NTNU
Det skapende universitet

Multiplayer On One Screen
a framework for multiplayer games on one screen

SVERRE MORKA
ALEKSANDER BAUMANN SPRO
MORTEN VERSVIK

Master of Science in Computer Science
Prestudy *Submission date:* Fall 2006
Supervisor: ALF INGE WANG, IDI

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

Abstract

The goal of this depth study is to look into infrastructure to support a new concept; multiplayer on one screen (MOOS) games. This concept is especially well suited for cinemas which just recently were analogue but are rapidly becoming digital. The concept can also be used in other contexts such as arcade games.

The report starts by looking into previous/related work that is comparable to this new concept. We continue with evaluating technology that has to be used to make the concept possible. Further, we look into how we can make a framework which abstracts the required network support and gives possibilities for more features without making the development of games on the platform more difficult.

Another important part is what kind of games can be used in the concept. As there is currently not done any research into this work before, we describe what parts from different game genres fit in the concept, and come up with a game concepts that fits good.

To validate the principles of the framework, we made an implementation of the framework and tested it with all the required technology. Some pictures, a video from the session, a standalone binary of the game and source code is included.

This thesis is tied-up and should not be given or shown to any other person than the external examiner, Alf Inge Wang, Morten Versvik, Sverre Morka or Aleksander Spro without any permission granted by Alf Inge Wang, Morten Versvik, Sverre Morka and Aleksander Spro.

Problem Description

The task is to look at game concepts and solutions for electronic entertainment games, where multiple players play with/against each other on one big joint screen. One type of such a joint screen can for instance be a cinema canvas. The demonstrator shall be tested on a Sony 4K (digital movie) projector at Nova. This task is a collaboration between IDI and ITEM, and can be chosen by students from both institutes. Possible problem definitions can be:

- Mapping of different game types suited for the illustrated scenario.
- Development of a prototype game which demonstrates such a game concept.
- Looking into demands on a network to support such a game.
- Business models for such a game concept.

Assignment given: 2006-09-01
Supervisor: Alf Inge Wang, IDI

Preface

This depth study report is written by Aleksander Spro, Sverre Morka and Morten Versvik in the period from September 2006 to December 2006.

Acknowledgments

We would like to thank Alf Inge for this assignment and for his help and guidance throughout this project. We would also like to thank Bjarne Kjøsnes and Marlon Montejo Nielsen for technical support, and providing us with access to Nova Kinos facilities.

Trondheim, 19. December 2006

Sverre Morka Morten Versvik

Aleksander Spro

Contents

List of Figures

List of Tables

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 marked 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 limelight 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 marked. 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 outside showing movies. Cinemas have a lot of dead time during the day which they are now looking into using. Some business has opened open hosting conferences, while some are looking into other forms of entertainment.

This project will aim to look into what types of games are suitable in this scenario as well as design and implement a framework which can be used to host such games.

1.1 Problem definition

Development of games aimed for large-screen opens a world of new perspectives and approaches. First of all the game needs to support a multitude of players on the same screen, which means a specific genre of games needs to be developed. Our first task will therefore be to research into todays game genres to determine what might work in this scenario.

With a big number of players, a supporting network to handle input and output to the players is also important. Utilization of hand held devices for player input also requires a framework that handles location detection to determine which game the players are participating in (this is vital in cinemas with multiple auditoriums). A framework for this scenario also needs to handle login routines, including billing, between cinemas.

Our superior goal is to develop a framework for game-development for this platform. To support the development of such a framework will look at technology and solutions already made to see how we can utilize them and their experiences.

1.2 Limitations of Scope

Our superior aim will be focused on research and development on games for use in entertaining purposes. We will take into account appropriate game genres, technology, framework architecture, multiplayer solutions and limitations and constraints tied to our concept.

Our goal is to make a test implementation to give reflection on our framework suggestions. The test implementation comprise making server, client, and respective communication channels between them. Also we will make a simple game which use the attributes we find most useful.

We will not focus on extensions such as serious educational uses of the concept, peer-to-peer solutions for client communications and so on. We will also limit our focus on the business model, as this topic has been provided to another student.

CHAPTER 2

Project Context

The concept behind this project was conceived by Alf Inge Wang, whom works with projects looking at the technical aspects concerning video games. The projects are emphasized on several areas around video games:

- Peer-to-peer videogames on cellular phones
- Videogames and learning
- Historical development of videogames
- Videogames on large screens (Our project falls into this category)

In cooperation on this project, the institute of telematics is looking at networking- and payment issues related to these types of games.

Midgard Media Lab, specifically the the Av-Arena Norway project[2], has helped with the technical aspects of this project. They have made it possible to test the framework and the test game at Nova kino in Trondheim, as well as supply needed hardware to do so.

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 types for the domain Should read Chapters 1 and 4. Part III outlines central concepts and an in depth study into the domain.

Readers interested in the design of a framework for the domain Should read Chapters 1, 4 and 5 as well as Part III (to get an understanding for the motivations in design). Part IV presents the architectural choices and views.

Developers interested in improving the framework Should read and understand Part IV and Part V. Both of these parts present essential information on how the framework is designed and its possibilities. Developers might also want to read Part VII.

Developers interested in designing games for the framework Should read and understand Part V. This presents the framework implementation and a small howto on game and game client design needs. Part IV might also be of interest.

3.1 Chapter Description

This section outlines the chapters in the report.

➢ **Part I - Introduction**

Chapter 1 - Problem Description This chapter describes a more detailed look at the problem at hand along with a problem definition and limitations of scope.

Chapter 2 - Project Context This chapter describes the context of the project.

Chapter 3 - Reader's Guide This chapter presents an outline of the report describing the chapters.

➢ **Part II - Research Methods**

Chapter 4 - Research Questions & Methods This chapter described the research questions that is derived from the problem definition. To answer these we need a research method and a development method.

Chapter 5 - Development Tools & Software This chapter presents the tools and software used to create this report and the framework.

➢ **Part III - Prestudy**

Chapter 6 - Central Concepts This chapter deals with the central concepts related to our problem domain. These concepts should be understood in order to grasp the background and intention behind the framework.

Chapter 7 - Previous Work This chapter looks into previous work found concerning the domain.

Chapter 8 - State of the Art This chapter presents an in depth study into games genres of today.

Chapter 9 - Technology This chapter presents an update on relevant technology.

➢ **Part IV - The MOOSES Framework Architecture**

Chapter 10 - Architecture This chapter presents an introduction to the MOOSES architecture and the stakeholders of the system.

Chapter 11 - Requirements This chapter describes the requirements for the MOOSES framework.

Chapter 12 - Design Decisions In this chapter we presents the main issues we met during development, and how we dealt with them.

Chapter 13 - Design Overview This chapter presents the actual design of the new framework presented through different architectural views.

➢ **Part V - The MOOSES Framework**

Chapter 14 - Framework Design & Implementation This chapter presents framework implementation specific information.

Chapter 15 - Test Game Implementation This chapter presents test game implementation specific information.

Chapter 16 - Game Developer In this chapter we present a small description of how to develop games and game clients for the framework.

- **Part VI - Testing**

  Chapter 17 - Testing In this chapter we present our testing of the framework.

- **Part VII - Conclusion**

  Chapter 18 - Conclusion Here we summarize and find answers for our research questions.

  Chapter 19 - Further Work Here we present possible ways to expand the framework.

- **Part VIII - Appendix**

  Framework Source Code

  Testgame Source Code

  Framework javadoc

  Testgame source documentation

  Final test video

- **Part IX - Bibliography**

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 follow.

4.1 Research Questions

We want to provide some research into determining what type of games will work on a single large screen like a projection-screen. Development of games aimed for large-screen opens a world of new perspectives and approaches.

The huge amount of players on one screen opens new perspectives for development of video-games, and there is quite a few issues to cover. For instance we have to find solutions to provide good game-play to each player. Therefore we have to find solutions to how we can make the most of the large-screen to provide room for a clear visualization for each particular player.

The concept also face a great pile of technical difficulties. It's given that one computer alone will function as a server, and granted that it has to process instructions from as many players as possible and render to a display at more or equal to 1920x1080 pixels, it has to be optimized to the max. To provide a sound feedback to increase the gameplay for each player also gives a problem, because feedback from fifty players on one speaker system may just sound like a load of pebble granules. Range of Bluetooth signaling, bandwidth and processing may also cause problems.

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 games are suitable for use on large projection-screens in multiplayer mode?**

- ➢ We will compile a research to determine what makes up the different game genres of today. This research will then be discussed to determine what game attributes are best suited, how many players we should be able to host and what visualization methods work with that number of players.

a) What game attributes would be best suited for this scenario?

b) How many players will such a game be able to host without becoming chaotic?

c) What visualization types are best suited to host a great number of players?

RQ-II: **Is it possible to create a back-end framework to ease the development of games in this scenario?**

- ➢ With the in depth look at what games would work, we will design a back-end framework to ease the development of games in this scenario. An architectural design of the framework chosen will then be presented, implemented and tested.

a) What role(s) would such a framework fulfill?

b) How would the framework ease the development of games for the scenario?

c) Would the framework ease the implementation of the scenario?

Our goal is that the games should mainly be based on multiplayer opportunities supporting about thirty to fifty players. All the players will play on the same screen at the same time, and send instructions through a hand held device. The idea is to use cell-phones with bluetooth technology for the transfer of instructions.

In the first run we will research and discuss which game-types will fit for the concept. Further on we will look at technology and solutions already made, to see how we can use them. Our superior goal is to develop a framework for game-development for this platform.

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 [3] 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 [3] 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.

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.

The empirical approach will constitute the base research method for resolving the game related research questions, while the engineering approach will constitute most of the work behind the framework development.

4.2.1 The Engineering Approach

In Section 4.2 above, we described the engineering approach as; observing, suggesting improvements, analyzing and building. The observation phase consists of the Part III (Prestudy) where we will look into game concepts and aspects of the intended framework as well as some new technology.

With the main focus of this project being to design a framework for use of games on large projection screens, we will incorporate the last two stages of the engineering approach into the requirements, design and implementation chapters of Part V (The MOOSES Framework). The last part of the engineering approach, suggesting improvements, will be found Chapter 19.

When starting this study, we looked into some of the development process standards. We quickly decided that both RUP[1] and UP[2] might be a bit to heavy weight for a project of this scale, so we opted for the use of an adapted version OpenUP/Basic [1]. OpenUP is designed for projects sizes of 3-6 developers over a time frame of 3-6 months. The iterative and incremental process of analyzing and building described in the Engineering approach, is a significant part of OpenUP, though shortened down to fit the smaller time frame and project size.

The Open Unified Process / Basic

As described above we decided to use the OpenUP development process and we will in this section describe what the different elements and roles pose for this project.

In [1] the OpenUP/Basic process is described as:

> ”. . . an iterative software development process that is minimal, complete, and extensible”

It is characterized by four mutually supporting core principles:

➢ Collaborate to align interests and share understanding

➢ Balance competing priorities to maximize stakeholder value

➢ Focus on articulating the architecture

➢ Evolve continuously to obtain feedback and improve

OpenUP/Basic is an iterative process distributed throughout four phases: Inception, Elaboration, Construction, and Transition. Each phase consists of one or more iterations, where stable, working versions of the software are developed and released. The completion of each iteration represents a minor milestone for the project and contributing to the successful achievement of the Phase's major milestone (as seen in Figure 4.1).

The content of OpenUP is organized around four major areas and between six different roles as depicted in Figure 4.2.

4.2.2 The Empirical Approach

As mentioned in Subsection 4.2.1, we will use the empirical approach to research different game concepts in order to determine what game types will work in our domain. We will use

[1] Rational Unified Process
[2] Unified Process

an adapted form of literature study to categorize the more popular game types today. With this data we can determine what elements will work with our concepts, and which to stay away from. This will also help in spite ideas for game concepts for our domain.

We have chosen to categorize the different game types into the following sections:

Common features will try to summarize the common basic gaming elements the given genre make use of.

Types will try and summarize the given context and main concepts in the given genre.

Multiplayer solutions will try to look at how the given genre makes use of multiplayer gaming.

Common platforms will look at the different hardware setups the genre makes use of.

Solutions for our concept will look at what elements from this given genre can make good solutions for our concept.

When the detailed look at the game types is done we will summarize our findings to what is

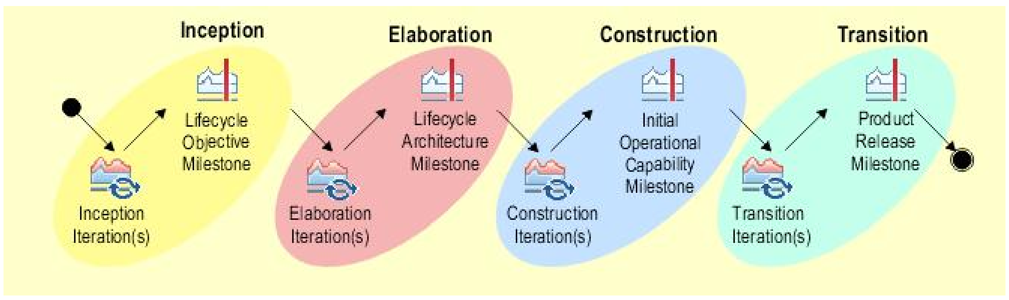

Inception
Elaboration
Construction
Transition
Lifecycle Objective Milestone
Lifecycle Architecture Milestone
Initial Operational Capability Milestone
Product Release Milestone
Inception Iteration(s)
Elaboration Iteration(s)
Construction Iteration(s)
Transition Iteration(s)

Figure 4.1: The OpenUP/Basic lifecycle from [1]

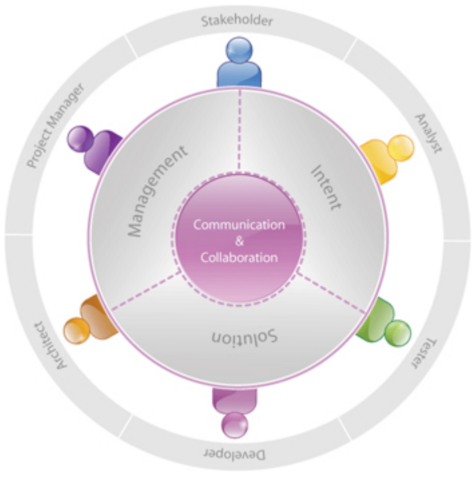


Figure 4.2: Four areas of content organization for OpenUP/Basic from [1]

best suited for our concept, and some possible loopholes which one should look out for. We will also come with some suggestions to workable concepts within the domain.

4.3 Test Environment

Since we are using an iterative development process, we will perform testing for each increment of the framework. The testing will reveal if our solution was satisfactory. The testing will be performed on the emulators on desktop computers and on mobile phones. At certain peak points in the development process the framework will be tested at the main auditorium at Nova with real cellular phones. This to make sure the framework performs acceptably in an optimal setting.

CHAPTER 5

Development Tools & Software

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

5.1 Development Tools

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

5.1.1 Eclipse With Plugins

Eclipse [4] 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 [5].

EclipseMe [6] is a plugin for developing J2ME MIDlets in Eclipse. A Java Wireless Toolkit must be installed on the system for the plugin to work.

5.1.2 MiKTeX

MiKTeX is an implementation of TeX and related programs for Windows on x86 systems [7]. The MiKTeX distribution contains many features including the pdfTeX compiler which generates a pdf document. The compiler is used to produce this document.

5.1.3 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.

5.1.4 Microsoft Visual Studio

Visual Studio 2005 [8] is Microsofts 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 test game in C++ with OpenGL library support.

5.2 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. Therefore, 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 the Sony Ericsson SDK.

5.2.1 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 [9]. 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 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.

Part III

Prestudy

CHAPTER 6

Central Concepts

In this chapter, we will present central concepts related to the MOOSES framework.

6.1 Gaming controllers

Game controllers are used to govern the actions and/or movements of elements within a video game. The type of element controlled depends on the game. Through the years, several types of game controllers have emerged.

GamePad is a type of game controller held with both hands where the thumbs are used to provide input.

Paddle is a controller that features a round wheel and one or more fire buttons.

Trackball is basically an upside-down mouse that is manipulated with the palm of one's hand.

Joystick is a computer peripheral that consists of a handheld stick that pivots about one end and transmits its angle in two or three dimensions to a computer.

Steering wheel usually consists of a steering wheel, power and break paddles and a stick shift.

Keyboard & mouse are typical input devices for a personal computer and are currently the main game controllers for computer games.

Light gun is a peripheral used to "shoot" targets on a screen. They usually roughly resemble firearms or ray guns.

Touch screen is an input device that allows the user to interact with the computer by touching the display screen.

The game controllers are often subjected to hard abuse from gamers ”interacting” too much with the game, and the hardware producers obtain a lot of their revenue by the sales to them. Most of the controllers are proprietary of the game system they belong too, meaning you would need to acquire a license in order to use them for other systems.

6.2 Bluetooth

Bluetooth is a radio communication protocol which lets devices using the protocol to easily communicate with one another. The protocol is designed for low poor cost, and is therefore found in almost all mobile devices made today. The Bluetooth protocol is explained in detail in Section 9.4.

6.3 State machines

Finite state machines is an old technique which originated in the field of mathematics, it was then used for language representation. FSMs consists of discrete states which define behavior and may produce actions based on input events. When a FSM receives input in the form of messages, it will typically execute code to handle that message and choose to transition to a new state or keep the same.

There are many advantages and disadvantages in using FSMs, [10] describes FSMs for AI development. We will only use deterministic state machines as we want predictability. We have included relevant pros/cons from the article here:

6.3.1 Advantages of FSM

- ➢ Their simplicity make it easy for inexperienced developers to implement with little to no extra knowledge (low entry level)
- ➢ Due to their simplicity, FSMs are quick to design, quick to implement and quick in execution
- ➢ FSM is an old knowledge representation and system modeling technique, and its been around for a long time, as such it is well proven even as an artificial intelligence technique, with lots of examples to learn from
- ➢ FSMs are relatively flexible. There are a number of ways to implement a FSM based system in terms of topology, and it is easy to incorporate many other techniques

- Easy to transfer from a meaningful abstract representation to a coded implementation
- Low processor overhead; well suited to domains where execution time is shared between modules or subsystems. Only the code for the current state need be executed, and perhaps a small amount of logic to determine the current state.
- Easy determination of reachability of a state, when represented in an abstract form, it is immediately obvious whether a state is achievable from another state, and what is required to achieve the state

6.3.2 Disadvantages of FSM

- The predictable nature of deterministic FSMs can be unwanted in some domains such as computer games (solution may be non-deterministic FSM).
- Larger systems implemented using a FSM can be difficult to manage and maintain without a well thought out design. The state transitions can cause a fair degree of "spaghetti- factor" when trying to follow the line of execution
- Not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. This means that all states, transitions and conditions need to be known up front and be well defined

CHAPTER 7

Previous Work

MTV described a game design contest [11] by New York's Parsons design school where the contestants were asked to come up with and make a game for mobile phones which should interact on a big screen. Atari and Glu Mobile were to pursue the results further, but we can't find any evidence of this.

Another “big-screen" project was using the lights in a building as a big playground [12]. Project Blinkenlights made it possible to control some simple games by sms messages or to send scrolling messages on the building.

Manchester and Liverpool had a big screen game where cities could compete and simulate rugby match on a big screen [13]. Another test at the same places was a big tv screen controlled by bystanders via bluetooth [14] where they interacted in a simulated 3d world.

We have not found any other similar game concepts or more mentions of the above games, so we are making new ground.

CHAPTER 8

State of the Art

8.1 Issues regarding multiplayer games

Multiplayer games has existed since the start of video games in the late 1950s. The first games were limited to one versus one playing, but as computing power grew, more complex games arrived to add more players to the mix.

Multiplayer games span over several genres and visualization methods in today's selection of games. By careful screening of these games we should get a good overview of what is doable and not in regards to the task at hand. The first games that supported multiplayer where based on players competing each other. As the games advanced new perspectives of multiplayer where born. We give a brief presentation of the most common perspectives below.

8.1.1 Competition

Competition has been element in multiplayer as long as it has existed, and the very first arcade game was based on competition. Competition can vary from two players just beating each other in a game, or games with several players comparing high scores.

- ➢ Extensive use of highscore lists
- ➢ In-game messages of who beat who gives a taunting effect
- ➢ Beating the opponent

8.1.2 Cooperation

Cooperation is a mode of multiplayer where players work together to solve the goal or defeat computer controlled opponents. Cooperation is often combined with competition, dividing players into competitive teams.

Cooperation in multiplayer usually involve either to complete the game together, defeat the other team(s) or just play as friends against the computer.

- ➢ Cooperate against the computer
- ➢ Cooperate against other cooperating teams
- ➢ Solve ”singleplayer game” together

8.1.3 Socialization

Socialization is an important element in both competitive and cooperative multiplayer games, but many games emphasizes this utility alone. It is often done in virtual fictional worlds where players can come together to talk and share things. For example **The Sims Online** where players control one or more characters in a fictional world, and are able to communicate and share actions.

Popular games gains communities consisting of people that have interest in the game, allowing them to share info and software related to the game. Especially the MMORPG games has great emphasis on sosializing, and players often make communities inside the game rather than just outside it. One of the biggest MMORG today is the populare **World of warcraft**, which gives the players opportunities to join clans, do objectives together, share items, communicate and so on.

- ➢ In-game chatting
- ➢ Out of game chatrooms
- ➢ Community forums

8.1.4 Platforms

The first versions of multiplayer games that were made public available where games developed to the arcades. The gameplay were constrained for all players within one screen. As multiplayer gained popularity and the computers became common property, the visualization techniques evolved. We will have a look at our options.

Single system multiplayer

In modern console games and arcade games, the term multiplayer usually implies that the players play together by using several controllers plugged into the game system which is hooked up to a single television monitor. For home console games, developers usually use split-screen so that each player can have an individual viewpoint of the action (important for genres such as the first person shooter), although most arcade games and some console games (ranging from the seminal Pong to the ever-popular Bomberman) make use of a single play area for all the players.

As many game consoles now support online or network games, split-screen is often supported in combination with these multi-system modes. For example, in a network or Internet game of Halo 2, up to four players may be playing in split-screen on each console in the network.

Single-system games may also involve several gamers taking turns playing a game on the same system using the same input devices. In PC gaming, a multiplayer game where the players share a computer is usually called hotseat.

Distributed Multiplayer

In modern computer games, the word multiplayer usually implies that the players play together by connecting multiple computers via a network, usually either a LAN or the Internet.

LAN games might be the most enjoyable because it gives are more subjective appearance of your opponents/team-mates, and it eliminates the problems common in Internet games such as lag and rude anonymous players. However, online multiplayer games have been the focus for game developers the later years. Some networked multiplayer games do not even feature a single-player mode.

This category of games currently requires multiple machines to connect to each other over the Internet, but before the Internet became popular, MUDs were played on time-sharing computer systems, and games such as Doom were played on a LAN. Spacewar!, created in 1962 for the PDP-1, is credited with being the first multiplayer computer game.

Making multiplayer distributed introduced a new problem, latency or *Lag* as it is usually called.

Gamers often refer to latency by the term ping, which measures round-trip network communication delays (by the use of ICMP packets). For example, a player on a DSL connection with a 50 ms "ping" will be able to react faster to game events than a modem user with 350 ms average latency. Another popular complaint is packet loss and choke, which can render a player unable to "register" their actions with the server. In first-person shooters, this problem usually manifests itself in the problem of bullets appearing to hit the enemy, but the enemy taking no damage. Note that the player's connection is not the only factor; the entire network path to the server is relevant, and some servers are slower than others. While latency is frequently complained about, lack of finesse and decent tactics is probably more lethal than a slow connection in most games. Major and frequent variations in latency, however, can be another story; these can make it very difficult to properly play the game.

Recently, games consoles have also begun to support network gaming, over both the Internet and LANs. Many mobile phones and handheld consoles also offer wireless gaming through Bluetooth or similar technologies.

8.1.5 Visualization tecniques

The different visualization methods give big differences in game play and constraints. As our concept obvious has to be based on single screen, we might scratch the surface of other visualization methods as well. All the user controls will have their own display, therefore we might move parts of the game play over the control device (As with Nintendo DS).

Given that we may use a hybrid of visualization methods we will give a brief presentation and a utility comparison of the different known approaches.

All on one screen

The single screen confine the area to play. The area is usually a bounded rectangle viewed from above or a perspective angle from the side or corner on 3d and isometric games. Some games hold focus on one object, and may display players if the object is close to them. For example a football game focusing on the ball, and if the ball gets near a players position the player is within the canvas.

Splitscreen

While splitscreen gives the advantage of letting people play together with limited needed hardware, it also has is downsides. By dividing the screen up in the number of players, the scope for each players visibility is reduced by the same amount. When players start reaching 3+ this greatly reduces the immersiveness for the players in the game, which can often dampen the fun factor.

On a strategic level, each player will also be able to see what the others are doing and/or where they are, and be able to use that to his/her advantage. This reduces the strategic elements of the game.

Network games with personal terminals

As computer became cheaper and network technology grew stronger, the multiplayer games were developed to fit the distributed networks. As mentioned the first games supported multiplayer over local network, but has later evolved to fit the World Wide Web.

Each player plays the game on his computer and is either connected to a server, or hosts the server, and all the other players are connected through this server. The server keeps track of every players posisiton and current status. The player has his own point of view, and will see the other players when they are in the same area.

8.1.6 Comparison

Our concept has to take a few steps back in time with respect to visualization techniques. As mentioned, the first multiplayer games let the players cooperate or compete on the same screen. Our consept needs to focus on this utility mainly, although we have the opportunity to visualize elements on the controller-screen. We will compare the three visualization techniques mentioned to provide some pros and cons regarding the artefacts mentioned above.

- ➢ Socialization
  How good does the tecnique reflect opportunities for socialization

- ➢ Tactic 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
  Reflect the techniques abillity to provide airspace and focus on players character.

- ➢ Latency
  As mentioned, multiplayer might step onto the problem of latency that might lower the fairness and fun factor.

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

- ➢ Game play limitations
  The visualization tecnique might constrain the opportunities for a good gameplay.

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

As the comparison shows (see Table 8.1), single screen visualization has some setbacks. Some of these setbacks, 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 bondaries, 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	Terminals
Socialization	Good socialization, all in same room		Depends on whether the game is Lan or internett. Internett limits sosialization.
Tactics usage	Impossible to conceal your tactics		Full tactic usage
View	Ok view for few players	Bad view that gets worse for each player added	Excelent view oppotunities
Latency	No Latency		Internet play may give huge latency
Freedom	Limited, has to show all players in one frame	Good	Good
Game play limitations	All using the same screen gives limitations	Weaker Game play limitations, every player is free	No limitations
Hardware requirements	Usually limited to conssole and screen		Expensive

Table 8.1: Visulization techniques comparison

8.2 Game Genres

In this section we will have a look at different video-game genres to help us get some ideas on which concepts can work properly on the big-screen. We will give a brief description of the genres that has provided the backbone of games throughout history. We will mainly discuss the most important genres and the genres that will be partly sufficient to our concept, but we will also have a look at the most popular and important sub-genres.

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 collections of own suggestions. Later on, it will either be placed in a genre, or define its own genre.

We will present the genres in the following pattern.

➢ 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
  Discuss 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 had an impact on which platform were most appropriate for the genre.

- Solutions for our concept
  Gives a brief discussion of whether the genre is suitable to the multiplayer single screen concept. Also we take into account features and attributes that we may take from the genre.

- Examples and screenshots
  Provides screenshots, with a brief description, from popular games from the given genre or it's sub-genres.

8.3 Platform

Platform is a videogame genre where the main character has to climb up or down, jump from or to platforms and ledges, often in the chase of so called power-ups. Power-ups are elements in the game used to gain power, health, weapons etc. Platform games are commonly based on cartoon-like graphics emphasizing an unrealistic nature. The characters of such games are usually based on unrealistic fairytale creatures such as gorillas throwing barrels, dragons or ghosts. In the early beginning, these games moved in a vertical direction, viewing the characters in a profiling angle. The levels were confined to static environments fitted on the screen, following a linear progression. Soon level advancement changed from being mainly vertical to largely horizontal, as well as introducing multiple screen-width spanning scrolling environments. As the game-developers started using 3d, this genre had to follow. The 3d platform games introduced some great changes to the whole genre. In the traditional 2d era the player had to reach a single goal to complete the level. The 3d environments made the player able to break free from the linear construction on the levels into levels that consisted of multiple goals and objective before it was conquered. For instance collect all instances from a certain object etc.

8.3.1 Common features

The different platform games usually differs only in the form of minor changes to gameplay and visual elements such as appearance of the character and levels. Therefore most of the platform games has the following elements in common.

- Cartoon like graphics

- ➢ Static, Linear level design
- ➢ Simple background story
- ➢ Fairytale-like characters
- ➢ Climbing, jumping between platforms
- ➢ No introduction needed
- ➢ Controls are limited to just a few buttons

8.3.2 Types

Some platform games take adventure or action elements into the game-play providing a more advanced game-play. However, these games often differs to much from their origins and fall inn under other genres. Different genres of platform games is therefore most commonly divided into their visualization techniques.

Single screen platformers

The whole level is viewed as one screen, and level advancement loads another screen.
Example **Mario Bros** and **Donkey kong**

Vertical platform/Horizontal platform games

On this genre the character is often viewed in the horizontal or vertical center, and the level scrolls to the opposite side of the players movement. Some games support scrolling in both directions. The genre often consist of a world where the player moves between different screens.
Examples: **Super Mario Bros** and **Prince of Persia**

Isometric platformers

Isometric platform games present a three dimensional scene by compositing two dimensional graphics that display the world with a fixed camera orientation and without perspective.
Examples: **Knight Lore** and **Congo Bongo**

3d platform games

Platform games in a 3D perspective, usually viewing the character from behind. This version changes the gameplay slightly from the simple one provided by the other genres since the player

can move in all directions.
Example: **Super Mario 64** And **Chrash Bandicot**

Platform Adventure

Platform adventures have inherited the gameplay of the platform games, but includes typical adventure game-play elements such as puzzle solving, character interaction and better emphasis on story.
Examples: **Jack and Daxter** and **Tomb Raider**

8.3.3 Multiplayer solutions

This genre has never been particularly based on either competition or cooperation. Although some titles, mostly on single screen or isometric, have made solutions where two players start in each corner trying either to help each other reach the goal or be the first one to it. The other types typically use turns to alternate which player who are currently playing. It also exist versions that allow two players at the same time, using the two or more characters as boundaries for the scrolling, but this reduces the fun factor since the players are dependent on the other player(s) to move forwards. Platform multiplayer games are therefore usually limited to two players and no positive effect on gameplay.

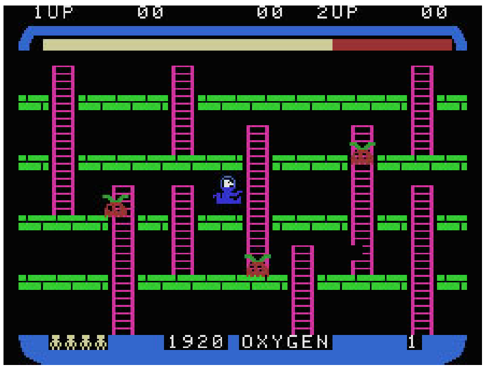
8.3.4 Common platforms

Platform games are typically found on consoles and arcades, and especially on hand held consoles, but they have been introduced to PC as well. Their simple construction and easy learning curve has made them very popular for young and ordinary players. Also the games are an easy fit to the limited controls of the consoles.

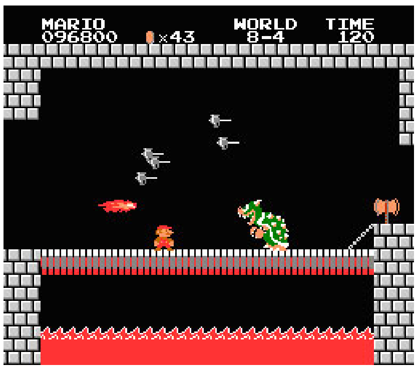
8.3.5 Solutions for our concept

Although the platform genre dose not provide any sufficient multiplayer solutions, it provides a good easy to learn- and instant gameplay. Games developed to our concept could take advantage of using platforms to divide map into sections and support a gameplay that dose not have to divide the screen. Also the isometric projection introduced mainly by platform games might give a good alternative to visualize third person perspective for multiple players on the same screen.

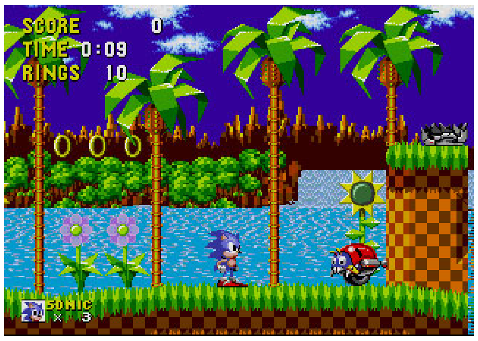
8.3.6 Examples and screenshots from various platform games



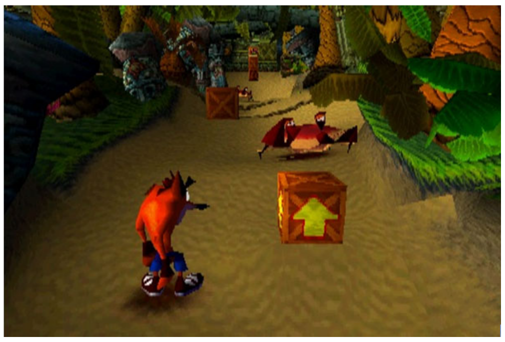
(a) Space Panic, Universal 1980, officially the first platform game ever



(b) Super Mario Bros, Nintendo 1985. was one of the first side-scrolling platform games of its kind



(c) Sonic the Hedgehog, Sonic team 1991



(d) Crash Bandicoot, Naughty dog 1996

Figure 8.1: Screenshots from different platform games

8.4 Adventure games

Adventure is a genre of video games typified by exploration, puzzle-solving, interaction with game characters, and a focus on narrative rather than reflex-based challenges.

The adventure genres focus on story allows it to draw heavily from other narrative-based art forms, such as literature and film. Adventure games encompass a wide variety of literary genres, including fantasy, science fiction, mystery, horror, and comedy.

Adventure games have a lot in common with Computer role-playing games (discussed later), except that the game play is more focused on the problem solving rather than combat and statistics, and the Adventurers are typically fixed and static while RPGs yield totally freedom of movement and objectives.

This genre dominated the eighties and the early nineties but has been declared deceased in the later because their market share drastically declined, as action based games took a greater share of the market, particularly first person shooters that feature strong, story-structured solo games. This slump in popularity led many publishers and developers to see adventure games as financially infeasible in comparison. However the genre lives on, it has just changed slightly from its origins and now takes forms as a hybrid with the other genres.

8.4.1 Common features

We will have a look on elements that is common for most of the adventure games and their sub-genres.

- Collect items, For making logical combinations or solve puzzles
- Strong focus on story and atmosphere, and introduction to characters
- Help or steal from people
- Non-linear level design
- Fairytale-like music themes
- Limited freedom
- Items are often used in absurd situations far from their original intended purpose
- Simple and easy learned controls
- Use of humour to strengthen fun factor

8.4.2 Types

There are many types of adventure games, depending on the criteria. Adventure games vary in their subject, interface, setting or plot. A definite categorization can not be done since some of them may belong to 2 or more of the types mentioned below.

Text based

The first adventure games to appear were text adventures which typically use a verb-noun parser to interact with the user.
Examples: **Zork** and **Mystery house**

Graphical adventure

Graphical adventure differs from textbased adventure by introducing visual elements to represent the atmosphere. Although graphic adventure where text based in the beginning, it later introduced the well known point and click interface which introduced a new era to the adventure genres.
Examples: **Kings Quest** and **Monkey Island**

Action adventure

Action adventure genres differs from the traditional adventure games by the focus on action elements rather than storyline and puzzles. Typically the story will be represented by breakpoints between action segments like real time combat or driving etc. Action adventure has been dominating the titles, with the result of many just referring to them as adventure games.
Examples: **Legend of Zelda** and **Resident Evil**

RPG-like

Some adventure games rely equally on the common adventure elements, but also on the "character building" of RPGs. The main character(s) usually has a certain "Hit point" meter and a chart of skills. Some puzzles and features need a minimum amount of skills in order to be solved (like Climbing above 5 to climb a tree and obtain a lost ring) so the player may have to choose one character over another to solve it, or spend time building the skills of the first character. As in RPGs, the games involve battles, the result of which depends on his characters skills and health (and on the players reflexes in the case of real-time combat). However, these kinds of games do not belong to the "Action adventure" above.
Examples: **Grand Theft Auto San Andreas** and **Diablo**

Puzzle adventure

Adventure games that do not rely on obtaining items, their use, and character interaction belong to this genre. It emphasizes exploration, reading logs, and deciphering the proper use of complex mechanisms, often resembling Rube Goldberg machines. The plot of these games is usually obscure, and relies on the players interpretation of the setting and the scenery, and information from the logs in order for him to understand the background scenario. Almost all of these games are played from a first person perspective with the player "moving" between still pre-rendered 3D images, sometimes combined with short animations or video.
Examples: **Myst** and **D**

8.4.3 Multiplayer solutions

Most adventure games are designed for a single player, since the heavy emphasis on story and character makes multiplayer design difficult. Some titles makes it possible to cooperate throughout the game using LAN or online multiplayer solutions.

8.4.4 Common platforms

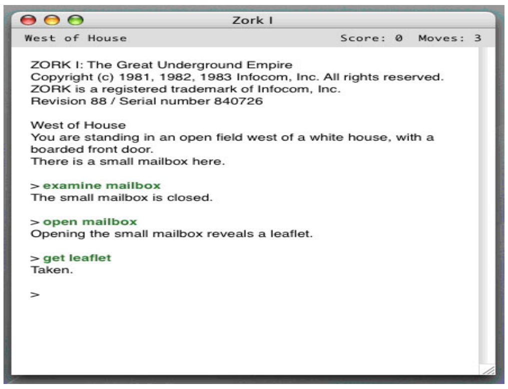
The vast majority of adventure games are computer games, though console-based adventure games are not unheard of.

8.4.5 Solutions for our concept

One superior strength of the adventure genre is the ability to spellbind the players to play several hours. The catchy story together with the fun and engaging gameplay makes it a hit. Although this game type does not support multiplayer well, at least not for many players, there are some elements we could grab onto.

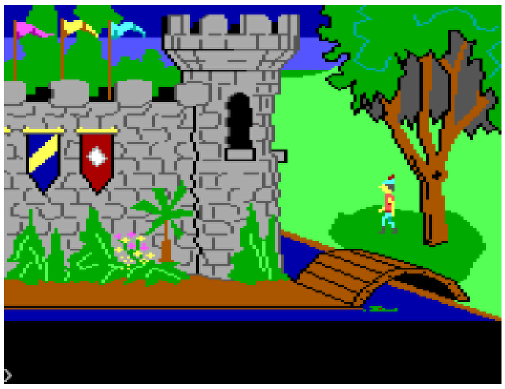
People in a cinema are expecting to be entertained by a fetching story, we could use this to ignite the players. The story would probably have to be based on cooperation, so people could help each other solve puzzles, and cooperate through out the story to the end. However, this sets a limitation on the amount of players together with challenges on how to display the game.

8.4.6 Examples and screenshots from various Adventure games

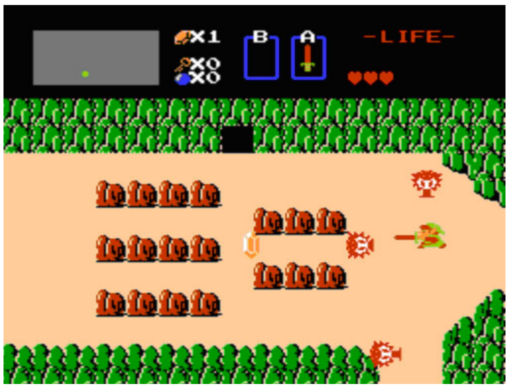


```
Zork I
West of House                                    Score: 0   Moves: 3

ZORK I: The Great Underground Empire
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west of a white house, with a
boarded front door.
There is a small mailbox here.

> examine mailbox
The small mailbox is closed.

> open mailbox
Opening the small mailbox reveals a leaflet.

> get leaflet
Taken.

>
```

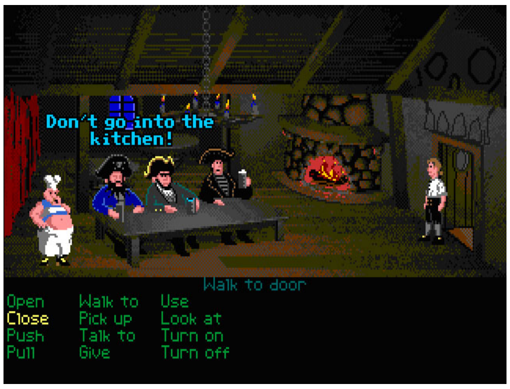
(a) Zork, Colossal Cave Adventure1979, was one of the first interactive fiction computer games and an early descendant



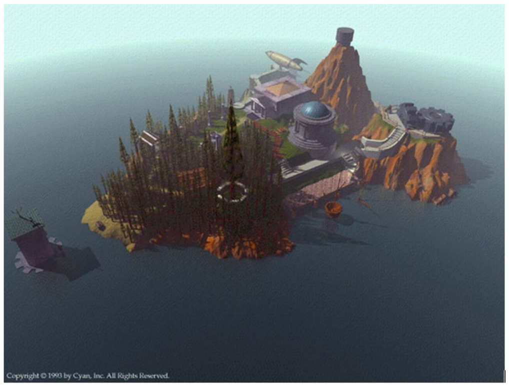
(b) King's Quest, Sierra 1984, one of the first graphical adventures, still holding on to text based input



(c) The Legend of Zelda, Nintendo 1986, is considered to have defined the action adventure genres



(d) The secret of monkey island, LucasArts 1990, introduced the point and click interface



(e) Myst Cyan inc. 1993. Puzzle adventure

Figure 8.2: Screenshots from different adventure games

8.5 Computer Role Playing game

Computer role playing games (CRPG) are originally derived from traditional role-playing games and use both the settings and game mechanics found in such games. The stories featured usually involve a group of characters who have joined forces in order to accomplish a mission or ”quest”. Along the way, the adventurers must face a great number of challenges and enemies (usually monsters inspired by science fiction and classic mythology). Characters have a variety of attributes such as hit points. These attributes are traditionally displayed to the player on a status screen as a numeric value, instead of a simpler abstract graphical representation, such as the bars and meters favored by video games in general. These attributes are to be upgraded during the game as the character grow stronger and gets smarter. Number of detail level on attributes varies from game to game usually between 5 to 100. Many games also use level based solutions to represent the character. The character has to gain a certain level to achieve, use or solve artifacts. The player has freedom to make his character as he want. Some titles even take personality into account and lets it affect the game play, such as how people will withhold you. Just like the adventure games, the character(s) has to collect certain items and artifacts for use in the game. In CRPG these items usually serve multiple ways to be used. An important element of CRPG is the possibility to make choices that will affect your progression later on. You will typically have a dozen paths to complete the quest. The themes are commonly based on one big world consisting of areas that the player can traverse in the sequence he chose.

8.5.1 Common features

These features are associated with the traditional CRPG. Modern games tend to use elements from the CPRG genre, but are not CRPG. These elements should be featured in most CRPGs.

- Focus on story
- Interaction with other characters
- Character development
- Themes
- Freedom to choose
- One big world, rather than level based
- Exploration
- Combat of some sort
- Sequence is not linear
- Takes a great amount of time to get into
- Multiple endings
- Building the character during the game

8.5.2 Types

Traditionally the computer based RPG is singleplayer and has a lot in common with the adventure genres. The current development is giving the players more freedom, ability to create and form the character, more focus on combat, and using skills and personality to achieve the subgoals along the quest. CRPGs has followed the same development path as the adventure genres, starting with only text based games and to more modern age graphics with huge worlds and endless possibilities.

Massively-Multiplayer Online Role-Playing Game

(MMORPG) is a type of online computer role-playing game (RPG) in which a large number of players interact with one another in the same virtual world. As in all RPGs, players assume the role of a fictional character (traditionally in a fantasy setting) and take control over most of that characters actions. MMORPGs are distinguished from single-player or small multiplayer RPGs by the games persistent world, usually hosted by the games publisher, which continues to exist and evolve while the player is away from the game.

MMOSG (Massively Multiplayer Online Social Game)

is a category of Massively Multiplayer Online Games that focuses on socialization instead of objective-based gameplay. MMOSG may also represent Massively Multiplayer Online Sports Game (example of Shot Online) or Massively Multiplayer Online Strategy Game

Tactical RPG

8.5.3 Multiplayer solutions

As mentioned the MMORPG involves a number of players competing and cooperate in an endless game. Other solutions are to make players cooperate throughout the game. Combining their skills to achieve the goals, artifacts and skill points.

8.5.4 Common Platforms

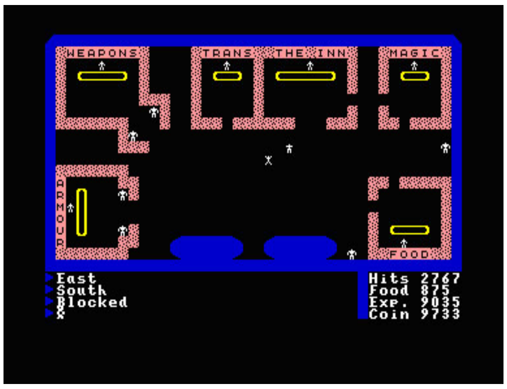
Modern CRPGs are complex and huge which make them hard to implement on most of the consoles. Therefore the CRPGs are usually developed to the computer platform, which also support online facilities, with an exception of Xbox 360 which is powerful enough to support these kind of games and also support online gaming. Lighter CRPGs has found their way to consoles, for instance the Final Fantasy series.

8.5.5 Solutions for our concept

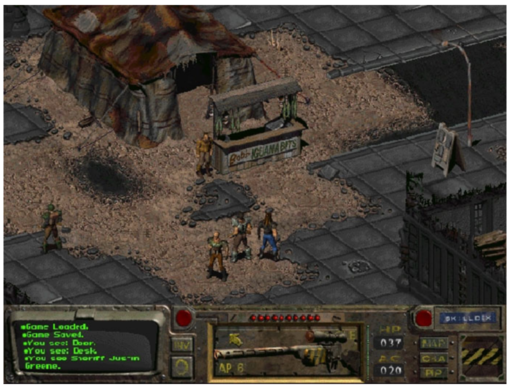
Elements from the RPG genre is desirable to our concept due to its ability to enchant players. Adopting RPG elements could be done, so that the player could store his character with current statistics, and maybe even interact with it through Internet (like buying upgrades) when not playing.

We can not make an advanced RPG gameplay due to the concepts limitations, but RPG elements are clearly an option.

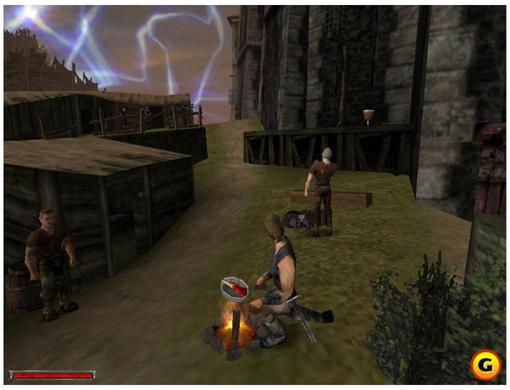
8.5.6 Examples and screenshots from various CRPGs



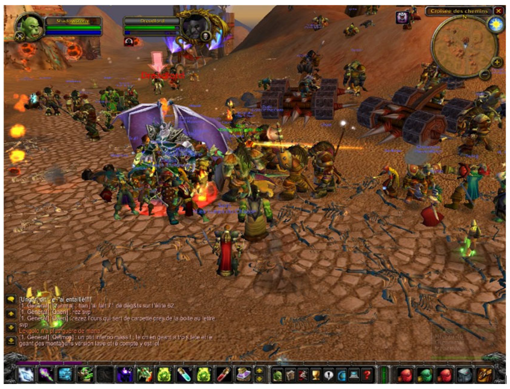
(a) Ultima 1: the first age of darkness origin systems 1981



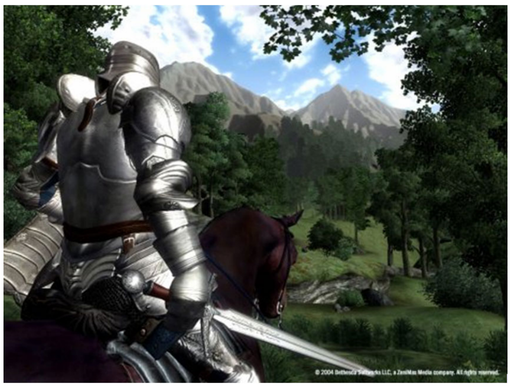
(b) Fallout Interplay 1997



(c) Gothic Piranha Bytes 2001



(d) World of Warcraft, Blizzard 2005. The biggest MMORPG to this date



(e) The elder scrolls IV: Oblivion. 2006 2k Games

Figure 8.3: Screenshots from different RPG games

8.6 Shooters

A shoot-em-up (also known as arcade shooter, twitch shooter, space shooter, or sometimes simply just shooter), is a computer and video game genre where the player has limited control of their character or machine (usually a jet fighter or spaceship) and the focus is almost entirely on annihilation of their enemies ,although some shooters does not rely on this feature at all. The extremely popular ”Paperboy” focus on riding a bike and deliver papers by throwing them at houses, but is to be considered a shooter due to it’s game-play. Shooter is in fact a sub-genre of the action genre, but we will refer to all modern action games as shooters to make it short. Almost all shoot ’em ups display the players score on a score counter, a feature not commonly found in recent videogame genres. While the genre can have 3D graphics, the game-play is almost exclusively in a linear, 2D style. The genre arguably started in the arcades with Space Invaders, and has experienced numerous different games in many formats. There are now several sub-genres that have their own particular game-play characteristics. Sometimes non-shoot ’em ups are described as "shooters", particularly because of the extensive amount of gunplay involved in the game. Light gun shooters are commonly referred to as shooters, because it is the primary action involved. Similarly, first-person shooters are also referred to similarly for the same reasons. While some shoot’em ups can be referred to as rail shooters, this term is an over-arching concept that can apply to interactive movies, light gun games, and action games.

8.6.1 Common features

Every game that has the main focus on shooting is considered a shooter. These are the elements that are most common for all the shooters.

- User controls either a spaceship or vehicle of some sort with a mounted projectile or laser weapon, or a character with a shooting weapon.
- Shoot at anything that moves
- Linear level design
- Weak or no focus on story
- Light easy game play
- Few controls
- Very unrealistic
- Intense Action

8.6.2 Types

The shooter genre gained enormous popularity when it came. Therefore the genre evolved to provide new and more challenging game-play. These are some of the shooters sub-genres.

Fixed shooter

Fixed shooters represent the bulk of the earliest shoot 'em up games. They have the most simplistic premises and the most simplistic controls, especially in terms of aiming. They are characterized by a static environment and a static number of enemies per level, although this stipulation does not preclude that each level can have a different number or enemies or a different setting.

Single screen shooters A single screen shooter, also known as generic fixed shooter or gallery shooter typically only allows players their one or two-dimensional position on the screen. Single screen shooters are basically the oldest popular type of shooters, and represents the bulk of shoot 'em ups from 1977 through 1983.

Tube shooter Tube shooters comprises games where players move forward through a "tube", essentially a 2D scrolling shooter plane rolled into a cylinder or extended to a three dimensional volume. Movement is usually restricted to the ring formed by the edge of the curved plane.

Multi-directional shooter Multi-directional shooters, also called arena shooters, allow freedom of movement and orientation in a two-dimensional environment. Most multi-directional shooters can be further put into two classes based on their control system. Some allow the player to move up, down, left, right, or in some cases, diagonally only.

Thrust-based Thrust-based games which use simplified physics for motion of protagonist's and enemies, most commonly in a zero gravity environment.

Scrolling shooters

Vertical scrolling shooter Vertical scrolling shooters, or vert shooters for short, are largely similar to horizontal scrollers, but the direction of scroll tends to force a different viewpoint on the game: vertical scrollers are nearly always viewed from above. This means that it is less common to have solid obstacles in these games, as the player is usually above them. Perhaps because of this difference, vertical scrollers tend to be more intense, focusing on shooting and dodging copious amounts of projectiles.

Horizontal scrolling shooter Horizontal scrolling shooters are played on an eponymously oriented screen. As well as battling enemies, much of the challenge in horizontal scrollers tends to come from navigating the environment, as invariably contact with the level results in either the immediate death of or damage to the players character. Some games feature a maze-like level that is almost solely focused on avoiding collisions. Enemies are more likely to come from behind the player's ship in these types of games than their vertically scrolling counterparts are. Almost all horizontal scrolling shooters view the players avatar from the side, and present the level in cross-section, such that the player appears to be flying ”through” something, such as a landscape or a mothership

Multi-scrolling shooter Multi-scrolling shooters are a combination of several different types of scrolling shooters. Typically, it involves the combination of vertical and horizontal levels.

Run and gun games

Run and gun games, a hybrid of the shoot 'em up and platform genres, are a genre commonly confused with shoot 'em ups. Run and gun games are popular and distinguishable in their own right and are considered a sub-genre of shoot'em up games. These titles are multi-directional as they mandate that the players fire in many different directions. Titles in this genre are characterized by the same amount of intense action as other shoot 'em ups, but with added abilities for evasion, such as jumping and ducking. As the name would suggest, the action revolves around evading enemies (running) and destroying enemies (gunning).
Examples: **Metal Slug** and **R-Type**

Isometric shooter

Also known as a 3/4 view shooter or three-quarter perspective shooter, an isometric shooter uses the vertical shooters playing field that is modified for perspective. In a traditional scrolling shooter situation, the upwards/forwards is diagonal and the player simulates moving by the game-world scrolling around diagonally. Perspective limits the size of the playing field, so generally there is additional focus on avoiding obstacles than shooting enemy ships. Isometric shooters are not limited to scrolling shooters, but can be multi-directional/area shooters as well.
Examples: **Viewpoint** and **Blazer**

3D shooter

3D shooters (also known as Forward scrolling shooters) pertain to the same game-play elements that occur in a standard shooter (hordes of enemies, obstacles, bosses, movable protagonist ship) but occur in a three-dimensional environment.
Examples: **StarFox (StarWing)** and **Megaman 3d**

First person shooter (FPS)

FPS are dominates the computer game marked. FPS in it self is a big genre that will be discussed in detail later.

Third person shooter

Another perspective of the FPS, will also be discussed in detail later.

Rail Shooter

A rail shooter or on-rails shooter is a specific form of game play in an action-based video games. In a rail shooter the player control is limited to directing where to fire a virtual gun; the player does not have direct control over the path their avatar takes from the start to the end, though they may be able to affect the path followed depending on what they shoot. The player's viewpoint moves as dictated by the game. It is commonly viewed as the player being tied to a rail.
Examples: **Area 51** and **Time crisis**

8.6.3 Strategy shooter

We refer to the shooters who give a time window to calculate shoots, and make further strategic moves in advance before shooting as strategy shooters. The players will usually be issued one or more units in form of a tank or characters who has one or several projectile weapons. The game is usually divided into turns, providing the players a limited window of time to get in position and calculate their shot, often choosing weapon from a heavy arsenal of everything from peaguns to nukes. The projectile will move in a bow across the screen, trying to hit opponents. The projectile will destroy the landscape on impact and decrease the opponents health-bar if it hits.
Examples: **Worms**, **Liero** and **Tank**

8.6.4 Multiplayer Solutions

This genre has almost since it where born supported both competitive and co-adjutant game play in various forms. Run and guns has supported 2 - 4 players at the same time running through the same level cooperating. Railshooters may support 2 players. FPS supports close to 100 players online, Isometric shooters may support 4 players on the same screen. Strategy shooters are intended to be played as multiplayer and yelds the best multiplayer support for single screen of all the shooters.

8.6.5 Common Platforms

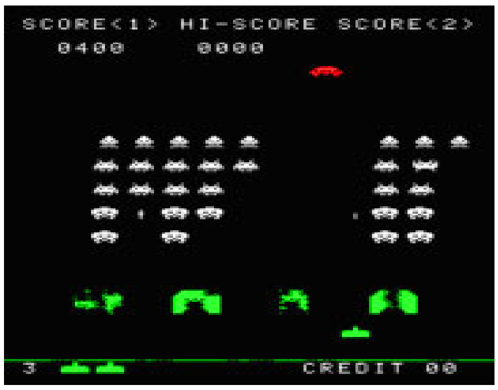
Shooters where introduced early on arcades, and has ever since evolved to fit into every console ever built.

8.6.6 Solutions for our concept

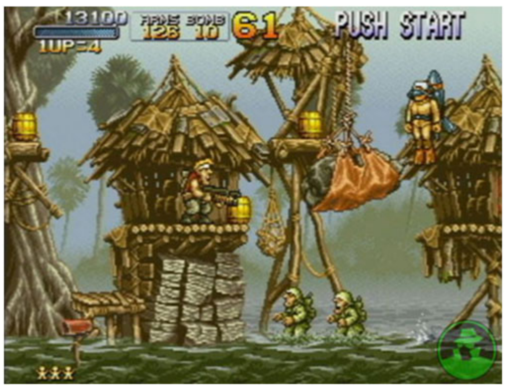
The old Arcade shooters provided an instant action single screen gameplay, hence we could probably use some of these ideas. Although support for approximately fifty players could unleash chaos.

The Rail shooters could make a funny game-play. For instance the players could use their cell-phone cameras to shoot. This option would support competition, but other shooter alternatives would probably be based on competition. A battlefield viewed from above consisting of fighting spaceships is just one example that could be fitted to our concept. We will probably have a closer look on the strategy shooter cause of its support for many players on one screen.

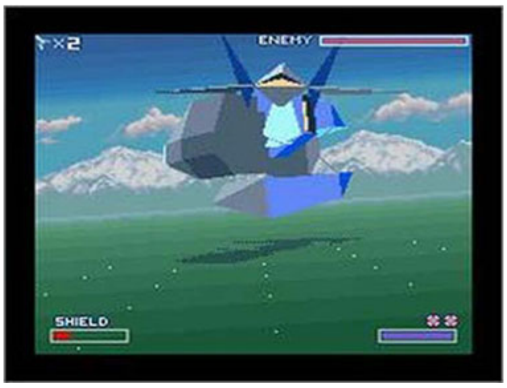
8.6.7 Examples and screenshots from various shooters



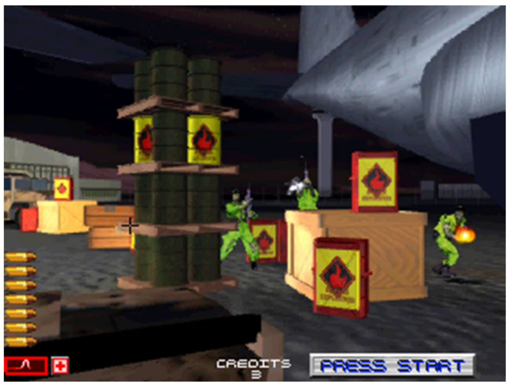
(a) Space Invaders ,Taito 1978 is assumed to be the first shooter. The game made the Japanese yen in short supply



(b) Metal slug, Nazca Corporation 1996, gun and run



(c) Starfox (Starwing), Argonaut Software 1993. One of the first 3d shooters



(d) Area 51 , Atari 1995. Rail shooter

Figure 8.4: Screenshots from different shooter games

8.7 First person shooters

First-person shooter (FPS) is a widely used sub-genre of shooters, which is characterized by an on-screen view that simulates the in-game characters point of view. The gameplay is close to always centered around the act of aiming and shooting handheld weapons, usually with limited ammunition. The modern FPS genre emerged during the early 1990s, at the point when home computers became sufficiently powerful to draw basic 3D graphics in real time. Though almost all other two dimensional shooter games, especially shoot 'em ups, are more concerned with the gameplay mechanic of dodging than of precise aiming. The term FPS has come to refer to games where the player has full control over a character and can interact directly with the environment. The FPS has gained enormous popularity and holds the largest amount of the videogame market, but it has also been criticized for destroying society by creating murderers.

Just like shoot ém ups the first FPSs tended to get a bit monotonous, so the developers saw the need to add more variations into the games. Modern FPSs are typically hybrids from the FPS and the adventure genres with emphasis on story. Also the developers have improved the physics and enemy intelligence to make the game play more ”realistic”.

8.7.1 Common features

These are the features from the games that defined the genre. Most of them are found in the vast majority of the first person shooter to day.

➢ aimpoint centered on the screen

➢ a hand held weapon on the bottom of the screen

➢ weapons get bigger and enemies stronger during progression

➢ Game play consists of collecting ammunition, kill enemies and push buttons

➢ Easy to get into

➢ Usually not too advanced controllers

8.7.2 Types

Sub-genres The realism in FPS games can vary from "arcade shooters", which are fast paced and have unrealistic elements (such as the player being able to shrug off bullets or falling large distances) to levels approaching reality, where players are routinely killed by a single shot. In practice, most games fall somewhere between the two. Distinct FPS sub-genres exist, which use a similar viewpoint and mechanics, but emphasizes different aspects of FPS gameplay. Most first person shooters are hybrids from one or more sub-genres. Below follows a description and examples of the most common sub-genres.

Run and gun first-person shooters

Are fast-paced and action-focused. They often contain a large number of enemies, and allow the player to sustain unrealistic amounts of damage without dying.
Examples: **Doom** and **Serious Sam**

The stealth-based game or "first-person sneaker"

Centers on avoiding detection by opponents. This usually involves sneaking outside the opponents views and stick to dark areas or behind cover. Whenever you get detected, all hell breaks loose and you have to defend yourself. Some FPS version of the stealth based genre are made, but this genre has proved to be much more feasible to the TPS genre cause of its emphasis on environment.
Examples: **Thief** and **No one lives forever**

The tactical shooter

Emphasizes tactics. The player may have to make a strategy to approach a certain situation, taking the environment and situation into consideration. The player either makes a plan of approach before the action, or plan as he goes.
Examples: **Rainbow six**, **Americas Army** and **Operation flashpoint**

Survival Horror

Survival horror is a prominent video game genre in which the player has to survive an onslaught of opponents, often undead or otherwise supernatural. Horror movie elements are used liberally. The player is typically armed, but not nearly as well-armed as the player in other shooter games. The players goal is generally to escape from an isolated house or town that is inhabited mostly by zombies and/or monsters through fighting and puzzle solving. Isolation is generally one of the most recurrent themes within this genre. Survival horror is possibly the

only video game genre that is defined primarily by theme rather than gameplay style.
Examples: **F.E.A.R** and **Doom 3**

The action/adventure shooter

Has larger environments and a greater emphasis on story, puzzle-solving and exploration. Often allows freedom of movement in one world and in which sequence objectives are to be done.
Examples: **Half-life** and **Far Cry**

8.7.3 Multiplayer Solutions

FPS might be one of the best multiplayer alternatives to this date. Many FPS games are designed primarily as multiplayer games. Common solutions are one big battle arena where everyone fights each other, team based, where two or more teams fight each other, and cooperative missions where friends can help each other solve a mission or complete the game.

8.7.4 Common Platforms

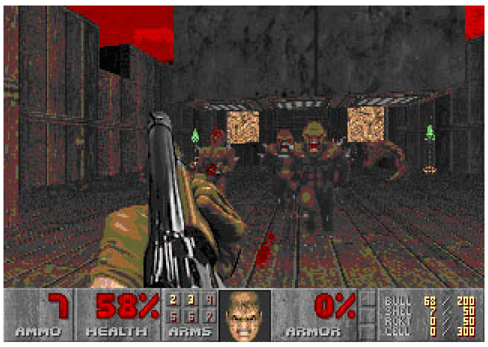
The FPS genre has primarily been reserved for the computer, mostly because the keyboard + mouse combination makes it easier to play. FPS has traditionally been a bit harder to play on consoles cause of the limited controls. It has also been a problem, especially for this genre, that consoles has to display all the players on one screen, dividing it up and making two or four smaller screens. Modern consoles have the ability to get online and provide a better interface for multiplayer, but the lack of sufficient controls are still a fact.

8.7.5 Solutions for our project

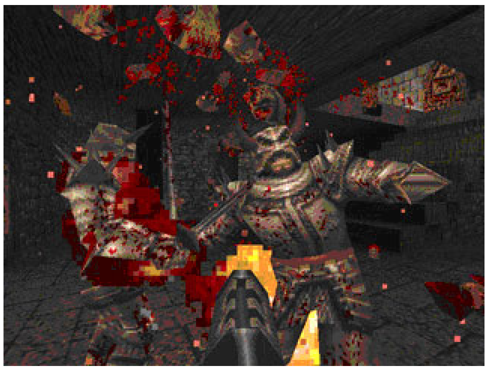
FPS could be implemented as a soulution on the big-screen concept, although we would propably have to divide the screens, which again would support only four or maximum eight players. Gameplay would also be lost due to the fact that every player could see their opponents location.

Another possibility would be to implement the FPS gameplay on the phones and provide a top view of the map and players location on the big-screen. The screen could also show statistics and replays for taunting.

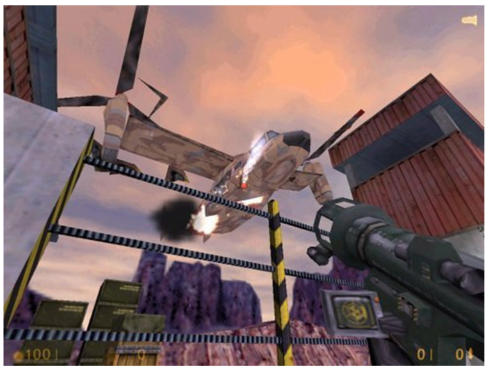
8.7.6 Examples and Screenshots from various FPS games



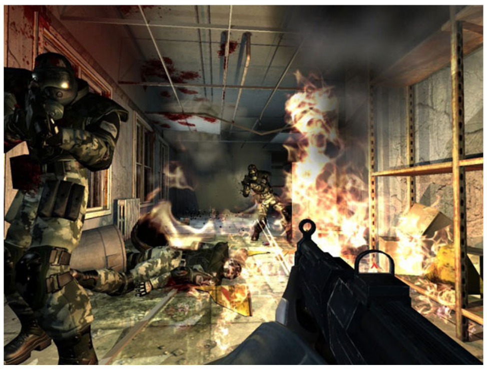
(a) Doom, Id software 1993, is considered to have triggered the FPS genre as we know it today



(b) Quake, Id software 1996, one of the first real 3d game



(c) Half Life, Valve software 1998, Adventure shooter



(d) F.E.A.R, Monolith Productions 2005, Survival Horror

Figure 8.5: Screenshots from different FPS games

8.8 Third-person shooter

Third-person shooter, or third person action, is a genre of 3D computer and video games where the camera view is outside of, and thought of as usually being behind, the main player character. The name became commonplace following the popularisation of the first-person shooter in order to allow the two types of ”shooter” to be differentiated from each other. Both First person shooters and third person shooters are to be considered sub-genres of the shooters genre. There are advantages and disadvantages of TPS games. Third Person perspectives gives the player more awareness of their surroundings, which allows them to focus more on this utility for example as in the Splinter Cell series. A disadvantage is the camera. Horrible camera angles plague many games, although solutions has been developed over the years to make the games more playable without giving the payer a headache. Third person shooters often emphasize story and exploration rather than traditional shooter elements and many TPS are therefore considered to be placed somewhere between these two genres. Games such as GTA III San Andreas involves elements such as driving, fighting, shooting and RPG elements, and may therefore not be considered through the narrow definition as a shooter. However, we will not take all these hybrids into consideration considered it would take a year or two.

8.8.1 Common features

The third person shooters inherits many of the game-play elements from the shooter and first person shooter genres. However, the perspective gives some new challenges and abilities to the game-play. You may expect the following.

- ➢ Emphasize on use of surroundings
- ➢ Emphasize on story
- ➢ Often very RPG/Adventure like
- ➢ Dodging bullets rather than precise aim
- ➢ More advanced controls, Confusing camera angles
- ➢ Many elements from FPS
- ➢ One control to move character, and one to control camera

8.8.2 Types

The types of TPS is pretty similar to the types of FPS,I.E. Stealth based, tactic based, Survival Horror, The action-adventure shooter and even gun and runs, but the game-play is slightly different. TPS close to always emphasize the story and adventure elements keeping a higher atmosphere. Some of these sub-genres has proved themselves to work even better in the third person perspective cause of the emphasis on environments. For instance the stealth shooter genre has been almost dominating as TPS due to the easy exploration of environment and situation. We will give some examples on games that represent the sub-genres.

- ➢ Stealth based TPS:
  Splinter cell and **Metal Gear Solid**
- ➢ Tactic TPS:
  Hitman codename 47 and **Reservoir dogs**
- ➢ Survival horror TPS:
  Resident Evil 4 and **Alone in the dark: The new nightmare**
- ➢ Action-adventure TPS:
  Prince of Persia: the sands of time although it's not really a shooter the game-play is pretty similar to make it fall into this genre, and **True Crime: Streets of LA**
- ➢ Gun and runs:
  Max Payne and **Matador**

8.8.3 Multiplayer solutions

TPS has not been preferred as a multiplayer solution to either pc or consoles. On pc the dominating FPS interface are much more common, and on consoles the multiplayer feature would result in a split screen which take the point of surroundings away. However, most games that are based on TPS are released with multiplayer solutions, and some of them, such as splinter cell chaos theory, becomes a hit.

8.8.4 Common platforms

The TPS works good on pc cause of the mouse+keyboard combination, however this genre is preferred rather than FPS on consoles due to their emphasis on moving the character rather than precise aim where the limited controls are insufficient.

8.8.5 Solutions for our consept

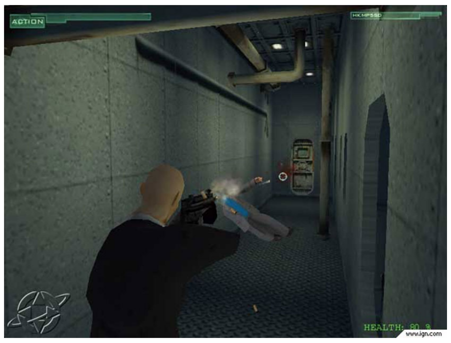
The TPS genre is merely a perspective to change the game-play. It might be desirable as a view for certain sequences, but it would be close to impossible to support twenty to fifty players with a third person perspective on one screen, not to mention the challenges it would give to implement camera control.

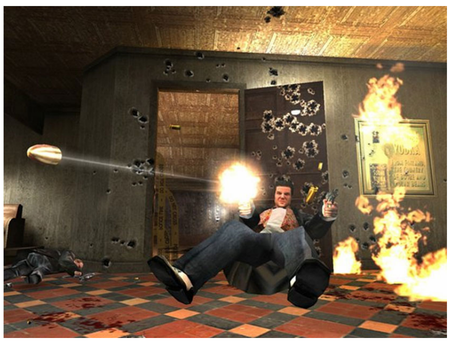
Although we could maybe take into account some of the game-play elements that TPS provides, and mainly the support for environmental overviews.

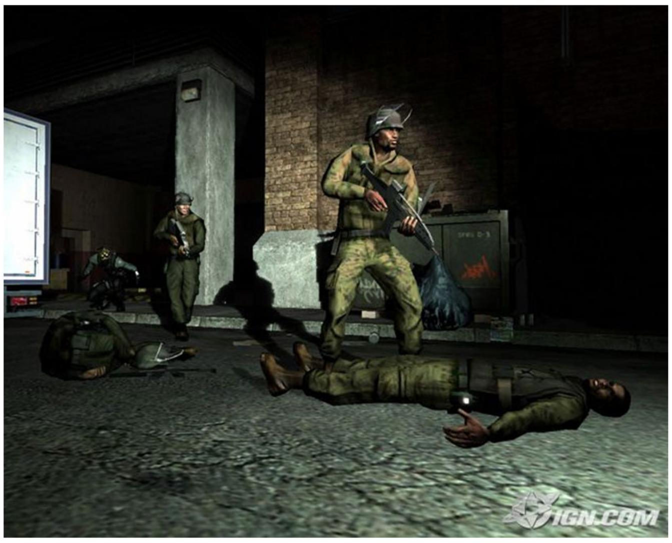
It could probably be a neat feature to have a fixed camera giving a overview from other views during intermissions.

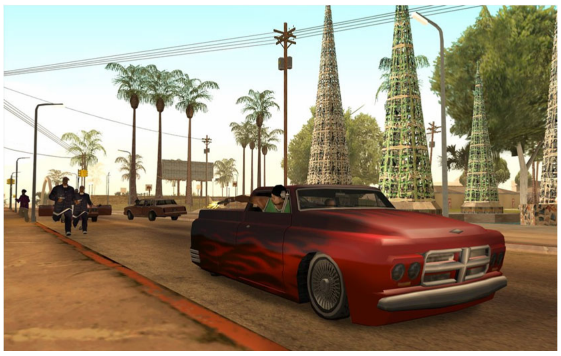
We could make a battlefield using isometric projection and a fixed camera angle. The players could use the environments to get cover, or mount stationary turrets and so on. Dividing the players into two or more teams, giving each team a base, and maybe provide capture the flag multiplay. Depending on the resolution this game would be able to support at least fifteen players.

8.8.6 Examples and screenshots from various TPS


(a) Hitman codename 47, IO interactive 2000, tactical shooter


(b) Max Payne , Remedy 2001, gun and run


(c) Splinter cell Chaos theory, Ubisoft montereal 2005,stealth shooter


(d) GTA San andreas, Rockstar north 2004, action adventure

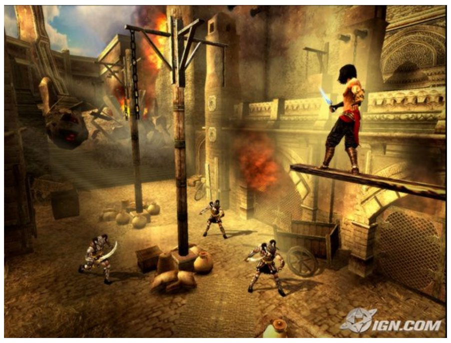

(e) Prince of Persia: the two thrones, Ubisoft montereal 2005, action adventure

Figure 8.6: Screenshots from different TPS games

8.9 Sports Game

A sports game is a computer or video game that simulates the playing of traditional sports or activities related to sports such as management. Almost every familiar sport has been recreated as a game, including baseball, soccer, American football, boxing, cricket, golf, basketball, ice hockey, tennis, bowling, rugby, hunting, fishing, etc. This genre has been popular throughout the history of video games and is extremely competitive, just like real-world sports. A number of games series feature the names and characteristics of real teams and players, and are updated annually to reflect real-world changes.

8.9.1 Common Features

The features of Sport games is of course dependent on what type of sport game it is. However there are some elements that are common for most of the sport games.

- ➢ Total control of one character (at least for modern sport games, possibility to change between characters when controlling a team)
- ➢ Easy controls
- ➢ Simple built
- ➢ Game play often consist of rapidly pushing one or more buttons
- ➢ Easy learning
- ➢ Comparing of high scores
- ➢ Enchantment in form of tournaments

8.9.2 Types

The sport game genre represent a wide span of games, depending on which sport they represent. Some sport are of course more popular than others, and some sports give better support for game-play. Games based on extreme sports has always been among the most popular. It would be hopeless to divide the sub-genres into different sport genres, cause the list would never end. Instead we will look on the different techniques to provide game-play.

Play the character

Game based on one or more sport genres where the player gains total control of the character and competes against cpu controlled players or other players.
Examples: **SSX** snowboard game and **Winter Olympics: Lillehammer 94**

Management

The player plays as a manager for a team or athlete. This genre may be claimed to be a sub-genre of the simulation rather than sport, but it borders to the sports genre due to it's theme. The game is often based on teams or athletes from the real world.
Examples: **Championship Manager** football managing game and **Boxing manger**

Racing

Will be discussed in detail later due to it's independence and popularity.

8.9.3 Multiplayer solutions

The most common solutions for multiplayer on sport games are competitive where players compete each other in sports. However, team based sport games are often based on team play as well. On consoles the players are often divided into turns, because of the narrow possibility to represent two or more users at the same time without splitting the screen.

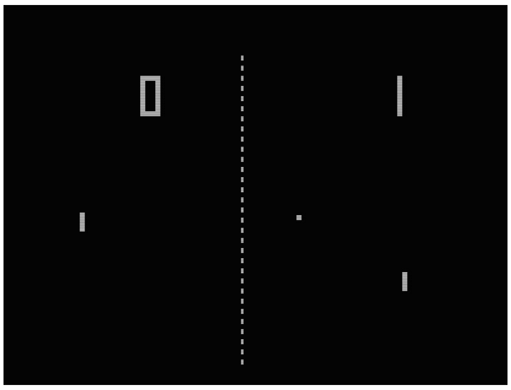
8.9.4 Common Platforms

Sport games has existed as long as videogames has, and has been represented on all consoles and computers.

8.9.5 Solutions for our concept

Sport games provides endless possibilities for implementations to our concept due to both their competitive game-play and popularity. The only limitations are visualization, hence we'll probably have to divide players into turns of some sort. For instance a football field viewed from a top perspective angle, and give the players control of one character might be one way to implement competitive cooperation. Of course we could make a turn based athletics game, with five players in each turn, but this would result in much waiting time. Two competitive teams may support at least 22 players, and it is therefore preferable in favour of the turn based alternative.

8.9.6 Examples and screenshots from various sport games



(a) Pong, Atari 1972, one of the first arcade video games

(b) Track and field, konami 1982

(c) Tony Hawk Pro Scater, Neversoft 1999

(d) Fifa 07, EA Canada 2006

Figure 8.7: Screenshots from different sport games

8.10 Racing games

A racing game is a video-game genre that involves competing in races through a vehicle or object of some sort, either getting it from one point to another or completing a number of circuits in the shortest time.

Racing games are usually in the first or third person perspective, although some racers are viewed from the top. They may be based on anything from real-world racing leagues to entirely fantastical settings, and feature any type of land, air, or sea vehicles. In general, they can be distributed along a spectrum anywhere between hardcore simulations, and simpler arcade racing games.

The player controls a vehicle of some sort (spaceship, car, bike) most commonly viewed from behind, but titles viewing the vehicle from profile or the top also exist. The first racers displayed a car in the bottom center and animated the road scrolling towards the player giving the feeling of driving on the road. Most modern racing-games do exactly the same, but combine this with multiple angles of the car to give a more realistic atmosphere. The player often has to choose attributes balancing the car between steering, acceleration, top speed, gripping power etc.

8.10.1 Common features

The Racing game genre has evolved from its origins, but all racers still hold on to the spine of the gameplay, competing other vehicles aiming for the lead position. Most of the traditional racers include the following elements.

- The player controls a vehicle of some sort
- The controls are limited, and easy to learn
- Game-play is simple, and based on competition
- Collecting power-ups upgrades your vehicle

8.10.2 Types

Racing simulators

Simulation style racing games strive to replicate the handling of a car so it feels like the real world. They often license real cars or racing leagues, but will use fantasy cars built to resemble real ones if unable to acquire them. Although these racing simulators are specifically built for people with a high grade of driving skill it is not uncommon to find aids that can be enabled from the game menu. The most common aids are traction control (TC), anti-lock brakes, steering assistance, damage resistance, clutch assistance, automatic gearboxes, etc. This softens the learning curve for the difficult handling characteristics of most racing cars.

Modern racing simulators tend to focus on story and RPG elements. You will start with a cheap car and limited possibilities for racing, and build up your character and skills through racing, often in various genres of motorsport
Examples: **Gran turrisimo** and **Need for speed: Most wanted**

Arcade racers

Arcade style racing games put fun and a fast-paced experience above all else, as cars usually compete through odd ways. They often license real cars and leagues, but are equally open to more exotic settings and vehicles. Races take place on highways, windy roads or in cities; they can be multiple-lap circuits or point-to-point, with one or multiple paths (sometimes with checkpoints), or other types of competition, like demolition derby, jumping or testing driving skills.
Examples: **Daytona USA** and **Out Run**

Futuristic racers

Futuristic or extreme racers are racers which has litle or no focus on reality. The vehicle is usually a spaceship, or some science fiction vehicle of some sort. The environment of the race is usually surface of a planet, a futuristic stadium or in the air flying over skyscrapers and so on. These racers often makes use of powerups that can boost the vehicle temporarily.

Futuristic racers are often refereed to as extreme racers, cause of their ability to break free from reality giving them lack of speed limitations and fictional laps that might consist of huge jumps and so on.
Examples: **F-zero** and **Wipeout**

8.10.3 Multiplayer solutions

Multiplayer support for racing-games is almost always competitive. Multiplayer on pc or online services for consols may support about a dozen players competing, while single screen consoles support two to four players usually on split screen or viewed from the top.

8.10.4 Common platforms

The racing games is considered a sub-genre of the sport games, and has gained popularity on all platforms. The easy gameplay and simple construction makes it suitable for even hand held consoles such as mobiles and PSP.

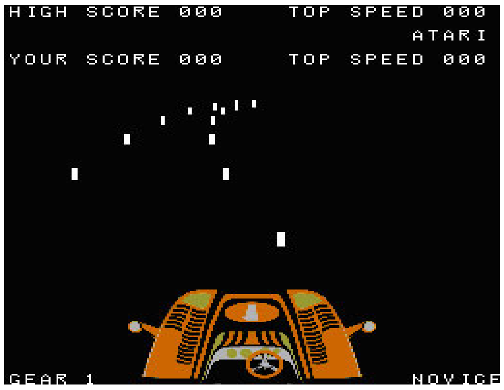
8.10.5 Solutions for our concept

The racing genre may have huge possibilities due to popularity and competitive gameplay. One problem with this genre applied to our concept is the single-screen constraint. One approach adapted from old arcade racers is to display a top-view of the course and divide the cars based on colors and shapes. This way we could propably be able to support at least ten.

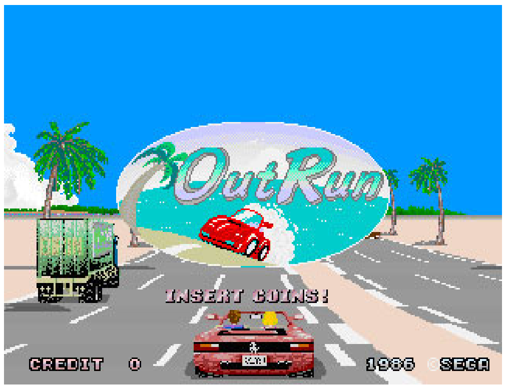
Another approach could be to take the gameplay to the hand held unit and provide status and focus on leading cars etc. on the big-screen. This solution would also support more entertainment for spectators.

The implementation will probably have to be based mainly on competition, but we could make cooperation in form of a relay race.

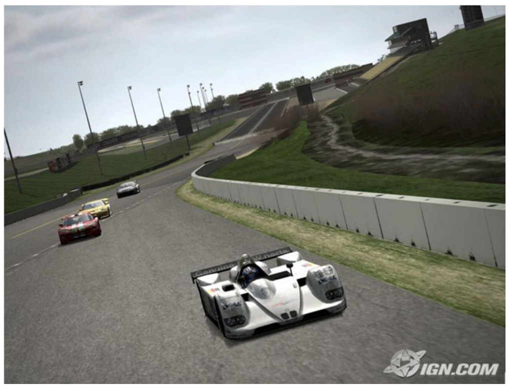
8.10.6 Examples and screenshots from various racers



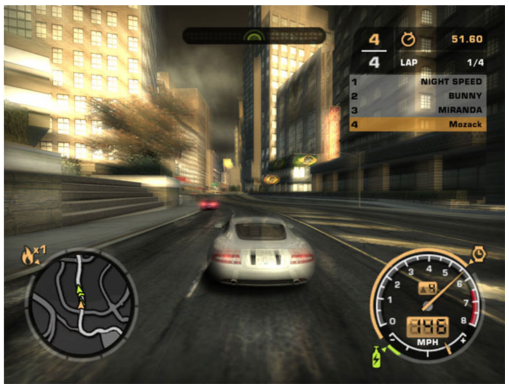
(a) Night Driver, 1976 Atari is probably the first racing game ever



(b) Out run, Sega-AM2 1986



(c) Gran Turisimo IV, Polyphony Digital 2004, probably the most advanced Racing-game so far



(d) Need for speed: Most Wanted, EA black box 2005, Offers a RPG-like adventure based storyline

Figure 8.8: Screenshots from different racing games

8.11 Fighting game

”Fighting game” is a term confusingly and interchangeably used, often depending on locality, to describe two separate genres of video games: ”Versus fighting games” (or ”fighters”) and ”Beat’em ups”, in which players fight each other or computer-controlled enemies, usually employing some variation of the martial arts. Along with fixed shoot ’em ups, they are traditionally at home in the arcades, and are considered separate from sports games such as wrestling, boxing and "ultimate fighting" games.

8.11.1 Common features

These elements are common for about all fighters. Of course, some fighters differ from these common features, and make their own solutions.

- ➢ main character can be chosen from a set of characters with different fighting abilities such as style or strengths.
- ➢ The game is viewed in a profiled angle
- ➢ Rather easy controls
- ➢ The player fights by pushing buttons in a certain order, known as a combo.
- ➢ No or weak focus on story

8.11.2 Types

We have divided fighters into the two most common sub-genres. There are other sub-genres from the fighter genre, but considering that they are either too different from their origins or just differs minimal, we chose not to cover them.

Beat ’em up

In this type of fighting game, typically known as a scrolling or side-scrolling fighting game, brawler, beat ’em up, or more rarely, walk-and-punch game, one or more players (most often two, but sometimes as many as six) each choose a unique character, and team up to punch, kick, throw and slash their way through a horde of computer-controlled enemies. The fighting occurs in a series of side-scrolling stages, some with a powerful boss enemy at the end. In the most common variation, players can move away and toward the screen as well as left and right, although earlier scrolling fighting games such as Kung Fu Master were more likely to allow only single-dimensional or linear (horizontal) movement, plus jumping.
Examples: **Turtles** and **Double Dragon**

Versus fighting game

In versus, or competitive type of fighting games, two players (sometimes more) each choose a character, and then fight against each other over several rounds. The winner of a round either knocks out his opponent (usually by depleting an energy indication bar to zero), comes closest to knocking him out, or (in some 3D titles) sends him out of the ring.
Examples: **Tekken series** and **Street Fighter II**

8.11.3 Multiplayer solutions

As mentioned there are mainly two sub-genres that define the genres. The versus type are intended to make two or more players to compete by beating each other, but many titles also support competitive multiplayer dividing the players into teams making them play in turns. The beat 'em up type usually supports cooperation, where two or more players fight together against a horde of enemies.

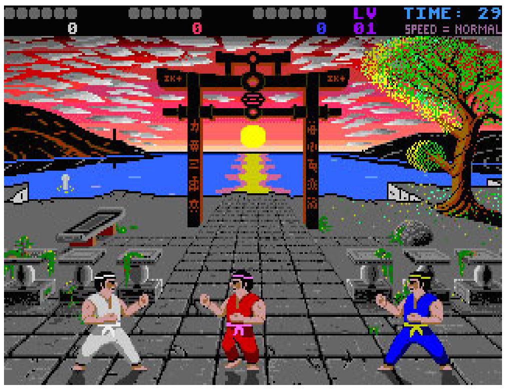
8.11.4 Common platforms

The fighters has found it's home both to the consoles and the computers, although mainly on consoles. Both the versus and beat 'em up sub-genres represent a perfect fit due to the easy implementation of multiplayer possibilities on consoles.

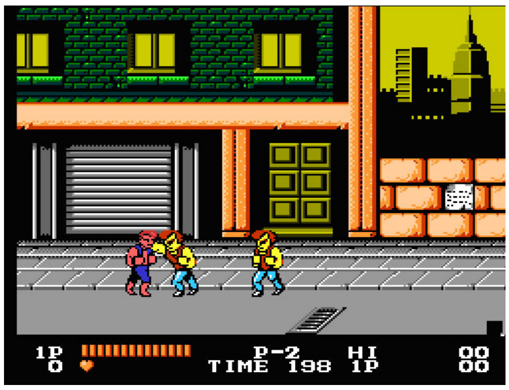
8.11.5 Solution for our project

A versus fighter could be a perfect hit on a big screen, but it lacks support for several players. Versus fighter could support 2 or maximal six players at the same time, and would probably have to be divided into turns. The same goes for beat 'em ups, the lack of support for many players is a barrier. However, we might be able to make a hybrid supporting up to ten players, but thirty to fifty won't work. The beat ém ups builds on cooperation, which could be a strength for several people sitting together in a cinema, and it could be enchanting both for players and spectators. The game could be developed in some sort of turns, where you where pushed into the game when someone other died, constantly holding the number of human controlled players in the screen about six or seven.

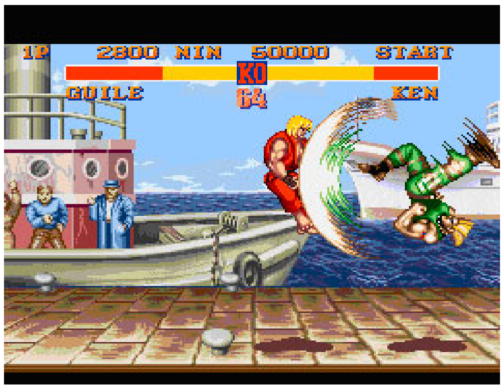
8.11.6 Examples and screenshots from various Fighter games



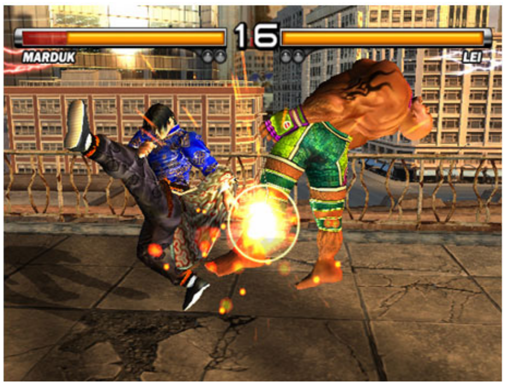
(a) International Karate +, System 3 1987, Early Versus fighter



(b) Double Dragon,Technos 1987, Beat ‘em up pioneer



(c) Street Fighter II, Capcom 1991, maybe the most famous fighter



(d) Tekken 5, Namco 2004, Supports both Beat ‘em up and versus mode

Figure 8.9: Screenshots from different fighter games

8.12 Simulation Games

A simulation game, or sim game, (also known as a game of status or mixed game) is a game that contains a mixture of skill, chance, and strategy to simulate an aspect of reality, such as flying a plane or soldier procedures in combat situations. Simulation game spans over a wide variety of different sub-genres, and does not particularly define game-play. Knowing that a game is a simulator does not necessarily provide enough info for a person to determine what to expect in addition to game-play and visualization techniques. Many of the simulators sub-genres is also associated with other genres, for instance many of the war simulators will be considered to be strategy games rather than simulators by many and strategy is therefore a common association to simulators. Most simulation games are intended to simulate the real world, while others simulate a fictional world or both. Simulation games are widely used as a tool to teach people how to drive or fly vehicles or train people on how to manage certain situations, but they have also struck root in the entertainment market.

8.12.1 Common Features

- ➢ No story
- ➢ Emphasize realism
- ➢ Often very advanced controls (every button on the keyboard has a function)
- ➢ May take some time to get into (often required to read a book)
- ➢ Game-play is rather constant

8.12.2 Types

Sim games or Simulations are a genre that might be implemented in several very variating forms. Many approaches focuses on serious gaming providing ways to teach or train people in certain situations or circumstances. However, there are many implementations of sim games made for entertaining purposes. The worlds best selling pc game, The Sims, is just one example. We present the most common ways to implement simulation games bellow.

Vehicular simulators

Vehicular simulators generally attempt a realistic representation of how to drive a certain vehicle. Flight simulators and Racing games are typical examples.
Example: **MS Flight simulator series**

Tactical war games

Tactical and operational war games simulate small-scale battles, typically involving a few hundred or at most a few thousand soldiers.
Example: **Full Spectrum warrior**

Strategic war games

Strategic war games simulate large-scale battles, campaigns, and entire wars. Grand strategy war games and nation-simulation games allow the players to control nations. Due to their scale they are commonly of the turn-based or real-time strategy type.
Examle: **Codename: Panzers**

In god games

In good games players represent an entity with supernatural powers. It can be argued that god games are a border category within simulator games because it is unclear how to simulate being God.
Example: **Black and white**

In life simulator games

In life simulator games, which sometimes overlap god games, a virtual life, career, etc. is simulated.
Example: **The Sims**

Economic simulation games

Economic simulation games present players with aspects or the entirety of an economy or a business.
Example: **Ports of call**

city-building

City-building is a sub-genre of the Economic simulation, simulates building and controlling of cities.
Example: **Sim city**

8.12.3 Multiplayer Solutions

Simulation games where never intended to support multiplayer, although some modern simulator games do support online gaming, and some titles are even based built specific for multiplaying. Examples here are The sims online, which is a social online game based on the elements of the original Sims.

Since simulation games most often are based on serious gaming rather than entertaining gaming the gameplay most commonly rely on cooperation, teaching players how to cooperate, and so on. But strategic war simulations often provide multiplayer gameplay based on competition.

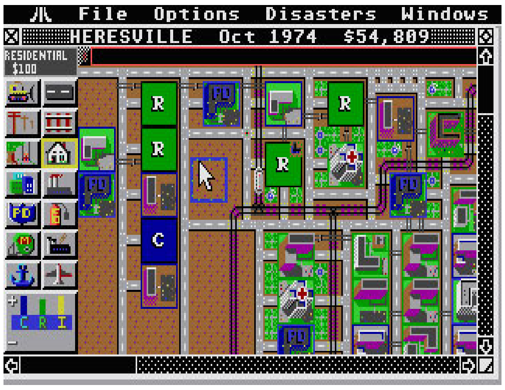
8.12.4 Common platforms

The simulator games have been reserved the heavy machines such as computers. This is due to their complex construction, big size and not to mention advanced controls. However, lighter simulation games have found their way to the consoles, such as The Sims and Black and white.

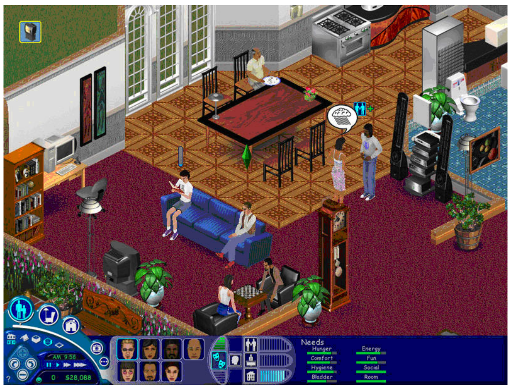
8.12.5 Solutions for our concept

Simulation games is often a great solution to train people and that could be a appropriate feature to our concept. However, the concepts boundaries would probably not allow the controls nor the advanced game-play. Also the lack of multiplayer support causes a problem. Although in putting the framework in use of serious games for learning or training, for example for people to cooperate to solve a given problem, may require some of the simulations abilities.

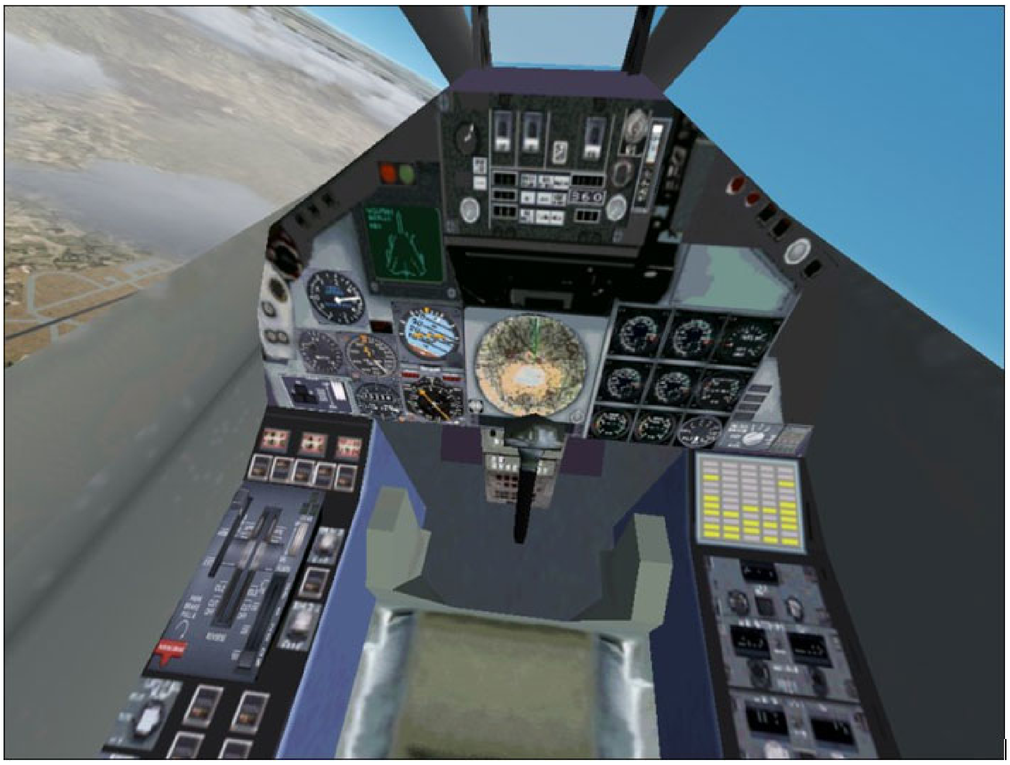
8.12.6 Examples and screenshots



(a) Sim City, Maxis 1989



(b) The Sims, Maxis 2000



(c) MS Flight simulator 2004, Microsoft

Figure 8.10: Screenshots from different simulation games

8.13 Strategy game

Strategy games, as their name indicates, is a video game genre that emphasize strategies to approach a situation, commonly a battle or war. We can divide these games further into two main sub-genres, turn based strategy and real-time strategy.

A turn-based game, also known as turn-based strategy, is a game where the game flow is partitioned into well-defined and visible parts, called turns or rounds. For example, when the game flow unit is time, turns represent units of time, like years, months, weeks, or days. A player of a turn-based game is allowed a period of analysis before committing to a game action, ensuring a separation between the game flow and the thinking process, which in turn leads presumably to better solutions. Once every player has taken his turn, that round of play is over, and any special shared processing is done. This is followed by the next round of play.

Real-time strategy are often characterised by being war-games which take place in real-time, wherein resource gathering, base building, technology development and the player exerting direct control over individual units are key components. While the word ”strategy” originally referred to high-level war planning (armies, campaigns, and entire wars), in real-time strategy games individual units or persons are given orders. Also integral to the game-play of real-time strategy games are production-economic aspects (resource gathering, construction, positioning of buildings, and production of units), and though military confrontation is a significant part of real-time strategy game-play, it is most often heavily stylised with relatively little emphasis placed on realism or the detailed aspects of military tactics as contrasted to games of the genre known as real-time tactics.

8.13.1 Common features

The features for strategy games is highly dependent on which type of strategy it is based on. You may expect the following features.

- ➢ Battlefield or world viewed from a top/perspective angle
- ➢ Management of recourses
- ➢ Control of multiple different units
- ➢ Minimap that shows where you are and what to expect from other areas
- ➢ Fog or shadow covering the areas you can’t see
- ➢ Use strategies to decide when and how to strike at your enemy
- ➢ Balancing between units abilities and weaknesses

8.13.2 Types

Strategy is element in many video-game genres, and many of the games considered to be strategy games may lean close to the simulation genre. The general strategy games may be divided into the following sub-genres.

Turn based

Turn based strategies have two aims, the one where all players (player and CPU) have a fixed amount of time to deploy their strategy, and a fixed window of time runs in realtime. This approach is also called ”WeGo”. The other aim is to let the game work in turns, with one player making the moves at a time, called ”IGOUGO”.

Sometimes turn-based Additionally, many other games that are not generally turn-based retain the notion of turns during specific sequences. Notably, the role-playing game Fallout is turn-based during the combat phase, and real-time throughout the remainder of the game.

Acting outside your turn Some games (notably X-COM) allow you to act outside your turn by designating different automatic re-actions that are executed when certain conditions apply in the opponents turn, adding an element of strategic guess-work.

Turn-based gaming Turn-based gaming refers to browser-based game sites that allow for game-play to extend beyond a single session, often taking months for complex games like Go or Chess to finish.

Turn-based ”tactics” This sub-genre is principally used only in RPG-derived games as an alternative to the traditional turn-based system. The system has been tailored to incorporate RPG characteristics. The genre has its origins in tabletop role-playing games, where each player has time to decide his or her character’s action. Turn-based tactics games generally feature no more than a dozen characters on either side of the battle (usually less). The term ”tactics” had not showed up until Final Fantasy Tactics was released, where it popularized the genre in the U.S., although games such as Shining Force have utilized the genre years beforehand.

Play-by-mail games Play-by-email TBS games allow the orders to be passed in a very loose synchronization mechanism: email.

Real-time strategies

The real-time strategies are more fixed into a defining standard, and throughout the history of RTS, no bigger changes from the original concept has been made. However we have three superior aims.

Micro-management games Micro-management games allow an army and base to be built, but they limit the size of the army (sometimes rather severely). The purpose of this is to create more of a tactical atmosphere. By limiting the size of the army, the game requires a player to intelligently utilize his or her limited troops. Good examples of this type of game are Warcraft III, where further units require more "upkeep", and Battle Realms, which allows only a maximum of 40 units. To simplify the control, however, a player may combine individual units into groups. This is even more prominent in the game ArenaWars, where every player only has 1,000 credits to build units.

Macro-management games[On the other end of the scale are the macro-management games. These titles have more of a focus on economic production and large-scale strategic maneuvering, and include games such as Age of Empires II And Total Annihilation where there is no population limit on units and there is no limit on how many units may be controlled at once.

No-management games Focus on the battle and does not concern either building or recourse collecting. Recourses, such as units, are typically fixed or chosen before the game starts. This genre is inherited by the turn-based style. Some games give the player ability to stop time to make strategies.
Examples: **Microsoft's Close Combat** and **Company of heroes**

8.13.3 Multiplayer solutions

The RTS genre is commonly intended for multiplayer abilities. Usually players compete against each other, but many titles also support cooperation by dividing in teams, or control of the same unit. The TBS is often included in other genres that are singleplayer, but it is a common interface for multiplayer games. The strength of TBS is the ability to hide your strategy before you strike. Depending of strategy type the amount of players are usually limited by the map size, typically up to eight players at the same time.

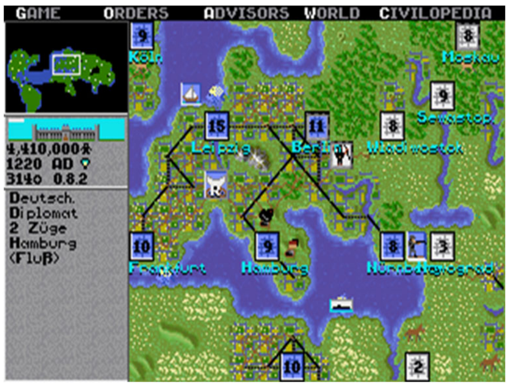
8.13.4 Common platforms

The first RTS was built to computers. However, the popularity and growing strength of consoles has made them sufficient for the genre. Strategy games often requires the user to be able to act fast, which might make the PC mouse preferred in favour of the console controller.

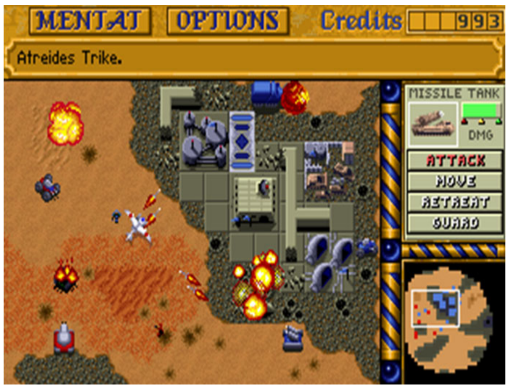
8.13.5 Solutions for our project

The strategy game-play is highly desirable due to the genres popularity. We would have to make some modifications and new solutions to fit the genre onto our concept. For instance we could divide the screen into regions, and have teams starting in each region. It could also be possible to move more of the game-play onto the control, making strategies during intermissions and see them visualized in a real-time sequence. It would also be possible to have a superior world view where players owned regions, and switch to a battlefield view whenever one player attacked another. Game play probably have to be turn-based, with more than one player pr round, and team based to work well with the concept. We will take this genre further into consideration.

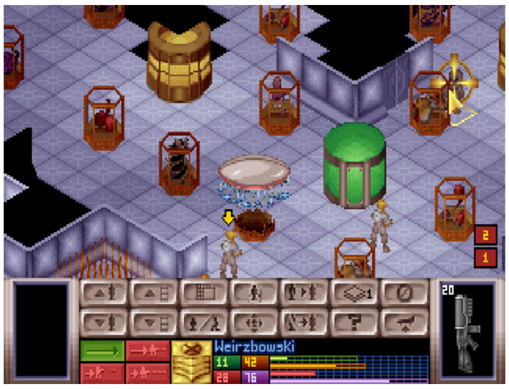
8.13.6 Examples and Screenshots from various strategy games



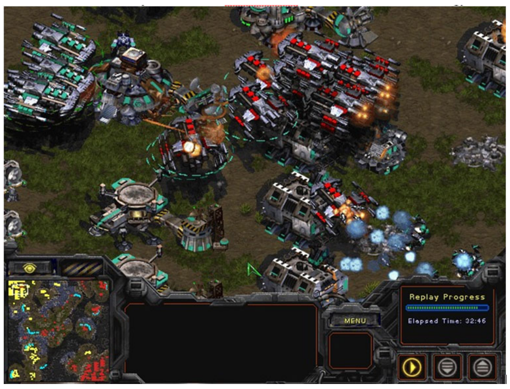
(a) Sid Meier's Civilization, Microprose in 1991,turn-based strategy



(b) Dune II: The Building of a Dynasty, Westwood Studios 1992, Defined the RTS genre



(c) X-COM UFO: enemy unknown, MicroProse 1994, Turn based Strategy



(d) StarCraft,Blizzard Entertainment in 1998, best selling real-time strategy

Figure 8.11: Screenshots from different strategy games

8.14 Discussion

The research on game genres gave us the conclusion given in Table 8.2.

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

* fifteen players using isometric fixed projection, eight using splitscreen

We have covered the most important and the most popular genres. Genres such as action and puzzles have been neglected due to the fact that most of them are somehow baked into the other genres together with the fact that they alone, specially puzzles, yield no opportunities for supporting up to fifty players on a big-screen.

We have reviewed the most important and the most common video-game genres, but few modern games tend to be tied to one genre. In fact almost the only games strictly holds to one genre is the game who defines it. Most modern games are so called hybrids from different genres. Take *Counter Strike* and *Americas Army* for instance. Both are made as tactical shooters, but where Americas Army tend to lean over to the stealth genre, the Counter Strike goes the other way and ends up in a more instant action "gun and run".

We have realized that we would need to look on these hybrids both to define games that will work properly on a big-screen and to support multiplayer with up to 50 players.

Further we will discuss some features and concepts that we would like to include to make the most out of the gameplay experience, however we will not go deeper into the business opportunities around the concept.

8.14.1 Features

Our game has to be quite intense and instant action, more arcade-style like. It also has to provide good gameplay to 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

Gendre	Multiplayer opportunities	Max players	Features to concept	gameplay elements
Adventurer	Poor, cooperation	2 - 8	Storytellling, enchanting	Solve puzzles, communicate, collect things
CRPG	poor, both cooperation and competition	2 - 8	Storytelling, statistics, players can manipulate their characters	collecting points and artifacts, develope character, freedom
Shooter	Good, both	about 50	Instant Action, good projection methods	instant action, shoot everything that moves
FPS	Limited, Both competition and cooperation	8 (split-screen)	Same as shooter	Same as shooter
TPS	limited, both	About 15 or 8*	Use of surroundings, isometric battlefield	Bullet dodging, use of environment
Sporters	Good, Both depending on sport	Enough (sport dependent)	Endless possibilities, every sport can be used	Depending on sport, should be implementable to our consept
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 army, make strategic moves

Table 8.2: Genre sufficiency

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 issued:

- ➢ 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.

8.14.2 Multiplayer support

With this amount of players we have considered to use multiplayer mainly in form of competition, but we also see opportunities to base this competition on cooperative teams. Due to the fact that fifty players alone would take a great size of the screen, there would be no room for AI-payers, therefore the multiplayer solutions have to be competitive in some way.

8.14.3 Concepts

There are several things to take into account to make game elements work on our platform. Most of the games should have to 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 some concepts that can prevent or minimize the latency.

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 provides better support for players that come and go.

8.14.4 Test Game

To get started we will take some ideas from old games that has allowed more players on one screen. In 1995 Team17 software released Worms to the Amiga platform and later to the PC. Worms is a videogame with a main focus on multiplayer that gives each player control over a platoon consisting of four worms. Each worm carries an arsenal consisting of a wide span of different weapons such as rifles, bazookas and nuclear bombs.

The game is turn-based, each turn gives the player a limited amount of time to position one worm, choose weapon, aim and fire at one of the opponents worms.

The game takes place in a 2D world viewed from a profiled angle. Worms will be placed on platforms on the map, shooting the platform or the ground will make a crater. Under the ground there is water or lava, and a worm falling in the water will drown.

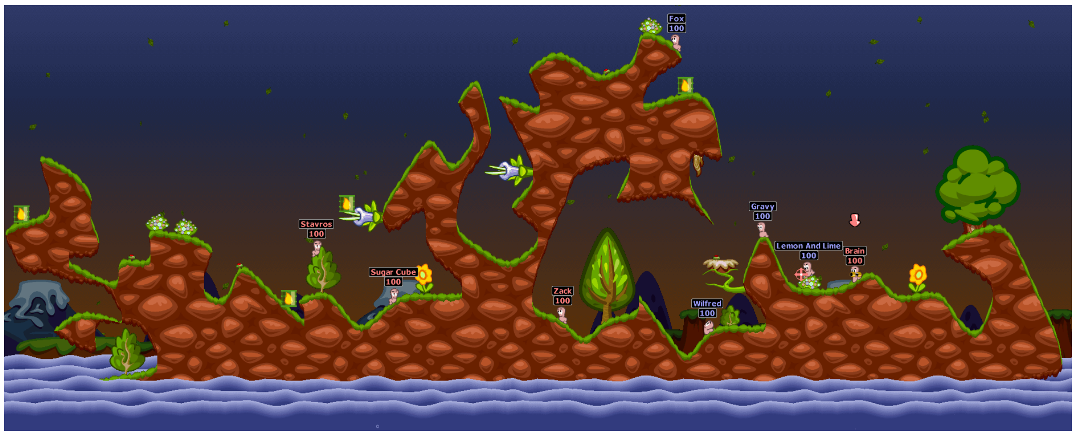


Figure 8.12: Worms Armageddon

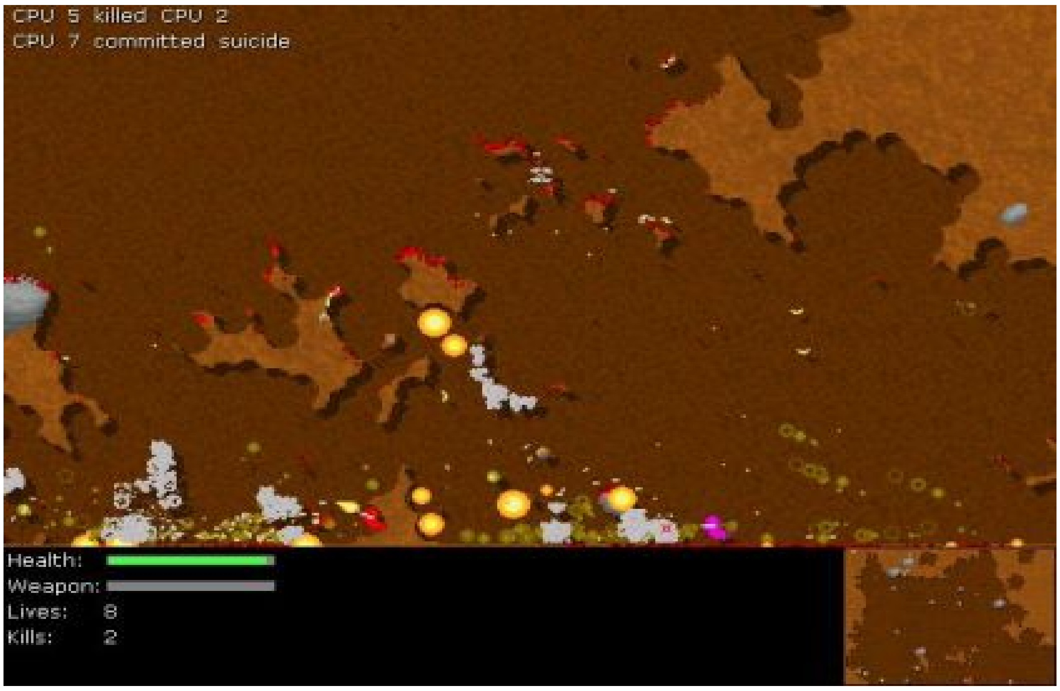
Worms provides a good game-play to support greater number of players on one screen, but it would need a few modifications to fit our concept.

The turnbased structure will not fit our concept due to the great amount of players. Besides that, we will also have to cut down on the squads. A player will only be able to keep track of one character at a time, together with the fact that it would not be enough space to display 4 times 50 worms.

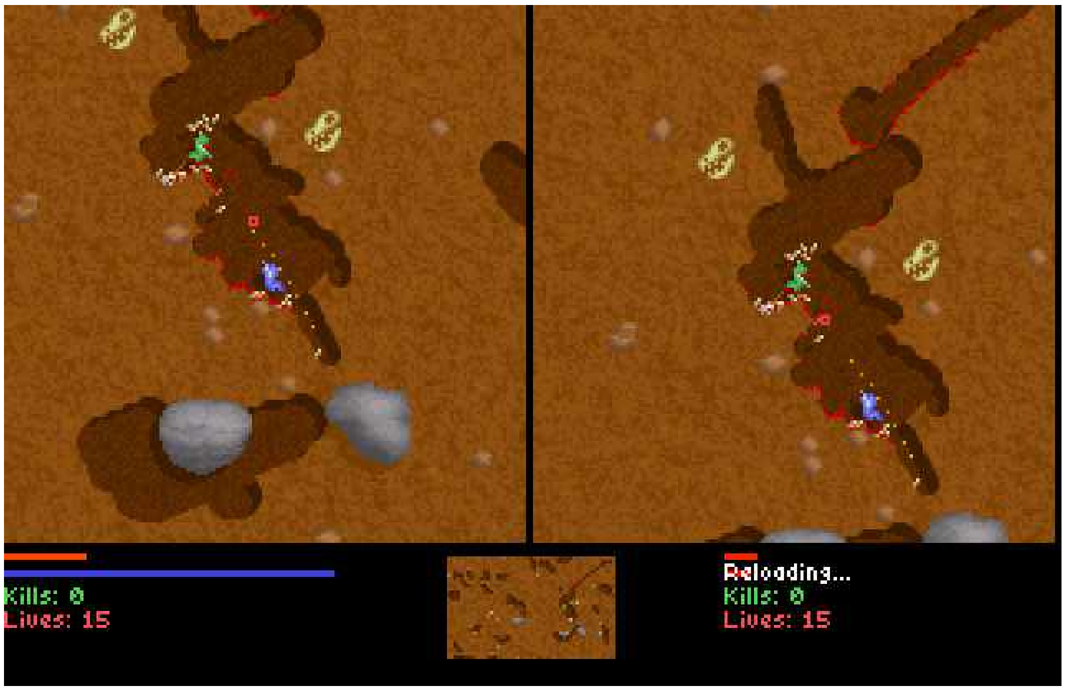
In 1998, Joosa Riekkinen made Liero, which is a realtime version of worms. Liero inherited many of the features from worms such as using a 2D landscape as a battlefield, and a wide span of various weapons. But that also made some great differences on the gameplay.

In Liero the player will gain control of one worm only, and move around on the map until he meets other worms to kill. If a worm gets killed, he will spawn at another point of the map and the killer will be credited for one frag (kill). The worms proceeds by digging tunnels with their weapons or by using rappel gear to reach higher levels.

We plan to make a modified version of Liero to test our framework. Every player will be given



(a) Liero



(b) Liero with splitscreen

Figure 8.13: Different Liero running options

a worm controlled by their mobile device. Liero provides a perfect match to our concept, and therefore only a few modifications need to be made.

CHAPTER 9

Technology

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

9.1 Sony 4k SXRD projector

The largest cinema auditorium on Nova Cinema has installed the first Sony 4k [15] projector ever produced (see figure 9.1). 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 has a card 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 on a computer by using a converter.

9.2 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

Figure 9.1: Sony's 4K projector

provides a nominal data rate of 1.485 Gbit/s which supports a resoultion up to 1080p at 30Hz.

9.3 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 [16], 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 graphics card from Nvidia [17]. This card can render and output directly to 4xHD-SDI interfaces.

9.4 Bluetooth

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.

9.4.1 Specification

The protocol operates in the license-free ISM band at 2.45 GHz. In order to avoid interfering with other protocols which use the 2.45 GHz band, the Bluetooth protocol divides the band into 79 channels (each 1 MHz wide) and changes channels up to 1600 times per second. Implementations with versions 1.1 and 1.2 reach speeds of 723.1 kbit/s. Version 2.0

implementations feature Bluetooth Enhanced Data Rate (EDR), and thus reach theoretical 2.1 Mbit/s. 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).

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 have no performance and lag issues. As we can tolerate more lag than both of these gaming consoles, we shouldn't have any problems using bluetooth.

9.4.2 Limitations

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.

Bluetooth specification allows connecting 2 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 are supposed to appear in 2007.

Further research into the use of bluetooth technology also shows that the Windows OS only supports the use of 1 bluetooth hub (usb stick) per machine which has a maximum of 7 slave connections, while Linux supports multiple.

9.4.3 Our implementation

With a look at the current possibilities and limitations with bluetooth, we have the following hardware setup options:

Fortunately we found a bluetooth hub which can support up to 21 bluetooth connections per access point [18], and has support for converting bluetooth rfcomm connections to standard tcp/ip connections to the game server. With those, we have almost no limit on how many players we can have at one location, just get more access points.

We have also looked into how many players we can support at one cinema simultaneously. If every bluetooth device transmits continuously, each piconet is unsynchronized and placed

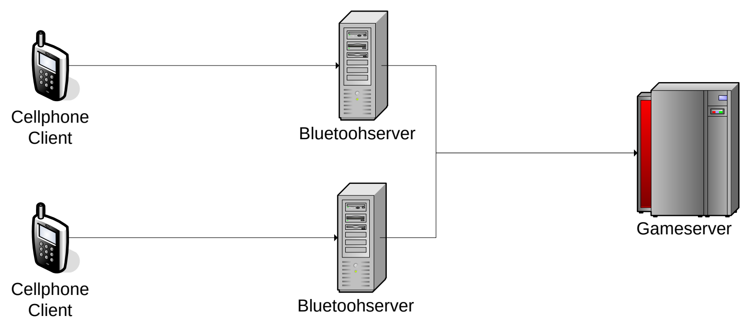


Figure 9.2: Several windows servers working as bluetooth hubs

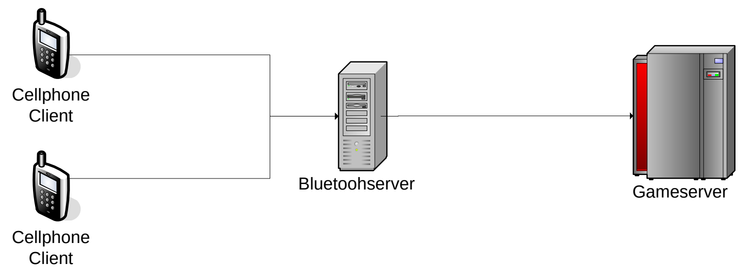


Figure 9.3: Single linux server working as a bluetooth hub

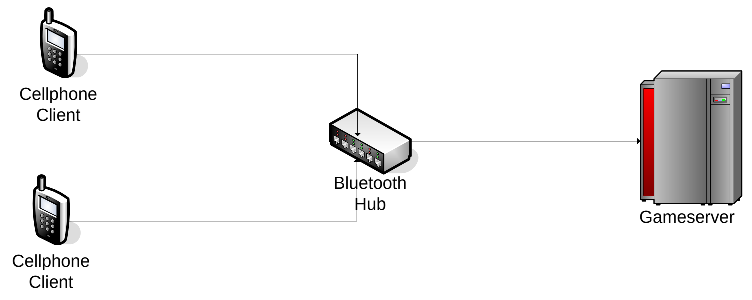


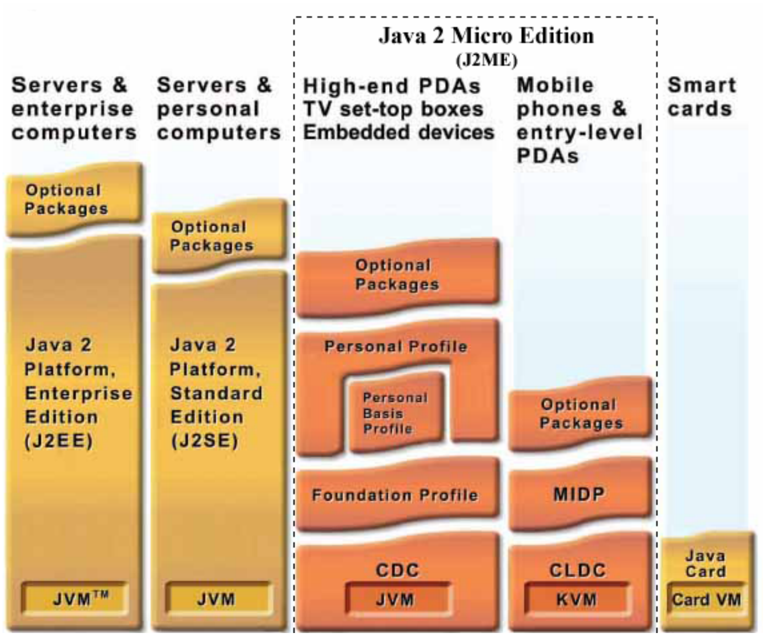
Figure 9.4: Standalone bluetooth hub

close to each other (worst case) we will loose 50% of the packages sent when we have around 28 piconets [19]. When we consider that each piconet can have 7 players, that traffic will be more sporadeously and low traffic, and that the access point probably has synchronized its 3 piconets, we can conclude that bluetooth should have more than good enough support for our needs.

9.5 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 9.5).



Java 2 Micro Edition
(J2ME)
Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Personal Profile
Personal Basis Profile
Optional Packages
Foundation Profile
MIDP
CDC
CLDC
Java Card
JVM™
JVM
JVM
KVM
Card VM

Figure 9.5: Overview of the Java environments

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.

J2ME has some limitations in addition to the J2SE, such as functionality being removed and the CLDC needing a preverification of the classes to reduce memory usage.

9.6 HP xw9300 Workstation

We have a HP xw9300 Workstation available (Figure 9.6), and we plan to use it as a server. The xw9300 is built on the most advanced hardware available at the time of writing this, making it able to deliver excellent computational and visualization performance.

Special features includes:

➢ Power-efficient simultaneous 64-bit computing

- Complete software compatibility with Intel®-based platforms
- Dual core processors
- AMD Opteron Direct Connect Architecture
- Dual PCI Express x 16 graphics and I/O
- SLI technology for NVIDIA Quadro enabled
- Built-in, high-performance disk subsystem
- Highly expandable
- Range of professional PCI Express 3D OpenGL-certified and multi-display 2D graphics offerings HP Performance Tuning Framework
- Intelligently designed tool-less chassis

Figure 9.6: The HP xw9300 workstation

Specifications for the workstation is as follows:

- Processor & RAM
  - Dual AMD Opteron 200 series processors with AMD64 Technology & HyperTransport; dual core processors 270 (2.0 GHz);
  - 16 gb ram
- Chipset
  - NVIDIA nForce Professional with AMD-8131 HyperTransport PCI-X tunnel
- Graphics card
  - NVIDIA 2xQuadro, SLI enabled

➢ Storage

  – 4x400 Gb scsidiscs

➢ Operating System

  – Windows XP x64

9.7 TellU

TellU is a small company located in Asker, which one of the project members has been working for during the summer. TellU has developed a library for use in developing of mobile applications. This library provides functionality that differs from the general by shoving more of the work to the server, and making the mobile device more passive.

The applications consist of so called actors, equal to the Java Beans in regular java. One actor serves a service-role, and will initiate actions from given input.

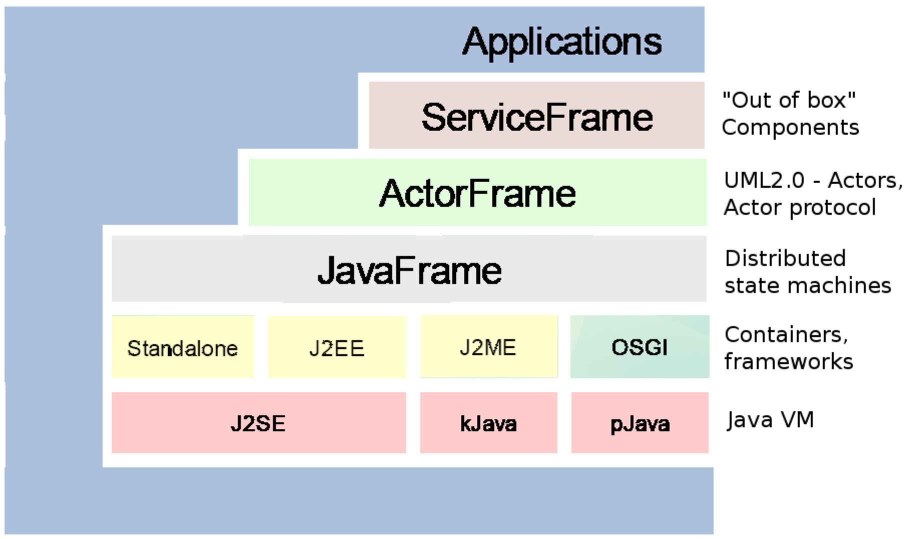


Figure 9.7: Serviceframe and actorframe overview

9.7.1 ActorFrame

An Actor is a composite object having a state machine (ActorSM) and an optional inner structure of Actors. Some of these inner Actors are static, having the same lifetime as the enclosing Actor, and others are dynamically created and deleted during the lifetime of the enclosing Actor. The state machine of an Actor will behave according to generic actor behaviour, common to all actors, and a Role type, which is bound when the Actor is instantiated. If the Actor shall play several Roles, this is accomplished by creating several inner Actors each playing one of the desired roles.

Communication between the Actor and its environment takes place via an in-port and an out-port. Internal communication among the inner actors is also routed via the ports.

The actor has a generic behaviour, inherited from the base Actor type, that provides management functionality. It manages the inner structure of Actors and the Roles they play. It knows the available Roles and the rules for Role invocation. This is provided to it in the form of a DeploymentDescriptor (DD). The generic behaviour handles role requests and will either deny the request or invoke an Actor to play the requested role or an acceptable alternative role. The generic behaviour also has the capability to add and remove roles, and to perform other Actor management functions. It keeps track of what Plays the Actor takes part in and is able to track and release all Roles in a play when the Play shall be ended.

The Actor has an assigned Role type, which defines the application specific behaviour of the Actor. The behaviour of a Role type is defined by a composite state - RoleCS.

A service designer will primarily work with RoleCS and its constituent Role features to design service behaviour.

An Actor has a unique identifier and ActorAdress. In addition, the ActorSM holds both generic attributes and application specific attributes.

ActorFrame itself is defined as a special Actor, the RootActor. This Actor will normally have an inner structure of Actors reflecting the needs of the environment. Each of these inner Actors may recursively contain an inner structure until atomic Actors having only a role behaviour and no inner structure are reached.

- Actor Forwarding

➢ Forward actor messages

➢ Unknown actors are sent to default gateway

➢ Skips lost messages

➢ No End2End security

➢ Supports multiple links types

- Router

➢ Exchange of actors with default gateway and known routers regularly

➢ Own actors and actors from non persistent actor domains are sent

➢ Dead actors are removed

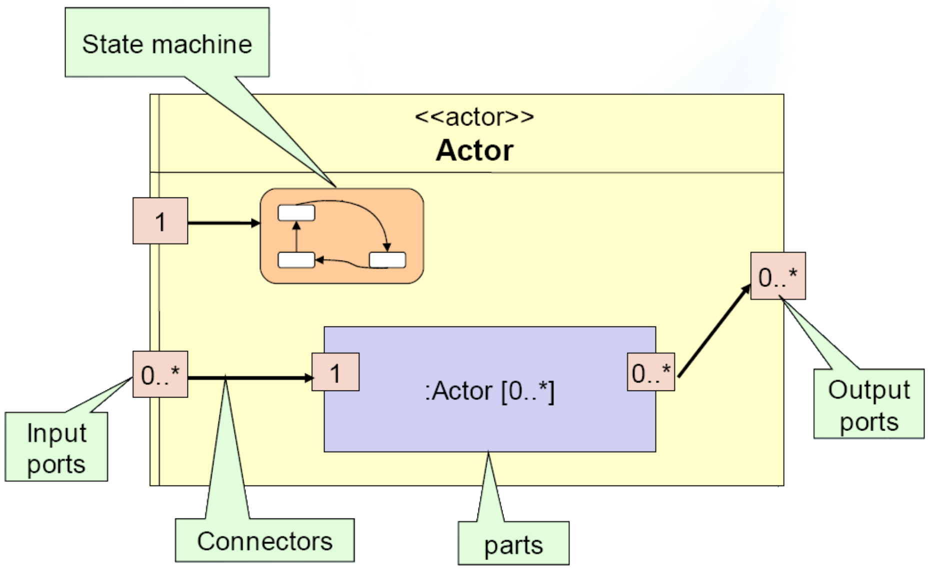


Figure 9.8: UML actor-statemachine

9.7.2 TellU ServiceFrame

Statemachine and message-routing framework for java J2ME, J2SE and J2EE. Has support for tcp-, udp-, bluetooth- etc routers. Events are sent using messages and the routing system takes care of where the messages gets sent. Very scalable.

ServiceFrame is basically a RootActor containing an open collection of actors reflecting the needs of the application domain and a library of generic Actor types and role types (classes) tailored to the domain. This is a general framework that must be specialized by supplying application dependant Actor types and Role types and instances.

Service frame allows designers to add new types and to specialize existing types incrementally. The Actors are interconnected through ports. The ports are in charge of message routing and have access to routing databases, and also the name-servers/registries needed to perform name based routing.

- Builds on a Peer-2-Peer architecture where client server architecture is a special case
- Event driven architecture build on asynchronous message passing and use of state machines for complex behavior (UML based)
- Application routing for a flexible integration and distribution of applications
- Scalable architecture allowing a flexible and dynamic deployment of applications
- ServiceFrame platform supporting applications on large servers to pc to mobiles, to sensors

- ➢ Integrated with external service enablers (SMS, MMS, Map)

9.7.3 TellU J2Me Gui library

Gui library for J2ME. Support for themes, animations. Has screens which consists of elements, and many different types of elements. The GUI library builds on the J2ME GameCanvas, I.E. all the elements consists of elements painted on the canvas.

Part IV

The MOOSES Framework Architecture

CHAPTER 10

Architecture

Software architecture is a field of study that is becoming more important every day. As systems becomes more large and complex, the difficulty of meeting requirements increases dramatically. The software architecture discipline is centered on the idea of reducing complexity through abstraction and separation of concerns. By achieving this at an early stage, the system is more likely to meet its requirements since it allows for reviewing design issues before implementation, reducing risk and costs. In this aspect, software architecture is of high importance for this project in hopes of avoiding a poor, low-quality, non-flexible and over-complex framework. This chapter contains information regarding system stakeholders, quality attributes associated with the architecture, architectural tactics[1]- and patterns[2]-choices made to achieve quality requirements. The practices used here are adapted from the *Software Architecture in Practice*, by Bass, Clements and Kazman [20].

10.1 Stakeholders

In this section we will try and explain who the different stakeholders are for the system, what interests they have in the system and what views are important for them. The different views are covered in Chapter 13.

Developers: The developers in this project are us (Sverre Morka, Aleksander Spro & Morten Versvik), and anyone extending the framework in the future or writing games for the

[1]Architectural tactics: Known methods for achieving quality in a system

[2]Architectural patterns: A description of element and relation types together with a set of constraints on how they may be used

framework. All the views presented in this document are important to the developers to get a better understanding of the thoughts behind the framework and how it works.

Maintenance: The maintainers of the framework is initially us, but will include anyone using the framework in the future. NTNU foundation are interested here for future research. Nova kino is interested here during the testing of the prototype at Nova kino. The physical view is important to this group to better understand the layout and functionality of the framework.

Financial: As this is a research project, everyone currently involved in the project are financially interested. That includes NTNU, TellU and the project group.

Users: The users of this framework will be the players of the game and administrators of the framework. The administrators will find parts of the process view interesting as well as the physical view.

CHAPTER 11

Requirements

This chapter will present the functional, non-functional and environmental requirements of the MOOSES Framework.

11.1 Functional Requirements

This section presents the functional requirements of the MOOSES Framework. Functional requirements are a set of instructions reflecting the functionality which must be implemented in the application. The requirements are presented in Table 11.1.

Functional Requirements	*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 11.1: 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.

11.2 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 11.2. 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 11.2: The non-functional requirements for the MOOSES Framework

11.2.1 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

11.2.2 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 MOOS 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 couple of days

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 re vote 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

11.2.3 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.

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

11.2.4 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.

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 billing option

11.2.5 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.

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

11.2.6 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.

U1 - Log inn and start playing	
Source of stimulus	The user
Stimulus	The user wants to log inn 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

11.3 Environmental Requirements

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

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.

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

CHAPTER 12

Design Decisions

This chapter discuss many of the design decisions made on how different parts of the framework could be implemented. It elaborates why the different patterns and tactics are used, and in some cases, possible alternative methods. This is to help get a better understanding of the crossroads we encountered, and how we dealt with these.

12.1 State machines

We wanted the framework to be as flexible as possible, but due to our time frame we also wanted it to be quick to design and implement. We took a look at the FSM (Finite State Machine) technology and found it to suit our wants and needs good. They provide for good extensibility, are fast and maintainable. State machine technology is described in Section 12.1.

12.1.1 TellU - Serviceframe

Serviceframe was a perfect fit for our base system, as their platform has support for the clients, server and a good architecture. It has also got another perfect feature; that the state machines can be placed anywhere in the deployed system (ie at different servers) without changing addresses. This means that we can move state machines to higher level servers or down again without any big impact. We used this feature for rapidly testing.

12.2 Message system

The message system follows the guidelines put from the TellU Serviceframe framework. Each message extends a default interface, and adds the functionality needed by the specific message. Reuse of messages is highly advisable, specially toward the mobile phone clients to keep the application as small as possible. The state machines use these messages to change between states or to handle information between themselves. The state machines use the instanceof check in Java to differentiate between the incoming messages and perform the necessary actions. In case that the state machine does not recognize the message, it will be discarded and the next message on the queue will be processed. Serialization is necessary in order to make the information suitable for transport between the server and client.

```
package messages;

import java.io.*;
import no.tellu.common.javaframe.messages.ActorMsg;
import no.tellu.common.javaframe.messages.StringHelper;

public class StartGameMsg extends ActorMsg {


        public String game;

        public StartGameMsg() {
    }

        public StartGameMsg(String game) {
                this.game = game;
    }

    public String messageContent() {
                return super.messageContent();
        }



    /**
     * This function implements the serialization of the object.
     *
     * @return a byte array with the objects data
     * @throws java.io.IOException
     */
    public byte[] serialize() throws IOException {
        ByteArrayOutputStream bout = new ByteArrayOutputStream();
        DataOutputStream dout = new DataOutputStream(bout);
        dout.write(super.serialize());
        dout.write(StringHelper.persist(game));
        dout.flush();
        return bout.toByteArray();
    }

    /**
     * Use this function for resurrection of the object
     *
     * @param data The serialized data containing the object data
     * @throws java.io.IOException
     */
    public ByteArrayInputStream deSerialize(byte[] data) throws IOException {
        ByteArrayInputStream bin = super.deSerialize(data);
        DataInputStream din = new DataInputStream(bin);
        game = StringHelper.resurrect(din);
        return bin;
    }
}
```

12.3 Controller setup

The players have to be able to control their characters on-screen. We came up with different controllers that all could work, but all of them imposes 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 is 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.

Therefore the framework needs to be flexible enough to support several types of controllers. To do this the framework abstracts the controller from the user object in the system. The controller type is connected to the user as he or she is logged into the system.

The controller must follow an interface for basic functionality, but due to some controllers having more functionality then others, this can be added in the final implementation of the controller. This functionality can then be used by the game developers if they so wish. However, to make sure that they reach the full audience they should try and make sure this functionality is not critical to play the game.

An example of this is if we have two players in a game, one using a mobile phone as a controller and the other a console-type controller. The game controller modules default functionality is accepting key presses from both of these. The mobile phone however has added functionality as it can display game information which is not critical to playing the game. By using the state machine and message system described, both the controller modules will receive and send all the messages the game supplies (key presses and mobile phone screen updates). The state machine is however designed to ignore messages it does not support, so both should work perfectly.

(a) Playstation type controller (b) Mobilephone type controller

Figure 12.1: Examples of controller types

With the possibility of multiple controller use, there is an added responsibility to the game developers when creating games. Not only do they have to make the game, but also specify which controller to load. This could either be a default delivered with the framework or one they design themselves after the specifications given by the framework.

12.4 Game and game client

We do not want the framework to have to be heavily modified to add more games and functionality after a final deployment, so the framework should have to support some kind of ”pluggable” games. With the use of different controllers (as described over), this would mean that each implemented game would have to define what implemented controller module to use. To make the games pluggable, a default interface will be supplied to make sure the games support the minimum needed functionality of the framework. These games can either be written in Java or C++ with the use of JNI, in which case an interface for the C++ game to talk back to the framework is also needed (The JNI is described in Section 14.1.1).

The desired game control client is loaded in real time prior to starting the game.

Due to limitations in J2ME (no class loader) any added gameclient requires a repackaging and redistribution of the mobile client as of today. As mobile devices advance, we are certain this is a feature which is not far off (e.g. Anhinga framework mentioned in [21]).

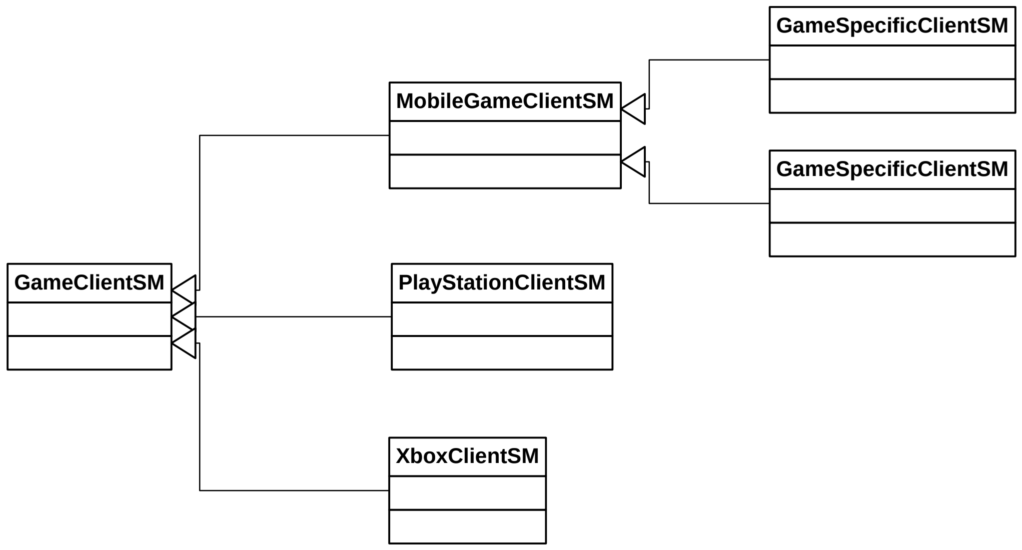


Figure 12.2: Overview of the client abstraction and possible implementation

12.5 Billing

To commercialize the framework, some sort of revenue income must be supported. Some possibilities are:

- ➢ standard: buy hours or a monthly subscription.
- ➢ arcade style: pay to join a turn, character lives or for upgrades. Rewards good players.
- ➢ items have a value, take a percent of transactions between players (micropayment).
- ➢ ad sponsored, include commercials in the game.
- ➢ combination of the above.

Since research into the business models of this concept is provided to another student (Section 1.2), we need to let the framework be open enough to support different billing methods. This would also let different vendors utilize different billing methods if needed.

CHAPTER 13

Design Overview

When the requirements for a system is established, it is important to get the base design down to get a rigid and easy-to-maintain system. This can help weed out problems at an early stage, and will help others to get a better understanding of the systems inner and outer workings.

This chapter will focus on the design and architecture of the framework. First we have a description of the high level architecture and the design patterns we have incorporated into our design. Subsequently more detailed views tailored to better explain the architecture for the different stakeholders is provided.

13.1 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 13.1. 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.

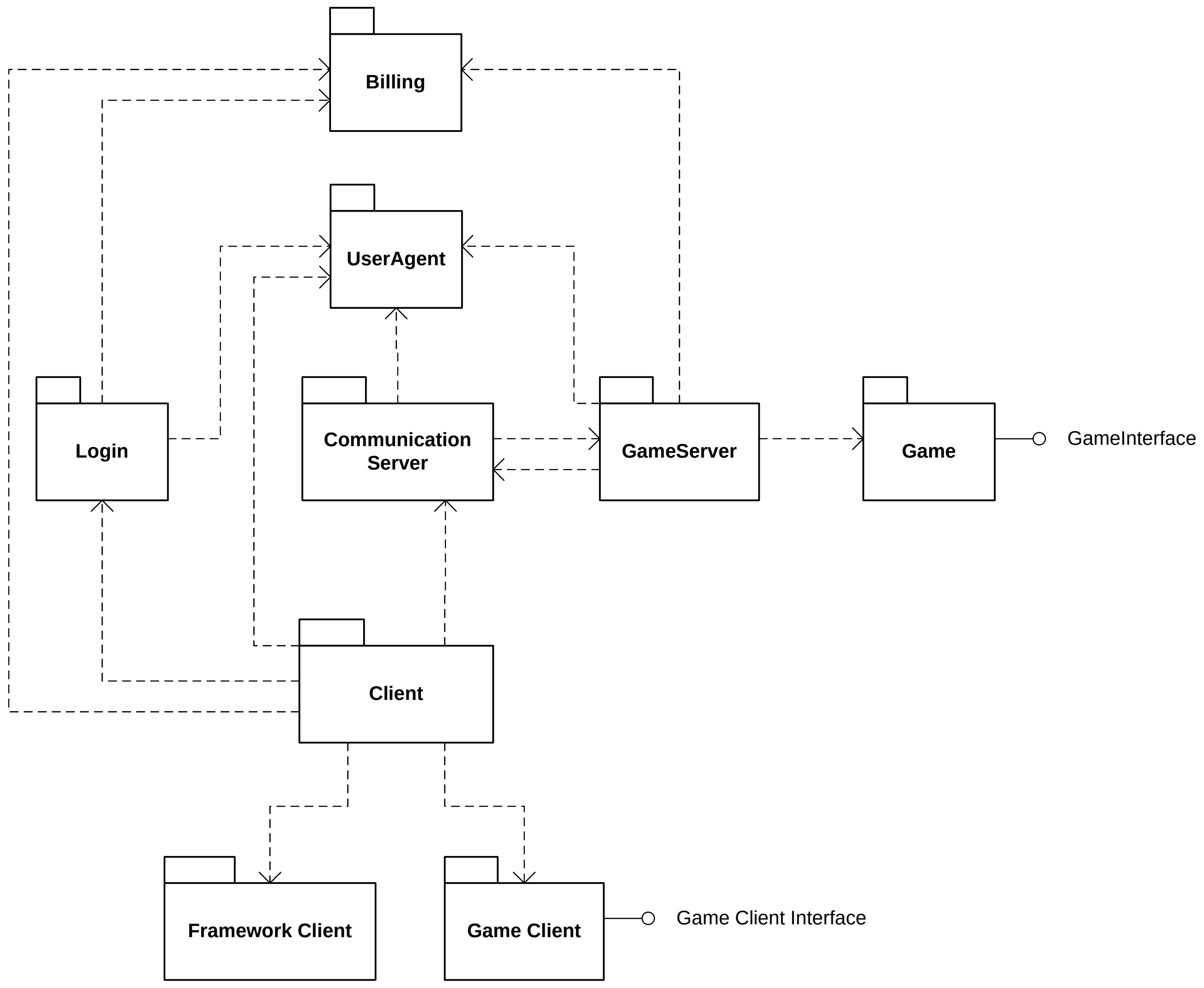


Figure 13.1: High level architecture view of the framework

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 on 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 tailer login types specifically to the system owner.

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 tailer 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 to ability to tailer controllers for their games.

13.2 Model View

The module view shows the decomposition of the systems overall functionality into modules and nodes. This is to help get a better overview of the workings of the system for developers. The module view can be seen in Figure 13.2.

13.2.1 Connection Server

This node is the backbone of the framework. It contains modules for administering users, logins, billing and running game servers.

Server starts up the other server modules and and runs a ”name server” in order for modules to easily establish communication with one another.

ScoreKeeper is a module for pushing and popping user scores to a database, it receives scores from the GameActor and stores the in the database immediately. Any other module interested in scores can request and have it sent to them from this module.

Billing is responsible for checking user billing when a user tries to log in. If the user does not have a viable billing option, one can be provided to the user.

Authentication is used to log users into the system and connect a client to a useragent. The authentication uses the billing module to check for viable billing for the user.

UserAgent module is the users ”object” in the system. Each user get their own agent. It is used throughout the system to verify the user.

GameServerActor is a module that keeps track of all the running GameActor servers in the system setup. This comes to use with the use of multiple game servers.

13.2.2 Mobile Client

The mobile client node is the users access to the framework. It contains modules for user login and framework user options, as well as game clients for interfacing with the games.

GameMobile is the framework client for mobiles, handling functionality specific to the framework towards the user like logging in, handling game setup etc.

GameMobileApp starts the client module and starts the initial connection to the framework.

GameClient is the client for specific games. Each game developer has to specify which game client to load. Handles input and output from the running game the client is connected to. More information on game client development can be found in Section 12.4.

13.2.3 Game Server

The game server node is responsible for handling the running of a single game. It contains logics for voting for what game to run, start and stop games as well as showing high scores at

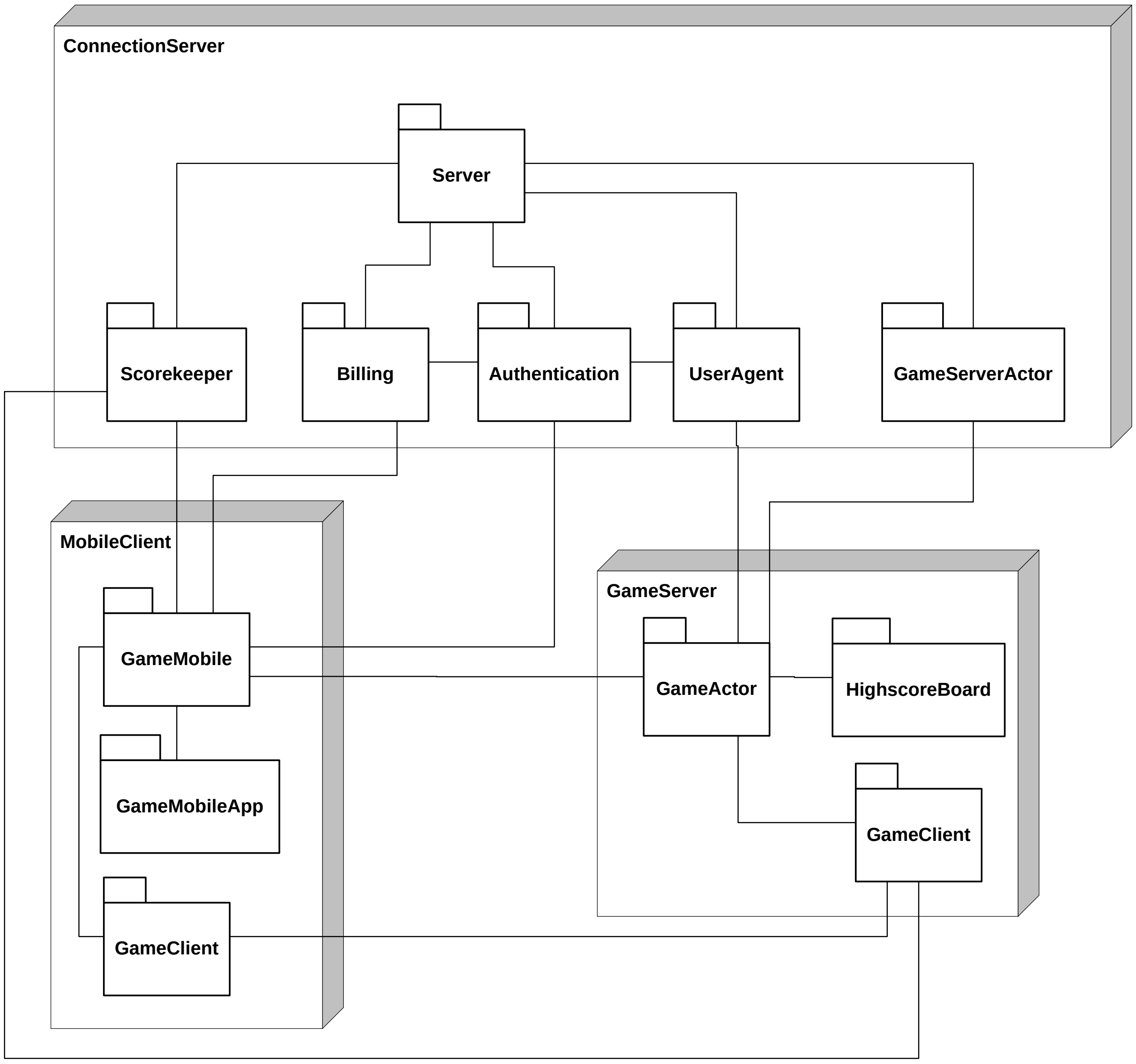


Figure 13.2: Module view of the framework

game stop.

GameActor is responsible for running the game server, handling the vote system, high scores and user connection. The user connection is also a ”presence” detection system, noting what game server the users should be connected to.

HighscoreBoard gets input from the GameActor once a game is finished to show the highscore for the previous game.

GameClient handles game input/output to the GameClient, as well as pushing high scores to the GameActor once the game has ended. Each game developed must follow a simple interface in the framework as a minimum of functionality. More detail for game development for the framework can be read in Section 12.4.

13.3 Process View

To get a better explanation and overview of how the different components run and relate in real time we have taken use of the process view. This view will enhance the understanding of how communication sequentially occurs between modules in certain tasks put on the system.

13.3.1 Login process

The login process starts with the client first requesting connection with the connection server. If successful, the connection server starts up a user agent for the client and returns connection information to the client. The client then moves into a login state where the user is asked to type in a user name and password, which is sent to the authentication module. The authentication module checks the login information, and informs the user of the success. On successful login the authentication module also requests a billing check on the user with the billing module. If the billing is successful the user is granted access to join games. In the event of a user not passing the billing check, the billing module will inform the user and present him/her with a billing option(this depends on the billing module implemented). The login process view can be seen in Figure 13.3.

13.3.2 Joining games

Once a user is logged in with the network he or she can choose to join games. There are two scenarios in which this can happen:

- ➢ Join while a game vote is in progress
- ➢ Join while a game is already running

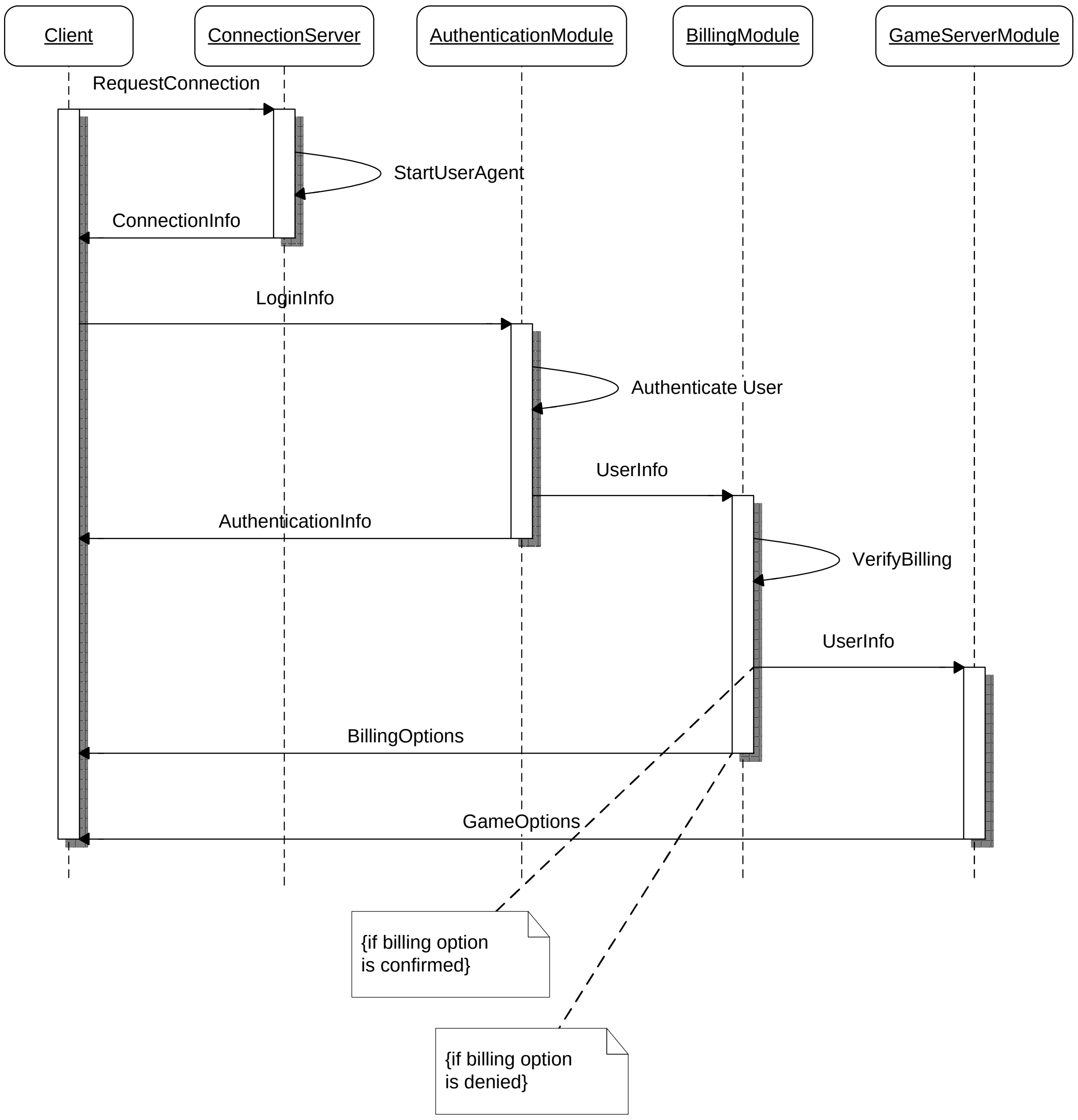


Figure 13.3: Login process view of the framework architecture

Once a user has asked for a connection to the GameServer module, the server checks what state it is in. Following is a description of how these two different processes proceed.

Vote and join game process

If the server is in a voting state, it sends a list of the available games to the client and lets the client vote. In a case of a draw, this process continues until a winner is found. Once a winner is found, the server starts up the game, and tells the client which game client to load. Once the game is started, the server sends over the users to be added to the game, and sets up a connection so the clients and game speak directly to each other. While the game runs, the client and game exchange input and output. When the game ends, the game sends a message to the game server, which tells the client to unload the game client, and lists the highscore for the game. The process view for voting and joining a game can be seen in Figure 13.4.

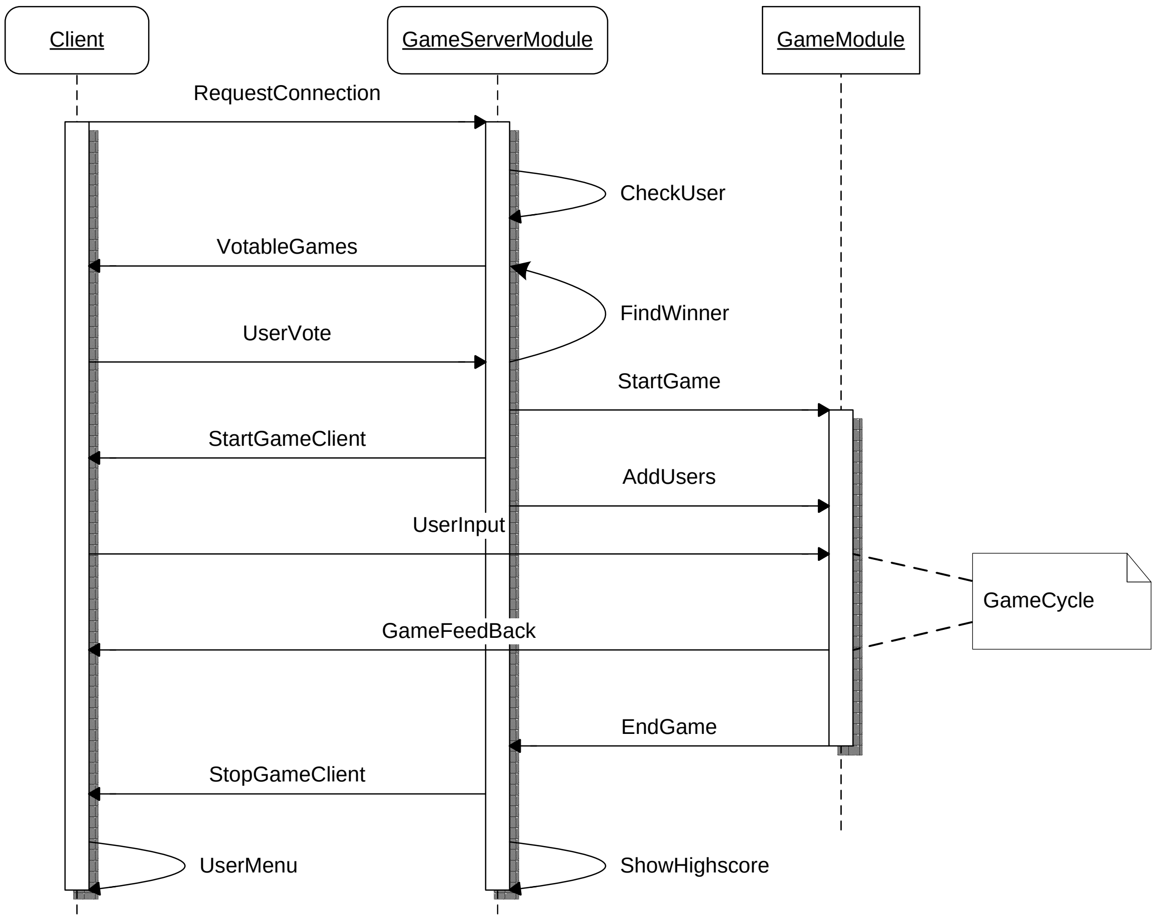


Figure 13.4: Gamesetup process with voting

Join a running game process

If the server is already running a game when the client joins in, the server checks the running game and lets the client know which game client to load. The server also sends the user information to the game, adding the player and setting up communication between the two. The user and the game then exchange game information while the game runs. At game end, the game sends a message to the server letting it know the game has stopped. The server then tells all the clients logged to the game to stop the game clients and shows the highscore for the game. The process view for joining a running game can be seen in Figure 13.5.

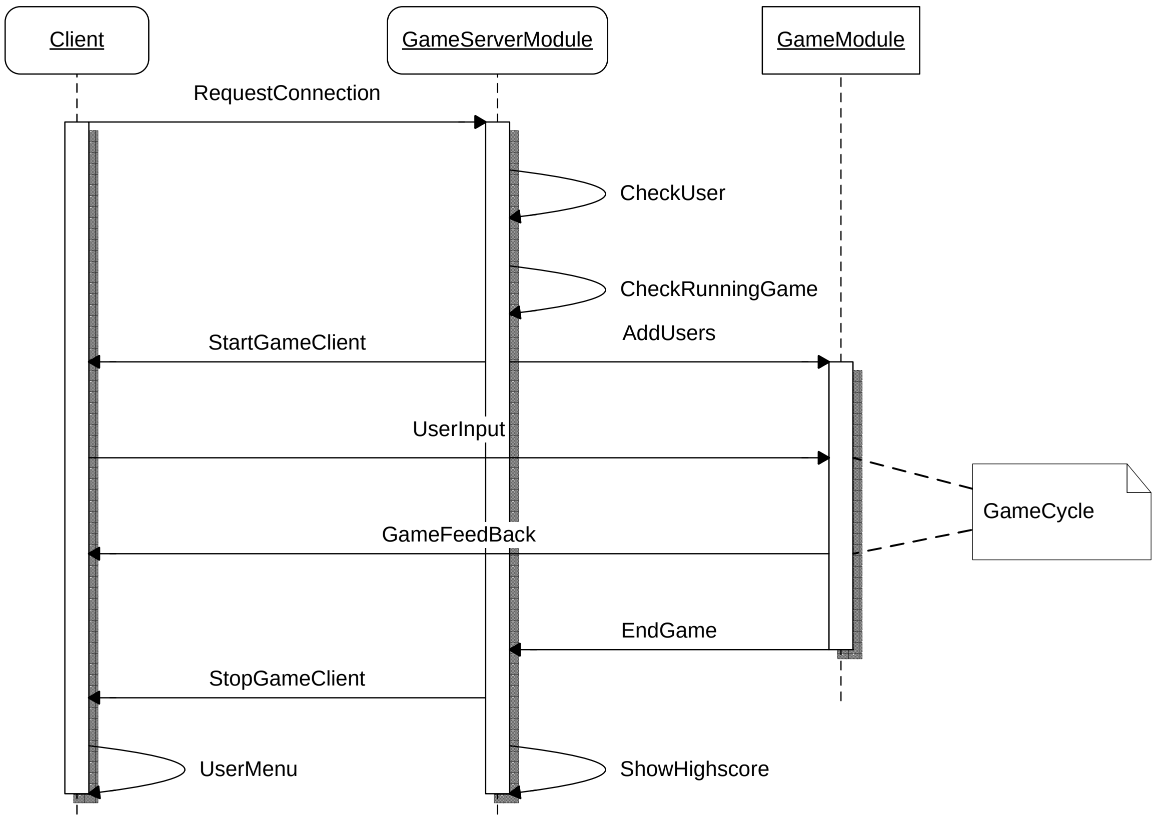


Figure 13.5: Gamesetup process with joining a running game

13.4 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 a 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 13.6.

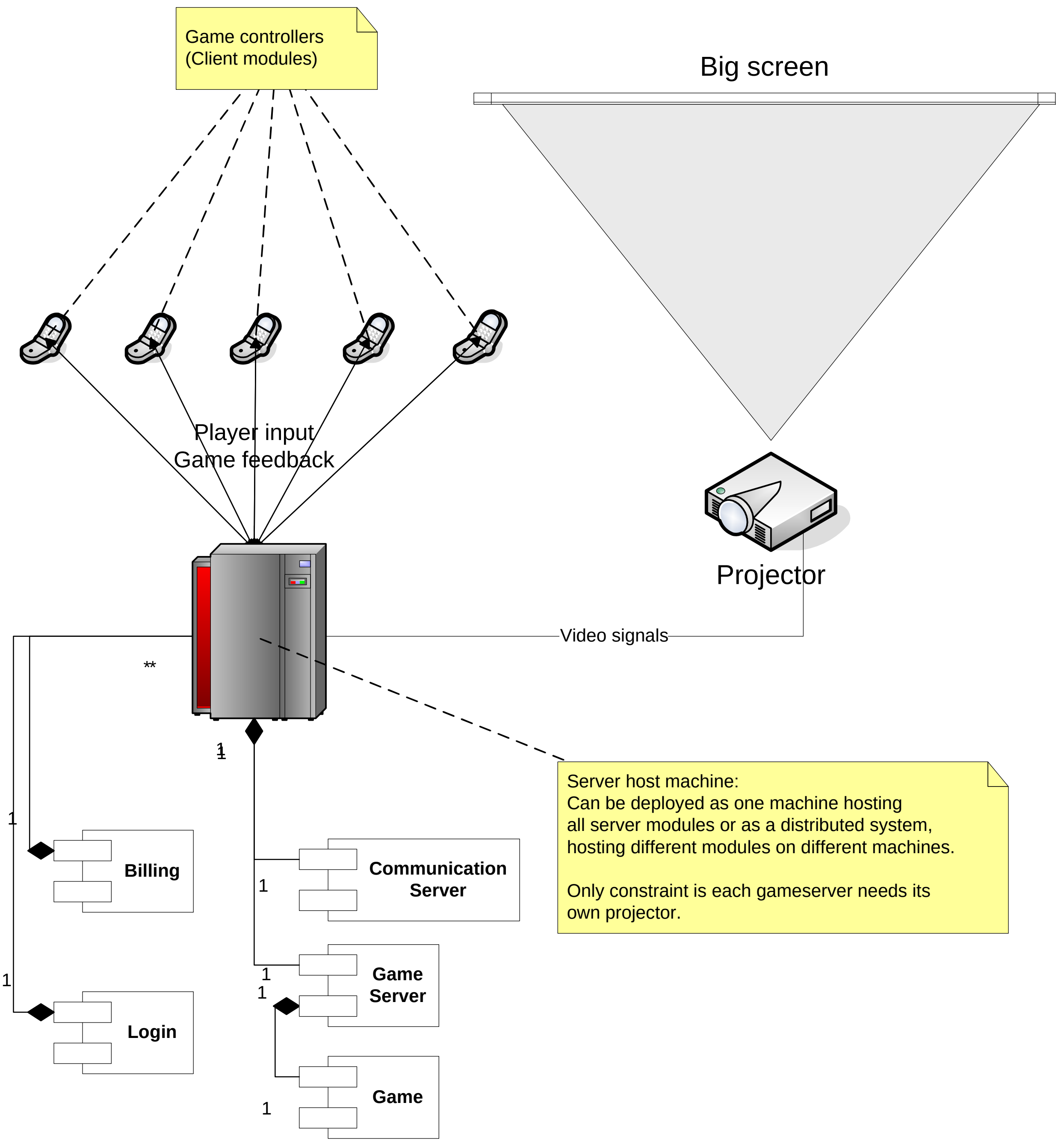


Figure 13.6: Physical view of the framework architecture

Part V

The MOOSES Framework

CHAPTER 14

Framework Design & Implementation

In this chapter we present decisions specific to the implementation of the architecture.

14.1 Framework implementation

The implementation of the framework presented a few issues which we solved as described below. Our framework will use TellUs J2ME library for device clients implementation. However, the games has to be built in C++ because of Javas limitations with respect to resources. This lead to the need to connect Java with C++ code. We solved this by using Java JNI.

14.1.1 Java JNI

JNI (Java Native Interface) is a programming framework that allows native access to and from Java. It is used to be able to use Java directly with other languages, access libraries and use OS-specific capabilities that Java can not include in its standard class library. JNI is also commonly used when performance is of utmost importance, you can write the function(s) in a faster language and return the results to Java.

14.1.2 Billing

The current implementation of the billing module is fairly empty. It validates any user check given. This was due to lack of information on billing systems, which was apart of another project. The module is however very modifiable, and it should be easy to implement any chosen billing functionality in the future.

14.1.3 Modifications to Serviceframe

To decrease startup time to the minimum, the currently connected Bluetooth address is cached. So if the client is used at the same place, it will connect immediately.

The bluetooth accesspoint could not use UUIDs larger than 32 bits, which made discovery fail because Serviceframe was hard coded to 128 bits. So we had to make some changes as to how UUIDs are set to support both 32 and 128 bits UUIDs.

We wanted to make the devices context aware in the sense that they should start the correct game on the device to match the game in the area where the player is. To be able to support this TellU made it possible to get the actors (state machines) visible from the routers. We used this to filter out the actors we were interested in and send them which game it should start.

CHAPTER 15

Test Game Implementation

In this chapter we present decisions specific to the implementation of the test game.

15.1 MOOSE (Multiplayer On One Screen Extermination)

The test game is written in C++. It uses OpenGL for graphics, OpenAL for audio, GLFW for window management and GLTT which is a truetype font library for OpenGL. Threading is managed by pthread which is a POSIX standard thread library. To stream music it is using ogg libraries which are royalty and patent free. Image loading/saving is accomplished via the image library DevIL. All of the libraries are compatible with linux and windows.

We chose to use OpenGL for graphics because it utilizes the 3d graphics card for rendering. The graphics card is much faster at drawing than we could have done in software, and we can use it to draw other nice graphics effects at almost no cost. Scaling and clipping is also available free of charge. Todays cards can also provide us with extremely fast advanced mathematics per vertex and per pixel via vertex- and pixel-shading. Effects we could have taken advantage of is ie lightning, bumpmapping, shadowing to increase the visual quality.

The glue between Java and C++ is JNI. We used this to be able to take advantage of Serviceframe and the performance and availability of libraries for C++. We tried to keep the interface between the languages as few and simple as possible. This is recommended when using JNI, and also reduces dependencies and simplifies the development. This made it easy to create a standalone version of the game (controlled via keyboard) which used the same underlying code as the version using JNI, which in turn made debugging easy.

15.1.1 Test game features

The test game implements the following features:

- 2d graphics with animations, loaded from gif and png images.
- background music player (repeating all tracks from a directory).
- randomized sound effects (machinegun firing, planes, bombs etc) placed randomly in the 3d audio system with a random delay inbetween.
- sound events (explosions and deaths).
- player movements controlled via the framework or via keyboard (standalone). Movements can be aim up/down, move right/left, jetpack, switch weapon and fire.
- player names rendered over the player with an energy bar under the text.
- 2 layered map (foreground and background), with 1 blending layer for 1 pass hardware blending.
- keeps track of and updates the players frag and death count, score. Sends vibration messages to players who loose energy.
- simple physics with accelerations for gravity, friction.
- collision detection, sprite - geometry.
- has support for 5 weapons as of today, more are easy to add. Timed explosions (grenades and nukes), shots (pistol and mp5) and trajectory weapons (bazooka). Variations include reload time, number of explosions, explode on contact, explosion size and damage.
- has support for unlimited instances of weapons, players and explosions.
- the map can be completely destroyed, players shoot holes in the ground.

15.2 Test game controller

The controller application has been developed in J2ME using the Serviceframe framework. The client is based on one state machine controlling the other canvases. The state machine also handles communication with the game server. The client application switches mainly between two functions, voting and game controlling, in whihc state is set by the server. Although the framework support generic messages to abstract the communication, we choose to specify the messages discreetly, creating a new class for each message.

The client handles communication with the game written in C++ through JNI. Fire messages will be sent to the game, and status messages and death messages will be sent from the game to the client. Whenever a death message is recheived, the client is suspended for ten seconds. The controller controls the ammunition, reloading times, current weapon and ability to shoot. The health, score, frags, and deaths are controlled by the game. Current status for each weapon, score, deaths, frags and remaining health are displayed on the controller. The controller also provides sound feedback from the weapons fired, while the game handles the impact sounds.

The controller has another function, the ability to vote for a game. The server will determine if the client should initiate the voting canvas or the game controller canvas. The server sends a list of available game alternatives, in which the client votes on one game, and the vote is sent to the main vote canvas through the server. Once a the vote is closed and a game has received more votes than the others, the server will tell the client to initiate the right game canvas. If a client logs on during the game, the server will send a message to the client indicating which game canvas to start.

15.2.1 Test game controller features

The features provided by the test game client is as follows:

- Server determines which canvas to load
- Ability to join in the middle og a game
- Graphic representation of status, score and selected weapons
- Sound feedback when firing weapons
- Death canvas
- Voting opportunities
- Simple animation
- Communication to game through JNI
- Weapon handling
- movement controls

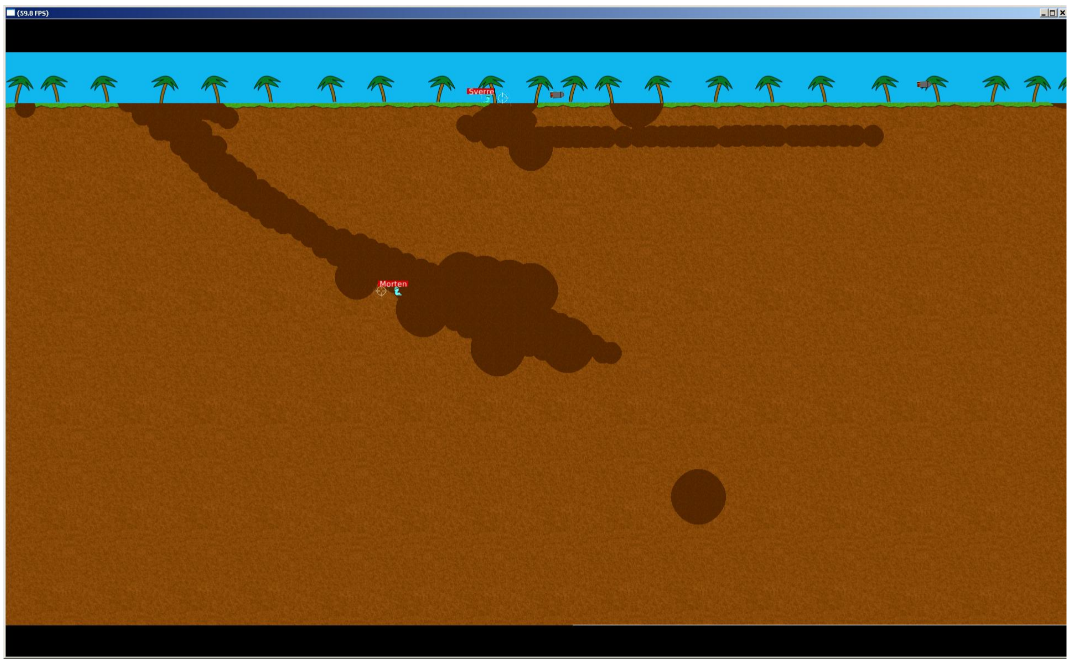
15.3 Limitations

Due to J2MEs missing support for user-defined class loaders we can't transfer new classes over wireless and start them on the client. This means that we can not support new games by downloading standard Java classes and starting them. To work around this we would need to make a standard interface for all games, upload new midlets when new games are introduced or use a higher level scripting language.

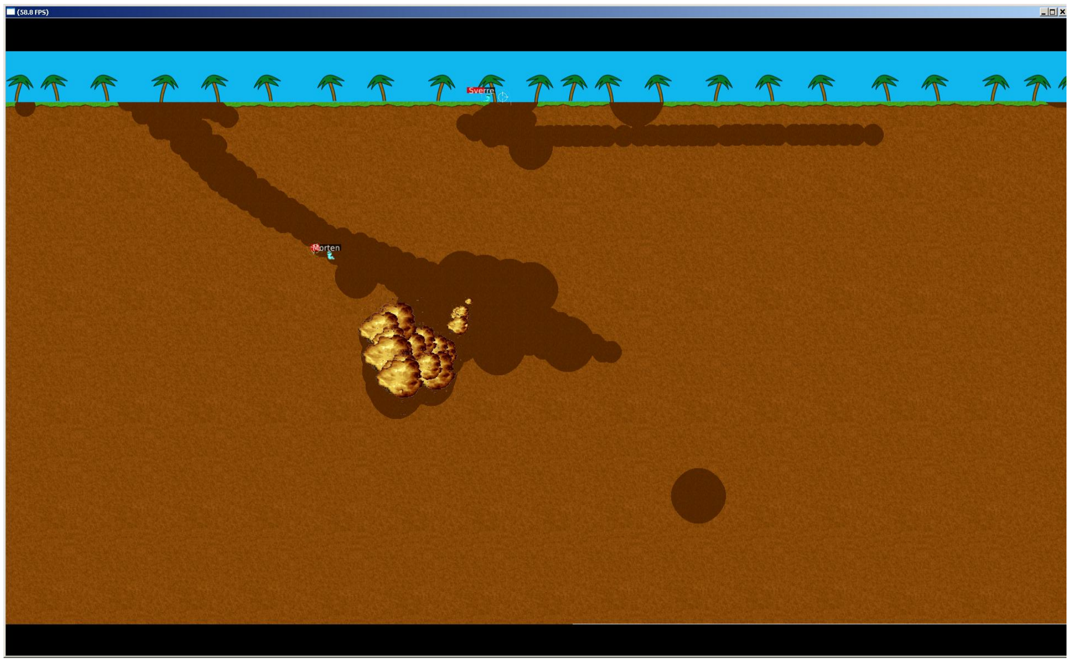
A standard interface wouldn't utilize all the possibilities a game can have. Uploading a new midlet each time a new game is introduced is very cumbersome. There are several existing implementations of scripting languages for J2ME, ie Hecl [22] or Simkin [23]. This is outside this projects scope, but is a very desireable feature to implement.

15.4 Screenshots

We will present some screen shots from the game and the game controller.

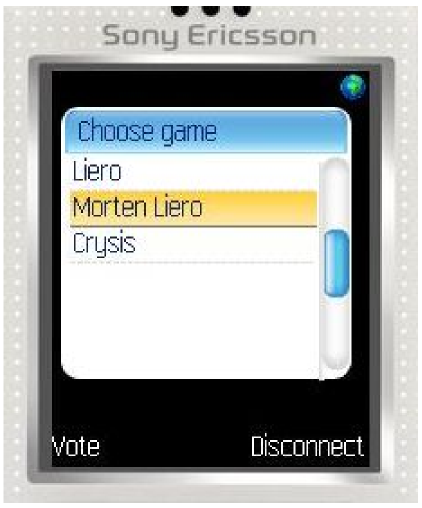


(a) Testgame

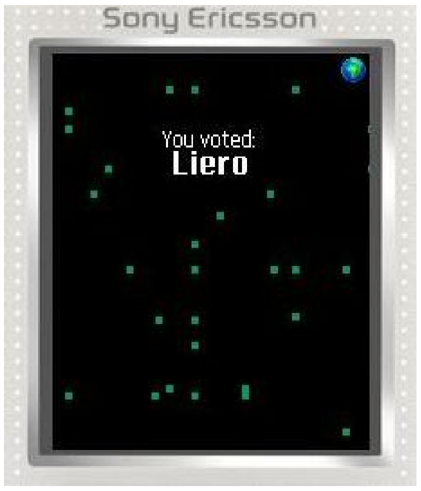


(b) Testgame

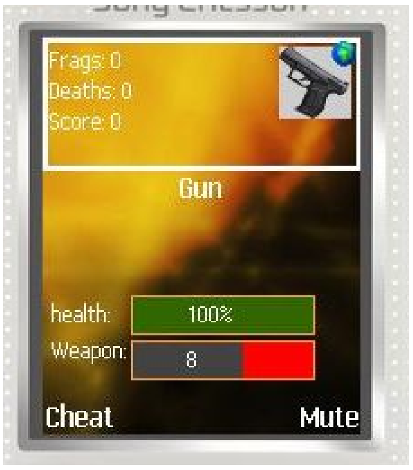
Figure 15.1: Test game screenshots



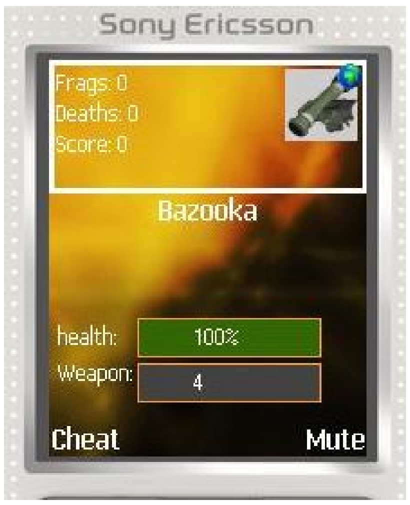
(a) Voting canvas



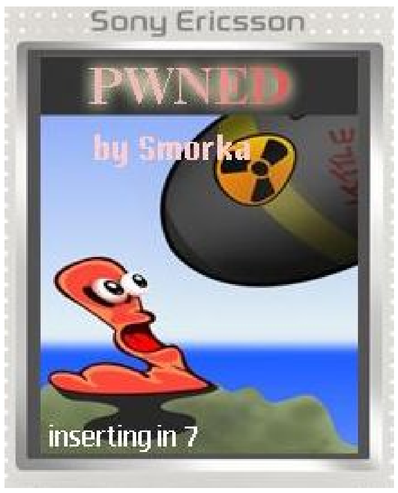
(b) Vote sent



(c) Game controller



(d) Game controller



(e) Death canvas

Figure 15.2: Test game controller screenshots

CHAPTER 16

Game Developer

In this chapter will give an introduction on what is needed to develop games and game clients for the MOOSES framework. It is recommended the developer has made him / her familiar with the architecture (Part IV) to get an understanding of the underlying technology. The test game and its game client should also be studied.

16.1 Game Development

Game development with the framework is almost the same as making a stand alone version. The interface is kept very small, you have to implement 3 functions to make a game controllable by every player.

The game must be made in a new thread, and should run full screen at the highest possible resolution. When the server starts the game it will run the `startGame` function, which should return without too much delay. The next step is adding players. The server will loop through all ready players and run `addPlayer` with each players id and nickname. The server takes care of authenticating the players, checking for payment and/or billing.

While the game is running a player should be removed when `removePlayer` is run, either because the player quit or timed out.

When the game is finished it must call `stoppedGame` to notify the server that the game is complete. The server will then save the scores and display a highscore list.

To leverage the vibration capability of the controller, we have made a function to send a message to the controller with how long it should vibrate. The developer can just call `vibratePlayer`, with how many milliseconds it should vibrate, and the framework will take care of the rest.

Further extensions, like providing for new features on the controller, can be done by adding functions to an extension of `GameInterface` at server side and on top of GameClient at the client side.

16.2 Game Client Development

Currently the only implemented game client is for mobile phones. The superclass for this client is located in the package `actor.gameclient` in the hpjavaj2me directory. The state machine for this client follows the standard TellU setup with one composite state class and one state machine class, respectively `GameClientCS` and `GameClientSM`. To follow the frameworks package structure, new clients should be made under the packagename `actor.gameclient.gamename`.

The basic class for the clients state machine only handles the loading of the client canvas to the screen. The canvas must be implemented in the extended client package. The canvas must extend the `no.tellu.cdlc.gui.GuiCanvas` class. The canvas handles user input and use the client state machine to send the information on to the framework. The test game canvas `MOOSECanvas` can be used an example to study one way of implementing it.

The extended client state machine will also need to implement the handling of user input and feedback. This logic is based in the composite state of the new game client in the `active` state. Basic messages for handling key input and vibration feedback in the mobile phone is present in the framework for use (see the implemented client for the test game). The test game client in the package `actor.gameclient.moose` can be used an example for study on implementation.

Part VI

Testing

CHAPTER 17

Testing

We have performed testing at Nova cinema in three occasions. The first time we gathered information, the second time we tested a limited version to gather intel regarding bugs and technical challenges at the cinema. The third time we had a representative version, and we tested multiplayer. Along with all the testing at the cinema we have tested the game locally several times during development of the framework.

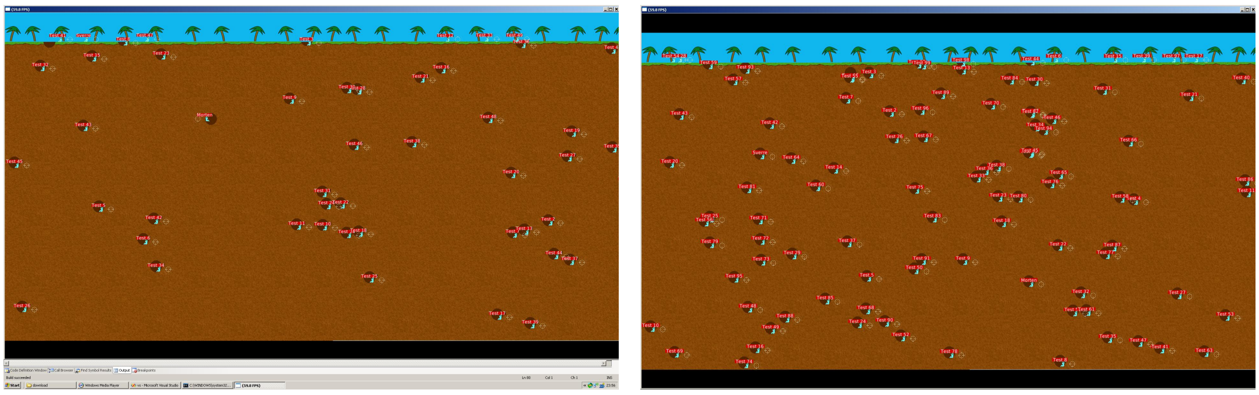
17.1 Development of testgame

The modularity of the framework architecture let us test different modules as they were being implemented. We wanted to test a sufficient number of contemporary players, the Bluetooth coverage and latency, and the physical number of players supported over Bluetooth.

We have found that our game may be capable to support approximately fifty players, but this represents the upper threshold.

As you can see from the pictures (see Figure 17.1), fifty players on the same screen provides just enough airspace to focus. As the land gets destroyed, there will eventually be even less airspace. Which constrain the session to a few minutes at a time.

We planned to use usb Bluetooth sticks during testing. Our test environment was Microsoft Windows, which unfortunately did not support more than one Blutooth device at the same time. Fortunately we were able to borrow a Bluetooth hub, which solved that problem. With the usb Bluetooth we experienced minor lag as the distances increased, but this problem also seemed to be eliminated with the Bluetooth hub.



(a) Implementation with fifty players (b) Implementation with houndred players

Figure 17.1: Visual test of how many players the screen will support

We have not been able to test the game with more than four mobile clients due to the lack of phones.

17.2 First test session

The first time we tested resolutions and appropriate sizes of objects to be displayed using JPEG pictures. We concluded that the characters should have a 16x17 pixels resolution to give room for quantities of them without compromising the empty space on the screen. We also saw that the 1920x1080 resolution fitted the canvas as well. The most appropriate font size proved to be 12 pt.

17.3 Second test session

After weeks of development we booked the cinema once again to do some testing with a nearly complete test implementation. Unfortunately things where not going as smooth as we had expected.

We brought all the required equipment, the HP Workstation, Bluetooth router, 2 mobile phones with MIDP 2.0 and Bluetooth 2.0 support as well as the implementation files. As mentioned we had some starting problems. The first one to appear was to find a keyboard that worked with the HP server. From there we had some problems with the server running on a 64 bit operating system. When the server was deployed, all that was left was to deploy the router in the cinema auditorium. Unfortunately, the network in the auditorium was not connected to the net in the cinema. However, we where able to deploy the router in the control-room. This way we got to test one of the mobile phones in the auditorium. The communication to the worked ok, but we experienced some lag due to the obstacles between the router and the mobile. At least we got some reflection and attributes that provided a basis for further developing.

17.4 Third test session

Monday the fourth of December we returned to the cinema together with our supervisors, and an audio technician. Our test implementation lacked only a few features to make it a complete implementation. These features where mainly the ability for clients to come and go as the game where running. This feature where fixed later on the same day, but we did not get to test it in the cinema. Also the high score did not represent reality, but were statically coded into the framework.

This time things worked more smoothly, and it did not take much time before we had the game up and running. All we needed to do was to install the game and server to the HP machine, and deploy the router in the cinema auditorium.

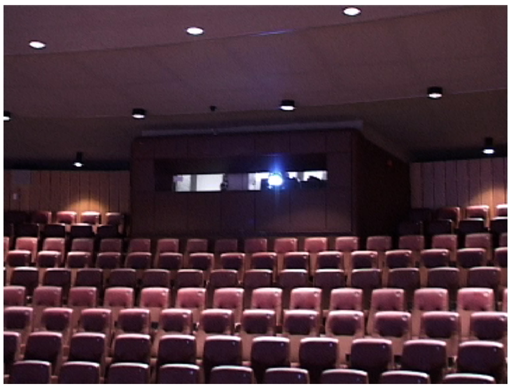
The server started the voting screen, waiting for mobile phones to connect. Our current implementation require the client to be installed manually on the mobiles. After the distribution of the client, all players initiated their votes, and the game started.

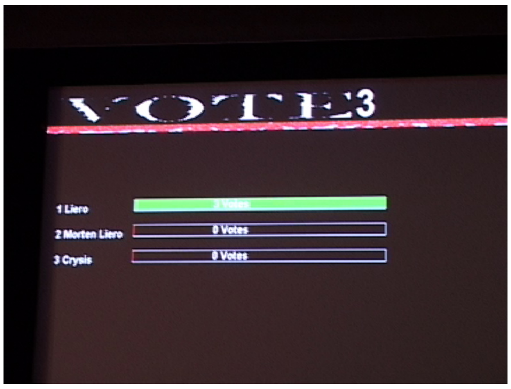
From here everything went smoothly. There were no lag worth mention, and the controller worked optimally even in the back row of the cinema. In fact, all we did this day was to play the game for hours. The problems we experienced was tied to the game itself rather than the concept. The game proved to be a good fit to the concept, although we did not get to test it with more than four players because of the lack of phones. We plan to later on test the game with 21 players which is the maximum of peers the router supports. Also some minor bugs where discovered. We had an issue regarding the background music due to a damaged file causing the game to crash. Removing the file solved this problem. Also there was an incident with synchronization of the voting canvas on the client and the server. Voting in the last second caused the client not to connect to the game although it where represented in the game visually.

We received good feedback from the people present, and everyone was very enthusiastic to the concept. We take this criticism as a good motivation for further work and research in the concept.

Our supervisor Alf Inge brought a video recorder with him and made a movie of some parts of the event. The movie is included on a DVD.

17.4.1 Pictures from testing at Nova

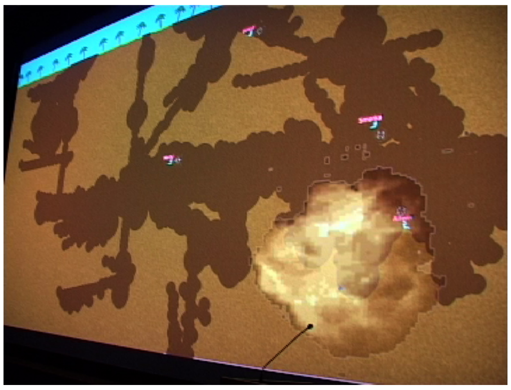

(a) Nova projector

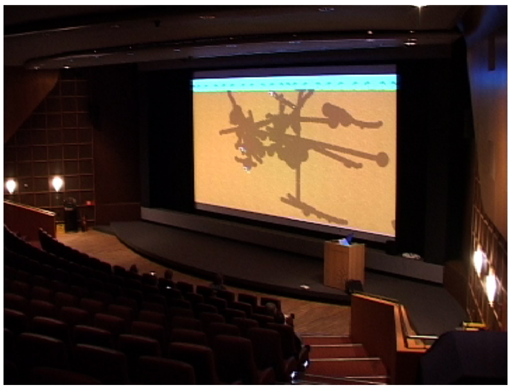


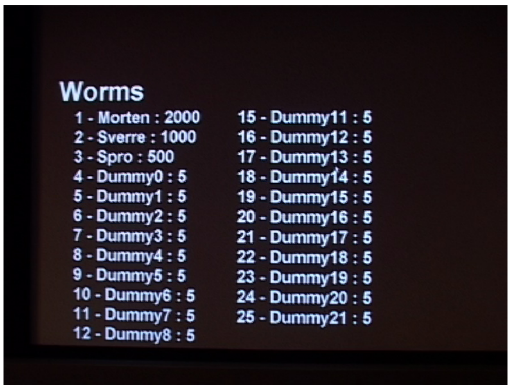
```
VOTE3

1 Liero          3 Votes
2 Morten Liero   0 Votes
3 Crysis         0 Votes
```

(b) Voting screen

(c) Happy testers


(d) In-game screenshot


(e) Overview



```
Worms
 1 - Morten : 2000     15 - Dummy11 : 5
 2 - Sverre : 1000     16 - Dummy12 : 5
 3 - Spro : 500        17 - Dummy13 : 5
 4 - Dummy0 : 5        18 - Dummy14 : 5
 5 - Dummy1 : 5        19 - Dummy15 : 5
 6 - Dummy2 : 5        20 - Dummy16 : 5
 7 - Dummy3 : 5        21 - Dummy17 : 5
 8 - Dummy4 : 5        22 - Dummy18 : 5
 9 - Dummy5 : 5        23 - Dummy19 : 5
10 - Dummy6 : 5        24 - Dummy20 : 5
11 - Dummy7 : 5        25 - Dummy21 : 5
12 - Dummy8 : 5
```

(f) Highscore

Part VII

Conclusion

CHAPTER 18

Conclusion

In Chapter 4 we stated two research questions. We wanted to study current games to find out what type of games were suitable for multiplayer games on one screen. We also wanted to see if it was possible to create a back-end framework to ease the development of games in such a scenario. Throughout this project we have worked according to the methods described in Section 4.2. We have designed and implemented a back-end framework for use with multiplayer games on one screen, as well as provided an in depth study into which games works for that scenario.

In this chapter we summarize the projects results by answering the research questions. The content of Chapters 8, 10 and 14 constitutes the basis for answering these questions.

RQ-I: **What games are suitable for use on large projection-screens in multiplayer mode?**

- ➢ The research showed that most games today borrow features from one another, and that the lines between genres are becoming more diffused. We concluded that games from genres like Adventure and CRPG work very poorly with our concept, while genres like Sports, Racers and Fighters could work in a limited fashion. Shooters seemed to be the best genre to work with our concept and we used this for our test game (see Table 8.2 for a full comparison of the genres). This does not rule out that new concepts can emerge in other genres that might work as well.

a) What game attributes would be best suited for this scenario?

- ➢ The important attributes for games in this scenario is that players can easily distinguish themselves from others, and that the game supports almost instant access into the game. The game must also be fast paced enough for players to not get bored. Besides this, the limit is the developers imagination and creativity.

b) How many players will such a game be able to host without becoming chaotic?

  ➢ During our testing, we found that 50 players was the limit for the screen resolution we used.

c) What visualization types are best suited to host a great number of players?

  ➢ Looking at both 2D and 3D visualization type, we found the 2D type to give the most usable game area, but both are viable options.

RQ-II: **Is it possible to create a back-end framework to ease the development of games in this scenario?**

➢ Yes, by designing a framework that handles back-end tasks like user administration and network code which any game can easily access through a simple interface.

a) What role(s) would such a framework fulfill?

  ➢ The framework would perform roles like:
    - User administration
    - Billing of services
    - Game controller input and feedback
    - High score lists
    - User community functionality

b) How would the framework ease the development of games for the scenario?

  ➢ By suppling a simple interface to the game developers it would rid them of designing and implementing code for all the roles the framework supports.

c) Would the framework ease the implementation of the scenario?

  ➢ The framework simplifies the development of games as mentioned above. The framework is also easily distributed and set up using one screen and a server connected to a Bluetooth hub. Starting the framework for both users and maintainers is easily done. Server side is started with simple commands, while the users only need to download the client application and run it. Registration by the users is however required.

18.1 Final notes

The framework we have made has no limitations on what kind of screen it will be displayed on. Therefore we have not imposed any restrictions of the setting where the user can play. We have tested the framework and the test game at a cinema, but it is the same interface ie for cafes, pubs or other public areas. The only limitation is that the display should be big enough to be viewable and have a good enough resolution to be playable, a standard hdtv should be good enough for this use.

CHAPTER 19

Further Work

We have only skimmed the surface of what is possible with further development. During our brainstorming for the framework we came up with many nice-to-haves and a few must-haves. Future modifications to the framework should try and include the following suggestions:

Small scale:

- ➢ 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 scale:

- ➢ 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.

Part VIII

Appendix

CHAPTER 20

Source code

20.1 Framework source code

The framework source code is found on the CD supplied with the project.

20.2 Test game source code and binary

The test game source code is found on the CD supplied with the project. The binary is located under binary/moose. A short description is included in the Readme.txt file.

CHAPTER 21

Source code documentation

21.1 Framework javadoc

The framework javadoc is found on the CD supplied with the project. It is divided into three parts:

- Javadoc for common classes
- Javadoc for the J2ME functionality
- Javadoc for the J2SE functionality

21.2 Test game source documentation

The test game source documentation is found on the CD supplied with the project.

CHAPTER 22

Test video

A video filmed by our supervisor Alf Inge at the final test at Nova kino is found on the DVD supplied with the project.

Part IX

Bibliography

Bibliography

[1] Eclipse contributors et al. Openup/basic. http://www.eclipse.org/epf/index.php, 2006.

[2] Bjarne Kjøsnes. Midgard media lab - av arena norway. http://www.ntnu.no/midgard/av-arena.html.

[3] 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.

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

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

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

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

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

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

[10] Jason Brownlee. Finite state machines (fsm). http://ai-depot.com/FiniteStateMachines/.

[11] MTV News. Game-design contest hatches out-of-control moths, one-armed whirlwinds. http://www.mtv.com/news/articles/1526118/20060314/index.jhtml?headlines=true, 2006.

[12] Blinkenlights. Building arcade games. http://www.blinkenlights.de/arcade/games.en.html.

[13] Maria N. Stukoff. Big screen games. http://mobilebox.typepad.com/game_design/2006/04/big_screen_game.html, 2006.

[14] Maria N. Stukoff. Proximity games. http://mobilebox.typepad.com/game_design/2006/08/proximity_games.html, 2006.

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

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

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

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

[19] Gianni Pasolini. Analytical investigation on the coexistence of bluetooth piconets. http://ieeexplore.ieee.org/iel5/4234/28579/01278302.pdf.

[20] Paul Clements & Rick Kazman Len Bass. Software Architecture in Practise. Addison-Wesley Professional, 2 edition, 3 2003.

[21] Alan Kaminsky. Infrastructure for distributed applications in ad hoc networks of small mobile wireless devices. Technical report, Rochester Institute of Technology, 2001.

[22] David N. Welton and Wolfgang Kechel. Hecl. http://www.hecl.org/.

[23] Simon Whiteside. Simkin. http://www.simkin.co.uk/.

[24] Esperanza Marcos María Lázaro. Research in software engineering: Paradigms and methods. Technical report, Rey Juan Carlos University, Kybele Research Group, Madrid, Spain, 2005.