NTNU
Norwegian University of
Science and Technology

Online Location-based Mobile Gaming

CityZombie - A basic approach to introducing location in mobile games

Øyvind Rolland

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Alf Inge Wang, IDI
Co-supervisor: Anne Marte Hjemås, Telenor

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

This thesis' main objective is to explore the challenges and possibilities of online location-based gaming in a present day urban environment.

To achieve usable data we will design, implement and test an online multiplayer location-based mobile game using Java Micro Edition. The design and implementation will be based on a prototype concept developed in our depth project, which was a state of the art analysis of location-based mobile gaming.

User testing will be performed, and the feedback will be documented and evaluated in this report. This testing will include aspects related to infrastructure, such as GSM and UMTS base station coverage, overlapping and availability. Testing also include aspects related to network, such as practical GSM and UMTS response times, network stability and operator specific properties. Finally, aspects related to game playability and developer specific challenges when making and testing a location-based game will be tested and evaluated.

Assignment given: 15. January 2008
Supervisor: Alf Inge Wang, IDI

Abstract

Mobile phone gaming has seen an enormous growth over the last decade and many countries now have more cell phone subscriptions than they have people. Combined with the ever increasing interest in games, the mobile gaming market still hasn't reached it's full potential. Newer and more powerful phones with interesting features hit the market every day. Many of those features are directed at locating position, and that opens up for the prospect of location-based gaming. This branch of gaming is still in the starting phase, and public awareness is relatively low. Meanwhile, developers and operators explore the opportunities that are yet to be tested.

In this project, we have developed a game to test different aspects related to infrastructure, network and playability in location-based mobile gaming. From the testing of GSM and UMTS networks, results show that basic network properties such as CellID can be utilized to make challenging and social games. Furthermore, the tests hinted at UMTS as the best suited network to perform location-based gaming, as the smaller and less overlapping zone structure, response times and bandwidth facilitates a multiplayer game better than the GSM network. This is not a global truth, though, since EDGE increases the GSM response times and bandwidth to an acceptable level. As long as a suitable mapping of game zones and real-world cells can be made, GSM is still a candidate in many environments.

Developers face new challenges with location-based gaming, as testing no longer can be confined to the development environment. Extensive field testing is vital to both flesh out techincal issues as well as gameplay-related issues. Outdoor distances makes testing take longer time, and communication between test team and developers is harder than in a studio.

Finally, playtesting shows that most people are open to the idea of physical movement as input when playing in a virtual reality. When incorporating social elements and multiplayer options in a dynamic setting, location-based games can be seen as a replacement for a friendly football game or similar outdoor activities. When such an approach can be made, while keeping communication and team-feeling among players high, the full potential of location-based gaming migh be unlocked.

Preface

This report is the result of a master's thesis project that was performed in TDT4900 *Computer Science* by a fifth year student finishing the Master of Science degree at the Norwegian University of Science and Technology. The work was carried out from January to June 2008.

I would like to thank my supervisors Anne Marte Hjemås at Telenor for supporting this project with ideas and feedback as well as a test phone and Alf Inge Wang at IDI for equally valuable feedback and ideas.

Trondheim, June 10, 2008

Øyvind Rolland

Contents

List of Figures

List of Tables

Part I

Introduction & research

Chapter 1

Introduction

In the introduction we will state the main project objectives and our motivation related to the assignment. Further we will explain the project context and give an outline of the rest of this thesis with a presentation of the contents in each chapter.

1.1 Motivation

The gaming industry in general is one of the fastest growing industries in the world, surpassing even important parts of the movie industry in revenues, but the games created so far have largely been directed at personal computers and gaming consoles. However, with the development of more and more advanced mobile devices, the industry has begun to look at these as gaming machines, not only communication tools.

In addition to displaying relatively large computational power, mobile phones are also small, connected and available to the average person. Add the location technology present in today's devices and you get many new opportunities for games and applications, not just another console or computer.

More specifically, while the sheer size of a phone gives us mobility, new wireless technology gives us connectivity. The general population's communication dependency gives us phone availability and the population phone density gives us a large player base with which to cooperate and compete. Finally, modern infrastructure gives us the ability to locate our devices. Adding these together should give us a great base for developing online location-based mobile games.

Currently, there is little research and development being done in the field of location-based mobile gaming in Norway, even though such games have proven to be successful both at an academic and commercial level elsewhere. However, Norwegian mobile infrastructure is highly developed and mobile users are abundant, with a population phone density of 92% [EKO], giving plenty incentive to perform such research.

1.2 Problem definition

This thesis' main objective is to explore the challenges and possibilities of online location-based gaming in a present day urban environment.

To achieve usable data we will design, implement and test an online multiplayer location-based mobile game using Java Micro Edition. The design and implementation will be based on a prototype concept developed in our depth project, which was a state of the art analysis of location-based mobile gaming.

User testing will be performed, and the feedback will be documented and evaluated in this report. This testing will include aspects related to infrastructure, such as GSM and UMTS base station coverage, overlapping and availability. Testing also include aspects related to network, such as practical GSM and UMTS response times, network stability and operator specific properties. Finally, aspects related to game playability and developer specific challenges when making and testing a location-based game will be tested and evaluated.

1.3 Summary of previous work

A research project carried out by the author during the fall of 2007 [ONL] forms the basis of this thesis. Since that project has not been published, a brief summary related to this thesis is given here.

The project's main objective was to perform a state of the art analysis of online location-based games and to evaluate these in relation to what telecom companies can provide to the field. Thus a number of games and technologies were evaluated and picking from these, a new concept was introduced. It is this concept we elaborate upon and further develop in this thesis.

1.4 Important definitions

Before we begin the formalities, a quick definition of the main topics of this report is in order.
According to the mobile gaming industry, mobile games are defined as:

"...those that are delivered via wireless networks to devices whose primary function is a mobile phone".[MGW]

A location-based game is not that easy to define, and many answers are given, depending on who is asked.
From an academic point of view, a location based game is defined as such:

"In a location-based game(...) the players themselves move around in the real world. They interact with the game by changing their position and visiting certain places that are of interest to the game. They still interact with(...) computers using various standard input devices[1]*, but that is secondary to the game. Players can meet in the real world, and interact with each other in the game context".*[NPM]

While from a marketing point of view, a location based game is defined in this way:

"In a location-based game, the developer creates a virtual world on top of the real world in order to use real world objects and surroundings in the game. (...) Mobile positioning technology makes it possible to always know where the user is located."[JDB]

1.5 Project outline

The project outline serves as a reader's guide to the entire report, with a summary of each chapter.

Part I - Introduction

- Chapter 2 introduces our research questions, and the methods we use to answer them.
- Chapter 3 outlines our development methods and contains a description of the tools used in our project.

Part II - Prestudy

- Chapter 4 explains central development concepts relevant to out project.
- Chapter 5 is a review of the different technologies that we make use of in constructing our game.
- Chapter 6 summarizes the state of the art in location-based gaming.
- Chapter 7 introduces the methods to be used in the game playtesting.

Part III - Own Contribution

- Chapter 8 offers a deeper look into the premise and concept of our game prototype.
- Chapter 9 shows the requirements attached to our project.
- Chapter 10 presents our implemented game architecture and design.

Part IV - Test results and user feedback

[1] Primarily mobile phones and personal digital assistants (PDA's)

- Chapter 11 summarizes the test results.
- Chapter 12 explains any problems encountered throughout the project.
- Chapter 13 shows tables identifying whether the requirements have been fulfilled or not.

Part V - Summary

- Chapter 14 is an evaluation of the project, both in technical and methodical terms.
- Chapter 15 summarizes how the research questions were answered.
- Chapter 16 gives a conclusion to our project work.
- Chapter 17 suggests further work.

Chapter 2

Research methods & questions

In this chapter we state the research methods that have been used during the process of writing this report, as well as the research questions that are addressed and answered.

2.1 Research questions

The purpose of the research questions is to give the output of this project and dictate the constraints of the project work in line with the topics given in the Problem Definition section of Chapter 1. The questions are derived from discussions between the author and the advisors.

The research questions to be answered in this report include:

- **RQ 1 -**Which challenges exist when developing location-based games for mobile phones?

- **RQ 2 -**What is the impact of adding an online multiplayer client/server-structure to a location-based game?

- **RQ 3 -**What basis does the current infrastructure offer to support a location-based mobile game?

- **RQ 4 -**Which technologies support the continued existence of a location-based multiplayer mobile game the best?

- **RQ 5 -**How do people respond to a location-based game as opposite to a passive/stationary game?

Now that the question are asked, we identify what methods we must use in order to reach a conclusion to our research.

2.2 Methods used

Software engineering is a multi-disciplinary research subject, consisting of both social and technological factors. This is especially true when it comes to game development, a field dominated by individual ideas and creativity, and as such a formal approach is required in order to be able to experiment on the research at a later stage.

The most frequently used research methods in software engineering are given by Basili [BAS] below.

- The empirical method - *The empirical method relies on a statistical method used to verify a hypothesis. After collecting and analyzing empirical data, one can verify or falsify the hypothesis made.*
- The engineering method - *The engineering experimental method is based on engineers building and testing a system according to a hypothesis. When confronted with the result of a test, the engineers improve the solution until no further improvements can be made.*
- The mathematical method - *The mathematical method is a formal approach and depends on proposing formal theories. The results derived from a theory can in turn be compared to empirical observations.*

Answering RQ1 and RQ2 requires experience through trying and testing. In the project description of the introduction chapter, we defined the development and analysis of a location-based game as our main goal. The experiences from developing that game will help us answer the first two questions. In this respect, the *engineering approach* seems to be best method. The game will be implemented in stages, with continous tester feedback. When finished, the game will be applied to tests in a real-world environment where we can evaluate different aspects concerning infrastructure, network and playability. The *empirical method* will be supplementally used when extracting objective data concerning these three aspects from the phones and test group in order to answer RQ3, RQ4 and RQ5.

In addition to the research methods mentioned above, for answering research questions,

different methods for technology validation are also available. Zelkowitz and Wallace created a taxonomy [MZW] of twelve such methods, divided into three main categories. These are listed in table 2.1.

Historical	**Observational**	**Controlled**
Literature Search	Project Monitoring	Replicated Experiment
Legacy Data	Case Study	Synthetic environment experiments
Lessons Learned	Assertion	Dynamic Analysis
Static Analysis	Field Study	Simulation

Table 2.1: Validation Methods

To accomodate the nature of the research questions, the validation methods used as a part of the engineering method in this report can be split in three, *Lessons Learned*, *Literature Search* and a *Case Study*.

In our depth project [ONL] we looked at literature covering different technologies for implementing location in a mobile device, examined programming languages that support mobile game development and studied an array of developed location-based games. All sections contained a comparative analysis of the different elements within. The experiences from the depth study lay much of the foundation for this project, and we will use this knowledge in improving this project according to the lessons learned method. We will add to these experiences by studying and analyzing further games and concepts according to the literature search method.

Furthermore, the developed protype will be subject to a controlled test, a limited case study with feedback from users, from which empirical data will be extracted. These data, paired with the analysis from and additions to the previous literature search, form the basis for our conclusion.

Chapter 3

Development methods and tools

In this chapter, we first describe the development methods we will use during this project. At the end of the chapter, a list of the development tools we will use is provided, along with a short description of each tool.

3.1 Development method

In a software development project, many process methods exist to help the development team perform and organize tasks such as design, implementation and testing. Some methods are better suited for larger teams or long development periods, some fit shorter periods and smaller teams.

3.1.1 Waterfall

The waterfall method is the traditional and most used method of software development. It entails a five-phase step by step way of progressing through a project. These phases, as seen in figure 3.1, are Requirement analysis, Design, Implementation, Verification and Maintenance. While being simple and intuitive, the method does not contain any

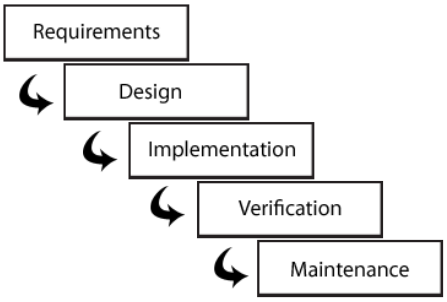

Figure 3.1: Illustration of the Waterfall method phases

feedback loops, just a straightforward and linear sequence. Projects using the waterfall method as a starting point often modifies it with overlaps, in order to incorporate feedback into the design and implementation.

3.1.2 Scrum

Given the nature of this project, an agile [AGI] development method seems more efficient and natural. Agile development is a concept based on developing software in short amounts of time, with many iterations throughout the project. Iterations normally last from 1 to 4 weeks. This means that the focus is not on planning and making documentation, but on doing and communication between the stakeholders. The resulting advantages of agile development are adaptability to change, and working software at an early stage.

One of the agile methods is Scrum [SCR], which is really more of a project management framework for agile development. Figure 3.2 shows a diagram describing the Scrum process for Agile development. Iterations in Scrum are called **Sprints**, in which the development team creates working increments of the software product. Scrum focuses on being adaptive through **Project Roles** and best practices.

One such practice is **User Stories**. These stories are a way for users to communicate wishes for a system without specifying them as a rigid requirement. A user story emphasize the user's goals, not a system's attribute. When

The **Product Owner** is the one who at first compiles any changes planned for the product and prioritizes the functionalities. This is similar to a customer representative giving demands as in a requirement specification, but mainly in the form of desired functionality or **User Stories**. User stories emphazise the user's goals, not a system's attributes. The list of stories/functionalities results in a **Product Backlog**, which is a to-do list that is reprioritized as fit. Before each sprint is initiated, a number of objectives, including the ones with the highest priority are placed in a **Sprint Backlog**, which is a list of goals to be implemented this sprint.

After finishing a Sprint, the Scrum Team holds a **Sprint Retrospective**. This is a meeting where the team members reflect about the sprint they just went through. Positive and negative experiences are dealt with and analyzed. This way, the team can learn something from the process, and improve in the next sprint.

The **Scrum Team** consists of the actual project members and a product user. The goals of each sprint, the Sprint Backlog, are discussed between the Scrum Team and the Product Owner, and the functionality is broken down to detailed tasks. These tasks dictate what the Scrum Team will be doing this sprint. Finally, the **Scrum Master** acts as an intermediary or buffer between the product owner and the scrum team. He coaches the development team and makes sure they are on track and comfortable without interfering in their work. He or she also makes sure that the team is not disturbed by

external elements during the sprint.

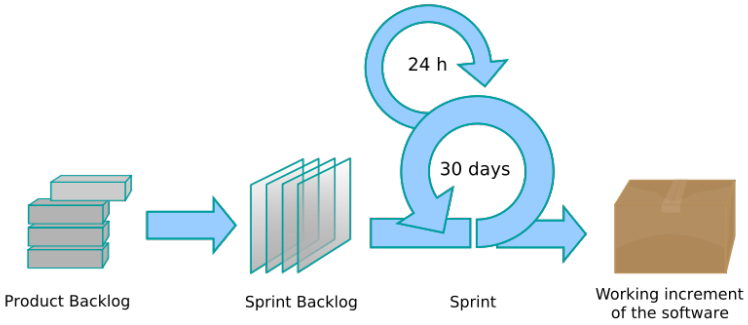

Figure 3.2: Illustration of the SCRUM management process. [SMP]

3.1.3 Our development method - Solo Scrum

Solo Scrum [SSC] is an adapted version of Scrum for use in single person projects. While team communication and meetings, a forte of Scrum, is nonexistant, the projects still benefits from general Scrum principles such as: the product backlog, the sprint backlog, the sprint and the sprint retrospective. While we exclusively use Scrum in our development process, the Waterfall method phases will influence this report through the chapter structure.

In our project, the author will act as both Scrum Team and Scrum Master. The advisors fit the role as Product Owner. Our sprints will last approximately 14 days each and will be monitored through ScrumWorks.

Our user stories will follow the who-what-why form suggested by Mike Cohn [USR]:

- As a <user role>, I want to <perform some task>, so that I <reach some goal>.

3.2 Development tools

The following tools and applications have been used in this project, not only for developing the prototype but also for organizing the project and writing this report.

Netbeans 6.0.1- Netbeans is a Java-based programming environment, or Integrated Development Environment (IDE), developed by SUN and designed to increase a programmer's productivity. It supports development in both Java EE, Java SE and Java ME, where the last two are the most interesting aspects for our project.

MiKTeX 2.6 - MiKTeX is a Windows implementation of the typesetting system TeX. It includes a compiler, a LATEX to PDF converter, and other utilities [MIK].

TeXnicCenter Beta 7.01 - TeXnicCenter is a LATEX-editor used to easily write and structure larger documents written in LATEX [TXC]. When combined with a compiler such as MiKTeX, TeXnicCenter provide a complete environment for writing and compiling LATEX documents.

Sun Java Wireless Toolkit 2.5.2 - The Java Wireless Toolkit (JWT) (formerly known as WTK) contains the packages and classes supported by the standard Java ME implemen- tations. The toolkit is needed to compile Java source files and run these on a computer using the standard Java emulators.

Sony Ericsson SDK 2.5.0.1 for Java ME - The Sony Ericsson toolkit for Java ME is an extension to the JWT and provides new device profiles [SEJ].

Java Development Kit 6.0 - A Java Development Kit (JDK) contains packages and applications needed to compile and run Java applications. The JDK 6.0 is the latest JDK from Sun and its new enhancements include improved I/O support, improved performance and security, support for generics, and an improved Virtual Machine [JDK].

PaceStar UML Diagrammer 6.02 - A basic diagramming tool for creating Unified Modeling Language (UML) diagrams. This program will be used to create state charts, class diagrams and architectural descriptions in this project.

Adobe Photoshop Elements 6.0 - Adobe Photoshop Elements is a relatively advanced image editing program that is easy to use. It will be used in this project to complement PaceStar in creating illustrations and figures.

ScrumWorks Basic Edition - ScrumWorks [SCW] is an Agile process automation tool through which development teams are able to organize themselves and the development process. It is based on the Scrum process, an agile development framework described above, and will be used to create task lists and backlogs during the project.

Altova UModel 2008 rel.2 - Altova UModel is an application for creating UML models automatically from source code. The class diagrams in this project will be created using UModel and stored in the .png format.

Part II

Prestudy

Chapter 4

Concepts

A mobile online multiplayer location-based game is based on concepts from many domains. In this chapter we present most of these concepts. The presentation will show what these concepts are, why they are important in our project, and how they will be used.

Figure 4.1 shows how different concepts are related to this project and computer gaming in general. While the client-server architecture and location-based concept provide a technical frame, mobile and multiplayer gaming create a social context to our prototype.

4.1 Mobile gaming

Mobile games can be defined as games played on devices like mobile phones, smartphones, handheld computers, or PDAs. The largest advantage over other platforms is that mobile games can be played anywhere and anytime, due to the fact that people bring their mobile phones with them wherever they go. [MJG]

While computer games in general have existed and evolved since Pong [PON] and Spacewar! [SPW] in the early 60s, mobile gaming is a rather new arena. The first mobile game developed was Snake, which came embedded in the Nokia 6110 back in 1997 [SNA]. This game and those that followed in it's vein were simple and small games played for brief periods of time. Later on, the online aspect was introduced with games based on WAP and SMS. However, it wasn't until the color display and downloadable games emerged in 2001, that wireless gaming opened up to the larger population [PEL].

Today mobile gaming is becoming ever more popular, and according to Gartner Inc. the mobile gaming market tripled from 2004 to 2005, with sales ending at $4.3 billion in 2007 and expected to reach $9.6 billion in 2011 [DQI]. Juniper Research is more optimistic, predicting mobile game sales to grow from $3 billion in 2006 to $10.5 billion in 2009, and surpass annual revenues of $17.6 billion in 2011 and beyond.

The largest contributors to the explosion in mobile gaming seems to be the traditional and simple puzzle games in new and better looking versions, these constituted one third of the total revenue for the first quarter of 2006. But other genres are also on the rise, and 3D multiplayer games have a large following in the Asian-Pacific countries (APAC). According to a Park Associates research carried out in 2006 [PAR] multiplayer games were played weekly by 7% of the total APAC consumers. This interest seem to be driven by the massive success by computer MMOs like World of Warcraft [WOW].

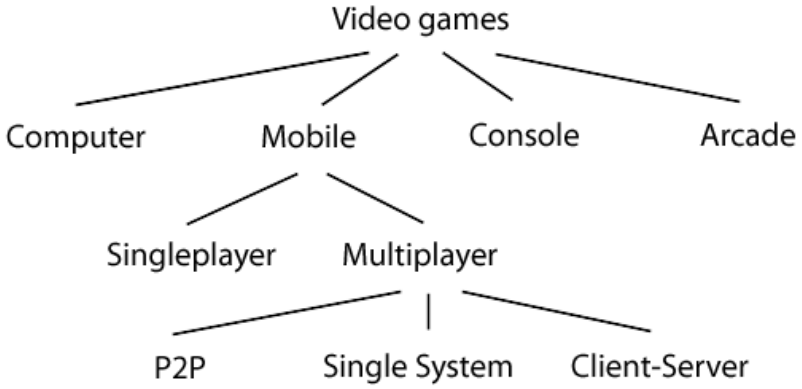

Figure 4.1: Illustration of mobile gaming in a larger gaming context

4.1.1 Multiplayer gaming

The increasing interest in multiplayer games has already been mentioned, in fact a 2005 Ovum Asia Pacific report estimated that mobile multiplayer games contributed 12.5% of the total revenue on the Asian mobile market [CHA].

The social and community aspect of multiplayer games started with simple online high-score charts and simple chat channels in single player games. Later on turn-based games became available and popular, as these didn't require much in terms of bandwidth or latency, which are the two main constraints of multiplayer gaming. Latency affects the delay between action and response in the network, and bandwidth sets a limit to concurrent data transfer between the devices of a network. The latter is affected by the fact that the more players a game has, the more bandwidth is used to relay the actions taken by the players [POW]. As larger network bandwidth has become available, due to newer network technologies, the limit has been raised dramatically, and real-time multiplayer games have now been successfully implemented.

4.1.2 Location-based gaming

In addition to being small and light-weight, a mobile phone is also relatively easy to track, which opens up for new modes of gameplay.

The first location-based mobile game was introduced not long after Nokias aforementioned Snake became popular. Pirates! was developed in Sweden in 1999 [PIR]. This game laid the foundation for the first commercial location-based mobile game, Botfighers, which hit the shelves in 2000 [BOT], see figure 4.2. Botfighters, and its sequels, are search and destroy combat games fought between avatar[1] robots. The first game utilized Short Messaging Service (SMS). Users sent an SMS to see if there were any other botfighters in the area. As each telephone is connected to a GSM antenna with a corresponding CellID, the game server could positively reply if there indeed was another registered telephone in the area. Upon receiving such a confirmation, the player could send another SMS with an attack command and destroy the other robot.

Figure 4.2: Screenshot from the early Botfighters game

Later games have been developed as stand-alone programs with no need for SMS to facilitate the gameplay. Other methods of implementing location have also been utilized, like GPS and WiFi[2]. These games, and the technologies upon which they rely, will be presented further in Chapter 6.

Regarding entertainment as one of many location-based services (LBS), there has been stated that this is what will truly differentiate mobile devices from computers and consoles, as they can never surpass these in terms of audiovisual quality. Games and social networks are at a stage where technology no longer poses the greatest threat, and neither does popular interest. It is merely a matter of creating an enticing concept, or a killer application if you will [LBS][LTB]. Meanwhile, there is an ever increasing population beneath the casual masses that devote time and effort on the alternatives already existing [STE].

[1] A digital representation of a player in a virtual reality
[2] See Chapter 5

4.2 Client-server networking

The game concept we present in Chapter 8 is based on a client-server architecture, which is briefly described in this section.

The client-server architecture pattern describes an architecture that separates a server from clients with graphical user interfaces (GUI). Each client can send requests to one or more connected servers and receive appropriate and processed responses. The server stores central data and sends this to clients which manipulate it according to local input.

The clients do not communicate directly with each other, and all communication goes through the server. If the server goes down, the entire network goes down. Figure 4.3 below shows a simplified illustration of a client-server network consisting of a server and mobile phones as clients.

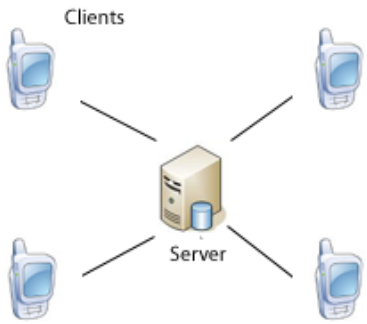

Figure 4.3: Illustration of a client-server network with mobile phones as clients

Chapter 5

Technology

This chapter presents the different technologies that will be used throughout the project and evaluates them with respect to the task at hand.

5.1 Java Platform, Micro Edition

In [ONL] we reviewed the most commonly available mobile programming technologies. Of these, Java ME stood out as the most sensible choice for our project. In the following sections, we will give a deeper presentation of Java ME, based on our earlier findings.

Java Platform, Micro Edition (Java ME) is a collection of specifications and technologies aimed at mobile devices, created and maintained by Sun Microsystems, with the help from external expert groups consisting of leading mobile industry representatives. Any mobile device will only implement a subset of the available mentioned collection of specifications, or Application Programming Interfaces (API). As such, an application developed for one phone may or may not work on the next, depending on the Java ME support implemented by the manufacturer.

5.1.1 Java ME Architecture

An illustration of different Java implementations as well as the different components in each are shown in figure 5.1. Depending on the requirements, different implementations of Java are used, from the very lightweight JavaCard meant for smart cards and embedded devices, all the way up to Java EE for enterprise server solutions.

The Java ME implementation is divided in two, one aimed at the lower end of the mobile phone scale, including most normal handsets, and one for the more powerful devices, like high-end PDA's and tv set-top boxes.

This project focuses on making a game that will be available for the majority of mobile phones today, which makes it natural to base our development on the implementation

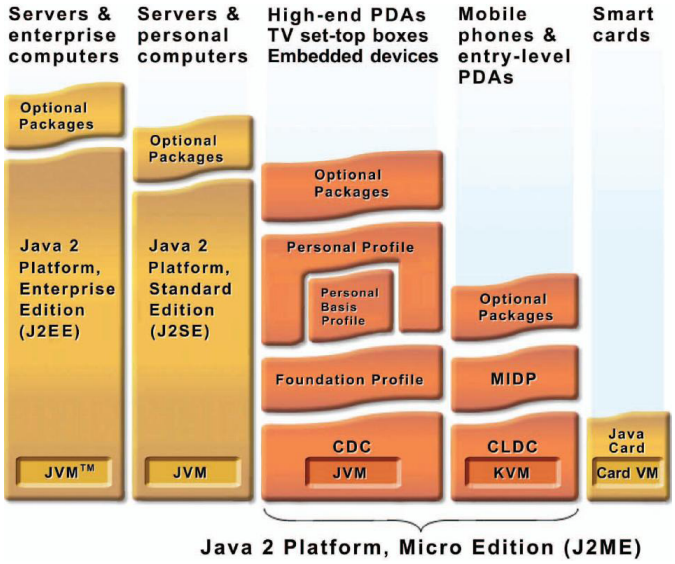

Figure 5.1: An overview of the Java ME Architecture

stated in the right-hand column, namely "Mobile phones and entry level PDAs", called CLDC. A more detailed explanation of the components in this implementation will be presented in the following sections.

Virtual Machine Layer

The Java Virtual Machine (JVM) is the execution engine for Java applications, containing pre-defined machine instructions. Java code is translated into byte code to be exectuted in the JVM, using the instruction set of the operating system on which it currently resides. This is what makes Java a portable language, the platform independency coming from separating source code from OS machine code, and means that any device that implements JVM can run Java applications.

The Kilobyte Virtual Machine (KVM) is a limited version of the JVM, aimed at devices with limited resources [PER]. It contains all central aspects of the Java language, but was built from ground up with no additional functionality in mind in order to work in memory-poor environments. There also exists a CardVM, the virtual machine in the JavaCard implementation. This is simply a further limited version of JVM lacking even threads and garbage collection [ACM].

Configurations

A Java ME configuration defines a minimum set of JVM features and core java class libraries for a range of devices [TOP]. A configuration does not offer optional features, because all Java implementations using the same configuration should behave in the exact same manner.

The configuration specified to run on top of the KVM on low-end Java ME devices is called Connected Limited Device Configuration (CLDC). The configuration for the more powerful devices is called Connected Device Configuration (CDC). These configurations come in backwards compatible revisions, and the current CLDC version is 1.1, which added floating point and weak reference support to version 1.

Profiles

The profile works as an extension on top of the configuration, providing extra APIs for added functionality, as can be seen in figure 5.2. These APIs include networking, user inteface and gaming specific properties like layers and collision detection. In general, a single device can support several profiles, and a profile is meant to create a conform environment for a device family. The profile designed for CLDC is is called Mobile Information Device Profile (MIDP), the current version in MIDP 2.0, which is backwards compatible with MIDP 1.0.

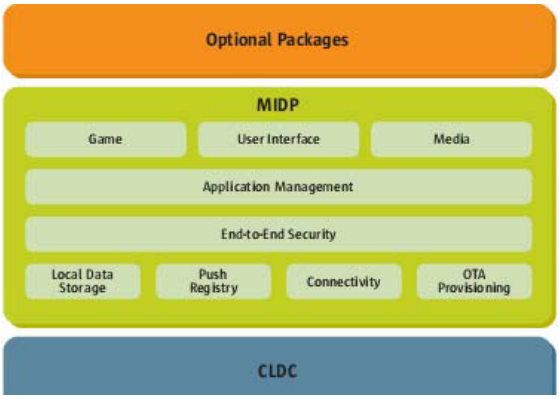

Figure 5.2: An overview of the MIDP Profile components

With the profile application model added to the CLDC, a three-layered runtime environment for applications is complete.

Optional Packages

As you can see in figure 5.2, there is a layer called optional packages on top of the profile. These packages are applicable to a large number of devices, and contain functionality not defined in the Java ME platform minimum requirements specification. These optional packages include, but are not limited to:

- JSR[1] 82 - Bluetooth API for Java ME
- JSR 120 - Wireless Messaging API
- JSR 135 - Mobile Media API
- JSR 179 - Location API for Java ME
- JSR 184 - Mobile 3D Graphics for Java ME

JSR 179 & 293

These additional specifications contain the Java ME location API, version 1.0 and 2.0 respectively. The location API includes a package called javax.microedition.location. This package contains the basic classes needed to request and get a location result. The package supports satellite based methods like GPS, cellular network based methods like GSM and short range positioning methods like Bluetooth Local Positioning. While these methods have many features, the only mandatory ones are:

- Provide latitude and longitude coordinates and their accuracy
- Provide timestamp of the location measurement

JSR 293, which is not yet final [JCP], adds to these features, introducing amongst other things interfaces for accessing location-based services like map, navigation and geocoding.

JSR 248 - Mobile Services Architecture

MSA is a new umbrella standard which aims to reduce fragmentation for the Java community. This is done by defining a set of component JSRs that must be supported on next-generation mobile phones. These JSRs are strictly defined, giving less room for handset developers to interpret the implementation themselves. Regarding this project, the most important component included in the MSA is JSR 179, the location API. The

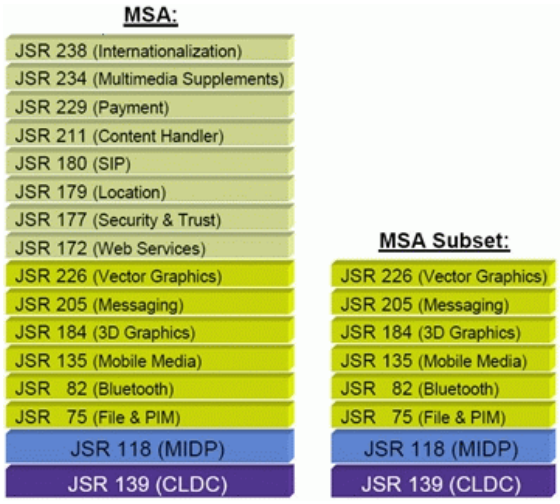

Figure 5.3: An overview of the MSA standard

earlier mentioned JSR 184 is also included. Figure 5.3 shows the entire MSA standard.

The MSA effectively forces the mobile industry to share some common ground, giving developers an easier time in creating applications for a broader group of handsets.

5.2 Mobile phones

While mobile phones are less powerful than their console or computer counterparts, they do have some other features which makes them interesting from a gaming perspective. This section covers these features and shows how they can be useful in a mobile game.

5.2.1 Screen

The most obvious limitation of a mobile phone is the reduced screen size and resolution. While the average size has increased over the years, from the aforementioned Nokia 6610s 128x128 pixels and 1.7" screen to the state of the art iPhone with a 320x480 pixel 3.5" touch screen. However, the iPhone is nearing the maximum physical size consumers are willing to use. Steve Jobs hinted at this himself when claiming that a 3G chip would make the iPhone too big, and so it would have to be shipped without [IPH]. As a side note, as this is being written, iPhone v.2 was just launched. It has both a 3G chip and reduced size, while keeping the same screen.

At any rate, the screen size is important for a mobile game, since it dictates the number of items that are available to the eye at any given time. In addition to this, screen sizes on modern phones vary a lot, with the most common at 176x220 pixels and 320x240 pixels.

[1] JSR stands for Java Specification Request, and is the formal prefix to Java platform additions.

Any game will have to compensate and adapt the playfield to each phone group, making as much information as possible visible, without the items becoming hard to discern.

5.2.2 Vibration

Vibration is a feature most mobile phones today have adopted. While it has traditionally been used for making the user aware of phone calls in very silent or very noisy surroundings, it can also be used to give the user feedback much like the Nintendo Rumble Pack game pads [RUM] or Playstation's Dualshock counterpart [SHO].

In mobile games vibration can be used to simulate chrashes, recoil, explosions and more. However, vibration takes a toll on the battery, which is another limiting factor of mobile gameplay, meaning vibration should be kept at a minimum or have the option of being turned off.

5.2.3 Camera

Mobile phone cameras are a driving factor in their own right, and some phones are marketed mostly for their ability to compete with the traditional compact camera. While the average mobile phone camera not yet is in a position to rival traditional cameras, it has found uses outside MMS[2] and party snapshots. Location-based games like Conqwest use the mobile phone camera to capture semacodes [SEM], a type of barcodes, in city surroundings. The semacodes are given a point value, placed on stickers and hidden like treasures. When someone captures a semacode and uploads the picture to the game server, the point value is given to the players team. The entire game plot can be found on the game homepage [CQW].

5.2.4 Sound

Since 2005 most mobile phones have had support for truetone audio output. Truetone is generally defined as high quality encoded formats like mp3 and aac. While the progress in sound quality have been driven by the ringtone market, most games since the beginning have incorporated some kind of sound in the gameplay. Music has been used for atmosphere and sound effects have been added to crashes, gun fights, death and more.

5.2.5 Keypad

As important as the keyboard and mouse is to a computer game, the keypad is as input source for mobile games. Most mobile phones have a standard 12-key setup with an additional d-pad or joystick for menu manouvering. Other buttons vary greatly from phone to phone, and one generally can not make assumptions about these without excluding some users from playing the game in a satisfying manner. Keeping the input interface intuitive while still maintaining a high degree of control can be difficult. Buttons are

[2] Multimedia Messaging Service, a system allowing multimedia content to be sent via cellular networks.

generally small on mobile phones, and some care must be taken when deciding which ones to use when.

In addition to the aspects covered above, many newer phones come without keys at all, and solely rely on a touch screen interface for input. However, these phones are still few in number and most tend to emulate a 12-key setup anyway.

5.2.6 Internet connection

While data transfer services were available since the start, a mobile internet connection wasn't common until year 2000, when the General Packet Radio Service (GPRS) data service was being implemented in most GSM phones, following a standard release in 1997 [GML]. The higher bandwidth of GPRS allowed use of services such as WAP and MMS. Later on upgrades have been made to both bandwidth and latency of the mobile data transfer services, and today network gaming via a mobile connection is within acceptable terms.

An internet connection not only makes multiplayer gaming available, it also allows games to make us of external web pages and services in the gameplay. Players outside the game can interact via a web portal and the game itself can collect weather information or extra game content from the internet.

5.2.7 Location

Location introduces a whole new aspect to mobile gaming. According to Srivastava [LOC], location can be:

- Absolute position, as achieved in GPS.
- Location relative to a base station.
- Location relative to a starting point.
- An area, where location can be relative to a defined context, like city or municipality.
- Location inside a building.
- Location compared to other people or objects.

When having identified the location of a player, it can become a driving factor, not merely a side-effect of being mobile. It can be utilized as contextual input like often is the case in location-based services (LBS), or a direct input factor in the game, as the keypad is in most games. Meaning that whenever the player moves in the real world, his avatar in the game world changes as well. This can happen according to any of the interpretations of location mentioned above.

The location of a mobile phone and its user can be identified using a number of technologies. The different technologies that are available today provide a varying level of accuracy, power consumption, and time to complete. Generally, a seamless interaction between several technologies would be preferable when locating a device, but not all technologies are supported in every handset. This constitutes a great obstacle for developers of locationbased mobile games and operators offering internet services to handsets [NIC]. The networks that technologies reviewed in this section rely upon can be further studied in Section 5.3.

The most accurate way of getting the location is through use of the Global Positioning System, or **GPS** [GPS]. Some phones come with a GPS module built in, while most other phones can access such a module externally via Bluetooth [BLU]. The GPS is in practise a system consisting of some 24 satellites orbiting the earth, controlled by a number of monitoring stations, and maintained by the US government. In addition comes the mobile receiver modules installed in handsets. The GPS module in the phone trilaterates[3] it's position based on a signal from at least 3 of these satellites. The location can be found with an accuracy of 2-10m all over the planet. The main drawbacks are the need for a GPS module installed in the phone, which both is costly and power intensitive, and the need to be in open air, since the signal is diminished indoors.

Another set of location techniques can be found in the GSM network itself. Via the GSM base stations a handset is connected to, position can be identifyed in many ways. Each base station has a unique identifier, or **CellID**, this number can be used to identify position within the covered area of such a station. These areas often have a radius of 100m or less in urban areas up to several kilometers in rural areas. Adding timing information to this CellID one can achieve a somewhat higher accuracy, at the cost of complexity. Another GSM technology used in locating the phone is triangulation of a signal between several base stations, not unlike the GPS system. This triangulation can be calculated by the phone itself, as in Enhanced Observed Time Difference of Arrival(**E-OTD**) or by the network, as in Uplink Time Difference of Arrival(**U-TDOA**). The latter technique is mostly used, since it has no handset requirements other than a GSM connection. The former technique need a module to be installed in the handset, which effectively removes the advantages of using a GSM approach. These advantages are based on cost and market: All GSM phones can make use of GSM localization, and at no cost to the user. Furthermore, GSM localization works well indoor.

3G or UMTS provides the same means of location as the GSM network, but adds to it. E-OTD is slightly different in UMTS, and is called Observed Time Diference of Arrival (OTDOA) [OTD]. In addition, the UMTS network provides a native system for locating users. This system can be accessed by Parlay API's in the 3G OSA architecture [OSA]. These provide an easier way of fetching location information with no pre-knowledge of

[3]Calculating position based on known distance to two or more refrence points

the current network. However, access to these requires operator support, which comes at a cost.

Lastly, a Wireless Local Area Network (**WLAN**) can be used to locate mobile devices in any area covered by such a network. This can be done in two ways. One way is to have all the WLAN antennas of an area listen to ongoing traffic and then identify the position of the MAC adresses that are found within each antennas area. The other way entails analyzing the signal strength of surrounding WLAN access points. This information can be compared to a database of already collected signal strengths connected to locations. The calibration required to build such a database can be time consuming and complex. WLAN position can be very accurate, down to 0.5 to 5 meters [WLA], and it is suitable for indoor use. However, when covering larger area with one WLAN it can be very costly. In addition, WLAN is supported by very few mobile phones today, these are normally only the most advanced ones.

5.3 Mobile network technologies

This section will focus on the different network technologies available for transfer of data to and from a mobile phone. Since the first commercial GSM networks were opened for service in 1992, these networks and subsequent technology updates have been categorized into generations. These generations are 2G, 3G and the future 4G. Each generation offers new services, new features and upgraded capacity. In addition, each generation typically offers a performance upgrade or a hybrid feature before a new generation is standardized. These upgrades are popularly referred to as 2.5G and 3.5G respectively.

5.3.1 Pre GSM

While mobile telephone services have been around since the radio telephones in the late 1940s (sometimes called 0G) [HIS], the first modern generation of mobile technology was introduced in the 1980s (1G). At this time mobile phones used a different cellular system than GSM, for instance the analogue NMT (Nordic Mobile Telephony) in parts of Europe and Russia [MPG]. It was at this point the idea of dividing geographic regions into cells covered by base stations could successfully be applied. The idea itself had been introduced at the very start. Another aspect of this idea is to reuse frequencies in cells that are not neighbours. These first cells had a radius ranging from 2km to 25km.

5.3.2 2G

The first digital mobile telephone systems in Europe were introduced with GSM (Global Systems for Mobile Communications). Second generation technologies can generally be divided into CDMA-[4] and TDMA-[5] based standards, depending on which type of multiplexing used. GSM is based on TDMA. The main advantages to 2G over 1G are less

[4]Code Division Multiple Access
[5]Time Division Multiple Access

power consumption and increased sound quality. In addition, the hugely popular SMS service was made available with the use of digital second generation systems.

GSM

GSM is the most popular standard for mobile phone networks in the world, and had over 3 billion users by the entrance of 2008 [GSB]. The system is based on cicuit-switching with 200kHz channels split in 8 time slots. Data transfer can reach speeds of 9.6kbit/s. In Europe, GSM occupies the 900MHz and 1800MHz wavebands.

GPRS

GPRS is a mobile data service that uses packet switching. This feature, which it has loaned from 3G services, has given GPRS a description as 2.5G, since it does not meet the speed requirements of 3G. Because of this overlay service, mobile phones with GPRS have normal access to TCP/IP networks and can access the internet through WAP, send MMS or use instant messaging programs. Since GPRS is packet switched, multiple users share the same transmission channel, and only transmit data when needed. Contrary to the old circuit-switched pay-per-minute approach, GPRS is billed per megabyte and is thereby cheaper to use.

While GPRS uses the same modulation, 8 time slots and channel size as GSM, it offers different coding schemes that allow manipulation of these. Uplink and download channels are separately reserved, giving possibility to combine these as needed. The fastest coding scheme, CS-4, has a theoretical throughput of 160 kbit/s using all 8 time slots for transfer and no error correction, but a normal throughput peaks at about 40-60 kbit/s [PRF]. In mobile gaming, latency is as important as transfer rate, and GPRS offers very high latencies. A message going from a server to the client and back can take as much as 1000 milliseconds [GPR].

EDGE

EDGE, or Enhanced Data rates for GSM Evolution, is a digital data transfer technology also labeled as 2.5G or 2.75G. The technology can be used in both packet-switched and circuit-switched services, but does not meet the speed standard of 3G. In a packet-switched network EDGE uses a GPRS technique called Enhanced GPRS, which allows for regular speeds up to 245 kbit/s on average. In this mode, the regular GPRS bandwidth and time slots are utilized, but a more advanced signal that is able to carry 3 times the data per modulated symbol is used, allowing for 3 times the data transfer rate of GPRS [PRF]. An additional advantage is the latency, which reaches end-to-end times

of 150 miliseconds and better.

5.3.3 3G

The third and present generation of mobile networks offers new services, higher capacity and higher security compared to 2G. The largest branch of 3G development is standardized and maintained by the 3rd Generation Partnership Project (3GPP), which strive to make a globally applicable 3G specification building on evolving the GSM system.

One of the new services available is video calls, long predicted to be the "killer app" for 3G phones, these are phones with a front-mounted camera. Time has since proven video calls' failure, and 3G's general upgrade in quality has attracted people instead. Since 2001, when the first 3G service was launched in Japan, the number of 3G providers have spead to service more than 300 million subscribers by the entrance of 2008 [GPB]. The first operator in Norway providing 3G services was Telenor in 2004 [NOR]. The 3G technology deployed here is UMTS.

UMTS

Universal Mobile Telecommunications System (UMTS) is one of several technologies providing the right capacity and services to fit in the 3G specification. UMTS supports speeds up to 384 kbit/s in regular use. However, the theoretical throughput is as much as 14 Mbit/s. The most common form of UMTS transfer interface deployed, Wideband-Code Division Multiple Access (W-CDMA), comprises 72% of all 3G services. In this mode, two 5 MHz channels are used for uplink and downlink. According to 3GPP standard, these channels are found in the 1885-2025 MHz and 2110-2200 MHz bands respectively, but some countries use other frequency bands, like the USA /citeUSU.

UMTS latency reaches end-to-end times as low as 100ms in general. Somewhat faster than EDGE, and far better than GPRS.

HSPA

HSPA, or High Speed uplink/downlink Packet Access, works as an upgrade or addition to the UMTS standard. With the use of HSPA protocols, the download and upload speeds of UMTS are reaching their theoretical maximum. The download speed maximum is 14.4 Mbit/s, but in the first installment, a general peak speed of 3.6 Mbit/s is achieved. Some rollouts reach as much as 7.2 Mbit/s as well, depending on which collection of protocols and modulations are used. Upload speeds are generally 384 kbit/s, with a theoretical maximum of 5.76 Mbit/s. Newer installments in Finland achieve upload speeds of 1.4 Mbit/s [ELI].

HSPA is recognized as a 3.5G technology, due to the massive increase in speed, while not bringing any new service to the field. Not only speed is increased, though, as latency is improved over UMTS reaching the area of 70ms.

5.3.4 WLAN

Although not considered a traditional mobile technology, a WLAN chip is included in an increasing number of advanced handsets. WLAN deployment has increased rapidly over the years, and as more and more companies, organizations and households sets up routers enabling wireless networks, they become more accessible for mobile phone users as well.

WLAN, as we have mostly come to know it, is a set of standards defined by the IEEE[6], in a family called 802.11. The family standards are a, b, g and n[7]. The only difference between them is channel use, range and transfer speed. They all work in the 2.4 and/or 5 GHz spectrum. Data rates in 802.11b are limited to 11 Mbit/s, in b and g to 54 Mbit/s, while the new n standard boasts up to 250 Mbit/s. Indoor range varies from 25m to 70m. Practical use will see somewhat lower speeds and ranges than the stated maximum, though. At any rate, speeds are faster than those of a traditional mobile network, and latency is lower as well. The downside is reduced range and coverage.

5.3.5 Comparison

In a location-based multiplayer network game, there are several crucial factors that depend upon the network technology used.

Let us first consider the location-aspect of the alternative networks. Both GSM and UMTS support the same methods of location. Namely CellID and triangulation. However, the naming convention in base station CellID differs both from operator to operator and from GSM to UMTS, this is left open for operators to decide in the standard. In Norway a 4 cipher hexadecimal code is used for GSM base stations and an 8 cipher hexadecimal code for UMTS. This is examplified in table 5.1 below. The base station positions vary geographically between operators and network technology as well. GSM cells are naturally a bit larger, since they send on a lower frequency. Correspondingly, UMTS stations must stand closer due to the higher transmission frequency. In addition, the actual placement of stations vary between operators. In figure 5.4 you see an overview of how one operator's base stations might be placed, and the relative signal strength in the area.

[6] Institute of Electrical and Electronical Engineers

[7] The n standard has not yet been finally released, and is expected in 2009. However, products that are guaranteed to work with the standard are on the market already

Location	**Network**	**Operator**	**CellID**
St. Olavs Hospital	GSM	Telenor	40b4
	GSM	Netcom	3f00
	UMTS	Telenor	01f6ba31
	UMTS	Netcom	00059316
Nidaros Cathedral	GSM	Telenor	3f45
	GSM	Netcom	931b
	UMTS	Telenor	01f6b9c1
	UMTS	Netcom	0005e532

Table 5.1: Base station CellID

Triangulation by base stations is not a trivial task, and needs to be offered as a service from the operator. These services are also known as Content Provider Access (CPA), and CPA Positioning is only one of them. With CPA positioning, the means of retrieving location is transparent, and GSM, UMTS and WLAN are all supported. Both Telenor and Netcom offers CPA [**?**][CPN], but a license to use this can be costly. In addition, location through the network may take as much as 4 seconds [**?**]. Thus, it is not for any game developer to make use of this technology, both economically and practically.

Network throughput and latency are other factors playing a role in deciding which technology to use. Generally, any real-time game application will have to produce an effective latency of less than 250ms for it not to affect gameplay seriously [RTG]. This means that GPRS without EDGE is out of the question, while any other combination of technologies should suffice. Transfer rates are important to any large package sent via the network. A large package sent with an insufficient bandwidth will take too long to transfer, and perceived latency will be unacceptable.

While gaming via WLAN offers both excellent latency and transfer rates, the location complexity and limited coverage suits a mobile location-based game badly. There are mobile games on the market that make use of WLAN, but they are very few. WLAN is not so widely used to locate the phone as to provide a free connection to the internet for game data transfer. Especially indoor or in urban areas. We will not discuss gaming via WLAN further in this thesis, but if you are interested in the subject, check out Plundr for the Nintendo DS by area/code [ACP]. It was originally a PC game for laptops, but it has been ported to the DS, and a mobile version is not far away. The game is a pirate adventure, where players move from island to island based on Wi-Fi zones. Your position in the physical world affects the game. You'll find different islands, different market prices and different ships to fight based on where you are.

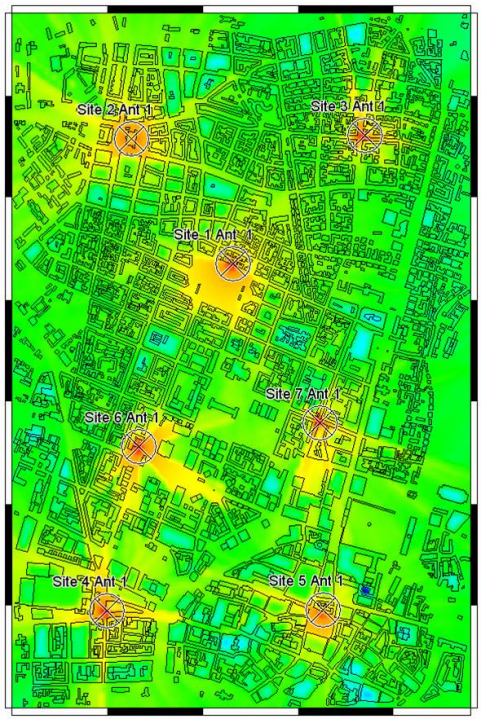

Figure 5.4: Map showing base station coverage in Dachau, Germany [Courtesy of Enorm GMBH].

5.3.6 Test phones required

In this project, we will need phones that support:

- Java ME
- A 320x240 pixel screen resolution
- JSR 118 - MIDP 2.0
- JSR 139 - CLDC 1.1
- JSR 179 - Location API

While emulators have been and can be used to test games without a mobile phone, the nature of our game dictates that we perform real world testing with suitable mobile phones. Currently, very few handsets support the JSR 179 location API. A full list can be found at [LOA]. However, both Nokia and Sony Ericsson(SE) have recently provided their own network API's, which access the JSR 179. This way, CellId can easily be

retrieved by importing the `com.sonyericsson.net.cellid` or `com.nokia.mid.cellid` packages. The method only works on SE phones that support JP-7.3[8] and later [SJP] and Nokia phones with a Series 60 3rd edition FP2 system and later [NSE]. Series 40 3rd edition FP 1 systems can retrieve CellId as well, but require midlets to be signed by manufacturer or operator [NSE].

Our test phones are the Sony Ericsson K850i and W910i, a modern camera phone and a modern multimedia phone, respectively. They both support JP-8

Phone Model	**Features**	**Screen Resolution**
SE K850i	MIPD 2.0, CLDC 1.1, HSDPA, MSA	320x240
SE W910i	MIPD 2.0, CLDC 1.1, HSDPA, MSA	320x240

Table 5.2: Mobile test phones

[8]Sony Ericsson Java Platform - A technical increment of SE Java implementations

Chapter 6

State of the art

In our depth project [ONL], we reviewed the state of the art in mobile location-based games. This chapter will build on our previous analysis, and add new information we have discovered since.

6.1 Location-based games

As mentioned in Chapter 4 location-based mobile games have existed since the turn of the century. Since that time phones have developed both in networking abilites, processing power and display options. In the depth project we found that the large majority of the 40 reviewed location-based mobile games used either CellID or GPS as the means of locating players (>70 %). Furthermore, most of them were developed in Java ME (>50 %). This seems natural due to the larger market of these approaches versus the alternatives.

In the following sections we will present three location-based games that are active today. These are all multiplayer games, incorporating location in different fashions.

6.1.1 The Target

The target is a game developed and maintained by the Belgian company La Mosca, and first released in May 2007 [TAR]. The game is described as a pursuit game. The plot is based on a gangster escaping from prison, and the policemen sent after him. These policemen will have to catch the gangster before he commits too many crimes and leaves the city. The gangster must commit crimes and steal money in order to leave the city. Crimes are commited by stealing objects virtually spread around the city. However, when a crime is commited, the policemen are informed, and can follow the trail. These are the player roles, and each game is thus limited to at least 3 persons playing in a known environment. However, up to 36 players can play the game at the same time, with groups of policemen chasing a group of gangsters. In addition, several games can be played simultaneously, allowing for large groups.

Figure 6.1: Screenshot displaying an overview map from The Target.

Technology and business

Groups of players wishing to partake can rent a timeslot of 2 hours where they are free to play. Distribution is done at game centers, places where people come to play video games, mostly online [LAM]. The game is developed in Java ME, and the midlets were originally preloaded on Nokia 6110 GPS phones. The phones were mostly sponsored by Nokia, wishing to promote themselves in this segment. Now the game is played on Nokia 3250 with Globalsat BT 338 external bluetooth GPS receivers. Some game centers also invest in their own phones and buy game licenses from La Mosca. At the moment the game is based on a few cities in Holland and Belgium, but any expansion of the game is done by porting the plot to a different base map, a fairly trivial transfer [LAM].

The typical players are groups of friends wishing to play for a few hours and event agencies giving companies an alternative team building option [TAR]. As many as 130 players have been taking part in a session at a time, with several games going on at the same time. Location is updated by GPS, and as with most multiplayer games, only actions and location is transferred via the network, meaning each game has an average device transfer of 500Kb [LAM]. This traffic is paid for in the game session.

6.1.2 Mogi - item hunt

Mogi is a treasure hunting game released by France-based Newt Games to the Japanese market in 2003 [MOG]. Players are supposed to follow a virtual map over Tokyo, move through the real-world city, talk to other people in game and pick up virtual items. Items are then traded with other players in order to complete item collections. Items are not

limited to sticks and stones, but may be animals as well. Some items and animals may only be available at night. The goal is to get items and complete item collections in order to get points. By getting points you increase in rank and fame. The game comes with an instant messaging system and a system to locate friends in the city.

Figure 6.2: Screenshots displaying different scenes from Mogi. Top right: the radar.

Technology and business

The game is written in Java ME, released via the carrier KDDI, and designed to be played on their AU phones which have GPS receivers [KDD]. The operator charges 315 Yen per month, or about $3. You get access to both a mobile and web-browser client which will allow you to achieve them same results, only differing in interface. Mogi uses both CellID and GPS as means of locating the player. When in CellID mode, one can not pin-point items, and will have to rely a little on luck to get a special item one notices on the radar, since you have to be within 400m to pick it up. However, random items nearby are shown and ripe for pickup. Urban mobile base stations will also stand close enough for players to reach most items. The creator of the game intended CellID mode for when rushing through the map, just checking the radar for interesting item or players [MOS].

6.1.3 The Shroud - Harvesting

The Shroud is a location based RPG for mobile phones, developed by Hardcode 3D Wireless [SHR]. First previewed in spring 2006, this Zelda-like game lets you explore parts of the game based on real world locations through the GPS in your phone. The gameplay is largely single player fighting and gathering. You play the hero Taro in different modes as he harvests from his farm, solves puzzles, mines for gems and fights monsters that are pouring through from a parallell world called The Shroud. In order to pay for his

fighting expenses he will have to farm to make money.

Figure 6.3: Screenshots displaying everyday activities from The Shroud.

Technology and business

The game's social aspect is nurtured via an online community where players can compare statistics and share tips. In this online "International Farmers League" (IFL) players are also ranked, meaning you can improve your character and place higher in the ranking. Through the IFL a player can also take out a timed challenge where he for instance may have to plant 10 potatoes in an hour, or fish 10 fish within a similar time limit. When successfully completing a challenge, points will be added to the player's rank. The highest ranking player will be awarded a prize each month. These prizes could vary from in-game currency to real-life snowboards [SMO].

While the main parts of the game are single-player fighting and gathering, the incorporation of GPS location means that you at certain times can be given in-game challenges based on your real world location. More specifically, you may at one point be challenged to go somewhere in the real world, a so called hotspot or "breach", and upon arriving there a special challenge will become available in the game. These spots may be meters away at first, but later on they can appear further away from the player, even at set places like the Empire State Building [SMO]. The challenges themselves may consist of a fishing competition or fighting a monster. In addition, special events happen at certain days of the game year, like an animal race taking place on July 7 [ANI].

The game is developed in both Java ME and BREW [BRE]. BREW, or Binary Runtime Environment for Wireless, is Qualcomms proprietary counterpart of Java ME's KVM . Games and applications for BREW are written in C and C++. The game has publishing agreements via Your World Games (YWG) and Sony BMG Music. Since it is not yet publicly released, pricing and transfer rates are unknown. Games bought through Sony BMGs gaming portal typically cost $5.99 [BMG]. Statements made by a representative from YWG made it clear that the game is finished, and will ship with episodic content, in the form on quests. At release the number of quests are 13, but the infrastructure is

reported to support 50-100 new quests in the form of downloadable content. This strategy has been used previously by games such as Final Fantasy VII:Before crisis [FFA] and Era of Eidolon [ERA]. In FFVII a montly subscription is needed to unlock new quests and storylines, while EoE sells each new content package at a fixed price.

6.1.4 Summary

In this chapter we have reviewed three mobile multiplayer games which all incorporate location to some degree. While the two first games use location as a more direct part of player input, the last game offers other options of gameplay as well. Common to all the games is the support for GPS, but only The Target actually requires it to play.

All games have a sense of roleplay to them, but The Target takes a more action-oriented approach and Mogi is more about socializing and collecting, common to puzzle games. Only The Shroud can be counted as real roleplaying game among these three.

As for networking, some games are more traffic-heavy than others. The Target has significant real-time update requirements, and will therefore have a larger network traffic than the others. In Mogi, network traffic is only required when scanning with the radar, picking up items, trading or changing map zones. These actions have a lower frequency than updates required for real-time play. In The Shroud network acticity will only be required when accessing the International Farmers League for challenges, chat and comparison, not counting set events where GPS location plays a part.

One of the fundamental advantages of multiplayer games is the social aspect. All games incorporate this in some way or other. In the first game, direct contact between players is a vital part, and the time before and after play is often used for discussing the event [LAM]. Then again, the entire business model of the game entails people socializing and competing in a direct and relaxed context. In Mogi, socializing takes a different approach, where competition is balanced with friendly trade and communication. The Shroud has a less obvious social context, but introduces leaderboards and a community portal for sharing tips and secrets to give the feeling of multiplayer gaming. The location-based events also hopes to bring players together at certain places and times. In addition, extra content is made available for download, giving the sense of a living and expanding community.

Chapter 7

Game playtesting

Any game intended for actual use must think of playability[1] as a factor equal to or greater than the technical factors. Since mobile games have developed so rapidly the last few years, following in the wake of handset manufacturers' progress, mobile players have high demands for game playability [IMP].

While playability has been a slightly overlooked part of mobile game development so far, it is essential for some game developers. Multiplayer games in general, and MMORPG's in particular have a high level of playtesting before being shipped to the masses. In short, one can say that the level of playtesting is proportional with the time and money invested in a game. This is why mobile games developers overlook playtesting; short time tables and tight budgets [IMP].

In this chapter we will look at some important playtesting aspects and how they can be used to improve and stimulate the different stages of mobile game development process.

7.1 Playtesting candidates

Different people have different areas of interest and expertise. Using the right kind of people at the right stages of the game development process is vital to get useful feedback. In [IMP], Ollila mentions some tester categories along with their properties, these are presented in table refplaytesters below. In our project, the expert roles may be filled by our supervisors, colleagues are fellow computer science students and the real players are the natural target group for an online multiplayer location-based game.

[1] The fun-factor

Testers	Advantages	Disadvantages
Experts	Know their focus area(games, usability)	Do not typically represent the target group, they are more experienced gamers.
Colleagues	Easy to recruit, no problems with confidentiality	Do not often represent the target group well
Real players	Match the target group well. Provide realistic data on both functionality and content	Confidentiality may be an issue. Recruiting takes time. Need to be rewarded.

Table 7.1: Playtesters

7.2 Game development stages

Any software project has many stages of development. In games, the stages can mostly be condensed to the ones in figure 7.1 below. Our project will only include the ideas-, game design-, implementation- and alpha stages. The alpha stage means that all features of a game are in place, but lacking final content. Beta means both features and content are in place, and only final tweaking and bugtesting remains.

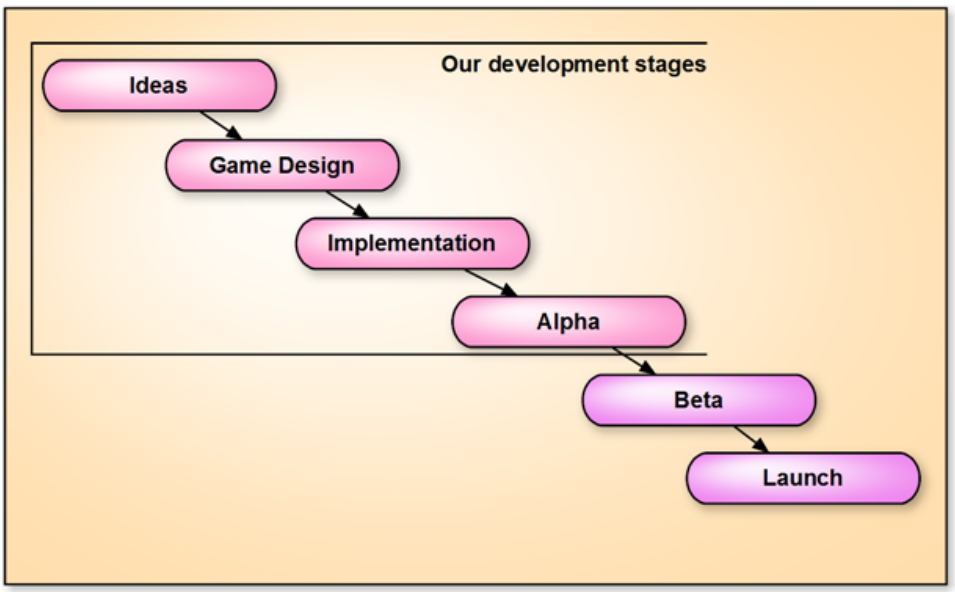

Figure 7.1: Overview of game development stages

7.3 Our playtesting

During the different stages we will take different opinions into account, and use different settings to achieve results. Table 7.2 below, shows a plan of the who, how, what and when.

At the early stages, concept and idea discussion is important. The process of fleshing out desirable and practical solutions to technical challenges will benefit from expert opinions. At the implementation stage, when the concept is set, fellow students and colleagues may provide a fresh look and point out challenges with early versions. Having the developers present and directing the players through a game session will help putting the focus on the parts that need to be tested, skipping unfinished parts of the game. When a prototype is finished, real players need to be taken in to test the solutions. This can happen both in the development environment or, preferably, in an actual gaming environment. Since our game is location-based, which means the gaming environment is outside and affected by external events not reproducable in a development setting, field testing is vital for both playtesting and debugging.

Testers	**Stage**	**Setting**
Experts	Idea & Game Design	Focus groups
Colleagues	Implementation	Direct Observation
Real players	Alpha	Direct Observation and Field Testing

Table 7.2: Project playtesting

Part III

Own contribution

Chapter 8

A Prototype game

Based on the research into location-based gaming, we introduce a mobile multiplayer game using CellID. The concept is based on work done in the depth project [ONL], and is meant to provide a development setting where we can apply concrete ideas for a location-based game. See Appendix A for more details on GSM/UMTS base station cells and CellId.

8.1 CityZombie - The concept

CityZombie is a multiplayer game set in urban areas. The players of the game are split into factions of humans and zombies. The goal of the game is to take control of the city zones and thereby generate "Fear" or "Hope", depending on your faction. Zones are the various important parts of the town, divided into real world CellID areas. We envision one game per city, as each city will have to be mapped and recorded into a database of zones. Each game can have 2, 20 or even 200 zones of a city activated, the concept stays the same. The zones are represented in-game via a map and a color showing who controls the zone, if any. The first player to register a CellID by walking into it will seize control of that game zone for his faction. The zone will stay that way until a rival player attacks the zone and declare the zone taken over by the warring faction.

The map also shows in which zones the other players are located, both allied and enemy, and gameplay can be enhanced by items of power. These items can be mines that secure a spot, radar information telling you where the rival players are (if they are not visible by default), attack robots that grant you the possibility of testing a zone for mines and thus rendering a mine useless etc. These items could be scattered across the map, or randomly placed in zones, available for the first zone controller.

The game is played as an open world where players come and go as they please, never having to wait for a game session to start. If there are no other players the first player starts the game. These individual game sessions last until one faction has achieved a set limit of fear or hope. After each game a new session can be started, keeping the game

pace high. Each player is rewarded points for beating an enemy player and winning the game session for his or her faction. These points are kept after a session is over and maintained in a high score list.

In alternative settings the CityZombie concept could be a single player game where the goal is for the last human has to take control of the city versus NPC[1] zombies or it could be a player zombie wreaking as much havoc as possible within a time limit. As a multiplayer game the goals could be relative team dominance within a time limit, or the initially intended faction favour gathering setting, where one team must hold control of majority of the city for a certain amount of time in order to gather enough favour to win. These modes could be set in the client before game starts.

The zone takeover sequence is played in several ways. If a player attacks a zone which is merely controlled, and not defended by a rival player, takeover is a trivial change of flag color and status update. If other players are present, a more elaborate battle is fought. Both synchronous and asynchronous battles are usable. However, due to latency and traffic in the network synchronous battles are more complex and costly. In our implementation the server gathers information on both attacker and defender, simulates a fight between them and sends the results back to the players. The clients then show the battle round for round in a turn-based manner.

Zones can also be defended with a mine. A player walking on a mine is put out of battle for a certain amount of time, not capturing the zone but destroying the mine. The same thing happens to any player who loses a fighting sequence. By being put out one will have to leave the zone and enter a neutral revival zone in order to be able to head back into action again. This gives the defending players time to rush to zone defence before the attacker "regains counciousness" and can attack the zone again.

8.1.1 Gameplay overview

The game is based on a set number of zones that can be attacked and defended. This is a general model that can be laid on top of any set of CellID zones, and is why CellID is used in the first place. Without a pre-defined zone structure, the game would have to rely on fixed spots like a capture the flag-game, and such a game would require high-precision targeting like GPS. The CityZombie game might not be appropriate for some city environments if GPS was used, though, because of the urban canyon problem.[2] Besides, the GPS mobile phone market is still too small to attract casual players to the game, unless a strategy similar to The Target is used, as discussed in Section 6.1.1 At any rate, the zone model opens for a game that can be geographically linked to any real-world city, as

[1] Non Player Character
[2] Tall buildings or indoor environments block/obstruct the signal from satellites.

in our prototype.

In CityZombie, virtual buildings are placed on the map according to their type and position in the real world, which would make a logical sense of direction for the players. The zones would not have to be neighbours, and the entire city could be used, either making room for more zones, or making the area between the zones larger.

Figure 8.1 on the next page shows an early illustration of a game set in downtown Trondheim. Some zones are free and up for claim, while others have been seized by a faction. As mentioned above, virtual buildings are placed on the map according to the location in Trondheim. The zones are from top left to bottom right: The railway station, the tramway Graakallbanen, the fire station, the arena Nidarøhallen, the Nidaros Cathedral, the Kristiansten Fort, the Student's Society Samfundet, St. Olavs Hospital and NTNU Gløshaugen.

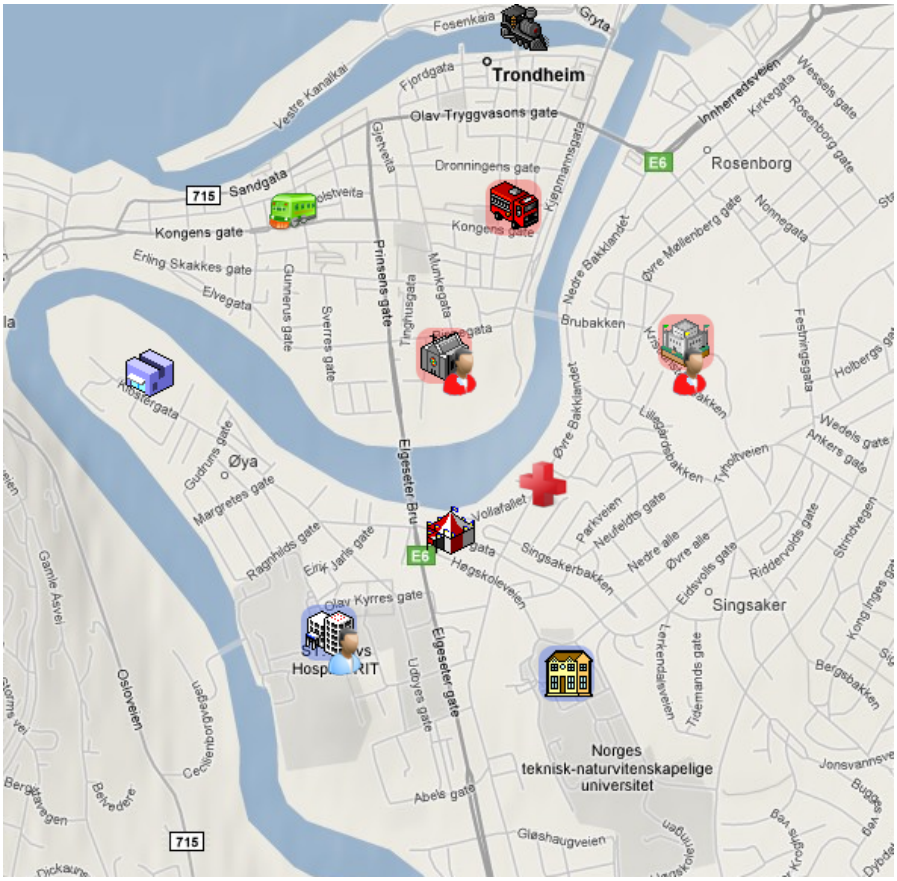

Figure 8.1: The game zones in a game with 1 human player and 2 zombie players

Players who join an ongoing game will likely be scattered out over these zones based on each client's CellID, along with the players already in the fray. In order for a team not to get too much of a head start, each player will have to enter a neutral revival zone before heading into battle.

A smaller version of the early game map, more in line with the dimensions of a mobile phone screen can be seen in figure 8.2, below.

Figure 8.2: The game zones in a smaller game with 1 human player and 1 zombie player

8.2 Server and Client concurrency

As in any other networked multiplayer game, information is not instantly forwarded from client to client. As one client updates his information, this information is being sent to a server, which in turn forwards this information to the other clients. This gives the server the most complete picture of the situation at any given time. While this aspect is vital in any real-time application with strong synchronization demands, our game updates more slowly due to the limited input options and large distances involved in the gameplay, thus largely camouflaging any latency problems in the network and issues rising from those.

This does not mean that clients and server are equal in our prototype. Some information is better processed in the clients and some information is better processed in the server.

8.2.1 Client calculations

Mobile clients are resource constrained and many, meaning that as few operations as possible should be placed here.

Position

Each device in the game contionously checks its position according to a game speed factor and relates this position to the recorded zones of the game. If there was no difference in position from tick to tick, no position update is sent to the server. However, if the player enters a new zone, the client will detect this and the position will be forwarded to the server.

Battle initiation

Battles are initiated by the clients. In addition to listening for positional updates, the clients also checks for player collisions. Meaning that if an enemy avatar is situated in the zone you just entered, your client will send an attack request to the server with both yours and the enemys id.

8.2.2 Server calculations

The server has more computing power as well as acting as a synchronization point, meaning it should take on any operation suitable.

Battle calculation

After receiving a battle request from a client, the server will ask both attacker and defender to respond to this request by submitting their battle tactics. When receiving this information, the server simluates a battle, hit by hit, and sends the resulting information back to the clients involved. This simulated calculation takes player strength and tactics into account and lets it affect a randomization of hits. Meaning that an inferior player could win a battle, but it would only be against all odds. The winning player is also granted a modest hit point increase for that session, making him tougher to beat.

Favour generation

The server keeps its own model of the game zones and whether or not anyone has claimed them. For each zone a faction claims, a larger amount of favour is granted that faction. The server has its own internal game clock, and each time this ticks, the server checks each zone and hands out support according to the owner. This information is then sent to each player.

Winning criteria

As the server clock ticks, more and more support will be handed out, until one faction has achieved the support limit, be that hope or fear. As the limit is reached, a notification is sent to each player informing that the game session is over and declaring who won. A new session can then start.

Player score

Similar to the zone model, the server also keeps a record of the individual players, awarding them points whenever they win a fight, captures a zone or wins the session. Each client is sent his or her points at regular intervals, but only the server manipulates these points in any way. Clients keep a high score list.

8.3 Game framework

The implementation of our game prototype will be based on the framework peer2gaMe, developed by Martin Jarrett and Eivind Sorteberg in 2007 [RTG]. In their master thesis "Real-Time Online Multiplayer Mobile Games" they presented a multiplayer game framework consisting of a client and a server. This framework provides the means of creating and joining game sessions, polling position, collision detection and forwarding different messages to other clients through the server. It will be further presented in the following chapters.

Chapter 9

Requirements - The user stories

In Scrum, requirements are not dictated beforehand or received in its complete form from any customer. Instead, requirements can be derived from User Stories, which are scenarios describing use of practical game features from a user perspective. After collecting a number of user stories, one may identify requirements to be set up in requirement tables. These game requirements can be divided into functional and non-functional requirements. While the functional requirements address the functionality, data processing and general behaviour of the system, the non-functional requirements address the constraints of the game in terms of design and implementation.

This chapter presents some high level user stories gathered for our game, and summarizes these in tables, both for client and server.

9.1 User stories

Our game is a simple application, with a very homogenous user base. All users will see the game from the player perspective, and provide stories accordingly. High level user stories are sometimes called Epic stories [USR], and can be broken down into several smaller stories and requirements. The following stories are the ones used to create all tasks and requirements for our game.

Epic stories

- As a player, I want to capture zones, so that I may win.
- As a player, I want to choose which server to connect to, so that I am able to play the right version.
- As a player, I want to achieve kill points, so that I can top the high score list.
- As an administrator, I want to configure the game, so that I can set it up according to my own specification.

9.2 Client requirements

The client application runs on mobile phones and communicates with our game server using the telenetwork. In this section we present the requirements for our game client based on the described functionality in the concept part of Chapter 8. The game client is based on an existing framework, peer2gaMe [RTG] as described in Section 8.3, and some of the requirements material here is based on and will be fulfilled by that framework.

Our game client has three main state categories, as shown in figure 9.1. The off-line part of the client before connecting to the server is regarded as the pre-game state. The lobby players are presented with before and after an actual game is called the Connected-/Post-game state. And finally, the in-game state category covers all states within the actual game.

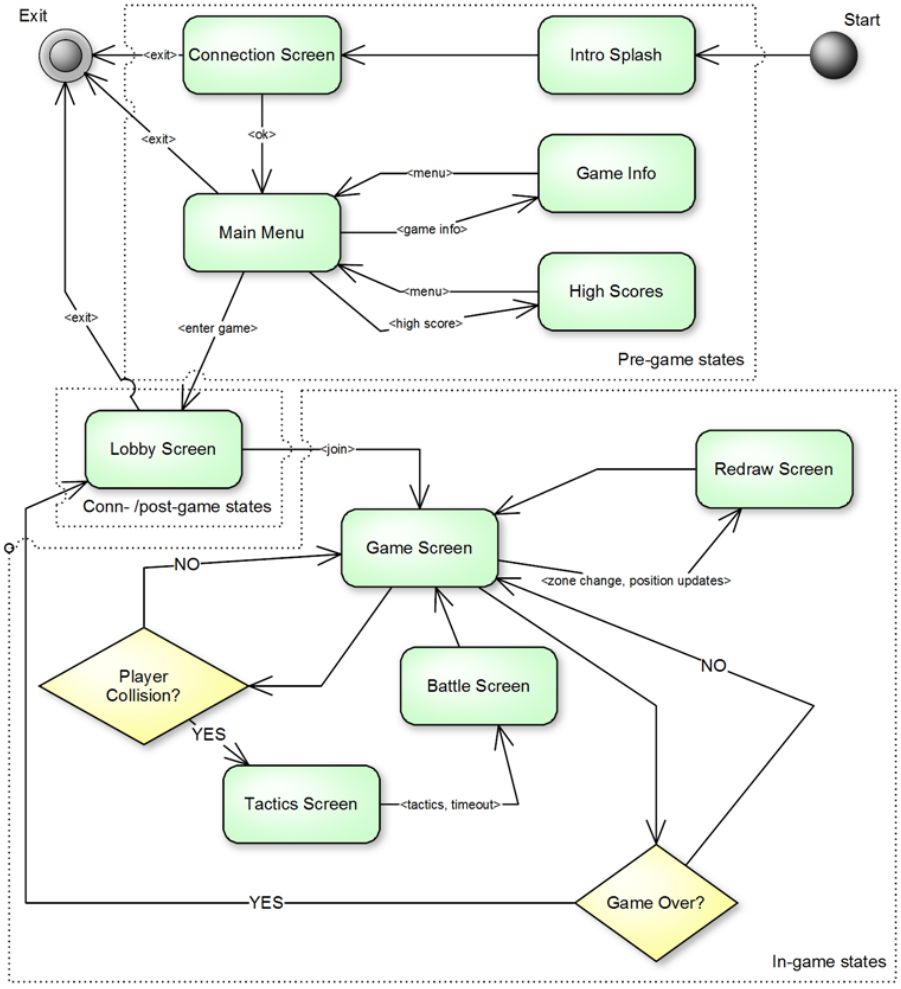

Figure 9.1: Overview of client states

9.2.1 Functional requirements

This section states the functional requirements for the CityZombie game client.

Pre-game requirements

These are requirements applying to the client before joining a game of CityZombies.

- **FR-C1 - The player shall be able to provide his/her own name, limited to 12 characters, no numbers** Choosing one's name is important in any game, as this will be listed among the high scores.
- **FR-C2 - The player shall be able to choose faction before he/she joins the game** The only choices are Zombies and Humans. This is unique for every game, and cannot be switched between sessions, unless the player restarts the game client.
- **FR-C3 - The player shall be able to enter server IP and port number upon starting the game** Allows the player to enter the adress to the server of choice. The client will provide a default setting for server and IP, stored on the phone. These defaults will adapt as the player enters new values.
- **FR-C4 - The client shall provide a main menu with information on how to play the game** The main menu screen of the game gives the player an option called "Game Info", here the player can get some insight regarding the story and actual gameplay before joining any session.
- **FR-C5 - The client shall provide a main menu with a high score screen** The main menu screen of the game gives the player an option called "High Score", here the player can view the names of the top killers in the game and how many battles they have won. High scores are only recorded during a continual game. New day sessions carry over the score, but exiting the client or disconnecting from a game will store your score.
- **FR-C6 - The client shall not initiate communication with the server before the player chooses to join a game** All actions taken up to and including the main menu shall not require or intitiate a connection to the specified game server.
- **FR-C7 - The player shall be provided the ability to exit at any give point** Every game screen will provide the player with the option of exiting the client.

Connecting to game requirements

These are requirements applying to the client during and after connection a game of

CityZombies.

- **FR-C8 - The client shall connect to a server when pushing "Enter Game" in the main menu** Most client will ask for confirmation to use the network. This is implemented differently by most manufacturers, but always gives some option of yes and no. This requirement adresses what happens after pressing yes.
- **FR-C9 - The client shall refresh the session list automatically if the game server does not provide it on the first try** If the client does not get any response from the specified server at first, it should automatically try to reconnect, as this will negate some instances of false negatives. After one automatic refresh, the client will provide the player with options to manually refresh or exit.
- **FR-C10 - Any player shall be able to create a game session if one does not already exist** If after connecting to the server an empty session list is provided, a new session will automatically be created.
- **FR-C11 - The client shall enter a game session automatically if one is found** If after connecting to the server a session list with a valid and open session is provided, the player will automatically join the open session.
- **FR-C12 - If a game session does exist, the player shall not be able to create a new one** Though the framework supports creating new sessions, our implementation should not provide the players with the option of creating several sessions at a time. Any open session will be joined automatically as specified in FR-C7.
- **FR-C13 - The client lobby shall present all connected players in the lobby** The lobby view shall provide a list of all connected players, regardless if they are inside a game or just loitering in the lobby.
- **FR-C14 - The client lobby shall list all connected players with kill scores and according to their team** The lobby view shall provide a list of all connected players sorted by faction. Each faction is listed with a total kill score followed by the names of their individual players and their individual scores.
- **FR-C15 - The client lobby shall automatically update reflecting session changes** The lobby view shall change as new players connect or old players disconnect. It shall also present updates scores at all times.

In-game requirements

These are requirements applying to the client after entering a game session of CityZombies.

- **FR-C16 - The client shall provide the player with an overview map** This is the main game screen showing a town map, individual players, zones and ownership.
- **FR-C17 - The player shall be able to enter and return from the lobby to view connected players during the game** Through commands in the interface the player shall be provided with the option of viewing the lobby to check all player's names and scores.
- **FR-C18 - The client shall check for positional change at regular intervals** According to a specified interval, the game client will poll the phone for it's position and move the player accordingly.
- **FR-C19 - The player shall have to move into a dedicated neutral zone before being able to attack any zone** As players can enter the game from anywhere in the city, including game zones, they will first have to move into a dedicated neutral zone before being able to move to and attack the other players. This is to prevent annoying surpises for the established players.
- **R-C20 - The client shall automatically inititate an attack on a zone when the player enters its vicinity** When the client registers that the player has moved into a particular game zone, a takeover attempt will automatically be initiated. If no enemy is present, the zone will simply change ownership according to the attacking player.
- **FR-C21 - The client shall check for enemy players at each zone change** After moving into a zone, the client checks for collisions with enemy players. If an enemy is present, a battle initiation will be sent to the server.
- **FR-C22 - A player shall automatically defend a zone upon it being attacked by an enemy, if presently in the zone** The battle initiation sent from the attacker should be relayed to the defending player and he informed in the same way the attacker is.
- **FR-C23 - The players shall be prompted to send their battle tactics prior to any battle** After sending and receiving information of a battle, both defending and attacking player should be given the ability to send their battle tactics to the server before receiving the simulated battle report.
- **FR-C24 - The client shall present received battle reports as an ongoing battle in a dedicated window** After receiving a complete battle report from the server, each client shall present that information to the user in a turn based manner, making it look like the battle is being fought as the player watches the screen.
- **FR-C25 - After a battle, the client shall take action corresponding to the outcome of the battle report** After receiving a complete battle report from

the server and showing it to the player, the client shall treat the player according to the result; notifying the server of the kill if positive outcome and rendering the player neutral and uncouncious if negative outcome.

- **FR-C26 - After having lost a battle, the player shall have to move into the dedicated neutral zone to resurrect his avatar** In order to provide losers with some form of punishment as well as facilitating a good game flow any dead player must return to the spawn zone to be resurrected, just like a new player joining the game.
- **FR-C27 - The client shall update the game board every 1500ms** The main game loop should be checked and the game board updated accordingly every 1500ms.
- **FR-C28 - The client shall attempt to send the player's position every 350ms and after every fight** The player's position is sent to the server at the specified interval and after every fight, to reflect any changes in position that may have occured at defeat.
- **FR-C29 - The client shall only send the player position if there has been registered a change in position** In order to avoid unnecessary network use, the client should not send a position update if there has been no change in position since last poll.
- **FR-C30 - The client shall provide the player with an optional status information window during the game** A status window displaying information on score, location and support should be visible at the top of the game screen if the player wishes it. A simple command should be provided to set it visible or not.

Post-game requirements

These are requirements applying to the client after finishing a game session of City-Zombies.

- **FR-C31 - The client shall take the players back to the lobby view when a game is over** After the support limit has been reached, all players should be notified and taken back to the lobby view.
- **FR-C32 - The client shall provide the players with a restricted lobby view when a game is over** The only options available to the player during the post-game lobby is to start/join a new game day session or to exit the game.

9.2.2 Non-functional requirements

In the non-functional requirement categories, we would emphasise usability, availability and modifiability for our game clients. These categories are explained and rationalized

below.

Usability

These are the non-functional requirements applying to the usability aspects of the City-Zombies game client. Usability addresses the ease with which the player carries out a task and the level of help given by the system in connection with said task. The threshold of understanding any game should be low, in order to appeal to all levels of patience.

- **NFR-C1 - The game mechanics and goals shall be understandable to the player** Left on his/her own device, the player shall be able to understand the goals and mechanics of the game within 10 minutes.
- **NFR-C2 - The client should provide ample feedback to the player** The game should make every action the user takes known and clear to him/her through vibration, sound and visuals.

Availability

These are the non-functional requirements applying to the availability aspects of the CityZombies game client. Availability is about system stability and failure and what the consequence of such failure is. A high availability demands a system which is very stable and/or handles instability well. Players will want a game that does not crash on them.

- **NFR-C3 - The game shall hide any failure from the user, and provide means of carrying on playing** Any failure caused by internal or external events should be kept silent as long as it does not directly affect player progress. If it does, the system should present the failure in a legible manner and provide the user with options to fix/pass it.

Modifiability

These are the non-functional requirements applying to the modifiability aspects of the CityZombies game client. Modifiability concerns any modification or addition to the system and it's associated cost. A focus on modifiability means that developers should easily be able to both change values and the setup of the present game as well as change functionality in order to alter the gameplay into a different game. We wish to implement our game client while keeping the modifiable structure of the client framework intact.

- **NFR-C4 - Important game values shall be kept in a easily reachable place** If a developer wishes to alter game update speed, message send interval, cell phone manufacturer, network operator, player strengths and other values, these should be openly accessible at the beginning of relevant classes.

- **NFR-C5 - The game shall support alternative means of identifying location** If a developer wishes to use map and use GSM CellID values, triangulation or even GPS, only the location specific part of the client game class needs to be altered.

9.3 Server requirements

In this section we present the requirements for our game server as outlined in the concept part of Chapter 8. The game server is based on an existing framework, as given in Sectionframew, and some of the requirements material here is based on that framework.

As the server is dormant before a connection is requested, only two state categories apply to the server. The pre-game state covers listening for connections and waiting for session creation. The in-game state covers the server lobby which generates support, checks whether the support limit has been reached and allows for change in support limit. After a session is won, the server closes that session, and waits for a new session creation.

9.3.1 Functional requirements

This section states the functional requirements for the CityZombie game server.

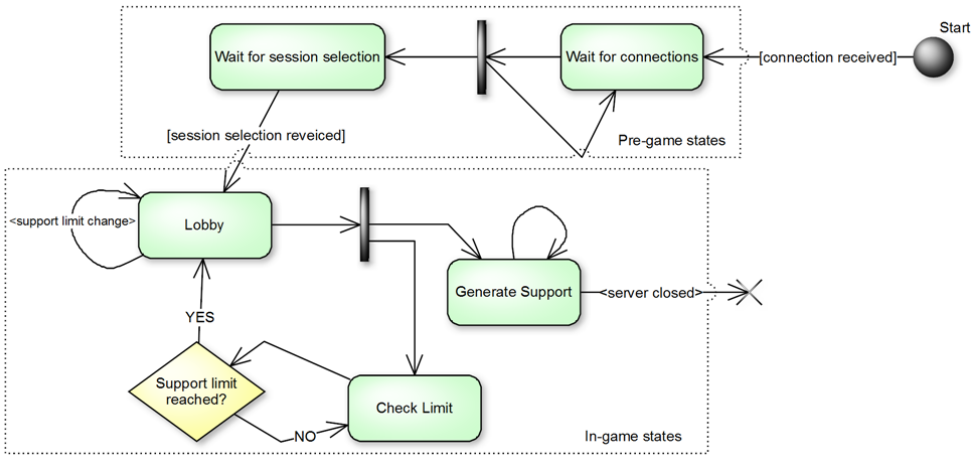

Figure 9.2: Overview of server states

General requirements

These are general requirements applying to the CityZombies server.

- **FR-S1 - The server shall be implemented in Java SE** In order for the server to be runnable on all computer systems, Java SE is used to create the server. Both client and server are created in Java, which makes reuse of code and functionality possible.

- **FR-S2 - The server shall be able to run on any computer with Internet** An internet connection is vital to the communication between server and client.

Pre-game requirements

These are general requirements applying to the interface before joining a game of City-Zombies.

- **FR-S3 - The server shall allow players to join ongoing sessions at any time** As these game sessions can take quite some time, players must be allowed to enter the game at their own desire instead of being forced to wait.

- **FR-S4 - The server shall send alive requests at regular intervals of 7000ms** As players may disconnect from various issues, including phone signal and battery loss, the server needs to keep track of which players that are active and which that are not.

- **FR-S5 - Any client not responding to an alive request shall be removed from the game session** If a player does not respond to an alive request, they are removed from the game. This is done in order to ensure that the active players are presented with an accurate view of the game.

- **FR-S6 - If several players are present in the lobby when a game session starts, start game commands are sent to all players** In order to avoid a situation where several connected players start a game at the same time, a start game command is forwarded to all players, sending them into the game screen. This also sends the server to its in-game state.

In-game requirements

These are requirements applying to the server during a game of CityZombies.

- **FR-S7 - All game zones shall be set to neutral upon game start** Every new session must present the players with a clean map where all zones are neutral.

- **R-S8 - Any client that joins an ongoing game shall be sent an updated zone overview** Players joining a game mid-way will have to receive an update on which zones belong to which team.

- **FR-S9 - The server shall transmit updated player positions in batches with regular intervals** In order to avoid message congestion in games with many players, player positioons are relayed in bundles at regular intervals. These intervals are short enough to make the latency unnoticeable to the players.
- **FR-S10 - The server shall keep track of support score and notify all connected players when the game session is over** At regular intervals, the each team's support is checked to see if it has reached the support limit. If so, an end-session message is sent to all players informing them on the winner.
- **FR-S11 - The server shall allow the server administrator to change the support score limit and notify all connected players as soon as a change has been made** The support limit dictates indirectly how long a game lasts, and longer games have higher support limits. In order to control this, administrators are presented with a sliding bar to change the limit at a whim, instead of modifying the server source code for each game.

Post-game requirements

These are requirements applying to the server after a game of CityZombies.

- **FR-S12 - The server shall close the gaming session if all players disconnect** In order to prevent the server from generating resources and carrying on the game after all players have disconnected, sessions are closed if all players leave.
- **FR-S13 - The server shall release all resources and threads when a game session is over** To avoid an ever increasing number of threads as new sessions are created, the server must release it's resources after each session and enter a wait-for-session-creation state.

9.3.2 Non-functional requirements

Essential non-functional requirements for our game server include modifiability, availability and performance. These categories are explained and rationalized below.

Modifiability

These are the non-functional requirements applying to the modifiability aspects of the CityZombies game server. Modifiability concerns any modification or addition to the system and it's associated cost. A focus on modifiability means that developers should easily be able to both change values and the setup of the present game as well as change functionality in order to use the server for other games. We wish to implement our game server while keeping the modifiable structure of the server framework intact.

- **NFR-S1 - Developers shall be able to add or change functionality in one part of the server without affecting the rest** If developers wish to use other forms of identifying location, or using this game server for similar games, changes in functionality should not affect the integrity of the server.
- **NFR-S2 - The server shall continue to support both TCP and UDP transport protocols** Even if we use only TCP in our implementation, leaving the framework UDP support be will be helpful for developers exploring other aspects of location-based gaming.
- **NFR-S3 - Important server values shall be changeable and easily accessible** Server values like alive request interval, support generation interval, support generation rate and similar should be presented at the top of relevant classes, as in the framework. Support limit should be directly changeable in the server GUI.

Availability

These are the non-functional requirements applying to the availability aspects of the CityZombies game server. Availability is about system stability and failure, and what the consequence of such failure is. A high availability demands a system which is very stable and/or handles instability well. Players will want to be able to play a game whenever they feel like it.

- **NFR-S4 - Players shall be able to connect to the server at any time.** The server should be listening for players trying to connect to the game during both night and day, regardless if a game is already being played or not.
- **NFR-S5 - Players shall be able to disconnect from the server at any time.** The server should allow players to disconnect from the session at any time, and inform the other players accordingly.
- **NFR-S6 - The server shall handle any disconnection without letting it affect the game** Even if a player disconnects during battle, the server should let the remaining player continue as usual.
- **NFR-S7 - The server shall handle any fault without letting the user know** Faults may appear many places in the system, if they appear, the server should fix them or circumvent them without letting the player know they happened.

Performance

These are the non-functional requirements applying to the performance aspects of the CityZombies game server. Performance adresses system response speed and quality. A high performance requires the system to respond effectively to change. Players will want the system to be responsive and accurate to have a good time.

- **NFR-S8 - Network data from the server shall be short and informative**
  To keep the clients from chewing too much information and keep the network cost as low as possible, data messages are short and only sent when required.

Chapter 10

Architecture and Design

In this chapter, we describe and explain the architecture of our CityZombie game application. Software architecture concerns the structures of a system. The structures are defined as software elements, the properties of these elements and the relations between them [BCK]. Since our application is based on an existing framework consisting of a client and server, that framework will be thoroughly gone through along with our own modifications.

10.1 Architectural overview

The framework was made with reuse of code in mind, and the server and client are nearly mirror versions, as methods and classes in the client have corresponding methods and classes in the server. In figure 10.1 below, we show the components of our server and client as well as the communication lines between them. Both server and client are constructed with a three level architecture.

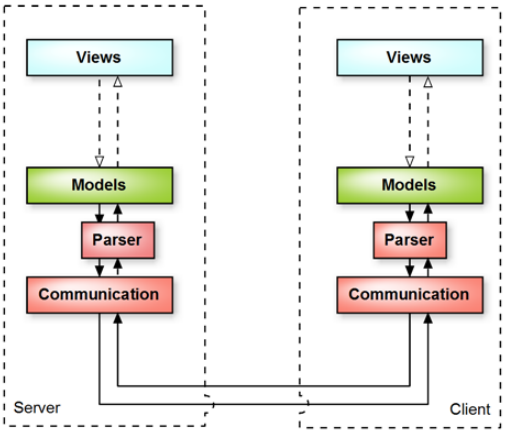

Figure 10.1: Overview of client-server architecture

Views

At the top of our figure, we find the views. These are the most different layers of the three, since the client and the server have different demands when it comes to Graphical User Interface (GUI). Our client needs to show several different screens to the user, including the main menu, lobby and game board, while the server only needs to show its lobby information.

Models

One step down we find the models. These contain all data presented in the views, and keep track of the game state. It is our goal to keep the information in both client and server equal at all times, and to accomodate this the framwork has them sharing the same models. This is true for both messaging, session and player information models.

Communication

The communication layer is situated at the bottom of our figure. The components here controls all communication between server and client. Server and client both share identical parsing interfaces

10.1.1 MVC Architectural Pattern

Our core framework, as described by 10.1, is based on the Model-View-Controller (MVC) design pattern. This pattern focuses on separating the logics of a system from the user interface. In this way, both underlying game logics and graphical user interface can be modified without disturbing the other module. While Java SE and ME provide different levels of support for this, MVC is achievable in both language editions.

Separating functionality into models, views and controllers is the primary principle of MVC. Models contain the system information which the application utilizes, views show this information in a way users can understand and interact with, while controllers handle events and respond accordingly.

In our application, the Communication modules act as global controllers. These modules detect messages and forwards them to the models, changing them in the process. Any view using one such model will then adapt and show the new information. Local controllers in the client and server are implemented through the functionality offered by each individual view. These views detect relevant user input whenever they are active and forwards this to the models.

10.2 Client design

In our implementation, we use the already defined architectural structure of the framework, and mostly add to the views and models, leaving the communication modules almost untouched. Figure 10.2 below shows the details of our client architecture. The main difference between this figure and the figure presented on the previous page is the bottom layer, which contains functionality found in Java ME. The communicator listens

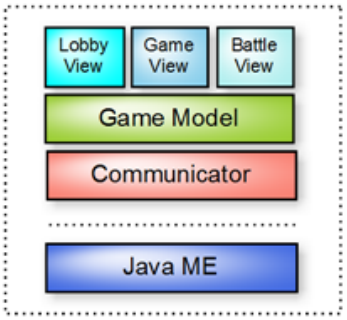

Figure 10.2: Overview of client architecture

to messages from both the server ommunicator and the client game model. When received, it sends these to the other part. Our game model keeps track of the game's state at all times and handles messages received or to be sent. The game view is separated into three parts; lobby view, game view and battle view. The lobby view contains game information like players and scores. The game view presents the players to the actual game screen where movement and zone ownership is displayed. The battle view simply shows battle reports to the player in a turn-based fashion.

10.2.1 Client classes

Figure 10.3 below shows the class diagram of our client classes and their internal relationships. There is a distinct separation between the game specific CityZombie classes and the framework classes. Although we made some additions to the framework views, we count them as part of the framework, seeing as they do not need to be functionally altered in order to be used in some other game. Methods and variables have been left out at this point. To study the classes in more detail, please refer to Appendix C.1.

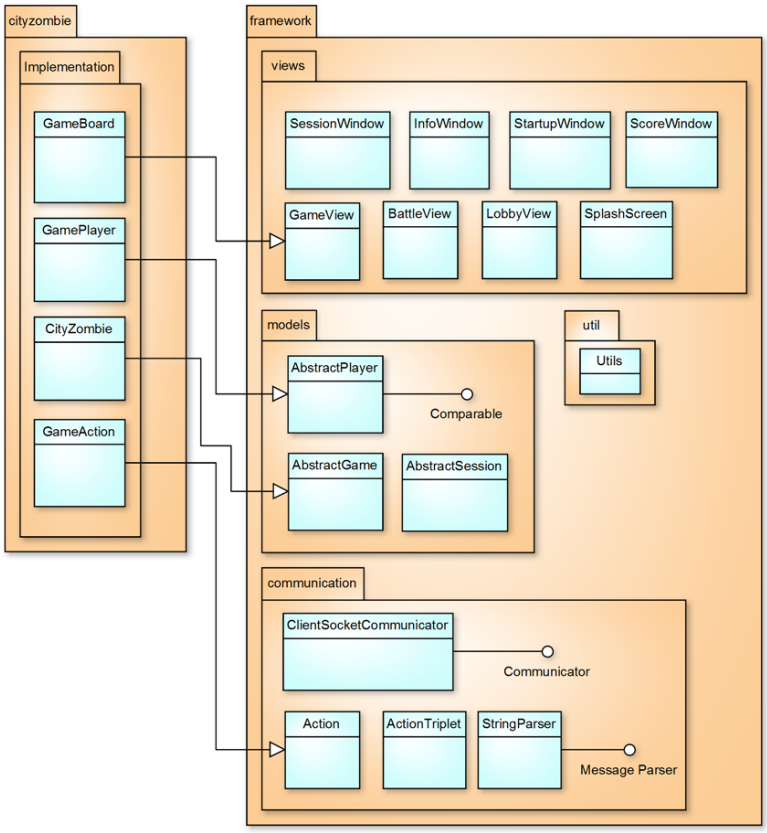

Figure 10.3: Overview of the client classes

10.2.2 Client models

Figure 10.4 below shows all models used for our game client and their relationships. To the right we find the framework specific models, and to the left is our game specific models. Though vector annotation us used to describe the possibility of several instances of both players and sessions, that only applies to the players in our game, as we only allow one session at a time. The client has three models, containing the core functionality of a multiplayer game. We expand this functionality through our implemented game-specific classes, which inherit general functionality from the framework classes. The following

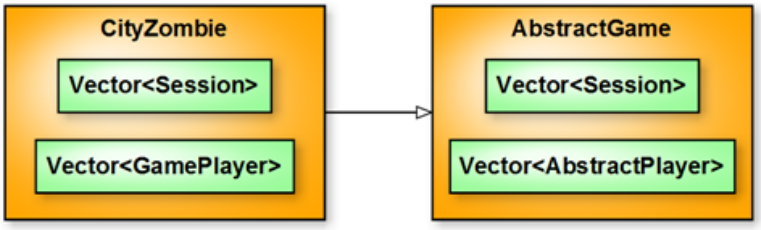

Figure 10.4: Overview of the client models

five classes constitute our three models.

- **AbstractGame** is our main model. It contains the methods for both starting and stopping the client as well as general game functionality like view handling.
- **CityZombie** is our implementation of the *AbstractGame*. It contains game specific functionality like battle initiation and zone updates.
- **AbstractPlayer** contains all connected players' information, and is thereby a representation of each player connected to the game.
- **GamePlayer** is our implementation of the *AbstractPlayer* model and contains information specific to CityZombie players.
- **Session** models the currently active session, containing a list of connected players and associated settings.

10.2.3 Client views

As seen in figure 10.2, our main views are the *GameView*, *BattleView* and *LobbyView*. In addition to these, several game window classes are used to display and receive information to and from the player. The AbstractGame model controls which view is visible at all times.

Some of the minor windows were not present in the framework, but we added them there, as they are not necessarily game specific, and adds supporting functionality. These are the *SplashScreen*, *InfoWindow* and *ScoreWindow* classes. The LobbyView is only slightly modified, while the GameView and its extension are heavily altered to make room for game specific functionality. Screeenshots from the different implemented views are presented here as well.

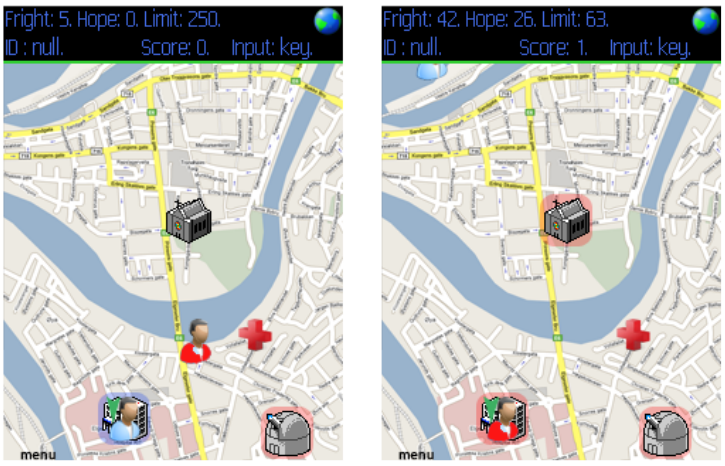

Figure 10.5: To the left: The human view before a battle. To the right: The zombie view after a battle.

- **GameView** is the main view containing the board where game movement actions are displayed. The GameView gets its game information through the *AbstractGame* model, but listens for local positional updates on its own. In our implementation, the player may choose his own input, either keypad or CellID, to move. This is for testing purposes only. The GameView also gives the option of showing the status bar, providing additional information like support levels and score to the player. Figure 10.5 above shows the main gameview as seen from the human player before a battle, and seen from the zombie player after a battle. The zombie player won and seized the zone, the human player lost and was rendered uncouncious.
- **GameBoard** is our extension of *GameView*. It contains all game specific images and sprites, as well as methods for displaying the correct zone structure.

Figure 10.6: Left: The human player after round 4. Middle: The zombie player after round 5. Right: The human player won.

- **BattleView** is displayed after combat is engaged. It simply draws the combat report on the player screen, displaying both hit points and health bars. No user input. Figure 10.6 above shows examples from the battleview. Each round, the player's hit points are both written and illustrated via a hit point bar.

Figure 10.7: Left: A zombie players starts a session. Middle: A human player joins the session. Right: The game is over.

- **LobbyView** is displayed before and after actual games. It is a intermediary holding screen for players connected to the game, but not actively playing. The LobbyView shows connected players, their score and team. Before games it gives you the opportunity to join the game and after the game you get the choice to start a new game. Figure 10.7 above shows the lobbyview from a starting zombie player's view,

a joining human player's view, and a recently finished human player's view.

Figure 10.8: Left: Before launching game. Middle: Initial splash screen. Right: Tactical splash screen.

- **SplashScreen** is a temporary screen shown as soon as you execute the game. In this setting it is merely an opening image. It is also shown when you engage in combat. In this setting it shows a different picture telling you to provide tactical input. Upon pressing any button the SplashScreen is dismissed. In our implementation some buttons send a tactical response to the server depending on your input. Figure 10.8 above shows the game folder before launching the game, the initial splash screen which shows up after launching the game, and the tactical splash screen which shows up after entering combat.

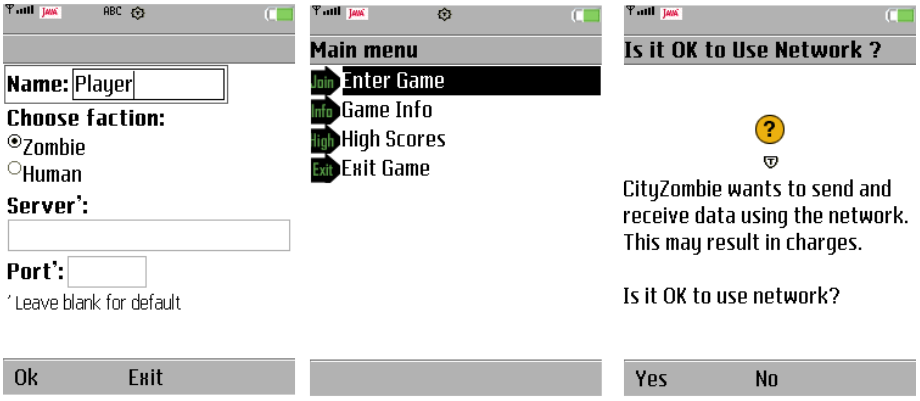

Figure 10.9: Left: Connection information screen. Middle: Main Menu. Right: Warning after pressing join.

- **StartupWindow** is the first window shown after the initial splash screen. In this window, the players may enter their names and server adresses in text boxes. A choicegroup for faction choice is also provided. This window also provides the next screen, which is the main menu. The main menu gives you the option of joining a game, viewing game information, viewing the high score list and exiting the game. Figure 10.9 shows the initial screen for providing name and connection information, the main menu and the default screen that shows up whenever a Java application tries to access the network.

- **SessionWindow** shows the currently active sessions on the connected server. This window is automatically bypassed, as players shouldn't be allowed to create additional sessions.

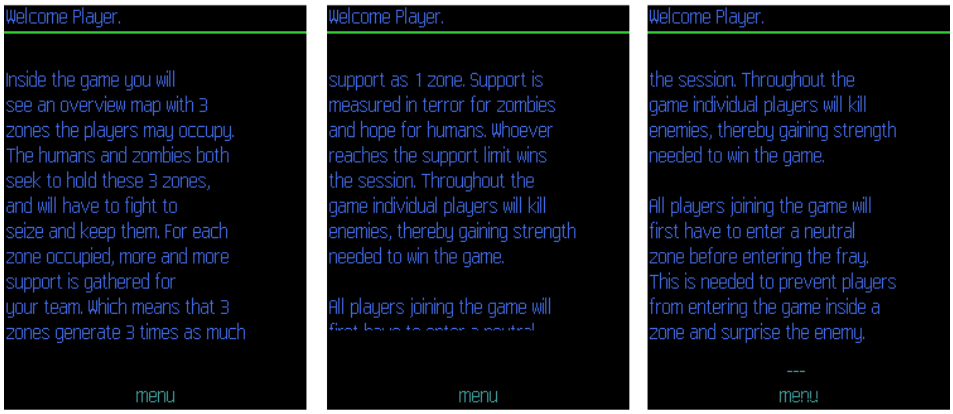

Figure 10.10: Left: Top of the window. Middle: While scrolling. Right: Bottom of window.

- **InfoWindow** contains information on how to play the game. No manipulation option. Figure 10.10 shows the scrollable information window at various stages. At the bottom of the screen, a mark shows up when trying to scroll further, marking the end of the screen.

- **ScoreWindow** is a simple display of the high scores stored in the *AbstractGame* model. No manipulation option. Figure 10.11 illustrates the differences in record-store implementation between emulator and mobile phone. The emulator score places itself where it should, while the values on the mobile phone place themselves randomly. This is discussed further in Section 12.2. Note: The two pictures are from different games.

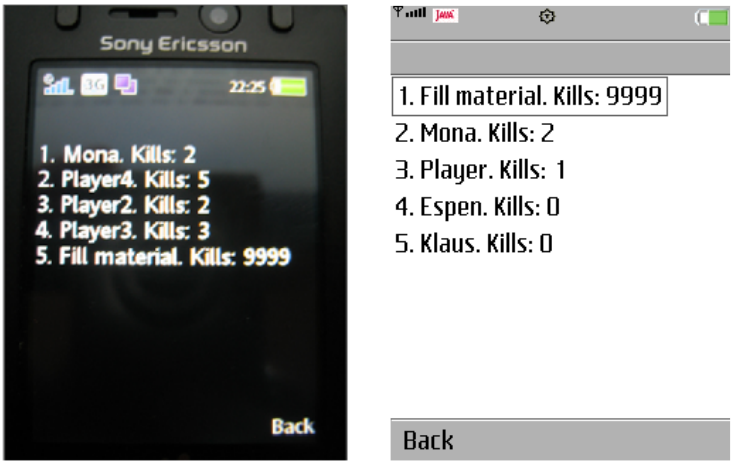

Figure 10.11: Left: High score window on a mobile phone. Right: High score window on the emulator.

10.3 Server design

In our implementation, we use the already defined architectural structure of the framework, mainly altering the view and models, leaving the communicator almost untouched. Figure 10.12 below shows the details of our server architecture. The communicator in the

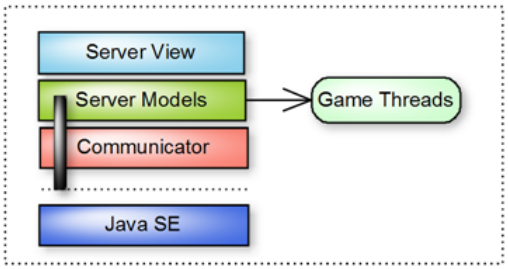

Figure 10.12: Overview of server architecture

figure above receives and forwards messages much in the same way our client communicator does. Both from the server models and the client. The main difference between the client and server communicator, is that the server contains several communicators, one for each connected client. Such a unique connection between server model and individual communicator is illustrated with the black bar on the figure. Our server model keeps a representation of the game state, just like the client game model does. The framework supports several sessions, each keeping their own server model with its associated game threads. These threads generate support and handle game events. The *server view*

shows the details of any active session, including connected players and scores, support generated and and a slider to regulate support limit.

10.3.1 Server classes

Figure 10.13 below shows the class diagram of our server classes and their internal relationships. There is the same separation between the game specific cityzombie classes and the framework classes in our server as there is in our client. Methods and variables have been left out at this point. To study the classes in more detail, please refer to Appendix C.1.

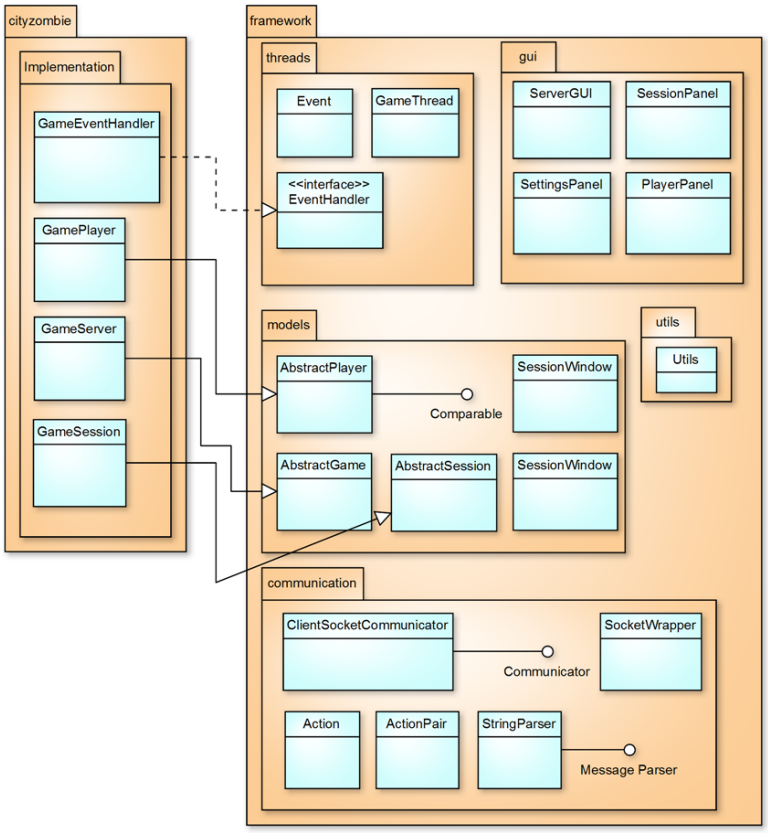

Figure 10.13: Overview of the server classes

10.3.2 Server models

Figure 10.14 below shows all models used for our game server and their relationships. Much like the client, some server models are allowed to exist in several instances. At the top we find the abstract framework classes, and at the bottom our implemented extensions of these. A list of the classes that constitutes the server models follows below.

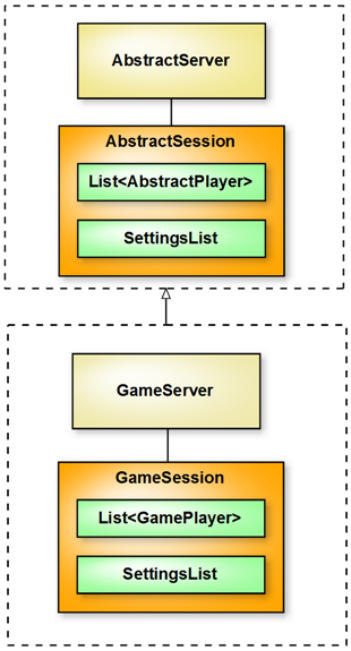

Figure 10.14: Overview of the server models

- **AbstractServer** is our main server model. It contains a list of players connected to the server and a list containing the one allowed session. The AbstractServer model is also notified by the *Communicator* when messages arrive.
- **GameServer** is our implementation of the *AbstractServer*. It handles game specific actions from the *Communicator* and keeps track of player models.
- **AbstractSession** represents a session, or the session. The session model contains a list of connected players and the settings associated with the session. In addition to this, the session is responsible for running the game threads and notifying the *EventHandler* when game event messages arrive.
- **GameSession** is our implementation of the *AbstractSession* model and contains information specific to CityZombie players.
- **AbstractPlayer** is identical to the AbstractPlayer found in the client, except for the individual *Communicator* object needed to transmit and receive messages unique to that particular player.
- **GamePlayer** is our implementation of the *AbstractPlayer*, no extra functionality is provided.
- **SettingsList** contains a list of the settings associated with a session. The session *GameThread* accesses the SettingsList to validate the game state.

10.3.3 Server views

As seen in figure 10.1, there is only one server view. This view allows administrator to easily see who's connected to the game, and the status at all times in a simple GUI. It also provides administrators with the option of changing the support limit, effectively altering the win-condition of every game as seen fit. A screenshot of the server GUI is provided.

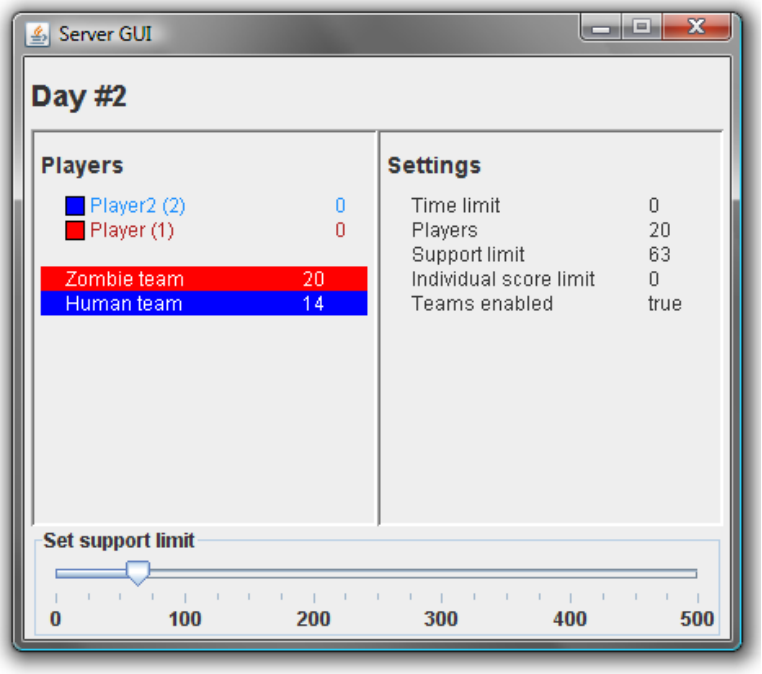

Figure 10.15: The different panels of the server GUI.

- **ServerGUI** is the main JFrame which contains a *SessionPanel* and a JSlider for changing support limit.Figure 10.15 below shows the server GUI with a playerpanel, settingspanel and a jslider for changing the support limit.
- **SessionPanel** is a JPanel containing a *PlayerPanel* and a *SettingsPanel*.
- **PlayerPanel** contains information on players connected to the session. Score, support and names are shown.
- **SettingsPanel** contains information on settings associated with the session. Support limit, player limit and kill score limit are shown.

10.4 Communication

The shared Communication layer at the bottom of both client and server in the framework, as seen in figure 10.1, uses a *Communicator* interface at both sides to ensure that

messages are sent and received in a orderly fashion. This interface specifies which methods that need to be implemented in this layer and makes sure that both client and server uses the same operations. Examples of these methods are `connect()`, `disconnect()`, `sendMessage()`and `notifyAboutMessageReceived()`.These commands specify connection to a server, disconnection from a server, forwarding of a message from the model to the network and finally notification to the model that a message has been received from the network. The following subchapters explain the specifics of the Communication layer in more detail.

10.4.1 Protocols

As mentioned earlier, the framework supports two of the three transport protocols available in the Transport Layer of the Internet reference model; namely TCP and UDP. Server sockets are used for both types of communication. To make the game behave equally with UDP and TCP, the framework also provides a `SocketWrapper` class for the server, which translates messages of both kinds into the one selected by the user, making the actual differences in implementation transparent. The client is written in Java ME, in which the `Connection` interface is used to specify communication for both TCP and UDP, and needs no wrapper.

While TCP uses connection-oriented communication via a `Socket` stream, where a connection only needs to be established once, UDP uses `DatagramSocket`objects, discrete data packets of a specified size, which each needs to be adressed to the receiver before sending them.

In our implementation, only TCP communication is left as an option for users. The reasoning behind this is described in Section 8.2

10.4.2 Messages

The framework also provides a message parser, which is implemented in much the same way on both server and client. This parser makes sure that communication between server and client follow the same set of rules. These message rules are explained in the following section.

Message Format

Our implementation of CityZombie uses the `StringParser` to implement the `MessageParser`. The `StringParser` only accepts messages in a `String` format. In the same vein, it only creates `String` messages as well. A set of actions are specified in order to control the parsing of messages, the framework actions are defined in the `Action` classes and the game-specific actions are defined in the `GameAction` classes of both client and server. This rule list, written in Extended Backus-Naur Form (EBNF)[1], can be found in the

[1]For more information on reading EBNF, see [EBN]

framework documentation [RTG]. Some message specification examples from this list are restated in the table below.

Action	Rule
`ALIVE_REQUEST`	: "ARQ:"<requestId>"|"
`START`	: "STA:|"
`POSITION`	: "POS:"<positionX>","<positionY>"|"

Table 10.1: Framework rules

Actions

As mentioned above, actions define the parsing of messages, and we use most of the actions specified in the framework in our implementation. In table 10.2, we list all the framework actions we make use of, and in table 10.3 we list the CityZombie-specific actions we have added. The tables describe what the different actions mean, when they are used, the string message associated with each action and in which communication direction they are used. The latter point specifies whether they are sent by client, server or both.

10.5 Threads

The server session contains one *GameThread* and one *EventThread*. These two threads run the game between them. For the framework, these are called `GameThread` and `EventHandler` , our implementation class of the `EventHandler` is called `GameEventHandler`. All threads running in one such server session can be seen in figure 10.16 below. The

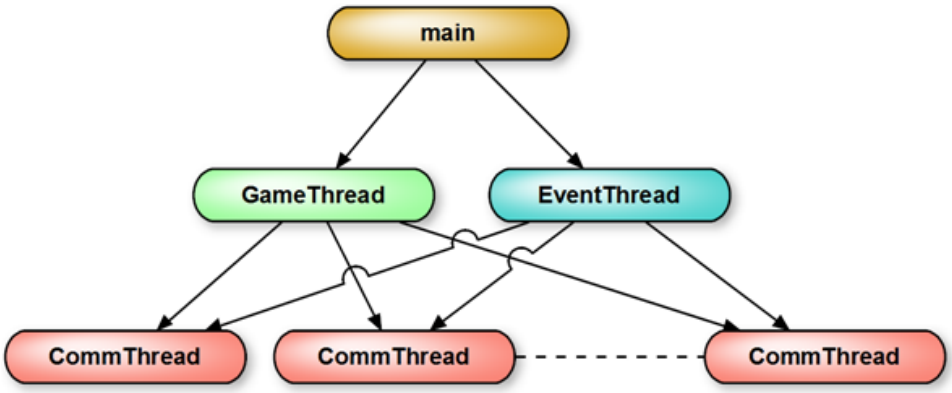

Figure 10.16: Overview of the server threads

Name	Value	Used when?	Describes what?	Sent to?
`Player`	PLR	A new player connects to the server, or requests a refresh of the session list.	Name, color, team and other player information.	Both
`ALIVE_REQUEST`	ARQ	A connection acknowledgement is requested or confirmed.	A unique request id.	Both
`SESSION_LIST`	SSL	A new player connects to the server.	A list containing information about the current session.	Client
`SETTING_LIST`	STL	A new player joins a session.	A list containing the current session's settings.	Server
`SESSION_SELECTED`	SES	A player selects a session.	The id of the current session or a negative value if creating the session.	Server
`READY`	RDY	A player changes his ready status.	The new ready status to server, player id included from server.	Both
`START`	STA	A player starts or joins a game.	Zone ownerships and starting position.	Both
`SCORE`	SCR	A player defeats an enemy.	The player's new score.	Both
`POSITION`	POS	A player or several players has changed his position.	Local position to server. All player's moves and id's from server.	Both
`GAME_OVER`	GAM	The support limit has been reached.	A description of why the game ended.	Client
`DISCONNECT`	DIS	A player disconnects from the session.	Nothing to server. The player's id from server.	Both

Table 10.2: Framework actions

GameThread keeps track of the players' positions and forwards these positions to the other players whenever they change. This forwarding happens in batches, with an interval that can be set to any value.

The EventThread contains functionality for generating events, this could include mines and robots, as mentioned in Chapter 8 In our present implementation, however, the EventThread contains methods for simulating battles between players and sending processed information back to involved players, based on external messages.

Each connected player also has a *Communication* thread associated with the connection. These communication threads are used by the GameThread and EventThread to send and receive messages to and from the different clients. Furthermore, each communication thread consists of one thread for receiving and one thread for sending messages.

Name	Value	Used when?	Describes what?	Sent to?
`ATTACKS`	ATC	A player enters a zone with an enemy player.	The id of defending player to server. The attacking player's attributes from server.	Both
`HITS`	HIT	A player is ready to fight.	Player tactics to server. Battle report from server.	Both
`ZONE`	ZON	Zone ownership has been externally updated.	A list containing information about the most current zone ownerships.	Both
`SUPPORT`	SUP	A session generates new support.	The current support values.	Client
`LIMIT`	RDY	An administor alters support limit.	The new support limit.	Client

Table 10.3: CityZombie actions

On the client side, we find similar sender- and receiver threads.

Clients also keeps threads for polling local player position, controlling the `BattleView` and performing client calculations. These threads are called `PositionThread`, `BattleThread` and `LocalThread`. They are all located in the `AbstractGame` class. While the `PositionThread` and `LocalThread` run continually, but at different rates, the `BattleThread` is only invoked when a battle report is received from the server and closed as soon as it is finished. `LocalThread` detects collisions and updates the game board while `PositionThread` fetches and sends player positions.

Part IV

Test results and user feedback

Chapter 11

Test results

This chapter presents the direct test results from our project. We will comment infrastructural results and some statistics from cell phone use, including game data transfer and it's associated cost. Finally we will comment on the various playtesting results from the different stages of the game development process.

11.1 Infrastructure

Since we opted to use the basic CellID to locate the players of our game, we did not have the exact positions of GPS to rely on. The use of base station location requires the developers to do some manual mapping prior to and during game implementation. Some services for mapping location and base station exist, like Cellspotting.com [CSP]. These services, unfortunately, are not yet accurate enough, and lack boundary information. They do have means of connecting latitude and logitude to base base station coverage, though. In this section we present results associated with exisisting infrastructure and landscape mapping.

11.1.1 GSM base stations

As expected from the studies carried out in Section 5.3.5, GSM base stations use a lower frequency signal with a higher intesity, making them carry longer. This results in larger base station coverage areas compared to UMTS. Figure 11.1 on the next page shows the relevant results from the manual mapping of some coverage areas in Trondheim.

11.1.2 UMTS base stations

Switching from GSM to UMTS on the cell phone, we were able to extract UMTS CellID values in the same way as with GSM. Logging these values and connecting them to a map manually as we progressed provided us with a complete map from which we could decide on the values to use. Figure 11.2 shows the relevant UMTS zones in Trondheim. As you can see from the figure, picking the right Telenor zones is not too hard. As long as there

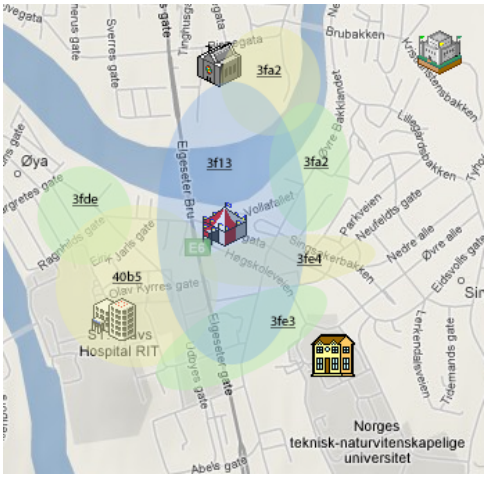

Figure 11.1: Overview of GSM base station coverage in Trondheim

is a cell between two game zones, the zone structure is pretty easily set up. From the results gathered, we find that zones cannot be any closer, meaning that the game map we implemented is the smallest possible map to use in practical manner. It follows that distances are pretty high. The walking distance from NTNU to the Nidaros Cathedral is about 1km, making it a physical challenge as well as a tactical one.

11.2 Game Statistics

The actual game testing also provided us with results concerning network use and practical game values. Most important of these are response times and game data transfer.

11.2.1 Response times

The response time of a network is the time it takes for a data packet to from the server, to the client, and back to the client again. When using GSM localization, the carrying network technology is limited to EDGE. For UMTS, HSPA is the highest achievable carrying technology. The framework used for our game has been thoroughly tested with regards to response times. In our implementation, however, response times need not be as low as possible, only within a certain limit. The more important factor is reliability. Positional changes don't happen to often, but when they do, it is imperative that all players are made aware of that one positional change. With UDP, packet loss can constitute a major problem, despite the low response times. We tested the game at an early stage with both UDP and TCP, since the functionality for this already was implemented in the framework. Early testing showed that even though rare, packet loss would occur. The most important results of the framework response time testing carried out in [RTG] is restated in table 11.1 below.

Figure 11.2: Overview of UMTS base station coverage in Trondheim

Given any fast-paced game, the choice would be easy. However, in our game we

Network	Protocol	Avg. Response time
GSM	TCP	750ms
	UDP	300ms
UMTS	TCP	700ms
	UDP	250ms

Table 11.1: Framework Response Times

update the game board and checks for positional updates every 1500ms, while sending the location every 350ms, if it has changed at all. In that perspective, a difference of 400ms is not vital. On the other hand, if we use UDP and lose the packet containing the positional update, the client will not resend this until it changes position again, which may take a lot longer.

11.2.2 Game Data Transfer

Any use of network data transfer comes at a cost. Service providers charge users per kilobyte, both transmitted and received. The amount of data transferred depends on several factors. These factors are transport protocol, game type and number of connected players. Players more involved in battles will also see higher network use. The game data transfer values are retrieved from the mobile phone's internal data transfer counter. Our tests are performed with a Sony Ericsson K850i and W910i. When a player starts or connects to a game, he sends the name and screen resolution to the server. The server responds by sending the starting position and zone status. At game creation, all

zones are neutral.

The prices for data transfer are steadily declining, and as this thesis is written, Telenor offers all mobile access at a flat rate of 12.50 NOK per MB [TPR] while Netcom charges 20 NOK per MB [NPR]. Tele2 offers prices from 12 NOK per MB [TLP]. In our tables, we will calculate with a conservative price of 15 NOK per MB. The data amount in the tables are measured in bytes. Table 11.2 shows that very little information needs to

Type	Data transferred	Cost
Sent to create a session	1179	0.017 NOK
Received to create a session	1349	0.020 NOK
Total to create a session	2528	0.038 NOK
Sent in 1.min idling in lobby	1470	0.022 NOK
Received in 1. min idling in lobby	1750	0.026 NOK
Total in 1 min. idling in lobby	3220	0.048 NOK

Table 11.2: Simple game data statistics

be sent in order to create a game. Even idling in the lobby of a created session demands very little data transfer, as only alive requests are transmitted. This opens up for the possibility of staying in the lobby for a while until someone else logs on to play with you.

There is more to the game than creating it or being passive, though, and several complete tests were carried out to see the data transferred in a full game with different amounts of players. The values stated in table 11.3 are the average values between the two phones carrying out the test. In the tests with 4 players, an additional two emulators were used to generate data. Where not stated otherwise, the games were carried out with 3 battles each, though more battles are likely in the longer games. As we can read from the tables, CityZombie is a fairly low-cost game. Updates happen rarely because of the distances involved, and therefore data use remains low. Even when we double the number of players and triple the number of battles, we don't see a very high increase in total cost.

11.3 Playtesting results

During the game development process, several testing sessions were performed in order to get ideas and limitations for a game and to test the result.

Early stages

At the very early stages, focus groups can work as a catalyst for idea generation and provide a realistic frame for the game developers to test paper prototypes and possible solutions. A big difference between using experts and people from the target group at this point is the attitude towards the game mechanisms. While experts know the premise

Type	Data transferred	Cost
Sent in 5 min game w/ 2 players	8276	0.124 NOK
Received in 5 min game w/ 2 players	10535	0.158 NOK
Total in 5 min game w/ 2 players	18811	0.282 NOK
Sent in 10 min/200 limit game w/2 players	13140	0.197 NOK
Received in 10 min/200 limit game w/2 players	17190	0.257 NOK
Total in 10 min/200 limit game w/2 players	30330	0.455 NOK
Sent in 5 min game w/ 4 players	7950	0.119 NOK
Received in 5 min game w/ 4 players	9884	0.148 NOK
Total in 5 min game w/ 4 players	17834	0.267 NOK
Sent in 10 min/10 battle game w/ 4 players	15179	0.227 NOK
Received in 10 min/10 battle game w/ 4 players	19969	0.299 NOK
Total in 10 min/10 battle game w/ 4 players	35148	0.527 NOK

Table 11.3: Full game data statistics

of location-based gaming and are aware of the possibilites for locating a mobile phone, very few others knows of this field.

A questionnaire with questions regarding age, shape, previous experiences with mobile games etc. was sent out to 15 persons, ranging from technical savvy students to more novice users of mobile phones. All within the age group 15-25. While all had played games on their mobile phones, none had heard of location-based gaming, much less were they able to mention games or applications using location. 3 out of 15 knew of map applications on their phones when asked specifically. Because of the low amount of preferences and knowledge, it is hard to get feedback during the early stages of a game like this. Since players don't have anything against which they can compare the concepts revealed to them, answers vary from moderate support to very positive responses, based on nothing but the ideas from the developers. However, players that regard themselves as physically active above the average were generally more enticed by the game premise than those who regarded themselves under average.

Implementation stage

At this stage, players were presented with something more hands-on, which provided developers with more useful feedback in terms of both technical solutions and playability factors. Fellow students had a low threshold for giving supportive ideas based on technical challenges.

To retrieve data, users were guided through a game emulator by the developer, trying different features and getting a feel of the gameplay. However, since a game like this is so limited on an emulator, many were discouraged by the simplicity of the game. In

addition, using keys to simulate zone shifts in a game like this removes a lot of game time. A stretch of 500 meters is covered in a single click, leaving the players in a situation where they always wait for something to happen. During this stage, we became aware of the necessity for field testing at an earlier time, since emulation provides such a reduced gameplay.

Field testing

After having installed the game on two handsets, players were fed information on how to play the game and sent out in pairs to compete. The game was monitored by a developer connected to the game via emulator.

The first result was that players seem more wary at first, as opposite to the sessions on emulator where they were guided by developers and could hardly wait for something to happen. Since input comes so slowly, some wondered if the game was broken, seing as no visible movement was made. Another early aspect was players losing track of their own avatar. This was remedied by altering avatars to differ between fellow team members and their own player. Another comment at this stage was the limited ability to communicate with fellow players. Players are forced to shout to each other or simply deduce movement based on the display. In any game where you have to rely on fellow players for success, limits in communication is an annoying factor. Another discouraging factor was the occasional bug and connection issue. Even though they are not directly related to the playability of the game, it effectively forced the ongoing game to restart, since players had to return to the developer to have the session restarted. Again, the distances involved did not support this in a positive way.

11.4 Summary

Compared to other means of fetching position in a location-based game, CellId is rather time consuming, since cell areas will have to be mapped beforehand. On the other hand, the technical challenges are not very demanding, since a simple line of code will retrieve your actual position. The process of mapping an area could be made easier if connecting a GPS to the phone, and relating the absolute positions of the GPS to the phone's current CellID. Such an approach would likely provide more and thereby better values. This method was not used because of time constraints in our project.

Making use of the retrieved geographical mapping of base stations is another challenge. Few areas can be mapped to real-world locations, which is a way to make the virtual world easier to understand for the players. And those who do fit well with the street map are often overlapping with other cell areas. The big difference between operator networks makes it even harder to make a game that fits a broad range of players. How-

ever, our testing here in Trondheim proves that it is possible to find well-suited zones and structures that support location-based gaming.

Using TCP connections in multiplayer gaming has it's advantages and disadvantages. The main disadvantage is the increased latency and transfer times, making network operations slow. In order to mask this slow pace, the game will have to use the network more seldom and generally not demand a high level of input-output synchronization. Keeping the user unaware of when he actually moves into a zone effectively masks this effect. Zones are large, and no user can move 50 meters in the second or two it takes the message to reach its destination anyway, which means that the transition between local zone movement and display update is negligeable. A more positive aspect of the slow game pace, is use of network traffic. Any game that isn't oriented towards real-time action is playable with a low amount of data transfer, making the entire session relatively cheap. In our implementation users may play almost fifteen 10-minute games against 3 of their buddies for less than 10kr. A more likely situation, though, is fewer games that last longer, depending on the support limit at the time.

Players themselves seem relatively supportive of location-based gaming, mostly because it represents something new, but also because of the social aspect. Multiplayer games seem more appealing despite the network cost. Playtesting during the early stages of development were not so valuable when turned towards average players, at this stage experts or people with background in mobile development are much more helpful in pointing out concepts and ideas that are both challenging and practical. Furthermore, as developers, we were made very aware of the need for field testing at an earlier stage. Half of the bugs and gameplay issues arose first when tried with actual players outside. This part needs a lot of time, more so because of the sluggish testing process with different key locations instead of any relative movement. Having a mobile development studio would definitely help out at this stage, keeping developers more in contact with actual play sessions.

Chapter 12

Problems encountered

This chapter states some of the problems we encountered during this project. Most problems have yet to be solved, and these are described along with an asessment of the impact they had on our results.

12.1 Framework problems

This section describes problems related to the framework and our implementation of it.

Disconnection error

When using the client via mobile phones, the server will sometimes not register that a client has disconnected, and keep trying to send alive requests. This leads to an error in the SocketWrapper class, since it no longer has a TCP connection on which to send information. Using emulators never results in this error. This seems to be caused by a weakness in the mobile TCP/IP network compared to cabled TCP/IP. Sometimes, though not lost, TCP packets will get a low priority, and be cached somewhere in the network until they disappear. This leads to the server not knowing they are gone.

This is not a big problem, as clients will never know of this unless they try to connect shortly after disconnecting. Players suffering from this error will remain in the server list until they time out. The current timeout value is set to 7 seconds, and players will be removed from the server when this happens. The problem could most likely be fixed by having the client send activity packets, and letting the server keep a time-stamp on each client.

12.2 Java problems

Making a high score list

As we developed the high score list, we used the emulator exclusively. The way the high score list is implemented now, we create a `RecordStore` in the `AbstractGame` class. This is filled with dummy scores when the game is opened for the first time. As the game progresses, each client keeps a tab of every player's score. When a player disconnects, all clients informed of that disconnection checks his score against the values of their recordstores and places that player in his correct position. The same happens when a local player exits the game, he checks the scores of all players, including himself, and puts them in the high score list if applicable. In the emulator this worked like a charm, as seen to the left in figure 10.11, Section 10.2.3.

When testing this on the mobile phones, however, we realized that something must be implemented different by Sony Ericsson. Here, all values are scattered seemingly at random, as seen to the right in figure 10.11 in Section 10.2.3.

We were not able to fix this because of time constrains, but it should be relatively easy to retrieve information on the Sony Ericsson-specific implementation from official sources.

12.3 Field testing problems

This section covers problems encountered during the field testing stage.

Zone overlap

As base station signals overlap, there will be certain locations where the phone jumps from cell to cell very rapidly, this is naturally seen by the game as fast zone movement, and positions will be sent to the server accordingly. Because of this, we would have liked to implement a security mechanism in the locating algorithm, making it ask for location twice when entering a new zone, with some delay between polls. If the second location is not the same as the first, ie. a quick zone overlap jump has been made, the process is started over again. On the other hand, if the location is the same, a zone move will be initiated as normal. Instead, the game is optimized for zones where the overlap problem is less apparent.

UMTS coverage

When field testing, we noticed that not all locations, even very central locations, are covered by an UMTS base station. From time to time the mobile phones would switch from UMTS to the GSM network, changing CellId in the process. This could be a prob-

lem if the game or application relies exclusively on UMTS location and does not accept GSM location strings. When mapping the zones, we tried to avoid involving these areas in our gameplay, and the game now simply ignores such zones. Jumping to the GSM network has no effect on the ability to send or receive updates to and from the server.

12.4 Other problems

ScrumWorks

During the development process, we had a Linux server running ScrumWorks. In ScrumWorks we listed all user stories, split these into manageable tasks and kept track of the progress every day. This was a good solution for organizing the project, and a very accessible and understandable way of viewing the progress and remaining tasks.

The problems started when the server harddrive crashed, some halfway out in the project. This was later attributed to temperature issues and poor cooling. All information was lost, and with no backups, we were forced to abandon ScrumWorks and the daily update process. By the time this happened, though, most of the project was well underway, and the need for planning was less than in the early stages. We discuss this topic further in the method evaluation of Section 14.2.2

Chapter 13

Requirement fulfilments

This chapter evaluates whether or not the requirement stated in chapter 8 are fulfilled or not. Every requirement is restated here, with a description of the solution or the lack of one. Requirements stated in green are considered fulfilled, while requirements written in red are considered only partly or not fulfilled.

13.1 Client Requirements

This section covers all client requirements along with an evaluation.

13.1.1 Functional Requirements

The client functional requirements were stated in Section 9.2.1, this section states whether or not they were fulfilled and describes how they were implemented.

- **FR-C1 - The player shall be able to provide his/her own name, limited to 12 characters, no numbers**
  Players may enter their name in the initial startup screen.
- **FR-C2 - The player shall be able to choose faction before he/she joins the game**
  The startup screen provides an exclusive choice between Zombie and Humans.
- **FR-C3 - The player shall be able to enter server IP and port number upon starting the game**
  The startup screen provides the player with fields for both IP and port number. If nothing is written, default values are used.
- **FR-C4 - The client shall provide a main menu with information on how to play the game**
  The main menu has an option called *Game Info*. This screen provides information on the gameplay.

- **FR-C5 - The client shall provide a main menu with a high score screen**
  This requirement is only partially fulfilled. While the high score screen is provided, and works on emulators, it does not work as intended on the test phones.
- **FR-C6 - The client shall not initiate communication with the server before the player chooses to join a game**
  The client does not connect to the network until *Enter Game* has been pressed in the main menu.
- **FR-C7 - The player shall be provided the ability to exit at any give point**
  The player may exit the client at all times. An exit button is present in every screen exept the battle screens. At this point the player may still exit with no disadvantage to the remaining players.
- **FR-C8 - The client shall connect to a server when pushing "Enter Game" in the main menu**
  The client connects to the specified server.
- **FR-C9 - The client shall refresh the session list automatically if the game server does not provide it on the first try**
  The client automatically refreshes the session list once if no sessions have been returned the first time. Subsequent refreshes must be made manually.
- **FR-C10 - Any player shall be able to create a game session if one does not already exist**
  The first player to enter a game automatically creates a session.
- **FR-C11 - The client shall enter a game session automatically if one is found**
  If an open session exists, players will join this automatically.
- **FR-C12 - If a game session does exist, the player shall not be able to create a new one**
  The client joins the existing session without asking the player.
- **FR-C13 - The client lobby shall present all connected players in the lobby**
  All connected players are presented in the lobby.
- **FR-C14 - The client lobby shall list all connected players with kill scores and according to their team**
  All connected players are presented according to name, faction and score in the lobby.
- **FR-C15 - The client lobby shall automatically update reflecting session changes**
  The client lobby refreshes whenever a change has been made.

- **FR-C16 - The client shall provide the player with an overview map**
  The main game screen background is the overview map.

- **FR-C17 - The player shall be able to enter and return from the lobby to view connected players during the game**
  A menu with a "back"-command is provided, and marked on the lower left corner of the map, as commands are not visible in full screen mode. This command takes players to the lobby where they may view all connected players.

- **FR-C18 - The client shall check for positional change at regular intervals**
  In the current implementation, the client polls for location every 1500ms.

- **FR-C19 - The player shall have to move into a dedicated neutral zone before being able to attack any zone**
  When players first enter the game, the only check made is to see if they are in the revival zone. This remains the only check performed until players enter that zone. At that point, a flag is set and all positional checks are made.

- **FR-C20 - The client shall automatically inititate an attack on a zone when the player enters its vicinity**
  When the client registers that the player has moved into a particular game zone, a takeover attempt is initiated. If no enemy is present, the zone will simply change ownership according to the attacking player.

- **FR-C21 - The client shall check for enemy players at each zone change**
  After moving into a zone, the client checks for collisions with enemy players. If an enemy is present, a battle initiation is sent to the server.

- **FR-C22 - A player shall automatically defend a zone upon it being attacked by an enemy, if presently in the zone**
  If the defender is present when the attacker checks for collisions, his id will be linked to the attack message.

- **FR-C23 - The players shall be prompted to send their battle tactics prior to any battle**
  After sending and receiving information of a battle, both defending and attacking player are showed the tactics window where they may enter a tactical keycode.

- **FR-C24 - The client shall present received battle reports as an ongoing battle in a dedicated window**
  The `BattleView` shows battle report values with delays, simulating turn based combat.

- **FR-C25 - After a battle, the client shall take action corresponding to the outcome of the battle report**
  If an attacker wins, he is the new owner of the zone. If the attacker loses, he is sent

into limbo. A state where no checks are made except for revival zone. The same goes for the defender.

- **FR-C26 - After having lost a battle, the player shall have to move into the dedicated neutral zone to resurrect his avatar**
  After a defeat, the client only checks whether the player is situated in the revival zone or not. The player will have to move there in order for his client to start complete checking of position again.

- **FR-C27 - The client shall update the game board every 1500ms**
  The main game loop is checked and the game board updated accordingly every 1500ms.

- **FR-C28 - The client shall attempt to send the player's position every 350ms and after every fight**
  The `PositionThread` checks for positional changes every 350ms.

- **FR-C29 - The client shall only send the player position if there has been registered a change in position**
  The `PositionThread` only sends the position to the server if there has been a change in position from the last time such a message was sent.

- **FR-C30 - The client shall provide the player with an optional status information window during the game**
  A status window displaying information on score, location and support can be displayed at the top of the screen.

- **FR-C31 - The client shall take the players back to the lobby view when a game is over**
  After the support limit has been reached, all players are taken to the lobby view.

- **FR-C32 - The client shall provide the players with a restricted lobby view when a game is over**
  The after-game lobby view is passive and will not change if new players connect to the server. The players are given the option to exit or start a new game, which will refresh the lobby view.

13.1.2 Non-Functional Requirements

In this section, the client non-functional requirements presented in Section 9.2.2 are described and evaluated. Feedback from the field testing was used to evaluate the requirements concerning usability.

- NFR-C1 - **The game mechanics and goals shall be understandable to the player** *Left on his/her own device, the player shall be able to understand the goals and mechanics of the game within 10 minutes.*

All of our testing participants but one responded that the game was understood in less than 6 minutes, that person responded 6-10 minutes.

- NFR-C2 - **The client should provide ample feedback to the player** *The game should make every action the user takes known and clear to him/her through vibration, sound and visuals.*
  While visuals in the form of backlight flashes and game screen movement are provided, along with vibration, no sound is implemented in the game so far. Several testers commented this fact. However, the feedback that was provided was acceptable to all testers.

- NFR-C3 - **The game shall hide any failure from the user, and provide means of carrying on playing** *Any failure caused by internal or external events should be kept silent as long as it does not directly affect player progress. If it does, the system should present the failure in a legible manner and provide the user with options to fix/pass it.*
  Using try-catch statement faults are handled before they become visible to the user. Most faults are simply ignored by the client.

- NFR-C4 - **Important game values shall be kept in a easily reachable place** *If a developer wishes to alter game update speed, message send interval, cell phone manufacturer, network operator, player strengths and other values, these should be openly accessible at the beginning of relevant classes.*
  While not placed in a common, easily accessible class, all variables are stated at the top of their relevant classes with a brief description of role and use.

- NFR-C5 - **The game shall support alternative means of identifying location** *If a developer wishes to use map and use GSM CellID values, triangulation or even GPS, only the location specific part of the client game class needs to be altered.*
  The location fetching is implemented in a very simple manner, and the game simply checks whether a position value equals one out of five statements. One statement for each of the locations a player might be; Hospital, Ntnu, Cathedral, Limbo and Revival Zone. Altering this check value can be done through any means available, just implement a location algorithm, let it get checked by the main loop and alter the position value accordingly.

13.2 Server Requirements

This section covers all server requirements along with an evaluation.

13.2.1 Functional Requirements

The server functional requirements were stated in Section 9.3.1, this section states whether or not they were fulfilled and describes how they were implemented.

- **FR-S1 - The server shall be able to run on any computer with Internet** The server will run on any computer with Java Runtime Environment (JRE) 6.0 and an internet connection.
- **FR-S2 - The server shall allow players to join ongoing sessions at any time** The server allows players to join any ongoing session at all points during the game.
- **FR-S3 - The server shall send alive requests at regular intervals of 7000ms** The server sends alive requests every 7000ms.
- **FR-S4 - Any client not responding to alive requests shall be removed from the game session** After failing to respond to 10 alive requests, the player will be removed from the session by the server.
- **FR-S5 - If several players are present in the lobby when a game session starts, start game commands are sent to all players** A start game command is sent to all connected players when a player chooses to start the game.
- **FR-S6 - All game zones shall be set to neutral upon game start** The server resets all zones to neutral (2) when a day session is initialized.
- **FR-S7 - Any client that joins an ongoing game shall be sent an updated zone overview** Any player joining a game is sent the current game zones and a starting position.
- **FR-S8 - The server shall transmit updated player positions in batches with regular intervals** The `GameThread` keeps all connected players' positions and transmits these with regular intervals. When a player moves and transmits his new position, the server updates that player's position in the list. Every 100ms this list forms the basis of a position package that is relayed to all connected players. Only players that have moved are included in the package.
- **FR-S9 - The server shall keep track of support score and notify all connected players when the game session is over** Every time the session sends alive requests, it checks each team's support. If one has reached the support limit, a *Game over* message is sent to all players.
- **FR-S10 - The server shall allow the server administrator to change the support score limit and notify all connected players as soon as a change has been made** A slider bar is implemented in the server GUI. Changing this slider will alter the support limit accordingly. For every change, a message containing the new limit is sent to all connected players.
- **FR-S11 - The server shall close the gaming session if all players disconnect** Every time a player leaves, the server checks the number of connected players. If that number is zero, the session closes.

- **FR-S12 - The server shall release all resources and threads when a game session is over** Whenver the session closes, either due to players leaving, or the support limit being reached. The `GameThread` and `EventHandler` threads are stopped, releasing any resources held.

13.2.2 Non-Functional Requirements

In this section, the server non-functional requirements presented in Section 9.3.2 are described and evaluated. Feedback from the field testing was used to evaluate the requirements concerning usability.

- **NFR-S1 - Developers shall be able to add or change functionality in one part of the server without affecting the rest** *If developers wish to use other forms of identifying location, or using this game server for similar games, changes in functionality should not affect the integrity of the server.*
  The server remains unaffected by any change in the clients, as long as it still receives messages in the format dictated by the message parsers.

- **NFR-S2 - The server shall continue to support both TCP and UDP transport protocols** *Even if we use only TCP in our implementation, leaving the framework UDP support be will be helpful for developers exploring other aspects of location-based gaming.*
  No alterations have been made in the communication layer of both client and server.

- **NFR-S3 - Important server values shall be changeable and easily accessible** *Server values like alive request interval, support generation interval, support generation rate and similar should be presented at the top of relevant classes, as in the framework. Support limit should be directly changeable in the server GUI.*
  The server allows administrators to change support limit directly in the GUI. All other important variables are accessible, changeable and briefly described at the top of their respective classes.

- **NFR-S4 - Players shall be able to connect to the server at any time.** *The server should be listening for players trying to connect to the game during both night and day, regardless if a game is already being played or not.*
  Players wishing to join or create a game are able to connect to the server any time it is up.

- **NFR-S5 - Players shall be able to disconnect from the server at any time.** *The server should allow players to disconnect from the session at any time, and inform the other players accordingly.*
  The server allows players to disconnect at all times. A disconnection notice is sent with no waiting time involved. Other players are thereby informed of the disconnection.

- **NFR-S6 - The server shall handle any disconnection without letting it affect the game** *Even if a player disconnects during battle, the server should let the remaining player continue as usual.*
  A player may disconnect from the game during battle at no penalty, but if a player disconnects while the server is waiting for battle tactics, the other battling player will be stuck waiting to get answer from the server. This never happened during real play, because of the short time out periods of the tactics screen. But when testing the game via emulator we were able to force a disconnection as soon as the tactics screen was displayed, leaving the other player waiting.

- **NFR-S7 - The server shall handle any fault without letting the user know** *Faults may appear many places in the system, if they appear, the server should fix them or circumvent them without letting the player know they happened.*
  Using try-catch statements, like on the client, faults are handled before affecting the game. Most faults are simply ignored by the server.

- **NFR-S8 - Network data from the server shall be short and informative** *To keep the clients from chewing too much information and keep the network cost as low as possible, data messages are short and only sent when required.*
  Messages sent from the server are limited to a three-letter action identifier, a player id, a list of values and a character defining the end of the message. Several values may be sent within one message, keeping the number of messages as low as possible.

Part V

Summary

Chapter 14

Evaluation

This chapter contains an evaluation of both methods, framework and technology used. All will be desribed and analyzed based on our experiences throughout this project.

14.1 Technical evaluation

This section deals with the technologies used in this project, as described in Chapter 5. Both network and location technologies are evaluated.

14.1.1 GSM

Based on the prestudy, we knew GSM base station cells would comprise a relatively wider area than UMTS cells. This was not the largest problem, however, as each city or location will have a different set of cells of various sizes that may or may not fit the game plan. The biggest challenge was to find an appropriate way to map real-world cell areas to virtual-world zones. The process of manually mapping the GSM cells in Trondheim is relatively straightforward, but few good assoiciations to known locations could be made. Using GSM zones as a location identifier is easily implemented, but requires a relatively high effort beforehand. Though, we add, this need only be done once, and can be relieved by the help of a GPS used to connect absolute position to CellID value.

Another aspect of forcing GSM location is the reduced transfer speeds compared to UMTS. While connected to the GSM network, EDGE defines the highest achievable network data transfer rate. Despite this, the transfer speed did not impact the gameplay noticeably when testing the game. As seen in Section 11.2.1 the response times of EDGE have been measured as more or less equal to UMTS as well. This leads us to believe that if a good mapping between GSM zones and real-world locations can be made, the network technology does not restrain the options for a game or application like ours.

14.1.2 UMTS

When gathering UMTS base station CellID's, we quickly realized that they were somewhat smaller in size than the GSM cells. And not only were they a little smaller and more manageable, the also mapped more easily to real-world locations. This need not be true for every city or location, but we believe from our findings, that places chosen to be of strategic importance in an application most likely are seen as strategic locations in the real world, thus requiring a base station.

An upside with using the UMTS network to gather location was the increased transfer rates and slightly reduced response times. While not critical in our game, the vastly increased transfer speeds of HSPA might be useful in other applications.

14.1.3 CellId

As a means of locating the player, CellID is becoming increasingly easier. Now that large manufacturers like Nokia and Sony Ericsson are offering easy access to CellID through Java ME on all their newer phones, it is no longer an obstacle for developers wishing to incorporate cell location in their applications.

14.1.4 Java ME and the Framework

The experiences from using Java ME to develop the game modules of this project were mostly positive, especially since we could make use of an existing framework providing us with the means to make a multiplayer game. While getting access to network specific values like CellID and Operator at the moment requires new and relatively advanced handsets, it is positive that mobile phone manufacturers provide such options for Java developers.

Not all aspects of the game could be implemented as planned in Java ME, since the Java performance of phones vary greatly and added complexity of games require more resources. A more elaborate and synchronous battle resolve was not doable in our implementation, since adding more threads caused the game to stutter, destroying the playability of the game. This might be related to the framework already demanding a large sum of the available resources, though, and other implementations could produce better results. This only reinforces the need for strict resource management when making mobile applications. Otherwise, the framework we chose for building a location-based game worked very well. It's Java client and server are well supported by computers and mobile devices, and the framework allows for a large number of players, though we only had 5 players connected to a session at any time.

14.2 Method evaluation

This chapter deals with the methods used in this project, as described in Chapter 2. Both research and development methods are evaluated. All methods are described and analyzed based on our experiences throughout this project.

14.2.1 Research Methods

The research methods were described in Section 2.2. We mentioned five methods that we felt would help us achieve the results we needed in order to answer our research questions. Below is a list containing all methods and an evaluation of their appropriateness.

- **The Engineering Method**
  The *Engineering* method was used by implementing a location-based game in stages throughout the project. By allowing users to test the result at different points, problems were pointed out and remedied in order to create a useful application. The resulting game was vital in order to obtain results for most of our research questions.

- **Lessons Learned**
  This master thesis is based on the experiences from our depth study [ONL]. Especially at the early stages, the depth study guided our progress through this thesis. Game-specific ideas and concepts from the depth sudy were built upon to create a practical and challenging game, and helped us in answering the first two research questions.

- **The Empirical Method**
  Research question 3 required a more objective approach. Gathering data and information on the surroundings and statistics of the game using the *Empirical* method helped us achieve this. The data gathered both manually on infrastructure and automatically via handset retrieval could be analyzed and evaluated to find satifactory answers.

- **Literature Search**
  Part of this project has been to gather information on the field and state what has been and what has not been done previously. Searching related literature both on technical aspects and other games helped us understand the challenges ahead and provided us with answers to overcome these. This is especially true for both research question 1 and 2.

- **Case Study**
  Research question 5 in particular, but also most of the other questions needed a test study with actual users in order to retrieve usable data on playability. Controlled tests were performed at several stages, but mostly at the end of the project. These included answering questionnaires and direct interviews of players, and proved to be very helpful.

Together, these research methods helped us achieve acceptable answers to all of our research questions. These answers are stated in Chapter 15.

14.2.2 Development Methods

The main development method for this project was stated in chapter 3.1.3. A variant of Scrum modified for use in single-person projects was descibed as our choice. In this section we will explain the specific aspects of our method and evaluate whether or not they were useful to us in this project.

- **Project Roles**
  In a project with only one developer, roles aren't as definite as they would be in a proper Scrum setting. The project supervisors functioned loosely as Project Owners, and hinted at the desired outcome of our project. The developer acted as both Scrum Master and Scrum Team, with a strong connection to the code, the project environment and the outside stakeholders. While this juggling of responsibilities is shunned in Scrum, it is quite manageable for smaller projects, such as this one, and doesn't have too much of an impact on neither progress nor outcome.

- **Backlogs**
  A better example of Scrum a practice that is useful in solo projects is the backlog idea. Developing a prioritized list of outstanding requirements or features was very valuable in the sense of managing the project progress and scope. ScrumWorks helped keep all tasks and bugs visible and ordered, avoiding the ocean of yellow post-its that may otherwise have appeared in such a setting. Because of the problems described in Section 12, we didn't get to document the use of ScrumWorks, but luckily this happened at such a late stage that most value had been derived from the process already.

- **User Stories**
  User stories define a different approach at requirement gathering than the traditional specification. Since very few demands were made from outside the project environment, inventing and decomposing user stories was a good way of getting to the core of functionality and properties that should be offered by the system.

- **Sprints**
  One of the main ideas in agile methodology is incremental development. We had increments or sprints covering 14 days, in this time period we would figure out which functionality to add, implement it, document it and try to test it. This way, we could always keep focus on the most desired functionality and receive continual feedback on the progress.

- **Sprint Retrospectives**
  Having meetings with oneself may not sound all that productive, and most daily scrums were in fact dropped. But adding retrospectives after finishing a sprint is still useful. Fleshing out what went right and what went wrong provides a morale boost or strong incentive to change towards the next sprint. Connecting these problems and ideas to the project via ScrumWorks also helped taking them into consideration at later stages.

- **Adaptability**
  Another main idea of agile development is adapting to change. Sticking with a plan made before anything is tried and tested may be a large hinder, even in solo projects, where different responsibilities are filled by the same person. As change happened in our project, so did priorities. ScrumWorks was used to manage priorities by click and drag at all points. At the same time, documentation was updated, keeping most of the project report alive throughout the project.

The development practices of Scrum proved imperative in keeping track of project status and managing details. Scrum provided us with an easy way of progressing through the different stages of the development in a flexible and lucid manner.

On the other hand, in this small project the Scrum process and ScrumWorks demanded a substantial share of time spent on both study and setup in order for us to get value from it. However, after having attained the required knowledge, we see no problem in using Scrum for projects of all sizes in the future.

Chapter 15

Research questions answered

In chapter 2 we stated the research questions of this project. This chapter provides answers to these questions.

- **RQ 1 - Which challenges exist when developing location-based games for mobile phones?**

  This research question was asked to identify the location-specific challenges related to developing mobile games. The answers were gathered from our own experiences with developing a location-based mobile game.

  A key aspect when making a location-based game is deciding which role location should play. Should it be a supporting factor controlling weather, fellow players and background? Or should it be a direct factor replacing the directional keys of you phone? Naturally, one may decide to use both approaches. The decision will largely affect the type of players your game attracts. Games absolutely requiring physical motion seems to appeal to a particular group of players, players who wish physical ability to affect the outcome of the game. Much like strategic game players wish for other decision-factors than first-person shooter gamers. Motion-based games are also more restricted in terms of gameplay flexibility. They cannot be played everywhere and demand some planning ahead. Games using location as a supporting factor are more able to mask the lack of some special location when providing game challenges, meaning that you can almost always play the game and progress in your own house, only needing to go somewhere when you want something special to happen.

  Another aspect of location is handset support. Location retrieval is still an option reserved for a relatively small subset of mobile devices. GPS is the most

known way of getting mobile location, but though more and more handsets support GPS they are still few and far between. Especially in the target group for game applications. Simpler ways of identifying location, like CellID are common to all handsets, but cannot be retrieved in all devices, limiting this technique in much the same way as GPS. A larger number of phones support this method, though, including all Symbian phones and all Java phones in the newest generations from Nokia and Sony Ericsson[1]. Triangulation remains a method still requiring operator support.

In terms of playtesting, getting players out in the field as soon as possible is an advantage. Field testing a location-based game, especially one relying on locations with some distance apart, is time-consuming. And testing is vital; a bug free environment should be considered especially important, since the game continues even if a player gets disconnected, potentially interfering with the gameplay of other connected players.

- **RQ 2 - What is the impact of adding an online multiplayer client/server-structure to a location-based game?**

  This question was formed to provide information related to the more social aspect of mobile gaming.

  An often-mentioned factor in location-based gaming is the ability to play with nearby players. This is even more emphasized in active games like CityZombie. Players remark that since such a game competes with the exercise of a game of hide and seek or a football match, the social aspect is important and should be included to keep the fun-factor high.

  To facilitate a social game of this scale, a server/client-approach should be adopted. Allowing users to connect and disconnect as they please. With online play, cost becomes a vital factor for players. Server and client communication should be designed in a way that gives maximum value for the amount of data transfered via the network. If successfully implemented, the competitive nature of active social games can be kept and enhanced in a virtual game through the mobile network.

- **RQ 3 - What basis does the current infrastructure offer to support a location-based mobile game?**

  This question adresses the various infrastructural challenges that exist when developing a location-based game. While GPS is well-tested and active as a means of locating mobile phones, other methods haven't been as thoroughly explored.

  The GSM and UMTS networks both provide mobile phones with a CellID unique

[1] Nokia S40 and S60 FP 1 & 2 and SE JP-8

to the base station the phone is currently connected to. The area covered by such base stations vary, but in more urban areas, a radius of 100-500 meters may be considered normal. These areas provide a natural zone structure which developers may utilize in games and applications. Mapping these coverage areas to the map is a somewhat time-consuming and cumbersome task, but needs only be done once, and can easily be reused by others.

The current infrastructure is very operator-dependant, as each network operator provides their own base stations and naming conventions. Cross-mapping zones between network operators is in many cases not feasible, and will have to be evaluated dependending on each location. If such a mapping can be done, discerning between operators is programatically an easy task, allowing all players to participate on equal terms.

The differences between UMTS and GSM network cells are noticeable. Both in terms of cell area size and CellId syntax. Cross mapping UMTS and GSM cells and cross mapping between different operators can be considered equally hard. UMTS is the better choice, despite a somewhat lower coverage than GSM. This is due to a larger number of smaller zones, making it more accurate, and UMTS coverage in urban areas is good enough to mostly ignore the issue.

- **RQ 4 - Which technologies support the continued existence of a location-based multiplayer mobile game the best?**

  In terms of playability and technical properties, there is a difference between the GSM and UMTS networks. UMTS offers higher bandwidth and better response times. The EDGE technology provides GSM with a response time and bandwidth that is acceptable for most games, however. There is no difference in cost between the different technologies, as the same amount of data is transfered in both cases. In games such as ours, where demands for response times aren't crucial, both networks offer equal playability.

- **RQ 5 - How do people respond to a location-based game as opposite to a passive/stationary game?**

  This question is asked to gather player knowledge and attitude towards location-based mobile gaming.

  The thing that strikes us first is the lack of knowledge about location-based gaming. This may be attributed to the fact that Norway hasn't had a culture for developing games for mobile phones. Our neighbouring countries Sweden and Denmark were both pioneers in terms of exploring location and gaming on mobile phones with Botfighters [BOT] and ZoneMaster [ZON], early games using CellID location.

When confronted with a location-based game players are mostly positive, as it provides their phones with yet another feature that has remained unused thus far. There is a strong opinion, however, that these kinds of games are not time-killers such as the games they have played before. Actively having to visit places to advance the gameplay is a concept best received with players that see the game as a variation of other social games played outside. In this context, in-game communication becomes important. When spread out relatively far apart, players want a way of communicating with their fellow team members to organize themselves.

Chapter 16

Conclusion

The goal of this master thesis was to design and implement an online location-based game in order to decide which programmatical-, infrastructural- and playability-related factors that support this in the best possible manner. Real-world testing of the game was performed in order to evaluate the result in a realistic setting.

We started by introducing the different concepts associated with mobile gaming in general and location in particular. After evaluating these concepts, we introduced CityZombie, an online multiplayer mobile game relying on CellID location as the main input factor. This game provided us with the means to test both the GSM and UMTS infrastructure and the related EDGE and HSPA network technologies as well as the general knowledge and attitude towards location-based gaming in a test group.

In the process of identifying and mapping the GSM and UMTS infrastructure of the two major operators, we discovered differences both in naming conventions, base station cell sizes, base station locations and signal stability. All these factors influence a location-based game. Firstly, the differences between GSM and UMTS are equal to the differences between operators. GSM and UMTS cells have different sizes, with UMTS cells being somewhat smaller because of their higher frequency and lower output strength. This is in most cases not important, since the number of cells of both kind are sufficient to create an interesting zone structure. More important, and difficult, is the process of relating these cells to real-world areas. Most zones overlap, and while many locations offer stable connection to one base station, other locations will force the phone to jump between two cells. Finding stable areas to mark as game zones is vital. Related to this, is connecting the zones to real-world locations that are easily recognizable. A stable, well sized zone is sometimes useless if it cannot be related to a game zone in an intelligeble manner. This process effectively eliminates both zones, networks and operators from the equation.

We ended up implementing a game with Telenor UMTS zones based on St. Olavs Hospital, The Nidaros Cathedral and NTNU Gløshaugen. While Telenor GSM zones were plentiful in most areas, they were far more overlapping, and a direct mapping between

the GSM and UMTS zones could not be made. Netcom GSM and UMTS zones were also suffering from overlapping problems, in addition to bad mapping with the three real-world locations. This solution may not be the best in all areas, though, and an evaluation of operator and network appropiateness should be made when implementing this kind of game elsewhere.

Though incompatible in location, the GSM and UMTS networks both provide acceptable playability in the form of response times and data transfer via EDGE and UMTS/HSPA. While being a real-time game, CityZombie does not demand very high response times, since input speed is limited to the player's ability to move between zones. The cost of playing the game is the same in both networks, and is relatively low because of the slow rate of player input.

In general, the location-based game concept was well received among players. Likely because it represents something new; most players are still unaware of the possibilities in location-based gaming. Making the game multiplayer was another another positive factor, as many players feel that an active outdoor game has much in common with a virtual football match or a game of hide and seek, making the social aspect important and relevant. When testing the game with actual players, the main concerns were related to communication options, content and the amount of physical activity. The distances involved in CityZombie made the game more appealing to players interested in letting physical ability decide the outcome of the game, even though the game facilitates both passive and offensive roles. This result can to some degree be attributed to the relatively small size of the game and the lack of content.

Chapter 17

Further work

This chapter describes issues and properties that could be further explored within the scope of this project.

17.1 Extending the game

Though CityZombie was built as a concept for exploring possibilites in location-based gaming, the game itself and the surrounding concept could be enhanced by extending the game.

Adding content

In Section 8.1, we mentioned game content like mines and robots, used to defend and attack zones as proxies for actual players. Similar objects could be strength potions and weapons regulating the balance between the players. These items could be placed on the map before game start or generated by the server and then placed randomly. The framework used supports random generation of items, and adding this functionality into the existing game would not be hard.

Other random events could also occur, depending on location and time. Making the players feel they play in a dynamic world. Adding an in-game clock to record events could be utilized in many ways, ranging from the cathedral striking it's bells every in-game hour when you are present in that zone, to the hospital increasing your hit points every in-game hour when in that zone.

Expanding the playground

Presently, the game is relatively small, with only 3 zones to occupy. Adding to this number would give both a larger number of players something to do and make the game truly mirror the city in which it exists. A scrolling map could help keep the present overview at the current level and at the same time make the game boundaries larger.

There is no theoretical limit to how many zones a game could have, but the number of players would dictate the practicality of this to a large extent. Having two persons fight over 20 zones in a game covering ten kilometers in diameter might be a bit unrewarding to some.

Combat

As mentioned in Chapter 8, A more elaborate combat was envisioned when players attacked each other in zones. Giving the players the ability to use keypads to control their own actions during combat is desireable, from a playability perspective. The framework might have to be modified more in order to achieve acceptable response from the phone, but given time it is not an unimaginable task. If such a task proves impossible, combat could be implemented in a real turn-based manner. This might be a rock-scissor-paper strategy over several rounds, where the winner is best of three. Or letting the client send a value to the server based on the player's ability to perform some precision feat locally, maybe clicking a combination of buttons at the exact right times. The server would then see which player performed best and decide the winner. Any method that provides players with the ability to show skill would be good.

17.2 Testing

Part of this project was to perform playtesting, as location-based gaming is a relatively new genre on the market. However, with only two mobile phones supporting the game available, testing was limited to one on one combat and combinations with 2 actual players and some players using emulators. Most testers responded that the game would suit best for teams of 2-5 player, meaning as much as 10 phones in the field at the same time. This would give the players the option of taking offensive or defensive roles, some guarding zones and others attacking the enemy. Developers would also have the ability to test the framework and game's capability to support more players.

Bibliography

[ACM] Hartel, Pieter H. and Moreau, Luc . *Formalising the Safety of Java, the Java Virtual Machine and Java Card.*
Univ. of Southampton & Univ. of Twente. 2001.
citeseer.ist.pdu.edu/502446.html

[ACP] Miller, Ross. *Plundr, first location-based DS game, debuts at Where 2.0.*
Article, Joystiq. 4 June 2007.
www.joystiq.com/2007/06/04
/plundr-first-location-based-ds-game-debuts-at-where-2-0/

[AGI] Schwaber, Ken, Highsmith, Jim et.al. *The Agile Manifesto.*
Utah, USA. 11-13 February, 2001.
www.agilemanifesto.org

[ANI] Buchanan, Levi . *The Shroud.*
IGN Wireless . 8 June, 2006.
wireless.ign.com/articles/711/711888p1.html

[BAS] Basili, Victor R. *The Experimental Paradigm in Software Engineering.*
Lecture Notes in Computer Science. Springer-Verlag. 1992.

[BCK] Bass, Len., Clements, Paul and Kazman, Rick . *Software Architecture in Practice.*
Addison-Wesley, 2.ed. 2003.

[BLU] Bluetooth SIG Inc. *Bluetooth.*
Official homepage. 2008.
www.bluetooth.com

[BRE] Qualcomm Inc . *Brew: Bring wireless to life.*
Brew Homepage . 2008.
brew.qualcomm.com/brew/en/

[BMG] Sony BMG. *The Shroud.*
Sony BMG Games . 2007.
games.sonybmg.com/theShroud.html

[BOT] Baron, Paul . *Location Based Mobile Games.*
In-Duce.net. November 2004.
www.in-duce.net/archives/locationbased_mobile_phone_games.php

[CHA] Chau, Fiona . *Mobile Gaming Aims For Mass Market.*
Wireless Asia. 19 September, 2006.
www.telecomasia.net/article.php?id_article=1744&page=5

[CPN] NetCom. *Netcom CPA.*
Netcom Homepage . 2008.
netcom.no/omnetcom/partnere/cpa-innholdsleverandorer.html

[CQW] Vinh, T. *High-tech hunt nets a $5,000 prize.*
Article, Seattle Times. 24 October 2004.
homepages.nyu.edu/d̃c788/conqwest/press/seattletimes/

[CSP] Ferner, Carl J. *Cellspotting.*
Official homepage. 2008.
www.cellspotting.com

[DQI] Nguyen, T. H., Ekholm, J. and Ingelbrecht, N. *Dataquest Insight: More Growth Ahead for Mobile Gaming.*
Gartner Inc. Analysis. 2 May 2007
www.gartner.com/DisplayDocument?id=504622&ref=g_sitelink

[DTS] Datatilsynet. *Krav om samtykke for bruk av lokaliseringsdata.*
Letter from Datatilsynet to Telenor and Telia. 3. february 2005.
www.datatilsynet.no/templates/Page____874.aspx

[EBN] Garshol, Lars M. *BNF and EBNF: What are they and how do they work?.*
Personal Homepage. 21 July, 2003.
www.garshol.priv.no/download/text/bnf.html

[EKO] Post- og Teletilsynet. *Det norske ekommarkedet 2007.*
NPT. Lillesand, Norway. 27 May 2008.
http://www.npt.no/iKnowBase/Content/107094/Ekommarked_2007.pdf

[ELI] Elisa *Elisa kolminkertatistaa 3G-datanopeudet.*
Elisa.fi. 30 August, 2007.
www.elisa.fi/ir/pressi/index.cfm?t=100&o=5120.00&did=14237

[ERA] Watagame . *Era Of Eidolon.*
Official Homepage . 2004.
www.eraofeidolon.com

[FFA] Square Enix Co Ltd. *Final Fantasy VII: Crisis Core.*
Square Enix Homepage . 2007.
www.square-enix.co.jp/ccff7/

[GML] Hillebrand, F. *Milestones of the GSM/UMTS Development.*
3GPP. 11 March 2007.

www.3gpp.org/ftp/workshop/2007-03-14_20%20Years%20of%20GSM
/Presentation/07_GSM_UMTS_milestones_070311.pdf

[GPB] Umts Forum *Past the 200 million milestone: 3G/UMTS Grow.*
Umts Forum. 30 January, 2008.
www.umts-forum.org/content/view/2315/170/

[GPR] Michel, Dirk and Ramasarma, Nathan . *GPRS Measurement Methodologies and Performance Characterization for the Railway Environment.*
Wireless Communications and Network Conference. 2005.
ieeexplore.ieee.org/iel5/9744/30730/01424764

[GPS] Pellerin, Cheryl. *United States Updates Global Positioning System Technology.*
Washington File. 3 February 2006.
usinfo.state.gov/xarchives/display.html?p=washfile-english
&x=200602031259281cnirellep0.5061609

[GSB] GSM Association. *GSM Facts and Figures.*
GSM World. 28 January 2008.
www.gsmworld.com/news/statistics/index.shtml

[HIS] Farley, Tom . *Mobile Phone History.*
Phone Warehouse. 20 February, 2002.
affordablephones.net/HistoryMobile.htm

[IMP] Ollila, Elina M.I . *Improving the end user experience: Advances in playtesting for mobile games.*
Game Developers Conference Presentation . 2008.

[IPH] Wingfield, Nick and Sharma, Amol . *iPhone 'Surfing' on AT&T Network isn't Fast, Jobs Concedes.*
Wall Street Journal. 29 June, 2007.
online.wsj.com/article/SB118306134626851922.html

[JCP] Majakangas, Jaana . *JSR 293: Location API 2.0.*
Java Community Process. 2008.
jcp.org/en/jsr/detail?id=293

[JDB] Jonell, Jonas and Dahlberg, Gustav . *Framtiden för mobile spel-teknologispelarenaffärsmodeller.*
Diploma at the University of Gothenburg, School of Economics. 2003.

[JDK] SUN *Java SE 6 Features and Enhancements.*
SUN website. 2008.
java.sun.com/javase/6/webnotes/features.html

[KDD] KDDI Corporation . *KDDI.*
Official Homepage . 2008.
www.au.kddi.com/english/index.html

[LAM] Privat, Ludovic . *Multiplayer location-based gaming gets real: Interview with La Mosca.*
GPS Business News. 29 January, 2008.
www.gpsbusinessnews.com/index.php?action=article&numero=627

[LBS] R&D Neferi Team. *Entertainment Location-Based Services: the nest killer-app?.*
Report. 30 November, 2006.
www.scribd.com/doc/2573421/locationwhitepaper

[LOC] Srivastava, M. *Location Sensing for Context-Aware Applications.*
Lecture in EE233C, UCLA. 30 May 2000.

[LTB] Lomas, Natasha . *Location-based services to boom in 2008.*
Businessweek. 11 February, 2008.
www.businessweek.com/globalquiz/content/feb2008/qb20080211_420894.htm

[LOA] SUN Microsystems. *The Java ME Device Table.*
Sun Developer Network. 2008.
developer.sun.com/mobility/device/pub/device/list.do?sort=manufacturer
&filterIds=125

[MGW] Morton, Derrick and Wisniewski, Donald . *2005 Mobile Games White Paper, Game Developers Conference 2005.*
IDGA Online Games SIG. August 2005.
http://www.igda.org/online/IGDA_Mobile_Whitepaper_2005.pdf

[MIK] Schenk, C . *MiKTeX.*
Project website. USA. 2008.
www.miktex.org

[MJG] Fox, David, Verhovsek, Roman . *Micro Java Game Development.*
Addison-Wesley. Boston, USA. 28 April, 2002.

[MOG] Newt Games . *Official Mogi Homepage.*
Mogimogi.com . 2003.
www.mogimogi.com

[MOS] Hall, Justin . *Mogi: Second Generation Location-Based Gaming.*
The Feature Archives . 1 April, 2004.
www.thefeaturearchives.com/100501.html

[MPG] funSMS.net. *Mobile Phone Generations.*
funSMS.net . 2006.
www.funsms.net/mobile_phone_generations.htm

[MZW] Zelkowitz, Marvin V. and Wallace, Doris L. *Experimental Models for Validating Technology.*
IEEE Computer 31(5). May, 1998.

[NIC] Vaughan-Nichols, Steven J. *The Challenge of Wi-Fi Roaming.*
IEEE Computer 36(7). 2003

[NOR] Wikipedia. *List of Deployed UMTS Networks.*
Wikipedia entry. April, 2008.
en.wikipedia.org/wiki/List_of_Deployed_UMTS_networks

[NPM] Nicklas, D., Pfisterer, Ch. and Mitschang, B. *Towards Location-based Games.*
University of Stuttgart, Institute of Parallel and Distributed High-Performance Systems (IPVR). 2001.
www.adcog.org/adcog21/adcog21.pdf

[NPR] NetCom. *NetCom Data Prices.*
Official homepage. 2008.
netcom.no/priser.html

[NSE] Nokia. *Getting CellID in Java ME.*
Forum Nokia Developer Community. 14 May, 2008.
wiki.forum.nokia.com/index.php/CS000947_-_Getting_Cell_ID_in_Java_ME

[ONL] Rolland, Øyvind . *Online Location-based Mobile Gaming.*
NTNU. Trondheim, Norway. 18 December 2007.

[OSA] Grimen, G. et.al. *Service Context and Information Content in Mobile Information Services.*
Proceedings of the 12th IST Summit on Mobil and Wireless Communications. Aveiro, Portugal. June, 2003.

[OTD] Alsnes, R. *Location Aware Services.*
Master's Thesis, NTNU. 2003.

[PAR] Park Associates. *Electronic Gaming In The Digital Home.*
Survey. September, 2006.
www.parkassociates.com/research/reports/tocs/2006/multi-gaming.htm

[PEL] Pelkonen, Tommi . *Mobile Games, E-content Report 3.*
Anticipating Content Technology Need. February, 2004.

[PER] Helal, Sumi . *Pervasive Java.*
Pervasive Computing. January-March, 2002.

[PIR] Bjørk, S., Falk, J. Hansson, et.al. *Pirates! The Physical World as a game board.*
IOS Press. Tokyo, Japan. 9-13 July 2001.
www.viktoria.se/fal/publications/play/2001/pirates.interact.pdf

[PON] Winter, David . *PONG-Story.*
PONG-Story website. 2008.
www.pong-story.com

[POW] Powers, Michael . *Real-Time Constraints.*
Mobile Multiplayer Gaming, Part 1. 2006.
developers.sun.com/techtopics/mobility/midp/articles/gamepart1/

[PRF] Halonen, Timo, Romero, Javier and Melero, Juan . *GSM, GPRS and EDGE Performance - Evolution Towards 3G/UMTS.*
Wiley, 2 ed. 2003.

[RTG] Jarrett, Martin and Sorteberg, Eivind . *Real-Time Online Multiplayer Mobile Gaming.*
Master's thesis, NTNU . 2007.

[RUM] Wikipedia . *Wikipedia Rumble Pak entry.*
Wikipedia. 12 April, 2008.
en.wikipedia.org/wiki/Rumble_pak

[SCR] Kniberg, Henrik . *Scrum and XP from the Trenches.*
C4Media. USA. 2007.
www.infoq.com/minibooks/scrum-xp-from-the-trenches

[SEJ] Sony Ericsson. *Java ME Platform Docs and Tools.*
Sony Ericsson Developer World. 2008.
developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

[SCW] Danube Technologies Inc. *Scrum Lifecycle Management Tools.*
Danube website. 2008.
www.danube.com/scrumworks

[SEM] Semacode Corp. *Sem@code.*
Official homepage. 2008.
www.semacode.com

[SHO] Wikipedia . *Wikipedia Dualshock entry.*
Wikipedia. 6 June, 2008.
en.wikipedia.org/wiki/Dualshock

[SHR] Your World Games . *The Shroud.*
Official Homepage . 2006.
www.shroudgame.com

[SJP] Sony Ericsson. *Java Platform Versions and Screen Sizes.*
Sun Developer Network. 28 January, 2008.
developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

[SMO] Davis, Justin . *The Shroud's GPS Functions Revealed.*
Modojo . 10 March, 2006.
www.modojo.com/features/20060316/55/

[SMP] Lakeworks *Wikipedia Scrum Entry.*
Wikimedia Commons. 2008.
en.wikipedia.org/wiki/Image:Scrum_process.svg

[SNA] Dance With Shadows. *The man who made snake.*
DanceWithShadows.com . 19 June, 2005.
www.dancewithshadows.com/tech/snake-nokia-game.asp

[SPW] Parish, Jeremy . *Classic.1up.com's Essential 50.*
1UP.com . 12 January, 2005.
www.1up.com/do/feature?cId=3116291

[SSC] Bell, Peter . *Solo Scrums.*
Blog entry, Personal Homepage. 17 June 2007.
www.pbell.com/index.cfm/2007/6/17/Solo-Scrums

[STE] Stensrud, Geir . *Mobile lokasjonsbaserte dataspill: ny dataspillgenre?.*
Insitutt for Medier og Kommunikasjon. February, 2006.
www.media.uio.no/prosjekter/internettiendring/downloads/stensrud.pdf

[TAR] LaMosca. *About The Target.*
Official Homepage. 2008.
www.lamosca.be/thetarget_company.htm

[TLP] Tele2. *Tele2 Data Prices.*
Official homepage. 2008.
www.tele2.no/privat/mobil/priser/datatrafikk/

[TOP] Topley, Kim . *J2ME in a Nutshell.*
O'Reilly. 2002.

[TPR] Telenor ASA. *Telenor Data Prices.*
Official homepage. 2008.
telenor.no/privat/mobil/priser

[TXC] ToolsCenter.org . *ToolsCenter.org.*
ToolsCenter website. USA. 2008.
www.texniccenter.sourceforge.net

[USR] Cohn, Mike . *User Stories Applied: For Agile Software Development.*
Addison-Wesley. Boston, USA. 11 March, 2004.

[WLA] A QUALCOMM Company Snaptrack. *Location technologies for GSM, GPRS and UMTS networks.*

Snaptrack. 2003.
www.snaptrack.com/pdf/Snaptrack_Advantage.pdf

[WOW] Blizzard Inc. *World Of Warcraft.*

Official Homepage. 2008.
www.worldofwarcraft.com

[ZON] Unwiredfactory. *ZoneMaster.*

Official homepage. 2002.
www.zonemaster.myorange.dk

Part VI

Appendix

Appendix A

CellID infrastructure

In Figure A.1 below, a GSM grid is illustrated in a landscape where a road passes from an imaginary town (A) with a relative dense population into a highway area (B) with very scarce population. Weaker antenna power is used in urban areas, this is done to make room for more base stations with less interference. More base stations means higher total capacity, since the population is split into several zones.

Figure A.1: A CellID grid.

Appendix B

Running CityZombie

This chapter provides information on how to install and run the CityZombie client on a mobile phone, and the ZombieServer on a computer.

B.1 Running the server

This section contains a brief explanation on how to install and run the ZombieServer application.

To install the server, just copy ZombieServer.jar to your computer. The file is self-contained and needs no special environment in which to run.

To run the server, you will need Java Runtime Environment 6.0. If you don't already have it installed, it can be dowloaded for free from
`http://java.sun.com/javase/downloads/index.jsp`

To start the default server running on port 3724, you may simply double-click the ZombieServer.jar file. There are two disadvantages here. Firstly, your computer firewall may be blocking the port, not allowing any connections. Second, you will not see console messages like players joining your game. If you wish to decide port number yourself or wish to see the console messags, please start the ZombieServer.jar file in a console window. In Microsoft Windows, console windows can be opened by pressing Start->Run, then type "cmd" in the run-window and press ok/enter. Inside the console window, navigate to the directory where ZombieServer is located and type:

`java-jarZombieServer.jar` -to open a console server window on port 3724, or

`java-jarZombieServer.jar <portnumber>` -to open a console server with dedicated port.

B.2 Running the client

This section contains a brief explanation on how to install CityZombie on your mobile phone. Note that this program will not work properly unless you have a 320x240 screen resolution. Location will not work unless you have a Sony Ericsson JP-7.3 or newer phone.

To run the client on a mobile phone, the files CityZombie.jad and CityZombie.jar needs to be installed on the phone. Make sure you transfer the the files from "Client for phone testing", otherwise location will not work. We recommend you transfer the files to the mobile phone via memory card, usb cable or Bluetooth. After having copied the files to your phone, simply press CityZombie.jar to install the game. From now on, the game can be launched from the folder you installed it to.

Appendix C

Files

This appendix explains the catalog structure in the file attachment.

C.1 Class Diagrams

This folder contains the class diagrams for the server and client architecture.

C.1.1 Client

This subfolder contains all class diagrams for the CityZombie client.

C.1.2 Server

This subfolder contains all class diagrams for the CityZombie server.

C.2 Applications

This folder contains the runnable Java applications.

C.2.1 Client for phone

This subfolder contains the CityZombie.jad and CityZombie.jar files meant for phone testing, will not work on emulator.

C.2.2 Client for emulator

This subfolder contains the CityZombie.jad and CityZombie.jar files meant for emulator testing, will not work on phone. Has a dummy variable attached as CellID value to avoid *NullPointerException*.

C.2.3 Server

This subfolder contains the ZombieServer.jar file.

C.3 Source Code

This folder contains the source code for the developed client and server.

C.3.1 Client

This subfolder contains the game client source code.

C.3.2 Server

This subfolder contains the game server source code.

C.4 Javadoc

This folder contains javadoc for the client and server.