KTH Numerical Analysis
and Computer Science

Performance Critical Multiplayer Game Development for the mophun™ Gaming Platform

Gustaf Westerlund

TRITA-NA-E03160

NADA

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Performance Critical Multiplayer Game Development for the mophun™ Gaming Platform

Gustaf Westerlund

TRITA-NA-E03160

Master's Thesis in Computer Science (20 credits)
at the School of Electrical Engineering,
Royal Institute of Technology year 2003
Supervisor at Nada was Lars Engebretsen
Examiner was Johan Håstad

Disclaimer

This thesis is submitted in accordance with the requirements for the Master's degree (*civilingenjör*) in Electrical Engineering at the Department of Numerical Analysis and Computer Science at the Royal Institute of Technology of Sweden. It is substantially the result of my own work except where explicitly indicated in the text. The thesis may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Performance critical multiplayer game development for the mophun™ gaming platform

Games development for mobile devices is associated with several specific limitations. To study these, I designed and implemented a game slightly based on Tetris®. The hardware resources in mobile devices are always severely restricted making performance the primary goal. The new virtual machine mophun, optimised for games, is to prefer to J2ME, due to the greater performance of mophun. The software design requires a pragmatic view, due to the performance limitations, and object oriented design must therefore give way. Some features of object oriented programming can however be used, for instance the destructor functionality.

Sammanfattning

Utveckling av ett tidskritiskt fleranvändarspel för den mobila spelplatformen mophun™

För att undersöka begränsningarna vid spelutveckling för mobila enheter, designade och utvecklade jag ett spel som påminner om Tetris®. Hårdvaruresurserna i mobila enheter är alltid mycket begränsade, vilket gör prestanda till det högst prioriterade målet. Den nya virtuella maskinen mophun, optimerad för spel, är att föredra framför J2ME, på grund av den överlägsna prestandan. Mjukvarudesignen kräver ett pragmatiskt synsätt, på grund av prestandabegränsningarna, och objektorienterad design kan därför inte användas i sin helhet. Vissa delar av objektorienterad programmering kan däremot användas, t.ex. destruktor-funktionaliteten.

Preface

This Master's thesis treats an attempt to create a multiplayer game using a mobile virtual machine called mophun™. It was mainly possible due to the great generosity of Synergenix Interactive AB just outside Stockholm. They have embraced me and even though they are struggling with a new product and limited venture capital they have found the time and commitment to assist me with all the help I needed to finish this thesis. Above all I would like to extend a great thanks to Anders "Scary" Johansson, Johan Andersen and of course my magnanimous supervisor Harald Walden. At KTH and Nada, Lars Engebretsen has been a great help in understanding all the formal requirements and he has also been very busy with the red ink, thank you kindly! When you read this I will probably have a great cause to thank Prof. Johan Håstad who hopefully will have had the benevolence to pass my Master's project. I would also like to thank him for connecting me with my Nada supervisor Lars Engebretsen and for helping me start the Master's project up. Finally I would like to thank Kerstin Frenckner who has guided me through the labyrinth of paperwork needed for a Master's project.

Stockholm 2003,
Gustaf Westerlund

Table of contents

1 Introduction

1.1 Project Overview

The gaming industry is one of the most rapidly growing industries in the world as is the mobile phone industry; only China has about 1 million new mobile phone users every month according to Kathrine Hogseth at Ericsson. Games for mobile platforms in general and multiplayer games in particular have a possibility of becoming a great new market. Development of software for mobile devices is quite different from software development in general and games software is quite different from normal software, making games development for mobile devices very different from normal software development. Mobile devices have very limited access to processing power, partly due to hardware restrictions and partly to the fact that other more important processes use a lot of the processing time. Since consumers pay for games there is also a very limited possibility of updating the software in the mobile device; this is, however, quite similar to the game consoles (Playstation 2, XBOX etc.). Games differ from normal software by the fact that games are usually real-time applications.

These limitations must be taken into consideration when choosing the game concept on which to make a game. The popular game Quake is practically impossible to develop on contemporary mobile devices, due to the limitations on processing power. There are also other restricting limitations when developing multiplayer games that use the mobile telephone network for connecting the players.

The network communication between two mobile telephones uses the contemporary second generation mobile telecommunication which is very different from network communication between computers on the internet. The throughput (bandwidth) and round-trip time (the time it takes to send a packet to the other node and back) are severely limited. This requires special

consideration when designing a game in order for the gameplay to not be too greatly impeded.

On the game console market native programs that communicate directly with the hardware are common. This approach gives near-optimal use of the hardware and is feasible only when the devices' hardware is static.

On the PC-game market, a modified version of native games are used, where the games do not directly communicate with the hardware. Rather, they use vendor-supplied dynamically linked libraries with a well defined application programming interface, to access the hardware. This impedes performance slightly but enables the hardware to be modified to a certain extent without modifying the software.

On mobile telephones the hardware of two models are practically never the same, restricting the use of native programs to one model at a time. Moreover, each mobile phone vendor releases a number of phones each year. Hence, it is not feasible to develop native applications for mobile phones; therefore virtual machines are used. The most common one is currently (autumn 2003) Sun's J2ME. This virtual machine, however, is not optimised for games but for software in general. The more recent virtual machine mophun™ has been developed for the sole purpose of running games. The performance of mophun compared to J2ME has been measured in several benchmark tests; they all show mophun to be faster, in some cases, several orders of magnitudes faster. Development for mophun is however restricted to assembly language or C/C++ whereas J2ME only uses Java, which has shorter development time and higher stability.

It has been shown that in order to develop stable software, lower the development time, facilitate maintenance and enable many programmers to work in parallel, object oriented analysis, design and programming, should be used. Ideally this should also be used when developing games for mobile devices but due to the severe performance limitations a more pragmatic stance is needed. Strict object oriented design is not advisable due to the

increased overhead in execution time and program size. Instead a more practical view of the design is required, taking the better of the two worlds of imperative programming and object oriented programming, while constantly having the project's main goals in mind: a compact and fast executable. This was the manner in which the method was chosen.

With this as a starting point I designed a game remotely based on the classic game of Tetris®. This game was also implemented to an alpha stage where it was run and tested on two mobile telephones. Some post optimisation was done and was very successful due to the fact that the optimisation was done in the correct part of the code. This was the code that was run often, and it was easily found with the help of the work done during the design. I then executed a test of where the processing time was used and it showed that most of the processing power was used for waiting for the graphics-bus to get ready, in other word, a hardware limitation, in other word, further optimisation was futile. Even though the strict object oriented programming paradigms were not used, I had great use of some of the object oriented features, such as the destructor functionality which made sure the program released its acquired resources properly.

To obtain further insight regarding the process of developing games on mobile phones I interviewed a member of a development team that developed a game on the same platform (mophun) but whose game design differed greatly from mine. Essentially, the interview showed that we agreed on most points.

1.2 Organisation of this thesis

The remainder of this thesis is organised into four main chapters. This chapter aims at giving a general overview of this thesis and the project. Chapter 2 is focused on the background and techniques of mobile telephones and games. It attempts to describe a real-time game (2.2), a normal game, is and what restrictions it sets. The concept of native programs and virtual machines are treated (2.3), and the virtual machine mophun is treated particularly (2.4). Other large techniques used for games

in mobile devices, are also described (2.5). To get a brief understanding of mobile communication using GPRS, it is treated in short (2.6). Since the game developed is a multiplayer game that uses a generic multiplayer server, the details are described (2.7). Chapter 2 ends with a short discussion on how mobile games are played (2.8).

Chapter 3, deals with the game design and the reasons for the chosen design. It treats an evaluation conducted on the platform that was used to get an understanding of what performance limitations existed (3.1). This was the base for the choice of game described (3.2). The programming design was chosen carefully after a close review of a piece of literature (3.3). The programming design choices are detailed in the following sections (3.4, 3.5).

Chapter 4 describes the implementation work (4.1) and evaluation of the implementation (4.2), also including a part on the post-implementation optimisation (4.3).

Chapter 5 attempts to draw a few conclusions from the previous work on the project (5.1) and dwells into the future of mobile gaming (5.2). Finally the conclusions from a similar project are compared to mine (5.3).

2 Mobile gaming - background & techniques

2.1 Related work

The mophun™ platform is quite novel and so is the possibility of writing more complex games than a simple version of Snake on a mobile telephone (can be found on most Nokia mobile telephones). Hence there is very little published material. In essence a mobile telephone is a real-time embedded system but the similarities in the relevant problems more or less end there since most software developers of embedded systems have a tight connection with the hardware design. That is not the case when developing a game for mophun™. Nothing that closely relates to the subject has been found in the literature.

2.2 A real-time game

Most games are real-time applications. The difference between real-time applications and non-real-time applications are that the former have the limitation of having to perform all operations within a strict time-frame. For computer games in general, at least 25 frames per second is required (more is usually preferred). A frame is a picture that is similar to a frame on an old 9mm-film-reel. A television has a frame-rate of 25 frames/seconds.

A simplified view of a computer is that it can only perform a certain amount of instructions per second. This limits the number of instructions that can be processed during each frame. The mophun virtual machine does not have a limitation of how many frames can be displayed per second, but the mobile telephone Sony Ericsson T300 has a limitation of 13 frames per second. A game that uses a lot of processor power might have less than 13 frames per second but that usually impedes game play, making the game seem “jerky” or “shaky”. The higher the frame-rate, the smoother movement in the game will be.

The working of the video architecture is very hardware dependent but a "normal" (not event based) game still has a more or less generic main loop, indicated in *Figure 1.* This loop should not take more time than 1/(required frame-rate) seconds. For instance, the T300 has a maximum frame-rate of 13 frames per second which would require the main loop to not exceed 77 milliseconds. The main loop is seldom static in its time allocation but it is enough that a large majority of the frames are under the requirement as long as the tops in processor usage are widely spread.

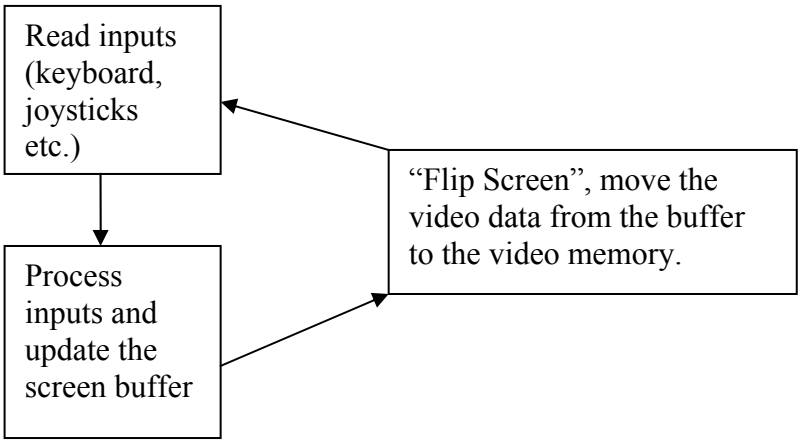
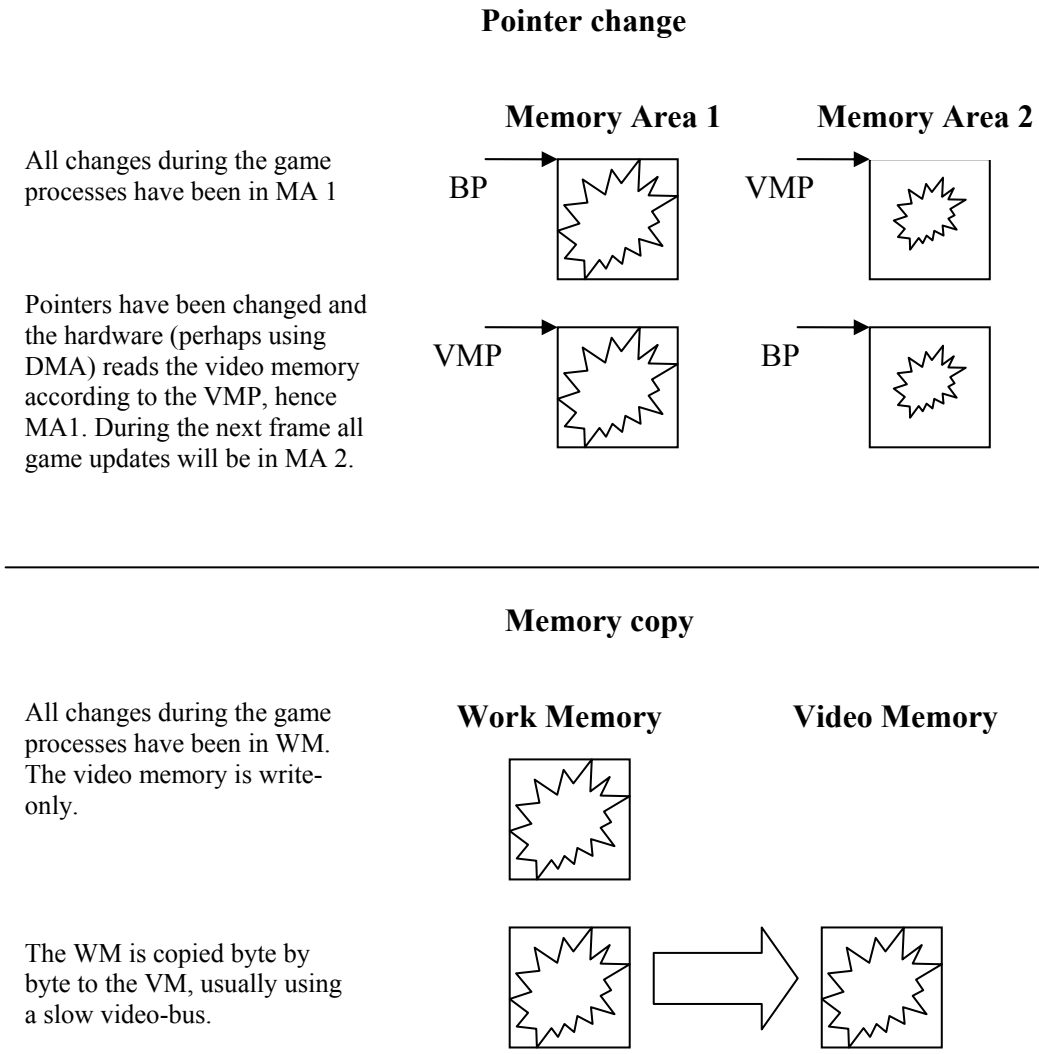


Figure 1. A typical main loop of a game (not event based). The sequence can be different between the parts.

Event based games are mostly found on Object Oriented platforms (Java, C#, Visual Basic etc.) and differ from "normal" games by the fact that as soon as any kind of input has been done (an event has been triggered), a new thread will execute a specific function. I will not describe such games in further details.

The "buffer" referred to in *Figure 1* is a workspace that having the same memory characteristics as the video memory. The buffer is used during the processing for changing the contents of the next screen update. When all processing is done, the video memory is updated according to the hardware. There are mainly two methods of updating the video memory: pointer change and memory copy. Which method is used depends on the hardware.



BP = Buffer Pointer, VMP = Video Memory Pointer, MA = Memory Area, WM = Work Memory (buffer in *Figure 1)*, VM = Video Memory

Figure 2. How a flip screen is done using the methods of pointer change and memory copy.

The difference is mainly in the implementation of the hardware: when using pointer change, the data is copied to the video memory by the hardware using Direct Memory Access (DMA) or using interrupts. Especially when using DMA, the processor does not have to handle the copying. The problem with using pointer change is that control is lost of when the copying is done. On the T300, memory copy is used due to the fact that the video bus is very slow and when utilising it fully the maximum frames per second is 13. On a TV, which has a raster beam, each new frame is shown every 1/25 s. There is a short time in-between the drawing of two frames

and when using a raster based technology, all copying has to be done during this short interval. When this technique is used, pointer-change is a lot more common, since timing the copying of the buffer to the video memory has to be very exact, and is therefore handled by the hardware (although it is not always used).

The LCD screen has not got a synchronised raster-beam and can be updated at any time, in a more asynchronous way. This is the main reason for the T300 using memory copying instead of pointer change so that the maximum frame rate can be achieved. In other words a frame rate of 12 frames per second can be achieved which would be very hard to get with a synchronised technology due to the fact that if the raster hardware had a normal hardware updating frequency of 13 frames per second, then a software that can only achieve 12 frames per second, might just get 6 frames per second due to an error in phasing. The mophun platform handles all this through the API and on another hardware platform; the other method of pointer change might be used. It is of no main concern to a programmer since it can not be altered.

2.3 Native programs & virtual machines

This chapter will attempt to show the differences between native programs and programs that run on a virtual machine.

2.3.1 Native programs

The most common manner in which games are developed for consoles in general and mobile phones before virtual machines is that each game is written exclusively for each hardware platform. This is in the context of virtual machines called native programs. Native programs have the huge advantage of being able to use the hardware to one hundred percent; however they also have one major disadvantage, lack of portability. When a game is developed for a hardware platform all the unique characteristics of the hardware has to be taken into account. For instance, to be able to display graphics on the display, a certain part of the memory is used, to be able to play a beep, a couple of values might be written to specific addresses of the

memory. The implication of all this is that the program can be run on only one specific hardware setup. If for instance, one of the addresses that had to be written to in order to play a beep is different on another hardware platform, then the program has to be fixed for this change. The result is that the porting of an application developed for one platform to another usually requires a huge amount of work. Due to the fact that parts of the code have to be changed, a new code branch has to be made for each model, complicating maintenance and distribution. This has not even taken into account the fact that each processor has a unique way of interpreting the binary code into instructions which means that a specific compiler has to be used for each platform. Another restriction is also that the hardware developer (i.e., Sony Ericsson) might also restrict which parties can write programs to be run natively, due to the security hazard of native programs, they might be viruses or other malignant programs.

If the hardware of several platforms is very similar and the performance requirements greatly outweigh the time and money put into a development project, then it might be feasible to develop something in native code. On the console market (Playstation 2, X-Box, etc.), more or less all games are native. The reason for this is that there is very little need for portability and the consoles do not support any kind of virtual machine in general. All the hardware for the consoles is very well known and very thoroughly specified in developer reference manuals. Consoles are also not replaced by the user at such a frequency as mobile telephones. If a game has been bought, the user will usually want it to work on future telephones as well, hence a large need for portability.

The modern PC uses native programs with a twist. Since PCs have different graphic cards, sound cards etc. it is not feasible to directly address the specific hardware since it would require each software developer to include a special version of the program for each hardware setup. Therefore the operating system has inserted an interface layer between the hardware and the specific software. Instead of directly addressing the hardware, the software calls functions in dynamically linked libraries (DLL). Theses

library files are supplied by the hardware manufacturer and it is in these libraries that the direct hardware calls are. This standard in Microsoft Windows is called DirectX with different sub-standards called Direct3D, DirectSound, among others. Please refer to *Figure 3* for a detailed description.

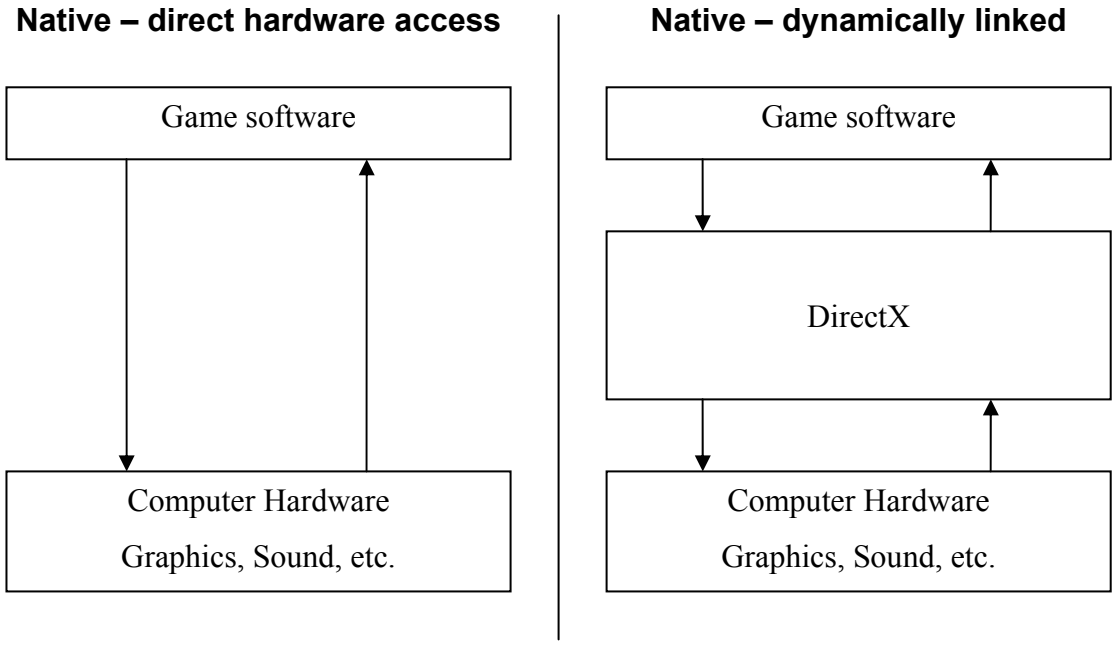


Figure 3: The difference between directly accessing the hardware and using intermediate functions to abstract the hardware. The Game software – DirectX connection is supplied in windows, and the connection DirectX – Computer Hardware, is supplied by the hardware developer.

2.3.2 Virtual machines

The concept of virtual machines has been around since the 1970:s (The language Pascal ran the so called p-code on a virtual machine [2]) and it attempts to solve the problem that different processors have different instructions and hence can not run the same program files. The solution is that instead of compiling a program to run directly on the hardware it is compiled to something called “byte code” that runs on a virtual computer or virtual machine. This virtual machine is very strictly defined and is then implemented on each hardware platform. The virtual machine program follows the strict definitions of how the byte code should be interpreted and attempts to run it.

2.4 The mophun™ Virtual Machine

The mophun™ platform is owned by Synergenix Interactive AB, a Swedish company that was founded in 1999. The creators of the company had noticed that the idea of a hardware independent gaming platform was desired by the mobile telephone producers. SUN had marketed their answer J2ME to a great extent. It was, in essence, a lightweight version of their fully fletched Java virtual machine. However Java has some inherent limitations and the two most important are that you cannot have a direct memory access and that it is made for all software applications and not for games in particular. mophun™ allows a developer to write assembly code directly to the virtual machine or write code in C or C++ which can be compiled with a modified version of the GNU C/C++ compiler gcc. The mophun API has full support for memory allocation functions, such as `malloc()` and `free()`. This solves the first problem with J2ME. mophun also includes an API (Application Programming Interface) that is created for the sole purpose of making games. That solves the latter problem.

To be able to create a truly hardware independent platform, all hardware specifics have to be removed and interfaced through a standardised API. The mophun™ API supplies this through several straightforward function libraries. It mainly covers input and output of the system and also a few portable data types, a stream library and a library for cooperative multi-tasking. There is also a special “Capabilities” library for testing what the current systems capabilities are.

The game part of the mophun API has several functions for handling sprites, or small pictures used in games, tiles, small pictures used to build repetitive background, and other functions to facilitate game development and handling of graphics which commonly is the most demanding part of games programming.

For further details on the mophun API please refer to the “mophun™ API Reference Guide” which can be found at www.mophun.com.

The mophun platform is developed for each platform independently and it is currently found in Sony Ericsson T300, T310, T610 (Spring 2003) and can be downloaded to mobile telephones with the operating system Symbian (Nokia N-Gage, Sony Ericsson P800 and many others).

2.5 Other mobile virtual machines

2.5.1 J2ME

Java 2 Micro Edition, J2ME, is more or less the industry standard concerning virtual machines for mobile devices. It is in essence a stripped down version of Java 2 (Sun Microsystems) and uses Java syntax. The main advantage of J2ME is that it is found in so many devices and the greatest disadvantage is its performance capabilities. These are mainly inherent in the language and are very difficult to by pass (ref. 2.6.2 ExEn). Concerning the development of games, it is extremely inferior to mophun mainly due to the fact that it has no game API and its poor game performance.

On low-end platforms, like the Sony Ericsson T300, J2ME is not an option but mophun is. No third party benchmarking to compare the virtual machines has been found but on the web-forum at www.wirelessgamingreview.com in the discussion on industry hash, some extraordinary figures are mentioned by several of the discussion contributors. It is stated that on a 600 Mhz PC the mophun virtual machine is some 150 times as fast as the J2ME virtual machine. In this discussion Antony Hartley at Synergenix (Tony Heartley in the forum) states that the current mophun virtual machine (1.1) can process 100 KIPS (Kilo Instructions Per Second) on a T300 with an 8 bit RISC-processor running at 12 MHz. In a short interview with Hartley, he states that the Siemens SL45 which has a 16 bit RISC-processor (he points out that the information – 32 bit – he submitted on the forum, was erroneous) running at about 16 MHz, in other words a lot quicker hardware, still can not get the J2ME virtual machine to exceed 20 KIPS. In other word the mophun virtual machine is at least 5 times quicker (if both devices had the same hardware performance)

than J2ME and probably more in the range of 10-20 times faster. A just comparison between the virtual machines will be difficult to conduct since each implementation is highly unique and many mobile devices have several processors and complex memory management routines.

It is also important to point out that this is an instruction-per-second comparison and it might not always be correct since the complexity of each instruction varies between different architectures. This comparison is also a bit misleading when comparing the virtual machines for games since games rely heavily on graphics and the virtual machines handle this differently and the device hardware is very unique.

2.5.2 ExEn

ExEn, Execution Engine, developed by In-Fusio, is in essence a J2ME virtual machine with a game API. Due to this, ExEn is significantly faster than J2ME when games are concerned. Since it is so similar to J2ME, an exact performance test is not applicable but according a review of the virtual machines [3], it can be up to 30 times faster. This review, however, clearly states that ExEn is inferior in performance to mophun. Again, it is very difficult to conduct a just test and each implementation will still be unique hence limiting the use of such a test and no official test has been found.

2.5.3 WGE

This chapter deals with the platform WGE. All information has been gathered from websites on the subject [4][5].

WGE or Wireless Graphics Engine is only partly a virtual engine. It is developed by the company TTPCom and is capable of running both Java and native code. Hence it is in essence two parts, a Java virtual machine and an environment set up to be able to run pseudo-portable native code. The native part of the WGE platform has tried to move the hardware specific calls from direct memory calls to standardised API-library calls, in the same way the DirectX works in Windows. A couple of libraries that access all the hardware are included in the platform and all applications running on it

dynamically link the function calls in these libraries to create some sort of portability without sacrificing performance too much.

This removes the problem of unique memory addresses for each platform but still does not solve the problem of the different interpretations of the binary code by the processors. TTPCom has created a work-around for this by requiring that all platforms that want to use WGE must have a certain processor.

There is a very limited amount of information available on the technical details of WGE since it has a very small market share. Due to the fact that it is able to run native programs it is one of the fastest technologies available.

2.6 General Package Radio Service (GPRS)

The GPRS extension of the Global System for Mobile communications (GSM) has been called the 2.5:th generation mobile telephone system. It is an addition to the existing GSM mobile telephone system and enables the telephone to communicate with the internet using packet based traffic, as opposed to the circuit-switched based normal GSM traffic using a modem. The main advantage is that the user only pays for the amount of data transmitted or received and not for the time connected. It is therefore more adapted to the characteristics of for instance WAP (Wireless Application Protocol) and mail which has very bursty traffic.

Due to the nature of GPRS and the fact that it rides on top of an already existing technology it has a huge lag of about 1.5 seconds point-to-point. In other words, a packet sent from one phone to another will not arrive at the other until 1.5 seconds later. This can be compared to what is said to be acceptable lag in normal multiplayer games played on a computer. If it is a real-time game (i.e., not turn based) lags of more than 200 ms or 0.2 seconds are regarded as far too large to be able to play. The lag on a GPRS connection is an order of magnitude larger. To create a real-time game like Quake or Doom with a lag of 1.5 seconds one will have to do huge amounts of extrapolations and it will become very hard to make a game that seems

just and even. This kind of lag limitation almost blocks the possibility of making a real-time game. A turn-based game like Chess or Backgammon is more suited, but not all people like those kinds of games.

2.7 A generic multiplayer server

Multiplayer games have been, are and will be the most popular games. At the dawn of computer games, there were no networks so all players had to play on the same computer. This kind of gaming is still appreciated and consoles like the Playstation 2 and the X-Box are designed for this kind of gaming even though not all games are for more than one player.

Games on computers have mainly been designed for one person at a time but there are exceptions here as well. For a couple of years it has been possible to play games on a computer connected with people all over the world. It has been so successful that even the consoles now have the ability to interconnect using the internet.

Mobile telephones are more or less restricted to only one user. To be able to play a game with a friend, two telephones are required and some sort of connection between them. In the past the most common way to play a game on a mobile telephone with a friend has been using an infrared beam. The problems with this are apparent. The phones have to be aligned at all times and the distance between them cannot be too great (more than a few decimetres).

Recently some mobile telephones have been developed that have Bluetooth™ technology and games connecting to each-other using Bluetooth™ won't have the alignment limitation but will still have a distance limitation (about 10 metres).

The latest step in this development has been to create games that use a GPRS connection and connect with each other on the internet. Now there is no longer a limitation on distance. However a new problem has arisen; the

internet is a vast place, how can we find our friend? A solution to this is a common game server that connects the players to each other.

Still there are problems, assume that Alice wants to play with Bob on the game server and they manage to establish a direct connection. What would happen if Bob was in a bus that just went into a short tunnel with no reception? The connection would probably be lost and they would have to reconnect, either halting the game or starting all over. If 8 players were playing at the same time, this problem becomes even more apparent. With no centralized infrastructure, maintaining a connection with 8 players all the time for all 8 players is an almost impossible feat. A solution to this is a game server that handles all the connections with the clients and distributes the necessary data among them. To develop this server software each time a multiplayer game is developed takes a lot of time and effort and hence the company Terraplay Systems have developed a generic multiplayer server that can not only run several games in parallel but several *different* games in parallel.

The Terraplay system consists mainly of two logical parts: the lobby and the gateway. In the lobby you connect to other players, either a random player in a public session or someone special in a private session. When enough players have joined together in the lobby, they are given a unique password and disconnect. Then they connect to the gateway server supplying their password as access code to the game created in the lobby after which they are connected via the server. Each client can create objects and assign keys to these objects. The created objects can be updated with data and each player can subscribe to the different keys. When an object associated with a key is changed, all subscribing parties are supplied with an object update automatically. When all necessary objects have been created the game starts (depending on the application). To keep the knowledge of if a player is connected or not, it is required of the player to send a heartbeat signal on a periodic interval. If the heartbeat signal is not received within this interval, the client is disconnected. If a player for some reason or other is dis-

connected it is possible to reconnect to the server and continue playing, if the game supports it.

If a game requires extra server functionality, that can be provided by a client that subscribes to keys different from the keys of the normal clients and updates objects different to that of the normal clients.

2.8 How are mobile games played?

This chapter is not based on any official research or literature on the subject but rather on personal experiences with mobile gaming and gaming in general.

Today, most people living in Sweden have a mobile telephone and they almost always bring it with them. Mobile games cannot compete with games on consoles (i.e., PS2, XBOX) or games on PCs hence the probability that a user will be sitting home in front of his console-equipped television set playing games on his mobile telephone for several hours, is quite small. It is more probable that a mobile phone owner will pick up his telephone on the bus, in a waiting room, on a train etc. to play a game while waiting.

It is therefore essential that a game either is short, like Tetris or Snake, or can easily be saved and resumed at a later time, like Chess. Since the Terraplay servers do not support long time persistent games (like a Chess game played for several days would be) the first alternative is the only real option for multiplayer games if a Terraplay server is going to be used.

The technology of mobile telephones also introduces limitations on the games. A connection with a user might be broken at any time and it might or might not be reconnected during an arbitrary time. The game has to take this fact into account and be designed so that these limitations have a minimal affect on the game play.

3 Design

3.1 Performance evaluation of mophun

As described in Section 2.5, the mophun platform on the Sony Ericsson T300 is very performance restricted. To be able to measure the performance a test was conducted, both with and without other tasks being run in the background. A program for evaluating the performance, a benchmarking program, was created and executed on the telephone in order to conduct this test.

3.1.1 Benchmarking program

The priorities of the parallel tasks in the telephone are set differently by the producer of the telephone, in this case Sony Ericsson. mophun is run in parallel with many other processes in the phone that have higher priorities. The effect of this can be that the virtual machine varies in speed.

To be able to study this, a test program was developed. The most important part of it was:

```
count = 0;
startmillis = vGetTickCount();
startmillis += 1000;
while (startmillis > vGetTickCount()){
            temp = count;
            count++;
}
```

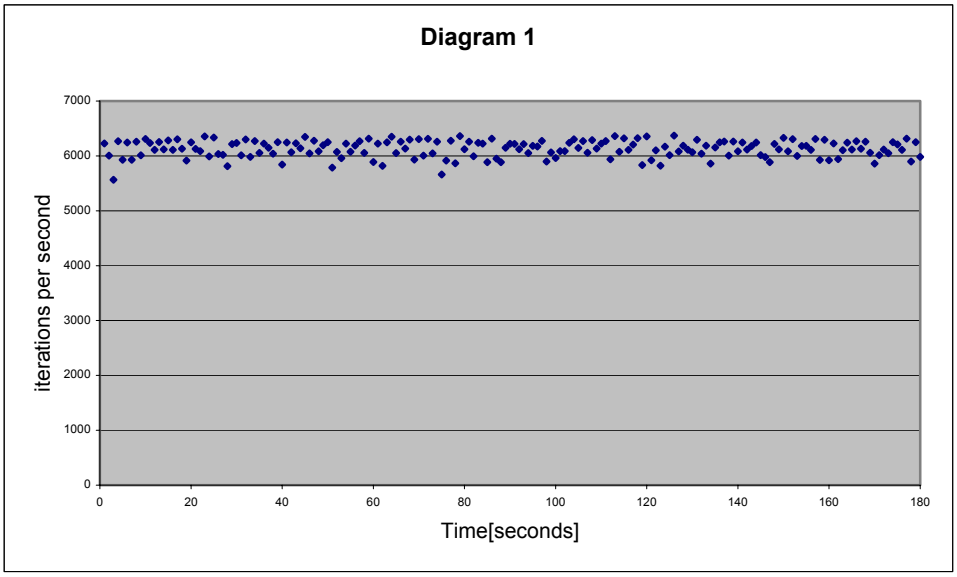
The function `vGetTickCount()` is a library function in mophun that returns a millisecond count.

The entire piece of code above was run for 180 seconds, i.e., it was looped 180 times. The "`count`" variable is there just to make sure that something is done each loop, so that the compiler would not remove it when optimising.

After each loop the value of "`temp`" was written to a file and then reset before it was run again. Briefly the program can be described to count how many of the above loops (what is inside the while-loop) can be executed during a second.

3.1.2 Office test, reference

To be able to get a reference, the program was first run in the office; the results can be viewed in *Diagram 1*.

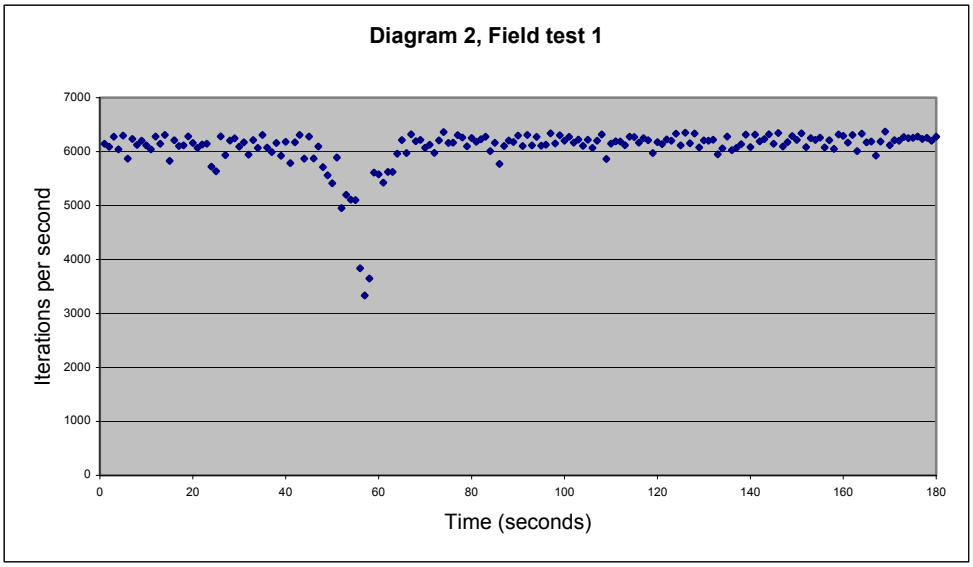


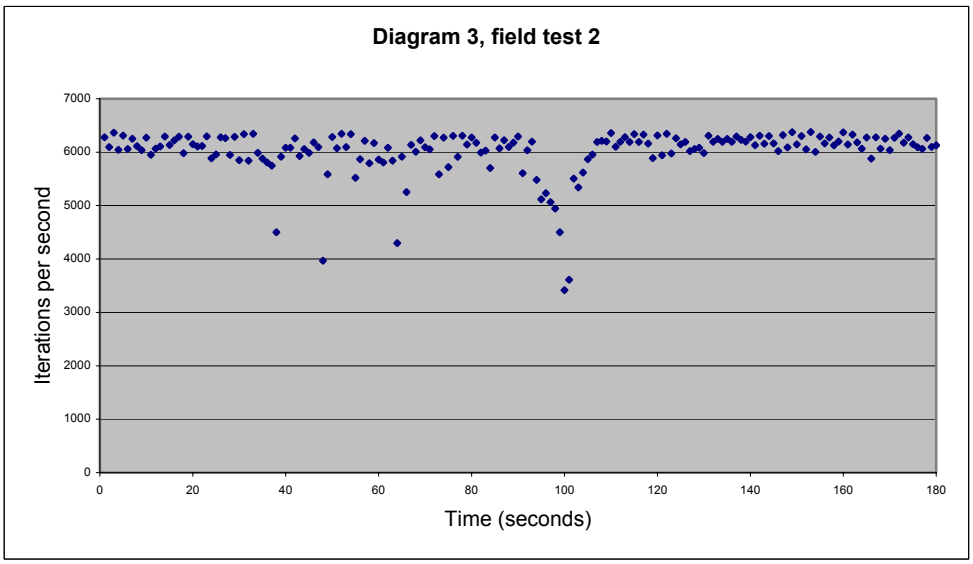
Interesting facts to comment are that the variation is quite large (the standard deviation is about 154) considering the telephone should not have too many demanding tasks in the background. The average is 6129 and the maximum value is about 4 % above this and the minimum is about 10 % below. The entire dataset can be found in the appendix A.1.

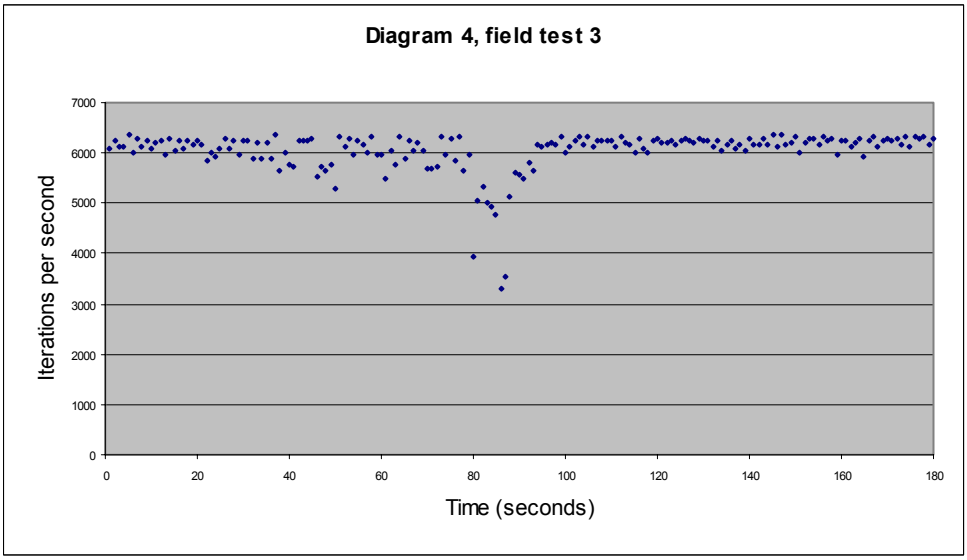
3.1.3 Field benchmarking

To be able to measure in what magnitude performance is lost due to the other processes in the mobile telephone, a means to activate one of these processes was needed. A hypothesis was drawn that when a mobile telephone switches from one base-station to another base-station, some processing power must be needed to handle this handover. Hence a location was found where the mobile telephone was certain to switch base-stations (an underground escalator) and since the results were quite spectacular, the

test was conducted 3 times to verify the findings. The results of the three field-tests can be found in *Diagrams 2 to 4.*







Interestingly a very distinct drop of performance was noted in all 3 cases. The drop in performance was almost as high as 50 % during a few seconds. This was induced when only one other major task of the phone was run. If several tasks of similar performance magnitude were to run in parallel with a game, even worse performance losses can be expected. The datasets for the field tests can be found in the appendix A.2.

3.2 Choice of game

As has been showed above, mophun limits what kind of games can be created. The GPRS connection is also a major limitation and last but not least, a game must be made that fits how users play mobile games! The time scope of this master thesis was also taken into account.

Most multiplayer games, like for instance a car race or No Refuge (Section 5.3), require the players to constantly update each other with, for instance, the current position of the players car. This is usually not a problem when the clients are connected using a low-delay network, like the Internet or a local area network. But when using a connection like GPRS with a high delay, it is very complicated and some technique for overcoming or working around this limitation is required. The problem is illustrated in *Figure 4*.

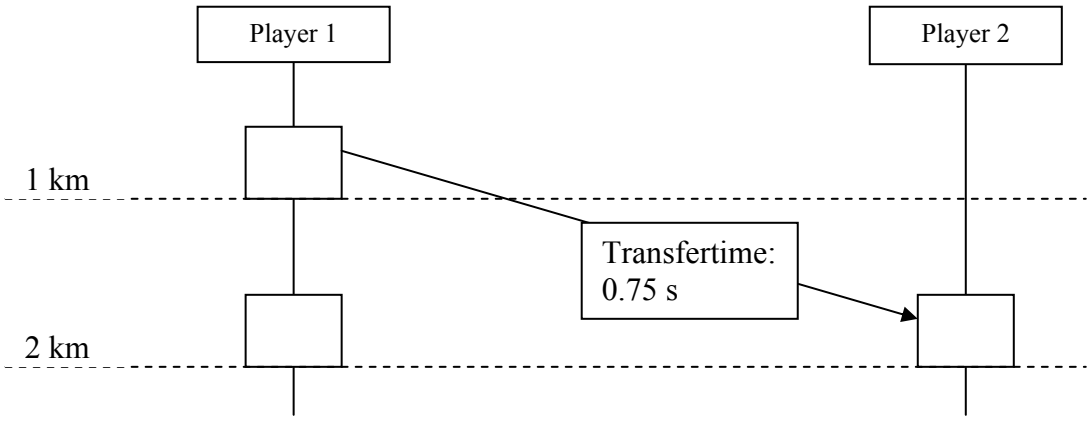


Figure 4. The problems with synchronisation on a high delay network: Consider two cars moving at the same speed and same starting point (unknown to one-another). Player 1 sends his position at 1 km to player 2. When the packet arrives at Player 2, the player 2 car is at 2 km and he then thinks player 1 is at 1 km when it in reality is at 2 km.

This problem can partly be solved with extrapolation. This means in the case of a car game that not only does player 1 send his position, but also his current speed vector and the time when the packet was sent. Player 2 can then if the transfer time is known extrapolate the position of Player 1 and as soon as a new packet arrives, update the position and speed vector. This introduces a new problem since if a car is in a turn its speed vector will change all the time and the extrapolation will be inaccurate. It can be partly solved by using more advanced extrapolations but if collisions between the cars have to be taken into account, it is virtually impossible.

In the game of No Refuge, several players are trying to shoot each other in a landscape using tanks. It is real-time so all action is parallel. Extrapolation is out of the question for a game like this since the position and speed vector of tank changes so often. The solution the developers of No Refuge have used is slow actions sequences. The details are unknown but can be assumed to be that when a player attempts to shoot another player the shot is sent as soon as the button is pressed but the shot is not displayed until a hit has been confirmed or denied by the other client. This makes for slow but accurate game play. Due to all these problems with synchronised games, other types of games were looked into so that asynchronous game play might be created.

The game Tetris (www.tetris.com) was ground breaking for the game genre of puzzle-games. Before Tetris, puzzle games had mostly been based on real-life puzzle-games. Tetris changed this by introducing the stress factor of real-time. Now the puzzle had to be figured out within a strict timeframe that increased all the time! Tetris has since its first appearance on the market 1985 been sold in over 70 million copies (accordning to the tetris-website) and numerous clones (games that are close to Tetris) and other spin-offs have appeared and been very successful. This type of game renders the possibility of asynchronous multiplayer gaming. Only when something like an extra row is to be sent to a player, is a packet sent, otherwise the network is idle (apart from keeping the connection alive). This also reduces network

traffic which is a nice feature for the users who have to pay for each kilobyte of transferred data.

Inquiries were made into the possibility of making a Tetris clone but due to the very aggressive licensing nature of the owner of Tetris, The Tetris Company, a remotely similar game concept was chosen. The base of the game is that the players receive points by getting 3 or more stones of the same colour in a line (horizontal, vertical or diagonal), the points can then be used for purchasing "bombs" of different kinds to throw at your opponent. The more points collected the more powerful the "bomb". The player that looses control first looses the game. Please see F*igure 5*.

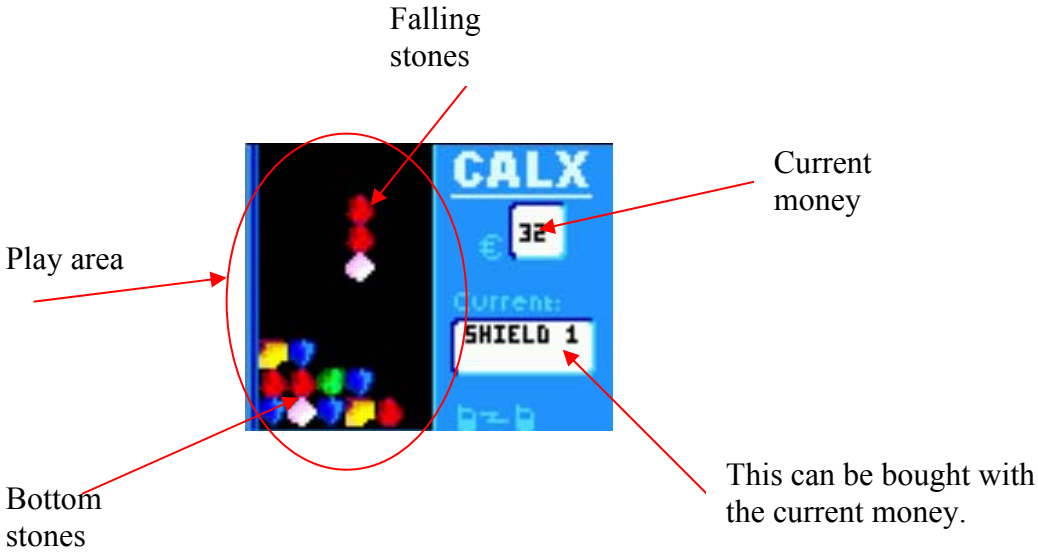


Figure 5. Calx, gameplay description. By moving and rearranging the falling stones, the goal is to get lines of 3 or more stones horizontally, vertically or diagonally.When this goal is achieved the line disappears and gravity makes the other stones fall.

This concept was documented by creating what is called a virtual screenshot, i.e., a picture of how the game will look when it is completed. The virtual screenshot looked like the screenshot in *Figure 5*.

3.3 Literature review

Contemporary software development is mainly divided into two diametrically opposing views, one, the hacker's way, is to create the fastest

and most efficient program with almost unmaintainable code and large stability problems. These programs are most often written in assembly language or C, due to the performance aspect of these languages. The other road, mostly adapted by large software developing companies and academic scholars, is the strict design that emphasises maintainability, portability and stability. A common paradigm in that context is object oriented analysis, design and programming. The most common languages used for this is C++, Java and recently C#. This development pattern usually impedes the performance of the software which is an extremely vital part of game-software. To research the performance problems of development in C++, an appropriate piece of literature was reviewed.

A few books deal with this problem [1]. It does, however, encourage a more pragmatic view of object-oriented design and programming based on the overall goals of the project. It treats performance critical programming techniques in general, not only for small embedded systems but also for heavy loaded web servers and similar applications. As mentioned above, it is heavily characterized by a very pragmatic view on performance critical software development. It describes techniques from all levels of software development, from design, down to, in great detail how the compiler compiles the code and how the linker works. This gives the reader a greater understanding of what performance critical programming consists of. An example of a small detail that the book treats is the use of temporaries.

```
s1 = s2 + s3 + s4;
```

This looks better than:

```
s1 = s2;
s1 += s3;
s1 += s4;
```

but as described by Bulka and Mayhew the former must create two temporaries. The first, containing the result of s3 + s4, and then one for the total result which is to be assigned to s1. The later does not need any

temporaries at all and is actually closer to how the same functionality would be implemented using machine code.

The two most important points are:

1. Only use as much object oriented design and programming as you need to satisfy the project goals. It has no end in it self.
2. 80% of time 20% of the code is run. Use most of the programming and optimization effort on these 20%, this will result in the best performance increase per hour optimisation programming. (Pareto's principle [7])

3.4 Object Oriented Design

Due to the stability advantages of object oriented analysis, design and programming, it was primarily considered as the development design manner. By thoroughly examining which objects exist and their interaction, it is possible to create a UML class diagram and sometimes some other UML-diagrams which describes user interaction or the states of software according to Craig Larman [6]. Which diagrams are produced, depend on the software product developed and what problems are inherent with that product. As an example it is not necessary to create use-case diagrams of great detail for a one-user product since user-interaction does not become very complex.

A brief venture into creating a class diagram for Calx would quickly result in the conclusion the a great many classes would be needed. This would make the overhead simply due to the object oriented design, very large.

3.5 Pragmatic Design

Due to the limitations of object oriented design described above, a more pragmatic approach was considered more rational mainly due to the know-ledge gained from Bulka and Mayhew [1] described in section 3.3. The initial task was to set up which priorities really exist in the project. It soon became obvious that only one developer was going to work with the code,

the entire piece of code was not going to surpass 5000 lines and when it later would be published, there would be no possibility of updating the product with further functionality or fixes. As more and more effort was put into prioritizing the goals of the product it became clear that the by far most important aspect of the product was execution speed during game play and file size of the executable file. The tests conducted on the platform (3.1) clearly showed that mophun™ suffers greatly from not being the highest prioritized process of the mobile telephone. To be able to bring the best kind of game play to the user at all times, huge amounts of free processor time must be available to be able to counter the possible drops of execution slots allocated to mophun. Therefore the design of Calx started with the creation of a flow chart detailing the most common execution path (fast-path) and then adding on all the different kind of exceptions to the fast-path. When the flowchart, *Figure 6*, covered the entire game play part of Calx, the different paths were graded so that the priorities of the different paths could easily be seen. The plan was to create the game from the fast-path and out. By focusing the post-optimisation using this flowchart, the goal was to put most of the optimisation on the 20% of the code that is run 80% of the time.

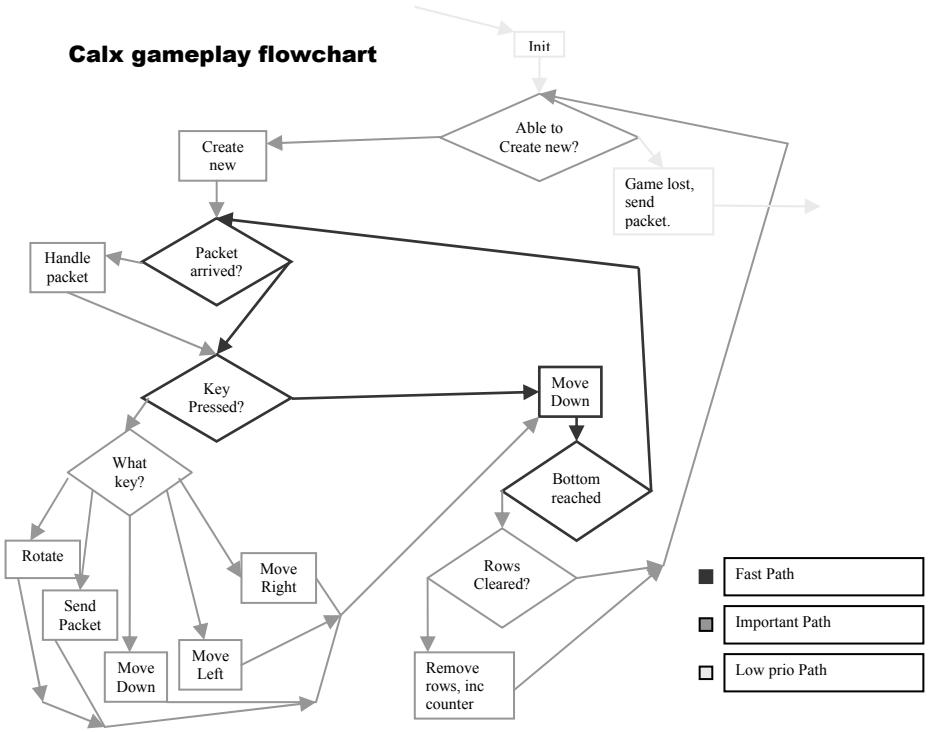


Figure 6. The game play flowchart emphasising the priorities of the different paths.

The design of the other parts of the software, menu etc., were trivial and did not need any design.

By now two modules of the software became obvious, one module to handle the game play and one to handle the network access. These two were made into two separate classes. This way, one of the most usable features of OOP was used: the destructor of an object. The class that handles the network could be made to sign out from the multiplayer-server when the object was destroyed. The main loop was conducted in the main global function which also instantiated the two classes. A class diagram for these two classes is trivial and of no interest, hence it will not be included.

4 Implementation

4.1 Programming

The implementation of Calx was centred on the flow-chart in *Figure 6* and therefore naturally started with the creation of the main loop and creation of the two classes with most of their interfacing (public) functions included with the correct arguments and return values but with an empty definition. This made the program compilable from an early stage which enabled early testing of the different functionalities. In order to familiarise with the mophun API , development of a basic menu system was first on the agenda.

No problems of greater magnitude arose during the development of the menu system and some experience with the API was gained. This lead to it quickly being finished and programming of the game itself began. A single player version of the game was a natural start. Neither during this development were there any greater problems, the few problems that did arise were mainly due to the lack of experience of the mophun API and could quickly be overcome with a helping hand from the programmers at Synergenix.

This was the end of the easy ride, for when the single player version was completed the networking part was started. The greatest problem with the networking was that it was more or less atomic. Either all of it worked or nothing at all. The protocol for communicating with the Terraplay server is quite intricate and requires a lot of detailed knowledge of how it works. After a lot of consulting with the developers at Synergenix and a lot of help from the helpful people at Terraplay, it started working properly!

An interesting error that occurred was that the global Terraplay server used TCP-port 80 (otherwise used for web) for connecting players (lobby) and port 110 (otherwise used for mail) for the game play (gateway). When running the mophun emulator, there was no problem in finding each other

on the lobbyserver (using port 80) but for some reason it was impossible to connect to the gateway. After several days of analysing the code for errors and checking what was actually sent on the network card, Martin Stenhoff of Terraplay finally figured out the error. I was using an anti-virus program that checked all incoming mail. In order to do this it acted as a tunnel for all mail traffic, i.e., all traffic on port 110. Since the data being sent on that port when attempting to connect to the gateway server was not in any way according to the POP3 mail protocol which is usually sent on that port, the anti-virus program freaked out and blocked it all. Hence it was impossible to connect to the gateway server when the anti-virus program was running. It was quickly solved by disabling the mail protection of the anti-virus program.

4.2 Evaluation

When the main part of the programming was completed, the game was transferred to a mobile telephone (a T300) and tested against another. The game worked according to plan. The delay in packet transmission was just as it was supposed to be but was not a big problem in the game play and could almost be seen as a feature of the game. There were still a few small bugs but these were ignored since only an alpha-version of the game had been specified during the specification of the Master's project. However, the game seemed to be a bit slower than it should be, hence there was need for some post-optimisation. This was mainly noted by the fact that the game did not get the desired 13 frames-a-second frame rate.

4.3 Post-optimisation

Post-optimisation of the game started by going through the functions included in the fast-path and it became obvious that there were some optimisations to be done in the redrawing of the screen. To be able to describe what was done, the actions of the fast-path will first be described.

In the following section the program is assumed to be in the fast-path state, i.e., only the most common parts of the main loop are treated. Firstly there

was a check to see if any packets had arrived on the network from the other player. In the fast-path state, nothing arrived. The second part was the reading of the keyboard to see if anything in particular had been pressed. In the fast-path state, nothing had being pressed. (Perhaps, depending on how the game is played, the down-button, to increase the speed might be the most common path, however it does not differ a lot from the no-button-pressed version of the fast-path.) After the keyboard check, a function was run in order to move the falling stones down one pixel. A check to see if the bottom had been reached was executed and the fast-path state concluded that it had not reached the bottom. Now the screen was updated. This last part of the updating of the screen was run every cycle of the main loop even when not in the fast-path state.

In the first version of Calx, when the screen was redrawn, the entire play area was redrawn (see *Figure 5*), this is however unnecessary since the bottom stones are not changed. The redrawing of the bottom stones consisted of two nested for-loops which in total redrew 46 stones (no stone in a part of the play area meant drawing a black box in order to make sure it was cleared properly). The program was changed to only redraw the bottom stones when the falling stones hit the bottom, i.e., it was moved out of the fast-path.

The game was tested again and it showed a considerable improvement in speed! In order to check if further optimisation was necessary, a timer was added so that each the time for each part of the main loop could be clocked. The time for the entire main loop in fast-state was about 70 ms and the largest amount of time (more than 90%) was taken by the flip-screen function. The other 10% was taken up by the other functionality of the main loop. The flip-screen function has to wait for the copying from the last screen to be finished before it can start. The program will also halt until the flip-screen function can start the copying. The result of this is that even if the time of the non-flip-screen functionality of the main loop was to be doubled or tripled, a decrease in frame rate would not be probable since the system has to wait for the last screen to be finished. Hence any further

optimisation was considered fruitless and the job of developing an alpha version of the game complete.

5 Results and the future

5.1 Results

This project has been an utter success. All development has been completely according to plans and no major holdbacks have been encountered. The mophun virtual machine has lived up to all expectations and so has the Terraplay multiplayer server even though it has not by far been pushed to its limits. According to the people who have tried the game, they find it entertaining and do not see the lag caused by the GPRS connection as a problem at all. The game is fast paced and induces a certain amount of stress which is just as the gameplay should be! This kind of work-around using a game with asynchronous game play is probably one of the best work-arounds for the huge delay. The solution used in No Refuge of slow action sequences could have been used but would have resulted in a slower game with impeded game play, i.e., less entertaining.

The fast-path oriented pragmatic design proved to be very successful both in focusing the program on the important parts and in enabling time effective optimisation. Due to the time that would be needed to rewrite the entire programming with a true object oriented design, it was not done and hence a comparison between the techniques can only be a qualified guess. It might not have been a problem due to the fact that after the post-optimisation there was quite a lot of time per frame left, but on the other hand, the time available was quite limited, according to the tests in 3.1. However, there is no easy and factual way of finding out without actually writing the program.

5.2 The future

Calx will be completed and made into a commercial product and it might already be for sale when this thesis is published. It can probably be found on the internet site www.mophungames.com where most games for the mophun platform can be found.

The virtual machine mophun is still very young and has already proved to be technically superior to any similar technique. New versions of it have already been released with a 3D-API and with an even faster virtual machine. The current VM, on the T300, is mophun 1.0 and the 2.0 is said by people at Synergenix to be at least double the speed mostly due to complex features of the VM such as dynamic recompilation. It has been released for the Symbian operating system in June 2003 which can be found on a number of mobile devices such as the Sony Ericsson P800, Nokia N-Gage. The hardware performances of these platforms are way ahead of the T300 and the games that can be developed for them are of a different dimension. Hopefully the mophun VM with its great API and unbeatable speed will become the market standard in a few years.

5.3 Results from a similar project

To put the above results into perspective, an interview with a developer of a similar project was held. This chapter is based on this e-mail interview with Erik Stackenland, a developer of the game "No Refuge". This game is a multiplayer real-time game that has been developed on the mophun platform using GPRS to connect to a Terraplay server. In the game, up to 8 players can fight each other with tanks on a large area.

The original interview is in Swedish and a translation in English can be found in the appendix A.3. For further information on No Refuge or the company behind it, please refer to www.mobileinteraction.com.

In coherence with the analysis of mophun and J2ME above, Stackenland agreed that J2ME in its current form was not very well suited for games development and spoke strongly in favour of the mophun virtual machine. An interesting fact was also pointed out by Stackenland: lacking support for networking in J2ME had been their main reason for moving to mophun. This is noteworthy since Synergenix does not use this fact as a main selling argument for mophun but rather emphases its superior performance which was also noted by Stackenland. Stackenland continues by noting that the

development of software for mophun requires more from the programmers due to the fact that not everything is served on a silver platter.

On the question on how they handle the high delay in network traffic when using GPRS, Stackenland referred to the use of slow action sequences. A simplified example of this is if tanks A and B are playing and tank A wants to shoot at tank B. The player of tank A will press the fire-button and as soon as this is done a message is sent to player B to update B's current position. When A gets this position, the phone can calculate if the shot was a hit or miss. With a RTT of 1.5 seconds, the time from when a button is pressed to when the shot goes off will be about 1.5 seconds. This is probably the best way to tackle the problem with a game that requires continuous synchronisation but it has the drawback of greatly impairing game-play by making it very slow.

On the matter of object oriented programming/analysis/design Stackenland mentions that the overhead of object handling is notable and when all tricks have been used to optimise performance, the disadvantages of object oriented development outweigh the advantages. This is in accordance with the pragmatic view of software development by Bulka and Mayhew [1].

Stackenland also emphasises an empirical and pragmatic view of optimisation; use all the tricks you know and conduct empirical testing of them to make sure that they really do make a difference.

6 References

1.

Efficient C++, Performance Programming Techniques

Dov Bulka & David Mayhew

Copyright © 2000 by Addison Wesley Longman, Inc.

ISBN 0-201-37950-3

2.

The homepage of Prof. Niklaus Wirth, author of Pascal.

http://www.inf.ethz.ch/~wirth/ (last visited 15:th of October, 2003)

3.

The Clash of Mobile Platforms: J2ME, ExEn, Mophun and WGE

Pedro Henrique Simões Amaro

Departamento de Engenharia Informática

Universidade de Coimbra

3030 Coimbra, Portugal

pamaro@student.dei.uc.pt

http://pedroamaro.pt.vu (last visited 15:th of October, 2003)

4.

Wireless Game Engine, a Powerpoint presentation by

Brian Møller and Gaël Rosset

http://www.3gpp.org/ftp/tsg_t/WG2_Capability/TSGT2_16_SophiaAntipolis/Docs/T2-020084%20(WGE%20General).pdf

(last visited 15:th of October, 2003)

5.

Wireless Graphics Engine homepage at TTPCom

http://www.ttpcom.com/ttpcom/wge/index.html

(last visited 15:th of October, 2003)

6.
Applying UML and Patterns
Craig Larman
Copyright ©2002 by Craig Larman
ISBN 0-13-092569-1

7.
Pareto principle
http://www.paretolaw.co.uk/principle.html
(last visited 15:th of October, 2003)

Appendix

A.1 Office/Reference benchmarking dataset

6226,6005,5566,6269,5931,6244,5930,6259,6012,6309,6235,6109,6253,61
18,6282,6110,6302,6130,5915,6248,6128,6088,6356,5990,6334,6036,6019,
5812,6214,6236,6012,6300,5981,6272,6056,6223,6149,6041,6249,5843,62
45,6067,6230,6140,6347,6047,6278,6085,6196,6249,5788,6073,5960,6225,
6076,6189,6269,6054,6316,5888,6224,5818,6248,6351,6053,6258,6135,62
96,5932,6305,6005,6313,6047,6258,5660,5914,6278,5864,6362,6122,6260,
5995,6240,6224,5886,6314,5950,5887,6149,6221,6219,6123,6215,6053,61
82,6171,6273,5897,6064,5961,6086,6086,6238,6304,6145,6270,6057,6287,
6133,6224,6271,5939,6361,6075,6321,6110,6208,6322,5833,6355,5921,61
03,5823,6169,6013,6370,6081,6187,6113,6068,6294,6039,6186,5859,6153,
6248,6261,6006,6263,6085,6244,6117,6181,6246,6013,5979,5887,6217,61
20,6328,6084,6306,6001,6179,6184,6109,6309,5928,6293,5921,6227,5942,
6105,6241,6117,6269,6132,6260,6058,5860,6015,6117,6050,6250,6209,61
09,6315,5897,6249,5982

A.2 Field test benchmarking datasets

A.2.1 Field test 1

6145,6088,6275,6041,6296,5869,6236,6126,6202,6116,6041,6280,6147,63
08,5829,6212,6104,6115,6281,6161,6068,6131,6147,5721,5635,6281,5934,
6206,6248,6092,6173,5944,6215,6068,6310,6074,5993,6160,5924,6181,57
86,6172,6310,5870,6278,5872,6094,5712,5562,5412,5890,4952,5200,5110,
5103,3837,3334,3649,5613,5579,5425,5623,5625,5962,6214,5975,6321,61
93,6217,6071,6128,5978,6209,6362,6162,6166,6306,6261,6098,6252,6180,
6230,6275,6009,6165,5771,6103,6205,6179,6296,6103,6309,6115,6273,61
09,6128,6338,6151,6302,6202,6273,6166,6223,6107,6221,6070,6202,6319,
5863,6148,6191,6187,6119,6275,6271,6167,6249,6211,5974,6176,6141,62
27,6203,6333,6117,6350,6156,6337,6078,6209,6203,6219,5949,6060,6278,
6027,6070,6138,6315,6083,6314,6189,6232,6321,6145,6345,6096,6175,62
88,6218,6340,6082,6251,6216,6257,6078,6212,6050,6319,6290,6166,6310,
6009,6333,6172,6186,5925,6188,6372,6120,6211,6202,6269,6250,6255,62
78,6234,6255,6201,6276

A.2.2 Field test 2

6279,6094,6365,6044,6310,6058,6251,6113,6039,6271,5951,6066,6108,62
94,6133,6221,6290,5980,6291,6148,6106,6115,6296,5886,5958,6279,6262,
5948,6286,5848,6338,5840,6345,5987,5878,5807,5747,4501,5915,6082,60
83,6258,5928,6060,5987,6182,6095,3967,5587,6285,6075,6343,6095,6339,
5521,5868,6213,5793,6171,5861,5807,6083,5839,4296,5915,5254,6137,60
07,6222,6092,6052,6303,5586,6273,5719,6304,5912,6307,6143,6275,6175,
5993,6030,5702,6275,6071,6224,6097,6178,6292,5607,6034,6198,5480,51
18,5232,5063,4944,4502,3416,3610,5506,5339,5620,5866,5953,6191,6212,
6201,6360,6102,6193,6281,6190,6339,6191,6330,6160,5887,6315,5943,63
43,5976,6259,6147,6192,6020,6059,6082,5983,6309,6193,6248,6193,6249,
6195,6292,6231,6197,6282,6131,6307,6157,6301,6164,6021,6319,6088,63
75,6147,6302,6054,6376,6006,6294,6163,6276,6128,6200,6372,6144,6330,
6182,6064,6276,5879,6279,6065,6250,6037,6269,6347,6175,6276,6148,60
93,6063,6265,6099,6128

A.2.3 Field test 3

6094,6258,6140,6137,6345,5997,6271,6135,6251,6103,6199,6251,5971,62
76,6062,6256,6095,6236,6155,6234,6184,5841,6024,5920,6070,6279,6094,
6234,5952,6232,6241,5868,6216,5903,6221,5869,6346,5667,5989,5784,57
13,6243,6242,6247,6271,5522,5746,5629,5772,5274,6309,6115,6270,5948,
6228,6181,6009,6315,5966,5984,5479,6045,5776,6313,5872,6235,6064,62
10,6050,5704,5684,5721,6314,5970,6274,5859,6331,5649,5961,3949,5038,
5311,5005,4923,4784,3319,3555,5141,5627,5551,5493,5788,5632,6149,61
29,6168,6198,6174,6322,6017,6106,6256,6318,6162,6321,6122,6231,6249,
6239,6256,6144,6304,6202,6149,5998,6295,6072,5998,6225,6286,6188,62
20,6249,6172,6241,6289,6228,6220,6277,6226,6234,6114,6240,6059,6156,
6240,6090,6167,6036,6278,6182,6181,6276,6152,6378,6119,6354,6183,62
23,6313,6021,6220,6280,6290,6155,6337,6257,6289,5984,6232,6241,6126,
6216,6271,5915,6225,6312,6107,6254,6293,6240,6277,6146,6304,6116,63
38,6267,6336,6182,6270

A.3 Interview with Erik Stackenland

This appendix contains the interview with Erik Stackenland of Mobileinteraction. The first part, 8.3.1 is the original interview in Swedish, and the second 8.3.2 is the interview translated in English.

A.3.1 Original interview in Swedish

Fråga: Har ni utvecklat för någon annan mobil VM och om så, hur ser ni på mophun i ljuset av den erfarenheten?

Svar: Vi har utvecklat med Java J2ME. Mophun är betyldigt snabbare och bättre anpassat för just spel. I våra ögon är inte J2ME lämpligt för spelutveckling, både med avseende på hastighet och möjligheter. Det är enkelt och smidigt att arbeta med mophun.

F: När ni började utveckla ”No Refuge”, visst ni att ni skulle göra det för mophun?

S: Nej. No Refuge började som ett WAP-spel. Uppföljaren till denna planerades att göras i J2ME. Framförallt beroende bristande stöd för nätverkstrafik gick vi över till mophun.

F: Om nej, vilken var den största utmaningen med mophun?

S: Resurshantering och att lära sig all tillvägagångssätt som krävs i mophun. I Java och J2ME är mycket serverat på silverfat.

F: Hade ni undersökt möjligheterna/prestandan innan och anpassade spelet efter det eller kodade ni allt och efteroptimerade tills det funkade?

S: Vi undersökte prestandan innan vi började och anpassade spelet för förutsättningarna, men givetvis så har vi optimerat så mycket som möjligt för att nå bästa möjliga resultat.

F: Med facit i hand, hade ni gjort annorlunda om ni skulle ha gjort om det?

S: Som med allt nytt man håller på med, vet man bäst hur man ska göra när man redan är färdig. Det finns saker som skulle göras på annorlunda sätt om vi skulle göra om det.

F: Finns det något speciellt som är av intresse för andra som ni skulle ha gjort annorlunda?

S: Nej, det är inget av speciellt intresse som vi skulle göra om nu i efterhand. De saker som varit kritiska har vi ändrat på under utvecklingen gång.

F: När ni började utveckla ”No Refuge”, visste ni att ni skulle göra det med Terraplays Multiplayerteknik?

S: Ja. Det var bestämt redan vi började.

F: Hade ni mätt RTT (Round trip time, ping) för GPRS innan?

S: Nej, men vi var väl medvetna om att delayen var hög och fluktuerande.

F: Kan ni avslöja några tekniker ni använder för att gå runt problemet med den extrema RTT:n?

S: Långsamma händelseförlopp.

F: Om ni vetat allt om mophun, Terraplay och GPRS begränsningar, när ni skulle bestämma er för vilket spel ni skulle göra, hade ni fortfarande valt att göra ett spel som ”No Refuge”?

S: Absolut

F: Om ni fått möjligheten att ändra en sak i mophun, en sak i Terraplay och en sak i hur GPRS fungerar, vad hade ni ändrat? (t.ex. Vi skulle ha velat att mophun's 3D API var klart, vi skulle vilja att man kunde vara 20000 användare i samma session på Terraplays server, vi skulle vilja att GPRS hade en datakapacitet på 100 kbyte/s)

S: Mophun: bättre komprimering av binärfilerna.
Terraplay: Om något, mindre protokoll (färre meddelanden). (Dock kan det väl vara så att det är omöjligt att genomföra och ändå uppnå den generalitet de erbjuder).
Expire dates för tjänst borde kontrolleras på serversidan i stället för på klientsidan.
GPRS: lägre delay. Större OTA [Over The Air] download size (kanske inte direkt med GPRS att göra)

F: Objektorienterad programmering, analys och design anses av många vara det moderna sättet att utveckla program på, vad anser ni och hur mycket av dess metodik/metoder har ni använt under utvecklandet av No Refuge?

S: Vi håller med om att objektorienterad utveckling är att föredra generellt. Tyvärr så innebär det också overhead jämfört med funktionell programmering, både i avseende på binärstorlek och prestanda (tror att det ligger på ca 10%). Vi har av de skälen valt att hålla oss till det senare (C istället för C++).

F: Vilken enskild metod när det gäller optimering eller skapandet av snabba program anser ni vara bäst/viktigast?(t.ex. manuellt konstruerade och programspecifika minneshanterare, return value optimisation) Om den inte är trivial får ni gärna beskriva den lite kort!

S: Omöjligt att svara på, vi har använt oss av alla knep vi kommit på. En sak som vi blev förvånade över är hur mycket mer utrymme än switch-sats tar än en if-else if-else if...

A.3.2 Translation of the interview in English

Q: Have you developed software for any other mobile virtual machine [than mophun] and if so, how do you view mophun in the light of this knowledge?

A: We have previously developed software using Java J2ME. The mophun VM is considerably faster and better suited for developing games. In our view, J2ME is not appropriate for games development, due to the limitations of speed and capabilities. It is easy and straight-forward to work with mophun.

Q: When you started developing "No Refuge", did you know you would use mophun?

A: No. No Refuge started out as a WAP-game. The sequel was planned for J2ME. Mainly due to the lack of support for network traffic we moved to mophun.

Q: If no, which was the greatest challenge with mophun?

A: Resource management and learning the workflows of mophun. In Java and J2ME, lots of things are served on a silver platter.

Q: Did you examine the capabilities and performance of mophun before development and adapted the game thereafter or did you just get to work and then post-optimized?

A: We examined the performance before we started development and adapted the game according to the limitations. Naturally we have also did considerable post-optimised to get the best possible result.

Q: Now that the game is finished, are there things you would have done differently?

A: As with everything new, you always know best when you are already done. There are things that we would have done differently, had we started all over.

Q: Is there anything in particular that might be of interest for other developers, that you did in a different manner?

A: No, there is nothing of particular interest that we would do now after the game has been finished. The critical matters that we found were changed during the development.

Q: When you started the development of “No Refuge”, did you know that you would use the multiplayer technology supplied by Terraplay?

A: Yes. It was decided from the beginning.

Q: Had you measured the RTT (Round Trip Time or ping-time) of GPRS before development?

A: No, but we were well aware of the fact that the delay was large and fluctuating.

Q: Would it be possible for you to reveal any techniques you used to work around this problem with the extreme RTT?

A: Slow action sequences.

Q: If you had known beforehand what you now know of the limitations of mophun, Terraplay and GPRS, would you still have chosen to develop a game like “No Refuge”?

A: Absolutely.

Q: If you had the possibility to change one thing in mophun, one thing in the Terraplay system and one thing in how GPRS works, what would you like to change? (for instance: We would have liked the mophun 3D API to be ready, we would have liked the Terraplay servers to be able to manage 20 000 concurrent users, we would like GPRS to have a throughput capacity of 100 kbyte/s)

A: mophun: better compression of binary files.
Terraplay: If anything, smaller protocol (fewer messages). (It might very well be impossible to do and still deliver the level of generalization it now offers). Expire dates of a service should be controlled server-side instead of on the client-side, as it is now.
GPRS: lower delay. Larger OTA download size (might not directly be connected with GPRS)

Q: Object oriented programming, analysis and design is considered by many professionals to be the best manner in which to develop software. What are your opinions in this matter and how much of this methodology did you apply during the development of No Refuge?

A: We agree that object oriented programming is to be generally preferred. Sadly it has the drawback of a larger overhead in comparison to functional programming, both in the matter of binary file size and performance (we think it is around 10%). Due to these reasons we have chosen to stick to later alternative (C instead of C++).

Q: Which single method concerning optimization or creation of high performance software do you consider the best/most important? (for instance manual construction of specific memory handlers, return value optimisation) If it is not trivial, please describe it briefly.

A: Impossible to answer, we have used all the tricks we can think of. One thing that we were astounded by was how much more space a switch-statement takes in comparison to a if-else if- else if…