A Mobile Multiplayer RPG: Methods and Limitations

Dan Kindeborg, David Linder

August 10, 2006
Master's Thesis in Computing Science, 2*20 credits
Supervisor at CS-UmU: Anders Broberg
Supervisor at Resolution Interactive: Matti Larsson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

With the performance of mobile phones rapidly increasing, more advanced games can be developed for these devices. However, there are still many technical restrictions that should be considered when developing a mobile game. This report presents methods and approaches for mobile multiplayer role-playing game development, considering the possibilities and restrictions mobile gaming implies. The focus lies on game scripting, map systems, networking and interaction between players and discussions are grounded in in-depth investigations. A game prototype is implemented using the presented solutions and from this conclusions are drawn regarding both the solutions presented and mobile game development in general.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last few years, the capabilities of mobile phones have increased rapidly. This has made them interesting for the entertainment industry and one of the growing areas is mobile gaming. Releasing sophisticated mobile versions of PC and console games is becoming more and more common, which games like Final Fantasy - Before Crisis [1], Quake Mobile[2] and DOOM RPG [3] are good examples of. The ability to launch Java applications and connect to the Internet provides new possibilities for mobile games, but there are still many limitations for developers to consider due to factors like limited computing performance and network bandwidth.

1.1 Background

Resolution Interactive is a Swedish game developer company which was founded in 2003 by former employees at Daydream Software AB. Resolution owns the titles Clusterball® and Ski-Doo X-Team Racing™ and develops games for various platforms [4].

Furiae is a fantasy world created by digital artist Linda Bergkvist [5]. It was developed as a traditional role-playing game (RPG) during 10 years and Linda have created more than 60 illustrations inspired by the world, see figure 1.1.

The Furiae mobile game concept has been developed during the last few years and takes off in Lindas world with its myths, legends, religions and characters. In the game, all living creatures radiate an aura of spiritual energy. Human characters can weave invisible connection threads to creatures and other humans in the world. These connections let characters make use of the energies of the connected creatures which enables them to wield magic. The medium for playing the game and establishing connections to friends in the world is a mobile phone and the goal of the game is to achieve a higher level/social position by defeating enemies and completing adventures.

[1] http://www.sqexm.com
[2] http://www.pulsemobilegames.com/QuakeMobile.html
[3] http://www.doomrpg.com
[4] http://www.resolutioninteractive.com
[5] http://www.furiae.com

Figure 1.1: Illustration from the Furiae world by Linda Bergkvist

1.2 Purposes and goals

The two goals of the work described in this thesis:

- To create a prototype for the Furiae mobile game which can be presented at the worlds largest game expo, E3 [6], May 10th 2006. The purpose of this is to attract publicists and other potential stake holders to invest in the development of the game prototype into a commercial product.

- To develop and present solutions required to cope with the limitations developing a game like Furiae for mobile phones might imply. Thus, a subgoal is to investigate the characteristics of mobile gaming in general and the technical limitations of mobile game development is particular. The purpose is to, with these solutions, provide a base for further development of the game.

One approach to achieve these goals could be to develop a fairly simple prototype which could present the game concept combined with a report explaining the limitations identified and the solutions required. However, it is determined that combining the goals and creating a working prototype with the solutions implemented would serve the purposes of both goals better, by both increasing the attractiveness of the prototype and providing a better base for further development.

To include the solutions required by the Furiae mobile game, the prototype should resemble the intended commercial product and was thereby specified to satisfy a number of requirements.

[6] http://www.e3expo.com

1.2.1 Functional Requirements

The prototype should include the following game elements and features:

Walk

A player should be able to walk around in a world made up of several maps, inhabited by NPCs (Non-player characters) and creatures, see figure 1.2. Characters in the world should be able to move behind objects on the maps, creating an illusion of three dimensions. Maps and objects are beautiful, painted images created by graphics artists at Resolution Interactive.

Figure 1.2: Walk mode from the prototype

Figure 1.3: Talk mode from the prototype

Talk

A player should be able to talk with NPCs and get to select what to say from a list of choices, see figure 1.3. NPCs should also be able to do things such as give the player items and trigger fights.

Fight

A player should be able to fight enemies in the world in a turn-based fashion, see figure 1.4. The player fights by selecting a spell to cast upon the creature, whereupon the enemy responds by casting a spell of its own. Creatures in the Furiae world can be defeated in two ways. Either a player can kill a creature using physical spells or the player can use special mental spells to break down the creatures mental resistance and, eventually, bind the creature. Killing a creature makes the player gain experience points and binding lets the player use the bound creatures energies, possibly allowing the player to cast new spells. When fighting a creature, the player can also choose to invite other players to participate in the battle. Thereby, several players can together cooperate to achieve a common goal.

Trade

Players should be able to create connections to other players in the Furiae world, see figure 1.5. When a connection is established, connected players can trade their connections to creatures with each other and thus enable new spells. However, the game should not support the writing and sending of text messages during trades. The purpose of this

Figure 1.4: Fight mode from the prototype

Figure 1.5: Trade mode from the prototype

game element is to further stimulate the players to collect creatures and create a feeling of collectiveness even though the game is mainly played in single-player mode.

Scripts

Adventures, maps, NPCs and creatures should be scriptable using some scripting language. If this requirement is fulfilled, developers can add new characters, maps and adventures to the game without interfering with the source code. A scripting system also makes it possible to, at a later stage, enable players to download new adventures from their mobile phones and, eventually, script their own adventures. The scripting system must be powerful enough to accommodate a prototype adventure, specified by Resolution Interactive.

Save

The player should be able to save the progress of his or her Furiae character.

Not always-online

The player should not have to be continuously connected to the Internet during game play. This is important since mobile data services can be expensive and users have little control over the amount of data transferred by an application. Instead, the application has to work like a e-mail client, fetching new messages from a server when necessary and there should be a possibility to play the game strictly single-player and thus never use mobile data services if the user chooses to.

1.2.2 Non-functional Requirements

The prototype should satisfy the following requirements:

Phone support

The game should be runnable on as many phones as possible and must at least support the Nokia 6600, Sony Ericsson W550i and Sony Ericsson W810i. It should be programmed as model-independently as possible to make testing on additional phones easy.

Platform

The game should be developed for the Java 2 Platform, Micro Edition (J2ME) platform.

1.3 Methods and tools

A number of areas have to be studied before any design and implementation of the system can begin. In chapter 2 we investigate the special needs of mobile games. Technical aspects and limitations are presented in chapter 3. Game scripting theory and game scripting systems are studied in chapter 4 as an aid in providing the prototype with a good scripting system.

Grounded in the knowledge gained from the pre-studies, chapter 5 presents design decisions made for the scripting system, the interactive modes of the game, the map system and other areas. Based on this design, the prototype is implemented and continuously tested on mobile phone emulators. The prototype is also tested on real phones as they are available and adapted to run as well as possible on them. Details on the implementation and experiences gained from implementing and testing is presented in chapter 6.

The prototype is implemented using the Eclipse environment with the EclipseME plug-in for J2ME development. Wireless Toolkits from Sun, Sony Ericsson and Nokia are used, which enable testing on emulated mobile phones. A MySQL database stores player data at a game server implemented in Java. A number of mobile phones, mainly Nokia 6600, Sony Ericsson W550i and Sony Ericsson W810i are used for continuous testing on real devices. Photoshop and ImageMagick are used to manage game graphics.

Chapter 2

Mobile Gaming

In 1997, Nokia started including the game Snake with their mobile phones and it became very popular [21]. Since then, the game capabilities of mobile phones have increased rapidly, resulting in more and more advanced games. Still, the opportunities and restrictions of mobile games differ from PC and console games in many ways. This section will introduce mobile gaming and explain in which ways such games can differ from regular computer games. A great deal of technical details are left out and instead presented in section 3.

2.1 Communication and mobility

Since cell phones are originally intended for communicating, mobile games have lots of possibilities when it comes to networking. It is common to include the capability of playing a game with other nearby players using the wireless Bluetooth technology included in many modern phones. Bluetooth enables the application in one device to form a spontaneous, ad-hoc, network with other nearby devices running the same application.

Mobile data services enables mobile phones to connect to the Internet, allowing a player to play games with other players located anywhere in the world.

An interesting fact is that people usually carry their mobile phones wherever they go. This opens up for new possibilities. In 1999, Nokia launched Nokia Game[1], an adventure where the game could be triggered at any time. The player could receive information through any medium and mobile phone technologies like SMS and voice calls played an important role. SMS notifications allow players to be able to react almost instantly to game events and they can for example be used to invite friends to join a multi-player game.

2.2 Positioning

Positioning systems are available as an accessory for some modern cell phones and hand held computers or Personal Digital Assistants (PDA). Positioning opens up the whole field of location-aware pervasive gaming, a field where the game is integrated into the actual physical environment. Magerkurth et. al. describes the game *Treasure* where the

[1] http://www.nokiagame.com

task of the game is to collect virtual coins hidden in a large outdoor area. Players are equipped with PDAs with Global Positioning System (GPS) receivers and learn about the positions of coins on the display of their hand held computers. Thus, a part of the physical world is seen as a game board [11].

2.3 Display and input

Mobile devices are small compared to stationary computers and naturally, so are the displays. Most modern mobile phones have a screen resolution equal to or less than 176x220 pixels as seen in appendix C. This fact makes it impossible to represent as much text and other information as on the display of a modern stationary computer having a resolution of at least 1024x768 pixels.

Figure 2.1: The Sony Ericsson W800i

Figure 2.2: The Nokia 6600

The input possibilities of mobile phones often differ from model to model. Figure 2.1 and 2.2 shows two phones with different input capabilities. All modern phones include the number pad and some kind of navigation tool. The navigation tool can be used to navigate right, left, up and down and can be pressed, or fired, to select options. On both models it is located directly above the key labelled 2. Most models also include left and right *soft buttons*. These are used to select one of two options appearing on the bottom of the screen. On both models they are located beneath the screen, to the left and to the right. Apart from number pad, navigation tool and soft buttons vendors can include a uniform amount of special buttons. For example, on the W800i which is intended to be used as a music player, there is a special music button located between the two soft buttons.

Games often utilize the navigation tool to move a game character and fire to perform some kind of action. It is also in many cases possible to use the number pad and button 2, 4, 6 and 8 to move and 5 to fire. If more than one action can be performed, there is the choice of using either the soft buttons or any key on the number pad.

2.4 Limited memory

The storage memory of mobile phones is growing but the size of mobile games is still a problem. If a phone contain 32 MB of memory, a user may be unwilling to install a 3 MB game, sacrificing almost 10% of phone memory. Most mobile games are even smaller than 3 MB which calls for sparse use of pictures, music and other memory consuming game elements. Therefore, graphical elements are often reused. Game maps are normally made up of *tiles*, small images put together very much like mosaic to make up the world.

2.5 Scripting

Due to the limited capabilities of mobile phones, it is important to optimize code for best performance. For games that use additional scripting systems, this is especially important. Careful consideration is therefore required when selecting such a system to be used in a game. This topic is examined in detail in chapter 4.

Chapter 3

Technical Aspects of Mobile Gaming

To identify what problems need to be solved to fulfill the requirements of the Furiae prototype stated in section 1.2 and provide a good base for further development, it is important to investigate the technical aspects of mobile gaming. In chapter 2, some of the opportunities and restrictions connected to mobile gaming in general were introduced. In this section, we will pursue investigating the limitations of mobile gaming at a more technical level to understand the technical issues developing a game like Furiae might imply.

3.1 Networking

Many mobile games, including Furiae, need to communicate with other devices. As mentioned in section 2.1, there are several ways of communicating. Here, we investigate the technical restrictions of different mediums and evaluate their appropriateness for use in mobile games.

3.1.1 Mobile network data services

All modern mobile phones feature the ability to connect to the Internet but there is a wide variety in the types of services for mobile Internet communication provided in the world today. Mobile phones differ in network support and network applications for mobile phones must consider the limitations in the network mainly in terms of bandwidth, availability, round-trip time (RTT), latency and jitter. In a mobile network, the RTT often defined as the time for a message to be transferred to a remote server connected to the Internet and back again. Latency is the delay between the moment something is initiated and the moment the effects begins and jitter is the unwanted variation of bandwidth. These parameters all affect the transferable amount of data and the response time, thus affecting the possible ways of interaction in a mobile game[1]. This section will give an overview of the most common mobile network data services provided and their limitations in terms of bandwidth and RTT.

GSM CSD

Circuit Switched Data (CSD) is the original standard for data communication, operating on the Global System of Mobile Communications (GSM) network which is deployed in more than 210 countries and territories in the world. Similar to a normal voice call, the mobile phone initiates a data call (setting up a circuit) and using modems, data is transferred at a rate of 9.6 kbit/s or 14.4 kbit/s depending on the service. RTT is typically 1 s for a GSM network operating in the 900 MHz frequency band and 0.5 s for a GSM network operating in the 1900 MHz band [4].

GPRS

A problem with GSM CSD is that it is a circuit-switching technology. A data call is made and the circuit must be maintained during data transfer. This makes poor utilization of the resources of the network since resources will be tied up during the whole duration of the call. General Packet Radio Service (GPRS) avoids this, still using the GSM infrastructure, and is available with almost every GSM network. GPRS is a packet-switched technology which implies that resources are used only actual data transfer only. There are four different Coding Schemes (CS) with different capacities, ranging from 9.05 kbit/s (CS 1) to 21.4 kbit/s (CS 4) [4]. The RTT for a GPRS connection varies depending on load and on a non-loaded network a typical RTT will be about 700 ms [20].

HSCSD

High Speed Circuit Switched Data (HSCSD) is a further development of CSD, enabling circuit switched data communication at a down-link data rate of up to 57.6 kbit/s and an uplink data rate of up to 14.4 kbit/s. However, to achieve maximum down-link data rate the call has to occupy four time-slots which is also four times as expensive. HSCSD is very expensive in many GSM networks which has lead to GPRS being more common. The RTT is, like CSD, 1 s in a 900 MHz band GSM network and 0.5 s in a 1900 MHz band GSM network [4].

EDGE

Enhanced Data rates for GSM Evolution (EDGE), also called Enhanced GPRS (EGPRS), is an enhancement to GPRS networks which allows for speeds of up to 236.8 kbit/s but the same RTT, 700ms with no load, as standard GPRS. Its attractiveness lies in the easy migration of GSM networks but EDGE require higher radio signal quality than found in the average GSM network. EDGE networks have been introduced many countries including USA, Finland, Malaysia, Canada, South Africa and France [23].

WCDMA

Wideband Code Division Multiple Access (WCDMA) is a technology for the third-generation (3G) mobile phone networks which the Universal Mobile Telecommunications System (UMTS) standard is based on. WCDMA provides both packet- and circuit-switched data and offers a data rate of up to 2 Mbit/s although most WCDMA-based networks today offer a data rate of up to 384 kbit/s. According to Nokia, a 384 kbit/s WCDMA network has an RTT of about 200ms [20]. WCDMA-based 3G networks have already been deployed in Japan and in some European countries like Finland, Norway,

Sweden and major cities in USA. Most western European GSM providers plan to offer UMTS services in the future [23].

CDMA2000 1xEV-DO

Code Division Multiple Access 2000 1 x Evolution-Data Optimized (CDMA2000 1xEV-DO) or just EV-DO is a rival 3G-technology, not compatible with WCDMA. Revision 0, currently deployed in North America, has a down-link data rate of up to 2.5 Mbit/s, an uplink data rate of up to 154 kbit/s and a specified RTT of 150ms. EV-DO is, unlike WCDMA, backwards compatible with its predecessors; older CDMA-based standards such as cdmaOne which is used in USA, South Korea, Canada, Mexico, Israel, Australia, Venezuela and China. EV-DO is already deployed in Japan and will be deployed in northern America 2006 [23].

3.1.2 Internet data communication protocols

Mobile phones, evolving from voice communication devices to entertainment devices, are now capable of Internet communication using the mentioned data communication services. However, mobile networks are quite different from Local Area Networks (LAN). The communication protocols successfully developed for the Internet may not be as effective and also, the range of protocols offered in a mobile phone may be limited. This will, combined with the data communication service and the host processing power, set the physical restrictions of interaction in a mobile multi-player game[26].

It is important to note that most GPRS users will be behind Network Address Translation (NAT) gateways. Thus, there will be no externally visible IP addresses. To transfer data between two mobile phones using GPRS there is no way of determining the IP address of a remote phone and thus, a proxy server has to be involved to set up a connection [22].

TCP

The Transmission Control Protocol is connection-oriented protocol which means that a bi-directional data stream is maintained between the communicating parts for the duration of communication. When the data stream is set up, either part may transfer messages to the other. TCP makes no assumptions about the reliability of underlying protocols and data is always received once, in correct order and error free. Packets include a certain amount of TCP overhead which makes it beneficial to instead of transferring many small packets, each with their own overhead, merge them into a single larger packet [19]. TCP was designed for reliable links and assumes that all packet losses are caused by network congestion. This is not the case for mobile networks where most of the packet loss is caused by transmission errors. Therefore, TCP will perform poorly in mobile environments where disconnections is known to occur frequently and no solution have had enough impact to become a new standard [34].

Taferner and Bonek (2002) measured the throughput and RTT of a typical FTP file download in GSM CSD. In a network with the theoretical throughput of 9.6 kbit/s the TCP throughput reached 8 kbit/s maximum with a RTT of about 900 ms and in a network with the theoretical throughput of 14.4 kbit/s the throughput reached 13 kbit/s. The optimum packet length was very much depending on the quality of the data channel, with an optimum packet length of about 500 bytes for low-quality channels and optimum packet length of about 900 bytes for high-quality channels. In a similar setup

using UMTS networks with different theoretical throughputs, TCP throughput reached a maximum of about 95% of the theoretical throughput [34]. Vacirca et. al. observed a TCP RTT of about 500 ms in a UMTS network and 1-2 s using GPRS very much depending on network load and amount of data transferred [38].

UDP

User Datagram Protocol (UDP) is an unreliable protocol which means that it does not guarantee that sent messages are received in the order they were sent or that messages will only be delivered once. Being transaction oriented, UDP does not set up a data stream between the communicating parts [19]. This can be problematic in the case of data transfer using GPRS because most users will be behind Network Address Translation (NAT) gateways. NATs typically prevent inbound connections and when a mobile client is sending a UDP packet to a remote Internet host the NAT may or may not allow the host to send back data. Thus, depending on the policy on the NAT gateway, the host cannot be sure that it can send data back to the client or for how long. This can be avoided by instead using TCP or TCP-based protocols like HTTP. [22].

HTTP

Hyper Text Transfer Protocol is a protocol on a level above TCP and UDP. A HTTP client makes use of TCP to send requests to a host which replies and HTTP defines the syntax of these requests and replies. Because many applications need a request-reply protocol HTTP can be convenient but it is also more restricting than developing an application-specific protocol since HTTP communication must be based on requests and replies. Also, HTTP typically add large headers where some information fields are transferred repeatedly due to its stateless nature [25].

TLS/SSL

Transport Layer Security is a protocol which provides authentication, privacy and data integrity between two communicating applications. It is based on and closely related to Secure Sockets Layer (SSL) and both protocols are layered on top of a reliable transport protocols like TCP.

3.1.3 Internet network application architecture

The application architecture specifies the components of a distributed system and how they will communicate. Designing an application communicating using mobile network data services, the application architecture has to be designed considering the limitations of mobile network communication.

Peer-to-peer

In a peer-to-peer, or highly decentralized, architecture, all communicating parts, peers, of the network have an equal role. Peers can request information from other peers and will also have to be able to respond to requests [9]. An important point is that all clients can provide computing power and storage space, thus increasing the computing power of the system as a whole. There is also no single point of failure in the system, if a peer fails the system can keep working as other peers takes over its role.

The peer-to-peer approach may seem attractive in its simpleness but it has two serious issues in mobile networks. Firstly, latency is very long compared to a regular Internet connection and when communicating mobile-to-mobile the time for transfer and reply is even longer, typically about two times the mobile-to-Internet host RTT [20]. Secondly, the processing power, memory and battery of mobile phones is limited. If data could be processed elsewhere, expensive resources could be saved.

Client-server

In a client-server, or highly centralized, architecture, there is a host, the server, which is always available from other hosts called clients. Clients never communicate directly with each other, instead all communication takes place through the server [9]. The main drawback of this approach is scalability, the server would need to have high network capacity and be available even when the number of clients increase [1]. Also, it is less fault tolerant than a peer-to-peer system since the whole system depends on a single server. In mobile networks, client-server architecture utilizes network resources well and will not expose the interaction devices to as large RTTs as peer-to-peer. If the server is not a mobile phone but a stationary computer connected to the Internet, the client-server approach will also solve the problem with resolving the IP address of another mobile phone which should be contacted.

In a networking game using the client-server application architecture, the server must be able to accommodate multiple clients concurrently. Typically, each server connection will run in a separate thread on the server, thus consuming memory and CPU resources. The amount of concurrent server connections can also be limited by the architecture of the operating system and, in the case of a Java server application, the Java Virtual Machine (JVM) used. VolcanoMark is a Java server benchmarking program which predicts the performance and connection limitations for different JVMs and operating systems. In the latest Volano Report from 2003 it is concluded that the best connection capacity is achieved with the BEA JRockit 3.1 JVM on Windows. Blackdown 1.3.1 Java platform on Linux is second best, scaling to 10000 connections easily. Both version 1.3.1 and 1.4.1 of Suns Hotspot JVM, which is the standard and most widely used JVM, performs poorly on Linux but supports a maximum of 3000 client network connections on Windows and 4000 on Solaris. However, after this test was made many new JVMs have reached the market. Also, the hardware used, a Dell OptiPlex with a Pentium III 500 MHz CPU, can be considered quite immodern [17].

3.1.4 Bluetooth

Bluetooth is another wireless communication technology included in most modern mobile phones. A main difference between mobile data services and Bluetooth is that Bluetooth is a short-range medium. Mobile devices with Bluetooth modules can communicate via Bluetooth if they are within about 10 meters from each other in contrast to mobile data services which should be available virtually anywhere on the planet[12].

Bluetooth devices form ad-hoc networks dynamically. Firstly, one device initiates a search for nearby devices. Secondly, each device is searched to determine which services are available. In most cases, devices are searched for one specific service provided by the communicating application. Each service is assigned a Universally Unique Identifier (UUID), a unique 128-bit value identifying it. The device performing the inquiry will take on the master role in the formed Bluetooth network, the *piconet*. Other devices will be slaves and all communication will take place between the master and the slaves

as depicted in figure 3.1. Thus, Bluetooth network application architecture is highly centralized even though it may be possible to switch master-slave roles[18].

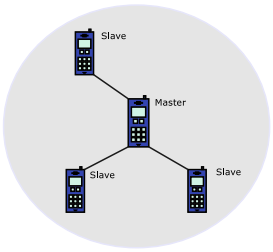

Figure 3.1: A Bluetooth piconet

The attractiveness of Bluetooth communication lies in the fact that Bluetooth communication is free. In contrast, the GPRS mobile data service offered by the largest Swedish subscription service supplier, Telia, for the subscription Telia Refill cost 0-9 SKR per day depending on the amount of data transferred[35]. Also, Bluetooth communication has superior quality compared to GPRS with a RTT of about 50ms and a theoretical bandwidth of up to 1 Mbit/s[12].

3.1.5 SMS

Short Message Service (SMS) is a functionality included in all modern phones allowing the user to send short messages. In some cases, applications can have the ability to receive and send SMS and thus it is one additional medium through which a mobile game can communicate. SMS is a slow service with high costs (0.75-1.50 SKR/message using the Telia Refill subscription[35]) but there is one advantage. In both mobile data service and Bluetooth communication, a connection needs to be established between two communicating devices. Thus, both devices must be running an application at the time of connection establishment. A SMS message can be sent from an application to a phone and will always be received. In this way, SMS messages can be used to initiate an Internet or Bluetooth communication session between two applications.

3.1.6 Handling latency

In real-time action games like Quake, latency is a critical factor. Anttila and Lakkakorpi performed a study where test persons graded the quality of experience in a real-time action game and determined that the maximal latency where the gaming experience is still considered satisfactory is about 320ms. Latency can be a result from other factors than network RTT but if we assume that the time it takes for a game to process the users actions and process the result gained from a game server is very small, 3G mobile data services like WCDMA and EV-DO and Bluetooth are the only mediums capable of delivering a satisfactory gaming experience in a real-time action game. Another game category is turn-based games, where players take turns acting similar to a traditional board game. In these kinds of games, players tolerate much higher latencies, up to 40 seconds[2].

Several methods for handling latency in action games have been proposed. Here, some methods for use in mobile networks presented by Upton are considered[37].

Ghost Players

This approach involves recording players actions, for example in a run on a race track, and uploading them to a server. Other players can download the recorded run and compete against the uploaded player which will be displayed as a semi-transparent player, a ghost, in the game. This approach works only if there is little interaction between players.

Deterministic Combat

In games where interaction takes place only if a conflict occurs, the conflict can be resolved using only game logic. Thus, there will be no real interaction and the player will have no control of the battle. There is only some brief connectivity to agree the result. This approach works only in very special cases, when combat can be made deterministic.

AI Proxies

In real time strategy games, the behavior of units is controlled only partially by the user. Units move autonomously, controlled by artificial intelligence routines and guided by user input. Thus, latency is not a big problem since it manifests itself in starting an action only.

Simplex Resolution

Since deterministic combat offers a poor interactive experience, Upton suggests an approach where the game proceeds in rounds consisting of two phases, movement and resolution. In the movement phase, players move their units independently. Conflicts arise if both players have their units in the same area. In the resolution phase, conflicts are resolved to determine who wins each conflict. Approximately half of the conflicts are played as a single player game against the computer on the first players computer and the other half on the second players computer. This approach has high latency tolerance but provides higher interactivity than deterministic combat.

3.2 Positioning

GPS consists of 28 satellites along with a set of tracking stations providing navigation coordinates to any user possessing a GPS receiver. Wing measured the accuracy of consumer GPS receivers and observed a positional accuracy of 5-10 meters with the top performers depending on weather conditions and forest denseness[43]. However, mobile phones with an in-built GPS receiver are rare and receivers are expensive. Currently, it would probably be hard to sell a mobile game requiring a GPS receiver.

3.3 J2ME

J2ME is the Java platform for mobile devices. It can be seen as a compact version of the Java 2 Standard Edition (J2SE) platform, lacking features like reflections and access to the local file system.

A J2ME implementation consists of a configuration, which specifies the core Java software environment for a range of devices, and a set of profiles which specifies additional classes. Java-supporting phones implement the Connected Limited Device Configuration (CLDC) which is intended for cell phones or PDAs. They also implement the Mobile Information Device Profile (MIDP) which adds user interface components, local storage capabilities and networking [36].

3.3.1 CLDC

CLDC, which defines the core APIs of J2ME and the capabilities of the virtual machine, exists in two versions, 1.0 and 1.1. The two do not differ very much, the later however relaxes some of the restrictions in CLDC 1.0, in particular it enables the use of floating point types which is restricted in version 1.0. Besides the floating numbers however, the differences between the versions are minor and mostly consist of some methods or error classes that are not included in CLDC 1.0 but are implemented in CLDC 1.1.

3.3.2 MIDP

CLDC does not include any APIs for graphics and user interfaces. For this matter, profiles are added to the J2ME core and the one being implemented by all manufacturers is MIDP. As with CLDC, it exists in two versions, 1.0 and 2.0, and the biggest difference between them is the added Game API in MIDP 2.0. Where MIDP 1.0 lacks advanced graphical features, the Game API in MIDP 2.0 both increases the possibilities to show something good looking on the screen, but also makes game programming easier through implemented classes for sprites, game canvases and more. Both MIDP 2.0 and CLDC 1.1 is backwards compatible with their predecessors MIDP 1.0 and CLDC 1.0, respectively.

3.3.3 Persistent storage

A J2ME application, often referred to as a MIDlet, cannot access the regular file system in a mobile phone. Instead, it has access to its own private record store in the Record Management System (RMS). A record store is a simple database consisting of a collection of records that remain persistent after a MIDlet exits. Each record has an unique identifier and can store an array of bytes [36]. The RMS is mainly intended for storing small amounts of data like game statistics and thus, some mobile phones have restricted the amount of storeable data in the RMS by a single application. Sadly, information on RMS restrictions are seldom documented. Another implication the lack of access to the file system has is that applications cannot share data and adding new elements to an application often means that the whole application needs to be replaced. New application elements could alternatively be downloaded from a server by the application itself and stored in the RMS but since the RMS is built up like a database the process of loading application elements could get tedious.

3.3.4 Networking

The latest version of MIDP, MIDP 2.0, contain classes for connecting via the Hyper Text Transfer Protocol (HTTP) 1.1 but this is the only means of network communication a MIDP 2.0 device must implement [30]. Many MIDP 2.0 implementations, like those of Sony Ericsson [27], and Nokia [19], support additional means of network communications like TCP sockets and UDP datagrams. Any MIDP 2.0 implementation must support

HTTPS, which is secure HTTP usually supported by either SSL or TLS [8]. In many implementations, like Nokias, it is also possible to use raw TLS/SSL sockets but this is not a MIDP 2.0 requirement [19].

Bluetooth is supported through an additional API labelled JSR-82[33] and the ability to send SMS messages is supported through the API JSR-120[31]. These APIs are both optional.

3.3.5 Positioning

To enable Java applications to use a GPS receiver, there is a J2ME optional API for fetching coordinates, the location API JSR-179[32]. Today, the location API is rarely implemented.

3.3.6 Garbage collection

The virtual machine which runs a Java application on a mobile phone is called Kilobyte Java Virtual Machine (KVM) and it works very much like the high-end Java Virtual Machine (JVM) [36]. The KVM is also responsible for garbage collection. When a variable is declared a chunk of memory is allocated and when there are no more references to it the KVM deallocates the memory used. When working in environments with extremely little memory, often below 1.5 MB, garbage collection becomes extra important. Garbage collection should be performed continuously in the background as the application executes but J2ME also provide means of calling the garbage collector explicitly with the method `System.getRuntime().gc()`.

3.3.7 The Java Verified^{TM}Program

The Java Verified^{TM}Program is a testing program for J2ME-applications. An application has to satisfy a number of test criterias defined in the Unified Testing Criteria (UTC) document. UTC consists of a number of test cases, testing various aspects of the application including stability, user interface design and network connectivity [29].The test cases are applied in a testing house approved by Sun Microsystems, Inc. Once Java verified, the application can be signed using a certificate and will gain some benefits. In an unverified program, the user will have to respond to warnings whenever the application wants to utilize i.e. mobile data services, Bluetooth and SMS receiving services. Being signed, the user can choose to disable these warnings. The application also have to right to use the Java Verified Program logotype [28].

3.4 Mobile phones

When developing a game for mobile phones, it is important to make it run properly on the devices that the intended consumers are most likely to use. This requires some knowledge regarding which the most popular vendors and models are and the hardware and software specifications of these. Appendix B shows public sales information from vendors in Sweden, United States and Asia and appendix C presents technical specifications for the phones in the sales lists.

3.4.1 Data Services

Nearly all mobile phone models released the last couple of years support a platform like GPRS or EDGE to provide data transfer at a relatively high speed over the GSM network. In North America, South America and parts of Asia, other networks such as CDMA are being used besides GSM and they have other ways of transferring data. An increasing number of popular phones also support the WCDMA/UMTS technology, often referred to as 3G, but still the majority does not.

3.4.2 Java version

Almost all researched models support Java technology through J2ME except some, mainly American phones, which instead use the application development platform BREW. Roughly, since J2ME is platform-independent, applications developed for J2ME can run on all devices which support Java, independent of the operating system. This vouches for that J2ME-based games can be played on a large number of phones, but there are differences in the versions of J2ME libraries. Seemingly all popular Java phones released during 2005 implement the CLDC 1.1 framework and the MIDP 2.0 specification. Older models may only support CLDC 1.0 or MIDP 1.0. Phones also differ in whether they implement additional API:s for support areas like Bluetooth and SMS.

3.4.3 Screen resolution and color depth

The resolution of the screen comes in numerous variants. Vendors often use the same resolution for similar models but between series there are differences, both regarding the total amount of pixels and the ratio between the number of vertical and horizontal pixels. The minimum resolution of the popular phones is however 128x128 and many models have 176x220. The color depth of all popular phones with new Java technology is 16 bit or better.

3.4.4 Memory capacity

Regarding memory capacity, a heap size of 500 kB is available in almost all studied phones and many have more than 1 MB. The memory capacity for permanent storage differs a lot from 3 MB to hundreds of megabytes. Many phones also have a restriction on the size of a J2ME-application, which is stored in a JAR (Java Archive) in the phone. The maximal jar size is on some phones restricted to a few hundred kilobytes but many new phones support any jar size.

Video memory is the memory where graphics are stored before they are drawn to the screen. On mobile telephones from Sony Ericsson, Java video memory is restricted to about 800 kilobytes [27]. Since each pixel of an image is stored as a 16-bit integer, it is possible to store 400000 pixels regardless of image compression or amount of colors. Thus, it is possible to load up to 400000 pixels into video memory before drawing the graphics to the screen. When graphics are drawn, the video memory is flushed and new graphics can be loaded. The limited video memory is a problem since it restricts the size of images to be drawn to the screen.

3.5 Conclusions

Modern mobile phones with J2ME can communicate very much like a stationary computer, but the quality of communication is in many cases low. RTT is probably the most important factor that needs to be considered by networking games. Even if WCDMA technology is being employed, RTT is 20 times the RTT of a WLAN [20]. There are methods for handling latency in games. All the methods presented does however either put restrictions on game design or reduce the feeling of interactivity.

Bluetooth is the ideal communication technology when distances are small. Mobile data services is a good form of communication when distances are large but it is important to note that the IP of a phone is often very dynamic and cannot be determined easily, which raises a need for a proxy server in many mobile applications. One option is sending SMS messages to trigger communication with a remote device but SMS is not good for continuous communication.

J2ME is the most widely supported development platform for mobile phones but before implementing a game, the supported APIs and details on the J2ME implementation of target phones must be investigated. It is also important to be aware of memory restrictions since both memory for storing a MIDlet, heap size, video memory and memory for saving data in the RMS can be restricted.

Chapter 4

Game scripting

According to the requirements in section 1.2, it should be possible to script adventures for the Furiae prototype. This would enable almost unlimited extendibility of game content and if a good scripting system including all the functionalities required for the prototype adventure is employed, adapting it to facilitate the requirements of a commercial game would probably be an easy task.

This chapter examines the benefits of scripting systems and common approaches used when scripting games. The architecture of scripting systems and how they are constructed is also studied followed by a review of existing systems compatible with Java, where particularly important features for J2ME are in focus, such as the size and required libraries of the system. Finally conclusions are drawn, primarily regarding whether there are any existing systems suitable for the Furiae prototype or if it would be a better idea developing a new, custom system.

4.1 Introduction

During the development of a game it can be advantageous to separate the game engine from what is called game logic or game content. If this logic is not written separately, the whole game engine must be recompiled in order for an adjustment of some part of the logic to take effect. This situation may arise thousands of times during development and all this compiling will become tedious. It is also likely that the person who designs an adventure has not been involved in writing the game code. As a result of this, every time the designer wants to make an adjustment, a programmer must be instructed to implement it.

The advantage of a scripting system is that the game logic is written separately and changes can be made to those scripts without tampering with the game engine. As can be seen in figure 4.1, the logical content of the game is then separated from the engine in a similar way as is often done with graphics, music and sound. Ideally, this means that the designer can implement changes and test the result of them without the involvement of a programmer. With such a scripting system, a game can be distributed with a certain set of adventures and objects and all that is needed to expand the game further is to write new scripts and make them available to the gamers.

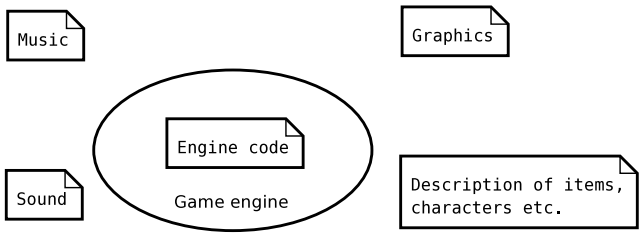

Figure 4.1: With the game content separated from the engine, it can be replaced and modified as easily as graphics and sound.

4.2 Types of scripting systems

Scripting systems may differ in complexity and structure. The three major groups are systems based on command-based languages, systems based on procedural or object-oriented languages and systems with dynamically linked modules [39].

Systems based on command-based languages provide a straight-forward way of scripting. A command-based language often contains program-specific commands which can accept a number of parameters. The commands are executed one after another from top to bottom. The following might be lines from such a language:

```
MoveCharacter 10,20
CharacterTalk "I found it!"
GetItem Diamond
```

Command-based languages may be useful for simple scripting tasks and the commands are often well-suited for the application that will handle them. However, as seen in the example above, the commands are very domain-specific and would not make much sense when scripting a different kind of game or application. These types of languages also lacks many useful features which another type supports, namely the procedural languages.

Systems based on procedural or object-oriented languages are widely used for game development. These scripting languages are often similar to C or Java regarding syntax and supports conditional logic and iterations. They are called high-level languages and are flexible and one specific language may be used in various contexts as opposite to the command-based languages. Since the code of the scripts are more complex than simple commands, procedural or object-oriented scripting system must include a virtual machine or interpreter to provide a way for programs to interact with the scripts. These parts will be described in more detail later in this chapter [39].

The third group of scripting systems are the dynamically linked modules systems. Here, instead of using a scripting language which needs to be interpreted by a scripting system in order to interact with the program, the scripts are written in the same language as the program. This increases the execution speed of the scripts. Increased execution speed is positive but this type of systems has some drawbacks. Since the code of the scripts runs on their own and not inside the program, the program has no control over what the script is doing, which obviously is not an ideal situation from a security point of view. Also, the syntax and features of the programming language may be a bit complex for the scripting purposes. Dynamically linked module systems are however successfully

used in games such as Quake[1] and Half-Life[2] [39].

4.2.1 Compiled and interpreted languages

As with the regular programming languages, the code of many scripting languages is compiled to machine-readable instructions before it is used in a program. Though, in some scripting systems the code is loaded into the program in its human-readable form where it is processed by an interpreter. This processing is not a trivial task and thus an interpreter is more complicated than a machine that runs compiled code [39].

The reason that many scripting systems use compiling is that it provides a number of advantages compared to interpreting scripts at runtime. First of all, it executes faster. Compiled code comes as numbers and is strictly ordered with one command on each row and the machine has the simple matter of executing basic commands, one row after another. The interpreter must perform many slow string comparisons to control commands and variables, and it must do many other demanding tasks, all which a compiler can perform in advance, before a script is loaded. Second, if the script code is compiled before being used in a program, syntactic errors can be detected by the compiler and can then be corrected by the writer of the code before trying to execute the script. It is often easier to deal with errors this way than to observe some malfunction during program execution and guess where in the script code the problem resides. Finally, scripts distributed in a human-readable form are easy targets for hacking and modification. For a multi-player game, this can ruin the game experience for many players since a hacker may edit some scripts to provide him with immortality, maximum strength or something else that ruins the balance of the game [39].

This does not mean that interpreted languages are without advantages. Since being interpreted at runtime, they can often offer a wider range of convenient operators and features like weak typing, which could boost productivity and agility for the script writers. The scripting process is also faster in an interpreted language, since any compilation and link steps are unnecessary. The procedure simply is to write the script and then run the code.

4.3 Scripting role playing games

Role playing games (RPGs) are often associated with scripting. They are suitable for scripting because they often contain a huge amount of game content. Compared to an action game where a player can be thrown into a world with a simple mission like "Destroy everything in your way!", RPGs tend to provide complex quests and stories, numerous items and weapons and also interesting interactions (both friendly and hostile) with other characters in the world. All these areas are appropriate to script and this section covers some general approaches for them.

4.3.1 Complex stories

At any given point in a game, the engine must know the status of the world and the player's accomplishments so far. If e.g. the player has completed a quest, maybe a village nearby has heard the rumor and will congratulate the player upon arrival. To support this, the common solution is for the game to maintain an array of flags, where

[1] http://www.idsoftware.com
[2] http://half-life.com/

each position often has a boolean value representing a state for a specific event [39]. Scripts may access the array through functions and act according to the information. In the earlier example there may be a position in the array which indicates whether the quest has been completed or not, and when the village script is executed it examines the value and congratulates if it is set to true.

4.3.2 Non-player characters

A distinguishable feature of most RPGs is the ability to make conversation with many of the NPCs inhabiting the world. To make these conversations realistic and believable, the NPCs must adapt to the player's actions, both according to what the player says during conversation and what the player has done earlier in the game. To handle this, a NPC is often associated with a script. First of all these scripts need to provide the NPC with the ability to speak. Through scripts, the NPC may also need to trigger suitable animations, give items to a player, let the player choose what to say from a number of alternatives and continue the conversation in different ways depending on the answer. Furthermore, it would be nice to prompt a player differently depending on the number of times he has approached the NPC.

These features require that the scripts are able to read and write data that is stored in the game and to act appropriately they also require conditional logic. To keep track of earlier conversations with a player and maintain this information between game sessions, NPCs need to have their own game flags which can be altered and saved by the game. Implementation of a conversation script is then a matter of checking variables and game flags and adding suitable actions [39].

4.3.3 Items and weapons

Large numbers of items and weapons often exist in the world of a RPG and many of them have some kind of special abilities. Since these may differ quite alot for each item, it is a good idea to represent each item with a script. These scripts mainly need to be able to do two things; alter game characters statistics such as hit points and play animations and sounds. When an item is used, the corresponding script examines the context through some conditional statements, alters some variables for involved characters and provides some visual feedback to the player [39].

4.3.4 Enemies

In many RPGs, a major part of the time spent consists of fighting enemies. Defining an enemy may often be thought of as setting values for such attributes as strength, speed and agility. While these attributes may be useful, this approach doesn't provide much flexibility and creativity when creating enemies. By representing an enemy with a procedural script with conditional logic, it can specifically be defined how it will act in various situations and it is easy to introduce new ways of fighting to the game. Basic fighting scripts may be implemented which all enemy scripts can refer to in order to reduce the amount of code needed for each enemy [39].

4.4 Architecture and design of scripting systems

The previous sections have introduced different types of scripting systems and some features required to script a RPG. The need for conditional logic and iterations in RPG scripts makes the procedural systems an interesting choice for a more detailed study. With the sole intention of using an existing scripting system without concerning about how it works, it may not be necessary to delve into more detail than what has been covered so far. However, it will be useful knowledge when estimating whether developing a new language could be a viable option for this thesis.

4.4.1 Compiled systems

Basically, a procedural scripting system using compilation can be divided into three components; high-level code, low-level code and the virtual machine[39]. The high-level code is what is often associated with the system, since it is the language in which the scripts are written by humans. It is translated by a compiler to low-level code, which is simply assembly language or machine code containing only basic instructions. The virtual machine runs inside host applications and is able to execute the low-level code. Since the virtual machine is integrated with the applications, it provides an interface between the scripts and the programs and allows them to communicate and exchange data. The relationship can be seen in figure 4.2.

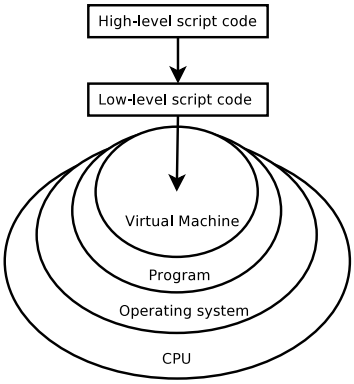

Figure 4.2: The virtual machine runs inside the host application and executes the compiled code

High-level code

High-level languages are designed to let programmers think and implement in an English-like manner, with an abstraction level more like the human mind than the mind of a computer [39]. Functions, conditional statements and support for iteration or recursion are some elements that support this. Designing such a language requires some thought and the language could preferably be influenced by languages such as C and Pascal, since that would provide a familiar base for the syntax. The design of the high-level language is crucial since it will be the part that the human users will spend most of their time with.

Low-level code

Low-level code or assembly language consists of one-line instructions and operands that are straight-forward and suited to be handled by a machine. For best performance when read by a machine, the assembly code is further assembled to machine code prior to being executed. When using a scripting system, the structure of the assembly language conveniently does not have to be considered. To develop a new system however, the low-level language must be known and considered in detail. It is necessary to know how to represent features such as conditional logic, iterations and method calls. The richness of instructions must be considered to weigh usability against performance. A runtime stack must be maintained in a correct manner. These aspects require much thought, but for the assembly code to ever be formed, it must be created by a compiler.

The compiler

The compiler is responsible for converting high-level code to low-level code. Five phases can be identified that most compilers go through when doing this conversion. These are lexical analysis, parsing, semantic analysis, intermediate-code generation and target code emission [39].

Lexical analysis breaks the code into meaningful units called tokens. The parser then analyzes the syntax of token strings. Correct syntax does however not imply correct semantics, so the strings are analyzed further to search for semantic errors. If the code is correctly written, it is converted to intermediate code, which is something halfway between high-level and low-level code. Before further conversion, the i-code is optimized to remove redundancy and make the code as effective as possible. Finally, the assembly version of the code is created.

Compiler theory is a complex area, and in developing an own scripting system, the compilation is probably one of the most difficult tasks, at least if the result is to be effective, optimized low-level code.

The virtual machine

The virtual machine mimics the behavior of a real computer. The difference is, instead of using electrical circuits, the computations are performed in a programming language such as C. In this environment, a virtual processor and virtual memory are used. When designing a virtual machine, it is probably a good idea to be inspired by the architecture of a physical computer. Some important components that need to be included in the machine is an instruction stream which lists the instructions and operands of the low-level code, a runtime stack to keep track of positions in the instruction list and finally global data tables. To be efficient, the machine should also be multi-threaded to be able to handle multiple scripts simultaneously. Most important however is for the virtual machine to be able to interact with the host application, else it would be unable to accomplish anything useful [39].

As with compiling, creating a virtual machine from scratch is not trivial, and would require some effort to make it work well.

4.4.2 Interpreted systems

In interpreted systems, an interpreter replaces the virtual machine and it is able to parse and execute code directly in its human-readable form, as shown in figure 4.3.

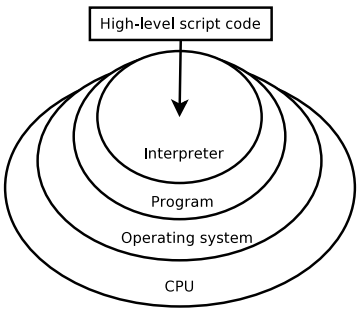

Figure 4.3: The interpreter executes the high-level code from inside the host application

When breaking up the interpreted systems in parts in the same manner as with the compiled systems, high-level code can be identified to have the same purpose and features in both systems. By using an interpreter however, the interpreted systems does not explicitly use a compiler and resulting low-level code in the same way as compiled systems. Roughly, it can be said that the interpreter replaces the virtual machine and eliminates the need for a compiler. The interpreter takes the high-level code as input and interprets each line at runtime. Most interpreters however performs an initial pre-compiling of the code upon loading it to increase the performance of the execution [39]. For example, a line of code that is iterated 100 times would have to be interpreted the same number of times without pre-compiling. This clearly shows the benefits of this procedure.

Designing and creating an interpreter is a challenging task. It has to perform the same things as the virtual machine does, but also do something similar to a separate compiler. To combine this is complex in itself, but further, everything the interpreter does has to be performed during runtime. The time it takes for the analysis and parsing of the code is thereby very critical, in contrary to a compiler where the time aspect is not that important. Thus, developing an interpreted scripting system is probably a bit harder than developing a compiled system.

4.5 Existing scripting systems

There exist many different scripting systems that can communicate with programming languages like C, C++ and Java. Many of them are open source and free to use. Python, Lua and Tcl are some languages that are well-known to the game developing community. Many companies also construct their own languages for their games to make them as useful as possible. UnrealScript for the 3D action game Unreal is an example. UnrealScript is an extensive, object-oriented scripting language which has been developed and used to define most of the elements in the game. Users can easily write their own scripts to extend and modify the game as they prefer.

The majority of game engines are written in C/C++. As a result of this, most of the scripting systems such as those mentioned above are implemented for integration with these programming languages and are incompatible with Java. For Java-based games, there exist some less famous scripting languages as well as a number of ports of the earlier mentioned ones to Java platforms. Regarding game development in Java for mobile phones, only a subset of these scripting systems are suitable. This is due to

the fact that some systems may be using parts of Java that is not implemented in the limited environment for mobile phones, the Java Micro Edition. The scripting systems also take up an amount of space called a footprint and this needs to be kept low due to the memory constraints of mobile phones.

A review of six Java-based scripting systems is presented below, with its focus on the systems suitability for mobile games. The main aspects that are interesting when trying to identify this is:

- The size of the systems footprint
- The syntax and features of the language
- Whether the system relies upon any classes not included in J2ME or not
- The documentation of the language
- How the system is licensed

With each language, a code example is presented to give a rough understanding about how code is written in that language.

4.5.1 FScriptME

FScript is a small and simple scripting language which intended use is as an embedded language in Java applications. FScriptME is an implementation of FScript designed for J2ME. It only exists as a beta version, but since it is based upon the FScript source code the developer thinks it should be fairly stable. FScriptME comes under GPL and may therefore not be used as part of a commercial product. Upon request, exceptions can however be made to this [15]. The footprint size of FScriptME is approximately 50 kB.

Code example

```
while count<500
   count=count+1
   value=value+calcvalue(count)
endwhile
```

4.5.2 Hecl

Hecl is a high-level scripting language implemented in Java and is based on the scripting language Tcl. It is designed to run on mobile devices and therefore has a minimal core, below 64 kB. Hecl is well documented on the project website, where examples show how to use the language and how to interface it with Java. Since Hecl comes under the Apache 2.0 license, it may be used in commercial applications [41].

Code example

```
set i 0
while { < $i 10 } {
   puts "i is now $i"
   incr $i
}
```

4.5.3 Jython

Jython is an implementation of Python, written in Java, which allows the use of Python on Java platforms. As a port of Python, Jython is a powerful scripting language with the features that has made Python so popular among game programmers. The language is also very well documented with an active community using it. However, the rich set of features comes with the cost of a huge footprint. The size of the standard build is around 900 kB, which is to much to be included in a mobile application. Further, without reimplementing the system to deal with the differences between the J2ME and J2SE, many features will probably be impossible to use together with J2ME [5].

Code example

```
from java.util import Random
rng = Random()
if rng.nextBoolean():
   print "Came up heads"
else:
   print "Came up tails"
```

4.5.4 LuaJava

LuaJava is a Java implementation of an interpreter for the Lua scripting language. LuaJava can be used at no cost for both academic and commercial purposes but uses the Java Native Interface (JNI) which is not supported in J2ME. Thus, LuaJava can not be used together with J2ME [7].

Code example

```
local id = tonumber(k)
if not id then
   return nil
end
```

4.5.5 Rhino

Rhino is an implementation of JavaScript, written in Java, to be embedded into Java applications. The footprint of Rhino is about 600 kB by default but it can be reduced to about 200 kB by removing various packages. As with LuaJava, Rhino uses reflections, a feature not implemented in J2ME. This makes it difficult to use Rhino in a mobile game [14].

Code example

```
var result = 0;
for (var i=0; i < a.length; i++)
   result += a[i];
```

4.5.6 Simkin

Simkin is a XML-based language to be embedded in Java applications. It features a special version for MIDP, which does not use reflections and other in J2ME unsupported features. This version has a small footprint of 87 kB. Simkin scripts consists of XML-tags defining attributes and values. The scripts are interpreted at runtime using a XML parser, but they may also be verified before execution since many XML parsers feature DTD-verification. In certain function tags, code that accesses Java functions and objects can be written in the Simkin scripting language. Simkin may be used in commercial products since it is licensed under the GNU LGPL [42].

Code example

```
<array>
<person><name>Simon</name></person>
<person><name>Lee</name></person>
</array>
<function name="main">
   num_children=array.numChildren;
   i=0;
   while (i lt num_children){
      trace(array[i].name);
      i=i+1;
   }
</function>
```

4.6 Conclusions

This chapter has presented some common approaches for writing game scripts and handling scenarios that often occur in role-playing games. It has shown how scripting techniques can be used in the game developed in this thesis to increase flexibility, efficiency and productivity during development.

Based on the examination of the general architecture and design of a procedural scripting system, it is assumed that the creation of a new system to be used with this game is a too complex and time consuming task to be included in this thesis. An existing system should therefore be used. Regarding the examined systems, restrictions have been identified which eliminates many systems from being suitable for the game. The systems must not only be able to integrate with Java, but more specificly, they must be well-suited for J2ME, by for example providing a way to expose application code to the scripts without using reflections and by having a footprint of a small size. Hecl and Simkin has been identified as seemingly suitable systems for the game.

Regardless of which scripting system is being used, the scripts must be well-designed and written so they facilitate and support all features and aspects required by the game.

Chapter 5

Design considerations

At the design stage of development, there are a number of aspects that need to be considered. In this chapter, design problems are presented along with suggested solutions. Both the problems and the solutions are based on the technical aspects of mobile gaming presented in chapter 3 and the study on game scripting in chapter 4. With these problems exposed and solutions proposed, a base is provided both for implementing the prototype and for, eventually, developing a commercial product.

One important aspect to design is the interaction between players in Furiae. There are two possible ways for players to interact; trading creatures and cooperative fight. These game elements need to be designed carefully to give the user enough freedom to not feel restricted and at the same time, provide few choices of actions so it is obvious to the user what the possibilities are. Technical considerations also need to be made in this area. As stated in chapter 3, there are several possible ways of communicating using a mobile phone and the usefulness of each technology strongly depends on the type of application. For the two user-interactive parts of the application network technologies and protocols must be chosen.

There are a number of other areas that need to be considered. A map system need to be designed to enable game characters to walk in front of and behind map objects. A scripting system needs to be chosen and the elements to be scripted need to be identified and outlined. The target platform should be powerful enough to accomodate a game like Furiae but common enough to make the game sellable. For the user to interact with the game in a convenient and intuitive way, choices for input design need to be made.

5.1 Interaction between players

Players can interact with each other in two ways, by trading creature connections and by fighting creatures together. Here, the considerations during the design of these game elements are presented.

5.1.1 Trade creatures

To get players to frequently use the game feature of trading creature connections with other players, this procedure must be carefully designed. Since it is stated that players should not be able to send text messages to each other in the game, the options for a

player must at all times be sufficient so players don't feel like their possible actions are limited. Trading should be a smooth procedure, easy to follow and control.

If a player is interested in proposing a trade to somebody, it is desirable to know what creatures the other players have. Showing a list of all creatures connected to all players in the world of Furiae doesn't seem like a feasible thing to do on the display of a mobile phone. The possibility for a player to choose another player and present that player's connected creatures would be a better suited approach for a mobile screen. Yet again, listing the names of all Furiae players for a player to choose from doesn't scale well. This list would be very long if the game gets popular. Instead, the user could be able to type the name of a player and, if the player exists, see the connected creatures. This approach should be well-suited for mobile phones.

One drawback with the typing names approach is that by playing the game on the phone, only the players with names known to a user can be found and added to the buddy list. The thought is however that players that want to be active traders also visit Furiae communities on the web where they establish contacts with other players. Also, it should be easy to obtain the game names of Furiae players in a players circle of friends outside of the game. Web communitites are also intended to be used when a player wants to get hold of a specific creature which isn't found at any known player. A person willing to trade it can be found there and they can then perform the trade from their phones.

For players who are often trading or exploring each others creatures, typing in the name each time may be a tedious process. Providing the players with a buddy list where other players can be added makes it easier for them to interact. Players having each other on a buddy list also probably increase the solidarity between them which may stimulate more frequent interaction. To prevent this list from getting too long, a maximum number of buddies should be specified. This can be motivated by three reasons. Firstly, if the number of buddies gets to large, the list will be as hard to overview as the list of all Furiae players mentioned earlier. Secondly, a restricted number of buddies may create a feeling of exclusivity for the ones being on the list. Thirdly, the shorter the list, the less amount of information must be transmitted to the phone to keep the list updated. Since experienced players probably will have more creatures and trade more than new players, a good idea could be to increase the maximum number of buddies as players advance in level.

To further increase the solidarity between buddies and to anchor the buddy concept in the world of Furiae, the buddy links are decided to be undirected. That is, if player A is on the list of player B, then player B must be on the list of player A. This can be thought of as a link between A and B and is the representation of the magic thread connections that wizards in Furiae can establish between themselves and other beings. The restriction of maximum links and the increase of this number when leveling also makes sense in this concept. A novice wizard will lack the power to maintain many connections but when the wizard grows stronger, more connections can be maintained. The fact that links are undirected means that before a link is established, both parties should have accepted to make this connection, that is, the establishment must be mutually agreed.

Due to the limitations of display sizes and input devices on mobile phones, it is necessary to make the trading procedure simple. An initial restriction that can be made is to only allow trades between linked players, as A and B described above. This will simplify the trade interface and also conceptually match the fact that trades only can be made between wizards that are linked to each other. Furthermore, it could be stated that only one trade may be ongoing concurrently over each link. Some players might be

annoyed by this but there shouldn't be much need to have simultaneous trades going on with the same person. In most cases, serialization shouldn't cause to much extra time. Presumably, this concern is outweighed by the benefit of a simpler interface which is easier for players to survey.

Since it is required that the game should be playable without being connected to any network, the trading procedure should be made asynchronous to allow trades between players without requiring them to be connected to the same network at the same time. Sent trade messages can then be handled by the recevier when appropriate, which may be many days after they were sent. With such delays in the messaging, it is important to make it possible for a sender of a message to cancel the message at any time if the answer is a long time in coming.

Finally, the actual exchange and display of trade messages needs some thought. The basic steps should be that player A sends a trade request and player B may accept or reject it. Player A then receives the answer and if the trade is accepted, it is carried out by the game. Something that will probably often occur is that player B rejects the request but wants to make a counter proposal. Thus, when a player rejects a trade, the game should automatically query whether the player wants to compose a counter proposal or not. When a trade message have been sent to a linked player, but the player hasn't yet replied, it should at any time be possible to view the sent message and cancel it at will.

Figure 5.1 visualises many of the considerations in this section by presenting from a player's perspective, the procedure of making a trade to get a desired creature. This flow chart will serve as a foundation for the implementation of the trade system.

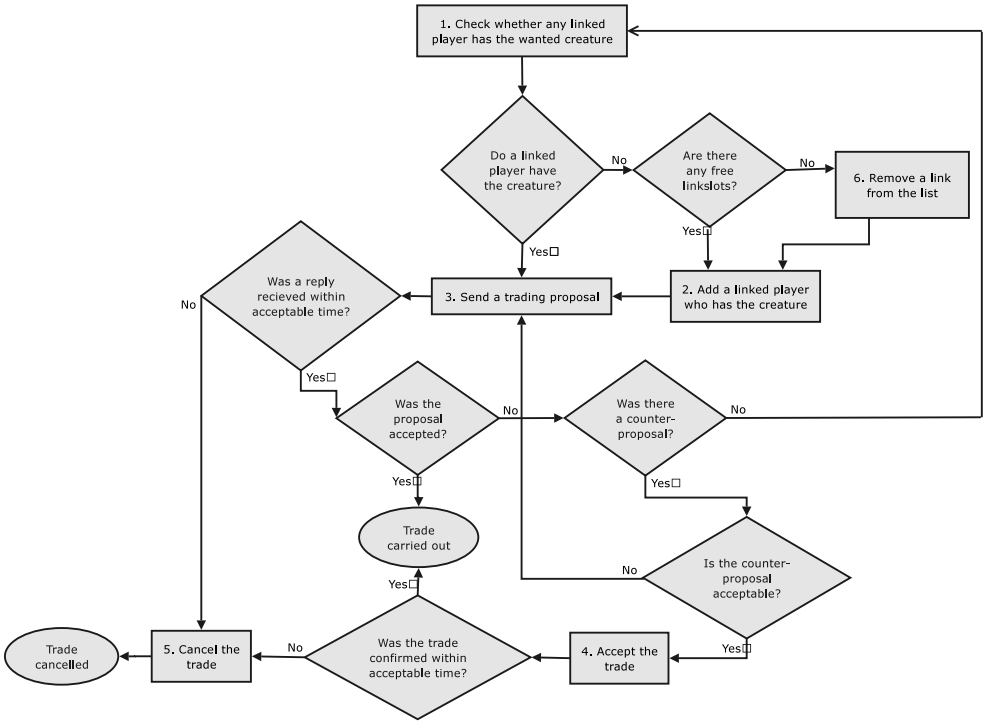

Figure 5.1: Trade procedure flow chart

5.1.2 Multiplayer fight

To initiate a cooperative multiplayer fight a player must first encounter a creature and invite others to join in. When the creature is defeated or the fight aborted, players will continue playing the game in single-player mode. This structure assumes that cooperative fights will be performed in small and sporadic sessions. An alternative could be coupling together a group of players and letting them fight all encountered creatures together during a longer time period. However, it is assumed that players rather will stay together for a shorter time period and that most fights will not be cooperative but usual single-player fights. This assumption was made based on the great variability of creature strength in the Furiae world, only particulary strong creatures will need cooperation to be defeated.

Master-slave structure

With two or more players in a cooperative fight, a need for control is raised. Every player involved must not feel trapped, there should be a way of disconnecting from a cooperative fight session. There should also be ways of communicating so that players performing actions that are not appreciated by others can be warned and kicked from the fight. Since one player is always encountering a creature and inviting others to join in, a centralized structure could be employed quite naturally. The player inviting others, the master, could have the power to end the whole fight and kick and warn invited players, or slaves. Slaves have less freedom but can disconnect from the fight at any time. An alternative to this appoach where all players have equal freedom could be employing voting systems, where any player can initiate a vote to e.g. end the fight or kick a player. The master-slave appoach is less complicated and faster but enables the master to exploit other players. A slave player could be invited to a fight and help the master to defeat a creature partially but later kicked, before the creature was defeated. Since experience points are given to the players alive when a creature is defeated, the master player could finish the fight and collect all of the experience points. The idea is that the system will be self-sanitary in this respect, players that have previously been exploited will avoid joining a new fight with an exploiting player as master.

Intentions

Since there are two ways of defeating a creature, kill or bind, a player should be able to communicate his or her intentions when inviting others to cooperate in a fight. Intentions are specified by the inviting player, the master, when inviting others. Thus, invited players won't just have information about the encountered creature but also about the masters intentions. An alternative could be having a chat system. The players could continuosly chat with each other and the intentions could be changed during the course of events. This would be more flexible but it is assumed that intentions will seldom change during a fight.

5.2 Target platform

To render possible Resolution Interactive's goal of the game being played by a wide range of customers spread over the world, the game must be made playable on the mobile phones which the targeted customers own. Based on the market investigation presented in Appendix B, decisions regarding this issue can be made.

That the prototype and the final game should be implemented in J2ME is already decided by the company before the start of this thesis. Making the game Java-based is a good choice, since Java is widely supported by all the major mobile phone manufacturers. All phones supporting Java do however not implement the same configurations and profiles. Based on how common it is that such configurations and profiles are implemented in popular phones and what benefits they would provide to the application, choices must be made whether or not to use them in the game implementation.

5.2.1 CLDC

Looking at the support for the versions of CLDC among popular mobile phones, the majority have got CLDC 1.1 implemented, but not all. However, since the main phone which will be used during the prototype development, the Nokia 6600, only supports CLDC 1.0, the application probably is not really in need of floating point arithmetic and the aim is to support as many phones as possible, the prototype is decided to be implemented using only CLDC 1.0.

5.2.2 MIDP

Appendix B and C shows that MIDP 2.0 is supported in almost all popular mobile phones. With this fact, together with the assessment that it will be hard to implement a complex game like Furiae without utilizing it, see section 3.3, MIDP 2.0 is chosen for the prototype.

5.2.3 Bluetooth API

As mentioned earlier, one aim with the game is to allow players to fight together and trade creatures with each other. If this is to be done from the game via Bluetooth, the phones must not only have Bluetooth connectivity that can be used by the operating system, but also an implementation of the Java Bluetooth API (JSR-82). This API is not that widely implemented, even if the phone itself supports Bluetooth. Yet, a number of popular phones supports the Blutetooth API and advantages of direct peer-to-peer and costless communication as opposed to GPRS traffic through a game server can be identified both for players and game developers. Also as it seems, support for the Bluetooth API will be more common in upcoming phones. These considerations are assessed to be enough to make it worth implementing Bluetooth communication.

5.3 Networking

As mentioned before, there are primarly three ways of data communication using a modern mobile phone; SMS, mobile data services and Bluetooth. Since SMS is a slow service with high costs it is not very appropriate for continuous data communication. In Furiae, there are two functionalities that needs to ultilize data communication: multiplayer fight and the trading of creature connections.

5.3.1 Trading creature connections

The adding of links to other players and trading of creature connections must to work very much like a messaging service since, according to the requirements, users should

not have to be online all the time when playing the game. It should be possible to send trade messages to offline players, creating a need for a central server storing these messages until the receiver gets online. Since Bluetooth is a short-range medium, mobile data services is the natural choice for communicating with a server. However, Bluetooth could serve as a great complement in situations where two users wanting to trade are within Bluetooth range of each other since it is free.

For communicating with the server TCP is chosen because of its flexibility and reliability. As mentioned in chapter 3.1.2, UDP isn't very reliable in mobile environments and HTTP reduces the flexibility and adds some overhead.

5.3.2 Multiplayer fight

In a multiplayer fight, participants are connected to each other and fight together against an enemy. The fight should proceed in a turn-based fashion; each participant gets to strike while other participants wait whereupon the enemy gets to strike. Thus, latency and bandwidth is not critical and both mobile data services and Bluetooth could be used. However, there lies a problem in implementing multiplayer fight using mobile data services.

To initiate a multiplayer fight, one user playing the game must establish an Internet connection to the ally that should be invited. As described in chapter 3.1.2, even if the ally is connected to the Internet, there is no way of making the connection without involving a proxy server. A solution is creating a proxy server which the mobile devices running Furiae stays connected to instead of being connected to each other. The proxy could simply forward all messages from one client to another based on some identifier like a username. This solution would require users running Furiae to always, or at least when they want to be invited to a fight, be online. But since one of the requirements of the application is that a user should not have to be online, the solution must be expanded.

When a user wants to invite an ally to a fight, an SMS notification could be sent. The SMS would be received and displayed even if the ally wasn't playing Furiae. He or she could enter Furiae, connect to the proxy and be set up to join the multiplayer fight. This solution isn't perfect since it yields a large latency; the sender will not know whether the receiver has read and reacted to the message until the receiver joins the fight which can take a long time since the receiver has to receive and read the message and possibly start the application.

Because of the problems with multiplayer fight via mobile data services and because implementing Bluetooth functionality provides a great learning opportunity and a good base for further extentions, multiplayer fight in Furiae is designed to work via Bluetooth only.

5.4 Input design

Most mobile phones are not designed as input devices for games. Considerations regarding which keys the player should use when playing the game is therefore important, since this input design will affect the users experience of the game. As well as deciding which physical input objects to use, the layout of navigation in menus and other parts of the game also needs to be designed with these limited input objects in mind. Ideally, the interaction with the game should always feel convenient and be intuitive to the user. What needs to be kept in mind during input design is that the layout and amount of keys

differs between phone models, so an input system which is well-suited for the majority of phones should be the target of the design.

The navigating device (the joystick, direction pad or direction buttons depending on phone model) is very commonly assigned as the object to navigate in phone menus and games. This familiarity for users and the ease of use of most navigators makes the navigator the obvious selection for navigating in this game as well, both on the map and in menus.

For selecting marked choices and performing highlighted actions, the action key which on most phones is placed in the middle of the navigator feels like the natural choice for the game, following the same reasoning as with the navigator. An alternative approach to where to place this key is to place it in the same way as on a controller to a gaming console like Xbox or Gameboy. That is, the player uses it with the thumb on the hand that is not controlling the navigator. However, due to the vertical positioning of the keys on a normal mobile phone (Nokia N-Gage is an exception to this), it is hard to find a placement of the action button where the hands get the same ergonomic position as on console controllers. When trying out this approach, an advantage of the previous one becomes obvious, which is that only one hand is needed to play the game. Furthermore, the thumb of that hand does not even have to be moved from the area of the navigating device. So, after comparing the two approaches, the action key in the navigator is chosen as selection key.

With keys for navigation and selection, most of the gameplay can be performed, but some way is required for accessing the in-game menu and backing one level in it or exiting it completely. Since the thumb is positioned on the navigator during play, the soft buttons directly to the left and right of the navigator, as well as any erase and back buttons, feel like a natural choice for this. Using these buttons makes the menu easily accessible with a small movement of the thumb. In the operating systems of mobile phones, the soft buttons are often used to select some text written in the lower corners of the screen, and are not used for going back in menus. Despite that, it should not be any problem for the players to get used to these buttons as menu buttons if this approach is consequently used throughout the game. So, to avoid confusion, text in menus may never be placed in the lower screen corners in a way that may mislead the user to beleive that it can be selected with a soft button.

The keys that are now chosen are the only ones needed for the game. This is a simplistic design, which should be easy for players to get familiar and comfortable with. It is also a design which should be well-suited for most phone models, however, some phones may lack an action key in the middle of the navigator or have a navigator that is only able to navigate in one dimension. To support these phones (and to provide an alternative input mode to players regardless phone), navigation in the game should also be possible with the number keys 2, 4, 6 and 8 and it should possible to use the number 5 key as the selection key.

To allow experienced players to access menus and options fast and directly, some additional shortcut keys could be used. These could tentatively be some of the unused number keys. However, with well-designed interfaces on the screen, shortcuts may not be that useful, so in the initial prototype, shortcut keys are not used. Should game testing in a later stage show the demand for such, a design and implementation of shortcuts may be performed.

The considerations about input design in this section only concerns phones where the keys are the only input device. Mobile phones with touch screens may have special layouts of the keys which may require an alternative design of the game input, but such

cases, if existing, are not considered in this thesis.

5.5 Map system

In Furiae, the player character should be able to walk on two-dimensional game maps inhabited by NPCs and creatures. The characters in the world should also be able to collide with and walk behind objects, a feature that creates an illusion of more than two dimensions and enhances the graphical experience of the game. Figure 5.2 shows a screenshot from the game with three characters inside a temple. A priest is standing behind an altar and there are two additional objects characters can walk behind, a doorway and a railing.

Figure 5.2: Screenshot from a temple

There must be a way to import game maps into the game, with objects and collision areas. This chapter will present and review two map systems inspired by existing games. Both map systems need a base map, see figure 5.3, and a collision map, see figure 5.4. The black area in figure 5.4 represents the part of the map where the player can't walk.

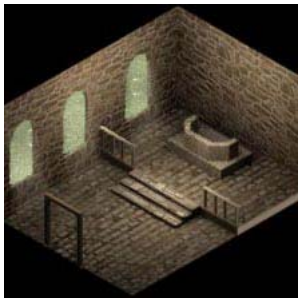

Figure 5.3: Temple map

Figure 5.4: Temple collision map

5.5.1 Map system one

This map system is founded on the idea that each area on the map will always either be above or under the player. First, the base map is painted, then all characters and lastly, an additional layer shown in figure 5.5 with objects that should cover characters. Maps and collision areas must be designed in a way that prevents characters from standing

in front of objects on the additional layer. That is, system one would not work for the temple example if characters weren't very short, shorter than the height of the collision areas in front of the railing in figure 5.4. With the character height in figure 5.2, the railing would cover a characters face if he was standing in front of it, beneath the collision area. The doorway works fine however because characters should always be covered by it and never cover it.

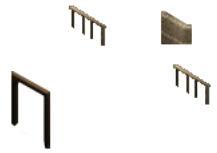

Figure 5.5: Layer above characters in the temple

Algorithm for showing the world with map system one:

1. Draw the base map (figure 5.3)
2. Draw all characters on the map, starting with the uppermost character
3. Draw the additional layer (figure 5.5)

Map system one is simple but it has some drawbacks:

- Since there can be no objects which a game character should be able to walk both behind and in front of, it is impossible to have a thin wall which a character should be able to stand in front of *and* walk behind.
- A maximal height for characters must be set to ensure that maps and collision areas will be designed right.
- When designing maps and collision areas, the maximal height of a character must be considered.

5.5.2 Map system two

In this system the map with black collision areas also include colored areas showing at which locations an object should cover a character. Each object is assigned a red color index on the RGB-scale which identifies it. For example, the railing in figure 5.6 could have index 1 and its area is painted with the RGB-value (1,0,0). The doorway in figure 5.7 could have index 2 and thus gets RGB-value (2,0,0). The doorway and railing would also be stored as separate images and there would be a file mapping each color code to a filename and a position on the map. A collision- and objectmap for the temple example is shown in figure 5.8.

Algorithm for showing the world with map system two:

1. Draw the base map (figure 5.3)

Figure 5.6: Railing object

Figure 5.7: Doorway object

Figure 5.8: Temple collision- and objectmap

2. For each character, starting with the uppermost character
   (a) Draw the character
   (b) For each colored area in the collision-map that the characters collision area intersects
      i. Draw the corresponding object on top of the character

In map system two, the restrictions of map system one are avoided. Any type of object can be represented and map designers have more freedom. One drawback may be the scalability of the system. If many characters on a map are covered by objects, many objects will be painted and the same object can be painted many times. However, minor modifications of the algorithm could overcome this issue.

For Furiae, map system two was chosen very much because the restrictions of map system one seemed unacceptable.

5.6 Scripting system

It is likely that in many cases when loading a map in the game, characters and other things may be initialized differently depending on some game states. This requires conditional logic. Other areas will also require this logic, like the conversation trees of NPCs where conversation choices and status of the player should rule how the NPC will act next. Many things are however just static attributes. The images of a map or a character, the name of a character or the description of an item are examples of this. Thus, it could be appropriate to write scripts for the game as a mixture of defined attributes and procedural code.

In section 4.5, a number of scripting systems that may be suitable for this game are reviewed, all which should be able to support scripts written in the recently defined manner. Out of those, Simkin is selected as scripting system for this thesis. The motivation for choosing Simkin is the possibility to use XML to define static values, the ready-to-use binary version of the system specifically compiled for MIDP and the rich

documentation of the system on the project website. However, as stated in section 4.2, Simkin scripts are interpreted at runtime, which may slow down execution.

The pre-defined approach for the game is that there should be scriptable adventures, which can be played one at a time, where each may take place on a number of maps. Additionally, different adventures should be able to use the same maps. With this in mind, it is decided to have a script for each map which defines the basic behaviour of it, such as how it is connected to other maps and what three-dimensional objects (as described in section 5.5) the map contains.

Each adventure can also be stored as a separate script. Adventure specific events, characters and objects for the maps involved in an adventure can then be added to them by defining this in the adventure script. In this way, the same maps can be used by many adventures. The adventure scripts must be able to alter a map differently depending on player progress and game states. Thus, game flags must be stored in the game, and the scripts must be able to define, alter and read them as well as have access to other information about the player.

Besides maps and adventures, the other main elements which will be represented by scripts can be classified as NPCs, creatures, items and spells. NPCs are the characters that walk around on a map which the player can interact and have a conversation with. These will often have a conversation tree, which will allow the NPC to approach the player in different ways depending on the situation. To do this, the NPC scripts, just as adventure scripts, must have access to player info and game flags. In addition, each NPC script needs to know how many times the player has interacted with the NPC, so the game must keep a record of this, as well as provide a way for the scripts to increase and read this value.

As opposed to NPCs with their conversation logic, the creatures, items and spells should be more suitable to represent mostly by static attributes

Chapter 6

Implementation

Based on the design solutions expressed in chapter 5, a prototype fulfilling the requirements stated in chapter 1.2 is implemented. For some areas, additional design considerations has to be made on a implementation-specific level. Some parts of the design also need to be revised due to technical aspects such as memory limitations. This chapter describes the implementation along with the new implementation-specific considerations and their solutions.

The most carefully designed areas in chapter 5 are the user-interactive parts, the map system and the script system. Naturally, these are also the most interesting areas to describe in further detail in this chapter. In addition to the MIDlet, a game server is developed mainly to facilitate the functionality for trading creatures by acting like a proxy. The game server implementation is described, along with the communication structure between the mobile phones and the server.

6.1 MIDlet overview

The MIDlet can be subdivided into the following modules as depicted in figure 6.1:

- Splash Screen - the main menu that is first displayed for the user when executing the MIDlet. Has functionalities for creating, loading and deleting a player.
- Game Engine - the game mode that meets the user when a player is created or loaded, allowing the player to explore the world by walking around on game maps.
- Bluetooth Server - the Bluetooth server is running whenever Game Engine is, handling incoming requests to join a battle from Furiae players in the Bluetooth neighbourhood.
- Fight Mode - this mode is executed when the player encounters a, possibly hostile, creature on the game map or when the player is invited to join a battle via Bluetooth.
- Bluetooth Client - is responsible for searching for other Furiae players in the Bluetooth neighborhood when a player wants to invite allies to join up against an enemy.

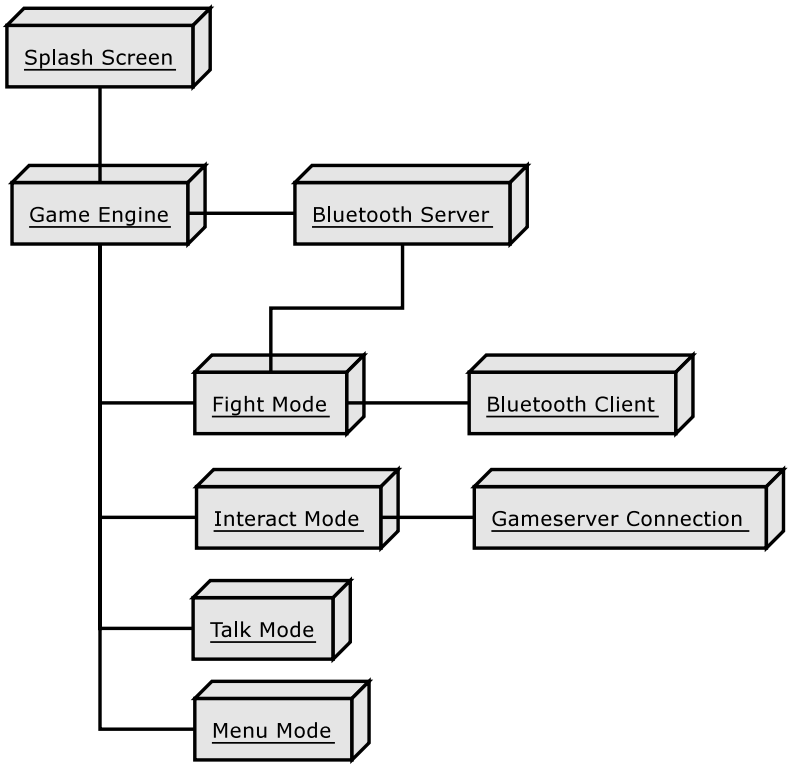

Figure 6.1: MIDlet overview

– Interact Mode - the game mode where the player can add new connections to and trade creatures with other players.

– Gameserver Connection - handles the connection between the MIDlet and the game server to allow for trading via the Internet.

– Talk Mode - the game mode for executing a talk session between the player and an NPC.

– Menu Mode - the game mode for displaying an in-game menu with player statistics, inventory and more.

6.2 Communication overview

Figure 6.2 shows communication paths and protocols in a typical situation. Three mobile phones, numbered 1-3, in a Bluetooth neighborhood are running Furiae simultaneously. Phone 4 is outside the Bluetooth neighborhood of the other phones but can communicate with them anyway, utilizing the game server. RFCOMM (Radio Frequency Communication) is the transport protocol used for Bluetooth communication [12].

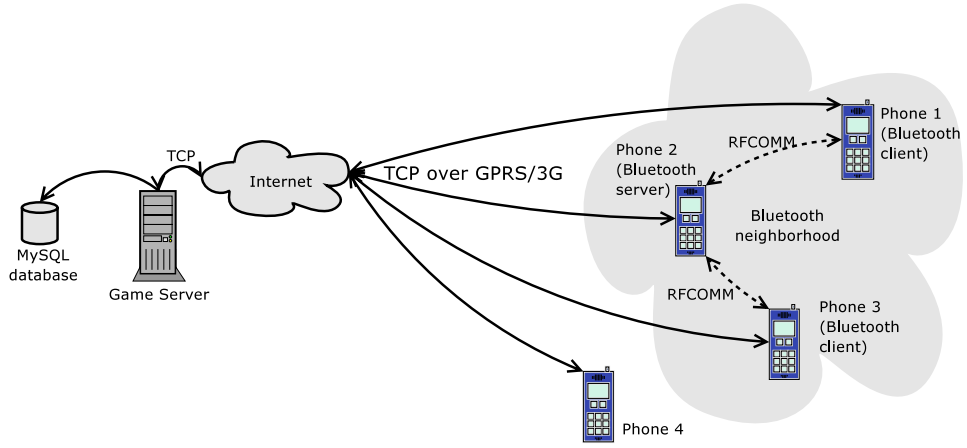

Figure 6.2: Overview of the data communication structure

The situation in figure 6.2 illustrates the highly centralized architecture of the system. The game server acts as a central server for all mobile phones running the Furiae MIDlet and phone 2 temporarily acts as a server in the Bluetooth neighborhood made up of phones 1-3. Being a server is not a role exclusive to phone 2, any mobile phone running the Furiae MIDlet can act as a server if it initiates a Bluetooth interaction. Thus, the server role is only temporary. If the current Bluetooth communication session in figure 6.2 is completed, phone 1 or 2 could initiate a new session and take over the server role.

6.3 Fight via Bluetooth

The Furiae game prototype includes the feature to invite other Furiae players in the Bluetooth neighborhood to participate in a battle against an encountered enemy. The

inviting application will always act as a server, issuing requests to connected clients according to the protocol in table E.1.

6.3.1 Server side

To use a Bluetooth service at a remote Bluetooth device and start communicating, the unit initiating a communication session must first perform a Bluetooth device search followed by a Bluetooth service search. A device search returns information about all discoverable Bluetooth devices in the neighborhood and the service search locates registered services at each device. When these steps have been undertaken, coupled devices can start communicating.

When a player wants to invite allies in the Bluetooth neighborhood to join a battle, a Bluetooth device and service search will be performed locally. Nearby discovered Furiae players will be listed and the user can select which players should be invited to join the battle. When players have been invited, information about the enemy, the server character and the characters of each connected player will be sent to each client. The server will then repeat the following steps:

- Prompt the user running the server to strike by selecting a spell to cast upon the enemy and distribute the strike among the connected clients.
- Prompt each client to strike, receive the information of the strike and distribute it to all clients.
- Let the enemy strike locally and distribute the strike among the connected players.

When the fight is over all connections are closed.

6.3.2 Client side

Every running instance of Furiae includes a Bluetooth server waiting for incoming requests to join a battle, making it discoverable for other Bluetooth units by registering a service identified with a UUID. The server is running whenever the application is, enabling the player to be invited to a cooperative fight at any time. When a request is registered, the Bluetooth server will start processing the clients commands. Thus, the mobile phone that made the request will take on the role as server, issuing commands, and the phone that received the request will take on the role as client, replying to the servers requests. When the request INVITE is received, the client will ask the user if he or she wants to join the battle. If he or she agrees to join, a multiplayer battle will be initiated. Since the server totally controls the battle, the client will do nothing but wait for the server to send information about strikes or prompt the client itself to strike.

6.4 Trade via mobile data services

The trading for this prototype is implemented as an asynchronous system which communicates using TCP connections over a mobile data service such as GPRS. Due to the network properties described in section 3.1.2, a server must be used in an initial stage to enable such data communication directly between two mobile phones. In this implementation, all phones communicate only with this server and no direct connections are ever made between phones. This client-server approach allows trades between players who

are not online at the same time, since the server can store messages and forward them to the recipient at a later time. If both parties in a trade are online, the server simply forwards messages between them as if they were directly connected to each other.

A communication protocol has been designed and implemented for the trading, which besides the actual trade messages involves adding and removing links between players and sending information about linked players and their connected creatures. The entire protocol can be found in table E.2. The same protocol can later be used if trading via Bluetooth is to be implemented. However, such a trading procedure can be made synchronous and without communicating through a server, which in that case requires the protocol to be revised and modified in a peer-to-peer fashion.

In section 5.1.1, a trade procedure is proposed which from a player's perspective is visualised in figure 5.1. The implemented trading procedure is based on this chart, with the intention to realize an asynchronous system without logical errors. A presentation of different parts of the implementation and the protocol now follows.

6.4.1 Connecting to the server

To be able to interact with other players, a player must connect to the game server. This is done by sending a LOGIN message to the server, who checks it to make sure the player is authorized to trade. If authorized, the player updates the server with information such as the characters current level and the currently connected creatures. Why this is important can be seen in the next step, which is for the player to download information about all linked players and what creatures they have connected. Because the occassions when players connect to the server are the only times the server can get information about things like connected creatures, the lists downloaded about linked players contain the information these players registered the last time they were online. This information may therefore not be accurate at the time of download. Due to this, it is recommended for active traders to connect to the server when major changes have been made to their list of creatures. This will minimize the risk that linked players propose invalid trades to them, which involve creatures no longer in their possession.

Finally, by sending a MESSAGES message to the server, the server replies with any messages the player has received since last time online. This may be trade messages or messages regarding opening or closing of links to other players.

6.4.2 Handling links

Having links to other players is required to be able to trade. To allow users to manage these links, the trade protocol contains the commands REQUESTLINK, ADDLINK and REMOVELINK. The server does not control these messages in any way, they are just forwarded to the recipient. If a player wants to establish a link to another player, a REQUESTLINK message is sent. If the other player agrees on establishing a link, an ADDLINK message is sent back and both players can examine each others info, which is automatically downloaded from the server. If two unconnected players would send a REQUESTLINK message simultaneously, no further ADDLINK message is required. A link between them will be established and information will be downloaded about the other player.

The server does not keep any record about links between players, this is stored locally at each phone. In the future it may be desirable to store link information at the server as well, but this will only require a simple addition to the server and will not require

any modification to the protocol.

6.4.3 Trading creature connections

As with link messages, the server stores and forwards trade messages without manipulating them. However, the server parses the messages to keep a record over ongoing trades. This is necessary to handle possible problems and special cases that may occur due to the system being asynchronous. This section describes how the trading is implemented and how it prevents problems from occuring.

In a gaming community it is likely that some people will try to find and exploit weaknesses in the design and implementation of the game. The trading is an area where much could be gained by exploiting such a logical error. To prevent this, the implementation of the trading has been thoroughly considered from this aspect.

In figure 5.1, a trade request is sent and if the receiver accepts it, the trade is completed. However, due to the asynchronous system the involved creatures can not just be swapped instantly. Consider player A sending a proposal to give a bird to player B. If B accepts, the trade could be seen as agreed and B could receive the bird. But if A cancels the trade at the same time as B accepts, A keeps the bird but B has also gotten it. This is an illicit situation, but it would be strange from B's perspective to then get a notification stating that the already completed trade has been cancelled and that the bird will be removed.

The previous example and similar situations explain why the server keeps a record over ongoing trades. Using this record in the example, a flag representing an ongoing trade is set when the server receives A's trade proposal. When B wants to accept the trade, it will only be permitted if the flag is still set at the server. If the flag is set, the trade is accepted and the flag is unset. If A now tries to cancel the trade, it will not be permitted since the server does not state that A and B have an ongoing trade. A has lost the bird. This situation is shown in figure 6.3. If the server however would receive A's cancellation before B's acceptance, the trade would be cancelled and B's attempt to accept would be rejected since the server states the trade is no longer going on. Figure 6.4 visualizes this scenario.

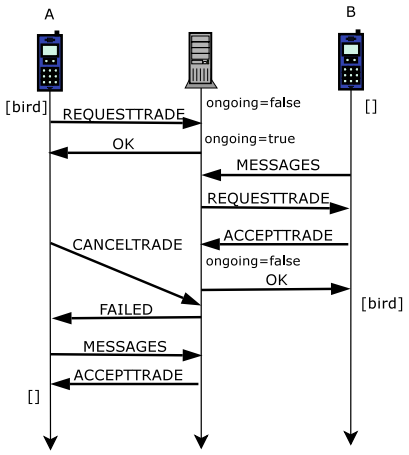

Figure 6.3: Trade where A offers a bird for nothing accepted

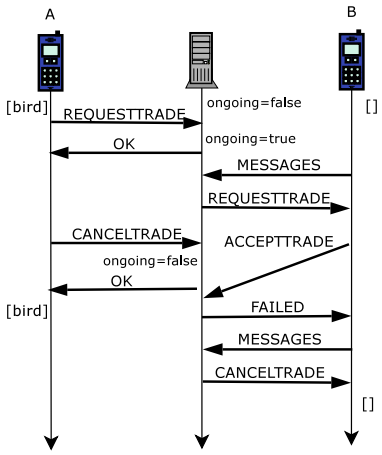

Figure 6.4: Trade where A offers a bird for nothing cancelled

To keep the world in Furiae consistent, the implemented trade protocol contains one additional step compared to the examples above. If B accepts the trade and gets the bird and A waits a long time before connecting to the server again, both will walk around in the game with that bird. With powerful creatures, this situation can be exploited and should be prevented. To do this, a creature involved in a trade becomes unusable for the connected player for the duration of the trade. Since players could be less inclined to propose trades if their offered creatures are affected, this situation is implemented not to arise until a trade is accepted. However, by doing this one additional step needs to be added to the trade procedure to maintain the desirable restriction. A trade thus requires three steps to be completed. Figure 6.5 shows the resulting procedure which is the one finally implemented in the game. For simplicity, the MESSAGES requests and all server replies are excluded from the figure. Noticeable is how A stays connected to the bird after sending the request, while B can not use the snake after accepting the trade. B must then wait for confirmation from A before a connection to the bird is establised.

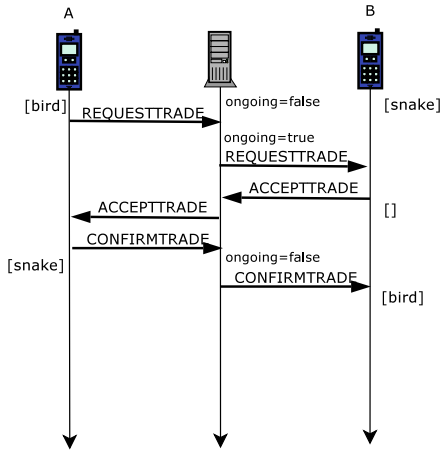

Figure 6.5: Standard trade procedure where A offers a bird for a snake

As a final detail in making a robust trading procedure, a players game progress is automatically saved after each session connected to the server. If this would not be the case, a player could give a creature to another player and then load a previously saved game where the creature is still connected. This automatic saving together with the implementation discussed above vouches for trading according to the rules of the game.

6.4.4 The event notification protocol

The protocol presented in table E.2 includes all commands needed in the interaction and trading between players. This is however a request-reply protocol. That is, for a player to get informed of a new event such as a trade request, the mobile client must poll the server with event requests (in this case the MESSAGES command) on a regular basis. To reduce network traffic, a better approach would be for the server to forward such information to all targeted clients connected to the server and store it for the targeted clients that are not connected. For this, the push protocol for event notification described in table E.3 is implemented and used. It consists of the two major commands UPDATE

and NEWMSG, which the server uses to notify clients. When connected to the server, the mobile clients continously listen for incoming notifications and present them to the user upon reception.

6.5 Map system

Implementing and testing the map system referred to as map system two, see section 5.5.2, on the test phones, some additions had to be made.

6.5.1 Game maps

When the a Furiae player is in walk mode, where he or she can explore a map and its inhabitants, it is desirable that all elements visible on screen are first loaded into video memory and then displayed to the user. This way, everything will be displayed at the same time and first displaying an empty map and then displaying characters is avoided. Maps in Furiae are up to 660*660 pixels which yields 435600 pixels and characters and objects also have to be drawn. However, the video memory of Sony Ericsson can only, as mentioned in section 3.4.4, store 400000 pixels. One solution is to split the maps into smaller chunks (220*220 pixels) and only load the chunks that currently need to be drawn into video memory. Since most mobile phone displays today have a resolution below 220*220, the maximal amount of chunks being drawn at the same time will be four, yielding a pixel amount of 193600 pixels. To ensure that the 400000 pixel limit will not be breached, map objects that should be shown on the screen concurrently must not be too large.

6.5.2 Collision maps

The images containing collision areas and areas where objects should be shown must also temporarily be loaded into video memory before colors can be extracted from them. Since these images are as big as the game maps, the video memory won't suffice here either. Instead of using the same method, splitting the collision maps into smaller images, they are processed and turned into text files describing collision areas and objects on a map. This reduces the file size substantially compared to storing the collision maps as images. It also avoids the need to having to use the MIDP 2.0 method `Image.getRGB` which have proven to work poorly on the test phone Nokia 6600, see appendix F for details.

The text files containing information about collision areas and objects according to map system two will have to contain an integer for each pixel on the layermap, see figure 5.8. Collision areas are represented with the integer 0, areas where the player can walk with 1 and all other integers up to 255 represent areas where objects should be inserted on top of a character. When representing each pixel as an integer, a smaller file size could however be achieved by compressing the information. A simple run-length encoding algorithm is employed [24]. Several repeated integers could, uncompressed, be represented like this:

```
0 0 0 0 0 0 1 1 1 2 0 0 0
0 0 0 0 1 1 1 1 2 2 2 0 0
```

Instead, we write the integer together with the amount of times it should be repeated:

```
0*6 1*3 2 0*3
0*4 1*4 2*3 0*2
```

As seen in figure 5.8, there are typically large areas with the same index so the run-length algorithm decreases the file size dramatically. Also, due to the algorithms simplicity, the phone doesn't have to spend a long time decompressing.

6.6 Script system

All scripts for the game are written for the Simkin script system. As a result of this, the script files consist of XML-tags. The information in these tags may as in ordinary XML be static values, but specific function tags may have Simkin script code embedded in them which can be called from the application.

Since J2ME lacks introspection, the Simkin script code can not examine the application code, and can therefore not get access to any functions or variables in the application without them being explicitly exposed. This implies that each game mode that wants to make functions accessible from scripts must implement a "glue" between them. A solution for doing this in the Simkin system is to implement an interface called Executable, where everything the application should expose to scripts can be listed. This approach is used in the game prototype and is visualized in figure 6.6. When executing a method in a script, the implementation of the interface is passed as a reference and through it, the script can access any exposed code.

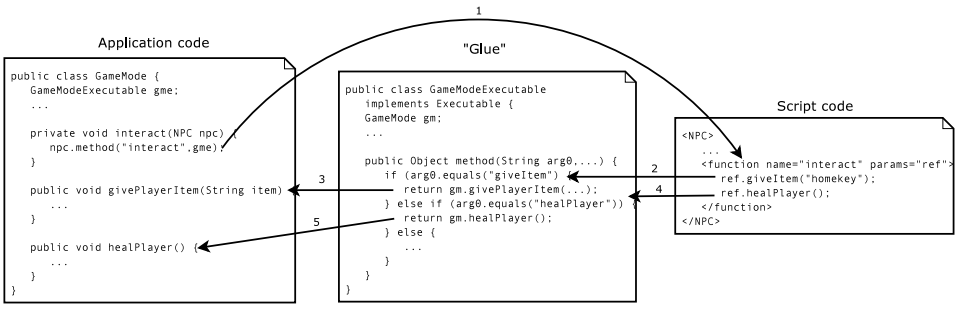

Figure 6.6: Accessing script methods and exposing application methods to scripts

In the game prototype there exist eight different types of scripts which are described below. Script syntax definitions (DTDs) can be found in Appendix C.

6.6.1 Adventures

When playing Furiae, the stories and happenings are part of an adventure. For example, NPCs may approach the player with quests, and creatures may appear at different places or change behaviour based on the players actions. Only one adventure can be played at one time but there is no restriction on the size of an adventure script. The adventure script describes what maps that are part of the adventure, which NPCs and creatures that will inhabit them and all adventure specific events that will happen on the maps. All adventure scripts must have a method named init which must contain Simkin code that specifies which map the adventure will begin on and sets the starting position of the player. This is because when a game is loaded from the main menu, the application

can not know which map to start from, so it executes the init method of the active adventure script with the intended result of loading the initial map for the adventure. For an adventure to add or alter something on a map to change it from its default behaviour defined in the map script, it can insert a method in the adventure script with the name of the targeted map. Inside that function, Simkin code can be written to insert NPCs or creatures or do other things, all which can be based on the status of game flags. Every time a map is loaded, this function in the adventure script will be executed in addition to loading the ordinary map script.

6.6.2 Maps

A map script holds information related to a painted map image. The primary role of a map script is to connect maps by defining trigger areas that causes the application to load another map. These areas are often located at the outer bounds of an image or at doors, gates or cave openings. The trigger actions are written in Simkin code so it is possible to write flexible events that alter their behavior depending on game flags or player status. They may also trigger many other events than map changes. Besides of the triggers, a map script can position NPCs or creatures that will inhabit the map, independently of which adventure is being played.

6.6.3 NPCs

Non-player characters often walk around on a map and conversate with the player. A NPC script defines the look and behavior of a these characters. It lists the characters name and move pattern and images for walk animations and a facial close-up. In addition, it has a method named interact which holds Simkin script code for the major part of most NPC scripts, the conversation tree. This method is called when a NPC and the player are about to talk to each other. The application provides methods for the script to perform actions like saying things to the player, prompting the player for answers, giving or taking items, teaching spells, checking and setting game flags and starting a fight. For a NPC to be able to fight the player, it must have an additional representation of itself as a creature script, which will be used in the fight.

6.6.4 Creatures

Creatures, as well as spells, shops and levels, are statically represented and their scripts contain no Simkin code, they solely have specified values for a pre-defined set of attributes. Many of these values are used when a creature is fighting and lists any resistances and vulnerabilities, minimum and maximum hit points and damage, if a connection can be established to the creature and the amount of experience gained for defeating the creature. As for NPCs, images and move patterns are also defined. For the case where the creature is connected to the player, there are values defining which energies the creature provides as well as the filename of a painted collector's image.

6.6.5 Spells

A spell script represents a spell that can be cast in battle and defines its cost, damage and icon as well as the name of any eventual audial or graphical effects to be played when the spell is cast. Listed are also the energies that the player is required to have to

be able to cast the spell and eventual special types of attack the spell uses. Finally, a value defines if the spell is mental or physical.

6.6.6 Items

For items that are not explicitly usable, their scripts only contain values defining their name, image, cost and a description. Scripts for items that the player can use, such as potions, must have a method named use which should contain Simkin code for execution upon use of the item.

6.6.7 Shops

A shop is a game mode where the player can buy or sell items. The intended way for a player to enter a shop mode is to talk to a merchant NPC, which then calls a method to run a shop script. In the prototype, the only information in a shop script besides its name is a list of the items the shop sells.

6.6.8 Levels

There exists only one level script and it lists all available levels in the game. Besides the number of the level, the script defines the levels rank title, experience required to reach the level and increases in limits for the player regarding hit points, mana points, number of links to other players and number of creatures connected. If the player should learn any new spells by reaching the level, these spells are also listed.

6.7 Game server

A game server is used to handle the sending and recieving of trades. Whenever a player enters the trade mode of the Furiae prototype, the application will connect to the game server and fetch new messages. Messages can either be trade requests or requests from another player to establish a link. The game server is developed in Java, with an underlying MySQL server storing data.

6.7.1 Application

Figure 6.7 shows the servers class diagram. TCPServ creates the server main thread, which starts listening to incoming TCP connections on a port. When a new client is connected a new thread, ServerThread, is started to serv the client. The ServerThread receives commands from the client and handles them, utilizing a DBHandler created by TCPServ to extract data from the database.

6.7.2 Database

The entity-relationship diagram (ER) of the game server database is shown in figure 6.8. The entity Player represents a registered Furiae character the isOnline flag indicating whether the player is online and various properties which can be shared with other players. The Link relation represents whether a pair of players have eastablished a link between each other and the Bound relation represents a link between a player and a creature. The Trade relation represents whether there currently is an ongoing

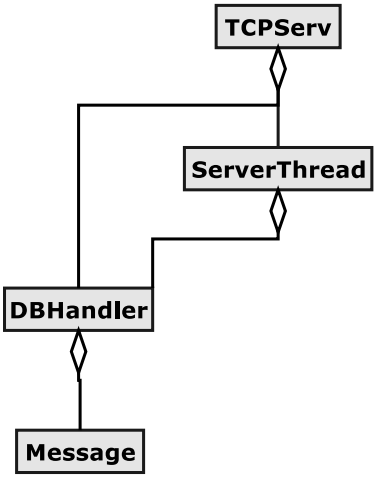

Figure 6.7: Game server UML

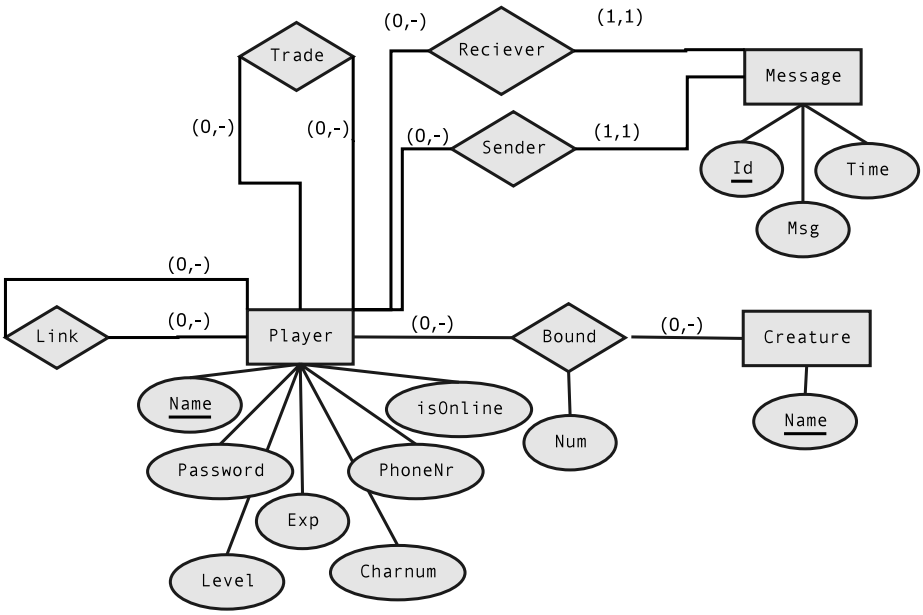

Figure 6.8: Game server entity-relationship diagram

trade between two players. Whenever a trade is being sent, this relation is checked to determine whether there already is a trade between the trading parts since there only can be one trade at a time between a pair of players. The entity Message represents a message, having a sending and a recieving player. When a player logs on to the server, all messages having this player as receiver is being fetched to the mobile phone.

6.8 Optimizations

The Furiae prototype is big for a mobile game, requiring almost 2.5 MB of phone memory when installed. It also has high demands in terms of heap memory, the memory which is used by the game during execution. To accomodate memory limitations and bugs in the test phones, several optimizations is made in addition to the map optimizations described in section 6.5.

6.8.1 Nokia 6600

Released in 2003, the testphone Nokia 6600 is the oldest testphone and also the one with the worst performance. The 6600 has an unlimited heap size but still benefits from smart allocation of variables. It was found that reusing old variables instead of throwing old ones away and creating new ones speeds up the gameloop dramatically. This phone executes the code much slower than the other test phones which is most apparent when loading maps into memory. The gameloop when in walkmode also executes slower than on the Sony Ericsson phones but the game is definitely playable. Calling the garbage collector explicitly blocks the application for several seconds and thus should be avoided.

Playing sounds continuously is another problem on the 6600. When performing tideous tasks, like loading a new gamemap, the sound being played simply stops. The apparent solution is to pause the sound before a map is loaded and start it again when finished. Also, the 6600 doesn't seem to be able to play several sounds simultaneously.

A list of additional discovered bugs and issues in the 6600 is presented in appendix F.

6.8.2 Sony Ericsson W550i and W810i

The testphones from Sony Ericsson has better performance than the 6600 but lacks unlimited heap size. Instead, a maximum of 1.5 MB can be used for both phones but testing has proved that the W810i can handle much more than the W550i. The phones also have a limited video memory of 800 kB, which lead to the modifications to the map system explained in section 6.5. To make the Furiae prototype work at the W550i, the code has to be reviewed and allocation of new variables minimized. It is found that the garbage collector does a much better job if variables that should no longer be used are explicitly set to `null`, telling the garbage collector to deallocate. Calling the garbage collector explicitly works better than in the Nokia 6600 but at some occasions it makes the W550i terminate the application. A much better result is derived from letting the main thread of the application sleep for a hundred milliseconds at critical parts, letting the garbage collector thread execute automatically. Most critical are the parts of the application where lots of information is loaded into memory, e.g. when new a map is loaded in walkmode. Despite the optimizations, the prototype sometimes deadlocks or crashes the phone when executing on the W550i.

The W810i was released in April 2006 and has no problems running the prototype, even without the optimizations made. If the memory of the W810i really isn't larger than on the W550i, the garbage collector must work much better on the W810i.

Chapter 7

Conclusions

The first goal of this thesis, to implement a prototype to be presented at E3 on May 10th 2006, is fulfilled. All of the required functionality listed in section 1.2 is implemented, even if the networking elements can be extended to work over additional network mediums. The game is runnable on all of the required phones but deadlocks or crashes sometimes on the Sony Ericsson W550i.

The prototype is a working application with many features and this report has exposed many of the problems implied by developing a game like Furiae and also suggested solutions which have proven to work. Thus, with implemented solutions like the ability to use scripts and communicate over Bluetooth and mobile data services, a good base for further development has been established and the second goal of this thesis is fulfilled.

The final size of the prototype game is 2.5 MB, which is big for a mobile application. Many of the phones in the market investigation in appendix C have a storage memory of less than 3 MB and thus will probably not be able to store the application. The data from the market investigation is, however, not totally up to date and many phones are considered old. New, more and more advanced models are being released and todays most powerful phones will soon be available for the average consumer as prices fall. Application size is probably a factor which should be considered if the application is to be further developed into a commercial game.

Using large images as maps in a J2ME game is problematic since many phones have restricted video memory, but the approach in section 6.5, dividing a large image into several smaller ones, significantly reduces the problem. Heap size is also a restricting factor which calls for smart variable handling to facilitate garbage collection, a feature which works with varied efficiency on different phones. Since memory is very limited it is important to keep track of the memory consuming parts of an application. Sadly, the memory monitor supplied by Sony Ericsson and used in this project did not perform well, executing the prototype at a very slow pace which made it as well as useless.

When developing applications for mobile phones, it is important to have access to real phones for continous testing. Phone models can differ from each other both in Java implementation and memory structure, even if they should work similarly according to the phone vendors. Two examples of this are the undocumented bugs in the Nokia 6600 and the memory- and garbage collecting differences between the Sony Ericsson W810i and the W550i. The emulators supplied by phone vendors does not behave in the same way as corresponding real phones and many things that work well on emulators does not work on real devices. The emulators are, however, useful and convenient since they

load and execute the application faster than real phones, resulting in a more efficient implementation process.

Conclusively, J2ME is a constantly developing platform which provides good opportunities for developing mobile games. Mobile development can however be quite a problematic area due to the hardware restrictions and differences in J2ME implementations.

Chapter 8

Future work

The developed prototype for Furiae is supposed to be further developed into a commercial game. Even if the prototype does work as is, the game is not complete. New features need to be added and a web applet will be developed to work as a complement to the mobile game.

8.1 Optimizations

Since the prototype doesn't work well enough on one of the testphones, the Sony Ericsson W550i, more optimizations have to be made. Since calls to the script functions in Simkin seem to use a large amount of memory, these calls have to be minimized. More effort also has to be put into reviewing the code to null and reuse more variables.

8.2 Security

Currently, the phones communicate with the server using TCP sockets, sending unencrypted messages in plain text. Since some of the information may be sensitive, like passwords, effort has to be put into encrypting. It is also important to protect the server and clients to attacks. A simple solution which would correct some security issues would be using SSL-sockets. Since SSL is supported in many phones this would be an easy task but the range of possible phones for running Furiae would be more limited.

8.3 Scalability

Currently, the game server has only had to handle up to five simultaneous clients. When the game is released as a commercial product the amount of clients will increase dramatically. The server have to be tested and possibly modified to handle many simultaneous threads. The standard JVM in the server should be replaced with a JVM ideal for server applications.

8.4 Trade and cooperative fight

Currently, trading is always performed via the game server but trades are also supposed to work live via Bluetooth. The medium for trading should be reasonably transparent to the user and the application should trade via Bluetooth if possible and via mobile data services if the linked player to trade with is not in the Bluetooth neighborhood.

Multiplayer fights should work in a similar way. In the ideal case a user can invite allies to cooperate in a fight via both Bluetooth and mobile data services at the same time, utilizing both mediums. Since multiplayer fights rely on a centralized network architecture, the only requirement is that the master player is connected to all the members of a fight session.

8.5 Downloading maps and adventures

The Furiae prototype with its prototype adventure suffices for testing but to release a commercial game more adventures have to be developed. The motivation for scripting adventures is very much that adventures should be downloadable in the future. Once the player has a version of Furiae, he or she can download new maps to the phone via mobile data services and without replacing the application itself. To realize this idea, the database where an application can store data, the RMS, must be large enough to accomodate a whole adventure. Research have to be made to investigate RMS size and loading times from the RMS with large amounts of data for different phones.

8.6 Testing

The Furiae prototype has been successfully tested on three mobile phones so far. Even without changes, it is safe to assume that at least some models will execute the application but many models have specific bugs and issues that have to be handled. Sadly, the only way to safely assess that the application will work on a phone model is to test it on the phone. Emulators are supplied by most phone vendors but experiences from this project have shown that they have poor correspondance to real phones.

The prototype also need to be tested by users to identify bugs and get feedback on gameplay and interface design. Such sessions are scheduled at the time of writing of this report.

8.7 Java verification

To release a J2ME application commercially, verification is important for the application to be considered serious. Furiae has to be reviewed and modified to fulfill the UTC, sent to a testing house and, eventually, signed.

8.8 Web applet

Furiae is, when released, supposed to work both as a mobile game and as a web applet. The user should be able to switch between playing on the phone and on the web, seamlessly. Thus, the server have to contain information on the players state. When switching from mobile to web applet, game flags have to be uploaded to the server.

Chapter 9

Acknowledgements

We would like to thank the following people for helping us during our work:

- Anders Broberg - our internal supervisor. Thank you for providing helpful suggestions during the report writing.
- Matti Larsson - our external supervisor at Resolution Interactive. Thank you for providing us with the opportunity to perform this thesis project and for the support during the whole period.
- Everyone else at Resolution Interactive for providing input during the implementation phase and interesting lunch conversations.

References

[1] Andersson, N., Broberg, A., Bränberg, A., Janlert, L., Jonsson, E., Holmlund, K., and Pettersson, J. Emergent interaction a pre-study. Tech. rep., UCIT, Department of Computing Science, 2002.

[2] Anttila, J., and Lakkakorpi, J. On the effect of reduced quality of service on multiplayer online games. *IJIGS 2*, 2 (2003), 169–173.

[3] Cnet. 10 most popular asian phones. Web page, May 2006. `http://reviews.cnet.com/4520-6454_7-1009643-1.html`.

[4] Hillebrand, R., and Wierlemann, T. Guidelines for the mobile internet. `http://mobileinternetguide.org`, 19 Jan. 2003.

[5] Jython project. Jython. Web page, 2006. `http://www.jython.org/`.

[6] Karlberg, L. T630 - Årets mobil i sverige. Web page, Jan. 2005. `http://www.nyteknik.se/art/38289`.

[7] Kepler Project. Luajava - a script tool for java. Web page, 2006. `http://www.keplerproject.org/luajava/`.

[8] Knudsen, J. Midp application security 2: Understanding ssl and tls. Web page, Oct. 2002. `http://developers.sun.com/techtopics/mobility/midp/articles/security2/`.

[9] Kurose, J., and Ross, K. *Computer Networking*, 3rd ed. Pearson Education, 2005.

[10] Lidström, N. K700i mest sålda mobilen hos the phone house 2005. Web page, Jan. 2006. `http://mobil.mkf.se/ArticlePages/200601/12/20060112143116_MKF435/20060112143116_MKF435.dbp.asp`.

[11] Magerkurth, C., Cheok, A., and Mandryk, R. Pervasive games: Bringing computer entertainment back to the real world. *ACM Computers in Entertainment 3*, 3 (July 2005).

[12] Miller, B., and Bisdikian, C. *Bluetooth Revealed*. Prentice Hall, 2001.

[13] Mobiledia. Best sellers. Web page, Jan. 2006. `http://www.mobiledia.com/shop/`.

[14] Mozilla.org contributors. Rhino - javascript for java. Web page, 2006. `http://www.mozilla.org/rhino/`.

[15] MURLEN. Fscript. Web page, 2006. http://fscript.sourceforge.net/.

[16] MYRATEPLAN.COM. The 20 most popular cell phones researched online by consumers last week. Web page, Jan. 2006. http://www.myrateplan.com/cell_phones/bestsellers/current/index.php?week=011506.

[17] NEFFENGER, J. The volano report. http://www.volano.com/report/, 30 May 2003.

[18] NOKIA CORPORATION. *Games over Bluetooth: Recommendations to Game Developers*, 13 Nov. 2003.

[19] NOKIA CORPORATION. *MIDP 2.0 : Introduction to Using Sockets and Datagrams v 1.0*, 8 Mar. 2004.

[20] NOKIA CORPORATION. *Multiplayer Game Performance over Cellular Networks v 1.0*, 20 Jan. 2004.

[21] NOKIA CORPORATION. Nokia mobile games development. Web page, 2006. http://forum.nokia.com/games.

[22] PAGONIS, J. *GPRS Facts for the Internet Application Developer - Part I*. Symbian Ltd., July 2003.

[23] POSSI, P. Umts world. Web page, 2005. http://www.umtswold.com.

[24] SAYOOD, K. *Introduction to Data Compression*, 2nd ed. Academic Press, 2000.

[25] SCHILLER, J. *Mobile Communications*, 2nd ed. Pearson Education, 2003.

[26] SMED, J., KAUKORANTA, T., AND HAKONEN, H. Aspects of networking in multiplayer computer games. In *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century* (Nov. 2001), pp. 74–81.

[27] SONY ERICSSON MOBILE COMMUNICATIONS AB. *Developers Guidelines Java Platform, Micro Edition, for Sony Ericsson mobile phones*, tenth ed., Jan. 2006.

[28] SUN MICROSYSTEMS, INC. *Java Verified^{TM} Program Guide*, 3 Feb. 2004.

[29] SUN MICROSYSTEMS, INC. *Unified Testing Criteria (UTC) 2.0 for Java^{TM} Technology-based Applications for Mobile Devices*, 23 May 2005.

[30] SUN MICROSYSTEMS, INC. *JSR-000118 Mobile Information Device Profile 2.0*, 2006. http://jcp.org/aboutJava/communityprocess/final/jsr118/.

[31] SUN MICROSYSTEMS, INC. *JSR-000120 Wireless Messaging API*, 2006. http://www.jcp.org/aboutJava/communityprocess/final/jsr120/.

[32] SUN MICROSYSTEMS, INC. *JSR-000179 Location API for J2ME^{TM}*, 2006. http://www.jcp.org/aboutJava/communityprocess/final/jsr179/.

[33] SUN MICROSYSTEMS, INC. *JSR 82: Java^{TM} APIs for Bluetooth*, 2006. http://www.jcp.org/aboutJava/communityprocess/final/jsr082/.

[34] Taferner, M., and Bonek, E. *Wireless Internet Access over GSM and UMTS*. Springer-Verlag, 2002.

[35] Telia AB. Telia.se. Web page, 2006. `http://www.telia.se`.

[36] Topley, K. *J2ME in a Nutshell*. O'Reilly, Mar. 2002.

[37] Upton, E. Realtime multiplayer games over mobile networks. Tech. rep., Intel Research cambridge, July 2004.

[38] Vacirca, F., Ricciato, F., and Pilz, R. Large-scale rtt measurements from an operational umts/gprs network. In *Proceedings of the First International Conference on Wireless Internet* (July 2005).

[39] Varanese, A. *Game Scripting Mastery*. Premier Press, 2003.

[40] Webfinanser.se. Samsung-mobil toppar försäljningslista. Web page, Dec. 2005. `http://www.webfinanser.com/site/artikel.asp?pmid=639850`.

[41] Welton, D., and Kechel, W. Hecl programming language. Web page, 2006. `http://www.hecl.org/`.

[42] Whiteside, S. Simkin language. Web page, 2006. `http://www.simkin.co.uk/`.

[43] Wing, M. Consumer-grade global positioning system (gps) accuracy and reliability. *Journal of Forestry 103* (June 2005), 169–173.

[44] Wirefly. Wirefly showcases 10 most-popular cell phones. Web page, Nov. 2005. `http://www.wirefly.net/articles/1105/1105-top-ten-cellphones.htm`.

Appendix A

Abbreviations

3G - Third Generation Technology
API - Application Programming Interface
BREW - Binary Runtime Environment for Wireless
CDMA - Code Division Multiple Access
CDMA2000 1xEV-DO - Code Division Multiple Access 2000 1 x Evolution-Data Optimized
CLDC - Connected Limited Device Configuration
CS - Coding Scheme
CSD - Circuit-Switched Data
DTD - Document Type Definition
EDGE - Enhanced Data Rates for GSM Evolution
EGPRS - Enhanced General Packet Radio Service
ER - Enitity-Relationship Diagram
EV-DO - Code Division Multiple Access 2000 1 x Evolution-Data Optimized
FTP - File-Transfer Protocol
GPS - Global Positioning System
GPL - GNU General Public License
GPRS - General Packet Radio Service
GSM - Global System of Mobile Communications
HSCSD - High Speed Circuit Switched Data
HTTP - Hyper Text Transfer Protocol
IP - Internet Protocol
J2ME - Java 2 Platform, Micro Edition
J2SE - Java 2 Platform, Standard Edition
JAR - Java Archive
JCP - Java Community Process
JSR - Java Specification Request
JVM - Java Virtual Machine
KVM - Kilobyte Java Virtual Machine
LAN - Local Area Network
LGPL - GNU Lesser General Public License
MIDP - Mobile Information Device Profile
NAT - Network Address Translation
NPC - Non-Player Character

P2P - Peer-to-Peer
PDA - Personal Digital Assistant
RFCOMM - Radio Frequency Communication
RGB - Red, Green and Blue
RMS - Record Management System
RPG - Role-Playing Game
RTT - Round-Trip Time
SMS - Short Message Service
SSL - Secure Sockets Layer
TCP - Transmission Control Protocol
TLS - Transport Layer Security
UDP - User Datagram Protocol
UML - Universal Modelling Language
UMTS - Universal Mobile Telecommunications System
UTC - Unified Testing Criteria
UUID - Universally Unique Identifier
WCDMA - Wideband Code Division Multiple Access
XML - Extensible Markup Language

Appendix B

Mobile phones

B.1 Sweden

The table below shows the most sold mobile phones during 2005[10] and 2004[6] according to the retailor The Phone House. The rightmost column shows a more recent list depicting the best selling phones in Telias stores during december, 2005[40].

	The Phone House 2005	The Phone House 2004	Telia december 2005
1	Sony Ericsson K700i	Sony Ericsson T630	Samsung SGH-X640
2	Sony Ericsson K750i	Sony Ericsson T610	Nokia 5140i
3	Samsung X640	Nokia 3410	SonyEricsson K300i
4	Sony Ericsson T290i	Nokia 1100	Nokia 3120
5	Nokia 5140i	Nokia 3510i	SonyEricsson T290i
6	Sony Ericsson T630i	Sony Ericsson Z600	SonyEricsson K750i
7	Nokia 2600	Sony Ericsson T230	SonyEricsson W800i
8	Nokia 3120	Nokia 3200	SonyEricsson K700i
9	Sony Ericsson W800i	Sony Ericsson K700	Samsung Z-140
10	Nokia 6230i	Siemens A52	Samsung SGH-X480

B.2 United states

The table shows the current bestsellers at Mobiledia.com [13] at 24th January 2006, the MyRatePlan.com's most popular mobile phones for the seven days ending in January 2006 [16] and Wireflys 10 most popular phones at 16th November 2005 [44]. These three vendors all carry a wide range of phone models from the market leaders Nokia, Motorola, Samsung and Sony Ericsson.

	Mobiledia.com	MyRatePlan.com	Wirefly
1	Motorola RAZR V3 (Black)	Samsung MM-A900	Motorola RAZR V3
2	Sony Ericsson W600i	LG Migo VX-1000	Motorola E815
3	Motorola E815	Motorola Razr V3	Samsung PM-A740
4	Motorola RAZR V3	Firefly	Motorola V330
5	Sony Ericsson Z520a	Motorola Razr V3c	Audiovox 8910
6	Samsung MM-A920	LG VX9800	Motorola V220
7	LG VX5200	Motorola Razr V3	Motorola i850
8	Motorola V557	Nokia 6101	Motorola i710
9	Samsung SCH-A950	Sony Ericsson W600	Motorola V188
10	Nokia 6102	Motorola V360	Motorola V260

B.3 Asia

This table depicts a top ten of the most popular mobile phones in Asian stores at November 10. The results are based on CNETs reader and telephone polls conducted with various retailers in Singapore[3].

	Asian phones
1	Sony Ericsson K750i
2	Motorola Razr V3
3	Sony Ericsson W800i
4	Nokia N70
5	Nokia 6030
6	Nokia 6021
7	Sony Ericsson W550i
8	Motorola Razr V3 (Magenta)
9	Nokia 3230
10	Samsung SGH-D600

Appendix C

Technical specifications on a selection of popular mobile phone models

Notes: Memory is the storage capacity of the phone restricting the size of an application. Jar size is the maximal size of a Java application. Heap size is the amount of memory available for applications during execution. A "-"-sign denotes that data is not specified by the vendor.

Model	Memory	Jar size	Heap size	Resolution	Colors	CLCD/MIDP	Network
Razr V3i	10 MB	-	800 kB	176 x 220	256k	1.1/2.0	GSM, GPRS
Razr V3	5.5 MB	-	800 kB	176 x 220	256k	1.1/2.0	GSM, GPRS
E815	40 MB	-	-	176 x 220	256k	BREW	GSM, CDMA
i850	2 MB	-	1.1 MB	176 x 220	256k	1.1/2.0	iDEN
i710	2 MB	-	1.1 MB	130 x 130	64k	1.0/2.0	iDEN
V557	5 MB	-	800 kB	176 x 220	256k	1.1/2.0	GSM, GPRS
V360	5 MB	-	800 kB	176 x 220	256k	1.1/2.0	GSM, GPRS
V330	5 MB	-	800 kB	176 x 220	65k	1.1/2.0	GSM, GPRS
V220	1.8 MB	-	800 kB	128 x 128	65k	1.1/2.0	GSM, GPRS

Table C.1: Technical details on phones from Motorola ($http://www.motorola.com$)

Model	Memory	Jar size	Heap size	Colors	Resolution	CLCD / MIDP	Network
6600	6 MB	∞	3MB	65k	176 x 208	1.0/2.0	GSM, GPRS
6230i	31 MB	500 kB	2MB	65k	208 x 208	1.1/2.0	GSM, GPRS
N70	20 MB	∞	∞	262k	176 x 208	1.1/2.0	GSM, WCDMA, GPRS
3230	6 MB	∞	∞	65k	176 x 208	1.1/2.0	GSM, WCDMA, GPRS
6030	4 MB	137 kB	500kB	65k	128 x 128	1.1/2.0	GSM, GPRS
6021	3 MB	125 kB	500kB	65k	128 x 128	1.1/2.0	GSM, GPRS
5140i	3 MB	125 kB	500kB	65k	128 x 128	1.1/2.0	GSM, GPRS
3120	473 kB	63 kB	205 kB	4k	128 x 128	1.0/1.0	GSM, GPRS

Table C.2: Technical details on phones from Nokia ($http://www.nokia.com$)

Model	Memory	Jar size	Heap size	Colors	Resolution	CLCD/MIDP	Network
SGH-X640	3 MB	-	-	65k	128 x 160	1.1/2.0	GSM, GPRS
Z-140	50 MB	-	-	262k	176 x 220	1.1/2.0	GSM, GPRS
SGH-X480	512kB	-	-	65k	128 x 160	1.1/2.0	GSM, GPRS
SGH-D600	4 MB	-	-	262k	240 x 320	1.1/2.0	GSM, GPRS

Table C.3: Technical details on phones from Samsung ($http://www.samsung.com$)

Model	Memory	Jar size	Heap size	Resolution	Colors	CLCD / MIDP	Network
W800i	34 MB	∞	1.5 MB	176 x 200	262k	1.1/2.0	GSM, GPRS
W600i	256 MB	∞	1.5 MB	176 x 220	262k	1.1/2.0	GSM, GPRS
W550	256 MB	∞	1.5 MB	176 x 220	262k	1.1/2.0	GSM, GPRS
Z520	16 MB	∞	1.5 MB	128 x 160	65k	1.1/2.0	GSM, GPRS
K750i	38 MB	∞	1.5 MB	176 x 200	262k	1.1/2.0	GSM, GPRS
K700i	41 MB	∞	1.5 MB	176 x 200	65k	1.1/2.0	GSM, GPRS
K300i	12 MB	∞	1.5 MB	128 x 128	65k	1.1/2.0	GSM, GPRS
T630/28	2 MB	-	256 kB	128 x 160	65k	1.0/1.0	GSM, GPRS

Table C.4: Technical details on phones from Sony Ericsson ($http://www.sonyericsson.com$)

Model	Memory	Jar size	Colors	Resolution	CLCD / MIDP	Network
LG VX-5200[a]	32 MB	-	65k	128x160	-	CDMA, AMPS
Siemens C65 [b]	-	-	65k	130 x 130	1.1/2.0	GSM, GPRS
Siemens SK65 [c]	-	-	65k	132 x 176	1.1/2.0	GSM, GPRS
Firefly [d]	-	-	7	-	NO	GSM/GPRS
Audiovox 8910 [e]	-	-	65k	128 x 160	-	CDMA/AMPS

[a] http://www.lge.com
[b] http://www.samsung.com
[c] http://www.samsung.com
[d] http://www.fireflymobile.com
[e] http://www.audiovox.com

Table C.5: Technical details on phones from other vendors

Appendix D

Script syntax

D.1 Adventure script DTD

```
<!ELEMENT adventure (maplist,function*)>
<!ELEMENT maplist (map+)>
<!ELEMENT map (npclist,spawnlist,triggerlist)>
<!ELEMENT npclist (npc*)>
<!ELEMENT npc (x,y,script)>
<!ELEMENT x #PCDATA>
<!ELEMENT y #PCDATA>
<!ELEMENT script #PCDATA>
<!ELEMENT spawnlist (spawnarea*)>
<!ELEMENT spawnarea EMPTY>
<!ELEMENT triggerlist (activetrigger*,passivetrigger*)>
<!ELEMENT activetrigger (x,y,width,height)>
<!ELEMENT passivetrigger(x,y,width,height)>
<!ELEMENT width #PCDATA>
<!ELEMENT height #PCDATA>
<!ELEMENT function #PCDATA>

<!ATTLIST map
script CDATA #REQUIRED>
<!ATTLIST spawnarea
action CDATA #REQUIRED>
<!ATTLIST activetrigger
event CDATA #REQUIRED
onlyonce (true|false) #REQUIRED>
<!ATTLIST passivetrigger
event CDATA #REQUIRED
onlyonce (true|false) #REQUIRED>
<!ATTLIST function
name CDATA #REQUIRED
params CDATA #REQUIRED>
```

D.2 Creature script DTD

```
<!ELEMENT creature (name,img,largeimg,framewidth,frameheight,
xstart,ystart,xsize,ysize,hp,mindamage,maxdamage,exp_reward,
resistances,vulnerabilities,capturable,move_pattern)>
<!ELEMENT name #PCDATA>
<!ELEMENT img #PCDATA>
<!ELEMENT largeimg #PCDATA>
<!ELEMENT framewidth #PCDATA>
<!ELEMENT frameheight #PCDATA>
<!ELEMENT xstart #PCDATA>
<!ELEMENT ystart #PCDATA>
<!ELEMENT xsize #PCDATA>
<!ELEMENT ysize #PCDATA>
<!ELEMENT hp EMPTY>
<!ELEMENT mindamage #PCDATA>
<!ELEMENT maxdamage #PCDATA>
<!ELEMENT exp_reward #PCDATA>
<!ELEMENT resistances (resistance*)>
<!ELEMENT resistance EMPTY>
<!ELEMENT vulnerabilities (vulnerability*)>
<!ELEMENT vulnerability EMPTY>
<!ELEMENT capturable #PCDATA>
<!ELEMENT move_pattern #PCDATA>

<!ATTLIST hp
max CDATA #REQUIRED
min CDATA #REQUIRED>
<!ATTLIST resistance
name CDATA #REQUIRED
max CDATA #REQUIRED
min CDATA #REQUIRED>
<!ATTLIST vulnerability
name CDATA #REQUIRED
max CDATA #REQUIRED
min CDATA #REQUIRED>
```

D.3 Item script DTD

```
<!ELEMENT item (name,img,,description,cost,usable)>
<!ELEMENT name #PCDATA>
<!ELEMENT img #PCDATA>
<!ELEMENT description #PCDATA>
<!ELEMENT cost #PCDATA>
<!ELEMENT usable #PCDATA>
```

D.4 Levelscript DTD

```
<!ELEMENT levels (level+)>
<!ELEMENT level (spellist)>
<!ELEMENT spellist (spell*)>
<!ELEMENT spell #PCDATA>

<!ATTLIST level
num CDATA #REQUIRED
exp CDATA #REQUIRED
maxhp CDATA #REQUIRED
maxmp CDATA #REQUIRED
maxspells CDATA #REQUIRED
maxlinks CDATA #REQUIRED
maxmp CDATA #REQUIRED
maxcreatures CDATA #REQUIRED
ranktitle CDATA #REQUIRED>
```

D.5 Map script DTD

```
<!ELEMENT map (objectlist,triggerlist,function*)>
<!ELEMENT objectlist (object*)>
<!ELEMENT triggerlist (activetrigger*,passivetrigger*)>
<!ELEMENT activetrigger (x,y,width,height)>
<!ELEMENT passivetrigger(x,y,width,height)>
<!ELEMENT x #PCDATA>
<!ELEMENT y #PCDATA>
<!ELEMENT width #PCDATA>
<!ELEMENT height #PCDATA>
<!ELEMENT function #PCDATA>

<!ATTLIST map
img CDATA #REQUIRED
battle CDATA #REQUIRED
width CDATA #REQUIRED
height CDATA #REQUIRED>
<!ATTLIST activetrigger
event CDATA #REQUIRED
onlyonce (true|false) #REQUIRED>
<!ATTLIST passivetrigger
event CDATA #REQUIRED
onlyonce (true|false) #REQUIRED>
<!ATTLIST function
name CDATA #REQUIRED
params CDATA #REQUIRED>
```

D.6 NPC script DTD

```
<!ELEMENT npc (img,largeimg,framewidth,frameheight,xstart,
ystart,xsize,ysize,move_pattern,talks,answers,function*)>
<!ELEMENT img #PCDATA>
<!ELEMENT largeimg #PCDATA>
<!ELEMENT framewidth #PCDATA>
<!ELEMENT frameheight #PCDATA>
<!ELEMENT xstart #PCDATA>
<!ELEMENT ystart #PCDATA>
<!ELEMENT xsize #PCDATA>
<!ELEMENT ysize #PCDATA>
<!ELEMENT talks (talk+)>
<!ELEMENT talk #PCDATA>
<!ELEMENT answers (answer*)>
<!ELEMENT answer #PCDATA>
<!ELEMENT function #PCDATA>

<!ATTLIST npc
name CDATA #REQUIRED>
<!ATTLIST talk
pos CDATA #REQUIRED>
<!ATTLIST answer
pos CDATA #REQUIRED>
<!ATTLIST function
name CDATA #REQUIRED
params CDATA #REQUIRED>
```

D.7 Shop script DTD

```
<!ELEMENT shop (name,itemlist)>
<!ELEMENT name #PCDATA>
<!ELEMENT itemlist (item*)>
<!ELEMENT item #PCDATA>
```

D.8 Spell script DTD

```
<!ELEMENT spell (name,img,maxdamage,mindamage,cost,sound,
type,effect,attacklist,energylist,creaturelist)>
<!ELEMENT name #PCDATA>
<!ELEMENT img #PCDATA>
<!ELEMENT maxdamage #PCDATA>
<!ELEMENT mindamage #PCDATA>
<!ELEMENT cost #PCDATA>
<!ELEMENT sound #PCDATA>
<!ELEMENT type #PCDATA>
<!ELEMENT effect #PCDATA>
```

```
<!ELEMENT attacklist (attack*)>
<!ELEMENT attack EMPTY>
<!ELEMENT energylist (energy*)>
<!ELEMENT energy #PCDATA>
<!ELEMENT creaturelist (creature*)>
<!ELEMENT creature #PCDATA>

<!ATTLIST attack
name CDATA #REQUIRED
maxdamage CDATA #REQUIRED
mindamage CDATA #REQUIRED>
```

Appendix E

Protocols

Server request	Client reply	Effect
GET_NAME	'player name'	Returns the name of the connected player
DISCONNECT	-	Disconnects the client
INVITE 'player name','creature name','intention'	OK / DECLINE	Invites the client to join a battle against a creature
CANCEL	-	Sent if the server cancels an invite before the client has answered to it, cancels the invite.
KICK	OK	Kicks the connected player from the battle
WARNING	OK	The connected player will receive a warning message
NEW_PLAYER 'name','maxhp','hp','maxmp','mp'	OK	A new remote player is added to the clients view of the battle
PLAYER_INFO 'name','hp','mp'	OK	The client updates the player information on a player
CREATURE_INFO 'name','hp','mp'	OK	The client updates the info on the enemy
PLAYER_STRIKE 'spell name','effect'	OK	The client displays the player strike locally
ENEMY_STRIKE 'damage','target player'	OK	The client displays the enemy strike locally
END_FIGHT	OK	The client displays a message showing that the fight was ended and ends it
STRIKE	NOT_READY	Prompts the client to start performing a strike
PING	NOT_READY / RAN_AWAY / EQUIPPED / MADE_STRIKE 'enemy name','spell name','effect'	Sent while the client is striking to see if the client has made the strike. Will return NOT_READY if the strike isn't finished or the performed action if the strike is made.
GET_PLAYERINFO	'maxhp','hp','maxmp','mp'	The server gets updated information on the client
DISCONN_PLAYER 'player name'	OK	A remote player is removed from the clients view of the battle
KICK_PLAYER 'player name'	OK	A remote player is kicked from the clients view of the battle
WAITING_FOR 'player name'	OK	The client displays a message telling the user which player is currently performing a strike

Table E.1: Bluetooth fight protocol

Client message	Server reply	Effect
REGISTER 'player name','password'	OK / FAILED	Register new player
LOGIN 'player name','password'	OK / FAILED	Login at server to start an online session
PLAYERINFO 'exp','level','face'	OK / FAILED	Update server with newest info about the client
CREATURES 'creature name'*	OK / FAILED	Update server with the client's current creatures
GETPLAYERINFO 'player name'	'level','online','face'	Get latest info about a player
GETCREATURES 'player name'	'creature name'*	Get the latest updated creature list for a player
MESSAGES	'message'*	Get any messages sent from other players since last time client was online
LOGOUT	OK / FAILED	Disconnects the client and ends an online session
REQUESTLINK 'player name'	OK / FAILED	Send a request to establish a link to a player
ADDLINK 'player name'	OK / FAILED	Accept a link request. A link is established.
REMOVELINK 'player name'	OK / FAILED	Close a link to a player
REQUESTTRADE 'player name', 'num offered', 'num wanted', 'offered creature name'*, 'wanted creature name'*	OK / FAILED	Send a trade request to a player
ACCEPTTRADE 'player name'	OK / FAILED	Accept a trade request
CONFIRMTRADE 'player name'	OK / FAILED	Invites the client to join a battle against a creature
CANCELTRADE 'player name'	OK / FAILED	Cancel a requested or accepted trade
INVALIDTRADE 'player name'	OK / FAILED	Notify that ongoing trade is invalid since it involves some creature not connected to stated player

Table E.2: GPRS trade protocol

Server message	Effect
NEWMSG 'player name', 'message'	Forward to client a message from another player
UPDATE CREATURES 'player name', 'creature name'*	Forward to client an updated creature list for a player
UPDATE INFO 'player name', 'level', 'online', 'face'	Forward to client updated info about a player
UPDATE LOGOUT 'player name'	Notify client that a player no longer is online

Table E.3: Protocol for instant event notification during online sessions

Appendix F

Discovered issues in the Nokia 6600

The following issues was identified in the Nokia 6600 implementation of MIDP 2.0:

- The phone deadlocks when using the method `Image.getRGB` with images with transparent areas.

  Example code:

  ```
  //load an image
  Image myimage = Image.createImage("/transparent.png");
  int[] rgbvalues=new int[myimage.getWidth()*myimage.getHeight()];
  //get the rgb-values from the image
  myimage.getRGB(rgbvalues,0,myimage.getWidth(),0,0,
          myimage.getWidth(),myimage.getHeight());
  ```

  Here, if `transparent.png` contains transparent areas, the phone will deadlock.

- `Image.getRGB` returns only the higher nibble of the color code.

  Example code:

  ```
  //load an image
  Image myimage = Image.createImage("/testimage.png");
  int[] rgbvalues=new int[myimage.getWidth()*myimage.getHeight()];
  //get the rgb-values from the image
  myimage.getRGB(rgbvalues,0,myimage.getWidth(),0,0,
          myimage.getWidth(),myimage.getHeight());
  //extract the blue index from the first pixel in the image
  int blueindex = rgbvalues[0] & 0x000000FF;
  //print the value
  System.out.print(blueindex+"");
  ```

  Here, if the actual blue index in `testimage.png` is 0-7, 0 will be printed. If the index is 8-15, 8 will be printed and so on.

The following issues was identified in the Nokia 6600 implementation of CLDC 1.0:

– The character encoding UTF8 is not recognized as UTF-8 when creating an InputStreamReader.

Example code:

```
//load a file
InputStream is = getClass().getResourceAsStream("/mytextfile.txt");
if (is == null)
         throw new Exception("File Does Not Exist");
//create an inputstremreader
InputStreamReader isr = new InputStreamReader(is,"UTF8");
```

Here, the encoding will not be recognized and an exception thrown even though UTF8 works fine on other phones and in the Nokia S60 emulator. Typing "UTF-8" instead of "UTF8" solves the problem.

– Filenames cannot include extra slashes ("/") and folders cannot include white spaces.

Example code:

```
//load an image
Image myimage = Image.createImage("/My Folder//testimage.png");
```

The issue can be solved by replacing white spaces in the folder name and removing the extra "/": `/My_Folder/testimage.png`