NTNU
Det skapende universitet

Multiplayer On One Screen Entertainment System

evaluation and improvements

ALEKSANDER BAUMANN SPRO
MORTEN VERSVIK

Master of Science in Computer Science
Master Thesis *Submission date:* July 2007
Supervisor: ALF INGE WANG, IDI
Co-supervisor: GEIR MELBY, Tellu AS
Co-supervisor: KNUT EILIF HUSA, Tellu AS

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

Abstract

This thesis looks at new game concepts for the MOOSES framework in regards to social gaming, as well as some performance improvements that are needed to support a larger number of players. The MOOSES framework is a framework for easy development of multiplayer games which lets many players play with/against each other on a big joint screen, for example a cinema canvas. As this concept allows interaction between many players at one location, another factor to be exploited is socialization between the players.

Games that feature social interaction are very popular, and has the effect of making players stay longer with the game. We look at why this happens and how we can integrate these features in the resulting games for MOOSES. These features can contribute to a success when commercializing all games (not limited to MOOSES).

This thesis also features the results of some of the possible optimizations outlined in our depthstudy and some discovered while writing this thesis, wjile also answers some critical open questions from our depthstudy. Performance is critical when the platform and games must support a large number of players, and an expected number of maximum players is important for business considerations and game development. To do this we look deeper into some of the underlying technologies that we are using, and use the gained knowledge to reduce latency and increase the perceived response time.

Problem Description

This research in this project is divided into two main parts. One goal is to look at how gaming can contribute to the social aspect of the MOOSES concept and its games. This research will be used in the design and implement of new games for the framework. The other is to evaluate and improve the framework and games in their current state.

Assignment given: 2007-02-11
Supervisor: Alf Inge Wang, IDI

Preface

This master thesis is written by Aleksander Spro and Morten Versvik in the period from mid February 2007 to mid July 2007.

Acknowledgments

First we would like to thank Alf Inge Wang for this assignment and for his, Knut Eilif Husa's and Geir Melby's help and guidance throughout this project. We also appreciate all the publicity we have gotten, which have resulted in us meeting random people that have heard about the concept and provided us with both positive and negative feedback.

Tellu has provided the framework we could base these theses upon, thanks for making it and letting us use it. Thanks also for believing in our concept enough to try to commercialize it.

The MOOSES crew will also like to thank everyone who showed up to test our product at the testing sessions, and all the journalists and critics that gave us feedback. We will also like to thank Kino 1 Sandvika which has said they will to be our first customer before the end of the year.

Trondheim, 9. July 2007

Aleksander Spro

Morten Versvik

Contents

List of Figures

List of Tables

Listings

Part I

Introduction

CHAPTER 1

Problem description

Electronic entertainment in the form of tv- and computer games has seen a steady increase in popularity over the last decade. With computing- and network technology continuously breaking new ground- fueled by the massive increase in Internet usage, we see exceedingly more realistic games hitting the market along with new multiplaying features opening for new game concepts. With the development of these new concepts, new and/or otherwise preliminary unused technology can be brought into the light and often open up new marketing potentials.

The advancements in technology has brought the cinemas into the digital age, with projectors able to deliver stunning visuals that surpass the old 35mm film rolls. With the move to digital projectors, the cinemas open up a whole new range of uses previously not thought of as their market. In addition, cinemas connected to the Internet and with each other through fiber-optic nets allow for almost instant communication with each other as well as opportunities within live streaming from any source connected to the Internet.

With the recent advancements in both of these categories, the opening for a merge between the two arises. This scenario become particularly interesting with cinemas looking to expand their business to other entertainment than movies. Cinemas have a lot of dead time that can be utilized in other ways.

This prospect lead us to, in our depthstudy[50], develop a framework and game prototype for entertainment purposes in this type of setting. This project will evaluate this framework against its domain and requirements, present improvements to the framework and look at where the framework development should be focused next.

1.1 MOOSES Framework

MOOSES stands for Multiplayer On One Screen Entertainment System, and is a framework designed for one concept: to let many players play with or against each other on one big joint

screen. One type of such a joint screen can for instance be a cinema canvas. The framework is designed to ease the development of games in this setting, so the framework is designed to take care of tasks like score keeping, billing, player authentication and controller connectivity. The framework has support for many types of controllers, but currently only a mobile phone controller is implemented.

The MOOSES framework was concieved, designed and implemented at NTNU the Fall of 2006. The current version has been successfully tested at Nova Cinema in Trondheim.

A more detailed description of the framework architecture can be found in Section 9.3.2.

1.2 Motivation

In our depthstudy[50] we looked into what game genres would fit this domain. Most of our focus then was on the technical challenges facing this type of concept. One aspect we barely touched was the effect of socialization in gaming. With the gathering of so many players in the same location, this is certainly one of the concept's greatest strengths and needs looking into.

We also designed and implemented a framework in our depthstudy. However, due to time limitations we were never able to thoroughly test and evaluate the framework against its requirements and users. As such, there is an uncertainty conserning whether the framework will meet up to its requirements and expectations.

1.3 Problem definition

Our main tasks will be to look at the effects of social gaming, both how the game affects the player and the player affects the game. The result of this research will be used in new game prototypes for the framework so we can test our findings.

Our second task will be to test and evaluate the MOOSES framework and perform any improvements wherever needed. We will also look at future possible improvements for the framework.

This project will be done in cooperation with Sverre Morka's master thesis, which is looking into moving the client side of games to a script based implementation which is loaded at real time, this to make MOOSES flexible for new games at the client side. Audun Kvasbø is looking into more game concepts suited for the framework.

CHAPTER 2

Project Context

This project is a part of the research program on videogames at The Norwegian University of Science and Technology (NTNU) [57], and based on our depthstudy [50]. This research program has the following superior goals:

1. To be initiator for new research projects regarding videogames at NTNU
2. Work to identify potential external resources that may support NTNU's activities regarding videogames.
3. Coordinate and arrange for research regarding video games internally at NTNU and between NTNU and external collaborators.
4. Work to promote and make visible the activities regarding videogames at NTNU.
5. Serve as advisors with respect to drawing up study offers related to videogames at NTNU.

The project runs in parallel with two other projects based on the MOOSES platform.

In Sverre Morka's master thesis, he will be looking at tailoring a scripting solution for use in the MOOSES mobilephone client's software. A scripted solution is required because of limitations in the phone's software. Some cooperation will happen as both sides find new features that should be added or modified. The new scripted client will define elements that can be reused in different types of games, and try to optimize it.

Audun Kvasbø works on new game concepts for MOOSES.

Tellu [54], a spin-off from Ericsson's Norwegian research center, has helped with support for the framework utilized by MOOSES. Knut Eilif Husa and Geir Melby, (both with Tellu), have also helped as co-supervisors along with Alf Inge Wang on this report.

CHAPTER 3

Reader's Guide

The contents of this report vary a lot, both with respect to focus and approach. Some parts of the report might be more interesting to some readers than others. In order to increase the readability of the report, we will therefore present a brief outline of each chapter.

If you are not interested in reading the whole report, you should identify yourself with one of the four following categories:

Readers interested in research into game concepts for the domain, should read Chapters 1 and 4. Part III presents technology used by MOOSES and a study into the social gaming domain.

Readers interested in design of a framework for the domain, should read Chapters 1, 4 and 6 as well as Part III (to get an overview of the architectural design and hardware used). Part V covers improvements done to the framework during this project and Chapter 20 evaluates the current implementation of the framework.

Developers interested in improving the framework, should read and understand Chapters 8, 9, 20 as well as Part V. Developers might also want to read Part VII.

Developers interested in designing games for the framework, should read and understand Chapters 11, 12 as well as Part IV. Chapters 21 and 22 which evaluates current games and the social aspect of MOOSES might also be of interest.

Part II

Research Methods

CHAPTER 4

Research Questions & Methods

In this chapter we will describe which questions we seek to answer through our work, and how we intend to find the answers. We will describe how we are planning to conduct the work and explain the different methodologies we are planning to use.

4.1 Research Questions

In our depthstudy [50], we provided research into determining what type of games will work on a single large screen like a projection-screen and presented a framework as a solution to work in this domain. We now want to get some research into utilizing the fact that all the players are gathered at the same location. We will also test the framework up against some of its requirements, this was lacking in our depthstudy [50].

While the huge amount of players on one screen opened up new perspectives on video-game development, so does the aspect of having all the players gathered at one location. This is truly one of the strengths of this concept and one we need to take advantage of. While still being under the constraints from our prestudy in regards to video-game design, we now need to look at social and multiplaying aspects with video-games.

A framework and a game prototype were designed and implemented in our depthstudy, where requirements covered the technical difficulties we anticipated. However, we did not have time to thoroughly test the framework against its requirements, and as such there could be some unforeseen issues we have yet not discovered. Neither was the hardware stress-tested to see how it coped with a great number of players.

In retrospect of this as well as from motivation and the problem definition, several concrete questions have arisen. This report will lead to answering these questions.

RQ-I: **What comprises social video-gaming today and how does it fit into MOOSES?**

➢ We will look at how socialization is utilized in todays video-games and categorize them both on a player- and a conceptual level. This research will then be discussed to determine how these can be applied to our domain and game concepts in an attempt to take advantage of having all our players gathered at the same location.

a) What different social video-gaming categories are there?

b) What different player types comprise social video-gaming?

RQ-II: **What type of game mechanics are suitable for use with cooperative multiplayer mode for MOOSES?**

➢ We will compile a research into todays multiplayer games to get an overview over their mechanics and how they are played. This research will then be discussed to determine what features work well within our domain. We also want to see how players utilize other hardware/software in order to enhance their gaming and how that can be implemented into our concept. Finally, we will present some new game concepts for our framework with focus on cooperative multiplaying choosing one for a developed prototype.

a) What existing multiplayer attributes work best for MOOSES?

b) Will this scenario work with our targeted number of players?

c) Can we utilize more then just game mechanics to enhance the gameplay in MOOSES?

RQ-III: **Does the MOOSES framework and the MOOSES game prototypes fulfill their roles in regards to their requirements and scenario?**

➢ We will test the framework with focus on the functional requirements from our depthstudy. We will do this through a stress test with our prototypes and test subjects. The test subjects will also be provided with a survey to give input on some of the requirements of the framework as well as how well the game prototypes and concept work.

a) Does the framework meet its functional and non-functional requirements?

b) Does the prototypes work well with MOOSES?

c) Does the framework need enhancement to work with cooperative gaming?

4.2 Research Method

In order to make prescriptions about the research methodology in software engineering we need a basic understanding of what software engineering is.

Basili in [47] defines software engineering as follows:

> "...can be defined as the disciplined development and evolution of software systems based upon a set of principles, technologies and processes."

With that in mind we can start to look at what models to use for research into software engineering. One issue with attaining good models for this type of research is that software engineering is still fairly new discipline in a scientific perspective. Unlike other sciences, models for components like processes and resources have been neglected so far even though a lot of research is going on in this field. Basili [47] does however describe three experimental models for use in software engineering research. The variations between the models are small, but focus on different areas and are parts of two distinct paradigms; the scientific- and the analytical paradigm. First model consists taking on an engineering approach, the second an empirical approach while the latter takes on a mathematical approach. A shost description of the three methods can be found in Appendix C.

The two approaches of the scientific paradigm, engineering and empirical, will constitute the base research of this project. The engineering approach will be utilized in developement of new game prototypes as well as improving the MOOSES framework. Both, framework and game prototypes, will be evaluated in Part VI throught the use of empirical data. The empirical research will also form the basis for the look into social gaming and multiplayer games categorization.

4.2.1 The Engineering Approach

For the development of new game concepts and improving the MOOSES framework we will be utilizing the engineering approach. Our prestudy will be used for analyzing current technologies and knowledge on the problem domain in order of finding good solutions to our research questions.

Experiences from our depthstudy [50] will form the basis for improving the MOOSES framework. This will then undergo an iterative process to see if we solved the issues at hand. Any other problems that might surface during the testing face of this process will be documented.

Prestudy

In our Prestudy we will look into and describe the central hardware and software configurations that concerns our domain. We will also make a summary of and outline the essentials of the MOOSES framework. Last we will look into and review gaming technology based on socialization and cooperation. This will give us an overview of what we have to work with and what challenges we have to overcome in development of new prototypes and improvements to MOOSES.

Design & Implementation

Due to the limited nature of framework design and implemention in this project we opted to go for a simple development approach. The evolutionary prototyping method consists of stages that can be repeated an unlimited number of times, see Figure 4.1. Due to the fact that we have an analysis from our depthstudy, we will start directly at the design stage where we will look at how to improve some of the issues we encountered in our depthstudy [50]. These will then be implemented and the updated framework tested against its requirements. Any new aberrations will then be looked into and a new version developed. Due to time limitations, we do not expect to solve every issue that surfaces and any issues not fixed will be documented in Chapter 24.

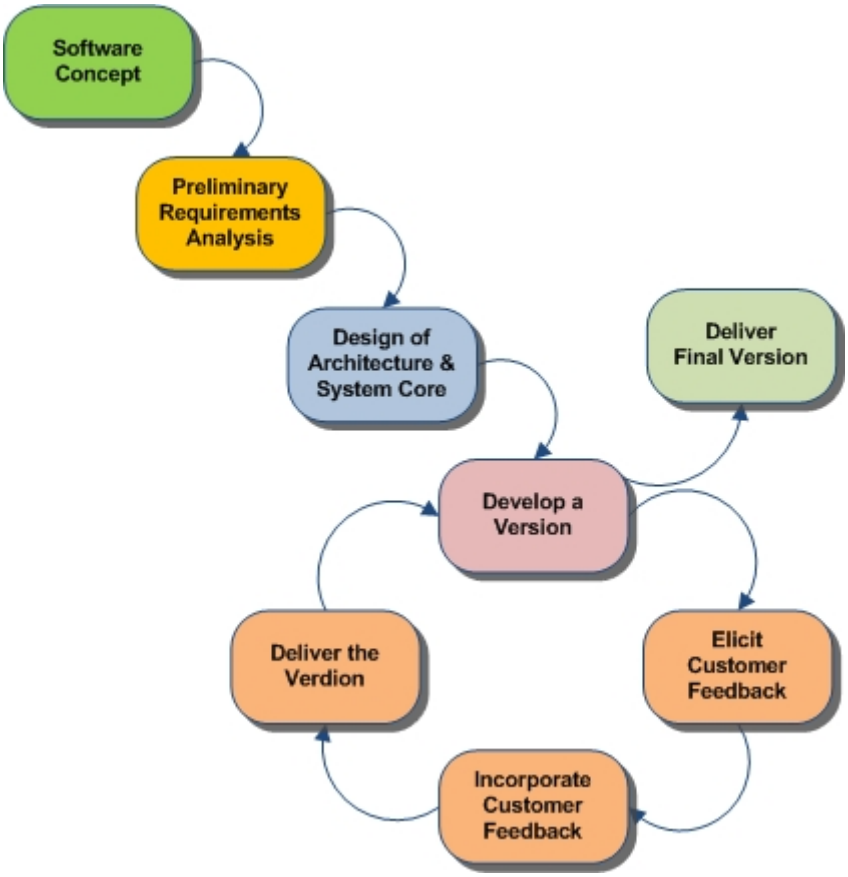


Figure 4.1: Evolutionary engineering approach life-cycle

Design and implementation of the game prototypes will follow a very rigid waterfall cycle. The waterfall method has five stages as illustrated in Figure 4.2. The strength of this method is that none of the stages needs to be repeated and planning and design can be done early in the process. The test will form the basis for an evaluation which will result in suggested improvements in Chapter 24.

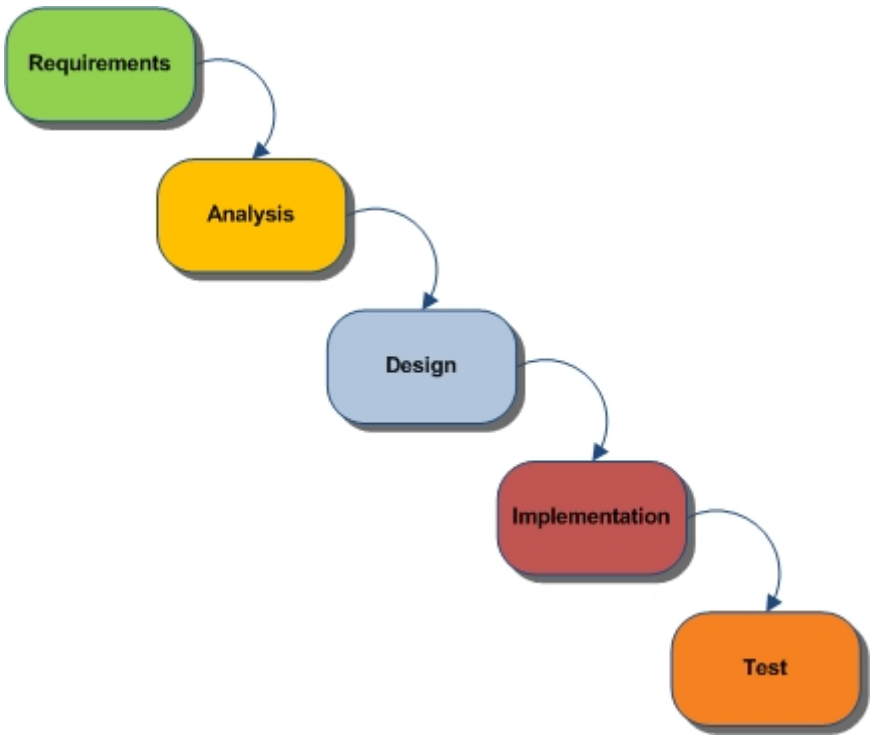


Figure 4.2: Waterfall engineering approach life-cycle

4.2.2 The Empirical Approach

Empirical research methods can be divided into two categories:

- ➢ Quantitative research methods: such methods collect numerical data (data in the form of numbers) and analyse it using statistical methods.
- ➢ Qualitative research methods: such methods collect qualitative data (data in the form of text, images, sounds) drawn from observations, interviews and documentary evidence, and analyse it using qualitative data analysis methods.

Qualitative methods will be used in defining how social gaming can be used and affects our concept through. It will also be used to classifying multiplayer games based on social gaming attributes, so they can be used to design and implement new game prototypes. Both of these will undergo a case study to build a theory for what works for our concept. Lastly we will use direct observation of players in order to evaluate user friendlyness and popularity of our concept.

Quantitative methods will be used to test technical aspects of the framework, mainly Bluetooth traffic and optimizations. We will use this by measuring the time information from the server use to reach the clients. We will measure this at the start of the project and the attempt to optimize the code. After optimization of the framework we will measure again in order to determine if we were successful.

CHAPTER 5

Test Environment

We want to have MOOSES running at public places and where groups of people meet. To be able to achieve this, we have to test and evaluate MOOSES with different hardware that are possible for groups of people to use and see. This chapter describes what hardware we have available to test on, which can give us some ideas to possible future uses.

5.1 Display Technologies

We have two important displays to test on (exclusive our computer monitors), both of these will MOOSES be running on in the future, and so must work good on these.

5.1.1 42" LCD-TV

An expensive newer type of LCD-TV can be bought today which have a native 1080p display. NTNU has one of these TVs available for us to test on, which makes testing easier than booking and going to Nova Cinema. These TVs have a resolution which is good enough for SlagMark in full resolution, and big enough that players are visible for a group of people 3-4 meters away.

5.1.2 Sony 4k SXRD projector

The largest cinema auditorium at Nova Cinema in Trondheim, has installed the first Sony 4k [49] projector ever produced (see Figure 5.1). The Sony 4k offers unprecedented features such as a 4096 x 2160 pixel resolution and a high contrast ratio. To be able to display this amount of pixels it has to use 4 HD-SDI inputs as a source, where each provides a 2048x1080 (2K) image which it composes by simply displaying one in each corner.

The projector has support for different video input interface cards with different types of inputs. Nova's projector has an interface-card mounted in the projector, with support for one HD-SDI input, which it scales to display as full-screen. This makes it possible to display an image from a standard graphics card from a computer by using a converter.

Figure 5.1: Sony's 4K projector

5.1.3 High Definition Serial Digital Interface

High Definition Serial Digital Interface (HD-SDI) is a digital video interface used for broadcast-grade video. This interface is used in high-end projectors and monitors, and is the interface for displaying an image on the Sony projector at Nova. The HD-SDI interface provides a nominal data rate of 1.485 Gbit/s which supports a resoultion up to 1080i at 60Hz or 1080p at 30Hz (dual HD-SDI links for 1080p at 60Hz).

5.1.4 Gefen DVI to HD-SDI scaler

To be able to display a signal from the computer on the projector we have to use a converter from DVI to HD-SDI. Nova has bought one from Gefen [17], which will allow us to display an image of 1920 x 1080 at 60 Hz, which we think should be good enough. It is possible to increase this resolution further, either by buying more converters, or by buying a NVIDIA Quadro FX 5500 [39] graphics card from Nvidia. This card can render and output directly to 4xHD-SDI interfaces. This card is currently priced at around 60000 kr.

5.2 MOOSES Server Hardware

We have two different servers we will test MOOSES on, one very good and one old computer.

5.2.1 Nova's computer

Nova and Midgard Media Lab has a very fast computer available for testing at Nova. It is a HP xw9300 Workstation which is close to 0,5 year old at the start of this thesis. It is the fastest machine we have tested MOOSES on, and we have not noticed any performance issues there. Unfortunately we have only tested with a maximum of 5 players with this computer.

Specifications for the workstation are:

- Processor & RAM
  - Dual AMD Opteron 200 series processors with AMD64 Technology & HyperTransport, dual core processor 270 (2.0 GHz).
  - 16 gb ram.
- Chipset
  - NVIDIA nForce Professional with AMD-8131 HyperTransport PCI-X tunnel and a fast SATA II disk interface.
- Graphics card
  - NVIDIA 2xQuadro, SLI enabled over PCI-Express.
- Operating System
  - Windows XP x64.

5.2.2 Morten's computer

Morten's computer has had the highest load, up to 12 players. It started suffering from the load in SlagMark where the framerate went visibly down. It is a 3-years old computer.

Specifications:

- Processor & RAM
  - Athlon XP 3000+ CPU running at 2.0 GHz, Socket 754.
  - 1 gb ram.
- Chipset
  - Abit KV8 Pro.
- Graphics card
  - NVIDIA Geforce 6600 GT, AGP.
- Operating System
  - Windows XP x32.

5.3 Java J2ME & MIDP 2.0

To run the client, we have to use a mobile which has support for Java's J2ME profile MIDP 2.0. We are personally in possession of one Sony Ericsson K800i and one Nokia N73. Our supervisor has lent us two Sony Ericsson K750i phones, and we have access to one Sony Ericsson K610 and W850. All these phones support MIDP 2.0.

5.4 Bluetooth communication

Bluetooth communication between server and clients will happen through Bluegiga's Bluetooth access point [52]. It has support for 21 simultaneously Bluetooth v2.0 connections, which makes it perfect for testing up to 21 players at the same time. One accesspoint provided Bluetooth coverage to the entire back of the auditorium at Nova 1, which has room for 440 people. More information about the access point is available at Section 8.4.5. There are few mobile phones that are on the market today which do not have support for Bluetooth. Phones that support Bluetooth v2.0 have been for sale since at least Sony Ericsson's K750i which came out in the second quarter of 2005.

CHAPTER 6

Development Tools & Software

This chapter presents the different tools used to create this report and the framework.

6.1 Development Tools

In the development of the MOOSES framework and the test game we have utilized a multitude of tools. This section presents these tools used.

6.1.1 Eclipse With Plugins

Eclipse [14] is a an open source development platform with lots of available plugins for different purposes, e.g. writing in LaTeX, creating applications for mobile phones and finding code statistics.

TeXlipse is a plugin for writing LaTeX documents in Eclipse. It includes features like syntax highlighting, command completion and bibliography completion [19].

EclipseMe [18] is a plugin for developing J2ME MIDlets in Eclipse. A Java Wireless Toolkit must be installed on the system for the plugin to work.

6.1.2 MiKTeX

MiKTeX is an implementation of TeX and related programs for Windows on x86 systems [48]. The MiKTeX distribution contains many features including the pdfTeX compiler which generates a pdf document. The compiler is used to produce this document.

6.1.3 Concurrent Version System

In order to keep track of versions of code and documentation, we have used a Concurrent Versioning System (CVS). NTNU has a Linux server with CVS support which we have used in this master thesis. The system also makes it easy to work from any computer that has Eclipse and the necessary plugins.

6.1.4 Microsoft Visual Studio

Visual Studio 2005 [33] is Microsoft's flagship in software development for the Windows platform for computer programmers. It centers on an integrated development environment which lets programmers create standalone applications, web sites, web applications, and web services. We used this exclusively for the development of our depthstudy testgame in C++.

6.2 Emulators

Applications made in J2ME need to be tested. The applications can run on mobile phones, but it is a tedious process to do for each iteration as we have to send the application over and install it. Fortunately, emulators can be used instead. There are several emulators available and each mobile phone producer has its' own toolkit. In our study we will use Sony Ericsson's SDK [1].

6.2.1 Sony Ericsson SDK

The Sony Ericsson software development kit is a wireless toolkit that can be used to emulate Sony Ericsson mobile phones that support Java ME technology [2]. The kit can emulate the following phones: W800, W600, W550, Z520, K750, K600, K300, J300, Z800, V800, S700/S710, Z500, K700, Z1010, K500, K508, F500i, P900, P910, Z600/Z608, T630-T628, T637 and T610 Series (T610, T616 and T618). P900 and P910 is not supported. The SDK can run applications that uses both MIDP 1.0 and 2.0. In contrast to Sun's wireless toolkit, the Sony Ericsson toolkit can run several instances of an application using emulations of different phones so that each phone has a recordstore, filesystem and PIM database. This enables us to test the applications with different phone models within the emulator, without having to use actual phones. However, using this SDK will only test how the framework and applications will work on Sony Ericsson mobile phones. Therefore, the framework must be tested on different real phones ensure that the software will run on different hardware.

[1] SDK - Software Development Kit

Part III

Prestudy

CHAPTER 7

Introduction

In this part we will provide most of the research material for this report and will cover the following subjects:

Hardware Technology describes what hardware is needed and used during the project period. It will also look at some technologies that are used on the client side as controlling mechanism in other concepts, to see how these can help bring the MOOSES concept to new levels.

Framework Technology describes underlying framework technologies used by the MOOSES framework and a brief look into the MOOSES architecture, which is needed for evaluating the framework.

Depthstudy includes essential research from our depthstudy. It is looking at game genres and visualization techniques to acquire what would work well with the MOOSES concept. This is used in developing new game concepts.

Social Gaming looks at how social gaming works from three perspectives, social gamer types, social gaming categories and how games affect real-life socialization. This research is used both in the development of new game concepts and when looking at improvements for the framework.

State of the Art looks into todays gaming communities, how it affects social gaming and provides a basis for one improvement to the framework. It also looks at todays multiplayer games, with focus on cooperative and social gaming to see what todays games can bring to our concept on those two aspects.

CHAPTER 8

Hardware Technology

We will have a look at the technical hardware resources that MOOSES requires. These give some clues and possibilities to what we can do and improve on. The resources together with a short description follows below.

8.1 Display

MOOSES is best suited for a big screen, and a bigger screen than a projector is hard to get. To get good graphics and a big playing field on a big display, we have to have a good resolution. In our depth-study we made a game suited for the 1080p HDTV-resolution (1920x1080), which is rapidly becoming affordable with new displays today.

There exists one widely known certification which ensures that the audience gets a good experience from a movie, THX. A THX cinema follows some specific engineering standards for acoustic performance, background noise, sound isolation and image quality, so that a THX Certified Cinema [31] promises that every seat in the auditorium is a good one. An example is that the THX movie certification standard says that the viewer in the seat farthest away from the screen should have the display fill at least 36 degrees of the person's vision, and that every seat should have a clear viewing (no seats obstructing the view). The same certification also gives some minimum requirements for the audiosystem, to ensures that the sound is good. These features ensures that the audience gets better immersed in the action on the display, so a THX Certified Cinema is very good for MOOSES use. We can certainly approve of this after testing MOOSES at Nova Cinema's biggest THX certified auditorium while writing our depthstudy.

If you sit too close to the display, you will be able to perceive the pixels which the image is composed of, this is unfortunate. On the other end, if you sit too far away, you can miss some of the details that may be important. In the standard EG-18-1994 from the Society of Motion Pictures and Television Engineers (SMPTE) recommends that the screen size for home theater

use should occupy a minimum 30 degrees field of view in the horizontal plan. Alternatively, the ideal TV viewing distance should be such that the screen width occupies an angle of 30 degrees from the viewing position.

A common guideline (see [20]) is that newer HDTV display have a recommended optimal viewing distance of twice the screen's width. For a 42" HDTV this translates to 2.1 meters. Further, the recommended minimum and maximum distance for a 42" HDTV is 5.5 - 10 feet which translates to: 1.7 - 3.0 meters. On a 42" LCD-screen this should mean that it is suitable for a group of around 15 people to play at the same time. We can improve the number by making the game scale the graphics up to a bigger size so people can see the game from farther away.

The same basic principles should work well for a video projector. A video projector's display size is usually limited by the size of the canvas it projects on, and the brightness of the room contra the brightness from the projector's lens.

MOOSES has not taken full advantage of surround-sound systems that exists today. This can help the audience get immersed in the games if they are surrounded by speakers, so this should be taken advantage of. It can be a problem with requiring bars and other similar public places to set up a full surround-system so the games should not require a surround-system to be played. Otherwise, the more sound a system can make, the better it is suited for MOOSES.

8.2 MOOSES Server Requirements

The server is a critical piece in our concept. It should allow good performance for developers for years to come, and it should be standardized to allow for easy testing at the developers end, while it should be as cheap as possible. During our depthstudy testing we have tried different hardware, and as such we have some requirements.

The most value for the money is an ATI or NVIDIA graphics card. Intel is not recommended as they are low performance. The cards come in various price ranges and are very fast for simple 3d-games. They also provide 2d-graphics acceleration and good display quality. The card should be PCI-Express as it has a faster connection to the motherboard.

Memory is important to reduce diskswapping and give the games enough free memory to use. We recommend 512 Mb of RAM, this should be more than enough for simple games.

The CPU is important for fast loading times and especially for unaccelerated games. A newer one, running at 3GHz, should do great.

8.3 Controllers

In this section we will look at what other possible controller hardware that is out in the market today, which can give MOOSES new gaming concepts.

8.3.1 Controller types

The players have to be able to control their characters on-screen. We came up with different controllers that all could work in our depthstudy, but they impose some restrictions.

- ➢ Players bring their own controller.
  Players that wish to play have to buy/rent their own controller which they have to bring. When they arrive at the cinema they must login/register with a representative because they can't login with user/password at the shared projection screen. This limits spontaneously playing and increases the cost for the cinema.

- ➢ Built-in controllers.
  Every seat which should allow a player to play must be modified to contain a controller. The controllers must be well secured so that they won't be stolen/broken when they are not being used. The player still have to register with the cinema as the previous item.

- ➢ Use mobile phones.
  Players can use their own mobile phone as a controller. This reduces the administrative costs for the cinemas, as the players take care of their own controller and it's possible to pay and register with the mobile phone. Mobile phones are so common today that it should not significally limit the user base.

  Other possibilities arise when we use a mobile too, most mobile phones today have a camera integrated. We can for example use the camera to take a picture of the user and display that as his avatar. We can use the screen on the mobile phone as an extra information display, and use vibration, light and/or sound for feedback.

The framework should be able to support all of these with proper implementation, but as of today only the mobile phone is actively supported.

Response times vary with the different suggested controllers, hardwired controllers have virtually zero response time. Mobile phones will transmit by Bluetooth which will add some time to send and receive. In [8] it says that with response times below about 150ms, the human eye cannot perceive the delay between when an action is requested and when it is fulfilled. This would be a very nice goal to achieve for games in the MOOSES platform.

8.3.2 Controllers available today

Today there are many different gaming-consoles available. The newest consoles for TV-use, are Sony's PS3, Nintendo's Wii and Microsoft's Xbox 360. Portable gaming-consoles include Nintendo's DS, Sony's PSP and Nokia's NGage. Another important gaming platform which has much hardware available, is of course computers.

The latest generation of gaming consoles use wireless controllers. Xbox 360 is the most basic one, which is just a regular wireless controller with support for vibration. Playstation 3's controller has been improved since the Playstation 2 controller, it supports a new technology called SixAxis. SixAxis is basically that the controller can sense rotational orientation and

translational acceleration along all three dimensional axes. This can make the games aware of the players movements and provide better immersion into the game. Last, Nintendo Wii has come up with a new controller which can also detect the position in 3d. This makes the console's games able to use the controller for instance as a pointing device, gun or tennis racket. Because of the controllers simplicity, Wii has become popular even for people not normally playing on gaming consoles, see for instance [21].

Nintendo Wii and Playstation 3 both use Bluetooth for communication between the controller and the console, and both protocols have been made available. So by installing special drivers, players can start using the controller with their computer (see [29] and [42]). These controllers should be easy to use with MOOSES, and performance should be as good as when they are playing at their console. Authentication could happen by entering a username, and password could be a specific combination of rotations of the controller and keypresses. The Nintendo controller has even got a possibility to, on request, store data in the controller's memory, which would let us store data, for instance information for a faster login.

There are some games that have become very popular because they imitate real-life things like playing a guitar with a simpler control. GuitarHero has its own controller with just 5 buttons and a bar you hit to play the note. This makes playing a guitar accessible to a wider audience, without requiring the player to spend much time learning the real thing.

Some controllers makes the game more realistic, like a steering wheel for racing games, usually with force-feedback which sends vibrations and accelerations to the wheel. Other types include guns, with sensors that help the game decide where you shoot at, to carryable swords that detects when you strike out and sends the command to the game. A very specialized controller we found was a chainsaw made for only one game see Figure 8.1(a), Resident Evil 4. Another type is the included controller with the quiz-game series Buzz!. These look like the controllers found on tv game-shows (see Figure 8.1(e)) where the audience participate with pressing buttons correlating with the answer they are making.

Dancing in arcades has been very popular in Asia for a while and is gaining popularity in the rest of the world. It has even become an official sport in Norway [11]. There exists several standalone dancing mats (see Figure 8.1(b)) which works as a controller for consoles and computers. These could also be used in a limited version of MOOSES which only features dancing-games where a great number of people would compete for the highest score.

Another interesting combination is to use a lightgun to allow the entire cinema to shoot at the screen. Older lightguns worked only with CRT-screens, but newer ones (an example is Figure 8.1(d)) work well with all types of monitors, which makes it possible to use them in cinemas. The Nintendo Wii controller was shown with a casing which makes it resemble a gun at E3 in 2006 [22], but the casing has not been announced or shown later.

A game which utilized a webcam that you plug in to your console, EyeToy, was released late 2003. This product can make some EyeToy aware games calculate where you are on from the picture, which makes the player able to control the game with his body. In EyeToy: Groove the player must hit targets with their arms on the edges of the screen to the beat of the music. This smart concept would be possible to use for MOOSES, it would for instance make an entire audience play together in a room to solve specific tasks, puzzles or record a video from a gaming session.

(a) Chainsaw controller for Resident Evil 4

(b) Dance mat

(c) Sword controller

(d) Lightgun

(e) Buzz! controller

Figure 8.1: Different controllertypes

Another common hardware-item which has recently become very popular in gaming is microphones. SingStar uses an external microphone to calculate the pitch of the singer and compare it to the notes in the song. The voice is then transported out through the audiosystem again for the enjoyment or terror of the audience.

An interesting possibility that came up during depthstudy testing at Nova, was that they Nova has equipment for sending out stereo-imaging for 3d-glasses. This could bring a whole new level of immersion in the games, as the players would see the world coming out at them and be able to judge distances in the game. Coupled with a motion-detecting device like the controllers for Nintendo Wii which would make the players be able to navigate in 3d, this would become one very impressive technological combination. Unfortunately Nova is probably one of few cinemas which has stereo-imaging capabilities, which makes the games limited to one place.

8.3.3 Future hardware

We will here take a look at what the future can bring so we are prepared to take advantage of these. Some of these have started to come out now, but will not gain a big enough marketshare for MOOSES for some years.

Figure 8.2: Controller for one special mobile phone

There exists one Pocket PC which has an integrated joypad (see Figure 8.2). If someone could provide these joypads for popular phones, it would have been ideal for MOOSES as it would

allow for a more comfortable gaming experience than the keypad of some of the smallest phones today.

GPS (Global Positioning System) is a system which enables a receiver to know its location, speed and direction outside. A GPS receiver is available as external devices, and even built into some high-end phones that have started coming out this year. These can enable us to build a better community around the players, as we can notify other players of their friends positions and set up meeting points. GPS could also enable positioning based games outside, for use with many pervasive gaming [1] concepts.

A recent development has been to start including accelerometers in the phones. Nokia's 5500 Sport [37] comes equipped with 3 accelerometers [2] which can be accessed with a C++ plugin they have available. By including this, the phone is able to notice movements and rotations in 3d. These features have been utilized in an included game, where the player rotates the phone to rotate the playing field in the game. More phones are starting to come equipped with these as they are cheap components and gives some new advantages like controlling menus with movements in the air [63]. But it will probably take some time before an interface with Java J2ME is ready and the devices gain a big marketshare.

Another possibility, is to add support for other platforms like PlayStation Portable which have integrated wireless network access, laptops or PDAs. These would require porting to the different consoles, but would allow for a greater detail in the games because of the faster platforms. It would also be possible to upload new client-games by air in a format which allows the client to be ran without scripting, which is much faster. This could also allow for games which support 3d on both client- and server-side.

Microsoft has recently announced a 'coffee-table' dubbed Surface [43] with a multi-touch input sensor, which can detect many fingers at the same time. This specific device uses a display resolution at 1024x768, but with a bigger device we could accomodate for many players at the same time. This device suits MOOSES well, but will require other types of games than the ones we will look at, due to different input controllers and the different placings around the table (players are looking down). This device would make people gather around it closer, and will probably be even more social than players spread around in a cinema.

A cheap method to make paper-interfaces was recently announced in Sweden [15]. These results could enable us to print controllers on a big wall, ie outside in the city, and let a crowd play with MOOSES on a big-screen placed a distance away. The paper can also transmit sound and could have some small lcd-screens in it or over it, which would make the controllers resemble and work as good as the mobile solution we developed in our depthstudy.

Samsung was mentioned on a lot of mobile review sites when they sought a patent [41] for a mobile phone with perfume. This could mean that we one day can have smelling games for MOOSES!

[1] Pervasive gaming is gaming that transports the classic computer game from the virtual world into the real world. The players move through the physical world and experience the game through interactions with the mobile terminal and the physical world.

[2] An accelerometer is an electromechanical device that will measure acceleration forces. These forces may be static, like the constant force of gravity pulling at your feet, or they could be dynamic - caused by moving or vibrating the accelerometer.

Figure 8.3: Multitouch surface

8.4 Bluetooth

MOOSES uses Bluetooth for its communication with the mobile clients. The server controls what packet-structure we send over Bluetooth, and to optimize these and to know limitations, we need to get a deeper understanding of the protocol. Bluetooth is a communications protocol in the standard radio band. It is designed for low power consumption, and the different power schemes are classified by range (1 meter, 10 meters, 100 meters). It is based around a low-cost transceiver microchip in each device. Bluetooth lets these devices communicate with each other when they are in range.

8.4.1 Specification

Bluetooth is a wireless radio protocol in the license-free ISM band starting at 2.402 GHz and stopping at 2.480 GHz. The protocol uses 79 channels displaced by 1 MHz that it hops between each sent packet (only 23 in a few countries). Each Bluetooth network has one device that is assigned as master, which then can support up to 7 active clients. The master and the clients hop between the channels in a pseudo-random pattern based upon the master's Bluetooth address. This happens at a rate of up to 1600 hops per second. Every packet that is sent is divided by a 625 microsecond time slot, in which the master controls which device should send or receive a packet. The master can transmit at even-numbered time-slots, the slaves can only transmit at odd-numbered. If a device wants to send a packet greater than one time slot every device will stay at the same channel until the whole packet is sent.

A Bluetooth device playing the role of the "master" can communicate with up to 7 devices playing the role of the "slave". This network of "group of up to 8 devices" (1 master + 7

slaves) is called a piconet. A piconet is an ad-hoc computer network of devices using Bluetooth technology protocols to allow one master device to interconnect with up to seven active slave devices (because a three-bit MAC address is used). Up to 255 further slave devices can be inactive, or parked, which the master device can bring into active status at any time.

At any given time, data can be transferred between the master and 1 slave; but the master switches rapidly from slave to slave in a round-robin fashion. (Simultaneous transmission from the master to multiple slaves is possible, but not used much in practice). Either device may switch the master/slave role at any time.

There are various protocols on top of Bluetooth which for example can synchronously send data, but these are not interesting for our purpose. When we send data, we will be using L2CAP or RFCOMM which is based upon L2CAP. Both of these protocols are again based upon an asynchronous communications link (ACL) protocol. This limits the packets we will be using to ACL-packet types. So there are 6 packet-types of interest to us in Bluetooth v1 (Figure 8.4 over the black line), they have different properties with regards to payload size and error-correcting. The Bluetooth device will automatically select the best packet-type based upon the link quality (RSSI - Receive Signal Strength Indication). L2CAP is packet-based which means that it resembles a UDP-packet from the Internet terminologies, RFCOMM is stream-based which makes it resemble a TCP/IP-stream.

Logical Link Control and Adaptation Protocol (L2CAP) is the lowest-level Bluetooth protocol that can be accessed by an application. The protocol overhead for L2CAP is 4 bytes. L2CAP is recommended if you have a small amount of data and you need fast response times.

RFCOMM is a Bluetooth protocol based on L2CAP. The protocol overhead for RFCOMM is between 4 and 5 bytes for small packets. For every 127 bytes of data, the header increases in size by 1 byte. So the overall protocol overhead is about 8 to 9 bytes for data less than 127 bytes (4 bytes from L2CAP and 4 to 5 bytes from RFCOMM).

Figure 8.4 shows an overview of packettypes available to us. RFCOMM and L2CAP both abstract away these packettypes, the Bluetooth hardware can select the transport packettype based upon the receiving strength (see [9] – 4.1.7 Channel quality driven data rate change). This means that we can not know which resulting packettype the data is sent with and it might vary depending on receiving strength and interference.

The paper [55] simulates throughput and collisions with Bluetooth version 1.1. The paper advices that 42 piconets in the same area should be the maximum. Bluetooth version 1.2 and 2.0 brought improvements not accounted for by this paper, so we should have tolerance for even more piconets. This is very good news for our concept, we should be able to support a very high maximum number of clients. The recommended value of piconets makes us able to support close to 300 active clients in one area! When we make MOOSES use as many 1 time-slot packets as is possible, we should get both good scalability and good performance in-game.

Bluetooth is used for time critical gaming today. The controllers for both Nintendos Wii and Sonys Playstation 3 uses Bluetooth, which tells us that Bluetooth should have little performance and lag impact for our needs. As we can tolerate more lag than both of these gaming consoles, we should not have any problems using Bluetooth. Some research into these controllers showed that someone had discovered the Bluetooth protocol used by Nintendo Wii [29], and we could

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)	Asymmetric Max. Rate (kb/s)	
						Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	no	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6
AUX1	1	0-29	no	no	185.6	185.6	185.6
2-DH1	2	0-54	no	yes	345.6	345.6	345.6
2-DH3	2	0-367	no	yes	782.9	1174.4	172.8
2-DH5	2	0-679	no	yes	869.7	1448.5	115.2
3-DH1	2	0-83	no	yes	531.2	531.2	531.2
3-DH3	2	0-552	no	yes	1177.6	1766.4	235.6
3-DH5	2	0-1021	no	yes	1306.9	2178.1	177.1

Figure 8.4: Available Bluetooth packet types and theoretical throughput

see that they communicated by using the smallest 1 time-slots packets to keep the performance up. These controllers are made with performance in mind, probably without big Bluetooth buffers.

Nokia has made Table 8.4.1 (from [10]), which shows real-world Bluetooth latency with regards to gaming. This table is based on the fact that every client always send packets and the master sends the state to every client (communication from every client to the master and out to all clients again). This table is not updated with regards to Bluetooth 2.0 which will help much with bigger packetsizes.

Number of players		2	3	4	5	6	7	8
Net data size of client packets [bytes]	Host uses DM1	4 [byte]	2	2	1	1	1	1
		20.5 [ms]	24.0	33.4	51.1	79.4	99.4	121.8
	Host uses DH1	9	6	4	3	3	2	2
		20.5	25.0	34.6	51.1	79.4	102.5	123.1
Average round-trip time for collecting client packet(s) and sending host packet(s) [milliseconds]	Host uses DM3	56	37	28	22	18	16	14
		30.0	31.5	38.2	62.8	117.0	161.0	202.3
	Host uses DH3	86	57	43	34	28	24	21
		31.5	36.5	42.4	78.6	146.4	193.7	212.8
	Host uses DM5	107	71	53	42	35	30	26
		34.0	39.0	55.3	90.3	173.5	222.1	269.2
	Host uses DH5	164	109	82	65	54	47	41
		37.0	47.0	98.0	121.4	237.4	290.4	348.6

Figure 8.5: Real-world latency with different types of packets.

Green cells: Client uses DM1 packet
Dark green cells: Client uses DH1 packet
Yellow cells: Client uses DM3 packet
Orange cells: Client uses DH3 packet

8.4.2 Bluetooth v2.0

There were many improvements added to Bluetooth v2.0, and simulations showed an increased throughput [28]. The specification added a new modulation schema and used it for 6 more packet types (see under the black line at Figure 8.4). Another feature added was Adaptive Packet Type, which is an improved technique for selecting which kind of packet-type should be used depending on the quality of the link and properties of the packet-type. Yet another improvement was using a bit-error prevention coding in the new packets (FEC). Multicasting

is also introduced, but we haven't found any support for this in J2ME. Technically version 2.0 devices have a higher power consumption, but the three times faster rate reduces the transmission times, effectively reducing consumption to half that of 1.x devices (assuming equal traffic load).

8.4.3 Java's implementation

Unfortunately, we must rely on Java's implementation [45] of Bluetooth at the mobile phones, which as of today restricts our possibilities with Bluetooth. Java has not implemented every aspect of every Bluetooth protocol, and one thing that was not included is the ability to broadcast messages (send one message to every client). This would have proved to be very beneficial in our depthstudy [50], there were quite frequently messages that we could have sent once, instead of 7 messages when we have 7 clients. Another feature that was dropped, was the ability to connect with unreliable connections, which would have let us control which packets must be resent and which could be dropped for optimal performance.

8.4.4 Limitations

Bluetooth specification allows connecting two or more piconets together to form a scatternet, with some devices acting as a bridge by simultaneously playing the master role in one piconet and the slave role in another piconet. These devices have yet to come, though were supposed to appear in 2007. This does not have a big impact on our concept, we can support many players if we use many piconets in parallell.

With practical testing with Bluetooth technology we found that the Windows OS only supports the use of one Bluetooth hub (usb stick) per machine, which then has a maximum of seven slave connections. This could have made severe limitations on MOOSES, but we found several alternatives to using Windows. Linux supports multiple Bluetooth sticks per machine which makes this a suitable platform.

8.4.5 Bluegiga Bluetooth accesspoint

Today MOOSES utilizes a Bluetooth hub which can support up to 21 Bluetooth connections per access point [52], and has support for converting Bluetooth RFCOMM connections to standard network TCP/IP connections for the game server. This allows for almost limitless connections to the MOOSES framework, because the maintainer can buy more accesspoints and place them in the same room without changing anything at the MOOSES side (see Figure 8.4.5 for physical architecture). This access point uses Linux as its operating system, which then in theory can make it possible to modify it even more for our needs as Linux is open source. It has one unfortunate drawback however, it is quite expensive per unit.

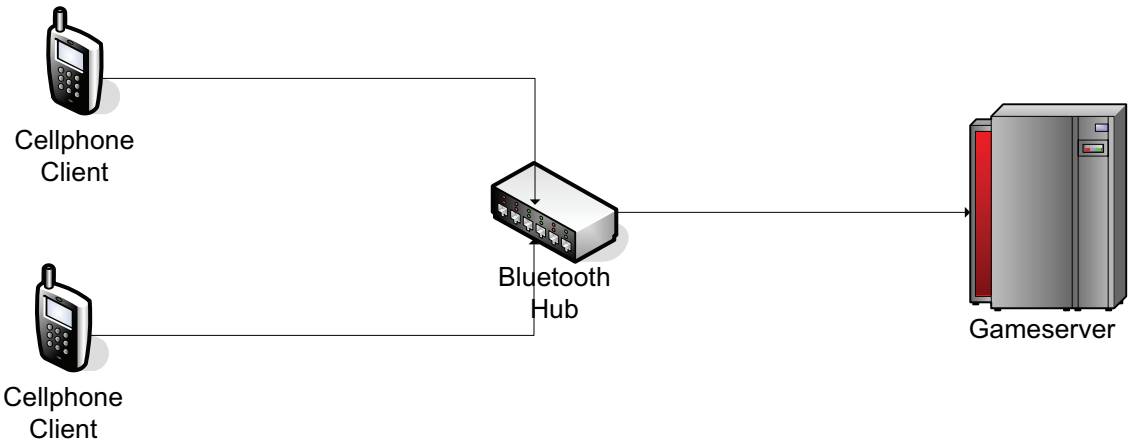


Figure 8.6: Standalone Bluetooth hub

CHAPTER 9

Framework Technology

We will have a look on the technical framework resources that are available for the testing and improvement. The resources together with a short description follows below.

9.1 Java 2 Micro Edition

Java 2 Micro Edition (J2ME) is a collection of Java API's developed by Sun Microsystems for development of software for resource-constrained devices such as PDAs, cell-phones etc.

J2ME for connection-limited devices, such as cell-phones in our case, consist of three layers. The Kilobyte Virtual Machine (KVM), the Connection Limited Device Configuration (CLDC) and the profile layer, MIDP2 in our case (see Figure 9.1).

The KVM is a smaller runtime environment for resource-constrained devices. The KVM's range is 40 to 80 KiloBytes.

The CLDC configuration defines a standard Java platform for small, resource-constrained, connected devices and enables the dynamic delivery of Java applications and content to those devices. CLDC is implemented as a set of additional classes contained in a separate package (the java.io, java.lang, java.util, and javax.microedition.io packages). This facilitates CLDC porting to different platforms.

MIDP2 is a more device specific collection of API classes that together with the CLDC provides the J2ME application runtime environment targeted for the mobile devices.

As J2ME is a subset of Java's J2SE[1], it has more functional limitations than J2SE, and the CLDC uses a preverification of the classes at the compiler side to reduce memory usage and device battery.

[1] J2SE - Java Platform, Standard Edition

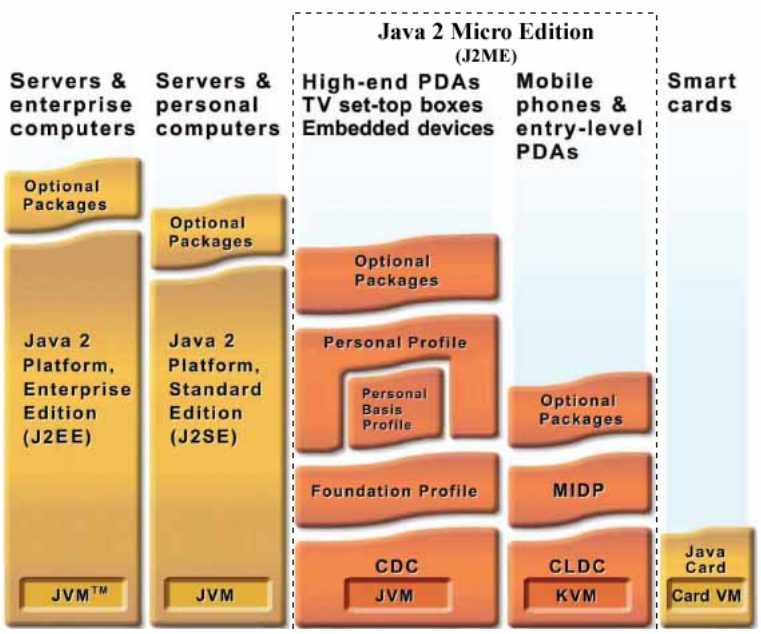


Figure 9.1: Overview of the Java environments [34]

9.2 Tellu Mobile Technology

Tellu [54] has several libraries which MOOSES is built upon. The most important libraries MOOSES uses, are ServiceFrame and ActorFrame both made by Tellu [54]. The ServiceFrame library provides functionality that differs from the general by making the developer easily able to make generic statemachines that can work at the server and or mobile, and thus making the mobile device more passive or active at the developers choice.

ServiceFrame is built over ActorFrame and contains common services in the form of components that are pluggable into applications. Figure 9.2 displays how the layers of ServiceFrame and ActorFrame are built up on the server side. MOOSES uses two services provided by ServiceFrame today, presence and login-systems. These provides login and discovery of participants at both the client and server-side. There are many services available that MOOSES does not utilize today, it would for example be easy to add mapping capabilities to guide a group of people to the nearest cinema or place featuring MOOSES by plugging in the correct components.

ActorFrame is an implementation of the concepts found in UML 2.0 (Figure 9.3. An essential concept in ActorFrame, which MOOSES uses, is Actors. Actors contain a statemachine where the behaviour of the Actor is made. Actions happen with received messages that are passed internally inside an Actor or with external Actors according to an Actor's unique ActorAddress. ActorFrame also contains different transportation types which the messages can be sent by, available types today are TCP/IP, UDP and Bluetooth. ActorFrame builds on a Peer-2-Peer architecture which can make MOOSES be deployed with different connection configurations, one could for instance allow clients to connect by a proxy from Internet.

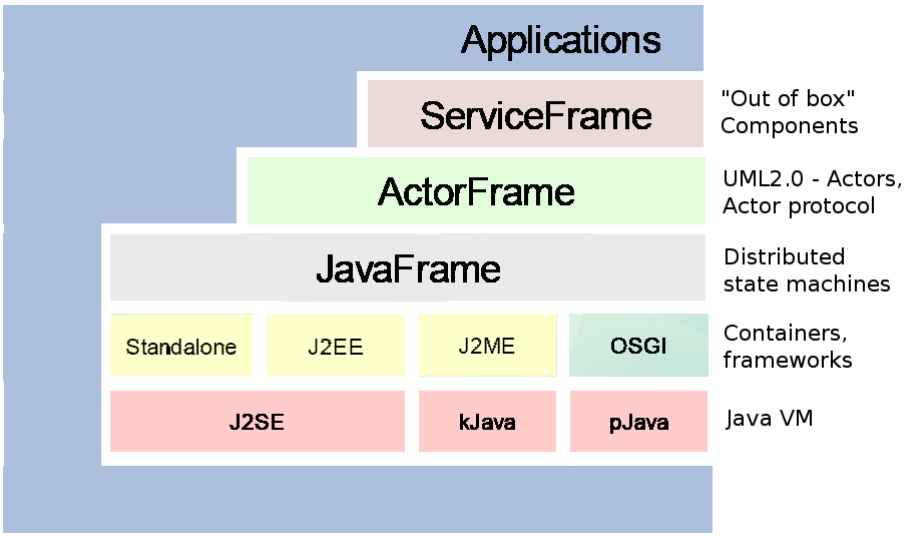


Figure 9.2: Serviceframe and ActorFrame overview [53]

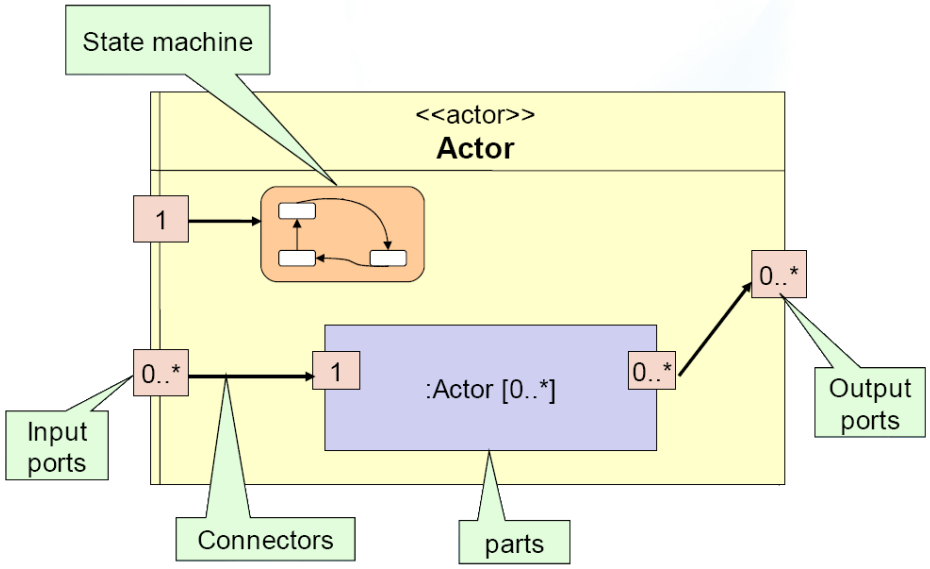


Figure 9.3: UML actor-statemachine [53]

9.2.1 Tellu J2Me GUI library

Tellu's GUI library for J2ME is a library somewhat similar to Java's Swing. The Gui-library consists of GuiWindow-classes that contains GuiPanels which again has GuiElements inside. These panels can have different LayoutManagers associated with them, which lays out the elements according to the optimal sizes of the elements and the mobile screen size. The library has support for themes, animations and various premade Gui-elements. The GUI library builds

on the J2ME GameCanvas, all actions are eventually painted on the GameCanvas.

The library has been tested on various phones, Sony Ericsson and Nokia phones are known to be working quite good. The GUI-library has also some optional extra connections to the ServiceFrame library, for instance resource loading can happen without bundling them with the application through ServiceFrame from an application-server where the libraries takes care of getting the optimal resource based upon the device, screen-size etc.

A newer version also has support for being ran on a regular computer, pda or setup-box. This portability can happen without changing more than a couple of lines of code at the application descriptor, and could enable us to easily make the client available for other platforms and communication links.

9.3 MOOSES Framework

We will here give an introduction of the MOOSES framework architecture and its requirements as designed in our depthstudy [50] with the inclusion of a new layered view. This is to give an overview of what is to be evaluated later in the report, and show changes done during this study. The functional-, non-functional- and environmental requirements for the framework, which are to be tested, can be found in Appendix D

9.3.1 High Level Architecture

The architecture framework is strictly module based with a separate package for each major type of classes. The top level structure of the architecture can be seen in Figure 9.4. This is to ensure that certain parts of the framework can be redesigned or be tailored for specific needs without rewriting the whole framework.

CommunicationServer Is responsible for providing the clients with access to the framework and set up the initial communication between all the modules. Once set up, the communication between modules should seem as though the modules were talking directly to one another.

UserAgent Is a module representing a user in the system, and should contain all information related to the user as well as communication to and from the user.

GameServer Is a module to handle information to and from the game into the framework. It is also responsible for loading game modules.

Game Is a module, which must support the supplied interface. The games are pluggable modules independent of the framework itself.

Login Is responsible for handle user authentication when the user tries to access the framework. Abstracting this as its own module gives the option to tailor login types specifically to the system owner.

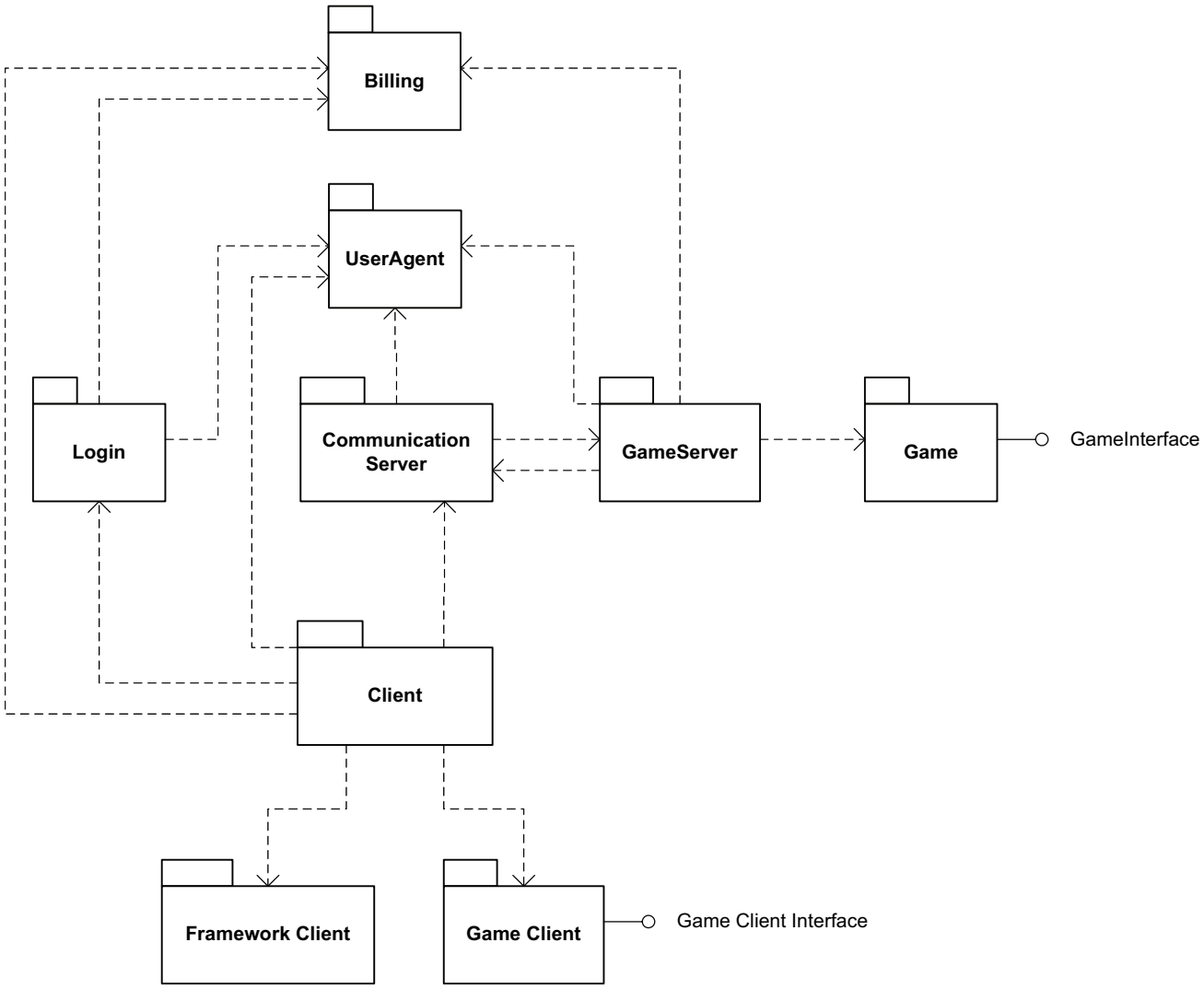


Figure 9.4: High level architecture view of the framework

Billing Is responsible for checking billing for the user as well as handling billing methods. Abstracting this as its own module gives the option to tailor billing types specifically to the system owner.

Client Is the client module of the system. It is responsible for handling client requests and feedbacks to the framework.

FrameworkClient Is responsible for handling client requests specifically related to the framework, like login and billing.

GameClient Is the module directly linked to a specific game. Having pluggable game clients gives the developers of games the ability to tailor controllers for their games.

9.3.2 Layered View

The MOOSES framework is devided into five layers (see Figure 9.5). At the bottom we have Sun's Java framework which Tellu's framework is built on top of. The framework is divided into two parts, the server side and the client side. The client and server communicates through the Serviceframe layer. On the server, the implemented games communicate with the framework through an interface. The games can be implemented in Java and C++, and even more languages are possible through C++.

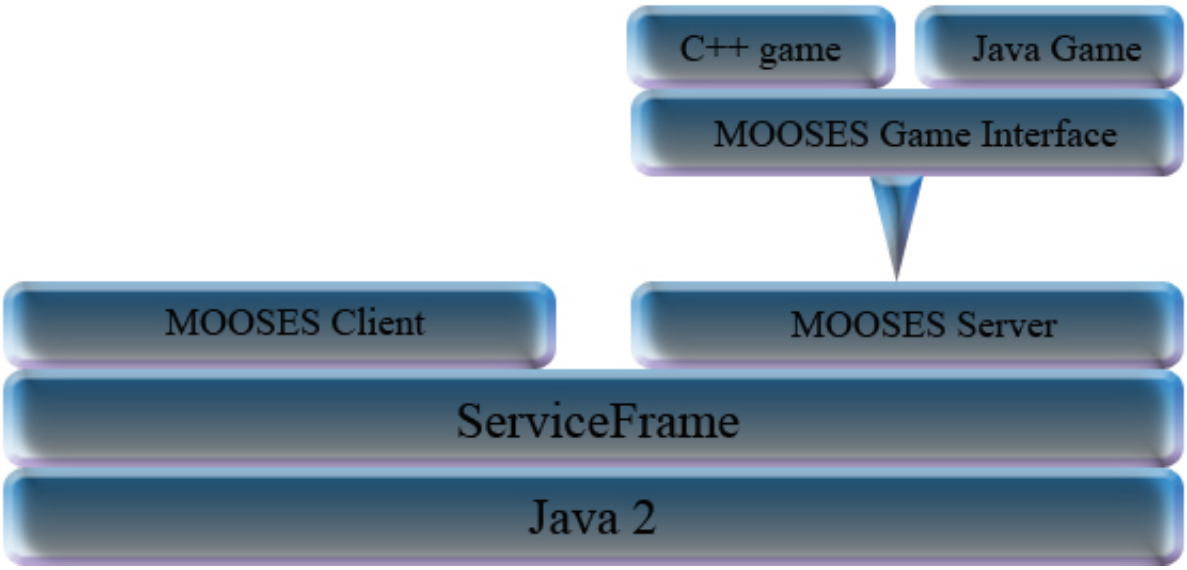


Figure 9.5: Layered view of the MOOSES framework

9.3.3 Physical View

The physical view is to help give an understanding on how to deploy the framework and give an idea of what hardware is needed to make it run. The framework requires at least one server and projector/screen to run. The server then runs the communication node as well as the game servers on one machine. The framework can however run the different modules on different servers with small modifications, if this should be required. More then one server is required to run more then one game server, though only one server is needed for the connection server-node. The server can then accept connection from various controllers(as long as they have an implemented client for the framework). At the current time, only a client for use with mobile phones is implemented. The client can connect to the servers with with either TCP/IP or Bluetooth. The physical view is laid out in Figure 9.6.

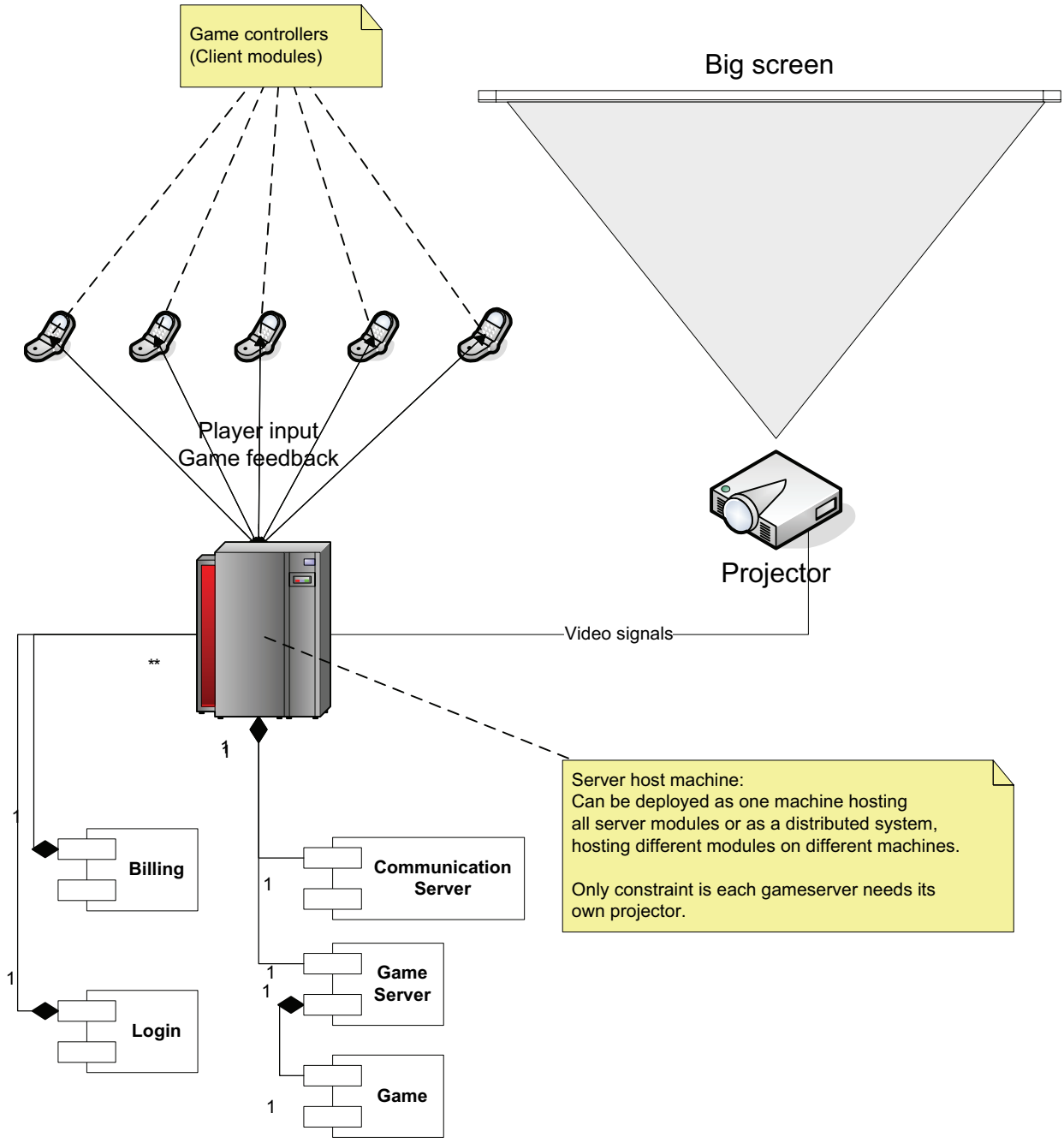


Figure 9.6: Physical view of the framework architecture

CHAPTER 10

Depthstudy

In our depthstudy we researched visualization techniques and game genres to find what game characteristics would work well with our concept. This research is important since we are developing new game prototypes for our framework, and as such we provide a small excerpt from our depthstudy of our findings.

10.1 Visualization Techniques

Due to the fact that our concept is limited to using only one screen, it is vital to look at how different visualization techniques perform on one screen. With the use of mobile phones as clients, we have the option of two screens, but since our framework does not require a mobile phone we cannot base our games on a two screen game design. We find that we can split the use of visualization into three categories:
All on one screen, Splitscreen and Network games with personal terminals.

Our depthstudy compared these techniques with regards to the following artifacts to find pros and cons for each type.

- Socialization
  How good does the technique reflect opportunities for socialization .
- Tactics Usage
  In multiplayer games, the winner is often the person which makes the best use of tactics and strategies. This requires the ability to conceal your tactics.
- View
  Reflects the techniques ability to provide space around the player and focus on the player's character.

- ➢ Latency
  As mentioned, multiplayer might get into the problem of latency which might lower the fairness and fun factor.

- ➢ Freedom
  Some games based on different visualization techniques might constrain the freedom of movement for players.

- ➢ Gameplay limitations
  The visualization technique might constrain the opportunities for a good gameplay.

- ➢ Hardware requirements
  We will focus on the number of components required, and exposes.

As the comparison shows (see Table 10.1), single screen visualization has some drawbacks. Some of these drawbacks, for instance the limited opportunities to conceal your tactics, may be solved by using the device display to make strategies and tactics and represent vital data. Limitations regarding freedom might be solved by zooming in and out within given boundaries, and in some special cases introduce splitscreen. As for the gameplay limitations, our solution using one main-screen and one personal screen opens new ways of thinking about multiplayer games and we will probably be able to make some good alternatives.

Visualization techniques comparison			
	Single screen	Splitscreen	Network Terminals
Socialization	Good socialization, all in same room		Depends on whether the game happens over Lan or Internet. Internet limits socialization.
Tactics usage	Impossible to conceal your tactics		Full tactics usage
View	Ok view for few players	Bad view that gets worse for each player added	Excellent view opportunities
Latency	No Latency		Internet play may give significant latency
Freedom	Limited, has to show all players in one frame but can zoom or move the camera	Good	Good
Gameplay limitations	Everyone are using the same screen which gives limitations	Weaker Gameplay limitations, every player is free	No limitations
Hardware requirements	Usually limited to console and screen		Expensive

Table 10.1: Visulization techniques comparison

10.2 Game Genres

In our depthstudy we researched different video-game genres in order to get ideas on which concepts would work properly on a big-screen. A brief description was given of the genres that has provided the backbone of games throughout history.

Game genres can be seen as a set of properties a game should have. If you know a genre of a game, you'll know what to expect from it. However, modern game development tend to not stick to one distinct genre, but make a collection of own suggestions. Later on, it will either be placed in a genre, or define its own genre.

The genres were presented with the following artifacts in mind:

- ➢ Description
  Gives a brief description of the superior genre.
- ➢ Common Features
  Providing a list of utilities that are common for the genre and some sub-genres.
- ➢ Types
  Provides a list and brief description of the most common sub-genres related to the superior genre.
- ➢ Multiplayer solutions
  Discusses common multiplayer implementations for the given genre.
- ➢ Common platforms
  Provides a view of the most common platform solutions for the genre. For instance arcade, consoles or PC. As consoles and PC converges the differences has been illuminated, but historically features like controllers, visualization techniques and processing power have had an impact on which platforms were the most appropriate for the genre.
- ➢ Solutions for our concept
  Gives a brief discussion of whether the genre is suitable for the multiplayer single screen concept. Also we take into account features and attributes that we may borrow from the genre.
- ➢ Examples and screenshots
  Provides screenshots, with a brief description, from popular games from the given genre or its sub-genres.

The research into game genres was then compiled into the following artifacts and put into Table 10.2.

- ➢ Genre
  Indicates genre.
- ➢ Multiplayer opportunities
  Describes the opportunities for multiplayer for the given genre, with respect to grade and mode.

- ➢ Max players
  Indicates the most common and most suitable number of players at the given genre .
- ➢ Features to concept
  Elements we will be able to take advantage of in potential hybrids or adoptions.
- ➢ Gameplay elements
  Brief description of gameplay and artifacts.

Genre	Multiplayer opportunities	Max players	Features to concept	Gameplay elements
Adventurer	Poor, cooperation	2 - 8	Storytelling, enchanting	Solve puzzles, communicate, collect things
Console Role Playing Game	poor, both cooperation and competition	2 - 8	Storytelling, statistics, players can manipulate their characters	collecting points and artifacts, develop characters, freedom
Shooter	Good, both	about 50	Instant Action, good projection methods	instant action, shoot everything that moves
First Person Shooter	Limited, Both competition and cooperation	8 (splitscreen)	Same as shooter	Same as shooter
Third Person Shooter	limited, both	About 15 or 8*	Use of surroundings, isometric battlefield	Bullet dodging, use of environment
Sport games	Good, Both depending on sport	Enough (sport dependent)	Endless possibilities, every sport can be used	Depending on sport, should be implementable for our concept
Racers	Good, competition	4 - 16	Either visualize from top, perspective or splitscreen	Competition and instant action
Fighters	Good, both	4 - 16	Cooperation in beat em' ups or Teamed competition in versus fighters	Instant action, fighting opponents
Simulations	Difficult, cooperation	Depends on simulation	Maybe in an extension	Often Educational, demands patience
Strategies	Good, Teamed competition	2 - 10	Reduced version, team-based, limited compared to the originals	Time demanding, collect resources, build armies, make strategic moves

Table 10.2: Genre sufficiency

* fifteen players using isometric fixed projection, eight using splitscreen

We found that our game has to be quite intense and instant action, more arcade-style like. It also has to provide good gameplay for all players. This limits our options quite a bit. The

arcade-style like games developed for this platform will therefore need to be ”single screen”, at least most of the time. But we also take into account that using mobile phones as controllers gives us opportunities to have some of the gameplay on the device. To provide a good gameplay to the player, we have to take into account the following issues:

- ➢ Give the player support to be able to focus on his character
  - Size of the characters
  - Visual differences of the characters
  - Clearness of the screen, providing airspace
- ➢ Giving the player subjective sound feedback
- ➢ Provide for the player to not get bored
  - Enchant the player

Both the number of players supported and the limited display yields limitations to the representation of the player. We want to give the player an interface allowing them to focus on their ”character”, which raises requirements to the size of the character as well as clearness on the screen.

10.2.1 Concept Considerations

There are several things to take into account to make game elements work on our platform. Most of the games should allow players to come and go as they wish. When a player logs on to the game, the game has to spawn the player in a appropriate location and maintain the balance, so that the players that have been logged on for a while would not dominate.

We will also have to take latency into account. Unfair balance of latency will decrease the enchantment of the player, so we will therefore look for concepts where latency have less importance where there should be a greater number of players. Latency can also be minimized by using predictive techniques to minimize the visible lag.

We have considered multiple concepts. The game could be semi-turnbased strategies, where the strategies can be made using the mobile device, and represented on the big-screen after each turn. Of course we can not have one player per turn because of the waiting time. Therefore the game may be divided into windows of time to give the players time to make their strategies. Each player will gain control of one unit. Assigning multiple units to each player could be an option if the number of players is low. The semi-turn based option would also eliminate the problem with latency.

Another turn-based approach could be to provide a 20 second window to perform strategies. Each players will wait in a queue, and every 10 second the window opens for a new player. This way there will always be action on the screen, so the waiting players will be entertained constantly. Progression causes a problem with this approach, and it might cause a problem keeping the balance when new players are added. Therefore the turnbased option would probably have to start the game with all the players, and be divided into sessions.

Realtime games usually provides better support for players that come and go. Realtime games are often more competitive which can provide facilities to drop or add a player in mid-game. If the game is more realtime strategical, techniques can be used to make the player start with a balanced properties (for example as much as the lowest player) or add the player to the currently loosing team.

CHAPTER 11

Social Gaming

Social gaming has existed since the birth of video gaming almost 30 years ago. Though the term was not used at the start, it was just as valid then as it is today. The first games to come out (like Pong or Tank for the Atari VCS/2600), were multiplayer games where players competed against one another on the same screen in the same room. The first version showed up as arcade versions, usually multiple machines at one location, and later moved to people's homes in the form of console games.

As technology moved on, allowing machines to connect to each other in a networking fashion, the games followed suit. This opened up for making games that allowed more players to participate per game, which in turn opened up a new form of socializing.

In our depthstudy [50] we only briefly touched the surface of social gaming. While it is a very important aspect of our domain, the main focus then was to solve many of the technical aspects behind a framework and game mechanics. With that research done, it is now imperative that we look into the social aspect of gaming, as this is one of the key strengths of this concept.

Two factors affect social gaming:

1. The players gaming style and how the games fulfill these
2. Game attributes that contributes to socialization

11.1 Player Gaming Types

In *Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs* [6] Richard Bartle investigate MUD[1] games to decipher and classify the different player types. He managed to compile players into the following four categories:

[1] MUD: Multi User Dungeon

Achievers give themselves game-related goals which they set out to achieve.

Explorers try to find out as much as they can about the virtual world.

Killers use tools within the game to cause distress (or, in rare circumstances, to help) other players.

Socializers want to communicate with other players, either through the game's communicative facilities or the use of other programs.

No player is specifically locked to one stereotype, and will often drift between all four depending on mood and current playing style. Bartle's research however states that most players will have a primary style of playing, and will only utilize the other styles as a means to advance in their main interest.

To get a better understanding of the different player types we will present them in more detail.

11.1.1 Achievers

Achievers main focus is the gathering of "points" in order to become more powerful in the game. All actions the gamer performs is subservient of this. Exploration is necessary in order to find new ways to gather points, socializing can give the player tips on how to advance faster or better, and killing might just be a way of gathering points in the game design. Example of a game focused on this player type is the strategy game Civilization IV.

11.1.2 Explorers

Explorers thrive on exposing all the inner and outer workings of a game. They want to visit and see all the content as well as figuring out the inner workings and mechanics. They often do this by unorthodox methods which were not perceived by the developers, and therefore often are the ones to find bugs and exploits within a game. Their involvement in killing or achievement is limited to the absolute minimal, and happens only when it gets in the way of their exploring. Socializing can be an informative way to acquire new ideas to try out, but the fun is finding these new ways on their own. Example of a game focused on this player type is the adventure game Sam and Max.

11.1.3 Killers

Killers get their enjoyment out of imposing themselves on others. While some do this to help fellow gamers, this is rarely the case and most killers seem to thrive on causing distress to their victims. The bigger the distress, the greater the killer's joy at having caused it. Point acquisition is often a necessity in order to gain power and ultimately cause more distress, while exploration is in its own way needed to find new ways to impose on players (or even new ground to impose on). Socializing can sometimes be a way of finding new tactics, taunt victims or potential victim's playing habits. Example of a game focused on this player type is the FPS[2]

[2]FPS: First Person Shooter

game CounterStrike.

11.1.4 Socializers

Socializers are interested in people, and use the games merely as a common ground to share experiences. Empathizing with people, sympathizing, entertaining, joking, listening, observing people are all keywords describing a socializer. Exploring might be necessary in order to understand what people are talking about or meeting new folks, while point scoring is needed in order to access new areas or ways to communicate and participate with players. Killing, however, is only utilized as an act of revenge on someone who has caused intolerable pain on a dear friend. Example of a game focused on this player type is the quiz game Buzz.

11.1.5 Player Interest

To get a better view of the players interest in a game, Richard Bartle made a representable structure in which to chart interest and thereby get a graphical representation (see Figure 11.1).

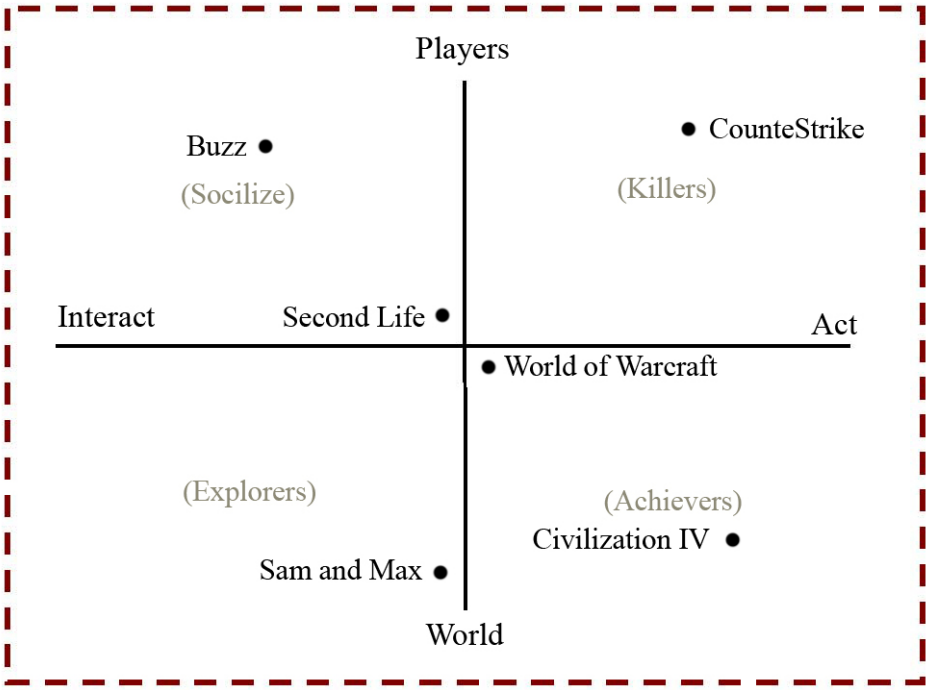


Figure 11.1: Richard Bartle's Interest Graph

The y-axis in the chart ranges from emphasis on player interaction to emphasis on environment interaction. The x-axis ranges from "playing the game" (Acting) to interacting with other players and the world (Interacting). The four corners of the chart shows which player type is associated with each quadrant. The axes can be assigned a relative scale reflecting the ratio of an individual's interest between the two extremes that it admits (or extremes that a game emits).

The games in the figure are placed to represent what players they are mostly centered on. The further from origo a game is, the more focused it is on that player type. On the other hand, the closer to origo a game is, the more its focused on satisfying all the player types. The figure comes a little short with games/players whose main focus lies in 2 or 3 areas and not the 3rd/4th however. Take for example Buzz, a game that mostly appeals to *Socializers* but also *Achievers*. It does not, however, appeal to *Killers* or *Explorers* in the same degree.

To summarize the player types we can list the following:

- ➢ Achievers are interested in doing things to the game, ie ACTING on the WORLD
- ➢ Explorers are interested in having the game surprise them, ie INTERACTING with the WORLD
- ➢ Killers are interested in doing things to people, ie ACTING on other PLAYERS
- ➢ Socializers are interested in INTERACTING with other PLAYERS

11.1.6 Player Type Balance

Following the description of player types, Richard Bartle [6] goes on to look at how population sizes of the different player styles affect one another. He describes how the different styles view one another, and then concludes with how more of one type will affect the other in population size. Figure 11.2 shows the conclusion of how an increase in the different styles influence one another. The lines on the arrows are associated with population increase or decrease from where they emerge while the arrowheads are associated with increase or decrease in the targeted type. Grey color indicates an increase, while black indicates a decrease. Scenarios which have no impact on population are not marked up.

11.2 Social Gaming Categories

In Section 11.1 we looked at how players could be categorized into four main groups. Using the graphical presentation of these on a gaming environment (see Figure 11.1) we can see that there are two groups that concentrate on social interaction, *Socializers* and *Killers*. It is not to say that the two other player types do not socialize, but in their extreme forms they are more interested in the game itself then the interaction with other players. In *Social Gaming Interaction, Part One: A History of Form*[3], Shannon Applecine looks at how games socially contribute to these roles.

11.2.1 Freeform Socialization

Freeformed interaction can be viewed as an un-goaled form of interaction. Its focus lies a bit outside how we traditionally view games. While games today almost always have something for players to achieve, freeform interaction looks only at socializing with other people like chat or storytelling.

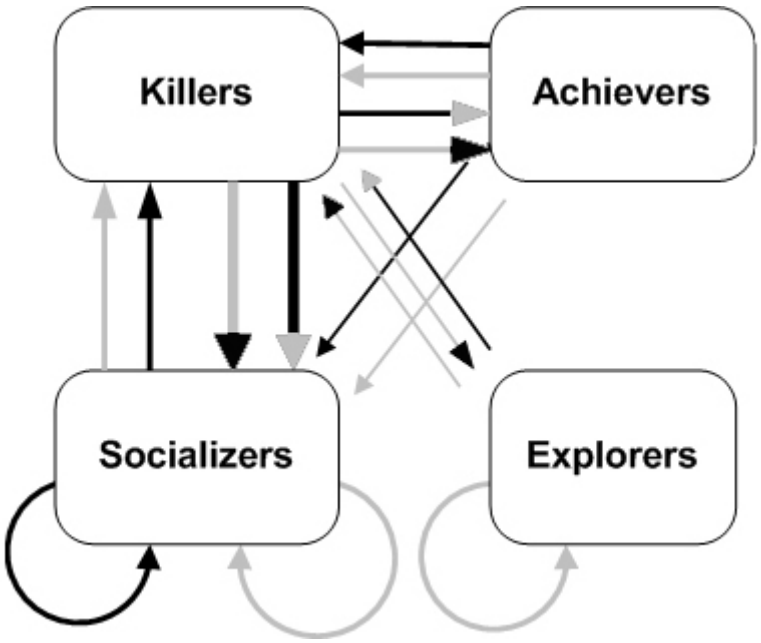


Figure 11.2: How the different gamer types influence each other

Most games today only support this form on interaction via a simple chat line, where the player has an option for colored text and the use of smilies [3]. Some games have gone beyond this level of freeform interaction by means of social engineering: the socialization emerged as part of the culture rather than the game itself. Some early MUDs showed this by creating societies of storytelling and human interaction. Sims Online is another where meeting and interacting with other avatars emerged as a side plot to developing your avatar (which is the ultimate goal of the game). XBox Live has taken the chat line a step further, implementing voice support with fellow gamers over the internet.

In *Social Gaming Interaction, Part Three: Cooperation & Freeform*[4], Appelcine list some of the many purposes of social interaction:

Boredom Relief - Players are tired of the game mechanics, and just looking for something to do.

Attention Getter - Players want to get attention from other players.

Ice Breaker - Players are looking for ways to make social interaction less threatening.

Creative Pursuits - Players coming together to create stories, pictures, or whatever else can be created in the game.

Hierarchy Building - Players developing their own hierarchy by latching onto something within your gameworld to use as a token of control.

Freeform Competition - Players creating competitions that are totally unsupported by the game system.

[3] Smilies: abbreviation or icon used to indicate an emotion or attitude

When the support for freeform interaction comes up short in a game, players usually find means through other 3rd party programs to, some extent, help cover the hole. These programs are also used for socializing with fellow players outside the game. Some of the most used categories of 3rd party programs used are:

Voice chat - Voice chat programs help alleviate text chat in intensive games. In some cases it is also needed, for example if a player has a real life handicap. Some of the most commonly used voice chat programs are: **Teamspeak, Ventrilo, Skype**.

Chat line - It's rare these days that social games does not support some form of chat line interaction, but most do not support interaction outside of the game. In such circumstances other programs are often used to socialize with game friends. Everquest[4] did for a time support an MSN style 3rd party program to interact with players inside the game. Some of the most commonly used chat line programs are: **Irc, Skype, Microsoft's Msn**

Video - With video interaction we mean both video conference (WebCam) as well as capturing live gameplay for sharing with fellow members. We have found no game today that so far supports ingame video interaction. Some of the most commonly used video programs are: **Fraps, Skype, Microsoft's Msn**

Community sites - A lot of todays online games see the creation of guilds[5] or clans of players with a common goal or interest. A few of these games offer some sort of administering these communities, but most fall short of what is needed. As such, a lot of web based communities have sprung up to fill this need. In these communities the players keep in touch with everything from game strategies to real life happenings to sharing of creative pursuits from the game itself. Some of the most commonly used community programs are: **PhPNuke, Geelix** (see Figure 11.3, NTNU phd. project)

Freeform Social Gaming Improvements

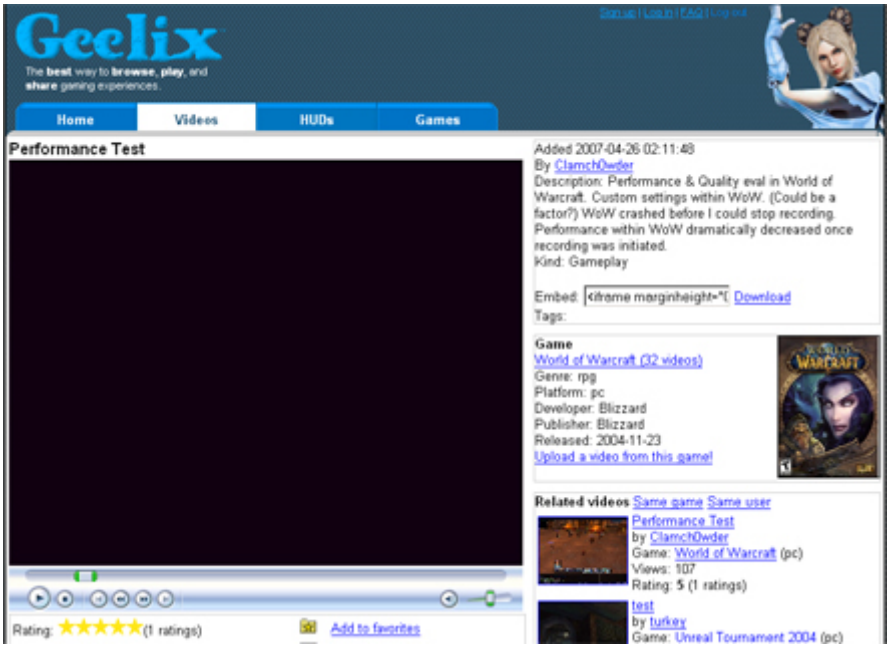
The status for social interaction in todays games is mostly limited to chat line communication. A few games have gone a little further, giving some options for voice chat, guild creations and to some extent story sharing. Coming games need to nurture this form of interaction to a bigger extent if the video games are to evolve to the next state. The first logical and easy step would be to implement the functionality already provided by 3rd party programs already in use today. To cover the other steps we need to think outside the box when we design games to find new ways to nurture social interaction. Appelcine[4] suggest 3 ways this could be done:

Build Good Socialization Hopefully, this is obvious. You need socialization that is more evocative than just a chat line if you want to encourage freeform social interaction.

[4]Everquest is considered to be the game that put the MMORPG genre on the gaming map with up to 450,000 subscribers at the end of 2004[60]

[5]Guild: an organization of persons with related interests, goals, etc., esp. one formed for mutual aid or protection.

(a) Voice chat while gaming



(b) Geelix community site

Figure 11.3: Various 3rd party programs

Consider, is your system robust enough to allow players to freeform Spin the Bottle? How about Pictionary? How about Charades? Each of these games provides a step-up from the one before it in system capability, and thus each one opens up whole new venues of socialization.

Build Good Hooks If your system has great hooks in it – neat objects, clever scripts, intelligent NPCs, etc – it is more likely that players will be able to build freeform social interaction around those hooks.

Support Support those things which players are already doing. If they have a neat little create-a-poem-based-on-a-random-topic game why not give them new hooks to: decide a random topic; write something in poetic form; and output a pre-typed poem? Players may or may not use these hooks, but if they do they will probably appreciate your work and be that much better equipped to freeform.

11.2.2 Competitive Socialization

Competitive socialization is often regarded as the player vs player method, or for short, PvP. In the early days of multiplaying, *Killers* nearly ruined games due lack of vision by the creators on how far players would go. Diablo online and Ultima Online were both plagued by cheating *Killers*. In recent years the developers have learned from these mistakes, building games that are more or less cheat free, and games that restrict player competition to specific areas of the gameworld. In [4], Appelcine lists three methods that can potentially make direct competition seem less threatening to players.

Increasing the Consent: Make sure there is a mutual agreement for direct competition. This can either be with having both players agreeing to fight, like in a duel, or design restricted areas set off for the sole purpose of player competition.

Lowering the Consequences: Not every death has to end in the death of a player character. Loosing the competition could result in other consequences as lost face, lost honor, lost funds or lost station.

Changing the Form: Direct competition does not have to be about fighting. Developers can find other ways of competing like for instance orating, negotiation or even running.

As noted earlier, competitive socialization is usually associated direct competition, which we have just described, but this does not have to be the case. The main excitement behind competitive socialization comes from risk and reward. Appelcine looks at other ways to achieve this.

Resource Competition

Competing for limited resources has been a part of multiplayer games for some time now. The first genres to embrace this type of competition in a multiplayer setting were TBS[6] and RTS[7]

[6]TBS: Turned Based Strategy
[7]RTS: Real Time Strategy

games. In *Social Gaming Interaction, Part Two: Competition*[5], Applecine divides this type of competition into two types, *Resource Collection* and *Penalty Avoidance*.

Resource Collection is usually revolved around the control of lucrative resource spawn points in todays games. This is a very limited utilization of this type, as in ie real life, there is competition for jobs, money, women, men, houses etc. Developers need to look at designing games that build upon these possibilities rather then around them.

The Command & Conquer series is a good example of RTS games that utilize resource collection. In this game, certain areas in the game spawn minerals which must be harvested in order to build buildings and units so you can wage war on your opponent. On the other hand this also shows some of the shortcomings in not utilizing the whole specter of resource collection in the RTS genre (most of the games follow the same suit).

Civilization on the other hand, a TBS game, uses much more of the resource collection specter. The world map has limited space in which to build cities and all parts of the map give different resources. One needs to gather resources to build units and more cities, do research etc. Full multiplayer options became supported with the release of Civilization IV[59].

(a) Borders between players (b) Land resources and cities

Figure 11.4: Various screenshots from Civilization 4

Appelcine gives a short guideline to how online games could better adopt resource competition.

1. Give players limited resources.
2. Allow each player to own more of a resource than his average share.
3. Provide a reward for the acquisition of a resource to ensure its collection.
4. Ensure that resource competition largely occurs between like-powered players.
5. Ensure that resource competition occurs entirely between active players.

The first three rules pretty much speak for themselves. One could, for example, make land a resource in the game and give each player the opportunity to claim it as their own. This could be for the benefit of building their own houses/towns, gather resources, set up shops to sell their goods etc. Games like Civilization, Ultima Online and Dark Age of Camelot have all used this to some extent. Thus we have an innate competition for the land. One problem, which was seen in Ultima Online and Dark Age of Camelot though was that the acquisition of land quickly was taken over by more hardcore players giving an uneven distribution of resources. This leads us to point 4 and 5. We need to ensure that new and slower players have a chance to participate in the resource acquisition. Ultima Online and Dark Age of Camelot both solved this by introducing cost in maintaining ownership over the land, and failure to pay would result in opening the land for new players again.

(a) Housement placement on land (b) Finished house placed (c) Furniture for player housing

Figure 11.5: Various screenshots from Ultima Online

The solutions seen so far to some of these issues have just scratched at the surface of dynamic resource competition. It is a manageable problem though, and the only limitation is the imagination in designing new games.

Penalty Avoidance is the opposite of collecting resources. Here the goal is to avoid penalties, a sort of anti-resource competition. This is not often used in todays games, and when used its mostly in junction with resource gathering.

Economic Competition

In this type of interaction, money transition acts as the main agent for competition. Most games develop some sort of currency when using this type of competition, but any sort of value token could be used. A lot of games utilize this, but as Appelcine notes as a problem, most fall under the problem of the economics running out of proportion after a while. A first step in making new types of this competition should therefor revolve around finding a new type of token which is more generally prized by the players.

Capitalistic Competition This type shows economic competition in its rawest form, centering on supply vs demand. In theory, this type will show itself in any game, as players

will figure out how to trade valuations in some form or another. This does not mean, however, that it should be ignored by developers. Developers can help nurture this type of competition by for example have NPCs[8] set a standard for prices. At higher level it is important to restrict item flow in the game, as nothing will destroy an economic system faster then hyperinflation of items (which is a big problem in todays MMORPGs[9]).

Auctions In this system, players can make scarce goods available through an auction system. World of Warcraft is one of the few online games to incorporate this type of system into their game (see Figure 11.6). Auction systems aren't necessarily highest bidder takes all, and Appelcine [5] brings up a few examples from tabletop games to show how it can be done.

Voting Here, each player gets a single vote or a number of votes based on some count. The player then gets to use the vote(s) to change the "rules" of the game. Voting in todays games are usually fairly simple, usually centered around a choice on what the majority of players want to do. This could be given much more depth, where players could be given the option to change rules within the game where he or she could potentially improve or hurt the other players. As with the auction competitions, voting competitions gain interest through restrictions.

Bids In this type of competition, the players place a bid on a specific outcome and thus odds change for other players placing wagers on the competition. It is perhaps the weakest type of economic competition, but could be made more direct by allowing players to, in some way, influence the outcome of the competition.

Bluffing about Competition

One adjunct that can be added to any other form of competition is bluffing. Bluffing is based on the players having some sort of limited information which other players do not know about. The player with this information must however be carefully in his playstyle so as not to let the other players onto his hidden knowledge. Poker is perhaps the best example on the use of bluffing in a game, and online poker has had a big success since its arrival to the marked. Bluffing can make *any* form of competition richer.

11.2.3 Cooperative Socialization

In cooperative socialization, the players work together against the system, and these players can be seen as a subset of Bartle's *Socializers* (see Section 11.1.4). This has become a central role in todays MMORPGs. These games have put some work into trying to make grouping seem natural via the use of group chats, automatic division of experience and items and tools that support finding players that want to do the same as you. So far though, most cooperative socialization has been about combat, and it does not have to be. Asheron's Call [58] tried to

[8]NPC: Non-Player Character
[9]MMORPG: Massively Multiplayer Online Roleplaying Game

Figure 11.6: Auction interface in World of Warcraft

take this one step further with its ideas for feudal structures, but didn't quite make it. Appelcine divides this type of interaction into three types.

Direct Cooperation This is the most typical form of cooperative interaction. The game provides some form of opponent in which the players directly cooperate in order to defeat it. The encounter is designed such that no single player can achieve this. The later MMORPGs are fairly sophisticated in this, and use this actively to make players interact with one another. Some encounters require the use of up to 40 people working together, and its through this type of grouping that large communities have formed around these games. In addition to this, the games usually support different character types that are able to provide very different resources to a grouping where several of the different characters are necessary in order to overcome the encounter.

Hierarchical Cooperation While hierarchies seem to develop quite naturally in the real world, the virtual world does not seem to share this phenomenon. Asheron's Call is one game that has tried to overcome this, but due to how it was implemented it has not quite worked out (social interaction seems to become undermined by the game mechanics of gaining experience). Some MUDs have tried to approach this differently, making it the core of a skill-teaching system, and thus forcing less skilled practitioners to look up to their more skilled brethren. Another way could be to implement the hierarchical system more directly, with top players (masters) to the worst players (minions). This could produce a lifelike feudal system, where you for example could end up with the desire for middle-class socials to overthrow the masters (to become masters themselves), offset by the fear of becoming minions themselves.

Supportive Cooperation This is a more indirect form of cooperation, where players offer resources to each other in order to achieve goals they desire. This is perhaps most visible in todays RTS games, where one player can send troops in battle to help out another player and as such have a greater chance of overcoming a third opponent. The use of this so far has been fairly one-dimensional (combat games), but it could be just as easily incorporated economic and social activities as well. The main point is that it tends to be a few steps removed from the game, and there tends to be low risk for the supporter.

11.3 Video Games Effect on Socialization

So far we have only looked at how playing styles and game attributes can help towards socialization within a game. We have not looked at the effects video gaming has on real life socialization. The traditional view on video games is that it is primarily the activity of nerds[10] and as such the subject of much mockery. Later research however seems to indicate this trend is changing.

A study done by Pew Internet & American Life Project [26] looked into one segment of the population, college students, to determine the impact video and online games have on their everyday life. Their study broke with the traditional view of video gaming, and showed some surprising numbers of playing trends. More then 70% reported to having played computer or video games at least once, and over 65% reported to play regularly. One of the surprising numbers weere that women seemed to play more online computer games then men (60% women vs 40% men).

(a) Friends playing together (b) People watching a video game tournament

Figure 11.7: Various screenshots from social gaming

The article believes that one reason for the shift in number of people playing or having played videogames is due to a generation shift in the population. The survey they did supported this by showing that 77% had played video games during highschool, and 69% had played since elementary school. Only 14% of the college students reported that online gaming is the game

[10]Nerd: an intelligent but single-minded person obsessed with a nonsocial hobby or pursuit

format they played the most. But the continuing saturation of wireless technology (cell phones, digital assistants etc) with gaming capabilities, along with the availability in the future, will allow college students to maintain and even increase their online gaming activities once they leave the college environment.

Next, the research looked at where the students played games. An observation of the campus computerlabs showed that students often had quick action Internet games up alongside their work, or popped in between classes for the same type of games. This indicates that students use these games as a pasttime or a quick distraction from work. When asked where they played games the most, the indication was that that they tend to make their home the primary gaming environment.

When asked about how gaming affected them socially, most students reported postive feelings towards gaming. The students cited that gaming was a way to spend more time with their friends, and one of every five felt moderately or strongly that gaming helped them make new friends (as well as improve existing friendships). When asked if gaming took away time they might spend with friends and family, two-thirds responded that gaming had little to no influence in that regard. Gaming also seemed to play a surrogate role for some gamers when friends were unavailable.

A study into Ultima Online[61] by Kolo and Baur[7] found that players (at least of this genre) often took friendships made ingame into the real world as well. Often with the use of chatprograms like ICQ, Messenger, Irc etc. Some guilds/clans also arrange real life gatherings where people can meet and socialize to make more lasting friendships. In some cases people cross country borders to attend these gatherings.

Steve Jones concluded in his report [26] that gaming seemed less a solitary activity for college students, and more one that is shared with friends and others. Increased adoption of ”always on” broadband technologies and Internet enabled devices will likely further contribute to the interactive uses of gaming and entertainment in the lives of college students.

11.4 MOOSES and Social Gaming

The look into social gaming in this chapter has given us valuable information on how to make sure MOOSES uses its strength as a social activity. The design of a game will greatly influence what players it will attract, and those in turn will affect one another. The MOOSES concept is based mostly on fast pased action games (see Depthstudy [50]), and as such it will be harder to accommodate player types like Explorers and Socializers. However, Socializers should be attracted with the fact that we have all the players at one location, and as such can socialize directly with their co-players.

With MOOSES focus on social gaming, it is also important to look at how games can contribute to enhance this. We found we could divide this into three main categories. These categories and how they can be achieved are summarized in Table 11.1.

Not all of the agents are feasible for our concept however. Freeform socialization methods will be difficult to implement in games due to the nature of fast paced action and the use of controllers for the framework (see Section 8.3). We do, however, have the advantage

Type	Achieved by	Examples
Freeform socialization	Voice chat Chat line Video Community sites Non-game related social activities	Video sharing of game sessions (Geelix), Technologies that allow talking/socializing outside gamesessions (Skype, Irc, community sites etc.)
Competitive socialization	Resource competition Economic competition Bluffing about competition	Resource collection, Penalty avoidance, Capitalistic competition, Auctions, Voting, Bids
Cooperative socialization	Direct cooperation Hierarchical cooperation Supportive cooperation	Game provides an opponent the players need to cooperate to defeat, Feudal system where the minions will want to overthrow the masters, resource sharing in order to achieve personal goals

Table 11.1: Social gaming categories

of having all the players at one location, allowing them to verbally communicate as they play. Enhancing the framework with a community website would also build up the Freeform socialization feeling (which then includes video/picture sharing, non-game related activities, chatting). Competitive- and Cooperative socialization should not be to limited by the concept, but some innovative solutions must be found to fit the short game durations.

We have also seen that gaming can help contribute socially, and in some cases is used solely as a social activity amongst friends. This information only enhances our belief that the MOOSES concept has a lot of potential and that we should keep focusing on sociality when further developing MOOSES and games.

CHAPTER 12

State of the Art

We will in this chapter first look at how communities affect gaming and how they are formed. We will then look at todays multiplayer games contributes to social- and cooperative gameplay.

12.1 Gaming Communities

Gaming communities are a sub genre of the term virtual community. The term virtual community is attributed to the book of the same title by Howard Rheingold, published in 1993 [62]. Rheingolds research went into a range of computer-mediated communication and social groups using the available technologies at that time. Rheingold pointed out the potential benefits for personal psychological well-being, as well as for society at large, of belonging to such a group.

One definition of online communities, as given by Preece [44], is that an online community consists of people who interact socially as they strive to satisfy their own needs to perform social roles. The people involved have a shared purpose, an interest or need, information exchange, or service that provides a reason for the community. A community has policies, rituals, protocols and laws that guide the people's interactions.

Player-to player interaction has a huge effect on a player's gaming experience [16], and one of the most prominent contributers to this are the gaming communities. Koivisto claims [27] that a game's community might actually be the most important reason for a player to stay in the game. Looking over the World of Warcraft forums might lay some legitimacy to this with quite a few bulletin board postings popping up regarding the issue. One player says:

> "In WoW, the community is the core of the game. It is why so many of us that stopped playing came back, and why the other ones are still playing. It is in fact the main reason why WoW isn't dead yet and why it still has so many customers."

This is also supported by Jakobsson and Taylors [24] research into Everquest, where they argue that social networks form a powerful component of the gameplay and the gaming experience.

Communities, however, does not exist without communication. Game mechanics affects how important it is for the players to co-operate and compete with others and how useful it is to form different kinds of sub-communities [27].

12.1.1 Game Design to Support Communities

With the importance of communities in multiplayer games increasing, the pressure is on the developers to incorporate ways to support them into their game design. We can divide design issues into five parts:

- ➢ Communication
- ➢ Player created content
- ➢ In-game personalization
- ➢ Sub-community support
- ➢ Game settings

Communication

As noted above, communities does not exist without some kind of communication. The more often a player can contact other players, the more likely he/she is to actively contribute to the game's social framework [16].

Communication can be both verbal and non-verbal. We will look more at the non-verbal communication in **Player created content** and **Visual character personalization**. By changing the state of the game, the players can also communicate with each other indirectly [16].

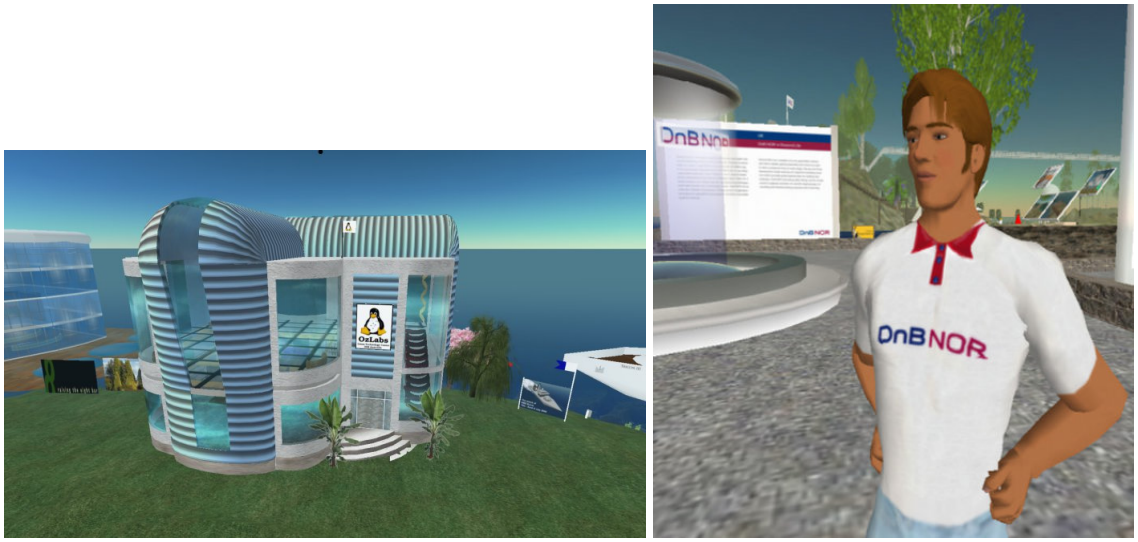
Verbal communication can either be synchronous or asynchronous [27]. Synchronous communication occurs when players communicate face-to-face, while asynchronous is when one player is unreachable (ie when trying to contact a player which is offline through a message board or leaving an ingame note). The verbal communication can be one-to-one, one-to-many or many-to-many, and can occur where players are in the same or in different locations.

We will look more at how todays games support communication in Section 12.2.

Player Created Content

Player created content lets the player leave his or hers impressions on the game world. As noted earlier this can also be seen as a non-verbal form of communication (i.e. by letting the player leave notes or books that stay in the game world after they log off). Player content is mostly used by letting the player place houses in the game world or create wearable and usable items

for other players. This give players a location to socialize and a way to share items which they share a connection to. Most MORPG[1] games today support this in some form or another, usally through the creation of wearable items for players. Second Life is a game that has embraced this to the fullest, letting players design their own houses as well (see Figure 12.1(a)). This has gone so far as to actually giving jobs to people designing virtual homes which are sold for real money. Sweden has even opened an embassy and the Norwegian bank DnBNOR has opened a branch office in this virtual world (see Figure 12.1(b)).



(a) House design in Second Life (b) DnBNOR joins Second Life

Figure 12.1: Player created content in the MMORPG Second Life

We will look more at how todays games support player created content in Section 12.2.

In-game Personalization

In-game personalization concerns the players digital self. It can be split into two major groups:

- ➢ Character design
- ➢ Visual character personalization

Character design concerns the abilities of the players avatar in the game world. This can again be split into characters with symmetric or asymmetric abilities. Asymmetric abilities make it more important for players to team up, as each player brings unique skills to the table. In some cases it might even be impossible to accomplish tasks without combining the right set of skills (players). A world inhabited by all types of players, (see Section 11.1), in balance is more likely to produce a sense of community [6].

Visual character personalization concerns the visual perception of the avatar in the game world. The simplest form of this is just giving the avatar a name. The sense of community has a

[1] MORP: Multiplayer Online Role Playing Games

tendency to grow stronger when players have the option to form their own social identities. Some games go as far as giving players the option to fully customize their appearance (see Figure 12.2). This let the player carry a message about the character's role or position within the game or about the player himself in real life.

(a) Character customization

(b) Variation in character personalization

Figure 12.2: Character personalization from Vanguard MMORPG

We will look more at how todays games support in-game personalization in Section 12.2.

Sub-community Support

The most common form of sub-communities are player-run organizations. There are just a few of todays online games that have mechanisms to support this natively, and most of those that do are in the genre MMORPG. Interviews with players from this genre asserts that these player-run organizations give the members a feeling of belonging and guarantees instant friends. It is also important to implement connectors that support these player-run organizations [27]. That means giving tools to those who run the organizations, which help them administer and entertain these communities so they stay alive.

Temporary teams cannot necessarily be defined as communities, because of their brief duration, but they are often the stepping stone to social interaction. Many players end up joining player-run organizations after having grouped with previous members (or they start one up). These teams are usually started by players who have a common short term goal (while communities have long term goals). Temporary teams usually share rewards gained from the outcome of the joint venture. The use of asymmetric abilities in character design usually encourage players to make temporary teams.

We will look more at how todays games support sub-communities in Section 12.2.

Game Settings

When we look at more direct game design like how the player can play and perceive the game, we find that this can also have an impact on communities and how they form. We can divide this into five categories:

Game world is the virtual world the player is put in. By designing different locations with different purposes, the developers can ensure that players with common goals will gather at the specified areas. This supports socialization among the players, and thereby the creation of communities.

Game story give the players something to interact with and around. If the story involves some form of conflict and factions, it will give the players a more meaningful tie between those belonging to the same faction. Working for a common goal is important in creating a sense of community [1].

Hidden information is game options the players have to figure out by playing the game, perhaps in untraditional ways. By leaving intentional holes in the game design, the developers give the players reason to socialize by sharing strategies, maps etc [16]. However, unforeseen holes in the design can often lead to exploiting which has a very negative impact on the game community as a whole.

Items and crafting can make the economy in the game more interesting and encourages co-operating [27]. Adding the factor of harvesting raw materials to use for crafting will encourage teamwork, as well as when players need to depend on one another to make finished items. The raw materials could also be located in a dangerous area, requiring protection for the harvesting players. This also support other playing styles apart from just fighting.

Learning curve also plays a role for gaming communities. If a game is easy to learn, the chances are that a new player is more likely to keep playing it and the game is able to keep its critical mass of players. On the other hand, the game can not be easy all the way through, or it will loose the veteran players more easily. Ideally the game is easy to learn, but difficult to master. This has a side effect as well, where the newer players will look to the older players for advice, and thereby creating communities for learning the game.

We will look more at how todays games settings support communities in Section 12.2.

12.1.2 Communities and Development Direction of Games

While gaming communities influence the players and their gaming, they can also have the reversed effect of affecting the development direction of games. Communities can be a valuable input when developers are looking at what direction their game needs to take. Developers are interested in this information in two different stages of development:

- ➢ Beta testing in todays games rely on a community of players to give feedback on the game design and bugs
- ➢ Bug fixes and development of new content after game release are dependant on player feedback

However, the feedback from players is not always a good thing. The active people in communities the developers get feedback from are not always representative for the whole game community, and as such any changes might hurt more then it does good. It is therefore important the developers also have a clear vision of their game and try to hold to that as much as possible. That way the majority of the players and fan base will know what to expect from the game and keep what they love about it. Communities also give good as well as bad publicity, and as such has a certain ”hold” on the developers on dictating the direction the game should take. This is most critical in beta- and early stages of game release where fan sites have an influence on more casual players interest in the game.

The feedback from players can be given in various ways, but the most used tools today are:

- ➢ User interface inside the game client which players can use to submit bug reports or game improvement suggestions
- ➢ Game supported website with bulletin boards where players can post information
- ➢ Fan based community websites with bulletin boards where players can post information
- ➢ Live chats between developers and players

12.1.3 Game Community Software

Most gaming community software today are based on standard Internet community software, modified to fit the specific community or game design needs. There are some communal traits for the different software used.

- Distribute news, articles and blogs
- Forums
- User submitted comments and discussions
- Photo and file galleries
- Polls
- Emails and private messages
- Personal profiles

One of the common modifications is the setup of hierarchies for members of the community, which is a factor in social gaming (see Section 11.2.1).

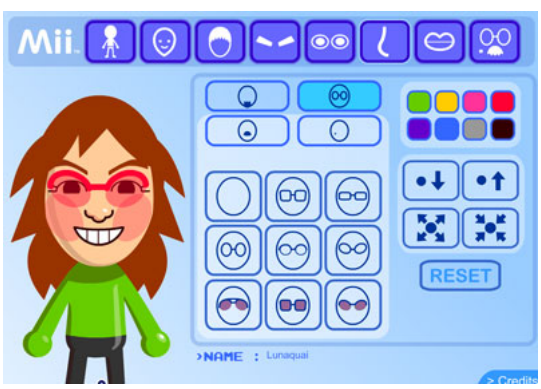
Many of these items draw similarities from what is important for communities inside the games, as discussed earlier in this Section (news and forums for communication, personal profiles for personilization etc).

Nintendo, Microsoft and Sony have implemented community based software with their gaming console platforms (see Figure 12.3). Microsoft with their XBox-live system, Nintendo with their Mii system and Sony with their "Home" system. Microsofts software uses a simple avatar and personal registration system, but has an enchanced system that allows for downloads of games, demos and other entertainment as video. Communication with other players and friends are done either through the software mail service or through built in voice chat. Mii on the other hand has a detailed avatar creation system. This avatar can then be used in supported games and on the online forums. The Miis (as they are called) can be shared between friends and stored on the console's controller. Sony's Home is a merge between Microsofts XBox Live and the MMORPG Second Life. They have created a virtual space where the players can move around with their avatar. Each player has their own apartment which they can furnish and invite players into. Joint recreational areas stream live video and has activites players can do together. They can also commuicate together via text boxes, gestures and audio.

12.1.4 Future and our Concept

We have seen that communites are important for social games, and that where the game infrastructre lacks the tools to fully support this the players turn to 3rd party community software to fill the gap. Games seem to inherit more and more functionality taken from these softwares rather then the other way around, so looking at the future of these softwares will give insight to where gaming communities should move next. Myspace and Facebook (two of the most popular social networking sites [25]) are both developing mobile clients so that the users can access the community anywhere they want with a mobile phone. An interview with an Electronic Arts executive shows that there is already some work in the field for games as well [32].

With our concept, to a large degree relying on mobile phones, a combination of community site and mobile phone access seems to be the next logical addition to the framework. With a



(a) Mii avatar customization

(b) Xbox live user interface

(c) Sony's Home space

Figure 12.3: Screenshots of a)Nintendo's, b)Microsoft's and c)Sony's community software

mobile community client, we would be able to make the concept more pervasive by sending notifications when people in the same community are close to eachother in real life. That way they can more easily join up for a quick game at the closest MOOSES server.

12.2 Multiplayer Games vs Social Gaming

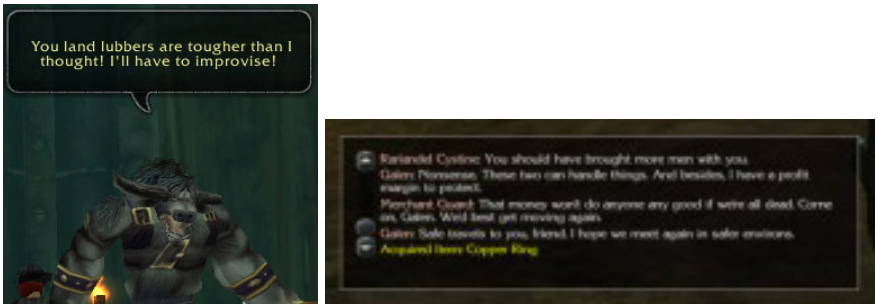
In this section we will look at how multiplayer games contribute to enhance the socialization within their games. We will evaluate the most popular genres used in multiplayer gaming. Each genre will be evaluated on the following criterias: Communication, Personalization and player created content, Visual character personalization, Sub-community support, Game settings, Cooperative options

12.2.1 Role Playing Games

Role Playing Games have their multiplayer roots from the early games of MUDs. These later evolved to todays graphical RPGs, with variations in multiplayer options. Some games only support playing the game storyline as a group, while others, (like the MMORPGs), support more player freedom in a persistant world. In this genre we have looked at the following games: **Neverwinter Nights, Ultima Online, World of Warcraft and Second Life**

Communication

Communication in these games are mostly done through chatlines. Players are presented with an interface which contains a simple chatwindow and chatline. The chatline can then be used to enter commands and player broadcasts, and players in the visinity recieve the broadcasts in their chatwindow. Some games also support chat bubbles, where the broadcasts then are displayer over the players avatar in the game (see Figure 12.4). The latest games also support voice chat via microphones.



(a) Chat bubbles in World of Warcraft

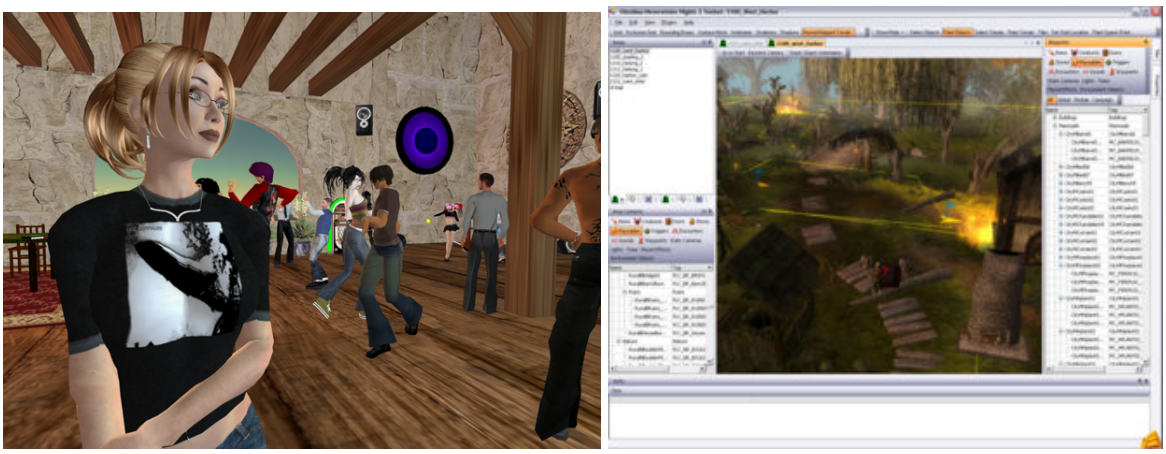
(b) Chat window in Neverwinter Nights 2

Figure 12.4: Communication methods in RPGs

Personalization and player created content

Todays roleplaying games support a variety of personalization options. Players can create or buy clothing and accessories so they can give their characters a personal style to their liking (see Figure 12.5).

Games in the RPG genre also has a good support for player created content. Neverwinter Nights is one example. It is supplied with a world editor, letting players create everything from buildings to landscapes and storylines. This has resulted in players creating persistent worlds supporting 50-60 players. Second Life, (a MMORPG game) supports players designing and creating their own buildings which they can place in the perstitant world (see Figure 12.5).



(a) Personalization through clothing and accessories in Second Life

(b) World editor for Neverwinter Nights

Figure 12.5: Personalization and player created content in RPGs

Visual character personalization

Gameplay in roleplaying games tend to center around the player playing a hero through a story line. In early games, this hero was usally preset, both in skill traits and visual traits. Todays games tend to favour giving the player alot of freedom cusomizing his or her character.

When starting a game, the player is usally presented with a character creating screen, where they can customize the look of their avatar in the game with everything from heigh, body build, hairstyle and facial traits. The user is also prompted to select a profession which gives him or her a preset of base skills the avatar will know. These are usally modifiable and the player will add on these skills as he or she plays the game. MMORPGs make good use of asymetric professions to make the players team up to accomplish certain tasks (see Section 12.1.1).

Finally the player gives the avatar a name which is unique for the game, so he or she can distinguish themselves from other players and more easily make and uphold ingame relations.

Sub-community support

The RPG games we looked at all use temporary teams as a means of progressing throught the games, and this forms a basis for socialization and sub-communities within the games. On top of this the games also support the creationg of guilds. The guilds support a hierarchy with a leader on top, supported by players with administration rights (adding new members, hosting ingame events for the guild, member support) and normal members. Some games support the creation of more hierarchy roles in the game, but these have no real game functionality other than helping administrators run the guild more easily (fex. differentiate between full members and applicants etc.). Guilds are also supplied with their own chat channel for easier communictaion between members and help support the feeling of belonging to a community between them.

Game settings

We will here look at how certain game settings in the RPG genre can help enhance the socialization in the gameplay.

Game world in RPG games, players are usally placed in designed starting areas when they first enter the game. As they progress, the game leads them ”naturally” into new and progressivly more difficult areas. This progression ensures that players with common goals are found in the same areas. The games also have designated areas for freeform socialization activities where players with common interests can meet up.

Game story in the RPG genre is centered around the classic good vs. evil. It can contain multiple factions on either side as well as neutral factions, but the main story line is such that the player has to choose sides and co-operate with players on his or her side to battle the other. One exception to this is Second Life. This game is centered more around freeform socialization and players are more likely to gather around activities associated with that form of socialization then the story of the game.

Hidden information can be found in various ways in this genre. Player abilities are usally only described shortly, leaving it up to the player to discover through trial and error how they may best be used. They players also have to discover how abilities fit together when grouping with other players and their professions.

The games usally ship with a minimal of maps of the game world, leaving it to the player to discover his way around.

Exploitation has become a problem in this genre, especially in the MMO genre. Players wanting to gain an edge in player vs. player combat is one reason for this. Another is player bots[2] which are used to aquire items and money which can be sold for real life money. This has become a big buisness and is currently a huge problem for the developers.

[2] Avatars which are controlled by the players computer without the player beeing present

Items and crafting is support in many ways, everything from player wearable items to desgining of virtual houses. Some of the games require the player to harvest resources found within the game in order to be able to craft.

Learning curve in these games can be pretty steep. The wast amount of options given to the player on what to do can seem pretty daunting at first. Developers have tried to counter this by giving the players tasks along the way which gradually teaches the players game mechanics and their chosen profession.

Cooperative Options

Cooperation is supported through grouping with fellow players to overcome common goals (like fex combating a joint enemy, either computer or player controlled). This can be grouping up to 40 or more players at one time. When co-operating the players share rewards given from the game (treasure, player vs. player kills etc.).

12.2.2 Real Time Strategy Games

Warcraft 1 was one of the first RTS[3] games with multiplayer options and, to a large degree, set the standard for how multiplayer is implemented today in this genre. The basic scenario of controlling one team on a battlefield has not evolved much since then. Games in this genre have an interface for creating and joining multiplayer games within the game client. Here the user selects how many players can be in a game, what game map to play and other options related to the game. In this genre we have looked at **Warcraft 3 and Command & Conquer 3**.

Communication

Communication in this genre is mostly done through simple chat lines and chat boxes. In the event of creating alliances, messages can be sent privately in the respective teams. Players can also commuicate when setting up games via the same method.

Personalization and player created content

A few of the games in the RTS genre, (like Warcraft 3 seen in Figure 12.6) are delivered with a world editor in which players can create maps to do battle on. In the editor, the designer places out resources, terrain and player starting points. Once finished, the maps can then be distributed and shared with other players.

Visual character personalization

This genre has litle support for character personalization. The player has an option for a unique name and color presentation in most cases. In some games the player also has to choose a race

[3]RTS: Real Time Strategy

Figure 12.6: Warcraft 3 map editor.

or faction to play which will present him or her with unique abilities and units to that race or faction. The units are visually unique from the other faction or races, but if players choose the same faction or race, only the chosen play color will distinguish them from other players.

Sub-community support

Direct sub-community support is not present in this genre at this time.

Game settings

We will here look at how certain game settings in the RTS genre can help enhance the socialization in the gameplay. A typical game sets up two players against one another on a set battlefield, where they have to gather resources to build buildings and units to combat and try to annihilate one another.

Game world in the RTS genre locks the users into a battlefield where the players fight for resources and try to eliminate one another. The games are usally shipped with mutliple different maps, supporting anywhere from 2 to 8 players.

Game story is presented to the player in single player mode but not multiplayer. The single player story campaigns do however help players get into the game and most players will select a race or faction based on how the liked playing them in the single player option.

Hidden information is found in the form of exploring and learnign the different maps in the game. How to best utelize your forces against the different opononents and gather resources. Exploitation is not a largely known problem in this genre.

Items and crafting is a natural part of this genre. Players need to gether resources in the game. These resoures are then used to construct buildings and units that are needed to battle your opononent.

Learning curve in this type of game favours veteran players. The games are usally easy to learn, but hard to master. Veteran players aquire strategies over time on the best building and harvesting methods and will have good knowledge of previously played maps which can give strategic advantages.

Cooperative Options

In this genre players can form alliances. This is set up whithin the multiplayer game options before the game takes place (see Figure 12.7 for an example on a network game setup screen interface). Balanced teams are usally created to ensure fairness (2vs.2, 3vs.3 etc.). Players on the same team can not attack one another.

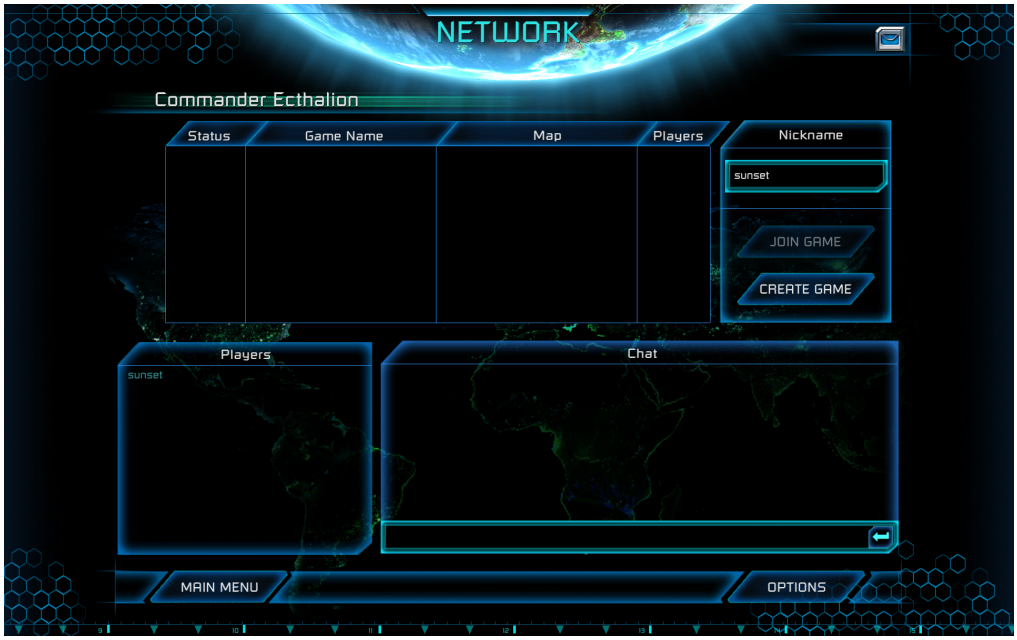


Figure 12.7: Interface for setting up network games in Command & Conquer 3

12.2.3 First Person Shooters

Multiplayer FPS games started with Doom which supported two types, death match and cooperative mode. The death match allowed 2 to 4 players battle each other out, while cooperative let 2 to 4 players gang up against the computer. The implemention by Doom is stil the basic norm today, with more variations and evolved cooperative play. In this genre we have looked at **Counter Strike, Battle Field 2 and Unreal Tournament 2004**.

Communication

Players are presented with an interface which contains a simple chatwindow and chatline. The chatline can then be used to enter commands and player broadcasts. Some games also have built in hotkeys which broadcasts chat or audio commands to the other players in the field. Voice chat is comming into this genre as well (Battlefield 2 has this implemented in the client), and lets players instantly communicate with one another.

Personalization and player created content

Most of the new games in the FPS genre, (like Battlefield 2 seen in Figure 12.8) are delivered with a world editor in which players can create maps to do battle on. In the editor, the designer places out resources, terrain and player starting points. Once finished, the maps can then be distributed and shared with other players.

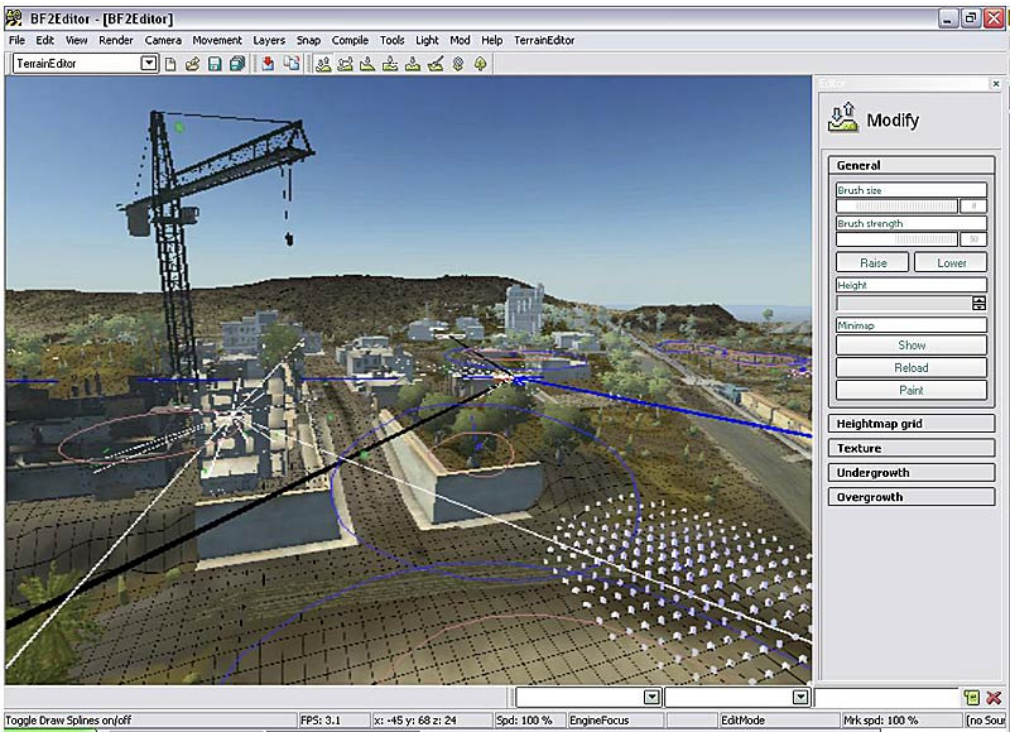


Figure 12.8: Battlefield 2 map editor.

Visual character personalization

In the FPS genre the players can choose between predefined avatars to play in the game. These are sometimes locked to certain professions within the game (eg. a sniper will look different from a normal marine soldier or terrorist). The players are also given a color scheme to help diffrentiate themselves from other players. Name-tags or nicknames are the only unique way to tell players apart however. In Battlefield 2 they have take all of this one step further, letting

players increase their rank from recruit all the way to General and unlock new weapons, medals, and more.

Sub-community support

Up until recently, this was only supported via team battles in cooperative play. Clans created outside the game would team up against eachother in cooperative battles, clan vs. clan. Later games have seen the importance of communities however and enhanced on this. Battlfield 2's community support includes buddy lists, stat tracking, live chat rooms, and in-game clan creation.

Game settings

We will here look at how certain game settings in the FPS genre can help enhance the socialization in the gameplay.

- DeathMatch
  A classic every-man-for-himself player vs. player combat. The objective is to finish the map with the most kills.

- Team DeathMatch
  Teams competing together to out-kill the opponent team.

- Capture the Flag
  Classic Capture the Flag. Players compete to capture the other team's flag and return it to their base. Competitive teams must use a great deal of teamplay. Teams must both defend the base from incoming attackers and get into the other team's base, take their flag and return to base unharmed. This requires that the team protect their flag carrier very well from enemies in order to complete their objective.

- Domination
  Teams compete to control various control points to earn points and win the map. Standard maps contain three control points. Control of these points can be accomplished either through occupation (physically occupying the space) or from a distance.

- Last Man Standing
  Similar to deathmatch, the objective here is to remain alive longer than your opponents, putting an emphasis on number of deaths rather than kills. Players have a set number of lives and once they run out of lives they lose and have to wait as spectators till match end.

- Assault
  his game type is played with two opposing teams, one assaulting a "base" and the other defending it. The map is set up with a number of objectives which the attacking team must complete (usually in sequence) such as destroying something, entering an area, triggering a button, et cetera. The team who first attacks then defends, and attempts to defend for the entire time they attacked. If they can accomplish this, they win the map.

If the team defending first assaults the base faster than the other team, they win the map. If both teams defend for the maximum amount of time the map is a tie.

Game world in the FPS genre locks the users into a battlefield where the players fight to eliminate one another or accomplish various obecjtives. The games are usally shipped with mutliple different maps, supporting anywhere from 2 to 40 players.

Game story is usually set in a designed universe by the developers (this can be anything from the past to the present or a thought future). In most games this is presented in a single player option of the game where the players go through a campgain with predisigned storyline.

Hidden information is found in the form of exploring and learnign the different maps in the game. How to best utelize your forces against the different opononents or perform the objectives are important information to learn to gain stratigic advantages. Exploitation has been known to be a large problem in this genre, and developers continualy battle this.

Items and crafting not supported by current games in this genre.

Learning curve in this type of game favours veteran players. The games are usally easy to learn, but hard to master. Veteran players aquire good knowledge of previously played maps which can give strategic advantages. These games also require good coordination skills in using input devices in order to hit other players.

Cooperative Options

Cooperation is supported through team vs. team combat. When setting up games, the interface tries to ensure that the teams are fairly balanced. More advanced comunities form community vs. community fights which lets the players co-operate better through practise and experience as they play together over time. The teams and players share points for winning games. Cooperation in this genre is also heavily influenced by game type selected (see Section 12.2.3).

12.2.4 Party gaming

Party gaming is not a normal classification of games, but one we chose to cover some of the new games arriving which are sentered more around social events between players. These games are today console based, and support from 1-8 players. In this genre we have looked at **Buzz and Gitar Hero**.

Communication

These games are created for use with consoles. Due to this fact they have limited input devices and as such does not support any ingame communication This might change as consoles evolve (XBox and Playstation 3 both support usb keyboards). As such these games only support players talking to eachother in real life at the same location.

Personalization and player created content

There is stil no party-style game released that support player created content by game design.

Visual character personalization

Some of the games in this genre allows players to choose a simple avatar pregenerated by the game. These usally have different personalities letting the players choose one they feel fits them (see Figure 12.9 to see different predefined character profiles from Buzz).

Figure 12.9: Different character profiles from Buzz

Sub-community support

Direct sub-community support is not present in this genre at this time.

Game settings

We will here look at how certain game settings in the party gaming genre can help enhance the socialization in the gameplay. The games usally revolve around collecting points on either doing coordinated actions with the controller or completing tasks.

Game world is extremely limited in these games with no world for the player to move around in.

Game story is limited, with a setting for the game to a special time area (80's, 90's etc.) or specific domain (like sports, music, etc.)

Hidden information is usally in the form of skills or knowledge the player has from real life.

Items and crafting not supported by current games in this genre.

Learning curve is very dependant on skills the player has before hand from real life. With limited experience, the player will have a steep learning curve.

Cooperative Options

Some of the games allow players to create teams of two or more players to battle eachother. The joint score of the players of one team is compared to other teams to decide a winner. The players need to choose what teams they want to be on themselves making the teams predefined. This can result in uneven teams.

12.2.5 Options For MOOSES

Communication

The best social solution would be full voice and chat line support, but this is not very viable in our concept with the use of mobile phones and quick action. One solution could be quick macros which could send predefined messages to opponents (like done in todays FPS genre, see Section 12.2.3). These could also be enhanced to include small animated pictures to make them more personalized.

Personalization and player created content

Ingame personalization and player created content is difficult in the "come and go" concept of MOOSES. However, the framework has an easy interface, (and now a scripting system for the mobilephone controller), so it should be quite feasible for players to create their own games to be used by MOOSES.

Visual character personalization

The MOOSES concept does not support a global avatar and, with the allready large variation in gameworld settings, will not be very feasible. MOOSES as a framework only support unique player names. Each game created should therefore support their own mechanics for personalizing their ingame character. Due to the instant action feeling of MOOSES this should be kept quite simple (eg. selecting the color of your avatar or have predifined characters to choose from like todays RTS or Party games).

Sub-community support

The ”come and go” concept of MOOSES also makes this difficult to implement. There is litle time for administrering communities while playing fast paced games. One option is to implement this on a higher level, in MOOSES itself. The creation of a community site (like the ones described in Section 12.1.3) could be one such option and should be given further studies on how it should be implemented. Some of the basic functionality should include: different communication options between members, sharing of player created content, hierarchies. This type of community site could also give better support for character personalization.

Game settings

The RTS, FPS and Party game genres are all somewhat in the same fast passed action genre as MOOSES, and as such their game settings suit the concept well. Resource gathering, player/team elimination (eg. all game scenarios from FPS) and point gathering are all good winning methods for a game and some are allready in use (eg. SlagMark).

Game world which is persistent and large will not work very well with MOOSES. However, games should support different maps or player areas to give variation in gameplay. The maps need to include any resources or special areas the players need in order to accomplish the game goals.

Game story might be difficult since MOOSES is solely a multiplayer concept, and as such can not support a single player option where the player is lead into the gameworld and its history. Some clues could be left in the game, and an out of game information site could be created to give the players some feeling of relation to the games.

Hidden information can be anything from aquiering game skills and knowledge to any form of skill or knowledge the players has learned in advance. Any form of cheating should be eliminated with the framework handling all commnuication and the dynamic loading of clients with scripting in real time.

Items and crafting is feasible in RTS type games which requires gathering of small number of materials and quick creation of items. Todays MMORPG style which requires a longer crafting time is not ideal for MOOSES.

Learning curve in MOOSES games should follow a line of easy to learn, but hard to master. This will let anyone who drops by to play the chance to win, but also create discussion on the best ways to master a game (which would create socialization around the games). How the game should be difficult is up to the developer and type of game. It could be either through controller coordination, information knowledge or a combination of the two.

Cooperative Options

All the games looked at supported some form of team-based interaction in order to achieve cooperation. A mix of the FPS, RTS and party game genre would probably fit MOOSES the

best. To achieve good cooperation a game would need to support teams which consists of players with different game abilities. This would encourage communication between team members in order to maxmize their potential. In order to ensure fairness the game would need to have methods to devide the players in even teams. If the developer wants to support sub-communities he or she might want to let the players choose teams, but should allways make sure the teams end up balanced in the end. Since the framework doesnt support clans or guilds (as of yet), it is not an option to the developer to devide teams as such.

Part IV

Game Concepts

CHAPTER 13

Introduction

In this part we will present new game concepts for the MOOSES framework. The concepts have been sketched out by first using a creative process of brainstorming, and then looking at present games which we have used to come up with the rough idea for a game. We have then applied the research from our depthstudy [50] and our new research into social gaming and multiplayer games to flesh out games that fit well within the overall concept.

Once we have presented the new concepts, we will pick some of these which we will refine and design further before we implement them. The choice on what games we choose are based on our focus for this report (cooperation and social games).

CHAPTER 14

Game Concepts

In our depthstudy, we made a simple worms-like multiplayer game to test the framework. The game was very fun, but to captivate the audience for a longer time, we have to make more games available for them to play. In this chapter we have come up with some game concepts that we feel are suited for MOOSES, which would be interesting to make and play in the future. First we discuss some common gameplay-variations that we can use in some common games.

14.1 Possible arcade variations

There exists lots of games that have simple and fun gameplay. There are many of these games that have some common gameplay which usually are easy to add for a programmer if the variations are kept in mind while developing a new game. These variations let the programmer reuse many aspects of a game like art, game rules sounds etc, but make interesting variations of the same game to increase the longevity of a game. The variations we have found are especially suited for 2d / semi-2d arcade games where the game happens on one screen like MOOSES.

- ➢ all against all (kill ratio) - This is usually the basic arcade style.
- ➢ team match - another variation that is popular is to divide the players into teams. This variation will usually foster the development of tactics.
- ➢ cooperative versus computer - probably hard to implement for a satisfying opponent, but can make the game playable with few participants.
- ➢ capture the flag (team/single) - players can collect flags and bring them to a base or hold them, and will receive points based upon the time they can hold the flag.

- bounty rabbit - one player is the rabbit which everyone else targets. When a player kills the rabbit he will become the new rabbit. The rabbit receives points per second and maybe some bonuses to help the rabbit stay alive longer.
- collecting flags - players receive points per flag that they get to.
- turf (collect flags) - players must collect from a limited pool of flags, bring them to their base and guard the flags in their base to get points.
- king of the hill - survivor style gameplay, will probably feel very alike arcade games. A noticeably timer is displayed over each head which counts down to zero and removes the player. When a player kills another player, the killer will get extra seconds until there is only one player left.
- speed kill - kill as many as possible in the timelimit without gaining extra time for kills.
- soccer - everyone shoots at one ball to get it into the opposites goal. Hitting the ball at different angles or with different weapons should make the ball behave according to physics. A direct hit with a missile could make the ball explode and give the opponents a goal kick.
- zombie - one player starts infected which makes him look like a zombia. The infection spreads to other players when he manages to kill another player, and those will then help the original player spread it to further players. Zombies should have a slight disadvantage so the other players can gather some points. Zombies get score per kill based upon how many are left, survivors get score per second they survive.
- VIP - one player is selected as a very important person, and the player's team has to escort him across the playing field alive for a big bonus. The other team tries to assassinate the vip. This could further be modified to include one vip per team and a timelimit.

14.2 Space Battle

SlagMark was a very fun game when we tested it for our depthstudy [50]. It is an easy game where everyone plays against everyone. This time, we will try to make a more strategical game with a focus on teamplaying.

We sought influence for strategical games on Internet, and we found and were inspired by Netrek [36]. Netrek is a real-time space battle simulation internet/lan multiplayer game where development started already in 1972. It is open-source, but it would require too many changes to make the game compatible to our concept of one big screen, to use Netrek as a basis. See 14.1 for an in-game screenshot.

Netrek is a very strategical game, where two teams have planets with armies on them. Players need to have killed another player to be allowed to pick up armies, and can then deposit the armies on another planet to defend it. Players can choose between different types of ships, including scout, destroyer, assault ship, cruiser, battleship and starbase. These types have different properties like speed, toughness and carrying space, and there can only be one starbase per team.

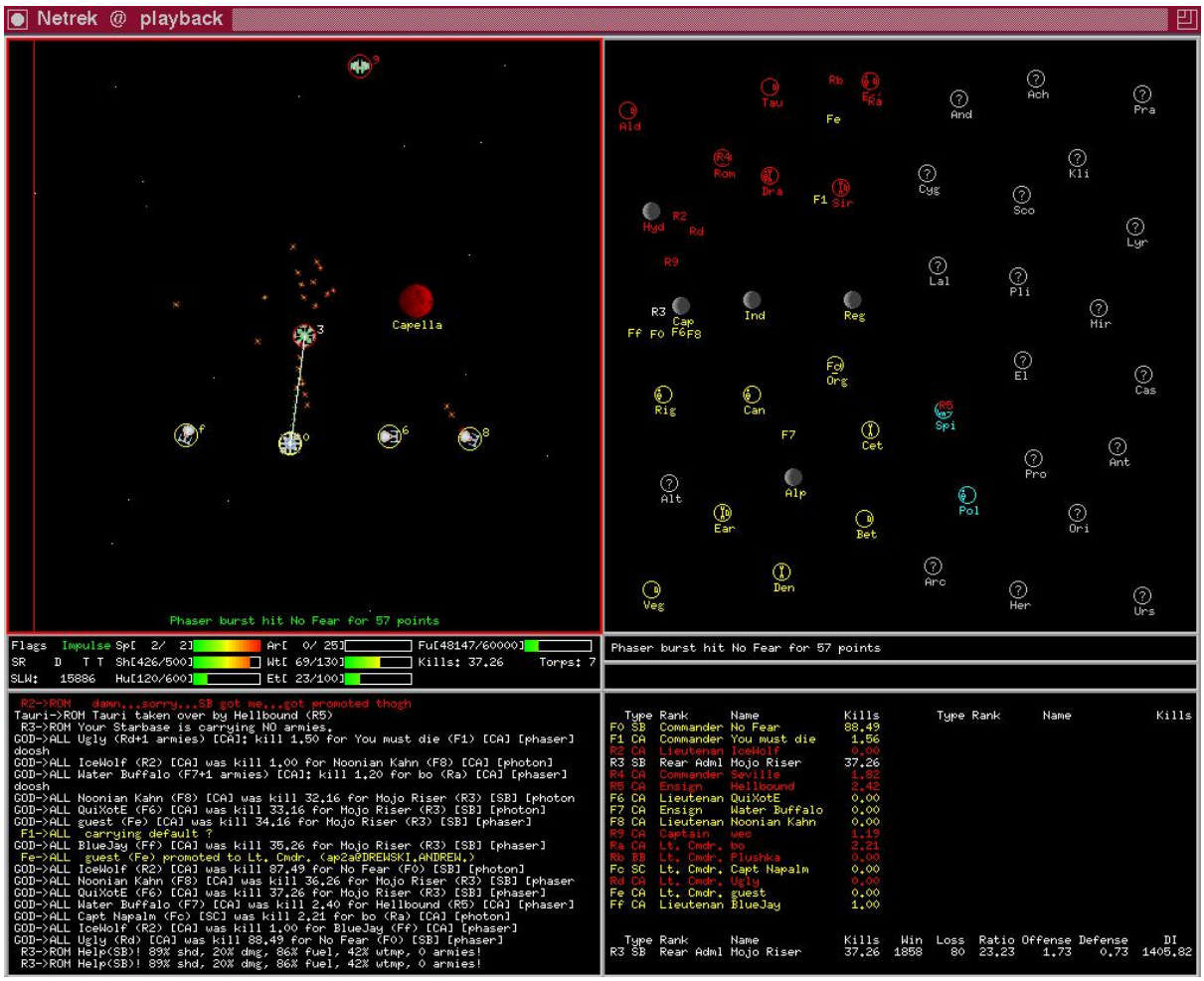


Figure 14.1: Netrek screenshot

14.2.1 Game details

The game will be a 3rd person top-down viewed 2d-battlefield in space, a multiplayer strategical thrust-based scrolling shooter. The players will be divided into 2 or more teams, where each player will be able to choose a spaceship from a list of classes. There will be one starbase, and many fighter-, scout-, and bombing-ships, where each type of ship has unique abilities. The starbase is the most important ship a team has. A team has only one commander who sits in the starbase. The commander controls building and spawning of ships, so if the commander is killed, the team can not spawn more players and the game is lost when every player on the team is killed. The starbase has not got any weapons, so the players have to guard the base.

Each teamplayer can choose the type of ship they want to spawn in, and the players will be spawned in an orderly fashion (by time of death). There is room for one player to spawn each 30 seconds. The different types of ships the players can choose from, are:

- ➢ The fighting ships are strong, slow and have heavy weaponry.
- ➢ Scouting ships are weaker, faster and have lighter weapons.

- ➢ Bombing ships are very slow and have weapons that can only hit buildings, which they are very strong against.
- ➢ Rescue ships are fast and has just simple guns. They can rescue dead players which, if succesful, reduces the cost of the dead player's spawn by a great percent.

Each team starts with a premade base, with repair- and refuel-bay. The commander and players are spawned in the team's base, and the game starts.

The game should be balanced so that a good team is a team where they adapt the number of different types of ships according to the other teams, or utilize every unique aspect of the ships strategic.

To be able to see your ship we have to make each ship have a unique color, and display the name over the ship. Which team the ships belong to have to be recognized too, maybe we will use a different ship-style or a different color for the name. When the battles get big however, ships will probably crash into each other a lot, which will affect the visibility.

14.2.2 Mobile client

Different roles will have different screens on the mobile client. The commander should be able to see the base, and order some buildings built. There should be an amount of resources which limits spawning and building.

Different screens can be:

- ➢ movement/status-screen – everyone
- ➢ building-screen – the commander
- ➢ rescue-screen for rescuing escape-pods – rescuers

Due to limited Bluetooth bandwidth we will send as little information that we can, and let the mobile handle the info.

14.2.3 Possibilities

There are many possibilities that we can do to variate and/or enhance this game. It could be possible to discuss a strategy before the game begins and mark on maps that are sent out to the clients. Another possibility is to facilitate for alliances between weaker teams. There could also be an opportunity to hire them with resources. Some variations can be to limit choosing ships at startup only, disabling respawns and give out powerups randomly at the map. Friendly fire is also a consideration. To make the game even more strategical, it could be possible to place walls and thus make strategic points in the map. Ships could also be allowed to dock at turrets, and add their firepower to the turret. This should be limited to 2 players at each turret.

We can also use other gameplay variations from the List 14.1.

14.2.4 Social solutions

This game involves several social enhancing attributes. First of all it is team based, favouring a team which communicates well and helps eachother out. By using different roles the players can/must fill, it will intensify the cooperative aspect of the game. The use of resources (economic attribute to help socilization), killing opponents and strategic areas (bases, turrets which is a limited resource social attribute) should encourage the players to communicate in order to defeat the opponent team. The game also support hierarchies with the use of a base commander role. With these options the game has very good support for both competitive (in the form of lowered consequences and resources), and cooperative socialization (in all three forms). The game only supports freeform socialization through real life talking. This game appeals mostly to the achiever- and killer type players.

14.3 Bridge builder

Another concept that would utilize teamplaying is ie a bridge-building game like BridgeIt [30] or Pontifex II, see Figure 14.2, where each team could have different roles. The finished construction could be tested real-time on the screen. The final result would be dependent of successful cooperation between the teamplayers, as everyone contributes. When the timelimit is exceeded or everyone has finished, testing of the results will start. Scenarios here could be randomly selected, examples can be how heavy train can pass the bridge, what happens if a boat crashes into the bridge etc. The bridge that survives the longest would then win.

Elements from real-time strategy games could also be included, perhaps to guard the bridge against terrorists, pay someone to sabotage materials or pay to check for faulty materials at your own bridge. This game would probably be more suited for older people, and would probably favour real-life bridge builders. This gametype could also be well suited for simulating construction of a building, natural disaster prevention or other similar scenarios. These games could also function as training and enjoyment for engineers or people interested in these fields, and can be very fun for anybody that likes to create constructions and see them tested by obscure tests.

14.3.1 Social solutions

This game enchances socialization through cooperation and resources (limited resources to build a bridge is a social economic aspect). In order to construct a solid solution, the different teams are dependant on good communication and utilization of different roles. With strategic elements of sabotage of opposing teams, the competitive socializing aspects would be even greater. This game has good support for competitive socialization (through lowered consequences and resources), but only average cooperative support (through direct cooperation). The game only supports freeform socialization through real life talking. This game appeals mostly to the achiever player type, but with the added functionality of sabbotage would be very liked by killers too.

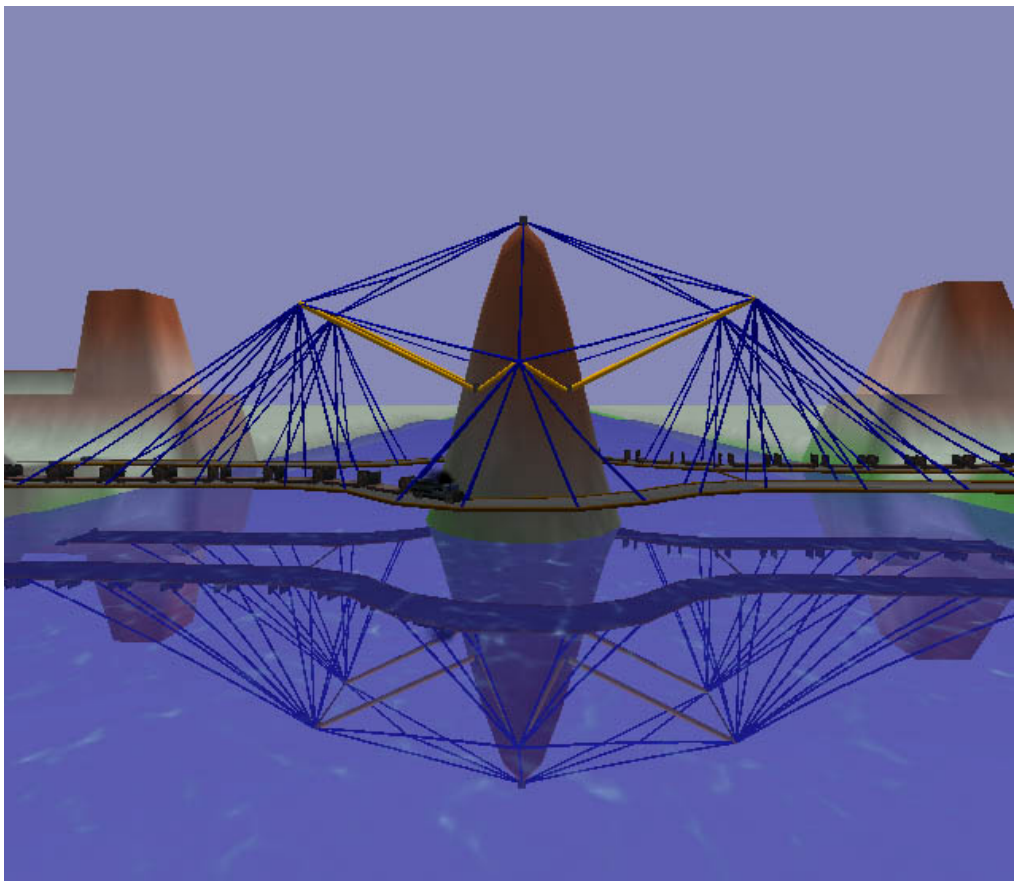


Figure 14.2: Screenshot of Pontifex 2

14.4 BandHero

A cool concept could be a band-game similar to Guitar Hero[51] and Singstar[12] in one. On the big display, we could have the notes for the different instruments in different corners of the screen and the teams will have different colors to display hits and or misses. We could have all of the instruments emulated easily by pressing buttons on the mobile. The vocal can be captured using the mobile's microphone, processed and be displayed on the big screen. If we find that the mobile is not fast enough, we can stream the vocal to the game and process it there. That would also make it possible to send it out the speakers for everyone to hear.

The screen is divided like in Guitar Hero (Figure 14.3) so that each player gets his own rectangle. In the rectangle the notes flow by on one of a set of lines like in SingStar (Figure 14.4). The player should then press and hold the corresponding button on his mobile. If the player misses the sound of his track will be muted. This makes your contribution feel more important as the entire band will hear that you missed. Finally score is collected by increasing the score every time the player holds down the correct buttons at the correct time. This, unfortunately, makes the highscore irrelevant because different instruments have different number and length of tones. So the highscore should show the team's score, or failing that show the instrument everyone played so that they can compare against other players with the same

instrument.

Figure 14.3: Screenshot of Guitar Hero

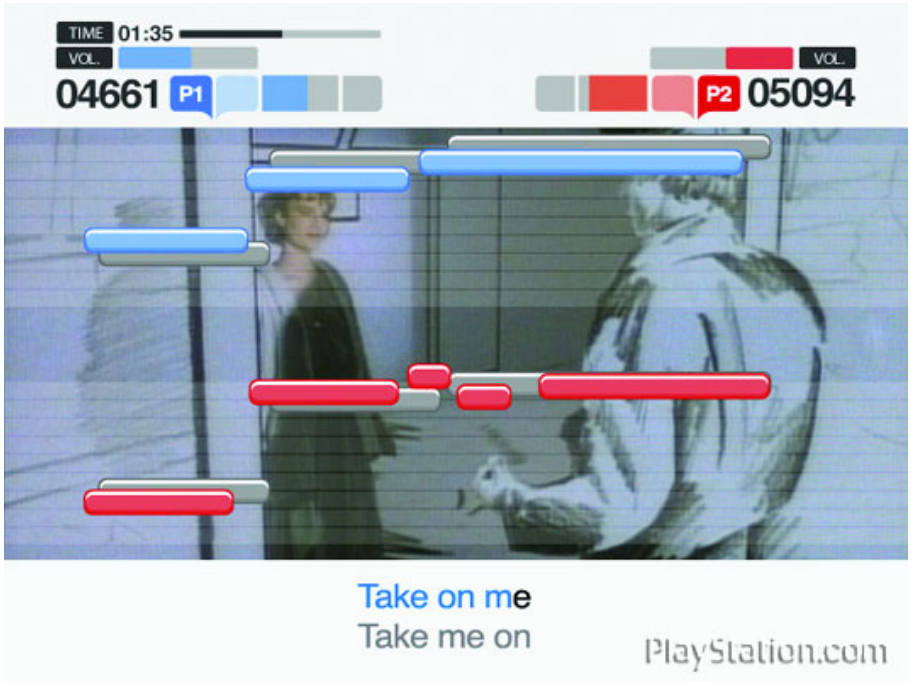


Figure 14.4: Screenshot of SingStar

14.4.1 Social solutions

This game divides a team into different band roles, where each player's performance affects the total performance of the group. As such this game only has limited cooperative support. The point system also give the game an average competitive socialization support (through lowered consequences). The game only supports freeform socialization through real life talking. This game favours the achiever player type, but socilizers will also like the cooperativeness in this type of setting.

14.5 Cops and Criminals

GTA was one of the first games to show a city from a top-down perspective. We could use the same perspective and show an overview of a city with a police headquarter. The players will be divided into teams where one team is a police squad, and one team is a criminal gang. The game would be divided into two phases, the first phase is to detect and find the criminal gang hidden somewhere on the map and the final phase is capturing or killing the criminals.

The first phase would be where the criminals go around robbing, stealing and killing. This should happen hidden on the mobile until the crime is committed, or have enough people in the city to hide the criminals amongst the crowd. The criminals will be able to buy weapons for stolen cash.

14.5.1 Roles

Criminals

- ➢ 1-2 players can be hitmen and earn money at missions.
- ➢ 3+ players can be bankrobbers and earn much money on big robberies.
- ➢ 3+ players can be thieves and earn money by robbing houses.

Police squad

- ➢ 1-2 players can be police officers in a squad car. They will be the first to arrive at a criminal scene. Medium strength. Usually hang around donut-shops.
- ➢ 3-4 players can be a SWAT team. They are good at entering buildings and have good guns, but are slower.
- ➢ 1-2 players can be a chopper team. They can provide cover outside, can't go in. One player is the pilot, the other is a shooter.
- ➢ 1-2 players can be detectives and point at probable locations. The detectives have information about the criminals, and can bribe non-player criminals for more.

Every civilian car can be used by anyone for cover or as a get-away vehicle. Final score will be based upon earned/stolen cash, the police will start with much cash and loose them as time progresses. The police will get bonuses for a successful arrestation, and a lesser one for killing of a criminal player.

A screenshot from the game Crime Life: Gang Wars 14.5 shows how this could look like.

Figure 14.5: Screenshot of Crime Life: Gang Wars

14.5.2 Social solutions

This game supports two teams, each with their own role. Communication is needed in the police squad to successfully trap down criminals, or in the criminal group to alert fellow team members of areas to avoid. The game supports direct competition through resources (stealing), and cooperative socialziation through direct- and supportive actions (police surrounding a criminal or a criminal alerting fellow team members or creating diversions). The game only supports freeform socialization through real life talking. This game crealy favours the achiever and killer player type.

14.6 SelfFish

A simple and cool concept which our supervisor came up with, is a fish-game. Experienced players will have an edge over newer ones, but not too much as we will keep the controllers

very simple. The players will have 5 buttons they can use, four buttons for steering and one for a temporary boost. The display displays a rendered view of an aquarium with fishes in it. At the game start every participating player will be able to choose their own fish which they control. When everybody has chosen their fish, the game will start. Everyone starts as a small fish which can only eat plankton. The player's mobile phones will display their fish and a status-bar which shows how much energy they have to gather before level-up. If a player is in a tight situation they can press a button to temporarily boost the swimming speed of their fish. The boost costs energy, so if they boost too much they will not gain a level. The plankton will be distributed randomly dropped from the top of the aquarium.

The first two levels are meant as a quick introduction to the game. Things are peaceful but has an element of competition as to gather energy fast to grow fast. When a fish gains a new level, the fish will noticeable grow and get new performance attributes dependent on the type of fish. After the first level, the player will be able to also eat larger insects like worms. These insects are rarer than plankton but the player gets more energy from them. At the 3rd level, the player can begin eating other players fish, but only the fishes that are one level under them. This will be a major turning point in the game as players that have gained levels slower will have to swim away from the bigger fishes. This will probably foster more cooperation between the smaller fishes as they should gather in packs to guard against predators, just like nature does.

Depending on the gamemode, when a fish is eaten it will be game over for the owner for the rest of the game, or the player will spawn as a little fish again. The game should probably not be longer than around 3-4 minutes if a player has only got one life. When the fish is eaten, a cool effect would be that the fishes leave a skeleton after them.

We searched the Internet for similar games to get inspiration from, and we found Feeding Frenzy by PopCap Games Inc [23], which is very similar to this idea (see Figures 14.6).

14.6.1 Social solutions

In "SelfFish", direct competition is supported through players eating (and thereby killing), eachother. A limited form of cooperative socialization is found in that the smaller fishes(players) have to stick together in order to defend themselves from the bigger fish. The game only supports freeform socialization through real life talking. The game also takes advantage of resources in the form of limited food to grow your fish. This game appeals mostly to achiever and killer type players.

14.7 Hightech paintball

With the availability of gps devices for use with mobile phones, we could use these to gather every players position in realtime. When accelerometers arrive, the phone can probably supply improved accuracy of the gps position, and more information like the direction the body of the player faces, if he is running, lying down etc. This information can be displayed on a big screen for the audience or on the phones for the team members. If a player spots an opponent he can alert the others by pressing a speedbutton which will send out a messages to the other teammembers with the information provided by the phone's hardware. Other buttons could

(a) Feeding Frenzy 1, eaten by a whale

(b) Feeding Frenzy 2, just eaten a smaller fish

(c) Feeding Frenzy 2

Figure 14.6: Screenshots from Feeding Frenzy 1 & 2

serve as shortcuts to commonly used soundmessages like “Help me!”, “Cover me!” or other messages prerecorded by the player.

The phone could also be used as a cheap (in the sence that many have one available) substitute for a walkie talkie. All phones today have an option for handsfree, which makes discrete communication possible by recording the input and sending it to the other players phones. The voices could be mixed together at the server so that overlapping voices can be heard, unlike walkie talkies where only one can speak at the time. Other features we could take advantage of, is to use the display of the phone to look around corners if the weapon has an integrated Bluetooth camera. This idea is stolen from the American Land Warrior project [40] where soldiers can see and shoot with their weapon around corners. Another possibility could be to change teams, gamerules and objectives as the game progresses by alerting players with vibration, sound or messages on the client. The teamleader could also use the mobile for planning on a minimap of the playing field, on which he can see spotted opponents and teamplayers.

14.7.1 Social solutions

In this game the framework is used solely for communication between team members, or as a means to scout around corners or other tight knit areas. As such, the game only supports freeform socialization through using the mobile phones for voice communication (like a walkie talkie or throught shortcut keys), or map information which a team does not want to share with the opposing team.

14.8 PopQuiz

A very popular concept that we can borrow ideas from is Buzz! [13] (Figure 14.8). This gametype can be very educational, fun to play and at the same time very competitive. Buzz! has included a new controller with the game, to make it feel more like a gameshow seen on tv. The game progresses as a typical quiz show, and has a gameshow host, audience, music and cheering to immerse the players. In the original, up to 4 players can play at the same time, and the game features many multiplayer variations to make the game last.

Our game could use much of the same basics. The client’s keys can choose question and the player gets a score based upon the number of questions answered, and or time spent to answer. An important distinction we can make from Buzz!, is that we can support as many clients that we like because information that can limit the screen estate can be at the clients.

This game could also be an opportunity to make people socialize, we could have questions that introduce the participants to each other. Questions could be about one randomly chosen person in the audience. The person would then set the correct answer and the audience has to guess or know him. It would also be possible to make the game ask everyone what color they prefer and then make everyone vote what they think is the most preferred color in the audience. A microphone situated above the players could listen for cheating or detect if the questions are making people socialize, which can also be used to provide feedback to the developers.

(a) Music question

(b) Round status

Figure 14.7: Screenshots from Buzz!

14.8.1 Social solutions

This game has very limited direct competition support through point acquisition (lowered consequences). In some implementations direct cooperation is supported in that teams have to work together on the same question or task in order to complete it successfully. The game only supports freeform socialization through real life talking. As with BandHero this game favours achievers mostly, but sociliazers will enjoy the game setting as wel.

14.9 Other utilities

As MOOSES can support a greater number of people, we can use this concept to provide lectures for a large audience where the audience can provide realtime feedback to a lecturer. The lecturer could ask which topics an audience is interested in or he/she can get feedback if the audience thinks he/she has spoken enough about a topic. It would also be possible to write instant messages to the lecturer's screen or the lecturer could make notes of the presentation available for free download by Bluetooth. A multiple choice exam would also be possible to do both locally on the client and with the use of a big screen, where the questions including multimedia could be displayed on the big screen. With this method the teacher can get live results on his monitor.

Another possible use can be to play poker (or other cardgames) versus the entire audience. This could happen in the form of online poker games where the client has a player's hidden cards and the big screen displays the common cards, whose turn it is, the current pot and who has folded etc. This could also work well in a standard livingroom which would allow for a more fast-paced cardgame because the computer shuffles instantly.

14.10 Issues

In our quest to find new games suitable for MOOSES, we have identified some issues that are common when you try to find a gaming concept that fits well.

- time for a new player to get to master the controls and logic of a new game is important. The controls should not be too difficult for an it-illiterate person to use. The gameplay should be easy to understand, close to the point of not requiring to read a help-screen while the vote times out. A text which describes the game's controllers should fit on one screen (without scrolling). Games where the player has only one life for the entire session, should provide a warmup period.
- length of a game session. A game should typically last 5-7 minutes. Games where everyone can drop in and out in the midst, can last longer than games where a player has one life or can not join midgame.
- number of players maximum supported. To raise this, the concept should not be based on a split screen but display a common level which players are placed in.
- social aspects. The games should facilitate to make players in a team communicate and cooperate to solve problems.
- assigned roles. How can the players agree on different roles in a team. Depending on their interests, socializers could for instance be helpers, explorers scout-ships, but in games where roles are limited the player's interests could be taken into account and randomly assign the categories as good as possible.
- scoring should, if possible, be based on successful cooperation. For instance if a teamplayer is under attack another player will come to help, or people can gang up in one ship and control various weapons in that same ship.

CHAPTER 15

New Prototypes

Due to limited development time on this project we were forced to select just a few game concepts for development. When selecting what game concepts to implement into prototypes, we had two criterias: Cooperation and socialization value and development time.

The following games were selected for development:

BandHero Was selected with emphasis on the current hype on this type of game as a social event. The development time was also cut down considerably with the use of open source libraries.

Space Battle Was selected due to its great support for cooperative gaming, and that it covered the most social gaming aspects of all the concepts.

SelfFish Was selected due to the its simplicity in game style and would have a quick learning curve for starters. It also seemed to be the most humour filled game which we thought would help on social interaction between the players.

PopQuiz Was selected since it had potential outside pure gaming environments (like educational quizzes) and its ease of implementation.

Part V

Improvements & Development

CHAPTER 16

Introduction

Our results from the depthstudy included a platform and results which we would like to improve on. This part gives an introduction to the platform we found, and possible improvements we identified from testing in our depthstudy.

16.1 Improvements

We identified many features that we would like to see improved in our depth-study.

Small scale improvements included:

- ➢ Removal of actors at the moment the connection is down.
- ➢ Implement security.
- ➢ Better key layout on the mobile client.
- ➢ Better graphics,more sound effects for the game and balancing the weapons.

Bigger improvements included:

- ➢ Implement client emulators to be able to use for instance usb joysticks.
- ➢ Develop a community around the games. Players should be able to compare and compete on scores, chat etc.
- ➢ Support for tournaments and competitions to promote playing.
- ➢ Implement payment option(s).

- ➢ Find and test more game concepts.
- ➢ On-demand downloadable games and updates.
- ➢ Look into what kind of gui support should be available in the library for gaming.
- ➢ Better solution for loading different game clients for the J2ME client until class loading becomes viable, or enable scripting.
- ➢ Business models.
- ➢ Scalability testing and improvement.
- ➢ Utilizing soundsystems in the cinemas better.
- ➢ Dynamic music. Change and blend music when there is much or little action.
- ➢ Cross-cinema gaming.
- ➢ Look at interoperability between different types of mobiles, find minimum requirements.

Many of these were not a good choice for our thesis, so we have chosen a few of them that we feel fit into one category to look at. Socialization is an important idea, and to test that we had to find some new concepts. To be able to support many players at once, we have to find performance issues, improve them and test if the performance is good enough.

CHAPTER 17

Framework

In our depthstudy, we were more interested to see if the concept was well received and had a viable future, than to look at performance issues. This chapter describes changes we have made to the framework to make it more suitable for gaming with many players.

17.1 Communication/Performance Improvements

Throughout the writing of this thesis, we and our supervisors have shown MOOSES at different occasions. The first testing session this year ended pretty bad, when a little bug ruined a testing session in front of many people from Telenor. This bug was caused by the fact that some phones found a similar id to the one MOOSES uses and connected to that id only. We have added a test to try to connect to more ids if the other will not allow them to connect. This worked very good on the following test-sessions.

When we began working with this thesis, we upgraded our code to work with the latest changes of the framework from Tellu. We used a new class from the newest version of actorframe; AFPropertyMsg. Unfortunately as the message is completely generic, a simple keyPressedMsg is 185 bytes. When we tested with a Bluetooth stick at version 1.2 we could notice more lag than we got with our access-point which is using Bluetooth-version 2.0. As we can see from Table 8.4, Bluetooth versions under 2.0 has to use a 5 slots packet to send this simple message, Bluetooth version 2.0 can send it using a 3 slots packet. This supports our observations that the accesspoint was much more scalable than the usb-stick, as the usb-stick was using Bluetooth version 1.2. If we reduce the packet size to fit into 1 timeslot we should be able to enjoy a similar finding again.

With some simple considerations into what data we need to send, we could reduce latency by at least 2/5 at full transmission because of the reduced timeslots. This should make older phones much faster as well and better scalability at intense transmissions. We should easily

Datatype	ID	Description
byte	Router messageID	Type of Router-message. Used by ActorFrame to differentiate between types of messages
byte	Programmers ID	Programmers chosen ID to differentiate the messages
byte	Values	Number of values included in the message
byte	Value type	0 = byte, 1 = short, 2 = int, 3 = long, 4 = float, 5 = string, 6 = booleans
value	Programmers value(s)	The values the programmer wants to send. Many chars if string
...	More values	More values in the same message as the box overhead

Table 17.1: New optimized message

be able to fit the most time-critical game-messages in 9 bytes so we can utilize the speed and errorcorrecting-benefits of the smallest DM1 packet-type. This can for instance be done by replacing the message string id with an integer id and try to use as few properties as possible.

To make this change we had to make a new type of message and make some changes to ActorFrame. Our proposed solution can be seen in Table 17.1.

With this new message we would typically use 4 bytes for a simple message with a single value which would fit into any Bluetooth packettype and ensure good performance. These byte sizes can be seen in Table 17.2. To further reduce the number of bytes necessary, we could package two smaller 4-bits variables into one byte. This should happen ie when the server sends maps, where 16 distinct colors could be enough and reduce the map's size by half.

We made each value identify itself to be able to make the script-client easier. With this syntax J2ME can parse the message and provide it simple to the script, if we had dropped the identifiers the programmer would have to make scripts to read every kind of message which would have increased the lag.

The changes we have outlined will require some changes to ActorFrame. To make us able to test this quickly, we made a new Optimized-message that inherits ActorMsg and which overrides the serialization routines. We gave the new message a new identifier in the router-system to enable it to work as a standard message and make the router-system aware that it should be treated special. This worked great, but there was one field that we could not remove easily. This field is the address of the sender. To remove this, we would have to make the routing-system aware of which actors are at which connections. At the end, some simple changes made the final message's size be 44 bytes which is good enough for testing with Bluetooth v2.0 as it should fit into both 2-DH1 and 3-DH1 packettypes (see Figure 8.4). These packet types will probably be used when the signal is good.

As we will not use much time modifying the client, we modified it quickly to send the new OptimizedMsg when the player has pressed regular buttons. This will happen without much script interaction which will give us the least computation at the client for these events, and thus the fastest possible response times.

These changes have and will improve the security, by removing the trust on the address from the received messages. Without the addresses a malicious person can not send messages as

Datatype	Meaning	Size in Bytes	Min Value	Max Value	Sig. Digits
byte	Integer	1	-128	127	
short	Integer	2	-32,768	32,767	
int	Integer	4	-2 billion	2 billion	
long	Integer	8	-9.2×10^{18}	9.2×10^{18}	
float	Floating Point	4	-3.4×10^{38}	3.4×10^{38}	7
double	Floating Point	8	-1.8×10^{308}	1.8×10^{308}	15
char	Character	2	Any Unicode character		
boolean	Boolean	1	false	true	

Table 17.2: Sizes for various primitives in Java

another actor.

17.2 Modifiability Improvements

This section describes some improvements that were necessary for an increased modifiability for external developers.

17.2.1 Plugin

We made a generic interface in our depthstudy which used class-loading to start the games available. But the games were still hardcoded into the server. Another important ability for our framework is to make the Server find the games available to it without having to modify any code. To do this we made some changes to the interface provided by the games and the framework itself.

We could have done this in a number of different ways, one method could be to use configuration files which would then have to be modified every time a new game was made available. But in the end we decided that the most flexible would be to make the GameActor find votable games in a plugin-directory by class-loading and using reflection to check that it is implementing the GameInterface. A complete listing of functions that are required by a developer to be implemented is at Table 17.1.

```
1  package gameinterface;
2
3  import java.util.List;
4
5  import no.tellu.common.javaframe.messages.AFPropertyMsg;
6
7  /**
8   * Interface for games to access the framework
9   * @author Morten Versvik
10  */
11 public interface GameInterface {
12         /**
13          * Inserts VotableGame-objects into the list.
14          */
15         abstract public void populateGames(List list);
16
17         /**
```

```
18          * The server wants to start this game with gameId from the VotableGame-
                object.
19          */
20         abstract public void startGame(Object gameId, GameServerConnection pc);
21         /**
22          * The server wants to stop the game in time seconds.
23          */
24         abstract public void stopGame(int time);
25
26
27         /**
28          * A message was received from a player.
29          */
30         abstract public void messageReceived(int player, AFPropertyMsg msg);
31
32         /**
33          * A player has joined.
34          */
35         abstract public void addPlayer(int player, String name);
36         /**
37          * A player has timed or logged out.
38          */
39         abstract public void removePlayer(int player);
40 }
```

Listing 17.1: GameInterface class.

If the plugin is of a correct type, it will be instantiated and asked for a list of variations in it. BandHero for example, has 3 songs that can be chosen between. These will be returned and added to the voting screen by the plugin. The developer of the game can provide different IDs for each of the variations, which will be sent to the start procedure if the vote stops at this game. See Listing 17.2 for the content the developer can use. For an example that illustrates how to add votable games, see Listing 17.3.

```
1 public class VotableGame {
2         String displayName, name, className, scriptName;
3         Object id;
4 }
```

Listing 17.2: VotableGame object available to the plugin developer

```
1         public void populateGames(List list) {
2         list.add(new VotableGame("BandHero - Sweet Child of Mine", "BandGame", "
              gameinterface.BandGame", "/bandhero.hcl", new Integer(0)));
3         list.add(new VotableGame("BandHero - Smells Like Teen Spirit", "BandGame", "
              gameinterface.BandGame", "/bandhero.hcl", new Integer(2)));
4         list.add(new VotableGame("BandHero - Alfs RockHalf", "BandGame", "
              gameinterface.BandGame", "/bandhero.hcl", new Integer(1)));
5 }
```

Listing 17.3: Example of VotableGame use (from BandGame)

This change makes the framework very adaptable to new games, as new games can be copied to the plugin-directory and the server will discover them and add the content to the voting-screen and highscores automatically. Likewise, it will also be easy to make the server download updates over Internet when the framework testers have accepted a new game.

17.2.2 GameServerConnection

Another aspect of communication, is from the game to the framework. Scores, movements, information etc have to be sent from the game to the plugin and then to the mobile. This object

is provided with the StartGame procedure which is ran from the framework. The developer can then use these methods for a generic conversation with the mobile client.

```
1  package gameinterface;
2
3  import java.util.ArrayList;
4  import messages.StoppedGameMsg;
5  import messages.VibrateMsg;
6  import no.tellu.common.actorframe.ActorSM;
7  import no.tellu.common.javaframe.ActorAddress;
8  import no.tellu.common.javaframe.messages.ActorMsg;
9  import actor.gameactor.GameActorSM;
10
11 /**
12  * Interface for games to access the framework
13  * @author Morten Versvik
14  */
15 public class GameServerConnection {
16         ActorSM sm;
17         public ArrayList playerAddresses;
18
19         public GameServerConnection() {
20           playerAddresses = new ArrayList();
21         }
22
23         public void setListener(ActorSM sm) {
24                 this.sm = sm;
25         }
26
27         public void vibratePlayer(int player, int duration) {
28                 ActorAddress adr = null;
29                 if (player < playerAddresses.size())
30                         adr = (ActorAddress)playerAddresses.get(player);
31
32                 if (adr != null && sm != null)
33                         sm.sendMessage(new VibrateMsg(duration), adr);
34                 else
35                         System.out.println("Couldn't find player " + player);
36           }
37
38         public void sendMessage(int player, ActorMsg msg) {
39                 ActorAddress adr = null;
40                 if (player < playerAddresses.size())
41                         adr = (ActorAddress)playerAddresses.get(player);
42
43                 if (adr != null && sm != null) {
44                         sm.sendMessage(msg, adr);
45                 }
46         }
47
48         public void updatePlayerScore(int player, int score) {
49                 ActorAddress adr = null;
50                 if (player < playerAddresses.size())
51                         adr = (ActorAddress)playerAddresses.get(player);
52
53                 GameActorSM actor = (GameActorSM)sm;
54                 actor.updatePlayerScore(adr, score);
55         }
56                 /**
57                  * Method to let the server know the game has stopped
58                  */
59                 public void stoppedGame() {
60                         sm.sendMessage(new StoppedGameMsg(), sm.getMyActorAddress())
                                ;
61                 }
62
63                 public ActorSM getStateMachine() {
64                         return sm;
65                 }
```

```
66 }
```

Listing 17.4: Available methods for the plugin developer in GameServerConnection.

We have cooperated with Sverre Morka to remove the limitations with regards to having different game-clients. This has resulted in his master thesis [35], where he has made a scripting solution tailored for our MOOSES clients. To be a completely generic interface, we send the message sent from his script directly into the developer's plugin, and let the plugin take care of it. We keep track of the player numbering, so the developer can just deal with player-numbers, which makes things much simpler if the developer wants to use another language than Java.

17.2.3 Bugfixes

We identified a problem with the highscores in one of our testing sessions which happened when players left and subsequently joined a game. The code we made in our depthstudyy which comprised these functions were buggy and not a pretty sight. We have exported all code relating to player handling from the statemachine into a new class PlayerDatabase (see Listing 17.5). This class contains all the logic used when adding and removing players, and the statemachine calls the simple functions in this class. In this process we discovered that we had forgotten to remove some test code which reordered the player numbering when a player left. Testing shows that this is working good now.

```
1  package common;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.Enumeration;
6  import java.util.Hashtable;
7
8  import no.tellu.common.javaframe.ActorAddress;
9
10 /**
11  * Class to manage removal and addition of players.
12  * @author Morten
13  *
14  */
15 public class PlayerDatabase {
16         private Hashtable adrNick = new Hashtable();
17         private ArrayList gamePlayers = new ArrayList();
18
19
20         /**
21          * Gets a live list of all the players who should join a game. This list
                will contain a null value where a player slot has been removed.
22          * @return ArrayList of players. Don't remove/add anything to this list.
23          */
24         public final ArrayList getSortedPlayers() {
25                 return gamePlayers;
26         }
27
28         /**
29          * Adds a new player to the game, checks if the address exists before.
30          * @param adr Actoraddress of the client
31          * @param nick Nickname
32          * @return Player number
33          */
34         public int addPlayer(ActorAddress adr, String nick) {
35                 if (adrNick.containsKey(adr)) return -1;
36
37                 adrNick.put(adr, nick);
```

```
38                  int nr = findNextFreePlayer();
39                  gamePlayers.add(nr, adr);
40                  return nr;
41          }
42
43          private int findNextFreePlayer() {
44                  for (int x=0; x<gamePlayers.size(); x++) {
45                          if (gamePlayers.get(x) == null)
46                                  return x;
47                  }
48                  return gamePlayers.size();
49          }
50
51          /**
52           * Gets a nick based upon the Actoraddress of the client
53           * @param adr Address of the client
54           * @return Nickname
55           */
56          public String getNick(ActorAddress adr) {
57                  return (String)adrNick.get(adr);
58          }
59
60          /**
61           * Removes an Actoraddress from the game
62           * @param adr Actoraddress of the client
63           * @return Player number
64           */
65          public int removePlayer(ActorAddress adr) {
66                  adrNick.remove(adr);
67                  int nr = gamePlayers.indexOf(adr);
68                  gamePlayers.set(nr, null);
69                  return nr;
70          }
71
72          /**
73           * Removes a player based upon a  playernumber
74           * @param nr Player number
75           */
76          public void removePlayer(int nr) {
77                  ActorAddress adr = (ActorAddress)gamePlayers.get(nr);
78                  adrNick.remove(adr);
79                  gamePlayers.set(nr, null);
80          }
81
82          /**
83           *
84           * @return Number of active players
85           */
86          public int size() {
87                  return adrNick.size();
88          }
89
90          /**
91           *
92           * @return Enumeration of all client ActorAddresses
93           */
94          public Enumeration getAdresses() {
95                  return adrNick.keys();
96          }
97
98          /**
99           *
100          * @return Collection of all nicknames
101          */
102         public Collection getNicks() {
103                 return adrNick.values();
104         }
105
106         /**
107          *
```

```
108             * @param nr Player number
109             * @return Nickname
110             */
111            public String getNick(int nr) {
112                    return (String)adrNick.get(gamePlayers.get(nr));
113            }
114
115    }
```

Listing 17.5: PlayerDatabase class which contains functions for easy removal and addition of players.

CHAPTER 18

Test games

In this chapter we will describe improvements and the development of the new games.

18.1 SlagMark

SlagMark was mainly developed to test the results we found in our depthstudy, but has been the most popular game as of today. Because of this, SlagMark has become what people think about when we mention MOOSES today. SlagMark was developed in C++ with the use of OpenGL (3d graphics library), OpenAL (3d audio library) and various other libraries for displaying text and decoding music. In this game, the main screen displays a destructible map viewed from the side (see Figure 18.1). Every player is spawned in a random location on the map, and controls one worm with various weapons available. The player can press up and down to aim the cursor located at the main screen, back and forward to move, the fire button to fire the currently selected weapon, the hash-button to select the next weapon and a button to use a jetpack. The gameplay is very simple: kill all the other worms. When a player is killed, he is automatically spawned again 10 seconds after.

We found some problems with SlagMark in our depthstudy that we have fixed, and some issues were improved in response to the performance testing at Nova Cinema. We have added a countdown-timer that limits a gaming session to 5 minutes. The timer is visible the last minute to warn the players. The weapons were tuned, a direct hit with the bullets does now do more damage than an explosion in the vicinity. To do this we modified the damage-model to include direct hits which deliveres 100% of a weapon's specified damage to the player's character. This should hopefully make players use other weapons than the bazooka.

We fixed some performance issues with SlagMark to make the game run better on older computers and older mobile phone hardware. An explosion anywhere on the map would make every player's phone vibrate and update with the same information as it had. This has been

modified to just include the players that had damage inflicted upon them, so that we reduce the burst of communication. The older phones could not deal with the number of messages that were being sent when the biggest battles were fought. Another issue was that explosions would modify the map and upload it to the graphics-card every explosion. Uploading a map means that the graphics-card has to receive 2 MB of data before it can draw another frame (2048*1024 pixels). This worked very good for 4 players and the small battles which happened there, but this was the main culprit for the game's lag seen at the Kosmorama testsession. When an alteration to the map happens now, the map will be uploaded only at specific intervals which currently are fixed at no more than 8 times per second. To reduce the overhead of uploading the entire map it is possible to divide the map up into different parts. These could then be uploaded separately which would reduce the size of an image.

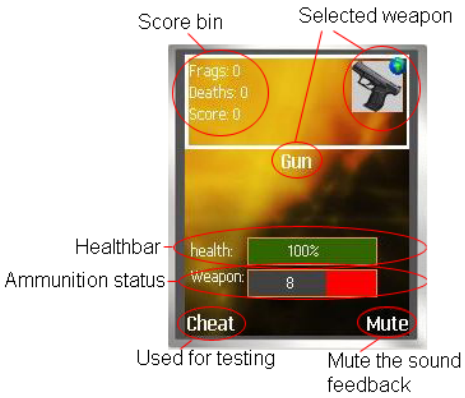
18.2 BandHero

BandHero sounded like it was very quick to make, so we decided to do it. What we ended up with is a functional prototype of a simple note-playing game. This time we wanted to see if we could use Java to make testing fast and easier due to having all the projects in one compiler. To make the game, we used a library called JFugue which can open and parse a midi-file. A midi-file is a file which has notes for every instrument in a song in it, so we parsed each note from all of the tracks and assigned them to each player. The client displays the same background as the selected instrument on the big screen (see Figure 18.2) to be able to recognize the instrument fast. The controls are simple, press 1-3 for the 3 topmost lines and 4 or 7 for the last line. As we found out, it was very hard to control a guitar with 6 buttons on the mobile device as the keys of the device are too close together. This made us modify the game so that the player will only use 4 buttons which are spread around the keypad. Unfortunately, a drawback to BandHero was soon discovered while testing. The game has a limit on the number of active players because we depend on the number of tracks in a midi-file. This can be fixed by making it possible for players to compete on the same tracks which reduces the need for many split screens.

18.3 Space Battle

Space Battle was the first game we started on at the beginning of this thesis. We were able to reuse much of the code from SlagMark, the only difference is that a new library was included which made it easy to load 3d-models and animate them. With this we can use an external modeller to get good 3d-models with textures. There are many sites on the Internet that provide models free of charge, we have found many of our models there. We will not zoom in on the models too much or view the models from the side, this means that we could remove many of the polygons to make the models uglier close on but with faster drawing.

We spent much time developing Space Battle, but unfortunately it became too late due to the fact that some functions in the script client were late. Much of the game is complete but some game logic is missing. These are routines for selecting teams, choosing roles and an interface to be able to choose between buildings to place. There should also be some messages to send when the player is in proximity of buildings so the client can display possible actions made



(a) SlagMark client controller

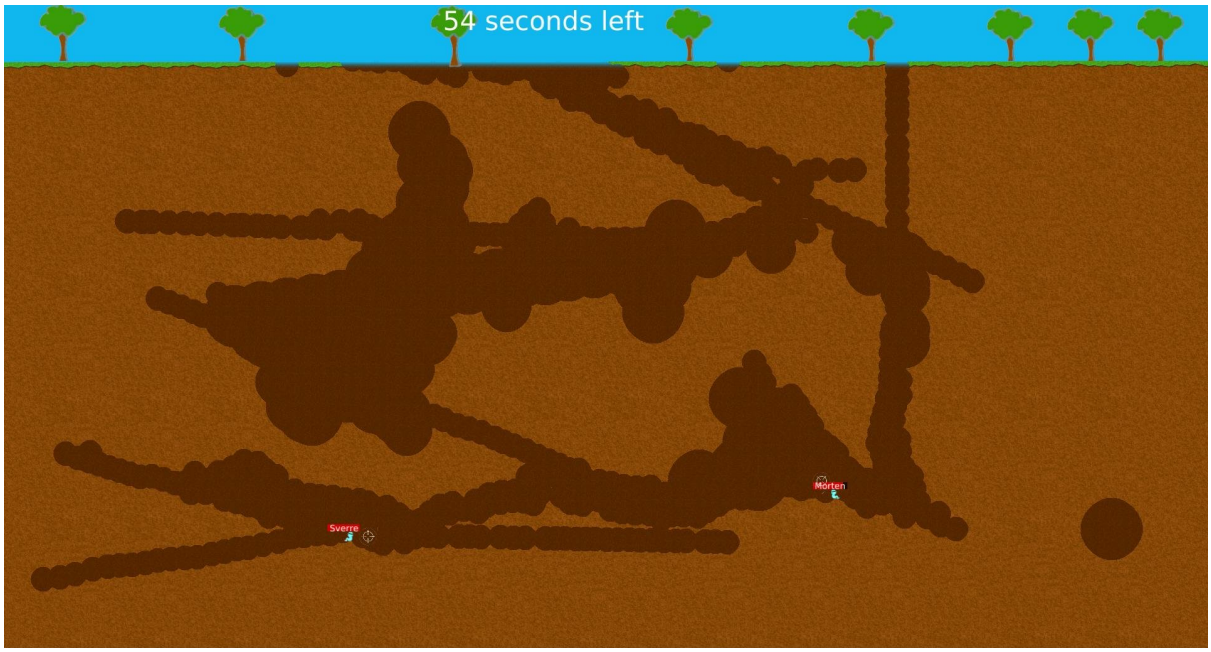
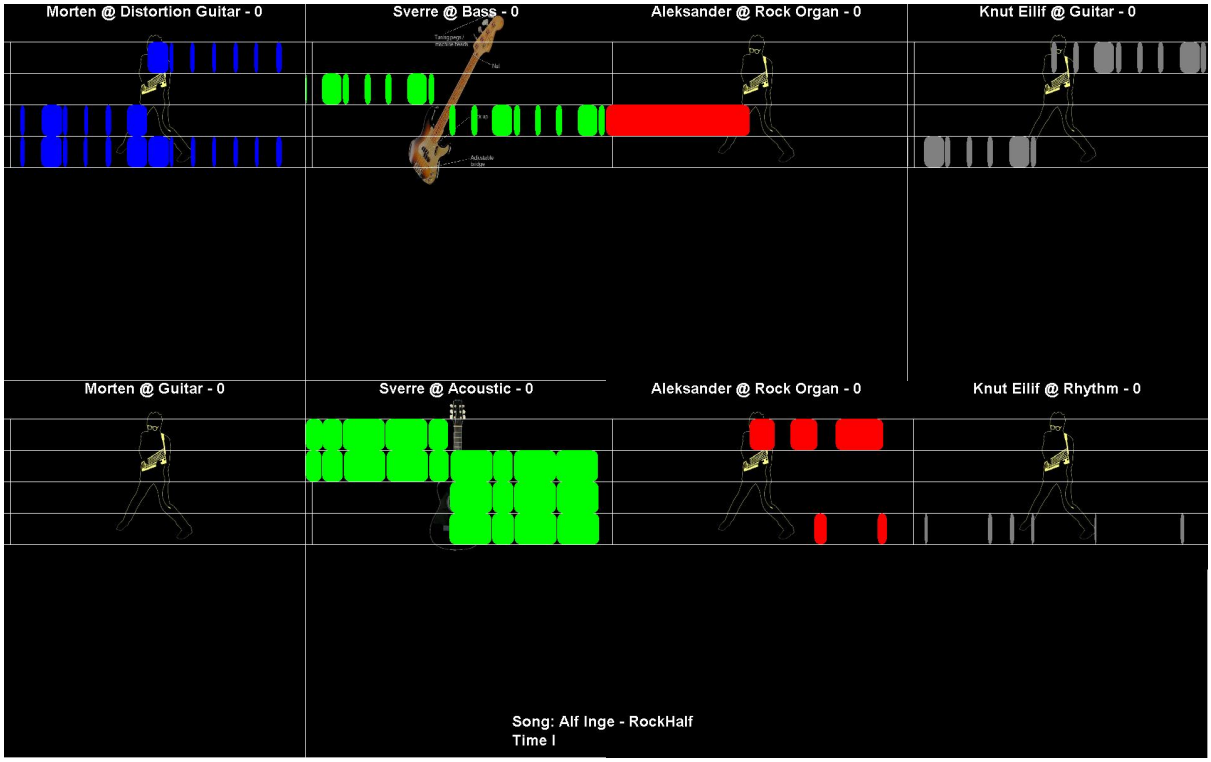


Figure 18.1: SlagMark screenshot



(a) BandHero main display

(b) BandHero client with guitar

Screenshots of BandHero

available by the building. Procedures to place a building, zooming of the building when it is spawned, player logic, collision detection for rotated rectangles, circles and more is available, so the game is almost complete.

The game is somewhat playable, and we can see that it has some potential. A nice feature is that the game has mechanisms available to zoom in and out on the playing field (see Figure 18.2), which gives the game a more movie feeling where big battles can happen and the endgame with fewer ships can be zoomed in for more focus.

18.4 SelfFish

SelfFish was developed in Java with the use of JOGL and JOAL which are Java bindings for OpenGL and OpenAL respectively. We used another platform to see if Java has enough performance to make games in. We are quite happy with the result which was developed in a couple of days. All graphics are borrowed from other similar games and applications (see Figure 18.4 and Figure 18.4).

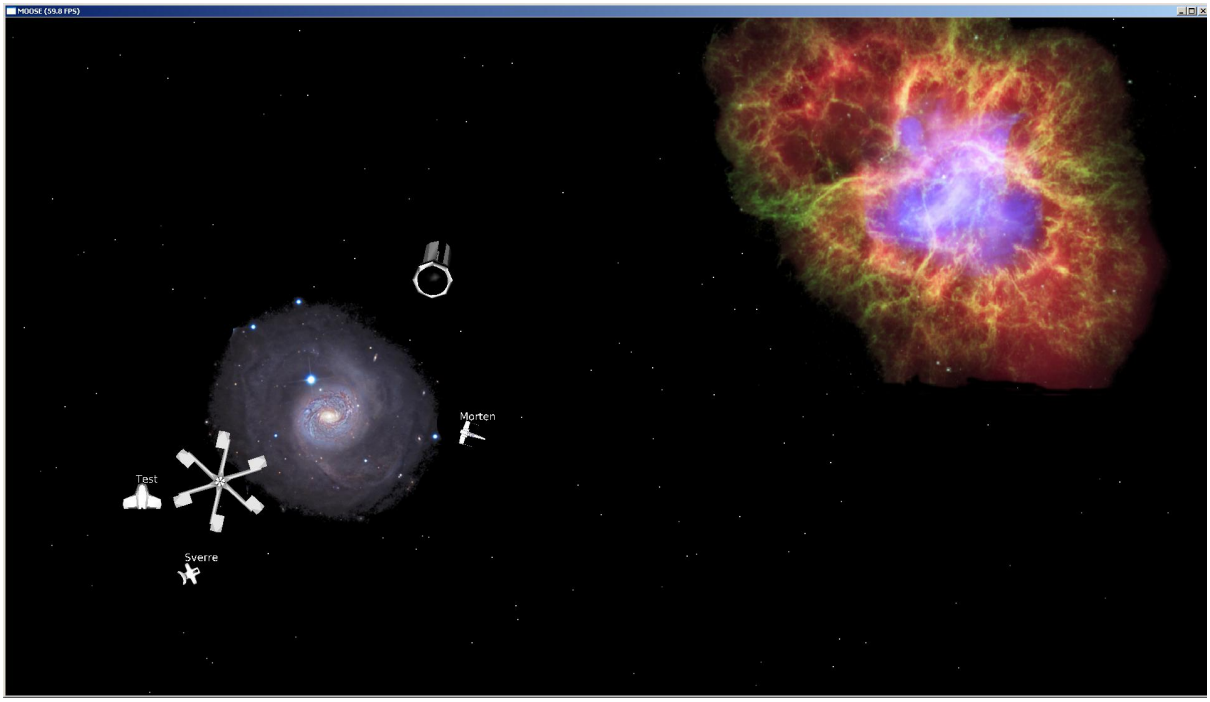
The mobile client shows an overview screen with level, energy and a boost status. We decided to have the booster in its own bar to make the gameplay less confusing, the booster can be used and must be refilled before the next use. The booster lasts 5 seconds in the current implementation. Further on, the client's controller options is only directional keys and a button to start the boost.

When the game starts, every player is assigned a random location on the screen. The game starts constantly dropping plankton (which is a simple rotating polygon with texture) from the top in a random location with a fixed speed. When a little fish at level 1 has eaten more than 20 planktons it gains another level. Higher level fishes gets more red, bigger and a little bit slower. But they can start eating other players which are much more nutritient than plankton (the eater will gain much of the energy the eaten fish had inside). The 3rd level makes the fishes bigger again and able to eat 2nd level fishes.

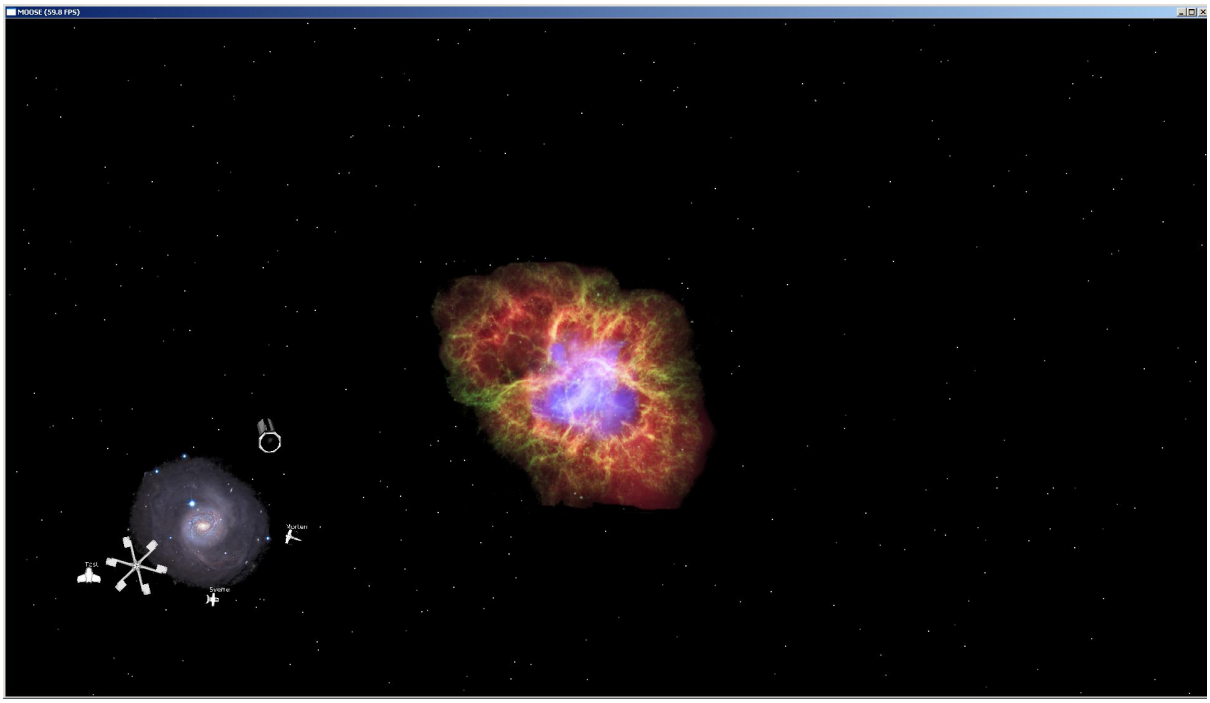
The game ends when there is one fish left or the game times out, in this implementation we limited the player's lives to 1. Scoring is based upon the energy gained and bonus for players eaten.

18.5 PopQuiz

PopQuiz is the simplest game we have, it is made in Java and simply displays a question and an overview of the participants' scores (see Figure 18.5). The game reads groups of questions from an xml-file, and displays the questions in succession where each question has changed the order of the displayed alternatives. We have found the questions on various websites, and they are all factual which actually makes the game a learning utility. This prototype can easily be improved with better graphics, music, videoes and more variations in what gives best score. We could for instance make the fastest answer get a big bonus for a correct answer or introduce other features found in Buzz.



(c) Screenshot of Space Battle



(d) Zoomed out

Figure 18.2: Screenshots of Space Battle

(a) Screenshot of SelfFish

(b) A bigger player has just eaten another player

Figure 18.3: Screenshots of SelfFish big screen display

(a) Normal client display (b) You have been eaten!

Figure 18.4: Screenshots of mobile client for SelfFish

When the game starts the questions are read from the xml-file in groups. The group is then randomly selected and the questioning starts in the sequence read from the xml-file. The various alternatives inside a question is also randomized, to make it difficult to remember a sequence, the players must know the answer. This implementation contains only text displaying facilities, but other media can easily be added in the xml-file and displayed instead of the text. The background is currently only black which is very dull.

The client displays a simple background, and the players are expected to press the number-button they want to answer. The answers are only checked when the timer runs out. It will be easy to add other gametypes where the first one to answer gets the score or other types.

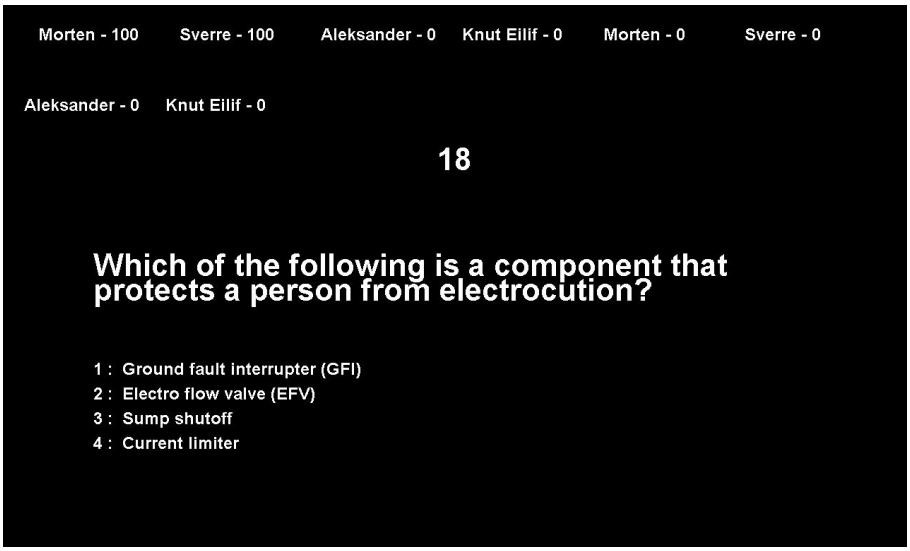


Figure 18.5: Screenshot of PopQuiz

Part VI

Evaluation

CHAPTER 19

Introduction

In this part we will present an evaluation of the results we have come to from making this thesis. The first chapter describes technological results measured up to the requirements from our depthstudy. We measured and presents some empirical data which underlines our findings and improvements.

We then present social feedback gained with regards to the entire concept from two testsessions, one with NRK Newton and one at Kosmorama. We got very positive results, the concept managed to generate interest and we got very good feedback from our target audience (a 5 out of 6 due to lack of good graphics).

The last chapter evaluates the different game concepts with different criterias, on behalf of user feedback and observations done during various testing.

CHAPTER 20

Technology

This chapter will evaluate the MOOSES framework on its functional requirements which were defined in our depthstudy [50]. It will also look at communication performance of the framwork between server and mobile phone controller.

We will give a summary of the results from the requirements testing here, for a full listing of the tested requirements from the prestudy see Chapter E.

20.1 Functional Requirements Evaluation

We have, as mentioned in Section 4, used evolutionary prototyping as a guideline, the testing has been carried out during the implementation process. If the requirement was not fulfilled, we would do another loop and then evaluate again.

What we wanted to ensure with our functional requirements was that the framework works as it should and is flexible. Most of these are important features that must work to be able to play the games. Our most important demo at Kosmorama tested therefore most of these. We had 12 persons at the same time voting on different "plugged in" games, playing and seeing their highscore, so requirements FR1, FR2, FR6 and FR7 are tested to positive results.

Other requirements are directed at future and possible improvements and these we had to test programmatically. As the framework from Tellu is very flexible and we do not hide this, we can provide the same flexibility with little added implications. So we can answer yes to FR3, FR4 and FR5 just by providing the connection types in the application descriptor. We made a simple client running on the computer with TCP/IP, by using Tellu's J2ME emulation library which provides J2ME classes for the standard computer's J2SE. This required only that we changed the connection parameters in the application descriptor.

20.2 Quality Requirements

The tested Quality Requirements from Section D.2 are summarized below.

We have improved the scalability and response time of the framework, so where there was a little delay before, it is imperceivable now. We measured a response time of 0.026 seconds from the client to the server or the opposite direction (used for keypresses), or 0.048 seconds through the unoptimized scripted solution by Sverre Morka. These response times are nevertheless very good, and much under the 150 ms limit we have outlined as our hopeful goal in Section 8.3. The games must also react to the events in a timely fashion, but that is the developer's response.

The next non functional requirement dealt with usability, that the system is easy to use. We passed around phones at the test-session, and people started logging in and voting before we had described how. This was alright as we could show that people could join in-game as well.

The modifiability requirements were tested with great success at Kosmorama, and the billing requirement were tested with us implementing a basic billing module which we could interchange.

Testability requirements were already included with Tellu's framework and we did not hide it. We used these testing features to debug a running system with great success.

Usability was also tested at the Kosmorama session and we have provided Tellu with a live running system. This was installed simply by assembling the hardware and starting the binary we sent.

20.3 Performance Evaluation

We made many synthetic tests to test the performance of different aspects of our framework. To find the limitations of what is possible to do, we have to test performance regards to throughput, latency and scalability. All the phones and the access point we have tested uses Bluetooth 2.0 EDR.

20.3.1 Ping performance with one client, different methods

This test was designed to show the performance improvement we have gained by utilizing the methods outlined in Section 17.1. The client sends messages to the server from the statemachine to remove the overhead of the script with the new and old message, and then from the script. Figure 20.1 shows a graph where we can see that by changing the message we could send 250 messages at 12.8 seconds instead of 15.5 seconds, which gives us a performance improvement of 18.2%. Another artifact we can see, is that the new message is also much more stable around 12.8-13.0 seconds which can mean that the old packet jumped between different packettypes according to the signal strength whilst the new one fits in one.

20.1

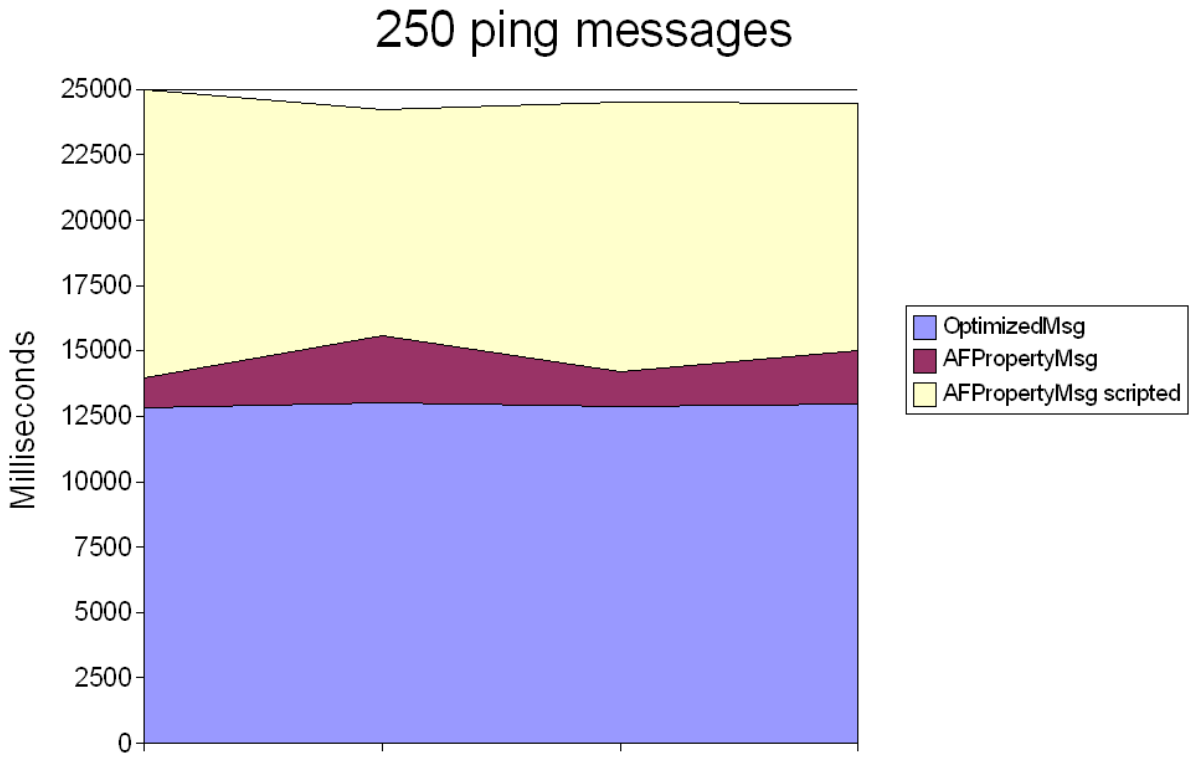


Figure 20.1: Graph over time spent sending 250 messages by different methods

20.3.2 Ping performance with more clients

This test shows how good Bluetooth scales with more simultaneous pinging with more clients. The total number of packets received and sent in 30 seconds with different numbers of clients are:

- **1** - 610
- **2** - 1330
- **3** - 1880

So with 1 client we saw 40.5 packets per second. The client does not send a packet before it has seen a packet incoming, so this means that each message spent 25 milliseconds from the client to the server or the other way around. Nokia measured a round-trip latency to 30 milliseconds with their implementation.

As we can see, we get more pings with more phones. This is probably caused by the fact that the mobile can not process the messages faster than they are transported by Bluetooth. This means that the limitation in todays real world performance is at the mobile's processor. Faster phones would probably send more pings. The scalability is also good for 3 clients even with maximum utilization.

Unfortunately, when we tried to run the emulator at the same computer, we got less pings. So it seems that the script is the limitation for our final client. This problem is being reduced by Sverre Morka in his thesis [35].

20.4 Framework evaluation

We have also cleaned up the code between the plugin and the framework. The class GameInterface is now only a interface, and to communicate from the game to the framework (and to the player's phone) we provide a PlayerConnection class. This class has functions that can make the player's phone vibrate, update a player's score and send a generic message to a player. This generic message will be received by the player's phone and executed by the script engine at the player's phone.

We made a simple game to test the simplicity of the framework, and to show that it is possible to make a game in Java. This game was BandHero, and was developed in just 3 days. We found some new issues we have to fix in the process, because the game was more team-playing we should have a possibility for a team-based highscore. This could also foster for a feeling of teamwork when everybody has to participate to make the final song.

One very important flaw that we have found is that by using JNI between our framework and C++ it makes the framework very fragile. If the C++ side crashes it will bring down everything. This is a performance/stability tradeoff, much testing before deployment of games should remify the issue. Another solution to this issue could be to make the game VM connect to the framework by a connection rather than run in the same VM. If the game crashes we could simply restart it from the framework. This could also make it easier to develop game engines

in other languages, an interesting possibility could be to provide a Macromedia Flash endpoint. Flash is used by many people and companies to make simple games or web-applications very fast. We could use this to prototype and test game ideas quickly, or provide games where the performance is not critical like in cardboard games or quiz-games.

Another issue that came up during testing was when a game has a limited number of players. The issue was which players should be allowed to join the game. In the demos we had at Kosmorama some people were a frustrated that they had to play BandHero when they did not vote for it.

A simple method to deliver resources from the server to the clients was developed by Tellu early this year. We have not incorporated this feature, but this would make the client application smaller and provide resources on demand when new games are plugged in to our framework without any cost.

To be able to reduce the size of the packets we send, we have to make our own gateway. Gateways are a convention in ServiceFrame that sits at the endpoint before that packet is sent. By making our own, we can use the code we have made without any modifications. Just by removing the possibility to address other statemachines than the game-statemachine, we should be able to reduce the overhead of the packets to nearly zero. Every packet sent should have an integer id and the game payload serialized as before. When the gateway receives a packet it will add the header needed to make ServiceFrame handle the packet. The header will just point it to the currently running game.

20.5 Game performance

During the demos at Kosmorama (Nova 10), we had up to 12 players playing at the same time. This was the first time we had more than 4 players, so we were very happy that it worked that well considering that those clients span more than one Bluetooth-piconet.

Bluetooth worked very good, we had some problems with performance of the old scripting solution on the old K750i, helped by the fact that every explosion were being sent to every player in the game, no matter the distance. Lag sometimes spiked happened 2-3 times the 3 hours we tested, the lag-spikes lasted 10 seconds but that was extremely annoying. Some bugs on the mobile client caused them to sometimes drop out or not be able to connect to the server, two persons even had their mobile spontaneously reboot! There are many possible reasons that could make this happen.

The SlagMark-game could be optimized more, it started to lag a bit when everyone was shooting at the same time. The reason here is that we used Morten's old computer as the server, and to improve the scalability even more, we have to remove other time-critical factors that are exponential in time with regards to the number of players. The only method we have in SlagMark which is exponential, is the collision-detection routine. We check each weapon instance against every worm each frame, which makes the performance hit very big if a large number of players shoots many shots at the same time (which is very likely to happen in a big battle). This exponential factor can be close to removed by utilizing a "binning system". A binning system is a simple system which divides the screen into a number of bins (as parts), and then places each gameobject into their corresponding bin. This will now cut down on the

required checks in the collision routine, as we only have to check against other gameobjects in the same bin.

CHAPTER 21

Social

The social test was done by studying players at two test sessions. One was done while NRK's show Newton[38] made a report on our concept, and the other at Kosmorama. The testers supported all our targeted age groups (see Figure 21.1 for the Kosmorama testers). The testing process was concluded with that the players entered the location and sat down freely. They were then given a brief introduction on the concept and how they should proceed to play. When this was done we handed out a few preinstalled mobile phones, and checked with the audience if anyone had any phones of their own. These where then sent the client via Bluetooth. Once everyone was set up, the server was started and the testers could log in. There was some confusion once the "Vote" screen came up as whether they had to specifically choose a game or if it was actually an open vote. Once this was resolved the voting went on without incident throughout the testing period.

Figure 21.1: Some of Kosmorama's participants

21.1 SlagMark Observations

The first game to be tested was SlagMark (see Figure 21.2). The testers needed litle help in getting into this game once the control buttons were explained, and they commented later that it was an easy game to get into. Once the players got started they seemed to be very focused on their own game, but as soon as someone was killed, the focus quickly shifted to whom the killer was and revenge was quickly sought after. After each game session, the highscore was scrutinized for whom the winner was, and that person quickly became the target for the next session (as he/she was seen as a deadly opponent). There was also a sence in community about this as the loosers seemed to team up against this one person. SlagMark also seemed to be the most favorite game among the testers, and clearly won the most ”voting screens”.

Figure 21.2: Kosmorama testing SlagMark.

21.2 BandHero Observations

After a few rounds of SlagMark, BandHero was voted up for playing (see Figure 21.3). A quick introduction to the controller setup was given and the players got started. There was a little confusion about who had which instrument, which was sorted out with a little explanation that the controller had the same instrument image as on the big screen. While this game had a shorter introduction then SlagMark, it proved much harder to master. The players found this a much more challenging game. Most players seemed to take this lightly and the laughter could be heard as the tried to get the notes correctly. As the highscore screened show at the end, the players were unanimous in that the concept was very good but needed some more work.

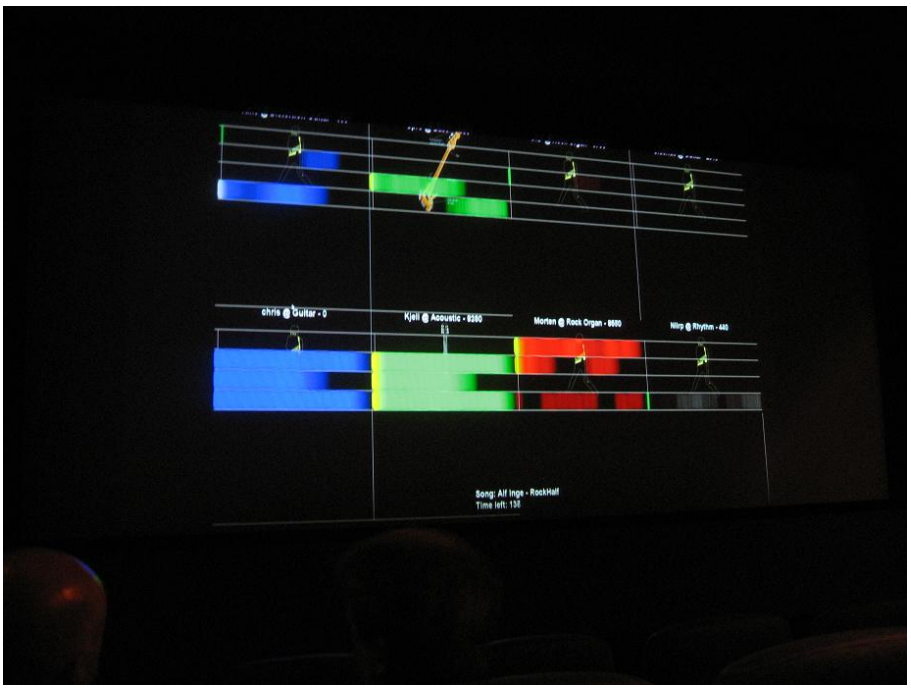


Figure 21.3: Kosmorama testing BandHero

21.3 Newton Critics

At the end of the Newton filming, the producer and kids went of to a closed location to give their verdict on the concept and games (see Figre 21.4 to see our critics). The program later showed that the kids gave us a 5 out of 6 points (see Figure 21.5). The only drawback was due to the simple use of programmer-art in the game. They were extremely impressed that they would be able to play on the cinema-screen and said that they would be very likely to pay to play in the future. They all felt it was a very social happening and would love to play more with their friends. Video of this is included on the DVD.

21.4 Conclusion

On both tests the concept was very well received, we could hear that the audience was excited between each round due to loud buzzing, laughing and talking. On certain occations you could hear players shouting out ”Who was that?” when they where killed, only to hear friendly laughter afterwards once they found out who it was. This also got people who did not know eachother from the beginning to talk to eachother. This proved to use that the social aspect of the concept, which we always thought was one of the key elements, was very much present.

We were very happy with the observations we made. Our concept seemed to have the social aspect we are aiming for: Players interacting with one another in a real life setting and enjoying themselves.

Figure 21.4: Newton's critics

Figure 21.5: Critics verdict's dice is a 5!

CHAPTER 22

Game Concept

In this chapter we will evaluate the different game concepts on behalf of user feedback and observations done during our testing. The different games will be evaluated on:

- ➢ Controller
- ➢ Visualization and sound
- ➢ Playability and user interaction

SlagMark and BandHero was tested at Kosmorama and by NRK's show Newton[38], while the rest of the games where tested by a smaller group of students.

22.1 SlagMark

This game had good reception all the way throughout testing, and was the most favorite game with the majority of the asked testers. They felt it was fairly easy to learn, and friendly competition grew quickly.

Controller was well liked on this game, both on the usage of the screen and weapon usage. The button usage was also well liked.

Visualization and sound got some critique by the testers. It was not always easy to spot where you entered the game after death, and the local mobil phone sound was lacking on some types of mobiles.

Playability and user interaction - The tester had some difficulty starting up the game, (specifically on what buttons were used to control what functions), and required som

explanation from the developer team to get going. Once a few rounds had passed though, all players seemed to enjoy themselves alot.

22.2 BandHero

The testers enjoyed the concept of this game, but with the difficulty of mastering it, found it the least enjoyable of the tested games.

Controller recieved mixed feelings. Some testers thought the usage of buttons could have been differently to better suit the gameplay.

Visualization and sound - Sound was fun and fitted the gameplay well, but it was difficult to spot which instrument the different players were assigned and as such what notes they were supposed to play.

Playability and user interaction - The testers found this the most difficult game. Once they figured out what instrument they where playing, they got into the basics easy enough. They thought this the game the most difficult to master.

22.3 Space Battle

The testers enjoyed the cooperation this concept provided and felt the game had much potential. However, they thought the game could use more development and more options to add more depth to the game.

Controller was well liked and had good utilization.

Visualization and sound - The testers thought the visulization could have more interaction and information.

Playability and user interaction - The testers found this game the most difficult game to start, and requested a lot of information on how to play it. Once they got going they seemed to enjoy themselves alot. Some of the teams managed to communicate better and that had a big impact on the winner (the team who communicated the best usually won).

22.4 SelfFish

This game got good reviews for humour and fun action and was thought to be the second most fun game testet.

Controller was well liked and had good utilization.

Visualization and sound was also very well liked by the testers.

Playability and user interaction - The testers thought the game was easy to get into and easy to play. The humor setting gave the game a huge pluss with the testers.

22.5 PopQuiz

The testers thought this game was fun, but not very original. Some work should be done visually to make it a more interactive experience.

Controller was well liked and had good utilization.

Visualization and sound - The testers requested more interaction visually to get more into the game.

Playability and user interaction - The testers thought the game had good potential but in its current state was a little bit boring.

Part VII

Conclusion & Further Work

CHAPTER 23

Conclusion

In Chapter 4 we stated three research questions. We wanted to study social video gaming today in order to see how social video-gaming works with MOOSES. We also wanted to look into cooperative multiplayer options for MOOSES and perhaps discover if there are other steps in order to enhance gameplay in MOOSES. Lastly, we wanted to evaluate MOOSES in order to see if it met the set requirements for the framework and games. Throughout this project we have worked according to the methods described in Section 4.2. We have developed four new games for the framework based on research into social gaming and their mechanics, as well as evaluated and made improvements to the framework.

In this chapter we summarize the project's results by answering the research questions and problem definition. The content of Chapters 11, 12 and Parts IV, V and VI constitutes the basis for answering the research questions questions.

The main task in our problem definition was at the effects of social gaming, both how the game affects the player and the player affects the game. Research question I and II along with our research into how video games affect social life (see Section 11.3), has given us valuable insight to answering this.

Our second task will be to test and evaluate the MOOSES framework and perform any improvements wherever needed. This is answered through research question III. Future improvements to the framework is listen in Chapter 24.

RQ-I: **What comprises social video-gaming today and how does it fit into MOOSES?**

- ➢ Our research found that the design of a game will greatly influence what players it will attract, and those in turn will affect one another. We saw that out of the four different player-types, MOOSES games could best support Achievers and Killers, while Socializers could be supported by the social nature of concept itself. We also found that game contribution to socialization

could be divided into three categories (see Table 11.1) and that looking deeper into these presented new game design options (see Section 11.2). Finally we looked into gaming communities and discovered that it had a big impact on social games, and that games and game systems are slowly building these into their systems. We concluded that building a community around MOOSES should be one the next big steps for the concepts evolution.

a) What different social video-gaming categories are there?

➢ There are three different video-gaming categories: Freeform socialization, Competitive socialization and Cooperative socialization.

b) What different player types comprise social video-gaming?

➢ There are four different player types: Achievers, Explorers, Killers and Socializers.

RQ-II: **What type of game mechanics are suitable for use with cooperative multiplayer mode for MOOSES?**

➢ Looking into social gaming, gaming communities and mutliplayer games we have found that several game mechanics can enhance cooperative play with MOOSES. Implementing games in which players need to work together to defeat an opponent, hierarchies where lesser players look up to higher ranking players or games where indirectly helping others moves the game forward are ways to achieve this. We also discovered another important mechanic, that having different archtypes implemented in the game which have different functions. Section 8.3 we found some new controllers that could also add new dimentions to the MOOSES games.

a) What existing multiplayer attributes work best for MOOSES?

➢ To achieve good cooperation a game would need to support teams which consists of players with different in-game abilities.

b) Will this scenario work with our targeted number of players?

➢ Yes, this will work well as our implementation of Space Battle or SlagMark shows.

c) Can we utilize more then just game mechanics to enhance the gameplay in MOOSES?

➢ Yes, by constructing a community around MOOSES we create more social interaction, which enhances the gameplay.

RQ-III: **Does the MOOSES framework and the MOOSES game prototypes fullfill their roles in regards to their requirements and scenario?**

➢ During our testing we found the framwork to surpass its functional requirements. We also found through oberservation of our testers that the games fit well with the concept and that the social nature we were looking for was very much present.

a) Does the framework meet its functional and non-functional requirements?

   ➢ Yes with regards to this thesis. The framework meet and surpass its functional requirements, there are some non-functional requirements which we have not tested.

b) Does the prototypes work well with MOOSES?

   ➢ Yes. The prototype works very well within the domain. We have tested it with up to 12 simultaneously players and performance was good.

c) Does the framework need enhancement to work with cooperative gaming?

   ➢ No. The framework works well as it is for cooperative gaming. Unforseen game mechanics (like playing over Internet on more canvases) might call for modifications, but for a standard game we provide a generic interface to send messages to the mobile clients.

CHAPTER 24

Further work

This chapter looks at some ideas to where the framework should be developed next. At the end we give a short roadmap for the MOOSES concept.

24.1 Programmatical Issues

Some concrete programmatical items are listed below which should be looked at. There are also items from our depthstudy which are not relevant here.

- **Add categorized descriptions to the games listed.** The voting screen should get an overview with which category the game is in, a description of the game and perhaps an average rating (perhaps generated from play count). An important item is to list the number of players minimum and maximum which is recommended for a game.
- **Faster Bluetooth search time and distributed connections.** The Bluetooth can sometimes use a couple of minutes to log in to the system, other times it happens in a couple of seconds. When the load gets high on a Bluetooth accesspoint, new clients should connect to that one to reduce the load and increase the scalability.
- **Testing 40 clients at the same time.** A big test should be performed to see if real world performance is as good as theoretical. Also we have not tested what happens when a Bluetooth accesspoint is filled up with the maximum number of connections it supports. It worked flawlessly when many clients connected to the same accesspoint, but we do not know if an accesspoint hides itself when it is full so clients must connect to the next one.
- **Binning system for the arcade games.** As outlined in Section 20.5 there could be a common library which reduces the load on collision detection or other features like artificial intelligence. This library could then be reused in most games for MOOSES.

- **Dynamic loading of resources.** At the end of this master thesis, the prototype implementation of the client is able to fetch the script from the server but it still relies on the resources in form of pictures and sound files already being embedded with the client. ServiceFrame has support for dynamic loading of resources, but we have not used this yet.

- **Community support.** Build community support around MOOSES for the players. Utils to be used can be forums, websites or others.

24.2 Further ideas

We got feedback that a great opportunity to use MOOSES for is project kickoff meetings. This fits great with our focus on socializing where people get to know each other a little better and remove the ”stranger” feeling.

A requested feature is to make a Flash bridge for easy development of new games. There could be very many MOOSES game developers out there because of the simplicity of development with Flash. Flash could be used as a prototype for new concepts if the performance is subpar. Flash has recently gotten a new feature which allows it to be 3d-accelerated with the possibilities this adds.

Improve the existing games. The prototypes we have made are very basic and graphics, sound and media is borrowed from external sources. These can not be used in a commercial setting, and the games will probably be more involving with better graphics. We should also add new variations to the games as outlined in Section 14.1 to increase the longevity of the games.

There can never be enough games and as such more games should be developed. New concepts and variations will make MOOSES more successfull. Development of our concept Cops and Criminals (outlined in Section 14.5), would also be interesting to showcase a different 3d-person perspective game.

Electronic learning is a topic which has become more and more common in the latest years, and some more research to how MOOSES can be used for this should be done.

Development of games or applications for trainee or team learning in a businesses perspective could also be done.

24.3 Further roadmap for MOOSES

The next mediaevent of MOOSES will probably be with cooperation with NTNU’s Jentedagen. This is an event where they sponsor (bribe) new girls who are possible new students and show what NTNU is and does. We have been invited to make a competition where the winner wins an item. The event will be placed in the biggest auditorium in Studentersamfunnet where they have a big projector and screen. There will be around 250 participants which makes this a very big test of MOOSES. We will aim at around 20 players at the same time, which we hope is doable.

Further on the horizon is deployment at Kino 1 Sandvika, where they will allow us to run a pilot project to test the framework live. We have also come in contact with a big distributor called Unique Promotions [46] which has equipment in most cinemas in Scandinavia. They are very positive to MOOSES and we hope we can get them to install MOOSES all over Scandinavia.

Part VIII

Appendix

APPENDIX A

Glossary

API An application programming interface is the interface that a computer system or application provides in order to allow requests for service to be made of it by other computer programs, and/or to allow data to be exchanged between them.

CDC The Connected Device Configuration is a framework for J2ME applications targeted at devices with limited resources.

CDLC The Connected Limited Device Configuration is even smaller than the CDC mentioned above and is used for pagers and mobile phones.

Flash Flash is a rapid application framework used by many websites. It was made by Macromedia, and is provided by Adobe today.

FPS First Person Shooter

Guild An organization of persons with related interests, goals, etc., esp. one formed for mutual aid or protection.

J2ME Java 2 Micro Edition, a collection of Java API's targeting smaller consumer electronics like mobile phones, PDA's and so on.

J2SE Java 2 Platform, Standard Edition. J2SE is a complete collection of API's that enables development of Java applications on several platforms of personal computers.

L2CAP A packet based protocol used by Bluetooth.

MIDlet A Java program specialized to run on the J2ME virtual machine, often on mobile phones. The main class of the MIDlet has to be a subclass of javax.microedition.midlet.MIDlet and the MIDlet classes have to packaged in a JAR-package. To be runnable the JAR-package has to be preverified by a preverifier.

MIDP	Mobile Information Device Profile. The J2ME architecture consists of the Virtual Machine, the CLDC and a so-called profile. MIDP is the only available profile and has reached version 2.0. The profile contains a collection of API's that offers IO-functionality and gaming among other things.
MMORPG	Massively Multiplayer Role Playing Games
MUD	Multi User Dungeon
NPC	Non-Player Character
NTNU	The Norwegian University of Science and Technology.
RFCOMM	A stream based protocol used by Bluetooth.
RPG	Role Playing Game
RSSI	Receiver Signal Strength Indicator
RTS	Real Time Strategy
TBS	Turned Based Strategy
VM	Virtual Machine, code is executed inside a protected virtual machine.
MIDI	Musical Instrument Digital Interface. It is a standardized method for MIDI devices such as synthesizers, samplers, sound cards, etc to communicate musical events and data to each other
Pervasive Gaming	Pervasive gaming is gaming that transports the classic computer game from the virtual world into the real world. The players move through the physical world and experience the game through interactions with the mobile terminal and the physical world.
Accelerometer	An accelerometer is an electromechanical device that will measure acceleration forces. These forces may be static, like the constant force of gravity pulling at your feet, or they could be dynamic - caused by moving or vibrating the accelerometer.
SDk	Software Development Kit
J2SE	Java Platform, Standard Edition
J2ME	Java Platform, Micro Edition

APPENDIX B

Contents on the DVD/Zip-file

We have included content which have been made by and about us in the process of writing this thesis.

- \media – MOOSES in media. Included are scanned versions of 2 Kosmorama articles and audio from an interview for NRK Radio (was sent on the news). Our supervisor will send a DVD with all videoes made of MOOSES, from our depthstudy, NRK Newton and the Kosmorama event. Links to news MOOSES has been featured in are available at our MOOSES homepage [56].
- \src\mooses – Source for MOOSES server, client and clientscripts.
- \src\games\slagmark – Source for SlagMark.
- \src\games\selffish – Source for SelfFish.
- \src\games\popquiz – Source for PopQuiz.
- \src\games\spacebattle – Source for Space Battle.
- \src\games\bandhero – Source for BandHero.

APPENDIX C

Research Methods

The three approaches in short:

The engineering approach (scientific) In this approach one have to perform iterations of observing the existing system, suggesting improvements and building and analyzing the new system. This continues until no more improvements can be found.

The approach is strictly evolutionary and implies access to existing models of processes, products and the environment in which the software is developed.

The empirical approach (scientific) Based on a model of the domain a set of statistical and qualitative methods are proposed. Then these models are applied to case studies, measured and analyzed, and the result is a validation of the model.

This distinct the approach from the previous one since a new model is proposed. It is also more reliable to validate the model through the use of case studies. This approach is widely used in all fields of research.

The mathematical approach (analytical) A formal theory or a set of axioms is presented, the theory are developed and a result is derived from it. It is preferable to have this results compared to empirical observations.

APPENDIX D

Functional- , Non-functional- Environmental Requirements

Here we present an exerpt of the functional-, non-functional- and environmental requirements for the MOOSES framework.

D.1 Functional Requirements

Functional requirements are a set of instructions reflecting the functionality which must be implemented in the application. The requirements are presented in Table D.1.

Functional Require-ments	*Description*
FR 1	The framework must support multiple players.
FR 2	The framework must support the ability to ”plug in” games.
FR 3	The framework must support the use of different player controllers.
FR 4	The framework must support the use of different billing methods.
FR 5	The framework must support running multiple games at different locations.
FR 6	The framework must support voting of which game to play.
FR 7	The framework must support the logging of high scores for the players.

Table D.1: The functional requirements for the Videogames Framework

Additional information about the functional requirements

FR 2 By ”plug-in” games we mean that the framework will only supply an interface for the games to work through. The interface will support player input and feedback. This is to

give game-developers full freedom when developing games for the framework.

D.2 Quality Requirements

An important thing to keep in mind when designing applications, is to consider the non-functional aspects; The quality requirements. They are often not so clearly stated by the users and stakeholders of a system, but are nonetheless very important for the user satisfaction and the architecture. An overview of these requirements are presented in Table D.2. These requirements will be refined and presented as ways to achieve the quality attributes; Usability, Performance, Modifiability, Availability, Security and Testability. The following roles is mentioned in this section and it is important for the reader to distinguish between these:

- **The framework developer** - The developer that is responsible for changes and modifications to the framework. This group is interested in the quality requirements: Modifiability, Testability
- **The game developer** - The developer that is responsible for designing and writing games for the framework. This group is interested in the quality requirements: Modifiability.
- **The user** - The person playing games on the framework designed by the game developers. This group is interested in the quality requirements: Usability, Performance
- **The system owners** - The organization / person(s) running and maintaining the framework and games. This group is interested in the quality requirements: Modifiability, Availability, Security

Non-Functional Requirements	*Description*
NFR 1	The framework must be able to transfer messages fast enough for real-time interaction. By fast enough, controller input should at worst be visible on the screen no later then .5 second after a player command is initiated.
NFR 2	The framework must make it seem effortlessly for the players to login and use.

Table D.2: The non-functional requirements for the MOOSES Framework

D.2.1 Availability

A systems faults and failures are associated with availability. A fault occurs when something goes wrong in the system and is not visible. If a fault becomes a failure, the error will be visible. For instance, if the network connection is lost between two devices without one device registering the loss, a fault has occurred. Then, if the application acts as if the connection is still available, a failure will occur.

A1 - Game server downtime	
Source of stimulus	Runtime issue
Stimulus	The game server goes down
Environment	Run time
Artifact	The game server module
Response	The framework notice the game server no longer responds and restarts it
Response Measure	The game server should have a maximum of 10 minutes downtime per day

D.2.2 Modifiability

Modifiability has to do with changes to the system. It is then vital that the changes can be performed without much hassle. For instance, if a maintainer wants to change an encryption algorithm, then he/she should only need to replace one module of the system and not many small changes in many modules. A change does not necessarily need to be made by a maintainer/developer. It can also be made by the end-user, for instance in a configuration set-up.

The MOOSES Framework should be designed in a way that allows future modifications and/or additional modules.

M1 - Modify the billing module to handle a different billing method	
Source of stimulus	The framework developer
Stimulus	The framework developer wants to add or change the current billing method to a different solution
Environment	Design time
Artifact	The billing module of the Videogame Framework
Response	The framework developer should only have to follow the same message systems between modules inside the framework related to the billing module, and not have to make any modifications to other parts of the framework
Response Measure	Presuming that the module works, it should not take more then a 2 days

D.2.3 Performance

Performance has to do with how long the system can respond to an event that occurs. Such events may come from several instances. These instances can be an end-user, the system itself or from other systems.

M2 - Dynamically choose and start games	
Source of stimulus	The user
Stimulus	The user votes for what game he/she wants to play
Environment	Run time
Artifact	The game server module
Response	The game server updates the vote list according to the users vote, and starts the game with most votes. In the instance of a tie, a revote between the tied of games is performed.
Response Measure	The user(s) have 1 minute to vote for a game. The game that wins should start up in no less then one minute after the vote is done

M3 - Dynamically add player to the running game	
Source of stimulus	The user
Stimulus	The user logs onto the framework and joins the running game
Environment	Run time
Artifact	The game server module
Response	The game server adds the user to the running game
Response Measure	The user(s) should be added to the game in no less then 30 seconds unless the game has added lockout timers

P1 - Response time from player actions	
Source of stimulus	The user
Stimulus	The player makes game actions via his/her controller
Environment	Run time
Artifact	The framework
Response	The action chosen by the player should play out on the screen
Response Measure	The action should happen on the screen in no less then .5 second presuming the game is a real-time game. If not the action should become visible in the next game turn

D.2.4 Security

Security is concerned with the systems ability to prevent unauthorized usage/access without compromising normal usage. Attacks can be unauthorized attempts to access or modify data.

D.2.5 Testability

In order to find bugs and faults in the system, it needs to be testable. Designing an architecture that can be easily tested for faults will save a lot of time. There are several different ways of doing this. Most of them involve monitoring the systems internal state and logging output that is easy to interpret.

S1 - Check that the user has a viable billing option	
Source of stimulus	The user
Stimulus	The player tries to log onto the framework
Environment	Run time
Artifact	The framework login and billing modules
Response	The login and billing module should respond if the user does not have a viable billing option.
Response Measure	Depending on the implementation of the billing module, the module should provide the user with a secure billing option

T1 - Status of running modules	
Source of stimulus	The system owners
Stimulus	The system maintainers wants to check the status on the running system
Environment	Run time
Artifact	The framework
Response	The framework must supply a gui based tool for debugging of the system
Response Measure	The framework should respond with the current running status of all modules currently running in the system

T2 - Test a new message or module state machine	
Source of stimulus	The framework developers
Stimulus	The developer wants to
Environment	Run time
Artifact	The framework
Response	The framework must supply a gui based tool for debugging of the system
Response Measure	The gui should allow the developer to test new messages and module-state machines in real-time

D.2.6 Usability

Usability is concerned with how easy it is for a user to perform a certain task and how the system displays information to the user. Usability is an issue that often must be considered in the early stages of architectural design. If major problems regarded to usability is detected late in the project phase, the more repair and modification has to be done to the architecture.

D.3 Environmental Requirements

In this section we will give a short description of the environment that is needed by the framework.

U1 - Log in and start playing	
Source of stimulus	The user
Stimulus	The user wants to log in to play the current game hosted
Environment	Run time
Artifact	The Client module of the MOOSES Framework
Response	The user should only have to start the downloaded client application, choose to log on to the framework with a premade user name and password. This should give him access to join the running game
Response Measure	Presuming the user has the latest client and server is not full, the user should be able to perform the action without any further assistance

U2 - Set up and install the framework	
Source of stimulus	The system owner
Stimulus	The system owner wants to set up and install the framework
Environment	Install time
Artifact	The MOOSES Framework
Response	The system owner should only need to install the jar file(s) on the hardware configuration and start the application via Java command line
Response Measure	Presuming the system owner has Java installed and set up the hardware correctly (server, projector, client communication), the setup should require minimal computer knowledge to perform

Java The framework is implemented in Java 1.5 and is therefore dependent on Java support.

Operating System (OS) With they use of Java implementation the framework is OS independent, to the extent of using a Java compliant OS. Some pluggable games may put restrictions on the OS, but not the framework.

Screen The framework is independent of screen and projector hardware. The pluggable games will be the modules that put restrictions on such hardware.

APPENDIX E

Requirements Evaluated

Here we have listed the requirements tested in full. See Chapter 20 for a summary.

E.1 Functional Requirements Evaluation

The results of the functional requirements are presented in Table E.1.

E.2 Quality Requirements

The tested Quality Requirements from Section D.2 are listed and explained below.

FR1: The framework must support multiple players.	
Executor	Morten Versvik & Aleksander Spro
Date	20.04.2007
Time used	4 hours
Evaluation	Success: 12 people played at the same time at Kosmorama.

FR2: The framework must support the ability to ”plug in” games.	
Executor	Morten Versvik
Date	18.05.2007
Time used	0.5 hours
Evaluation	Success: New games can be plugged in by providing a new Java class in the plugin directory.

FR3: The framework must support the use of different player controllers.	
Executor	Morten Versvik & Aleksander Spro
Date	18.05.2007
Time used	1 hour
Evaluation	Success: New controller types can be developed by using either a basic TCP/IP connection, Bluetooth connection or ServiceFrame.

FR4: The framework must support the use of different billing methods.	
Executor	Morten Versvik & Aleksander Spro
Date	19.03.2007
Time used	0.5 hour
Evaluation	Success: Billing methods can be changed by providing a new actor in place of the default one.

FR5: The framework must support running multiple games at different locations.	
Executor	Morten Versvik & Aleksander Spro
Date	18.05.2007
Time used	0.5 hour
Evaluation	Success: Each server runs one game, each one connects to a central server for a central userdatabase.

FR6: The framework must support voting of which game to play.	
Executor	Morten Versvik & Aleksander Spro
Date	20.05.2007
Time used	1 minute
Evaluation	Success: Players are able to vote on the different available games.

FR7: The framework must support the logging of high scores for the players.	
Executor	Morten Versvik & Aleksander Spro
Date	20.05.2007
Time used	5 minutes
Evaluation	Success: Highscore is logged and displayed after each game session.

Table E.1: The non-functional requirements tested

NFR1: The framework must be able to transfer messages fast enough for real time interaction. Fast enough means that controller input should at worst be visible on the screen no later than 0.5 seconds after a player command is initiated.	
Executor	Morten Versvik
Date	25.05.2007
Time used	1 hour
Evaluation	Success: OptimizedMsg fits into 1 Bluetooth timeslot, should provide good scalability. Measured responsetime is at 0.048 seconds with an unoptimized script solution.

NFR 2: The framework must make it seem easy for the players to log in and use.	
Executor	Morten Versvik & Aleksander Spro
Date	20.04.2007
Time used	4 hours
Evaluation	Success: 12 people played without much guidance.

Table E.2: Non-functional requirements tested

A1 - Game server downtime	
Source of stimulus	Runtime issue
Stimulus	The game server goes down
Environment	Run time
Artifact	The game server module
Response	The framework notice the game server no longer responds and restarts it
Response Measure	The game server should have a maximum of 10 minutes downtime per day
Result	Not tested: This is not important for our thesis. It should be easy to make a program which notices this and restarts the binary.

Table E.3: Availability requirements tested

M1 - Modify the billing module to handle a different billing method	
Source of stimulus	The framework developer
Stimulus	The framework developer wants to add or change the current billing method to a different solution
Environment	Design time
Artifact	The billing module of the Videogame Framework
Response	The framework developer should only have to follow the same message systems between modules inside the framework related to the billing module, and not have to make any modifications to other parts of the framework
Response Measure	Presuming that the module works, it should not take more then a 2 days
Result	Success: Another billing module can be made by using the same messages that allow or disallows an actor, and be changed by using the deployment descriptor.

M2 - Dynamically choose and start games	
Source of stimulus	The user
Stimulus	The user votes for what game he/she wants to play
Environment	Run time
Artifact	The game server module
Response	The game server updates the vote list according to the users vote, and starts the game with most votes. In the instance of a tie, a revote between the tied of games is performed.
Response Measure	The user(s) have 1 minute to vote for a game. The game that wins should start up in no less then one minute after the vote is done
Result	Success: 12 players voted and played 3 different games at Kosmorama. Ties are handled by voting between the games.

M3 - Dynamically add player to the running game	
Source of stimulus	The user
Stimulus	The user logs onto the framework and joins the running game
Environment	Run time
Artifact	The game server module
Response	The game server adds the user to the running game
Response Measure	The user(s) should be added to the game in no less then 30 seconds unless the game has added lockout timers
Result	Success: Players could join in-game when they started the client.

Table E.4: Modifiability requirements tested

P1 - Response time from player actions	
Source of stimulus	The user
Stimulus	The player makes game actions via his/her controller
Environment	Run time
Artifact	The framework
Response	The action chosen by the player should play out on the screen
Response Measure	The action should happen on the screen in no less then .5 second presuming the game is a real-time game. If not the action should become visible in the next game turn
Result	Success: Players did not notice any noticeable lag when the game is running normally.

Table E.5: Performance requirements tested

S1 - Check that the user has a viable billing option	
Source of stimulus	The user
Stimulus	The player tries to log onto the framework
Environment	Run time
Artifact	The framework login and billing modules
Response	The login and billing module should respond if the user does not have a viable billing option.
Response Measure	Depending on the implementation of the billing module, the module should provide the user with a secure billing option
Result	Not tested: This thesis did not have focus on the client. The server has a module ready for this which needs to be implemented.

Table E.6: Security requirements tested

T1 - Status of running modules	
Source of stimulus	The system owners
Stimulus	The system maintainers wants to check the status on the running system
Environment	Run time
Artifact	The framework
Response	The framework must supply a gui based tool for debugging of the system
Response Measure	The framework should respond with the current running status of all modules currently running in the system
Result	Success: ServiceFrame has an optional included gui dialog for sending custom messages to all actors, and also for discovering and printing the current states of all running modules, both locally and remotely.

T2 - Test a new message or module state machine	
Source of stimulus	The framework developers
Stimulus	The developer wants to
Environment	Run time
Artifact	The framework
Response	The framework must supply a gui based tool for debugging of the system
Response Measure	The gui should allow the developer to test new messages and module-state machines in real-time
Result	Success: ServiceFrame's included gui can send messages to any actor and system, can be used to test modules and messages.

Table E.7: Testability requirements tested

U1 - Log in and start playing	
Source of stimulus	The user
Stimulus	The user wants to log in to play the current game hosted
Environment	Run time
Artifact	The Client module of the MOOSES Framework
Response	The user should only have to start the downloaded client application, choose to log on to the framework with a premade user name and password. This should give him access to join the running game
Response Measure	Presuming the user has the latest client and server is not full, the user should be able to perform the action without any further assistance
Result	Big success: Everyone at Kosmorama ran the client without much explanation necessary. One of the kids with Newton logged in before we had time to tell him how.

U2 - Set up and install the framework	
Source of stimulus	The system owner
Stimulus	The system owner wants to set up and install the framework
Environment	Install time
Artifact	The MOOSES Framework
Response	The system owner should only need to install the jar file(s) on the hardware configuration and start the application via Java command line
Response Measure	Presuming the system owner has Java installed and set up the hardware correctly (server, projector, client communication), the setup should require minimal computer knowledge to perform
Result	Success: MOOSES is being demonstrated by Tellu located in Asker, this installation is just a directory with a system jar-file and necessaty resources.

Table E.8: Usability requirements tested

Part IX

Bibliography

Bibliography

[1] F. Mäyrä A. Järvinen, S. Heliö. Communication and community in digital entertainment services. Technical report, University of Tampere, Hypermedia Laboratory, October 2002.

[2] Sony Ericsson Mobile Communication AB. Java docs tools. http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp, 2006.

[3] Shannon Appelcline. Social gaming interaction, part one: A history of form. http://www.skotos.net/articles/TTnT_/TTnT_136.phtml.

[4] Shannon Appelcline. Social gaming interaction, part three: Cooperation & freeform. http://www.skotos.net/articles/TTnT_/TTnT_138.phtml.

[5] Shannon Appelcline. Social gaming interaction, part two: Competition. http://www.skotos.net/articles/TTnT_/TTnT_137.phtml.

[6] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. http://www.brandeis.edu/pubs/jove/HTML/v1/bartle.html.

[7] Castulus Kolo & Timo Baur. Living a virtial life: Social dynamics of online gaming. http://www.gamestudies.org/0401/kolo/.

[8] Harlan T. Beverly. Online gaming lag. http://www.killernic.com/technology/lag.pdf.

[9] Inc Bluetooth SIG. Bluetooth core specification v2.0 + edr. http://www.bluetooth.com/Bluetooth/Learn/Technology/Specifications/.

[10] Nokia Corporation. Games over bluetooth v1.0. http://forum.nokia.com/info/sw.nokia.com/id/2b17fb6f-b9a4-4cd8-80fd-94b8251a048e/Games_Over_Bluetooth_v1_0_en.zip.html.

[11] DDRNorway. Ddrnorway.

[12] Sony Computer Entertainment Europe and London Studio. Singstar. http://www.singstargame.com/.

[13] Sony Entertainment Europe. Buzz! http://www.buzzthegame.com/.

[14] The Eclipse Foundation. What is eclipse. http://www.eclipse.org/org/, 2005.

[15] Paper Four. Paper four, mid sweden university. http://mkv.itm.miun.se/projekt/paperfour/.

[16] M. Friedl. Online Game Interactivity Theory. Charles River Media, 2002.

[17] Gefen. Dvi to hd-sdi scaler specifications. http://www.gefen.com/kvm/product.jsp?prod_id=3874.

[18] Open Source Technology Group. Eclipseme. http://sourceforge.net/projects/eclipseme/, 2005.

[19] Open Source Technology Group. Texlipse. http://texlipse.sourceforge.net/, 2005.

[20] Practical Home Theater Guide. Tv viewing distance. http://www.practical-home-theater-guide.com/Tv-viewing-distance.html.

[21] The Sydney Morning Herald. Wii breaks xbox sales record. http://www.smh.com.au/news/games/wii-breaks-xbox-sales-record/2006/12/14/1165685799546.html.

[22] IGN. E3 2006: Light gun shell revealed. http://wii.ign.com/articles/707/707077p1.html.

[23] PopCap Games Inc. Popcap games. http://www.popcap.com/.

[24] M. Jakobsson and T. Taylor. The sopranos meets everquest - social networking in massively multiuser networking games.

[25] USA Today Janet Kornblum. Teens hang out at myspace. http://www.usatoday.com/tech/news/2006-01-08-myspace-teens_x.htm?csp=34, 01.08.2006.

[26] Steve Jones. Let the games begin. http://www.pewinternet.org/, July 2003.

[27] Elina M.I. Koivisto. Supporting communities in massively multiplayer online role-playing games by game design. Technical report, Nokia Research Center, 2003.

[28] Tony Sun (2) Ling-Jyh Chen (1) and Yung-Chih Chen (1). Improving bluetooth edr data throughput using fec and interleaving. Technical report, (1) Institute of Information Science and (2) Department of Computer Science, (1) Academia Sinica, Taipei 11529, Taiwan and (2) UCLA, Los Angeles, CA 90095, USA, 2006.

[29] Wii Linux. Wiimote. http://www.wiili.org/index.php/Wiimote.

[30] Chronic Logic. Bridge it. http://www.chroniclogic.com/index.htm?bridgeit.htm.

[31] THX Ltd. Thx cinema design. http://www.thx.com/cinema/builtTHX/index.html.

[32] Matt Martin. Mobile is key to uniting multiple formats, says ea. http://www.gamesindustry.biz/content_page.php?aid=23501, 14.03.2007.

[33] Microsoft. Visual studio 2005. http://msdn2.microsoft.com/en-us/vstudio/default.aspx.

[34] Sun Microsystems. The java platform for consumer and embedded devices. http://java.sun.com/j2me/docs/j2me-ds.pdf.

[35] Sverre Morka. A flexible client for multiplayer on one screen entertainment system (mooses). Master's thesis, Norwegian University of Science and Technology, 2007.

[36] Netrek. Netrek. http://www.netrek.org/.

[37] Nokia. Nokia 5500 sport. http://www.nokia.com/5500.

[38] NRK. Newton homepage. http://www.nrk.no/newton/.

[39] Nvidia. Nvidia quadro plex vcs. http://www.nvidia.com/page/quadroplex.html.

[40] Federation of American Scientists. Land warrior. http://www.fas.org/man/dod-101/sys/land/land-warrior.htm.

[41] US Patent & Trademark Office. United states patent application: 0060062408:. http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.html&r=1&f=G&l=50&s1=%2220060062408%22.PGNR.&OS=DN/20060062408&RS=DN/20060062408.

[42] Pascal Pabr. Using the playstation 3 controller in bluetooth mode with linux. http://www.pabr.org/sixlinux/sixlinux.en.html.

[43] PopularMechanics. Microsoft surface: Behind-the-scenes first look (with video). http://www.popularmechanics.com/technology/industry/4217348.html.

[44] J. Preece. Online Communities: Designing Usability and Supporting Sociability. Wiley, September 2000.

[45] Java Community Process. Jsr-000082 java apis for bluetooth.

[46] Unique Promotions. Unique promotions homepage. http://www.uniquepromotions.no/.

[47] Victor R. Selby & Richard W. Rombach, H. Dieter Basili. Experimental software engineering issues: critical assessment and future directions, pages 3–12. Springer, first edition, 1993.

[48] Christian Schenk. About miktex. http://www.miktex.org/about.html, 2005.

[49] Sony. Sony 4k projector specs. http://pro.sony.com.hk/product/spec/brochure/srxr110_105.pdf, 2005.

[50] Morten Versvik Sverre Morka, Aleksander Spro. Multiplayer on one screen. Depthstudy, IDI, NTNU, Norway, 2006.

[51] Harmonix Music Systems. Guitar hero. http://www.guitarherogame.com/.

[52] Bluegiga Technologies. Bluegiga access servers. http://www.bluegiga.com/default.asp?f=2\&t=1\&p=1400\&subp=200.

[53] Tellu. Presentasjon arts seminar. http://www.abelia.no/getfile.php/Abelia%20Innovasjon/Elektronisk%20kommunikasjon/Tellu_presentasjon_ARTS%20seminar.pdf.

[54] Tellu. Tellu. http://www.tellu.no.

[55] Yu-Chee Tseng Ting-Yu Lin. Collision analysis for a multi-bluetooth picocells environment. Technical report, Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsin-Chu, 300, Taiwan, 2003.

[56] Morten Versvik. Moose system - media. http://www.mooses.no/static.php?page=static070430-003018.

[57] Alf Inge Wang. Forskningsprogrammet dataspill. url-http://www.ime.ntnu.no/forskning/prosjekter/dataspill, 26.04.2007.

[58] Wikipedia. Asheron's call. http://en.wikipedia.org/wiki/Asheron%27s_Call.

[59] Wikipedia. Civilization iv. http://en.wikipedia.org/wiki/Sid_Meier%27s_Civilization_4.

[60] Wikipedia. Everquest. http://en.wikipedia.org/wiki/Everquest.

[61] Wikipedia. Ultima online. http://en.wikipedia.org/wiki/Ultima_online.

[62] Wikipedia. Virtual community. http://en.wikipedia.org/wiki/Virtual_community.

[63] ZDNet. Motion-sensing comes to mobile phones. http://news.zdnet.com/2100-1035_22-6169697.html.