Teazle Goes Mobile

Christina L. Nilsson
Hua Dong
Ningjing Chen
Anders Söderlund
Stefan Pettersson
Simon Lundmark
Joakim Hägglund
Anders Andersson
Christoffer Ramqvist
Sassan Ashkan Far

January 11, 2007

Contents

Abstract

This document concerns the distributed mobile phone game Teazle Goes Mobile created by ten computer science students at Uppsala University during their project course in the fall of 2006.

1 Introduction

Teazle is a classic computer board game from 1997. The purpose of Teazle goes Mobile (TGM) is to remake the computer game for mobile phones. Together with the creator of Teazle, Aniware AB, ten computer science students took on this task. The project was part of a project course in the students' fourth year of computer science at Uppsala University, Sweden. The goal of the project course was to give the students experience from working in a large software development project and learn how to create a complex distributed system.

The big challenge was to create a distributed mobile phone game. Just like the old Teazle game, the new game supports online gaming. Since mobile phones have several limitations this however was not easy to solve. Most mobile phones are still extremely primitive compared to todays desktop computers. Memory and speed are the two main problems when developing games for mobile phones.

2 Preliminares

2.1 Phone hardware

The target platform for the game is the Sony Ericsson mobile phone. The game is developed and tested for five Sony Ericsson models: K310i, K500i, K700i, W810i and K800i. All selected mobile phones support Mobile Information Device Profile 2.0 (MIDP) Java environment, Wireless Application Protocol 2.0 (WAP) and Connected Limited Device Configuration 1.1 (CLDC). All phones except K500i have Bluetooth API connections. They have a screen sizes of 128x160, 176x220 or 240x330 pixels and a possible heap size of 512 kB to 3 MB depending on phone model. Heap size also depends on available phone memory which is at least 15 MB. The keypad on the phone consists of the standard 0 to 9 number keys, * and #. It also contains a joystick, three soft buttons, a cancel button and a back button.

2.2 J2ME

Java 2 Micro Edition (J2ME) [1] allows developers to use Java and the J2ME wireless toolkit to create applications and programs for cell phones. They can be emulated on a PC during the development stage and easily uploaded to the phone. The technology allows portability of applications between platforms. The technology is open for anyone to use which gives lots of help on the Internet.

2.3 J2ME Polish

J2ME Polish [2] is an open source Ant-based tool for the development of wireless Java applications. It covers the whole circle of preprocessing, compiling, obfuscation, packaging and JAD-creation. J2ME Polish is ideal for creating device optimized applications with its powerful preprocessing capabilities and the integrated device database. With J2ME Polish no hardcoded values are needed and the portability of an application is not sacrificed, even though highly optimized applications are created from a single source.

2.4 Erlang

Erlang [3] is a general-purpose programming language and runtime environment. Erlang has built-in support for concurrency, distribution and fault tolerance. The sequential subset of Erlang is a functional language, with strict evaluation, single assignment, and dynamic typing. It supports hot swapping so code can be changed without stopping a system. Erlang was originally a proprietary language within Ericsson, but was released as open source in 1998.

2.5 GPRS

General Packet Radio Services [4] is a communications standard enabling packet data transfer for users of Global System for Mobile Communications (GSM). GPRS can be utilized for services such as WAP access, SMS and MMS, but also for Internet communication services. GPRS is a connectivity solution based on Internet Protocols which can be done using a socket connection set up in J2ME. The main advantage of GPRS is that a mobile phone can access the Internet from any location as long as it has a sufficient signal strength from the GSM base station but has disadvantages of low transfer rates.

3 The game Teazle goes Mobile

TGM can be described as a complex Tic-Tac-Toe or 4-in-a-row game. Instead of just putting 'x's and 'o's on the board each tile holds a minigame. The board consists of 4x4 tiles and each tile contains a random arcade game. To own a tile one must beat the tilegame's current highscore. This means that the owning of tiles is dynamic. The winner is the player that first owns 4 tiles in a row.

TGM can be played in three modes: single player, Hot-Seat and online. In single player there is no game board, the user instead plays the different tilegames. Hot-Seat is a form of multiplayer mode where only one mobile phone is used. Up to six players can play together on a single phone. The phone is passed around in a turn based fashion. The player's turn is decided by time, the one who has the least total playtime is next to play. TGM also supports online gaming where the users can create or join existing games. To play online the user must register an account on the web portal and pay a monthly fee. The game itself is free to download.

3.1 Teazle vs TGM

The main difference between the old Teazle and TGM is that TGM requires 4-in-a-row instead of the old 5-in-a-row. The only reason for this change is the mobile phones small screen sizes. During this project there was only time to create two tilegames, Breakout and Clock. In the original Teazle there were 50 different arcade games.

3.2 The tilegames

Breakout is a Pong-like game where the user hits a ball with a paddle that moves sideways. The point is to make the ball hit a set of bricks above. When a brick is hit it disappears, and when all bricks are gone the user have completed the level. For each brick the player gets 1 point. Also when a brick is hit it can drop an item that changes the game a little for a few seconds. The paddle can for example become longer or smaller depending on the item.

Clock is a simple game where the player gathers score only by staying in the game. One second equals 1 point. The game requires no interaction. The user simply quits when he/she feels it is appropriate.

4 System Description

The system consists of three main parts: the server, the client and the web portal. Together, these parts constitute a complex distributed system in the form of a mobile phone game.

4.1 The Server Solution

4.1.1 Server Structure

The server solution is designed with two main properties in mind. One of the properties is scalability: it should be possible to increase its ability to handle more client requests by adding additional physical servers. In this way, all the physical servers can balance the client requests and handle them fast. The second property is transparency: the entire server solution should appear as a single machine to the client. The user does not need to know that there are several physical servers, and will see the entire server solution as a single big server, and can join games that are hosted on different physical servers.

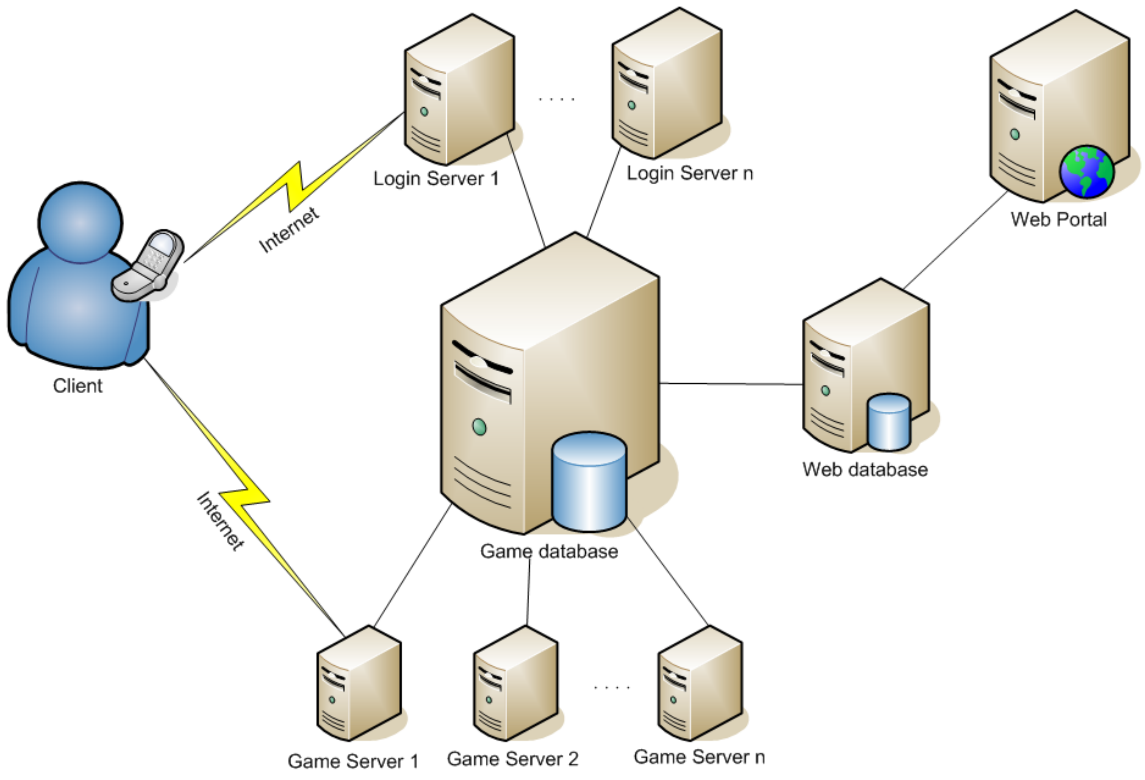

Figure 1: Server structure

There are three different types of servers: login servers, game servers and database servers. The login server acts as the interface between a game server and a client. It handles the intitial client login, and when the client is logged in, it can send requests for the lists of joinable games. When a new login server is added or old ones are removed, the current login server (the one that the client is currently logged on to) will automaticly send a new login server list to the client. This means that the user will not type in a login server address to log into the server. The client will automatically send login infomation (username & password) to a free login server. From the login server, the user can choose to join or create a game, after which he/she is redirected to the correct game server.

The bottleneck of the server solution is the database server. Since the server solution should appear as a single big server, all the physical servers must know of each other in some way. Hence, all the essential information such as the physical server address, status, gamestatus and their hostnames must be stored in a database that can be accessed from all the servers. A linux machine running Mysql 5.0 was chosen as the database server.

4.1.2 Server Design

Since the server should be able to handle multiple client connections, the server is designed in such way that for each client connection, a process is spawned to handle the incomming client request. In terms of complexity, the login server is simpler than the game server.

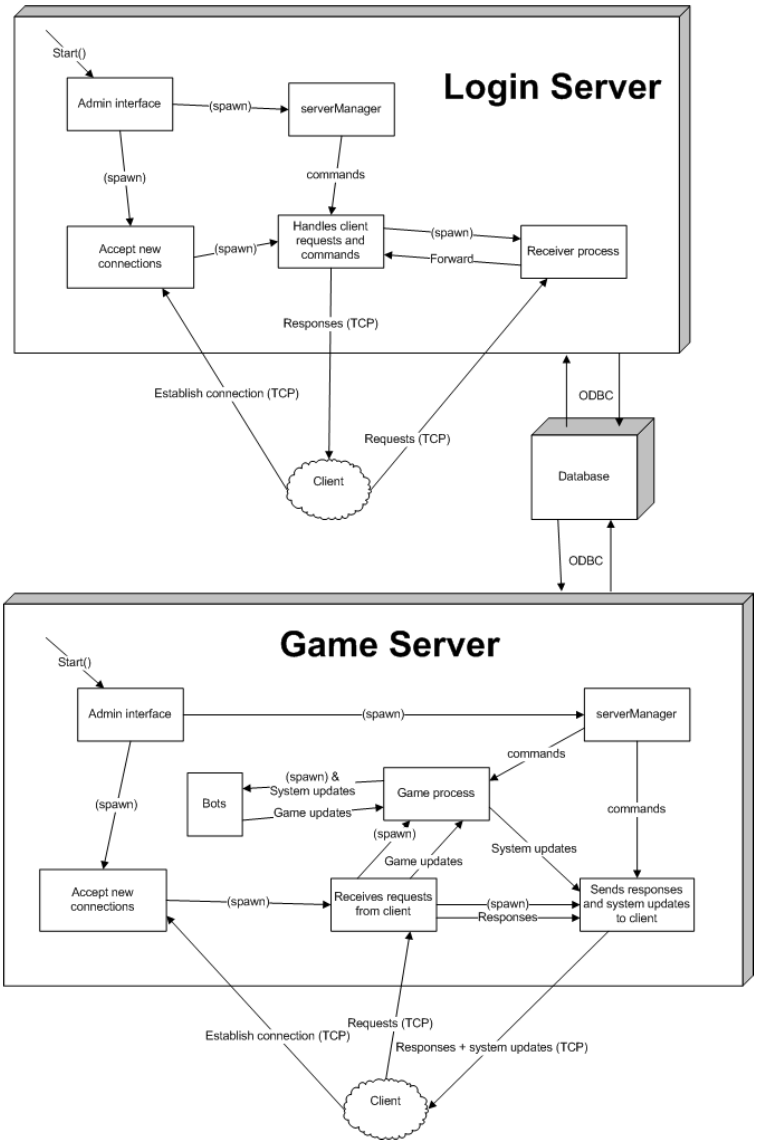

Figure 2: Server design

The main functionality of the login server is to handle the initial login request of the client and to register a session for each valid client in the database. It handles simple requests from the client such as: getting a game list, joining of a specific game and finding a free game server for new game creation. Upon start of the login server it will read from a file to configure the server. Configurations include database address, login server address and maximum client connections. The server automatically registers itself in the database. Once the login server is started, a process will accept incoming connections. For each incomming connection, a process will be created to handle further requests.

When a client chooses to join or create a game, the login server redirects the client to the appropriate game server. The main functionality of the game server is to host games for the clients. For each game on the server, a specific game process is created to handle the game logics. For each player in the game, a request handling process will accept requests from its client application and notify the game process with each update. One sender process per client accepts updates from the game process and sends them to the client.

The game server registers in the database in the same way as the login server.

4.1.3 Server Implementation

The server software is implemented in Erlang. Erlang was chosen for its ability to handle a vast amount of concurrent processes, and for its simple inter-process message sending. Erlang is also platform independent, hence it can run on all operating systems where the Erlang VM is supported. A module was created for each major part of the server.

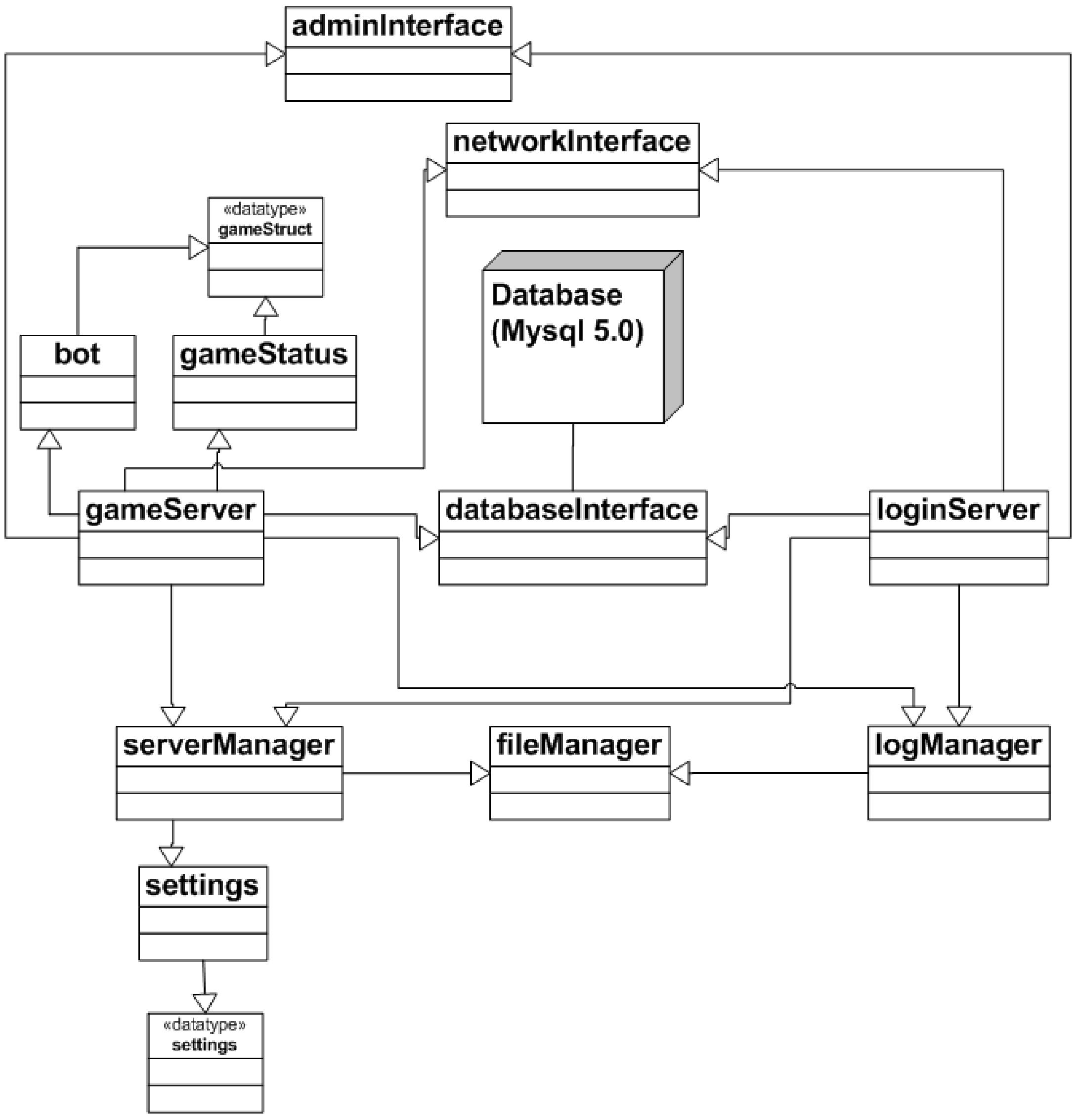

Figure 3: Server modules

There are two main modules: the loginServer module and the gameServer module, which represent the login server and the game server, respectively. Both servers can access the database through the databaseInterface module, which handles connections to the MySQL database using Erlang ODBC. To access the configuration file, a fileManager module was created. Both server types can access it through the serverManager module, which handles a process (a Server Manager) that stores configurations, and the logManager module, that logs messages into a log file. The networkInterface is a utility module which handles network connections. The adminInterface module is a simple interaction module based on the Erlang VM shell, and enables an admin to administrate a server.

Additionally, there are a number of sub modules that are used to access data structures (records), such as the gameStatus and settings modules. The bot module handles NPC:s, (Non-Player Characters), so when needed, it is used for generating NPC processes that play against players or other NPC:s.

4.1.4 Game logic

The game logic is implemented in the game server module (and a number of sub modules). It describes the game rules, winning conditions, AI behaviour and score calculation.

The winning condition is straightforward, making a big part of the game logic quite simple. The more difficult parts of the game logic are score calculation and the bot AI.

A player's final score is based on two values: Firstly, the base score that is calculated from the board status and, secondly, a modifier that is calculated from the player's rank and the average rank of all players in the current game.

The AI behaviour is more complex. Firstly, the AI needs to choose a tile to play. This is done by calculating a value for each tile, based on the board status, and choosing the tile with the highest value. Secondly, the AI ”plays” the tilegame. This is done by choosing an existing score and time duration for the specific tilegame from the database. If such a score does not exist, a random value based on the current tile score and duration is generated. After the time duration, the AI will send the score to the game process.

4.1.5 Problems encountered

During the server development, more than a few problems were encountered. These can be categorized into three categories:

- The most common problems are bugs caused by either bad design or bad coding. As many as possible of these solved through debugging.
- There have been problems caused by misunderstanding of the Requirement Specification and bad communication between members of the project team. These problems are generally fixed by spending more time (planning, designing etc).
- The most difficult problems are the technical problems. An example is the Erlang ODBC module which does not work properly under MS Windows. Problems with MySQL cause a server bottleneck. Therefore, the servers can only handle as many clients as the database server can handle connections. The ODBC problem is very annoying, but it does not seem to appear when running the server software on a Linux machine. The MySQL bottleneck problem, however, is more difficult. The load on the database server can be decreased by using as few and as short database connections as possible. There is built-in support in MySQL for setting up a more distributed database solution, which is a future possibility for TGM. Moving the runtime data to an Erlang Mnesia database is a possibility to increase the database performance.

4.2 Game client

The client design is based on the series of articles written by Richard ”superpig” Fine, a series called ”Enguinity” [5]. ”Superpig”’s idea when writing ”Enguinity” was to design a game engine that was supposed to be extendable and designed in the same way as modern game engines by professional game developers. The idea from the TGM client group was then to extend ”Enguinity” to be as portable as possible due to it being implemented in mobile phones.

To accomplish the idea set out by the TGM client group, a tool called J2ME Polish was used to provide all hardware specific constants - such as screen size, values for keys, etc, into the code by the tool’s precompiler. J2ME Polish [2] is a well-known and well-used tool within Java-development for mobile phones and as such the choice was easy.

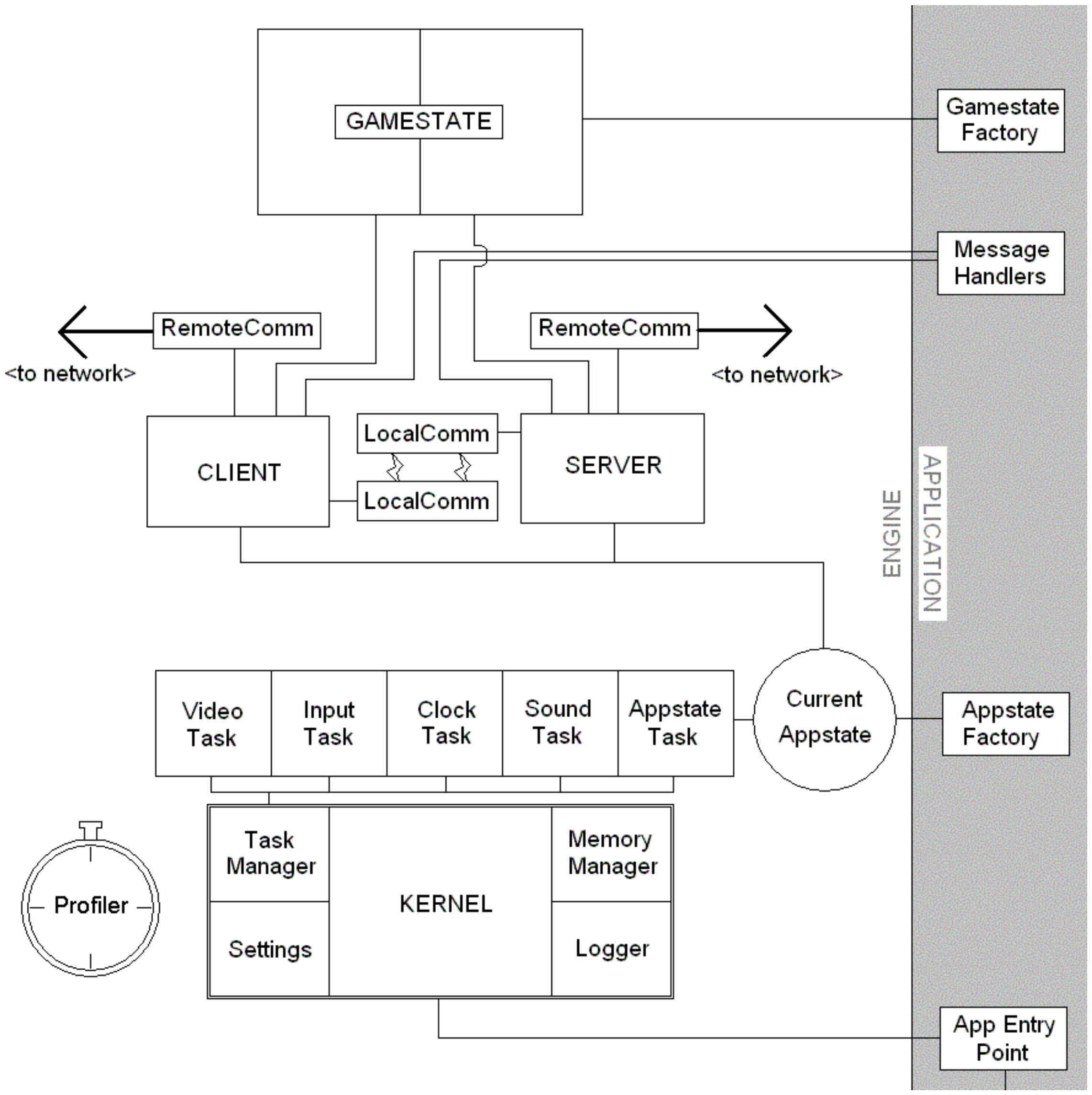

Figure 4: ”Superpig”’s engine design

4.2.1 Client Design

The client is designed in such a way that it is two-layered. One layer for the engine and one for the so called gamestates. There are different parts of the engine that are separately threaded, such as the output handlers. The part that renders the graphics, which is a so called GameCanvas, and the network handler is both on separate threads.

The engine layer keeps track of internal tasks and also retrieves information from the currently active gamestate that it uses to render the graphics. All state-information is also bound outside the engine itself which is what makes the engine easily extensible. Since the engine is finished, all that is needed to add extra contents to the game is done by adding additional gamestates.

4.2.2 Final Client

By the end of the project, the client has reached a substantial amount of size which is represented by the time, work and effort put into it. The code has reached a size of 450kB, which can be represented as a Total Lines of Code (TLOC standards) count on roughly 7500. The code itself is documented and a lot of effort has been put down into writing code that is easy to maintain and for others to work on. It is common that Java Mobile applications are written into very few, long files containing code that is hard to follow and difficult to maintain. The reason for developing such code is to save space since mobile phones often have very limited size for Java applications. However, the developers of the client has taken a step away from this convention and believes that the differance is not worth it. The more modern phones have less issues with larger sized Java applications and as such, it is important to try to push future developments into the right direction.

Because of the goal of having code that is easy to maintain and extend, a code specification was written during the early stages of the project. The document contained standards for using eDoc and javaDoc, amongst other things. This specification turned out to be quite large (40+ pages) which made it a little bit too heavy for everybody to read through. However, the standards were followed due to several reminders from the other project members. The result - the client became a fully object oriented design that produces an external documentation for programmers that want to continue working on the client.

J2ME Polish's code obfuscation is used in order to achieve protection against hacking in the client. This is needed due to java code otherwise being interpretated on the run, and as such .jar files contain the code straight down. Code obfuscation also reduces the size of the code, in bytes, which is helpful in mobile phone applications due to mobile phones having very limited memory and limited sizes on applications.

4.3 Web portal - teazle.se

Thw website is an important part of the Teazle goes Mobile project. Its purpose is to provide a meeting place for TGM users. It's also here the user must register if online gaming is desired. The game is available and free to download on the web portal. There are two ways of downloading TGM, either through a WAP page on the portal or through a normal download link, but then the user must transfer the game to the phone manually.

The main links, which can be found in the navigation bar, are 'Home', 'About', 'Games', 'Highscores', 'Shop', 'Downloads', 'Help' and 'Forum'.

5 Testing

5.1 General testing

During the TGM project a slightly iterative version of the waterfall model was used. Both the client and server were implemented top-down which made it easy to go back and change the requirements since the most important features were implemented first. However, a lot of effort was put into the requirement and design phase and after the requirement specification and design were completed a test specification was created, which made a few unreasonable requirements obvious before implementation. The tests were not used until late in the project to check the quality of the solution when both the client and the server were completed. However, a lot of testing was done during implementation but wasn't documented because of a tight schedule.

5.2 Beta test

Besides normal testing a live beta test was also done. At the later part of the implementation phase 12 people (ages 20-30) were invited to take part in testing by playing the game. Due to delays online gaming could unfortunately not be tested at that time. Instead, the players tried out Hot-Seat and single player. After playing the game they were asked to fill out a survey. The results were good. The testers liked the game, its design and also the menu layout. Clock, however, was not appreciated as a single player game. Breakout, on the other hand, received a high score in the survey. All the testers agreed that there should be more tilegames, that two was far from enough.

Sony Ericsson K310i, K500i, K700i, W810i and K800i were the phones used during the beta test.

6 Conclusions

6.1 The project

At the start it was pretty clear what needed to be done in this project, a remake of the original Teazle for mobile phones. After doing some research a good requirement specification was created without too much of a hassle. At this time a project plan and a rough time plan was also made. Not until after the design phase, when the implementation started, a more detailed time plan was created, a Gantt chart with specific tasks. The project has suffered a bit because of this. The rough time plan was a bit too optimistic. It was decided not to start the implementation phase until about the first of November. For the client side this was not really a problem but the server side implementation should have been started earlier. One team member left early in the project which made the situation tougher for the server side since they lost a member. The situation was believed to be solved by putting one person from the client group to work with both client and server. This solution seemed fine until the server side started to fall behind schedule. The server side wasn't entierly ready for implementation when it was scheduled to start because of design issues, and therefore couldn't follow the time plan. Because of this, online gaming couldn't be displayed at Review 2 or tested at the beta test as planed. The server side did, however, catch up later on and was able to demonstrate a well working online game at the final presentation.

Since the project was divided into three subprojects, server, client and web portal, communication between them was very important. This was known from the start and worked well in the beginning through the requirement specification and the design phase but decreased somewhat during implementation which was not good. Some misunderstandings were made by all sides but were solved reasonably fast. The lack of communication during the implementation phase didn't cause any major problems but probably caused some of the delays that could have been avoided.

6.2 The game

Creating a distributed mobile phone game involves creating both a server and a client. The client was made in J2ME since modern mobile phones only accept applications of such kind. However, since the server was set out to be able to handle a large amount of connections, the choice was made that the server was to be written in Erlang.

One of the important decisions in designing the game was that the code has to be easy to maintain and expand. This was accomplished partly through following eDoc's and javaDoc's specifications for writing comments, partly through following a coding standard written initially in the project. These tools extracted the wanted documentation for the code.

6.3 Final thoughts

The project as a whole has been a resourceful gain for all members of the development team. Experiences regarding team work, project work, planning, coding, designing, writing documents, and numerous more subjects has taught the members of the group a variety of things. Even though there has been some bumps on the road, and not everything worked out as planned, the TGM group has managed to develop a package of software that is something that just might be seen on the market in the future. If not so, it sure is good enough for it.

References

[1] J2ME, *Java 2 Platform, Mirco Edition (J2ME) Overview*
http://java.sun.com/j2me/overview.html

[2] Polish, *J2ME Polish*
http://www.j2mepolish.org/

[3] Erlang, *Erlang Programming Language*
http://www.erlang.org/

[4] GPRS, *General Packet Radio Service (GPRS)*
http://en.wikipedia.org/wiki/Gprs

[5] Enginuity, *The Enginuity Design by Richard Fine*
http://www.gamedev.net/reference/programming/features/enginuity1/