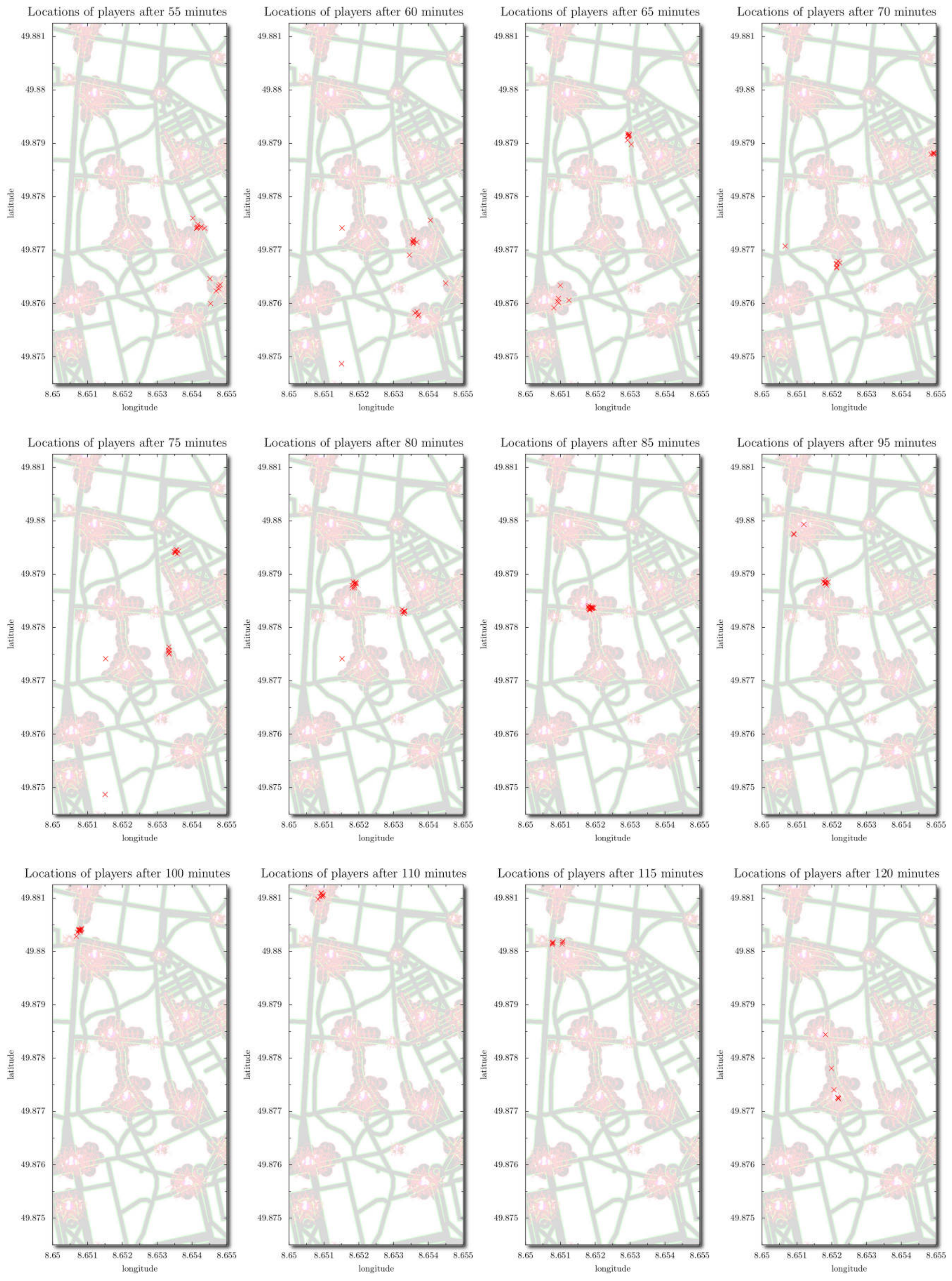

Locations of players after 55 minutes
Locations of players after 60 minutes
Locations of players after 65 minutes
Locations of players after 70 minutes
Locations of players after 75 minutes
Locations of players after 80 minutes
Locations of players after 85 minutes
Locations of players after 95 minutes
Locations of players after 100 minutes
Locations of players after 110 minutes
Locations of players after 115 minutes
Locations of players after 120 minutes
latitude
longitude
49.881
49.88
49.879
49.878
49.877
49.876
49.875
8.65
8.651
8.652
8.653
8.654
8.655

Figure 25: Visualisation of the collected data showing the location of players changing over time.

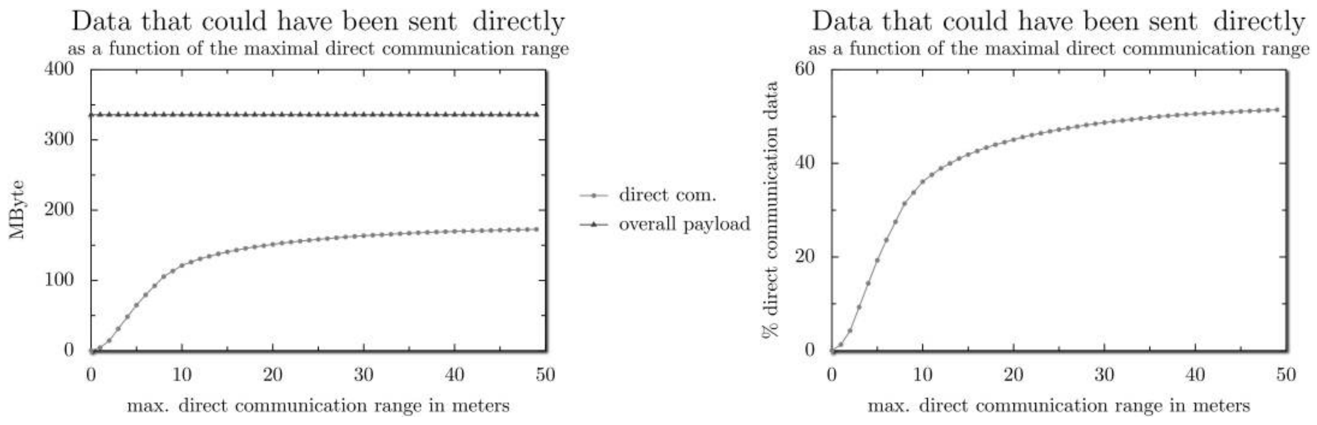


Figure 26: Two graphs showing the amount of data that can be sent directly rising with the increase of direct communication range. The left figure shows the absolute amount in MB and the right one shows % values. These diagrams are based on the collected data.

given by the horizontal axis. This way all messages between players that are recorded in the simulation server log can be sent directly from the sender to the receiver if their distance is less than the given direct communication range. Already over 30% of the bandwidth could have been saved using direct communication within 10 meters, which in most cases could have been accomplished via Bluetooth. Assuming that a Wi-Fi Direct communication would flawlessly work within a range of 20 meters approximately 45% of the transmitted data could have been efficiently distributed locally. However, further increase in the local communication range limit does not generate significantly better results evaluating the collected data. To illustrate the consequences of direct communication **Figure 27** shows the number of receivers that need to be notified indirectly for example using GPRS. It significantly drops until the range of 10 is reached, then the effect decreases rapidly. **Figure 28** shows that the results for low direct range values can be significantly improved when the data is distributed over multiple hops. The hop count of one in the diagram means that two devices are in direct communication range, which equals the results in **Figure 26**. To calculate the diagram two clients were assumed to be connected over two hops if a third client exists, that is in one hop range of both clients, this means that the range to each of the two clients is less than the limit for the direct communication. The communication over three, four and five hops is defined in the exact same way, but this time two, three or four intermediate nodes are allowed. **Figure 28** shows that, on the one hand the results can be significantly improved for short range communication technologies such as bluetooth, but on the other hand the improvements for ranges beyond 20 meters are not significant. In general considering the collected data the performance improvements seems to decline with the increasing communication range. The reason for the higher communication ranges to be not effective is the locality of players. At some point all player within close range can be reached directly if the direct communication range is high enough. It seems that during most of time the spread of the big groups did not exceed 30 meters, which implies that with the direct communication range limit of 30 meters already most of the close range players can be reached. Furthermore, independent of technology and its maximal direct communication range the communication over four or five hops does not bring further worth mentioning improvements. The best results are already achieved with a hop count of three.

Nevertheless, calculating the amount of possible direct communication the way discussed above implies that the best accessible result depends on the number of groups of devices that are further apart than the given limit for the direct communication range. For the collected data of this thesis this means that the best possible result using reasonable communication ranges is somewhere around 50%, which is also the result visible in **Figure 26** and **Figure 28**. This is explained by the fact that there were two big groups moving in different locations during most of the recorded time. If during the game session three

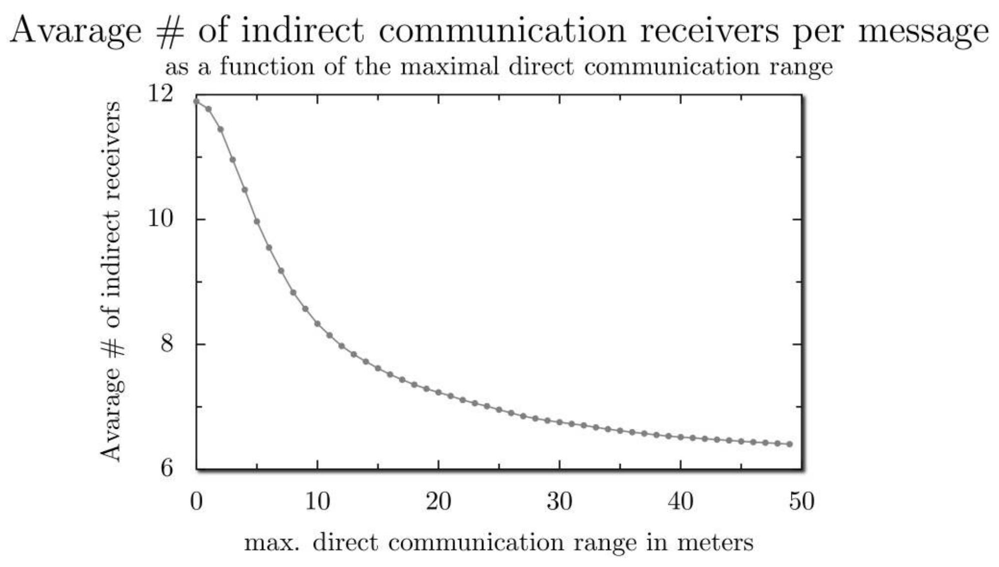


Figure 27: A graph showing the reduction of indirect receivers with the increase of the direct communication range. This diagram is based on the collected data.

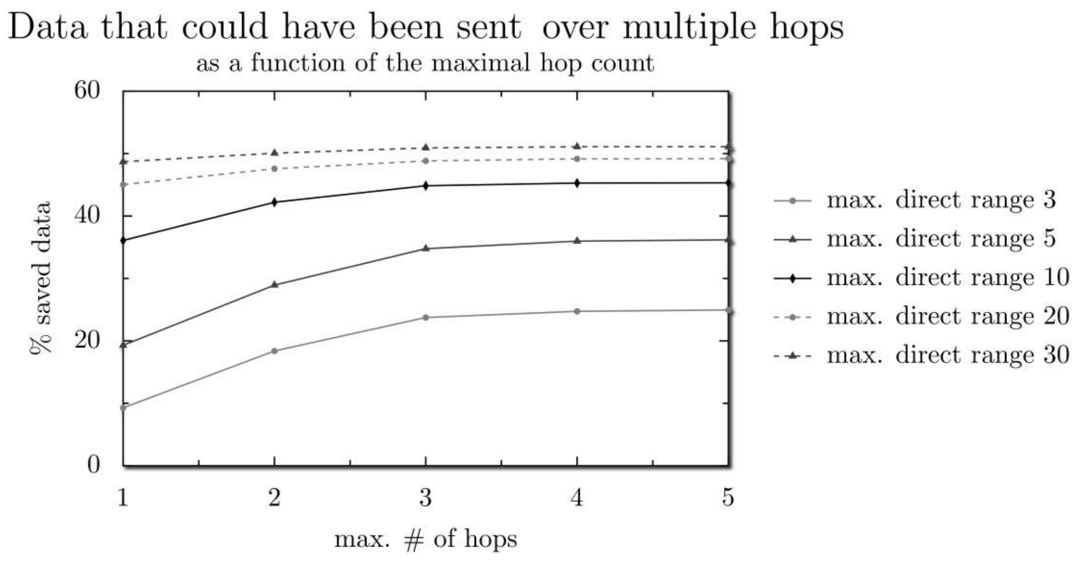


Figure 28: A graph showing the amount of data that can be sent directly increasing with the number of possible hops in direct communication for different direct communication ranges. This diagram is based on the collected data.

or four groups would exist in the vision range of each other than the best possible direct information propagation would be around 33% or 25%. These estimates are of course only true if the groups stay far away from each other most of the time. However, sending only 25% of data directly is already worth it. Even if the battery life and bandwidth usage are only slightly improved the latency to the players that are important will be better.

To further decrease the amount of data that needs to be distributed over the internet a real peer-to-peer network must be used. First of all, every group of closely located players, which can communicate over multiple hops, must form a mobile ad hoc network (MANET). All players within this group will need no internet connection to distribute the state of each player within the group. More information on MANETs, their characteristics and their management can be found here[16]. Details on possible pub/sub service organisations within a MANET are proposed by Anna Kaplun in her thesis[22]. To allow these groups organized in autonomous MANETs to communicate between each other some members of the target group must be selected. These selected members will receive the information, which they distribute over the whole MANET of the target group. This way only a small group of players gets the information over cell based communication and then this small group distributes the information over the whole MANET locally. The selection of such target group members is discussed in this paper[19]. Similar the gateway players (nodes) must be selected. If a player wants to inform another distant group of players of its state, than instead of using its cell communication to the internet the information is forwarded to the gateway nodes, which forward it to the other group using their internet connection. The selection of the gateway members is described here[21][29]. Some of the players will have to share their internet connection for a certain time interval. On the first glance one could be afraid of the high power consumption and the high bandwidth use for the single player. However, considering that altogether sending bundled information from one point to another needs much less bandwidth and energy as sending information from all points to all other points, obviously less energy and bandwidth is needed for the bundled solution. In addition, since close range communication such as Wi-Fi costs less energy than long range communication such as cell based communication all other nodes will save energy. Therefore, if the node selected as gateway changes and all nodes share their internet connection in a successive way all nodes will profit. The algorithms and systems allowing the above described operation of interconnected MANETs is a current research issue[17][19][21][22][29][31][39]. The collected data of this thesis can be used to evaluate algorithms in simulated environments. Different p2p approaches for mobile devices could use the data for simulation and evaluation of their performance. To gain more significance and validity in the statistics and simulations based on the collected data, it can be extended with further traces of future game sessions.

Another area of application for the collected data could be generation of realistic trace based simulations of social augmented reality games like Ingress. The recorded data is different from games like Ingress, since in Ingress the players cannot see each other and therefore there is no need for direct communication. However the location traces are very similar, since the game basics are the same. Players meet at POIs, move between those and hostile groups affect each other in their movement and actions.

5.2 Experienced Problems with Data Collection

All in all, the data collection was successful, however the data had to be cleaned of some unexpected log entries. The most important problem of the log was that the pub/sub simulation server was sending messages to not existing clients. When players reconnect due to communication technology switches for example from Wi-Fi to 3G they do not send a disconnect message for the old connection to the server. The server has no knowledge about the content of the messages, which is required in order to keep the simulation server replaceable. Because of this, the server cannot filter disconnected clients itself by removing the old connection if a client with the same player name reconnects. The server drops a client only if the network layer detects a time out, which can take a long time. In such a case the server simply forwards the message to both connections even if one does not exist anymore. The solution for this was to clean the log files of such entries afterwards. To do so the client id of the most recent message from every player was saved. If a message was forwarded to a player and the used client id was not the most recent one, then it was assumed that the message would not be delivered anyway and therefore it was removed from the log. This way every forwarded message for a certain player was sent to only one client. Since during the game session only students and university employees were playing and the Wi-Fi of the university could be used often in the municipal park the devices often switched between Wi-Fi and cell based communication. Over 130.000 forwarded messages had to be skipped. This reduced the average receiver per message number from approximately 15 to 12.

Another problem with the log file was that due to multi-threading issues some log passages were written by multiple threads simultaneously and therefore became useless. Using the synchronized standard Java component PrintStream it was ensured that single lines could not be interleaved. The processing of lists of receivers and logging every sent message was put into synchronized blocks to guarantee that log blocks describing to whom a message was forwarded could not interleave. However, the 'forwarding' and the 'forwarded' log lines (see implementation section) were not synchronized to minimize the IO related lock wait times and improve the performance. These lines are needed to determine from which client a message was sent. This way it was possible that multiple 'forwarding' lines were printed one after another, which made it impossible to determine from which client the following forwarded messages were sent. Luckily, the publishing rate of the clients was relatively low and less than 100 occurrences among over 50 thousand messages were found in the log file. This broken log passages were partly removed by hand and ignored by the scripts used to generate the statistics and diagrams above. The bug was fixed and will not occur in future data collections. No change in the log data format was required.

5.3 Game Design Discussion

The game design created in this thesis worked well. The social contact aspect of the game was fully satisfying. The players had fun working together, thus a feeling of group affiliation was formed, which allowed unfamiliar persons to speak with each other not only about topics concerning the game. Playing the game was motivating enough to make a group of people run up a steep hill to capture an energy source. Also no one complained about the multiple kilometres that the players travelled running between the energy sources, which indicates that the players were very concentrated while playing the game. The biggest achievement of the game session was that a group of players stayed together for lunch afterwards.

However, the current implemented game design has some draw backs. The biggest problem that needs to be solved next is that low level players cannot build towers in places where higher level player have played, since the power requirement to build a tower is much higher that the initial power that a new player has. One possible solution for this is to allow units to destroy towers like it is already proposed in the game design section. This way also high level players will profit. If a high level player captures a target he usually also cannot build many towers if the point was prior captured by a strong player. Destroying towers would allow the players to customize the world fitting to their needs, which would make the game more motivating and allow the player to make more interactions. Additionally, the low level players would be able to gain XP even if they are not able to capture a target by destroying its towers one by one. This way a low level player would be able to capture a target after many attempts, maybe even multiple days, which would be very satisfying, since the player would profit from the hard work.

6 Conclusion and Future Work

This is the final chapter of this thesis. It will conclude the contribution of this thesis and present its planned future use. Further research based on the collected data will be proposed and also possible extensions and improvements of the game design will be mentioned.

6.1 Conclusion

In this thesis an innovative Android based massively multiplayer online augmented reality game was designed and implemented. It was explained that using short range communication for mobile phone games has many advantages in comparison with server based solutions. The advantages are high especially for augmented reality games, which bring people spatially together. Some state of the art augmented reality games such as Ingress, Life is Magic and a few others were presented. To allow short range communication using diverse network technologies the created game uses the pub/sub communication paradigm for network data distributed within the vision range of the players. Additionally, the communication implementation used in the game can be easily exchanged by diverse other pub/sub solutions. The contributed pub/sub implementation is server based and allows to accurately log every sent message and its distribution in the network. In this thesis a game session with in average 12 players was accomplished and its pub/sub network traffic logged. This session not only collected data, but also proved the game design to work, be fun and to be able to motivate people to move in the real world. With the collected data is was possible to calculate statistics showing how different communication technologies such as close range direct communication over Wi-Fi Direct or Bluetooth could outperform the cell based client-server approach if they would have been used. It was stated that up to 50% of the data sent to the internet during the game session could have been distributed locally instead of using the fee charged internet connection. Further data collections can be executed with the game of this thesis in order to increase the validity and significance of the calculated statistics. Furthermore, the game developed in this thesis makes it possible to comprehensibly visualize the effects of different network technologies on games especially the latency implied by different technologies can be observed. This allows to present the game in conferences in order to explain the advantages and disadvantages of certain network technologies, also to people without technical background. Additionally, the battery consumption of different implementations and technologies can be evaluated in the case of application in a game.

6.2 Future Work

This section will present the planned future development of this project. Proposals for future use of this work will also be made. Also the possible improvements and extensions for the game will be discussed. This section is separated into a scientific section that discusses the collected data and a game design section which is focused on the game mechanics.

6.2.1 Scientific

The first step of further development of the game is to exchange its server based pub/sub network implementation by a real mobile ad hoc network (MANET). Once this has been done, the bandwidth use, the latency and the battery usage can be compared to the server based solution and interesting questions can be answered for example: is the bandwidth use of a fee charged internet connection reduced for every player? Is the latency improved in close range? Is the latency improved for far away players? Is the battery life increased for every player?

The game can also be used as a presentation of different network technologies. A possible scenario of a presentation in a conference would be showing the game with different network technologies. This way it could be comprehensively shown that (if it is indeed true, which must be researched first) the battery life is increased when cell based technology use is reduced. The latency in close range is improved using a MANET solution, which again improves the gaming experience. And that the bandwidth use of cell based communication can be reduced using direct communication for example in a MANET.

Another application area for the collected data of the game is to use it for the simulation of different network technologies. This way the performance especially the bandwidth usage and the latency between players can be compared for different network technologies. For example it could be tested if a MANET solution with a complex routing algorithm outperforms a Wi-Fi Direct communication of each player to each other player in the communication range. It would also be interesting to know how close real technologies are to the theoretical evaluation and statistics explained in the evaluation section of this work.

The last proposal for a future work is to collect BlueTooth and Wi-Fi connectivity between players during a big game session. Collecting such data should not be a big problem, since it was already done many years ago[30]. Maybe even an independent already existing solution could be used. The data of the game and the connectivity data could be evaluated in parallel and be compared later on. This way statistics could be based not only on assumptions that a BlueTooth communication would flawlessly work within 10 meters and Wi-Fi should work for 20 meters, but on real recorded connectivity states of mobile phones. This way the statistics of the evaluation section could be validated.

6.2.2 Game Design

The possible extensions that can be applied to the game design are already mentioned in the evaluation and the game design sections. To sum up the further development, a tower destruction feature allowing units to attack towers is a good starting point for future work, but also different tower types and mechanics allowing to move existing towers are imaginably. Also energy source upgrades increasing the energy and power generation could be added. The exploration component of the game could be improved by allowing players to find new troop types at certain places for example troop shops. Another already mentioned point is a majority vote system for new energy sources allowing to fill parts of the world that lack energy sources by user generated content. All of the features above are not only interesting, because they could make the game more fun, but because all of these features would generate more specific network traffic and allow more precise studies of network technologies that implement the communication of the game.

However, in this whole thesis no word was written concerning sounds. This did not happen accidently, since it is questionable if a mobile phone augmented reality game really needs sounds. Players playing the game with sound would attract attention, which is presumably undesired by players. No one wants the surrounding people to look strange, because of the sounds you phone makes. Because of this, sound was considered to be a minor feature, which does not need to be implemented in a scientific game. Nevertheless, sound could be used by players wearing earphones. Also games like Ingress have sound even if most players are probably turning it off. A game with sounds simply makes a much more professional impression even if they are heard only once and turned off afterwards. Therefore, sounds could also be implemented in a future work.

References

[1] Blizzard Entertainment on statistic brain. http://www.statisticbrain.com/blizzard-entertainment-statistics/. [Online; accessed 10-February-2014].

[2] Google Glass homepage. http://www.google.com/glass. [Online; accessed 10-February-2014].

[3] Ingress homepage. http://www.ingress.com/. [Online; accessed 10-February-2014].

[4] MAKI at DFG. http://www.dfg.de/en/research_funding/programmes/list/projectdetails/index.jsp?id=210487104. [Online; accessed 10-February-2014].

[5] MAKI homepage. http://www.maki.tu-darmstadt.de/sfb_maki/index.de.jsp. [Online; accessed 10-February-2014].

[6] Multi-Mechanism-Adaption for the Future Internet (MAKI). http://www.ps.tu-darmstadt.de/research/maki/. [Online; accessed 10-February-2014].

[7] Technische Universität Darmstadt (TUD) homepage. http://www.tu-darmstadt.de/. [Online; accessed 10-February-2014].

[8] World of Warcraft on IGN. http://www.ign.com/games/world-of-warcraft. [Online; accessed 10-February-2014].

[9] World of Warcraft on statista. http://www.statista.com/statistics/276601/number-of-world-of-warcraft-subscribers-by-quarter/. [Online; accessed 10-February-2014].

[10] World of Warcraft on Wikipedia. http://de.wikipedia.org/wiki/World_of_Warcraft. [Online; accessed 10-February-2014].

[11] mapsforge. http://code.google.com/p/mapsforge, 2013. [Online; accessed 30-January-2014].

[12] Matt Barton. The History of Computer Role-Playing. http://www.gamasutra.com/view/feature/3623/the_history_of_computer_.php, 2007. [Online; accessed 04-December-2013].

[13] 11 bit studios. Anomaly Warzone Earth. http://www.11bitstudios.com/games/7/anomaly-warzone-earth, 2011. [Online; accessed 04-December-2013].

[14] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device to device communications with wi-fi direct: overview and experimentation. *Wireless Communications Magazine*, 20(3):96–104, June 2013.

[15] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[16] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, 1999.

[17] Carlos Giraldo, Laurent Franck, Cédric Baudoin, and André-Luc Beylot. Improving manet routing with satellite out-of-band signaling. *International Journal of Satellite Communications and Networking*, 31(6):303–315, November 2013.

[18] C. Glenday. *Guinness World Records 2009*. Guinness World Records. Bantam Books, 2009.

[19] Bo Han, Pan Hui, V.S. Anil Kumar, Madhav V. Marathe, Guanhong Pei, and Aravind Srinivasan. Cellular traffic offloading through opportunistic communications: A case study. In *Proceedings of the 5th ACM Workshop on Challenged Networks*, CHANTS '10, pages 31–38, New York, NY, USA, 2010. ACM.

[20] Jeroen Hoebeke, Ingrid Moerman, Bart Dhoedt, and Piet Demeester. An Overview of Mobile Ad Hoc Networks: Applications and Challenges. *Journal of the Communications Network*, 3:60–66, 2004.

[21] Shahid Md. Asif Iqbal. *Accessing Internet from the MANET: Discovering and Selecting Gateways*. LAP Lambert Academic Publishing, Germany, 2012.

[22] Anna Kaplun and Roy Friedman. A density-driven publish subscribe service for mobile ad-hoc networks. *Ad Hoc Networks*, 11(1):522–540, 2013.

[23] Niko Kiukkonen, Jan Blom, Olivier Dousse, and Juha.K Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. In *ICPS 2010: The 7th International Conference on Pervasive Services*, Berlin, 2010.

[24] Mohit Kumar and Mishra Rashmi. An Overview of MANET: History, Challenges and Applications. *Indian Journal of Computer Science and Engineering*, 3:121–126, 2012.

[25] Denis Lapiner. Gameplay design and implementation for a massively multiplayer online game. Bsc. thesis, Technische Universität Darmstadt, March 2011.

[26] Max Lehn, Christof Leng, Robert Rehner, Tonio Triebel, and Alejandro Buchmann. An online gaming testbed for peer-to-peer architectures. In *Proceedings of ACM SIGCOMMÍ1*. ACM, August 2011. Demo.

[27] Max Lehn, Tonio Triebel, Robert Rehner, Kopf Stephan Guthier, Benjamin, Alejandro Buchmann, and Wolfgang Effelsberg. On synthetic workloads for multiplayer online games: a methodology for generating representative shooter game workloads. *Multimedia Systems (MMSJ)*, pages 1–12, February 2014. The final publication is available at http://link.springer.com.

[28] C.E. Palazzi, M. Roccetti, G Pau, and Gerla M. Online games on wheels: fast game event delivery in vehicular ad-hoc networks. In *Proceedings of the 3rd International Workshop on Vehicle-to-Vehicle Communications 2007*, In IEEE Intelligent Vehicles Symposium 2007, pages 42–49, 2007.

[29] Deepak Patel and Rakesh Kumar. A review of internet gateway discovery approaches for mobile adhoc networks. *International Journal of Computers & Technology*, 4(2b2), 2013.

[30] Anna Kaisa Pietilainen and Christophe Diot. Experimenting with real-life opportunistic communications using windows mobile devices. In *Proceedings of the 2007 ACM CoNEXT Conference*, CoNEXT '07, pages 65:1–65:2, New York, NY, USA, 2007. ACM.

[31] Faisal Rehman, Sajjad A. Madani, Kashif Bilal, and Khizar Hayat. Discovery of multiple mobile gateways in wireless mesh networks. *World Applied Sciences Journal*, 15(4):590–597, 2011.

[32] Sebastián Matas Riera, Oliver Wellnitz, and Lars Wolf. A zone-based gaming architecture for ad-hoc networks. In *Proceedings of the 2Nd Workshop on Network and System Support for Games*, NetGames '03, pages 72–76, New York, NY, USA, 2003. ACM.

[33] Suman Srinivasan, Arezu Moghadam, Se Gi Hong, and Henning Schulzrinne. 7ds - node cooperation and information exchange in mostly disconnected networks. In *ICC*, pages 3921–3927. IEEE, 2007.

[34] Philipp Svoboda, Fabio Ricciato, Werner Keim, and Markus Rupp. Measured web performance in gprs, edge, umts and hsdpa with and without caching. pages 1–6. IEEE, December 2007.

[35] Ozan K. Tonguz and Mate Boban. Multiplayer games over vehicular ad hoc networks: A new application. *Ad Hoc Netw.*, 8(5):531–543, July 2010.

[36] Max Lehn Robert Rehner Tonio Triebel, Benjamin Guthier. Planet Pi4 on Department of Computer Science IV Uni-Mannheim. `http://ls.wim.uni-mannheim.de/de/pi4/research/projects/planet-pi4/`. [Online; accessed 10-February-2014].

[37] Tonio Triebel, Max Lehn, Robert Rehner, Benjamin Guthier, Stephan Kopf, and Wolfgang Effelsberg. Generation of Synthetic Workloads for Multiplayer Online Gaming Benchmarks. In *International Workshop on Network and Systems Support for Games (NetGameś12)*. IEEE, November 2012.

[38] Sergey Tsalkov. MeekroDB. `http://www.meekro.com`, 2013. [Online; accessed 30-January-2014].

[39] Long Vu, Klara Nahrstedt, Rahul Malik, and Qiyan Wang. Coada: Leveraging dynamic coalition peer-to-peer network for adaptive content download of cellular users. *International Journal of Adaptive, Resilient and Autonomic Systems*, 2(2):1–22, April 2011.

[40] Jiazi YI. A Survey on the Applications of MANET, 2008.