NTNU

Norwegian University of Science and Technology

Mobile and Social Video Games

Prototype, Concepts, and Evaluation

Øivind Nøsterud

Master of Science in Computer Science

Submission date:	June 2008
Supervisor:	Alf Inge Wang, IDI

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

Problem Description

The goal of this project is to explore new game consepts for mobile and social games implemented in J2ME. The games can focus on either mobile multiplayer real-time games or mobile multiplayer asynchronous games where several players interacts using the mobile network. The game can also utilize the location of the player in the gameplay and all the functionality of the mobile phone (screen, loudspeaker, vibration, camera, microphone, etc.).

Assignment given: 15. January 2008
Supervisor: Alf Inge Wang, IDI

Abstract

The main goal of this project was to create new game concepts for mobile and social games. The project was a continuation of the depth study performed by the project in the course TDT 4570, Game Technology, at NTNU. The focus of this project was slightly more shifted toward the social side of mobile multiplayer games compared to the depth study. Additionally the project group were to create a prototype game using one of the concepts implemented in Java ME.

The project group performed a prestudy of the technologies required to create the prototype(such as Java ME and features specific to mobile phones, such as location), as well as looking at games from a general viewpoint, but also by looking more closely at mobile and social games, current game genres, and multiplayer games available on mobile platforms and looking at how these games implement social game mechanisms.

Several concepts were conceived and described during the project, with concepts from both the depth study and concepts created specifically in this project. The *Platform-puzzler* concept was chosen to be implemented as the prototype game. The prototype game is a 2D side scrolling platform multiplayer platform game with puzzle elements. The game was implemented in a development process, and at the end of the process the game was tested by two testers not from the project group. The test allowed the project group to assess how it performed in a simulated setting using emulators, as well as testing to see how the cooperative elements of the game performed, and additionally the test was performed in two sessions were in the first session the two testers were located in the same room, whereas in the second test the were located in the same room.

The test session performed on the prototype indicated to the project group that there for the prototype game there were quite clear differences between playing the game in the same versus playing the game in different rooms. This indicates further that special care must be taken when designing and creating cooperative games for mobile platforms. The additional research performed by the project group also indicate to the project group that creating mobile and social games for mobile platforms is highly viable, and that social gameplay mechanisms can strengthen the experience of playing games.

Preface

This project was performed as a master thesis in TDT 4900 Computer and Information Science, which is the conclusion of the Master of Science degree in Computer Science at the Norwegian University of Science and Technology(NTNU). The project group consisted of Øivind Nøsterud, and the work was carried out from January to June 2008. The project description was outlined by the project participants and the supervisor, Alf Inge Wang of the Department of Computer And Information Science(IDI) at NTNU.

We would like to thank the supervisor Alf Inge Wang for help and advice during the project. In addition the supervisor helped in giving feedback both to ideas and on the report, as well as giving help with the testing performed on the prototype created during this project.

Trondheim, June 10, 2008

Øivind Nøsterud

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

This chapter gives the motivation and definition of the project, and it also describes the research context the project is a part of, as well as a reader's guide for the rapport.

1.1 Motivation

This project is a continuation of a depth study performed by the same project group in the course TDT 4570, Game Technology, at NTNU[Nøsterud, 2007]. In the previous project the group looked at mobile and social real time multiplayer games, and created several concepts for mobile and social games. The concepts utilized the unique abilities of mobile devices, such as location, and other pervasive elements, as well as more mobile phone specific technologies such a SMS, and rumble functionality among other. To further experiment with mobile and social games a prototype game was built by using one of the concepts conceived during the project. The concept chosen was the *MMO Framework*, and the final prototype game was a multiplayer roleplaying(RPG) game that had three distinct separate parts, namely a single player part, a multiplayer part, and a simulation part. Among the conclusions drawn from the project where that games and genres found on computer and console video games could be used for games on mobile devices, at least in some form, and also that current games do have social elements in them, but that there is still room for improvements. The results from the project, along with the fact that the prototype did not perform as well as could be expected lead the project group to believe that there were still unanswered questions in the assignment.

Mobile devices are becoming increasingly popular, especially in recent years, and the devices are becoming increasingly technological advanced and useful for the users of the devices. Most families in Norway currently have at least one mobile phone per household, with some households containing upwards to one mobile phone per family member. Giving users the ability to be mobile, and supporting this mobility in the applications and in the devices themselves is becoming increasingly important as mobile devices gain popularity. The growth of the mobile user market, and the large user base either owning, or planning on obtaining a mobile device gives a large potential for new users of new exciting and fun mobile applications, there among mobile and social games.

Games have been around for a very long time, and humans have always looked for ways to entertain themselves in the form of activities and games. The growth and popularity of computers gave new possibilities for interactive and digital activities in the form of computer and video games, as well as hand-held games. Since the mid eighties video and computer games have grown steadily in popularity, and the game market has grown so much in size that i currently competes with the film industry in revenue. The growth of popularity of mobile devices, and the increased functionality and computing power offered in these devices also lead to games becoming an important part of the mobile devices. Early mobile phone games such as *Snake* came pre-installed with the mobile phones, and an increase in games available to the users, as well as bundled games on mobile devices have lead to an increase in the popularity of games on mobile devices.

Mobile devices give the players the addition of mobility to the gaming experience, and the players can enjoy fun and engaging games while waiting for the buss, on the train to work, or simply as a relaxing activity in their homes after a stressful day. The increase in popularity of mobile devices, perhaps especially mobile phones, smart phone and PDA's, give a large potential market for mobile games, and tailoring the games to the mobile community while still maintaining fun, engaging and exciting gameplay will be important to take advantage of the market possibilities.

The social aspect of gaming is equally important to the technological prowess of the platforms running the games. Long gone are the days when games where only played by teenagers in their parents basement. In the game market today games are created and marketed toward different groups of people, and the content of the games are as varied as the people that play the games. Playing games with other people both locally and over Internet have become an important part of modern games, and multiplayer have been part of computer and video games since the early days of playing *Pong* in the family living room with friends and family. The social interactions that spur between the players can in some games be equally important to the gameplay itself. Games like *Guitar Hero* and *Buzz!* are highly social experiences when played together with friends. Virtual environments such as *Second Life*, and MMORPG's like *World of Warcraft* or *Age of Conan* have a large focus on social interaction between its users, and indeed the social interaction supported in these games are a crucial part of their success. The social aspect of games is not restricted to simply allowing the players to move around in the same virtual world either, and social networks and hierarchies in games can give rise to exciting new possibilities within the games. Social gaming could give rise to new exciting game concepts and game types, and the mobile platform could be ideal to cater for such an environment.

1.2 Project Context

This project was conducted as a part of the MOSS project, or Mobile and Social Games project, which is a part of the Research Program in Computer Games at NTNU.

1.3 Problem definition

The goal of this project is to explore new game concepts for mobile and social games where the games should either be real-time or asynchronous mobile multiplayer games where several players interact using the mobile network. The use of mobile devices gives the games the option to take full advantage of the extra features found in mobile devices, such as vibration, camera, microphone, and location. These features could help increase the user experience in the games, and certain features could give rise to new game concepts. An important part of the concepts should the social part of the concepts, and how the social interaction between the players could help the concepts.

The second goal of the project is use any of the new concepts created in this project to create a game prototype implemented in Java ME. This prototype could take advantage of any of the features discussed above, and it could help answering the research questions in the project.

1.4 Reader's guide

Part I - Introduction The introduction explains the motivation and context of the project, as well as defining the research questions for the project, and details the development tools and methods used in the project.

Part II - Prestudy The prestudy gives background information about the various technologies used in the project, as well as give more information about the main problems of the project, namely about gaming in general, and more specifically mobile and social gaming. Java ME is also explained as it is the development environment to be used in the project.

Part III - Own contribution In this part, the contribution made by the project team is described and explained. The part looks at concepts created by the project team, both the concepts created in the depth study leading up to this project[Nøsterud, 2007], as well as new concepts for this project. The new concepts are then evaluated and one of the concepts are chosen to be implemented as a prototype game. The part then further looks at the requirements, design and implementation, as well as the testing of the prototype.

Part IV - Evaluation The evaluation looks at the prototype game, and the technologies used, and evaluates them, as well as taking a look at the problems encountered in the project, and looking at and trying to explain unresolved problems found during the development and use of the game and technologies. The results from the prototype testing are also evaluated in this part.

Part V - Summary The summary gives a conclusion to the research questions, and concludes the findings of the project. The summary also looks at further work that may be done to improve both the concepts and the prototype game.

Chapter 2

Research questions and methods

This chapter identifies and defines the research questions of the project. The chapter also describes the research methods that will be used to find answers for the research questions.

2.1 Research questions

The research questions identified and defined in this section are the guideline for the investigation pursued by the project team in this project. The questions try to identify the possibilities and challenges of mobile and social games.

1. **How does player co-location(i.e. an environment where two or more people are located in the same room) affect the experience of playing games?**
   (a) Are some game concepts more suited for co-located environments than others?
   (b) Can some game concepts affect the experience of playing the game differently depending on whether the players are co-located or not?
2. **How do gameplay mechanics affect social gaming?**
   (a) Can social gameplay mechanics in games affect the experience of the player?
   (b) What game mechanics can be used to make mobile games social?
3. **What game genres are most suited for mobile collaborative games?**

The first question along with its sub questions tries to look at whether the location of the player has an impact on the experience the player has of the game. Can being located in different rooms when playing a multiplayer game lessen the experience of playing certain games, and will then playing the game in the same room have the opposite effect of encreasing the experience of the player? This could help in deciding how to develop cooperative games, and what considerations should be made when making such games.

The second questions and its sub questions tries to look at how the specific gameplay mechanics affect games. Can social game emchanics increase the experience of the player, and if so; what game mechanics are used today, and which game mechanics could be possible to use in the future. The questions therefore takes a look at how social gaming is incorporated into games, and how best to incorporate social game mechanics into games.

The third and final questions takes a look at game genres, and how they are suited with mobile collaborative games. Could it be that there are certain game genres that are more suited because of the genre itself, or are there no differences in how game genres are suited in collaborative games? This could help developers decide which game genre they should develop mobile and social games in.

2.2 Research methods

This section describes the methods used to understand the context of the goal of the project, as well as give answers the research questions of the project.

Basili[Basili, 1992] identifies three main research approaches commonly used in experiments belonging to the software development domain:

1. **The engineering method:** When using the engineering method, the researchers create a test system which is meant to prove or disprove a hypothesis. The results of the test are then used by the researchers to improve the test system iteratively until there are no further measurable improvements applicable to the system.
2. **The empirical method:** The empirical method is a statistical method used to validate a hypothesis, and data is gathered to verify or falsify the hypothesis.
3. **The mathematical method:** The mathematical method is a formal method based on mathematics and formal methods for doing experiments. Empirical observations are compared to a formal theory created by the researchers.

In addition to the research methods described above, Wang[Wang, 2006] describes several other research methods, where one method is useful in this project. In a **literature search** the results of papers and other documents relevant to the subject matter are analyzed, and they can be used to either confirm an hypothesis, or to improve previously collected data in a project. This method will be used to give additional insight on certain aspects not easily found through any of the three methods above, by looking at previous work in the area to find information to answer the questions raised in this chapter.

The main focus of this project is to look at mobile and social games, and find new concepts for these games, as well as creating a prototype that implement one or more of these concepts in order to answer the questions raised in this chapter. The engineering method is well fit for such a task, especially since the prototype development will be an iterative process, and that the prototype will be used to answer research questions.

The empirical method is also applicable to the scope of this project since it can be used to watch players test the prototype game, and thereby observe how they react to the prototype, how they play the game, and how the players work together to solve the obstacles found in the prototype. To gather further data questionnaires can be given to the testers, and they can give further indication about how the prototype works, how it is to play the prototype, as well as indications to answers for some of the research questions defined in this chapter.

Chapter 3

Development tools and methods

This chapter looks at development tools and methods used in this project to create the prototype game, and to answer the research questions raised in the previous chapter.

3.1 Development methods

There are many development process models that may be used to create software, or in this project; to create a prototype game for one or more of the concepts identified in this report. Development process models deals with the distribution of usual development tasks, such as testing and design, and how the transition between each task functions.

3.1.1 The waterfall model

Figure 3.1 shows a diagram describing the phases and transitions of the waterfall model, as depicted in [Braude, 2001]. The waterfall model is one of the most straightforward and basic models. As seen on the diagram, the model has five phases, and the transition between the models is from the phase the development team is currently in, to the underlying phase. Normally though, most development teams will have to overlap the various phases, as certain aspects of the development may be difficult to perform otherwise.

The five phases of the waterfall model are as shown in the diagram; the requirement analysis, the design, the implementation, the integration, and finally the test phase. The requirements analysis phase consists of gathering requirements for the system that is to be developed, and the design phase deals with transforming requirements and wanted behavior of the system into a structured manner which can be used to develop the system. This is usually done both graphically by using diagrams such as UML class diagrams, or activity diagrams, and textually which further describe the details and functionality the system should incorporate. The implementation phase involves coding and programming the system, at all levels, and the integration phase handles the integration of all parts of the system into a working system and a complete product. The test phase tests the final

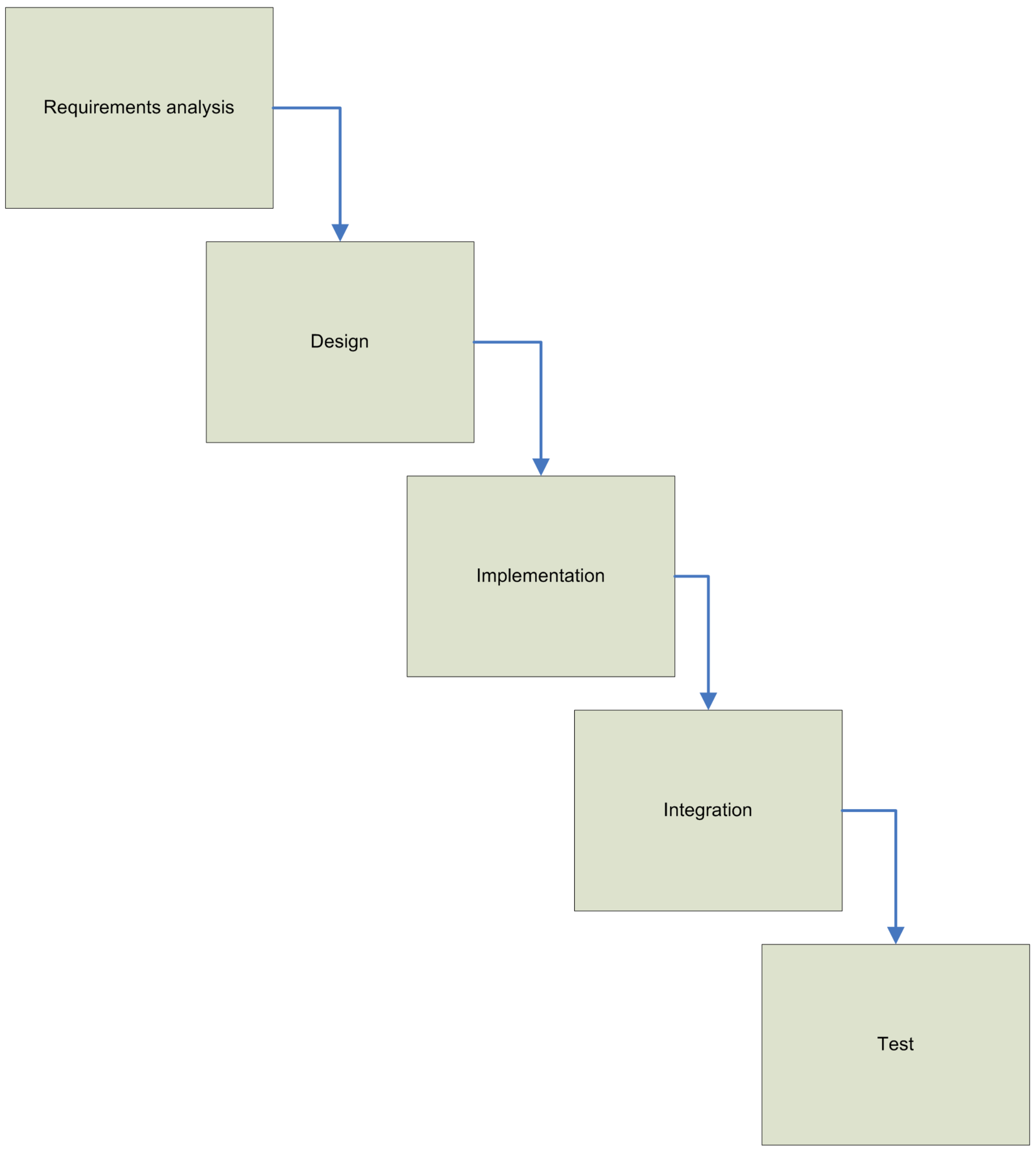


Figure 3.1: The waterfall process model

product and attempts to find flaws in the design, the program code, and any other issue that may reduce the functionality of the final product.

3.1.2 Modified waterfall model

The basic waterfall model is not a perfect process model, and often receives criticism for being an unrealistic development model, since in its most basic form, it does not allow for iteration of development phases, which could be very helpful if say for instance a customer halfway through the implementation phase finds new requirements that must be included in the final system. However, the waterfall model is a simple model, and a modification of the model as seen in Figure 3.2 allows for some backtracking between phases. The model is much like the basic waterfall model, except that it allows for developers to back to previous phases when the need arises, such as when tests force the developers to redesign and re-implement a part of the system. For this project such as modified model of the waterfall model suits the project team well, since the scope of the prototype game will be relatively low, making the need for more complex iterative process models low. If however the project team finds that they must go back and redesign certain aspects of the prototype game, the modified model still allows this.

3.2 Development tools

The development tools described in this section have been used in this project, some for creating the prototype, but also some that help in creating this rapport.

MiKTeX 2.6 MiKtex is an up-to-date implementation of $\TeX$, which is a typesetting system that can be used to write reports, technical papers, mathematical formulas and more. The MiKTeX distribution has several tools and compilers allowing the users to write reports in $\LaTeX$, and compile the reports into PDF documents.

TeXnicCenter 7.01 TeXnicCenter is a IDE(integrated development environment) for developing $\LaTeX$ documents, and the program gives the user an editor that helps in writing and creating documents using $\LaTeX$.

Java Wireless Toolkit 2.5 The Java Wireless Toolkit is a toolbox for developing wireless applications that are based on Java ME. The toolkit includes such features as emulation environment, performance optimization and examples of Java ME programs to help the programmer get started.

Java Development Kit SE 5.0 Java Development Kit SE 5.0 is a toolkit that allows developers to create and build Java applications in Java 1.5 SE(standard edition), and give the developer such features as compilers.

Eclipse 3.2.1 Eclipse is a IDE that help developers develop programs in Java. The program offer features aimed at helping the developer, such as debugging and construction tools. There are also modules available for Eclipse that helps developer creating Java ME applications.

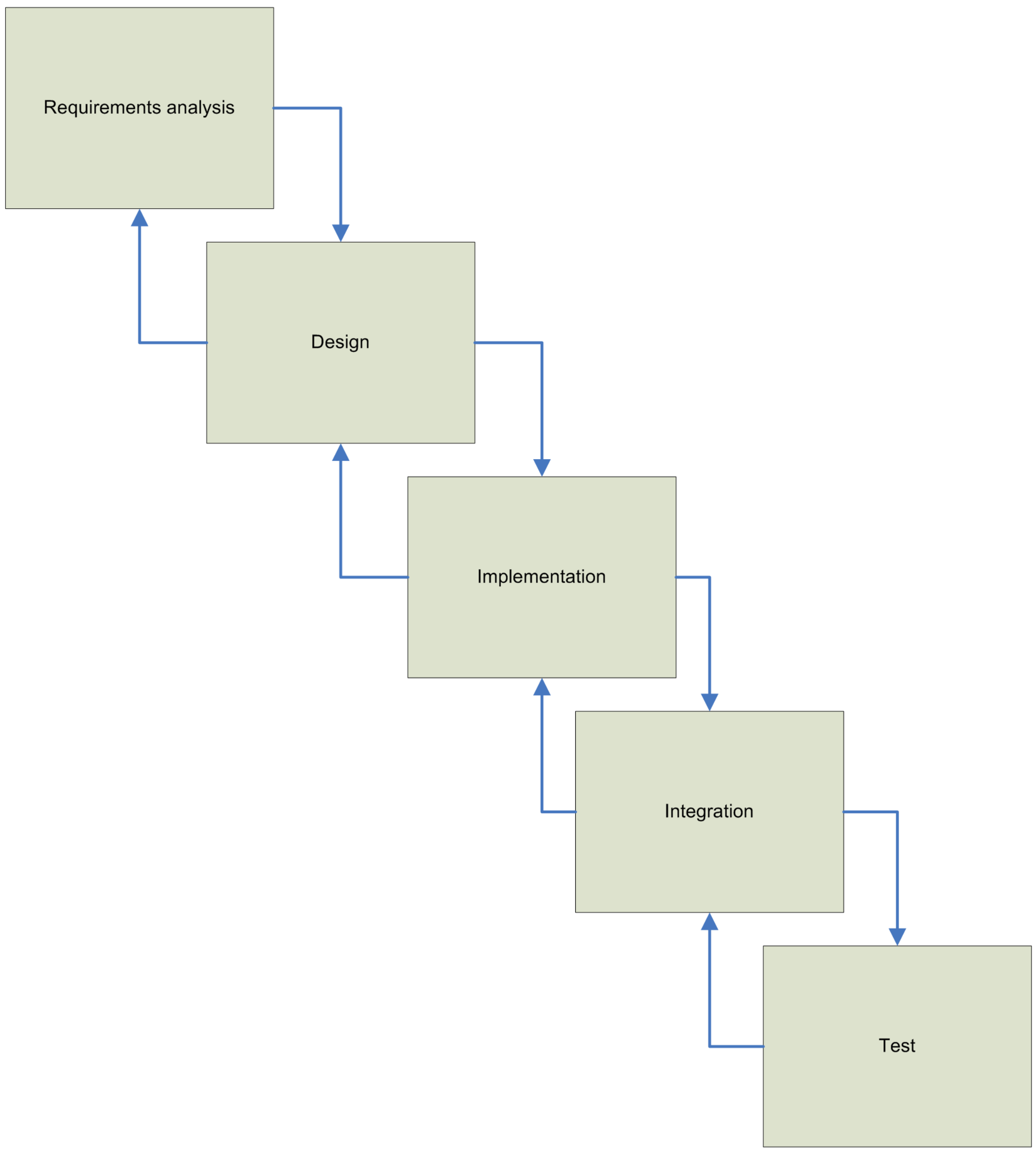


Figure 3.2: Modified waterfall model

Sony Ericsson SDK 2.5.0.2 The Sony Ericsson SDK 2.5.0.2 is similar to the Java Wireless Toolkit in that it is a toolkit to help in development of Java ME programs and that it includes such features as emulators. This toolkit is specialized in Sony Ericsson mobile phones, and gives a more proper environment to for example test programs that are meant to be used on Sony Ericsson phones.

Altova UModel 2008 Professional Edition Altova UModel 2008 is a visual design software that can either be used to visually design a system which can then be turned into code, or the tool can be used to reverse engineer already existent code to create diagrams from source code.

Microsoft Office Visio 2003 Visio is a part of the Microsoft Office package, and is a visual design program tool that allows users to create diagrams using diagram standards such as UML.

Part II

Prestudy

Chapter 4

Technology

This chapter looks at some of the technologies within the scope of the project. More specifically the chapter looks at the Java Micro Edition Platform, with its features and limitations. The chapter also describe functionality offered by modern day mobile phones, and how these functionalities can be used in mobile games. Finally this chapter looks at some of the mobile network technologies offered to consumers. This chapter is based off of the project members depth study[Nøsterud, 2007] in the same subject, however it has been altered to better reflect the new focus of this project.

4.1 Java Platform, Micro Edition

Sun Microsystems describe the Java Platform, Micro Edition(Java ME) as follows[Java ME]:

> Java Platform, Micro Edition (Java ME) is a collection of technologies and specifications to create a platform that fits the requirements for mobile devices such as consumer products, embedded devices, and advanced mobile devices. It is a collection of technologies and specifications that can be combined to create a complete Java runtime environment specifically to fit the requirements of a particular device.

I.e. Java ME is a collection of technologies and specifications created partly by Sun Microsystems itself with the additional help of expert groups in charge of standardizing the platform. The expert group consist of members from various industrial companies with many of the leading companies in mobile device development involved [Riggs, 2003].

4.1.1 Java ME Architecture

Figure 4.1 shows the different components of Java ME, and how they relate to the other Java Platforms. Java offers solutions ranging from servers and enterprise computers, which demand high resources, through Java EE, to smart cards, which require low resources, with Java Card.

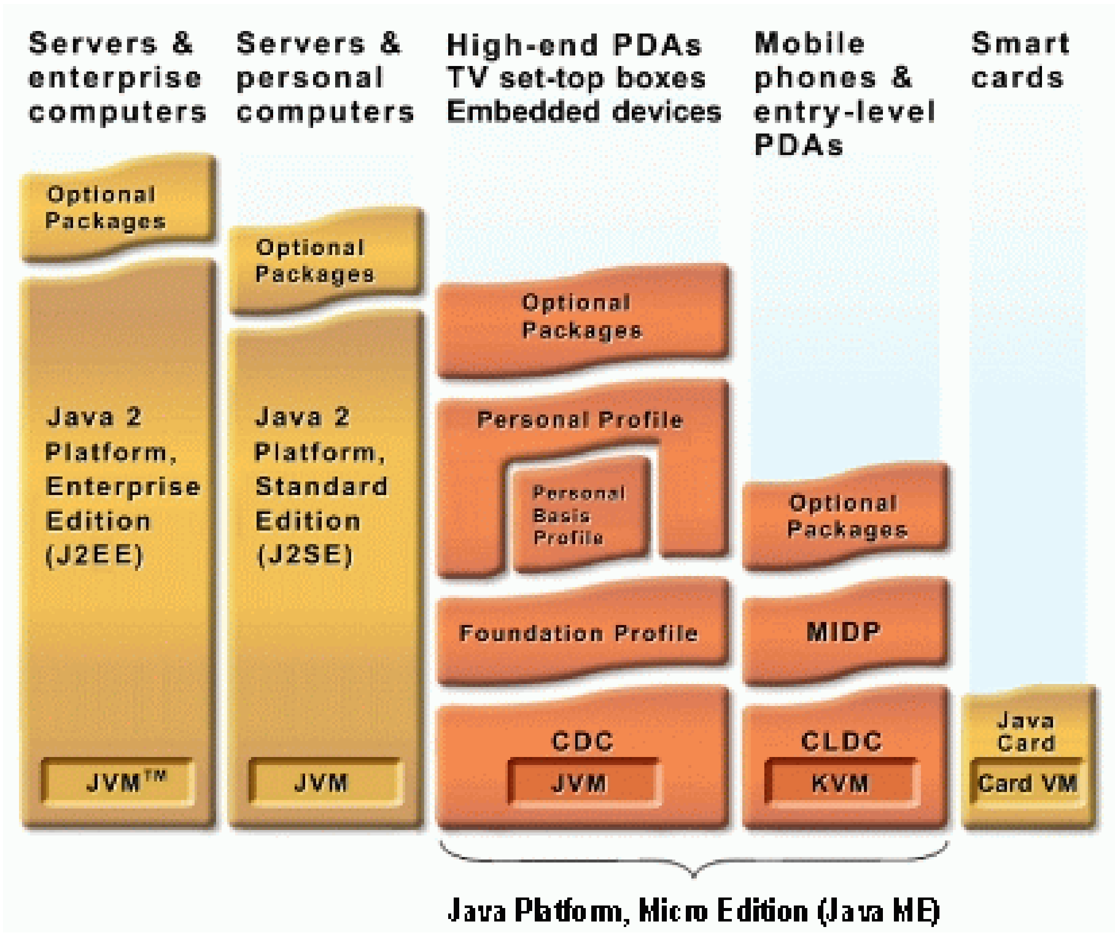

Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
Optional Packages
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition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

Figure 4.1: Java ME components

Java ME itself is divided into two parts with one intended for the high-end consumer devices such as smart-phones and set top boxes. The other part is intended for low-end consumer devices with limited memory, processing powers and graphical capabilities such as cell phones, pagers and personal organizers[Riggs, 2003].

Since this project is aimed more toward mobile phones, or at least more mobile devices commonly used by the public, the rest of the section will focus on the low-end part of the Java ME platform. Common though for both of the two parts are that they are divided into three parts; the configuration with the virtual machine, the profiles and the optional packages. These will be described in the following sub sections.

Configurations

A Java ME configuration defines a minimum platform for a broad range of devices, defining the Java language and the virtual machine features and minimum class libraries that a device manufacturer must provide in order to comply with the standard.

There are currently two configurations for the Java ME platform; Connected Device Configuration(CDC) targeting the high-end devices, and Connected Limited Device Configuration(CLDC) focusing on the low-end devices.

CLDC uses the K Virtual Machine(KVM)[J2ME Building Blocks]. It is a compact, portable Java virtual machine intended for small resource-constrained devices. The K stands for kilobyte, since the KVM was designed to run in an environment with a small amount of memory, down to just hundreds of kilobytes. A Java Virtual Machine(JVM) is an abstract computing machine[JVM]. Its main responsibility is executing Java code. When Java code is compiled(e.g. using javac), it's translated to bytecode and stored in a .class file. The JVM reads these class files and execute the bytecode using the instruction set of the operating system the JVM resides on. A KVM can be seen as a stripped down JVM.

The most current version of CLDC is CLDC 1.1 (JSR 139), which is an improved incremental version of CLDC 1.0 (JSR 30). Version 1.1 of CLDC adds features such as floating numbers and weak reference support.

Profiles

A profile is layered on top of a configuration, there by extending the configuration with more functionality. An important goal of a profile is that various device families or domains have a standard Java platform such that various devices in the device families can work with the same Java applications. The profile defined for CLDC is the Mobile Information Device Profile(MIDP). The MIDP adds a application model to the CLDC as well as giving user interface support, network support, persistent storage, control over sound, network security and various other classes. MIDP 2.0 also adds support for 2D games with classes for handling sprites, tiles, backgrounds, layers and layer management which simplifies game development [Williams, 2004]. The newest version of MIDP is MIDP 2.0 (JSR 118), which is an improved version of MIDP 1.0 (JSR 37) where MIDP 2.0 is backward compatible with MIDP 1.0.

An application created in MIDP is called a MIDlet. A MIDlet is a class that extends the class **javax.microedition.midlet.MIDlet** and implements the required methods startApp, pauseApp and destroyApp. The startApp method is called when the MIDlet is initiated, thus maintaining the role of the main() method in Java SE and EE. MIDlets are deployed in so called MIDlet suites, which consisted of one or more MIDlets packaged in a JAR(Java Archive). The MIDlets within the suite share the same namespace, run-time object heap, and static fields in classes. Each JAR should also be accompanied by an application descriptor file(JAD file) to enable third party distribution of the MIDlets.

Optional packages

Java ME also offers additional packages apart from CLDC and MIDP. These are layered on top pf the profiles and are generally packages that are applicable to a large number of devices offering functionality not found in the standard minimum Java ME platform(i.e. CLDC and MIDP). Some of the optional packages are:

- JSR 120: Wireless Messaging API
- JSR 135: Mobile Media API
- JSR 172: Java ME Web Services Specification
- JSR 179: Location API for Java ME
- JSR 184: Mobile 3D graphics for Java ME
- JSR 190: Event Tracking API for Java ME

4.1.2 Limitations

Java ME is generally built for devices with low memory and limited computational power. This gives limitations in how much functionality Java ME can offer. Java ME can thus not be used to develop highly advanced enterprise applications for mobile phones; the limitations of Java ME should thus be taken into consideration when MIDlets are being created.

Another issue is that the mobile devices support different packages and sometimes even different versions of e.g. MIDP. The developers must then consider various tradeoffs when implementing the applications since not all of the mobile devices will support all of the functionality offered in all of the available packages. This is especially true with the optional packages. The mobile device manufacturers decide themselves how many, if any, of the additional packages they should include in their devices, often on the basis of the equipment supported by the mobile device. The developers must therefore consider the mobile device they are developing their application for, and whether they want to sacrifice functionality over interoperability.

4.2 Mobile game devices

There are several mobile devices today that offer games to their users. Some of these devices are custom made for games, such as Nintendo DS or the Playstation Portable, but there are also many games for the more common mobile phones, often created in Java ME. Here common relates to the fact that mobile phones have become very common in recent years. Informal Telecoms & Media. According estimated in may 2005 that the number of mobile subscribers were 1,8 billion, and that the number would to rise to over 2,14 billion subscribers by the end of 2005. They also stated that many European countries were nearing 100% penetration [MobileTracker, 2005]. This project will therefore focus on mobile phones even though other mobile devices are able to use Java ME, e.g. PDA's. Hand-held gaming consoles are related to mobile gaming; however they are out of the scope of this project, especially since this project focuses on Java ME as its development platform.

4.2.1 Mobile phones

This section looks at some of the functionality offered by mobile phones, and more specifically the functions that can be used in mobile gaming, since the focus of this project lies on mobile gaming. All of the functionality described in this section is not offered by all mobile phones though, since mobile phones differ in the functionality they offer their users. Some functionality is now considered standard in mobile phones, while other is only offered by some phones. This must be taken into consideration when creating games for mobile phones.

Camera

Many, if not most, of the modern mobile phones offer camera functionality in their phones where the user can take photographies and save them to their phone. Some games use the camera actively in the game, such as Before Crisis: Final Fantasy VII [BeforeCrisis.net], where as other use the camera more indirectly, e.g. by allowing the users to take a photograph and use it as a background in the game. Though strictly not a game, CybStickers [CybStickers] use the camera of a mobile phone, as well as MMS to take pictures of stickers which contain information as a optical code. The information on each sticker can be gathered by sending the image of the code to a server, or information can be added to the sticker.

Even though cameras have become common these days, there may be difficulties with implementing it in a game since camera functionality is implemented differently between different mobiles, especially between manufacturers, and it may be difficult to create an application that easily use the camera functionality while still retaining modifiability. For a real-time multiplayer game the size of images in bytes can also be an issue because of the speed of the connection between mobiles.

Keypad

The keypad is the main source of user input for the mobile phone. Other parts of the mobile are also suited, e.g. microphone, camera, but the main source for user input in most games comes from the keypad, and most games uses the keypad to control the game in some way. Most mobile phone keypads have less buttons, or keys, than a computer keyboard giving less options of key usage, and the position of the buttons on the keypad are usually somewhat more cumbersome than on traditional gamepads used by consoles, some times making the games difficult or awkward to control. On top of that some buttons are standard for some manufacturers and phone models, while not on other. This could be buttons such as a small directional joystick which some newer phones have, or additional buttons that give menu shortcuts. These shortcut buttons are usually not the same from mobile phone to mobile phone.

All of this implies that extra care must be taken when deciding how the configuration of the buttons is to be done in game, and how many buttons the game should rely on for game control. Most mobile phones have a keypad with at least 12 buttons, i.e. the 1-9 keys with 3 additional keys for '0', '*' and '#', even though some phones have less keys, or some phones don't have any of these regular keys at all, e.g. iPhone [iPhone]. Nevertheless most mobiles have at least 12 keys on the keypad, so that should be a safe starting number when considering how many keys the game could have.

Vibration

Most mobile phones today have vibration. This is usually used for silent mode where instead of a ring tone you get vibration, or in conjunction with the ring tone to make the user aware of incoming calls. Vibration is also well suited for games where the vibration can help increase the game immersion e.g. by vibrating after a explosion, or using vibration to highlight a certain event in the game. Consoles have used vibration in their games via vibration support in their controllers for several years in consoles such as Nintendo 64, Sony Playstation 2 and Microsoft's XBox. The drawback of using vibration is that is uses the battery of the mobile phone. Thus the usage of vibration should be considered used where appropriate, but if battery time becomes an issue it can be left out or have its duration shortened in the game to increase battery time.

Screen

Most mobile phones have a screen and they are increasing in size, resolution and color range. Color displays are becoming de facto standard in modern mobile displays. Even so there are differences in size and resolution between the different phones. This is a crucial factor when designing games, since the size of the screen have a strong impact on the complexity of the game, especially on how many items can be shown on screen at once, and how complex the graphics and environments can be. Most of the displays on mobile phones are relatively small, at least compared to console and PC games. Most phone displays are also smaller than hand held consoles. The problem thus becomes to decide which resolution the game should use, and how detailed the graphics in the game should be while keeping the game clutter free, but still enjoyable enough that people are able to see what is going on in the game without to much trouble.

Touchscreen

A few mobile phones have touchscreen such as iPhone [iPhone] or the Sony Ericsson P990i(technically a smart phone) [P990i]. Touchscreen opens up exiting new possibilities for game control, as illustrated by the Nintendo DS [Nintendo DS]. The touchscreen could be used for steering a character on screen, to navigate menus and press buttons, which could help with the limited amount of keys available on the keypad, or to control any other objects on screen using either the users finger or a stylus.

There are however very few mobile phones available with touchscreen, so creating a game with only touchscreen control highly limits the phones, and users, that can play the game. Thus this feature is best added as an optional game control mechanism, if need be.

Movement sensors

A few mobile phones support movement sensors, such as the iPhone [iPhone]. These sensors may detect movement of the phone or the orientation of the device. This opens up new possibilities in the game design where the movement of the phone can be incorporated in the game, e.g. by moving a ball by tilting your phone. Being able to detect movement is one of the selling points for the Nintendo Wii [Nintendo Wii]. However there are few mobiles that incorporate this functionality, so creating a game for mobile phones using movement sensors severely limits the amount of mobiles and players that are able to play the game.

Speech

Every mobile phone have a microphone since the main functionality of a mobile phone is to be able to call and talk to other people. Using speech in gaming opens up interesting possibilities, such as players being able to speak to each other during a play session. Communication between players is often used in First Person Shooters such as Counter-Strike and the Battlefield series, both as a means of communication, but also more directly in the game by for instance having team leaders direct the other players, or discuss strategies and enemy positions. Speech could also be used more directly in the game by using speech to control the game with speech recognition. Speech recognition might however be very difficult to create on a mobile platform, not only because of generally low computational power on mobile phones, but also since speech recognition in itself is complicated. As for using speech as voice chat in the game, it adds more data that must be sent over the network, thereby increasing the cost of running the game, as well as straining an already limited bandwidth.

Sound

Most new mobile phones have support for either compressed(e.g. MP3) or uncompressed(e.g. WAVE) sampled audio, or at least MIDI. Sound support is usually incorporated into every game in some way or other as it increases the gameplay and if music is used; atmosphere. Sound is usually used to indicate events in the game, such as a gaining an item, or indicating a player action, such as a player jumping, or firing a gun.

SMS / MMS

Most, if not all, mobile phones offer SMS(Short Message Service), which offer the phone subscribers a way to send text messages to other subscribers, and most modern mobile phones also come with MMS(Multimedia Messaging Service), which offers a standard way of sending multimedia objects, i.e. images, audio, video and rich text, between subscribers. Even though sending multimedia objects and text messages in a game could be done through for instance a server client architecture by using TCP/IP or UDP, but SMS and MMS are standardized ways of doing the same, and most users are familiar with this technology, making it easier for the players to use. SMS could for instance be used by a server to send information about game state changes to the player, and MMS could be used to send multimedia objects from the server to the client applications.

Limitations

Mobile phones have several functions and possibilities in offering a platform to create games, but mobile phones also have limitations compared to PC's and gaming consoles. As showed above the mobile phones have relatively small screens, and small resolutions, which must be taken into consideration when creating mobile applications. The keypad of mobile phones may also have complications compared to PC's and gaming consoles, as they can be awkward for the user if not utilized correctly, and some types of applications and games may have a lack of input options because mobile phone keypads usually have a limited amount of keys in the keypad. Another problem can be that of computational power, as generally mobile phones are relatively low on computational power, when compared to PC's and gaming consoles, even though smart phones are becoming quite powerful. Battery consumption is also a problem with mobile phones, as running applications on mobile phones drains the battery life faster, and using advanced functionality in games and applications worsen the problem further. This is a problem which should

Manufactor	**Phone model**	**Java features**	**Screen Size**
Sony Ericsson	W810i	MIDP 2.0, CLDC 1.1	176x220

Table 4.1: A table showing the test phones used in this project

be taken into consideration when developing applications. Another problem with mobile phones is that the various phone models from the various manufacturers have different functionality offered in their phones, and even among phones from the same manufacturer there are differences in the offered functionality. By developing applications and games utilizing a relatively uncommon feature, the market for those applications is instantly lowered.

Requirements mobile phones in this project

These will be the minimum requirements to run the prototype games created in this project and the game will not run on mobile devices that do not support at least the requirements described below.

- An Java ME enabled mobile device
- JSR 118: MIDP 2.0
- JSR 139: CLDC 1.1
- At least 12 dials(since this is standard, see Section 4.2.1
- At least 94x54 screen resolution(defined as minimum for the MIDP standard [Riggs, 2003])

Test phones used in this project

This section lists the available test phones in this project. Even though there are several emulators available from various actors, e.g. from Sony Ericsson and Sun Microsystems, true performance of the game can only be tested in a real-life environment, i.e. on the mobile phones it is supposed to be played on. The prototype games created in this project should work on most mobiles that comply to the requirements specified in section 4.2.1, the only mobile phones the prototypes have been tested on are the phones mentioned in table 4.1, and these are the only mobile phones the project can guarantee will work with the games

Emulators

An emulator is a program that tries to emulate another program or device. In developing the prototype of this project, the project team will use emulators to emulate mobile phones, i.e. by testing the prototype on emulators. The project team will also test the prototypes on the test mobile phones, but emulators will be used to ease testing. Since the prototypes will be created in Java ME the emulators used for testing will emulate

Java-enabled mobile phones. In the Sun Java Wireless Toolkit, there is functionality built in which emulates a typical mobile phone with a virtual keypad and a screen emulating the mobile phone screen, i.e. showing the output from the prototype. The various mobile phone manufacturers' also provide their own mobile phone emulators, which may be better suited for testing the prototype on the various manufacturers phone models, as the emulation will be closer to that of the original. Since emulators can't emulate mobile phones a hundred percent accurately, the prototype will still be tested on the physical test mobile phones.

4.2.2 Location

Location offers a new perspective to gaming. The location of the mobile phone, and thus the player, becomes relevant in the game. This could be used in several ways. The entire game could be built around location by having the location of the player in reality have meaning to the character in the game world. The player could for instance move around in the real world and as the player moved the character in the game world would move accordingly, perhaps by having the character in the game world move in a computer representation of the world(this could either be a 2D drawn map, or perhaps even a fully 3D world). It could also be used as a interface tool, in the sense that the location of the user has an effect on the interface, or the objects shown on screen. For instance as an aid for museum tours, where the location of the user decides what information is shown [Wang, 2005].

As for the technologies used for finding the location of the user, there are several options. Some mobile phones support GPS, Global Positioning System, either directly in the phone or as an accessory. The GPS system consists of several satellites orbiting the earth, control and monitoring stations on Earth and the GPS receiver(incorporated in some way into the mobile phone). The GPS satellites broadcast one-way signals that give the satellites position and time. The user uses this information to calculate its three-dimensional position and time. The method used for calculating the user position is triliteration, which require at least 3 known positions, i.e. satellite positions. The benefit with GPS is that it is freely available, except for the equipment. The problem with it is that it is controlled by the US government, which means that in a crisis situation the acquisition of the GPS location in certain areas may be unavailable because of security reasons and military strategy. GPS can also not be used for indoor positioning. Another drawback is the need for a GPS enabled mobile phone.

Another way to find the location of the user is by GSM localization. There are several ways of getting GSM location information about a mobile phone, but most of them involve the Cell Identity of the phone, i.e. the identity of the cell, or coverage are of the base station, that the phone reside in [Anderson]. Just using the cell id only discovers which cell the phone resides in, which will not give an accurate position. By using timing information, that is using the time information to calculate more exactly where in the cell the phone is, the accuracy of the positioning can be raised. By using more than one base station in the positioning accuracy can be raised even further. This is done by using timing information from 3 or more base stations to calculate the position(Time of Arrival(TOA) [Anderson], [Drane, 1998]). There are also other methods that can be used. The main advantage with most GSM location techniques is that they are supported directly in the current networks and thus it work with all mobile phones that uses the network. Another advantage is that it can be used indoors. The disadvantages with GSM location is that it is not very accurate, it can find the location of a mobile phone with about 60-1000 meter accuracy [Fuglem, 2004], depending on the method used and the density of base stations. The accuracy also varies with the amount of base stations, making it more inaccurate in rural areas.

WLAN(Wireless Local Area Network) can also be used for location purposes. There are several ways of doing this. One way is by setting up several WLAN antennas around an area. The antennas then listens to WLAN traffic and give the positions of the MAC addresses found within the antennas coverage. The accuracy then varies with the distance to the antenna. Another way of getting WLAN location information is by retrieving the signal strength of all nearby WLAN access points. This information is then used to calculate a position based on an experience database. This system will require calibration of the experience database. The main advantages of WLAN location is that it offers a fairly accurate, 0.5-5 meters [Fuglem, 2004], position of a WLAN apparatus and that it can be used indoor by setting up several WLAN access points. The disadvantage is that several access points will be needed, thus increasing cost, and that not every mobile phone today have WLAN, which reduces the users that can use the technology.

In addition to the before mentioned technologies Bluetooth [Hallberg, 2003] and Ir(infrared) [Abowd, 1997] technology can also be used to get location information, however these methods require that the technology is available on the mobile phone.

4.2.3 Hand-held game consoles

Although outside the scope of this project, there have been many hand-held game consoles over the years, and some of them have been very successful. Hand-held game consoles are usually aimed exclusively toward video games, even though Playstation Portable(PSP) [Playstation Portable] offers the possibility to watch movies, listen to music, watch photos and connect to the Internet with the built-in Wi-Fi functionality. PSP, along with Nintendo DS(DS) [Nintendo DS], are the market leaders among the hand-held consoles today. The DS also offers support for browsing the Internet though WLAN. Both PSP and the DS offer multiplayer locally through Wi-Fi, or over the Internet through WLAN. Both the DS and PSP offer 3D graphics, even though the PSP is more capable in that area, and the DS also offers dual screen(hence the name; Nintendo Dual Screen), where one of the screen is touch sensitive which offers new opportunities for game design.

The DS and PSP both are the evolution of several hand-held game consoles that have come before them. Hand-held gaming consoles dates back to at least the 70's with the

Microvision [Microvision]. There have been many consoles such as the Sega Game Gear, Neo Geo Pocket Color, Wonderswan Color, Gizmondo and the N-gage and the improved version of N-gage, N-gage QD, [N-Gage QD] which was a combination between a mobile phone and game console created by Nokia. The N-gage was unsuccessful though and was discouraged, however Nokia are planning to reintroduce N-Gage as a gaming application toward the end of 2007 [N-Gage]. This will not be a console however, but purely a gaming application.

Arguably the most popular and most bought hand-held gaming console is the Game Boy [Game Boy] created by Nintendo and originally released in 1989. The Game Boy was originally in two colors only, but later a version with color was released, the Game Boy Color. The original Game Boy was also released as a smaller version, but with the same games, called the Game Boy Pocket. In 2001 Nintendo released the Game Boy Advance which offered much more computational power than the Game Boy Color, and had richer colors, almost rivaling the Super Nintendo. The Game Boy Advance has since received several redesigns including the Game Boy Micro and the Game Boy Advance SP which both improved lighting and redesigned the appearance of the console. The Nintendo DS is the newest addition to the Nintendo hand-held family. The Game Boy family is arguably the most selling console franchise and as of June 2004 outselling the entire Playstation family [Liz, 2004].

4.3 Mobile network technologies

There are several network technologies used for mobile phones, but they are generally divided into three so-called generations,1G, 2G and 3G, and 4G for the next generation. The different generations are divided by the functionality offered in them. The entire section is based from information from *Computer Networks, Fourth Edition*[Tanenabum, 2003], Wikipedia articles about mobile technologies[Wikipedia] and Ericsson technology articles[Ericsson, Ericsson, Edge], except for the Bluetooth section which takes its information from the Bluetooth webpage[Bluetooth].

4.3.1 1G

1G, or first generation, mobile phones had analog voice and were introduced in the 1980's. The idea of frequency reuse and the division of geographic regions into cells was introduced in this generation. Geographic regions are divided into several cells. Each cell uses some set of frequencies not used by its adjacent cells, but a cell may reuse frequencies used in nearby cells(but not adjacent). The system used in Norway which incorporated this cellular system was NMT, or Nordic Mobile Telephony, which had cell sizes in the range from 2 km to 30 km, and supported two frequency bands(450 and 900).

4.3.2 2G

The second generation introduced digital voice. The system introduced to Europe for the second generation was GSM[1]. GSM uses narrowband TDMA(Time Division Multiple Access), and allow eight simultaneous calls to occupy the same radio frequency. GSM operates on the 900MHz and 1800MHz wavebands in Europe. GSM has used various voice codecs to compress 3.1 kHz audio into between 5.6 kbit/s and 13 kbit/s. 2.5G is a term used for technologies which are not truly 3G, but makes improvements over 2G technologies, such as GSM. Two such technologies are EDGE and GPRS

EDGE

EDGE, or Enhanced Data rates for GSM Evolution, is an enhancement over GSM and allows for packet switched applications. EDGE is built upon GSM, but achieves higher capacity and performance by introducing sophisticated methods of coding and transmitting data. Today it can offer user bit-rates of around 250 kbit/s, with end-to-end latency of less than 150 ms, but has as a theoretical maximum bit-rate of 473.6 kbit/s. EDGE is often thought of as a 3G technology, dependent on how it is implemented, with the highest bit-rates in within the requirements for a 3G network.

GPRS

GPRS, or General Packet Radio Service, is an overlay packet network on top of other 2G technologies, e.g. GSM. It offers its users packet-switched data transmission, and the users are usually billed per kilobyte of information received. GPRS allows users to wireless internet and data communications, and offers users throughput rates of up to 40 kbit/s.

4.3.3 3G

The third generation brings improvements to voice transmission and gives possibilities for digital data to be send over the network(even though this is also supported in 2.5G technologies). Such data could be multimedia such as music, audio or video, internet access or messaging services.

UMTS

UMTS, or Universal Mobile Telecommunications System, or the most common form used, W-CDMA(Wideband Code Division Multiple Access), is a 3G technology which supports theoretical data transfer rates of up to 14.0 Mbit/s, but the expected transfer rate is transfer rates of up to 384 kbit/s. The frequency bands originally defined in the UMTS standard are 1885-2025 MHz for mobile-to-base and 2110-2200 MHZ for base-to-mobile.

[1]Global System for Mobile Communications

HSDPA

HSDPA, or High Speed Downlink Packet Access, is an improvement to the UMTS standard which improves latency and data rates from 284 kbit/s to 14 Mbit/s in the downlink and 5,8 Mbit/s in the uplink.

4.3.4 WLAN

Apart from the traditional mobile technologies described above, certain mobile phones also have support for WLAN, or Wireless Local Area Network, which is short-range packet data communication between base stations and user terminals(i.e. mobile phones or any other device with WLAN support, or between terminals. WLAN enables higher data rates than that of even 3G technologies, with data rates of up to 54 Mbit/s, however WLAN generally have shorter range in their base stations and thus require more base stations to have the same coverage as mobile technologies, and generally speaking the coverage of WLAN is lower than that of mobile network technologies. WLAN has several standards but the IEEE 802.11.x(x is one of several available standards) series of standards is arguably the most widespread used of the WLAN technologies. The newest standard is the 802.11g[2] standard, which operates in the 2.4 GHz band, and offers up to 54 Mbit/s data rates.

4.3.5 Bluetooth

The Bluetooth Web page describes Bluetooth as such: *"Bluetooth wireless technology is a short-range communications technology intended to replace the cables connecting portable and/or fixed devices while maintaining high levels of security"*. The technology is created to be robust, have low power consumption, and have a low cost. Bluetooth operates in the 2,4 GHz ISM[3] band, and divides its frequency band into 79 RF channels, and uses a frequency hop transceiver in order to combat interference and fading. Bluetooth incorporates three different classes which are separated by their power consumption, which in turn decides their effective range. Bluetooth defines two modulation mode, called Basic Rate, and the optional mode Enhanced Data Rate, which in turn has two variants. The Basic Rate mode offers 1 Mbps data rate, while the Enhanced Data Rate offers 2 Mbps and 3 Mbps, depending on which modulation used.

Bluetooth utilizes a master-slave topology, and forms piconets. A piconet consists of two or more devices that occupy the same physical channel, which means that they are synchronized to a common clock and hopping sequence. The synchronization and hopping sequence is controlled by the master device, and the devices that connects to the master becomes slaves. Up to seven slaves can be active in a piconet, but many more slaves can remain connected in a parked state. The devices in a piconet can be connected to more than one piconet, but one node, or device, can only be the master

[2]There is however a new standard, the 802.11n, but it is not yet approved as an standard

[3]Industrial Scientific Medical

of one piconet. When there are several connected piconets, they are called scatternets, however the Bluetooth core protocols do not offer routing capabilities between devices in piconets, but higher level protocols can be added on top which add this functionality to the scatternet.

As mentioned in the description of Bluetooth, it intends to replace cables connecting portable and/or fixed devices, and it can be used for instance to transfer data wirelessly between mobile devices, mobile and fixed devices, or mobile devices and extra accessories, e.g. between a mobile phone and a Bluetooth headset.

Chapter 5

Mobile games and social gaming

This chapter takes a look at mobile games and discuss some of the potential in mobile games as well as describing some of the game genres available today, where a game genre is a group of games with similar gameplay and game mechanics. The chapter also discusses social gaming and social interaction in games. This chapter contains information originally written for the depth study performed by the project group in TDT-4570 [Nøsterud, 2007]; though in some areas it has been altered to better fit the slightly altered focus of this project. However the information used here will not require prior knowledge of the depth study to be understandable.

5.1 Mobile games

Games have existed for a long time. Remains of old game boards of various games have been found that are as old as 5000 years [Fox. 2002]. Computer and video games themselves are rather new in the game concept, dating back to the 1940's, popularized on the home market by the release of the Nintendo Entertainment System in 1985 [Fox. 2002]. After 1985 games have become increasingly complex, both in gameplay and graphics, and modern PC and console games, now represented by the Nintendo Wii [Nintendo Wii], Sony Playstation 3 and the Microsoft XBox 360, all have very realistic 3D graphics, and they offer online multiplayer for its users. Hand-held consoles and their games have also been popular, as described in Section 4.2.3.

Mobile games outside of the hand-held console market are relatively new, even compared to computer and video games. One of the first games, at least in Europe, was *Snake*, which was embedded in Nokia phones, and was released in 1997. The first games for mobile phones where simple, small games compared to its console and computer cousins, and originally they were played in short time periods, often while commuting between the job and work. The games provided relaxation and a small escape from everyday concerns. After these initial small games, browser based WAP-games and SMS-based games appeared, but with the introduction of color display and downloadable games in 2001, wireless gaming was brought into the masses [Pelkonen, 2004]. The introduction of 3G(and beyond) network technology has also helped mobile gaming reach new levels in terms of quality and user experience, and today gamers can enjoy games ranging from simple puzzle games, to 3D multiplayer games [Chau, 2006].

Mobile games are growing in popularity. A estimate from 2002 showed that out of a total market sale for computer games of approximately $20 billion, the mobile game market was $0.9 billion. Other analysts from Informa Telecom & Media estimate that the game sales for 2005 reached $2.5 billion, and predicted that by 2010 the game sales would be over $10 billion. Juniper Research is even more optimistic, predicting that mobile games sales will grow from $3 billion in 2006, to $10.5 billion in 2009, and in 2011 and beyond the sales will reach annual revenues of $17.6 billion. An estimate by The ARC Group from 2002, predicted that the amount of mobile gamers would rise from 196 million in 2002 to 667 million in 2005. The increased number of revenue is also believed to come from casual gamers, and unlike the computer and console game market, the mobile game market is approximately equally split between female and male gamers. In *MOBILE GAMES BUSINESS* [Luukainen, 2007], Anssi Vanhanen claims that casual games such as puzzle games generated one-third of the total revenue for the first quarter of 2006, with a full 65% of all mobile game revenue generated by female wireless subscribers. Both the amount of players predicted and the market revenue predictions show that there is a relatively big market for mobile games; however several other reports show that while many people play mobile games, not everyone buys new games. Ovum Asia Pacific estimated that only about 5% of mobile users buy games, and the speculate that the reason for this is that most people who play mobile games, play the games that come embedded with the phone, and as an example of this, MMEtrics, a mobile data company, stated that in November 2004 over 10 million subscribers downloaded and paid for games for their games, but there were more than 47.1 million subscribers that played games that where already embedded into their mobile phones. Another reason that mobile users don't download that many games is that the download service offered to the users is difficult to use, and that the games are difficult to find. A representative from Telcogames claims in an interview with telecomasia.net that some of the responsibility for such a low percentage of users downloading games also come from an insufficient effort from the value chain actors that is responsible for marketing and promoting the games to the users. The estimates of this paragraph stems from [Chau, 2006, Pelkonen, 2004, Gibson, 2006].

The thing that separate mobile games from computer and video games, is the fact that they can be played everywhere since people tend to bring their mobile devices with them. *Micro Java Game Development*[Fox. 2002] names four qualities to aim for when designing games for mobile devices; **they should be easy to learn** since the small hand-held screens leave small room to have detailed descriptions and tutorials for new players, and most users won't bother playing the game if it's to complicated. **Clarity of visuals**, since screens on mobile devices generally are relatively small, the graphics should be as large as possible to make it more attractive and easier to play the game. **Simplicity of gameplay**, since many users are interested in short gaming sessions, and because the keypad can be awkward to play on, the games should have simple controls and be clear and easy to understand. **Quick game periods**, since users often play the games on their way to work or while waiting in line, the games should be broken down into short, quick sessions. However, other studies also show that many players play games in longer game sessions as well, from about 15- to 20-minutes, and that players are more frequently playing the games in the evening, while they are at home[Luukainen, 2007]. **Interactivity**, by giving the player the possibility to play against other players, the sense of community grows among the players making it more likely that they will continue to play the game.

5.1.1 Multiplayer mobile gaming

Ovum Asia Pacific estimated that mobile multiplayer games generated 12.5% of the total revenue on the Asian mobile game market in 2005[Chau, 2006], indicating that there is a market for multiplayer games. Multiplayer and community aspects in multiplayer games started with highscore support, tournaments and limited chat functionality in the earlier mobile games and where followed by turn-based games which did not require real-time transfer latency, since the user usually spent more time on each turn than the network latency. In recent years however, real-time games with multiplayer are becoming more common, but the main problem with real-time multiplayer mobile games is still network latency, measured in hundreds of milliseconds across the Internet, some times rising toward thousands of milliseconds over mobile operators' networks across the Internet. Another problem with such games is the network bandwidth, indicating that the data transmissions should be kept to a minimum. A factor with multiplayer games is that the network traffic increases with the amount of players making the bandwidth increase proportionally with the number of players[Powers, 2006].

5.1.2 Pervasive gaming

Pervasive gaming has its roots in ubiquitous computing, which looks at integrating information processing into everyday objects and activities. The mobile phone has since its first appearance become an ubiquitous consumer device[Rashid, 2006], by letting the user have a truly mobile device with a relatively powerful computer, at least when compared to the early mobile phones of the early 90's.

When a computer becomes pervasive, it becomes a companion for the user that the user can keep on him(her) constantly and everywhere, and the device becomes an offerer of informal, unstructured activities without clear starting or ending points. Pervasive gaming uses the notion of a constant companion, and implements and exploits the unique features it offers in game design. Pervasive gaming offers the users mobile, place-independent gameplay, in the sense that the games do not have to be played on a single location, but the player can bring the game with him and play it wherever it suits the user, and the location of the user may even be incorporated into the gameplay. Pervasive games also integrate the physical world into the virtual world. This can be done in several ways, and using data from the physical world open up many new ways of playing games and could even create new game types. The final point that pervasive games offer is social interaction between players[Jegers, 2006].

One particular type of pervasive games use the location of the player in the gameplay, and may be called location games. These games gather the location of the player, and sometimes locations of other physical world objects are gathered as well, through one of the methods described in Section 4.2.2, and this information is then used in the virtual world of the game. This can be done in several ways, and there have been several games made that use location information in the gameplay, such as *Human Pacman* [Cheok, 2004], which combines the physical world with the classical game *Pacman* and requires the user to wear complicated, wearable-computer equipment. Another example of a game that uses location information in its gameplay is a game called *Capture the Flag*[Cheok,2006], which creates a version of the classic Capture the Flag gameplay where the players are split into teams and places a flag, which represents their base, on an actual location in the physical world by using a GPS transmitter, and the objective then becomes to capture the other team's flags by finding the GPS transmitter of the other teams in the physical world, and then bring the flags back to their own base.

5.2 Social gaming

Social gaming, and social interaction between players have existed since the early days of gaming, and games such as *Pong* had the option for multiplayer where two players could play against each other in the same game, using the same screen and game console. With the growth and rise in popularity of networked games, and perhaps in particular, the Internet, gamers now also have the opportunity to have social interaction with other gamers within the virtual world of the game, without being in the same room as the other gamers. Thereby the social gaming grew from being social interactions between gamers located in the same room, playing the same game together, to also allow social interaction between gamers situated possibly at very long distances from each other, something which gives new challenges, as well as new possibilities.

In *HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS* [Bartle, 1996] Richard Bartle looks at MUDs[1], and which player types exist within these games. Bartle finds that there are generally 4 play styles to be found in MUDs, and in most virtual worlds, which are the achiever, the explorers, the socializers and the killers. The achievers focus on achieving goals specific to the context of the game, e.g. by gathering points or rising in levels. The explorers focus on exploring the virtual game world, and understanding the underlying mechanics of the game, e.g. by finding hidden places within the world, or by finding bugs in the game. The socializers focus mainly on the social aspect of the game, i.e. in the social interaction with other players. The focus of the killers is to impose themselves onto other players by killing them, and they are generally more interested in the fighting, or combat aspect of the game. Bartle uses the graph in Figure

[1]Multi User Dungeon ,or Domain

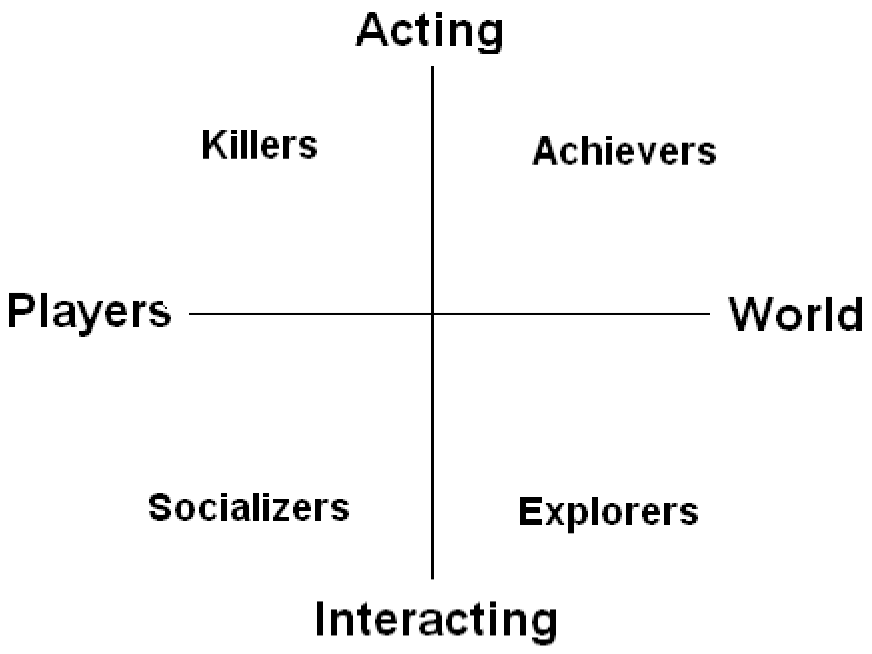


Figure 5.1: Interest graph

5.1 to show the main interests of the players who fit into one of the four categories, i.e. killers are more interested in acting out against other players inside, while explorers like to interact with the game world, but are not as interested in the other aspects of the game. Players according to Bartle all fall into one of the four player types, and while players in one player type may use several of the aspects of the other player types, it is usually to gain benefits in his(her) own player type, e.g. for a killer to be good at killing other players, (s)he must first be of a high enough level, and have good enough equipment to be able to kill other players, thus some degree of achieving is needed to be a successful killer.

In *Social gaming interactions* parts 1 through 3 [Appelcline,2003-1, Appelcline,2003-2, Appelcline,2003-3], Shannon Appelcline takes a further look at the player types found by Bartle. According to Appelcline the two player types of killers and socializers form a spectrum where the range of socialization goes from cooperative interaction, from the socializers, to competitive interaction, for the killer archetype. She also claims there is yet another type of socialization not found by Bartle, the game-agnostic behavior, or the freeform socialization. The difference between freeform socialization and the other two types is that freeform socialization is not goal oriented.

The competition socialization can have several forms, but the most basic and most used is direct competition. Direct competition in games is supplied by allowing players to fight other players. If not restrained or regulated, this type of gameplay can be considered an nuisance by other players, e.g. if the game allows players to attack other players regardless of whether the other players wants it or not, the players attacked by the other player may be annoyed, and repeated deaths caused by this will become a serious nuisance for the player. Appelcline therefore suggests several other mechanics that could help improve the direct competition. Agreeable fights are fights where both players agree upon fighting each other. This can be accommodated by having special areas created solely for player versus player combat, or to allow duels between players where both players agree upon the duel, or even have special servers dedicated to player versus player combat. The other suggestions were that of lowering the consequence of the competition, e.g. by having other penalties for losing a competition other than death, e.g. by getting loss in honor or reputation by losing a competition. Appelcline also suggests that direct competition does not necessary have to be in the form of combat, but could be in negotiation, in oral contests, or for instance in sport contests. Appelcline also describes two other forms of competition, the resource competition and the economic competition. The resource collection competition is aimed toward competition between players about the resources found in the virtual world, and she claims that for games to support resource competition the games should give players limited resources, but players should also be allowed to have more of one resource than another. Resource acquisition should also be rewarded in order to ensure that the players are interested in collecting the resources. The economic competition also have several forms; the capitalistic competition, which is the straight forward supply and demand mechanism, where the market decide the prices on wares, auctions; where players use auctions to sell and by their items, voting; where voting is used as a means of changing the game rules by giving the player a certain number of votes decided by certain in game factors, and then the player can use these votes to vote for game changes, and finally bids, which is the gambling of the game world, where the player places bids on certain events.

For the cooperative socialization the direct cooperation involves several players working together to achieve a goal inside the game world, which is often to defeat an opponent in the game that require several players to work together. There is also possibilities for hierarchical cooperation, in which the cooperation is based on hierarchical relations between players, e.g. by having a master/slave relationship between players in which the master gives tasks to its slaves, which in turn cooperate with the master to receive an award, or perhaps in order to grow in the social hierarchy. The supportive cooperation is a cooperation form where players offer resources to each other in order to achieve the goal desired.

The freeform of socialization is usually offered in games by giving the players the basic ability to chat with each other. There are however other possibilities for games to support freeform interaction between players. Freeform interaction is either not goal oriented, or it is not supported by the system, i.e. the game. This means that freeform socialization does not spur from goals set by the system, but rather from the players themselves, often because of boredom, to get attention, or as an ice breaker. Creative pursuits is an interaction form where the players come together to create stories, pictures, or any other objects the game allows. Even though some games do not have hierarchical structures, players will often create their own structures through hierarchical building, for instance by claiming ownership of specific items as a part of a social hierarchy. Freeform competitions are competitions created by the players and are not supported by the system, i.e. these are competitions originally not supported by the system, but created by players, e.g. by having races inside the game.

In *Living a Virtual Life: Social Dynamics of Online Gaming* [Kolo, 2004] Castulus Kolo and Timo Baur takes a look at the game *Ultima*, which is a MMORPG. Here they found four player types, based on the amount of time they spent in the game. They also found three distinct levels of social behavior and social interaction. The social micro-level consist of the individual player and their character, and why the individual player decide to play such games. The social meso-level consist of the social interaction among players, and the social formation among players. In *Ultima* players are often members of a guild, which is an organization of several players which often has its own social structure and leadership hierarchy, but common to all guilds is that the social relationship between the members of the guild is usually stronger than that of players outside the guild. The last level is the social macro-level, and is a broader sense of social relationship, and often consists of communities of players, e.g. the entire community of *Ultima* players.

The social interaction types, and player types presented in this section gives an indication of the level of social interaction in multiplayer games. It also gives an indication on which features should be incorporated into a multiplayer game in order to support social interaction between players, and shows that social interaction between players is more than simply chatting with, or killing, other players.

5.3 Game genres

Video and computer games have traditionally been placed in genres. Genres traditionally help gamers, or the players(users) of video and computer games, classify games in order to make it easier to talk about similarities between games [iHobo]. By looking at similarities between games one can subtract the essence of the game and place it in a genre. Games that are similar by nature may attract a certain type of gamers, i.e. some gamers like a certain type of games better than others. Some players may even exclusively play games from one genre. This can be used by market analysts and marketers in order to market games specifically for the appropriate audience, or it could help the developers and the marketers in deciding what kind of game they want to create based on the size of the user group, as opposed to creating games purely out of artistic or narrative reasons. Purely making games to fit a particular genre however may make them less enjoyable and unimaginative, if the developers choose to only implement features currently available within the genre. Such thinking will make games stagnate, making no room for new innovations in gameplay and game mechanics, and may cause new games to become less attractive to gamers simply because they are "just more of the same". Thus innovation and imaginative thinking also becomes a factor in game development. However thinking in genres is a good starting point in terms of thinking about basic functionality that the game should offer.

Genres are usually divided into genres and sub-genres, where the overall functionality, the core elements of the gameplay, and the game mechanics decide which genre the game belongs to. Genres are then the overall description of the genre, while the sub-genres give a more detailed view of the game mechanics usually found in games within the sub-genre. Thinking of game genres as taxonomies, the genre - sub-genre relationship is similar to subtype-super type relationship, or a parent-child relationship, or in object oriented terms; the subclass and the super class relationship. However not all games can be placed in only one particular genre or sub-genre, since they can be mixtures of genres. Some games may belong to one genre, but have certain elements from other genres. These elements may influence the game to various degrees. A game could for instance be a first-person shooter game, but have elements from role-playing games, e.g. by having an inventory system, or by having the player gain new abilities as (s)he gets further into the game. *Deus Ex* [Deus Ex] and *BioShock* [BioShock] are good example of first-person shooters that incorporate elements from role-playing games, but are still characterized as first-person shooters. When the various elements the game incorporates are of equal importance gameplay vice, the game is a cross genre game, i.e. the game belongs to more than one genre at once.

There are no standardized genre specifications or genre division mechanics, even though there are several taxonomies and genre description available. Mark J. P. Wolf created a list of about 40 genres [Wolf, 2000], while the game website GameSpot [GameSpot], which follows and reviews games, uses a list of more than 30 genres to classify the games that they review. There is also interest in genres and game classification in research. Craig A. Lindley [Lindley, 2003] suggests a high level framework for classifying games that classify games by using several dimensions. The framework looks at the amount of ludology, simulation and narratology. The framework can also be extended for instance by looking at fiction versus non-fiction games, or virtual versus physical gaming. There are however no final definition or standardization of game genres, and the classification frameworks that have been introduced so far are open for discussion and criticism. Thomas H. Apperley [Apperly, 2006] discusses several of the proposed classification frameworks and looks closer at four common and popular genres, concluding that putting games into certain genres is difficult in that several games should be able to belong to several genres at once. He also discusses the various methods for classifying games, which usually consist of looking at either the narrative aspect of the games, or those that oppose this notion, the ludologists. He also concludes that market-based categories often divide games by their similarity to previously released games.

The genres discussed in this section are based partly on genres used by the gaming industry and game web sites, but also through own experience and on how well suited games in these genres are for the project and the scope of the project.

5.3.1 Action

Action is a somewhat ambiguous name for a genre since most games contain some element of action in them, however there are certain games that have the action element as their main element and there are some sub-genres that fit well inside the action genre. The main focus of action games lies in combat by having the player facing one or more opponents which aim at harming or killing the player. The task of the player is then to dispose of or avoid these opponents. The combat can be the main component in the game, as in first-person shooters, or just be a tool to advance through the game, as in platform games.

First-Person Shooters

First-Person Shooters(FPS) puts emphasize in combat and shooting from the perspective of the character controlled by the player. Such games are usually 3D games where the player see the action through the eyes of the character they play. This gives a feeling of immersion into the game by having the player become the main protagonist. Become in the sense that the player becomes the character since he controls the character directly and sees what the character sees. FPS games usually involve weaponry of some sort and the player usually have the opportunity to aim the weapon in any direction, although in earlier games the player could not aim their weapon, but instead relied on auto-aiming, a mechanism where the game helps the player to aim at their opponents. Most FPS games have some degree of violence in them, where the player shoots opponents.

Multiplayer in FPS games has usually been a selling point of games in the genre, since this seems to be popular among many gamers. Multiplayer in FPS games usually has several players fight against each other. The players fight each other in one of several gameplay modes to gain what is usually called frags, or points, i.e. points rewarded to the player for killing other players. These modes range from all-against-all matches, often called deathmatch, to team-based gameplay. In team-based matches the players form several teams which then fight against each other. There can also be other objectives to the game than pure killing such as Capture The Flag(CTF) where the teams try to steal a flag located in the enemies base and return it to its own base to gain points. There are many more modes available in modern FPS games.

Multiplayer FPS games are well suited for mobile phones in theory. They offer possibilities for both quick and longer game sessions. Console and PC FPS games usually allow the players to decide on how long the rounds should last, how many frags are needed to win, or how many flags the team must get in order to win a CTF match. By allowing the players to choose the game settings the players can choose how long the want to play the games, making it ideal for mobile phones. The disadvantage with having such game styles on the mobile phone is the graphics requirement, especially when the games are in 3D. The keypad also becomes an issue since for traditional console and PC FPS games the aiming and steering of the character is done with either the mouse, or with analog sticks on the gamepad. Free aiming using the keypad could be awkward on certain phone models. The amount of data that must be send in such games must also be taken into consideration, especially when many players play the game. FPS games on mobile phones should therefore be scaled down appropriately with aim help, graphics that suit the mobile platform, and other concerns related to mobile phone development.

Third person shooter

Third person shooters share some of the properties with FPS games, but the player see the character in the third person perspective, often looking over the shoulders of the character(or in some cases the camera can be controlled by the player). Popular third person shooters include *Gears of War* [Gears of War], created by Epic Games and released in November 2006 on the Microsoft XBox 360.

Third person shooters incorporate most of the same multiplayer possibilities and game modes as FPS games, and thus have most of the same limitations for mobile games.

2D Shooters

2D shooters are essentially third person shooters in 2D, i.e. the perspective of the character is viewed in third person, but the character and the game world is drawn in 2D. 2D shooters generally involve shooting elements and they have most of the game modes and possibilities as the third person and first person shooters, however since the graphics in the game is 2D, the computational power needed by the mobile phone is less, and the information needed to be sent over the network also decreases, since the positions of the players are no longer 3D positions. Thus 2D shooters might be a better alternative than third- and first person shooters, however there will still be a problem will game control through the keypad. Figure 5.2 shows a screenshot of the 2D multiplayer shooter *Soldat* [Soldat], created by Michal Marcinkowski. The game is a multiplayer 2D shooter with several available weapons and game modes, and has several players fight each other.

Other

Other action sub-genres include such genres as:

Platform games Platform games usually involve jumping to and from various platforms, such as *New Super Mario Bros.* [New Super Mario Bros.] created by Nintendo for the Nintendo DS. *New Super Mario Bros.* offer both single player, and 2 player multiplayer, however usually platform games don't incorporate multiplayer.

Action-adventure Action-adventure usually involve adventure elements, action elements and puzzle elements, or more generally problem-solving. Games in this genre include such games as the *Zelda* series from Nintendo and the *God of War* series created by SCEA for the Sony Playstation 2. These games are usually not known to have multiplayer support, but there are exceptions.

Figure 5.2: A screenshot of the game *Soldat*

5.3.2 Casual

Casual games as a genre is perhaps not the most used in the industry, but the term casual here is meant to reflect on the fact that these games are mainly targeted toward what is known as casual players, i.e. players that play games purely for recreation and fun, and possibly not for long periods at a time.

Puzzle

Puzzle games usually have the players solve logical puzzles, to navigate complex environments or to use quick reflexes and hand/eye coordination to solve problem. The main gameplay mechanism in these games is problem solving where the player is faced with a problem that he must solve, some times within a time limit. Some games use point systems to make the game more interesting and competitive for the players. The small size of most puzzle games make them a good fit for the mobile phone platform, and the games often have an "easy to play, difficult to master" mantra, making it easy for new players to pick up and play, but hard to master.

Figure 5.3 shows the game *Tetris DS* created by Nintendo for the Nintendo DS, released in 2006. It is a remake of the original *Tetris*, which is arguably the most popular puzzle game. Tetris DS have several gameplay modes, including the original *Tetris* mode where the player must place different shaped blocks on the bottom of the screen and cover and entire row with blocks, which in turn makes the row disappear giving the player points in return. *Tetris DS* also have multiplayer functionality as shown in Figure 5.3, where the opponents' game boards are shown in the upper screen. The objective for the player then becomes to get more points than the other players.

Figure 5.3: A screenshot of *Tetris DS* for the Nintendo DS

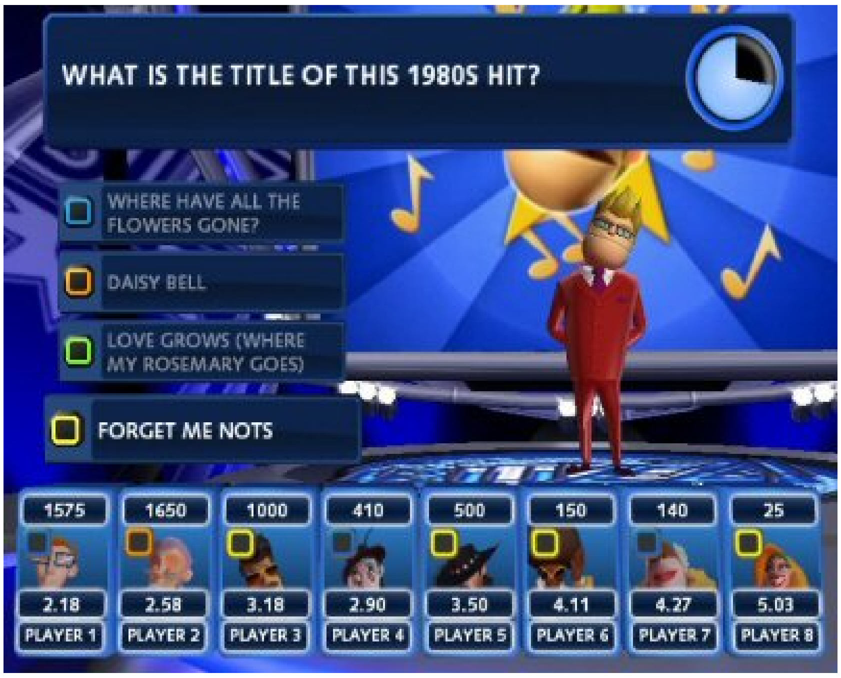


Figure 5.4: A screenshot of *Buzz! The Mega Quiz* for the Sony Playstation 2

Puzzle games are well suited for mobile phones, as described above, and competitive multiplayer games should be doable on a mobile platform; the graphics need not be to computational straining and the network traffic required between the players should be low, compared to other games.

Trivia

Trivia games are games that have players compete either against a computer controlled opponent or against other players in question answering. Players get points for answering correct, for answering fast or in some other way defined by the game. Trivia games are well suited for multiplayer gameplay and trivia games have existed long before the computer. The gameplay it self may not be exiting, but there is certainly a social aspect to the games, and the focus of the multiplayer matches becomes to beat your friends, or opponents, and perhaps show that you are smarter, or at least better at trivia games, than them. Figure 5.4 shows a screenshot form the game *Buzz! The Mega Quiz*, which is a game, released for the Sony Playstation 2 that have several players compete against each other using specially designed controllers formed as game show buzzers. The controllers help the players immerse themselves into the game, making the sensation of being a contestant in a game show stronger. *Buzz!* is also played on a single Playstation machine making it a social game. *Buzz!* include several different game modes and use elements such as video and music in their questions. Trivia games are well suited for mobile phones since the graphic requirement is relatively low, and the mobile phone is well suited to act as a game show buzzer. Using mobile phone features such as vibration could further increase the gameplay. There are however some issues that arise such as synchronization issues, especially if the different players use different network technologies that may have different transmission speeds. This is however only a problem in game modes that require timing, e.g. when deciding who answered first, or whether a player answered within the time limit. If these issues are solved trivia games are well suited for mobile phones, because of their relatively low computational cost, their competitive and social gameplay and the ease of playing such games(the ease of playing trivia games lies of course in the difficulty of the questions as well as the interface shown to the user).

5.3.3 Fighting

Fighting games usually have the player face an opponent in hand-to-hand combat. The opponents may be computer controlled or controlled by another player. Fighting games usually have the player comply to certain fighting styles, or martial arts styles. For instance the character controlled by the player could be skilled in kung-fu, while the opponent could be a wrestler. Older fighting games might not put that much emphasis on realistic fighting styles, but many of the newer fighting games have a bigger focus on having realistic fighting techniques, such as the game shown in Figure 5.5, *Tekken 5* created by Namco LTD. and released for the Sony Playstation 2 in 2004. Some fighting games are also restrained to a single fighting style, for instance boxing games, or wrestling games. The aim for the player is usually to make the opponent unconscious, or to kill the opponent. This is usually done by performing physical attacks against the opponent, which then looses health if the attack was successful(most fighting games incorporate a blocking mechanism which reduces damage). Some fighting games also allow the players to use items such as swords or spears to hit their opponents, and some games even incorporate some long range attacks, e.g. by having the player cast a fireball at their opponent if the player performs a series of key presses in the correct order(known as a combo).

A special sub-genre of the fighting genre is the beat'em'up genre, which usually consisted of side scrolling stages where the player fights against several computer controlled opponents before continuing to the next stage. Such gameplay is well suited for cooperative multiplayer, where two or more players play together to defeat the computer controlled opponents.

Fighting games can be well suited for mobile phones depending on the complexity of the game. Modern fighting usually use button combos to perform complex movements, and this could be an issue with the keypad control on mobile phones. The graphics need not be a problem, especially if the games are created in 2D, but network traffic could be an issue, and to some extent timing issues.

Figure 5.5: A screenshot of the game *Tekken 5* (2004) for the Sony Playstation 2

5.3.4 Racing

In a racing game the player have control of a vehicle, which could be a boat, a car, a motorcycle or any other vehicle used for racing, and then compete in races where the point usually is to get from point A to point B with the fastest time. There are different game modes available in racing games, but most of them strive to get the racing feeling and have the racers race either against computer controlled opponents or against other players. Racers often vary in the realism that they offer. At one end of the scale you have the simulator games which strive to create as realistic a game as possible, and on the other end of the scale you have the arcade style racers, which usually have their focus on creating a enjoyable and fun experience in a racing setting. The difference typically shows itself in steering, weather conditions, damage models, car models, etc. Modern racers usually offer different view of the cars, such as views from inside the car, and viewing the car from behind the car.

Racers can be well suited for mobile phones, especially when made in 2D, but simulation style racers may require to much computational power, so the arcade racers is perhaps the better choice. Network issues due might be an issue, but the problem becomes less for arcade racers.

5.3.5 Role-Playing Games

Role-playing games, or RPG's, usually give the player control over one or more characters, or adventures or protagonists, which (s)he must guide through combat and adventures. In single-player there is usually a storyline that the would-be heroes must follow, and the player controls the protagonists as they venture through the story. Many RPG's use magic as a part of the combat system, and the protagonists can cast spells, which usually have either a negative effect against he opponents, or give a positive effect to the player and his party. Most RPG's incorporate a character-development system, where the player can be upgraded and gain additional powers and abilities by gaining experience points, either by killing monsters or by doing quests, which are missions given by non-playable characters in the game that the player must complete. Most RPG's also incorporate items and statistics that define how well the player's character performs in the game(some times called stats by gamers). The items range from weapons and armor to potions that increase the player's health. The statistics usually decide factors such as how much health the player has, how much damage the player do in combat and how many magic attacks, or spells, the player can cast.

There are different sub-genres within RPG games, and the games differ in terms of how much they rely on action or combat sequences, whether they are real-time or whether they are turn or round based. Some RPG games are more similar to the old *Dungeon and Dragons* pen-and-paper games with an advanced system of dice rolls that decide the outcome of player actions, whereas other games are more action oriented. One of the more popular genres is the Massively Multiplayer Online Roleplaying Games genre, or the MMORPG genre, which is described in the subsection below.

RPG's are usually quite complex and require a lot of work, not only pure programming and art design, but work such as making sure the experience gain is balanced, that the stats are balanced and that the weapon and spell combat is balanced. There is also a lot of work involved in creating quests and in creating the background story for the game. If the gameplay and graphics are not to complex however, RPG's can suit mobile phones well, and there are several RPG's currently available, such as *Doom RPG* and *Orcs and Elves* both created by id Software. Figure 5.6 shows a screenshot from *Orcs and Elves*, which is a turn based RPG for mobile phones created in Java. Network issues could arise though, especially if the game supports many characters.

Massively Multiplayer Online Roleplaying Games

Massively Multiplayer Online Roleplaying Games, or MMORPG's for short, are online RPG's that support thousands of players at once. They usually have a client server architecture and incorporate several servers that divide the player mass into servers with equal loads. Much of the same features found in regular RPG's can be found in MMORPG's, such as experience gain, questing, and combat based on spells and weapons, but the scale of the multiplayer aspect of the games is huge in comparison. MMORPG's usually also focus more on the social aspect of gaming, and in player interaction. These games allow the players to work together in groups to complete tasks, and they support in-game chatting and some of the games also have a social system called guilds, or clans, which allows groups of player with similar goals and interests to join into groups that have a strong sense of belonging for its members.

MMORPG's have the same problem of complexity as ordinary RPG's for mobile phones,

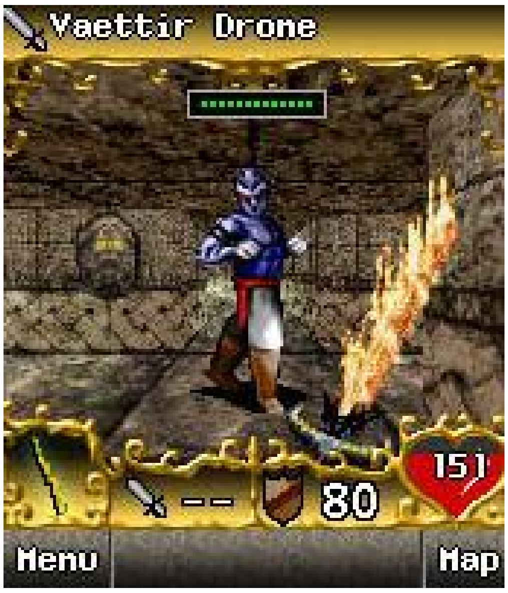


Figure 5.6: A screenshot of *Orcs and Elves* for mobile phones

but on top of that is the network traffic. With thousand of simultaneous players on the same server there is a lot of traffic on the network. This can be a bottleneck on mobile networks, and solutions must be found to the problem. Many MMORPG's use a system called instancing, where they create a special instance of a part of the world, that only the player and his(her) team can see and interact with, thereby reducing the traffic for the players in the instance. Such solutions will be necessary for mobile phones as well.

5.3.6 Sport

The sport game genre consists of games that try to either emulate existing sports, or create new ones. Some games try to simulate the sports relatively realistic, such as the *FIFA* series from EA SPORTS(Electronic Arts Inc.) which try to simulate football and have had many entries in its series. Other sport games are less realistic, or more arcade like, and shifts the focus from realism to other factors, e.g. by creating a more fast paced or simplified gameplay. An example of a more arcade style of football game is *Mario Strikers Charged* created by Nintendo for the Nintendo Wii and released in 2007, as shown in Figure 5.7, which is a arcade style football game where added elements such as power-ups and megastrikes(a special move each player have that is highly effective at scoring goals) shift the focus away from the realistic style of gamplay that the *FIFA* series use. Other games focus more of the strategy and tactics behind the sports, such as the *Championship Manager* series, now developed by Beautiful Game Studios and published by Eidos Interactive, which gives the player the opportunity to be the manager of a football, a job which involve elements such as buying players, planning strategies and supervise training sessions.

Figure 5.7: A screenshot of *Mario Strikers Charged* (2007) for the Nintendo Wii

Popular sports in this genre include sports such as football, American football, ice hockey, base ball and various other sports.

Multiplayer in sport games usually have players either compete against each other, or work together as a team to beat the opponents. In football games direct competition between two players could for instance happen when one player have control over one team, and another player have control over the other team, and the winner is the player that scores the most goals. Some games also allow more than one player on each team, and some games include cooperative gameplay where two or more player compete against a computer controlled team. Depending on the complexity of the games, sport games are suited for mobile phones. They will perhaps never be as complex as the most realistic sports games, but an arcade style sport game fits the mobile phones well, since these types of games can be less computationally requiring. There are however differences between different sports, as some sports involve more complex gameplay than other, and more complex control mechanisms. It is therefore necessary to decide how accurately the sport at hand should be portrayed. Since sport games are usually real-time games, with the exception of certain aspects of manager style games, network traffic and synchronization issues could arise.

5.3.7 Strategy

Strategy games require the player to use tactics and strategies in order to win the game. The games also usually require more attention to planning actions and many strategy games incorporate resource management, where the player is either given a preset amount of resources, or the player must gather the resources. The player must then use the resources to create troops, buildings and other items and artifacts in order to create a means of winning the game. Some strategy games incorporate elements such as construction, where the player must create buildings, roads, etc, commerce, or diplomacy, but other games focus more on pure combat, sporting two or more teams of armies against each other. The complexity of the games grows with the amount of features found in the games, and some games have several means of winning the game, such as the *Civilization* series, produced by Sid Mayer, where there are several ways of winning the game, e.g. by gaining control over all territories through combat or by gaining diplomatic connections with other nations.

Strategy games are usually divided into two categories; real-time strategy games, and turn based strategy games. The two genres differ in the way the game is updated where turn based games usually have each player update their armies, cities and other artifacts as much as the game allows, in turn, such that each player updates his artifacts and then passes over the turn to another player. When all the players have done their turns the current round is over, and a new round begins. Winning conditions in turn based games need not differ from real-time games, but it opens up new possibilities. Real-time strategy games do not rely on turns, but instead let the players update their troops and cities in real-time, even though some actions such as unit building may take a preset time, the game itself does not stop and wait for the player to finish up his or her turn, but instead lets the other players update their troops and cities at the same time, and at their own pace.

Strategy games have generally been difficult to create for mobile devices because of their complexity in gameplay and because of the difficulty of creating good control mechanisms on limited keypads. If the gameplay is simplified and the controls are created to suit mobile keypads, it should be possible to create strategy games, but the amount of network traffic that had to be sent, because of the number of units the player have, could become an issue.

5.3.8 Social game genre discussion

The genres described and discussed in this section are all genres that are present on today's mobile platforms. Multiplayer may not be as common on mobile platforms as it has become on video game consoles and in computer games, but there are nonetheless opportunities to create exiting and fun games in these genres that all support multiplayer in some form.

Apart from gameplay and game mechanical differences between the different genres, the genres may also differ in how well they support social gaming. Certain genres are for instance more suited to house competitive games and thus spur competitiveness between players, while others are more focused on the social experience itself, however it would be wrong to say that a genre that has some sort of multiplayer gameplay does not support social gaming, since even competition between players is a form of social interaction.

The games in the action genre usually have a natural focus toward competitive gameplay where the players fight against each other, especially the first- and third-person shooters, and the 2D shooters. In console games however it is not uncommon for such games to have so called coop modes, or modes where players work together cooperatively to solve problems, kill enemies, and overcome obstacles. First- and third- person shooters also generally have support for voice communication, either through outside applications such a Ventrilo[2] on computers, or built in voice capabilities, such as on the Microsoft Xbox 360. Such voice communication can be helpful to a team that wants to work together to win the match, as they may speak to each other to discuss strategies for winning the match, or to give the players in the team instructions on where to go. All players however may not want to use the voice chat in such a way, and people may for instance use it to talk about issues unrelated to the game entirely, or talk nonsensical in order to harass other players. Some of the current video game consoles also have features that support a sense of community, with features such as creating a personal profile where the players can add friends, chat with friends, and invite friends to player games with them.

In the multiplayer part of these games the social interaction also changes dependent on which team the players are on. The shooter games usually support team based modes, where two or more teams fight against each other, and thus the players that are able to work well together as a team may have an advantage over players who do not know each other, and simply play for their own motives. Some games have options to turn on "team-killing", or "friendly fire", which makes weapons able to hurt players on the same teams as well as players on other teams. Such a mechanic may further show differences in social behavior between players. Certain players will attempt not to hit team mates, while other players will try to hurt players on the same team on purpose, perhaps out of boredom, but it does nonetheless make it more difficult for the team to win the match.

The issues discussed above show some of the social interaction that may occur in games that are usually highly competitive, and it shows that social interaction between the players can be both a hindrance, and an enhancement to the games. Hindrance and enhancement here are however related to the context of the game, i.e. to winning the game. There could however be other objectives equally important to the players that play these games, such as simply having fun, even when loosing, or to simply have a conversation to a friend in a different environment than the players are used to in the real world. This shows the diversity of social interaction, and how it affects games. To certain players conversations about issues not related to the current game are not relevant and act as a nuisance, but for other players it is a part of an experience that make the game more fun.

The games of the platform genre have usually not offered much in the way of multiplayer, but new games in the genre are beginning to offer change this. *Super Mario Galaxy*, created by Nintendo for the Nintendo Wii has a mode where an additional player can join

[2]http://www.ventrilo.com/

a single player game and help the main player by gathering resources and items found throughout the world. The helping players has however no control over he movement of Mario(the controllable character in the game), but the two players are certainly having a social experience by sitting close to each other and helping each other player the game.

Casual games such as the *Buzz!* series are social games, where the players sit in the same room together using the same game console and compete against each other. While the games in the previous paragraph(i.e. the shooters) usually focus on offering multiplayer where more than 10-12 players are connected at once(except for the coop modes), many casual games focus more on having smaller number of players, where the players are often friends that play the games together locally together in the same rooms. This creates a different social experience and environment than the competitive online shooter games; however the experience can be relatively equal in terms of competitiveness, especially when friends are playing together and each player wants to show off to the other players. The competitiveness will also differ between the persons playing the games, as certain players are more competitive than others, while some players just want to play against their friends and have a good time. This goes to show that social gaming not only depends on the games, and what functionality they offer to the players, but to the players playing the games as well.

Fighting games have generally been very social games, especially the versus-fighting games where a player faces another player in direct combat. These games usually have computer controlled opponents to play against, but the more social experience can be found by playing against another human opponent, and they are also generally better opponents. Fighting games have a property where the experience and skill level of the players can affect the experience of playing the games because of the complex movements available to the fighters in these games(i.e. players that play more will have better control over their fighters, and will have learned to utilize the best combos and fighting techniques). A relatively experienced player may find it outright boring to play against a low experienced player, and vice versa, since the experienced player usually beats the less experienced player. Such a situation shows how game experience and the skill of the player may affect the social experience of the game, and how well a person likes that game. A player facing a highly experienced player and loosing repeatedly may be discouraged from playing the game again, even though that player may have a good time playing the game against an opponent of the same skill level.

Racing games usually has a focus on competitive multiplayer where several players race against each other in the same race, and they try to be the fastest. Newer racing titles often have support for online races, where players can race against players over the Internet. Although quite different from shooter games game mechanically, the issues regarding social interaction with other players are similar between the two genres, although the racing games usually do not have game modes where the players are on the same teams. In some sport games on the other hand layers can either be on the same team, or play against each other, e.g. in football games where the players can either control virtual players on the same team, or control a separate team and compete against each other. Newer sport games also incorporate online connectivity as part of the multiplayer experience. Features such as online leagues, tournaments, and world wide rankings help create and support a community round the games, which may in turn create more interest in the games by promising a social experience for new players, as well as a environment for the player to interact with each other, and keep track of each other. Rankings and high score lists can for instance help to create new goals for certain players, where they will try to improve their ranking or high score in order to rise in the ranking. The ranking feature then works as a motivator for the player to continue playing the game.

Role playing games range from non social experiences, to highly social experiences in the form of MMORPG's where often thousands of players are connected to the same game. In MMORPG's the social experience is one of the key factors for the games, as they offer many mechanics that support player interaction, such as textual communication tools(chats), and voice communication. Certain MMORPG's also have the possibilities of creating groups of players that share some sort of social bounds with each other. These groups are sometimes called guilds, and these guilds usually give the players tools to support a social structure for the players. The guilds social structures can be based on different ideals, but the social structures and the goal of the guilds may of course change during the lifetime of the guilds. A guild may for instance be created in order to overcome obstacles in the games that require several players to beat, thus making the reason for creating the guild more competitive in nature, but a guild could also be created in order for players to have a social hub where they can talk to like minded players, or a guild where friends join together to more easily find each other in the games. Once the players in competitive guilds start to know each other better though, the goals of the guilds, and the reason to continue to stay in the guild may change to be more social in nature, i.e. when friendships start to arise. There are other social features in the games than guilds however, such as friend lists.

Competitiveness is usually also supported by MMORPG's, and the player types described in Section 5.2 are usually all found within the realms of the games. Economical social interaction such as trading and bidding, and exploration and gathering are usually all well supported by games in this genre, and the complexity, and size and scope of these games create a unique social environment where many different personalities fit in with the games, and where they can play the games the way that they want to.

Strategy games support both competitive and cooperative gameplay, and some strategy games also have support different variations on how to play the games, such as offering both online multiplayer along with hotseat, in which two or more players use the console or computer to play in the multiplayer session. Competitiveness is usually well supported where the goal of the games are to use strategy and tactic in order to control the units at the players disposal to outmaneuver the opponent and beat their armies.

This Chapter took a more general look at games in general, as well as social and mobile games. The next Chapter will take a closer look at some of the games that are available on the market today.

Chapter 6

State of the art

This chapter looks at some of the mobile games available on the market today, with special focus on multiplayer games.

6.1 Mobile games on the market today

There are several games available on the mobile games market today that offer multiplayer functionality. The multiplayer functionality itself also comes in several varieties, from Massively Multiplayer Online Games(MMO's) over the Internet, to proximity based multiplayer utilizing Bluetooth for multiplayer with other people within a small radius of each other. The games also vary in the gameplay they offer. Some of the games are simple puzzle games with multiplayer functionality, while other games are Role Playing Games, RPG, some with relatively deep story lines and relatively advance combat mechanics. *Samurai Romanesque* [Krikke, 2003] and *Undercover 2: Merc Wars* [Undercover 2] are two MMO games where many players can play at the same time, where *Samurai Romanesque* is a RPG game and *Undercover 2: Merc Wars* is an action game. *Tibia Micro Edition* [Tibia ME] is another massively multiplayer online role-playing where hundreds of players can play in the same virtual world. *Pirates of the Caribbean Multiplayer* [POTCM] allows up to 16 players to compete against each other in real time where each player has control of a pirate ship and fight against the other pirates(i.e. the other players). There are also more casual games available that support multiplayer, such as *Pool Pro Online II*, which is a pool game that offer multiplayer, *Bejeweled Multiplayer*, which offer a multiplayer version of the puzzle game *Bejeweled*, *Chess Everywhere*, which is a chess game that offer multiplayer with not only other mobile users, but also PC and PDA players, *5-Card Draw Multiplayer*, which is a multiplayer poker game, and *Tetris Multiplayer* [Tetris Multiplayer], which is a version of the game *Tetris* with multiplayer supported by EA Mobile. *Duke Nukem Arena 3D* is a first person shooter created by MachineWorks Northwest LLC [MachineWorks Northwest], and is based on the *Duke Nukem* games. The game has multiplayer deathmatch between up to 4 players, single player modes, and is in 3D.

These are just some of the mobile games available with multiplayer support, and the rest of this chapter looks at three mobile games which offer multiplayer gameplay more thoroughly, and the gameplay, the business model, the network technology supported by the games is described for each game, as well as the social aspects of the games.

6.1.1 Pictionary

Pictionary[1] is developed and published by EA Mobile. It is a mobile phone version of the classic board game Pictionary created and published by Mattel. The core gameplay consists of one team drawing figures or pictures, while another team will attempt to guess what is drawn. The game offers single player, hotseat(two to four users play on the same phone by passing the phone between each other), and multiplayer over network. The single player part of the game consists of guessing what the computer is drawing by entering the correct word.

In multiplayer up to four players can play against each other. The players form two team where one each team there is a drawer and a guesser. The drawer is told by the game what to draw, draws it, and the guesser is tasked with guessing the drawing of the other team. Each round ends when either a guesser guesses correctly, time runs out, or a player leaves the match.

The drawing in the game is done one the phone using built in tools, such as brush, bucket, and shapes, in the game, as displayed in Figure 6.1(a), and the game also offer the drawer ten different colors to draw with. Figure 6.1(b) shows a game in session with a finished drawing in colors. The game also has a free drawing mode where the players can practice drawing, and save up to five drawings for later use.

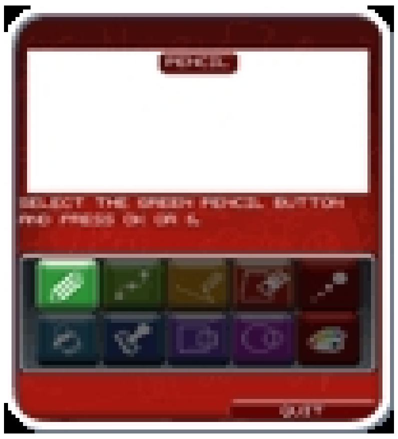
Business model

The game is published by EA Mobile, and is available in several countries by several mobile carriers; however the game is currently not available for download through EA Mobile in Norway. The game has a one time fee for buying, as well as fees for downloading the game which will differ between mobile carriers. During network multiplayer matches additional fees for transferred data may also apply, depending on the mobile carriers. Some mobile carriers charge for the data downloaded, and some mobile carriers also special payment arrangements, such as a maximum fee per day, i.e. regardless of the data down- or up-loaded, the fee is never larger than the maximum fee.

Network technology

Pictionairy has match making built into its multiplayer game, where the game finds suitable matches for the player to join. To join matches, the player need only start the

[1]http://www.eamobile.com/Web/Catalog/US/en/game/mobile/ProductDetailOverviewView/product-25602


(a) Blank drawing screen, showing the different drawing tools

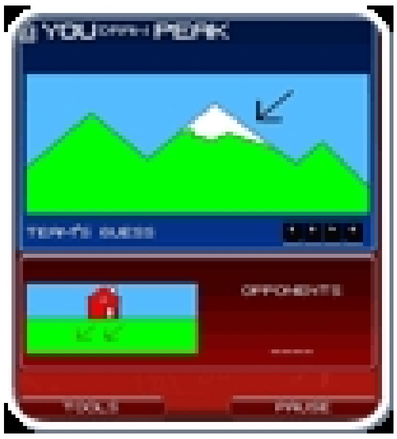

(b) A game session showing a finished drawing

Figure 6.1: Various screenshots from *Pictionary*

multiplayer game, but in order to host matches the players must create an EA account which is done in-game.

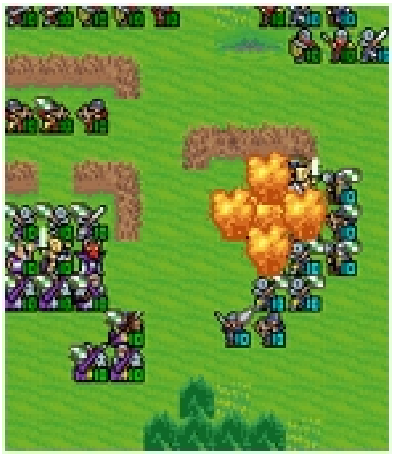
Social aspects

When creating an EA account, the player is given a buddy list, where they can add players and chat with, and send messages to them. The players can also send in-game pictures to each other, and the game also has a online leaderboard where the players can post their scores. The multiplayer itself is divided into networked and non-networked multiplayer, where the non-networked multiplayer has the players use the same mobile phone to play against each other. This may enable a collocated social experience for the players, and players using this option are usually acquaintances. The networked multiplayer force players to work as a team against another team of human opponents, which may give the players a more social experience than being purely competitors.

6.1.2 Mobile Battles: Reign of Swords

Mobile Battles: Reign of Swords[2] is a turn-based strategy game for mobile phones, with functionality for PC as well. The game is created by Punch Entertainment Inc.[3]. The game is fantasy themed, with gameplay mechanics similar to the Nintendo *Advance Wars*[4] series. Figure 6.2(a) and 6.2(b) show screenshots of the game running on a mobile phone. The game has both single and multiplayer modes.

[2]http://www.mobilebattles.com/
[3]http://www.punch-entertainment.com/index.htm
[4]http://www.advancewars.com/


(a) Combat effect

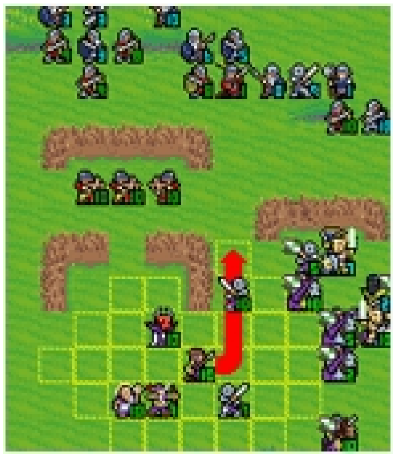

(b) Selecting troop movements

Figure 6.2: Various screenshots from *Mobile Battles: Reign of Swords*

The single player game utilizes a grid based system and has 23 different unit types the player can use in the battles. The different maps have different terrains, and the different terrains have different effects, e.g. slower movement rate. The different unit types consist of a range of different units, such as more traditional sword men, to shape shifting druids. The mode consists of 49 missions, and has a story that connects the missions together. Before each mission, the player decides which units to bring to the battle by using points earned through combat. Certain units work against other, so the player must choose wisely.

The multiplayer aspect of the game is not real-time, but instead uses a system where the player chooses which units to send to battle, as in the single player mode, and then decides the basic tactics for each unit as well as placing them on the battlefield. The player then sends the units to battle another human opponent, and the result of the battle is calculated by the game server without any further interaction from the players. After the battle is over the players are given the result of the combat, as well as given points for their efforts. Each player must create an avatar which is saved on the main game server, and the points earned in combat can be used to buy more forces.

The game also offer the possibility to play the game on the PC, and since the player stats are saved on the main game server, the players can play with the same character both on the mobile and on the PC.

Business model

Mobile Battles: Reign of Swords has a one time fee for buying and downloading the game. Additional fees may also apply when playing the multiplayer mode of the game, but that will differ from mobile carrier to carrier. The game is available for several mobile carriers in several countries, but it is not yet available in Norway[5].

Network technology

Mobile Battles: Reign of Swords is created in Java ME and should thus work on most Java enabled mobile devices which support online network. The game has a main server that holds the player account information, and that calculates the battles between the opponents in multiplayer battles.

Social aspects

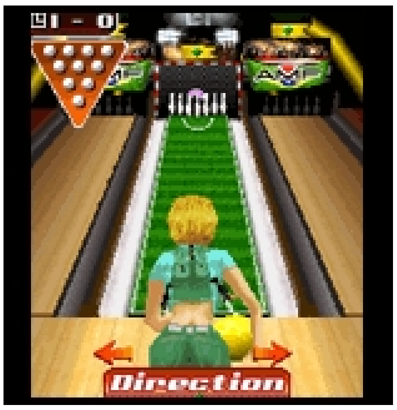
A part from the direct multiplayer combat, the game has online leaderboards with high-scores. The game world is divided into regions, and player may choose a region to join. Each week the leaders of each region are granted noble titles and bonus titles, thus giving more incentive to gain a higher score in the game. The creation of a character also creates more incentive to play the game, as the character, or avatar, is continuously improved by playing and winning battles. The players may create rival lists, which is a list of other players that can be used to compare highscores between players, as a means of keeping track on friends in the game, as well as in the game it can be used to send challenges to other players. If the other player accepts the challenge, a battle between the two players commences.

6.1.3 AMF Bowling Deluxe 3D

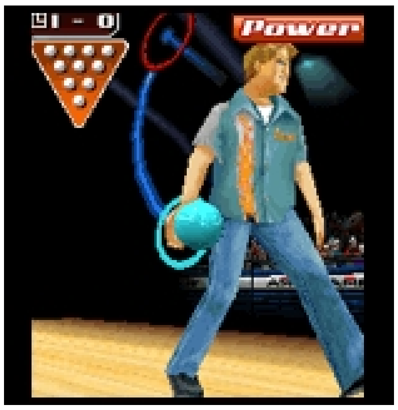
AMF Bowling Deluxe 3D[6] is a mobile rendition of bowling in 3D developed and published by Vir2L Studios LLC[7]. The game has several different modes, including single player and multiplayer modes, and is running in 3D.

The player throws bowling balls in the game by first lining up the character, which represents the player in the game, and is selected among several different bowlers, with the pins by moving to the left or right, which is seen in Figure 6.3(a), where the arrows indicate that the player can move the character to the right or left. After having lined up the character, the player decides the power of the throw, which is shown in Figure 6.3(b), where the characters arm moves up and the player presses a button at the wanted level, which indicates the throwing power. After having decided the power of the throw, the ball is thrown, and for a few seconds after the throw, the player can add spin to the ball. The bowling throwing in the ball is controlled the same in both single player, and multiplayer modes, and both single and multiplayer modes have full bowling matches, i.e. 10-frames bowling with 10 pins, and 2 throws per frame.

[5]Last checked 10 of March 2008
[6]http://vir2l.com/games/amf3d/overview.php
[7]http://vir2l.com/index.php



(a) A player lining up the throw



(b) A player throwing the ball

Figure 6.3: Various screenshots from *AMF Bowling Deluxe 3D*

The single player mode of the game has several different tournaments the player can choose, as well as several distinct lanes to bowl in, such as jungle and classic lanes, and the player can also unlock hidden extras for the game by completing in-game challenges. The game has both hotseat multiplayer where two players play a full match of bowling using the same phone, and online multiplayer. In the multiplayer, the player plays against other human opponents in real-time, but the players throw their bowling balls simultaneously, and after each throw the players are given the result of the throw.

Business model

As with the other games described in this chapter, *AMF Bowling Deluxe 3D* has a one time fee for buying the game, as well as additional fees that may apply during the online part of the game, which will depend on the mobile carrier of the player. The game is available on several different mobile carriers in the US, but it is currently not available in Norway[8].

Network technology

AMF Bowling Deluxe 3D is implemented as a BREW application, but the development team also have a version of the game in Java ME, even though the game has a different name. BREW (Binary Runtime Environment for Wireless) [BREW] is a application development platforms for mobile phones created by Qualcomm. BREW runs between the between the application and the mobile phones operating system, much like Java ME, and allows the user to develop code in C or C++ without coding explicitly for the particular mobile phone. As described above, the players play real-time, although they play in parallel with each other, and see each others results as they play after each turn.

[8]Last checked 11 of March 2008

The players start a online multiplayer game by connecting to the main game server, which is based on the Swervenet network services platform developed by Superscape[9].

Social aspects

AMF Bowling Deluxe 3D has online leaderboards and rankings, where the players can upload their online match scores, and see the scores of the other players. The leaderboard is hosted on the main game server, and is global. During multiplayer matches the players are allowed to taunt each other by using in-game taunts. After each turn the players can view the scores of the opponents via a in-game menu.

6.2 Summary

The games discussed in this chapter shows that there are several available multiplayer games for mobile platforms, more specifically mobile phones, and that there are many different type of games available with multiplayer support, both over network, on the same mobile, and using more local type of networks such as Bluetooth. The games discussed here were a strategy game, a virtual take on a traditional board game, and a bowling game, as well as several other games not discussed as thoroughly as these three. The way mobile multiplayer games implement their multiplayer also differ between the games, where some games use more traditional multiplayer seen in console and computer games, such as the multiplayer found in *Duke Nukem Arena 3D*, whereas other games take more untraditional approaches such as *Mobile Battles: Reign of Swords* which is more similar to e-mail multiplayer found in some PC games than that of the common multiplayer found in other strategy games on other platforms. This takes the limitations of the mobile platform, and focuses on multiplayer gameplay that the mobile platform is able to support and try to create an engaging experience which is not limited by the platform limitations. *Pictionairy* and *AMF Bowling Deluxe 3D* also support hotseat, where the players use the same mobile device and swap the device between each other after each turn. Such a multiplayer type creates a social experience where the players are collocated in the same room able to talk directly to each other with both verbal and non-verbal communication; however they rely on the players to find other players to play the game together with.

The way the various games discussed in this section support social gaming and social interaction also differs somewhat from game to game, although there are some similarities. A common mechanic that is found in all of the three games is online leader boards and rankings, where players can upload their high scores in the game. These leader boards can help create communities around games where the players compete against each other to get higher scores, and creates an incentive to continue to play the game by having the players compete against each other. *Mobile Battles: Reign of Swords* goes even further by giving the top players on the high score lists in-game advantages in the

[9]http://www.superscape.com/gamedelivery/process.php

form of in-game rewards, which further gives an incentive to play the game more, and learn to play it better. In that respect it can also act as a community builder, but it may depend on the player base itself, and the people inside the player base since if certain players dominate the lists, the players that are not able to get on top of the list may find it more of a nuisance than a feature that add to their game experience. *Mobile Battles: Reign of Swords* also divide the game world into several regions where the players may choose which region to join. Such a diversion could further spur a community sense, especially for players within each region since the players in each region may feel more connected to his(her) region than the other regions thus helping creating a community of players that can relate to a region where they can play and compete against each other.

Another mechanic that is used in some of the games, namely *Mobile Battles: Reign of Swords* and *Pictionairy*, is buddy lists, or as it is called in *Mobile Battles: Reign of Swords*, rival lists. The buddy lists act similar to buddy lists found in instant messenger applications such as MSN Messenger where the players can keep track of other players thus making it easier to keep track of players that the players wants to keep in touch with. The list itself is inherently a social mechanism that helps the players keep track of each other, and act as a list of people that each player wants to interact with. The list also spurs further social mechanisms and allows for additional in-game mechanisms that can be added to them, such as allowing the players to send messages to other players on their buddy lists, sending battle challenges to each other as in *Mobile Battles: Reign of Swords*, or as in *Pictionairy* sending in-game objects to each other where in *Pictionairy* the players can send images drawn in-game by the players to each other.

The way the games incorporated direct communication between the players differ between the games. *Pictionairy* allows players that are on each others buddy lists to send message to each other, as described above, or to chat with each other directly. *Samurai Romanesque*, which is briefly discussed here, has a separate application where players can chat together with a instant message similar service. *Mobile Battles: Reign of Swords*, even though it does have a rival list, does not have such a direct communication mechanism within the game. *AMF Bowling Deluxe 3D* as well does not have a mechanism within the game that allow the player to communicate directly with text messages or chat, but the players can taunt each other with in-game taunts which then acts as a game mechanism where the players can show their emotions, or simply gloat about how they are better than the other players.

The last mechanism that support social mechanisms in the games discussed in this chapter is having players work together in teams against other player teams. All of the game allows the compete against each other, and supports competitiveness with such features discussed above such as leader boards, but *Pictionairy* also has teamwork in their multiplayer where there are two players on each team that must work together in order to win against the other team. Such teamwork may further spur social interaction between the players since the players in some form or other must coordinate their efforts in order to win against the other team.

Part III

Own Contribution

Chapter 7

Previous game concepts

This chapter looks at the game concepts conceived in the depth study performed by the project group in TDT 4570 [Nøsterud, 2007], where the subject was similar. The concepts of the depth study apply to the scope of this project as well. The concepts here are general concepts and not full ideas, and to some extent some of them are also abbreviations of the concepts described in the previous project, however knowledge of the previous project will not be needed to understand the concepts in this chapter.

7.1 Pervasive game concepts

This section looks at several game concepts involving elements of pervasive gaming, e.g. by using location in the game. The games described in this chapter rely on pervasive game elements as a main component in the game. A similarity for all the concepts in this section is that the suitability for the game on mobile phones will depend on the implementation of the game, i.e. the graphic complexity of user interface, the network implementation, and to some degree, the location technology used in the game, since location technologies are not available on all mobile phones.

7.1.1 Museum Game

The idea of the Museum game is that the game should be used for educational purposes in museums. It could for instance be used by teachers who take their school class on field trips to museums, especially for children. On such field trips there are usually some kids who are not interested in the displays in the museum, and therefore might find it boring. This leads to them getting little or no use out of the field trip. Using a game could help the teacher in a situation such as this by offering an exiting learning environment in a familiar form for the kids. The game should be educational, as well as focus on aspects that make the kids want to play it.

The game itself could be implemented as a simple quiz, where the game was offered on some sort of mobile device with a quiz game installed. The questions from the game would be created from information available inside the museum, and the players would then have to look at the exhibitions in the museum to gather the necessary information to answer the questions. To make the game more interesting and competitive for the players, the scores from each player could be stored during the museum trip, and at the end of the trip the winner would be announced, possibly with a price, e.g. by giving the winner a souvenir from the museum, or maybe some other treat.

The game itself would then consist of a application on a mobile device, which gives questions to the players, and the players use the application to answer the questions. The questions should be presented in an interesting way, i.e. the presentation should be fitting for the target audience, and the answers should be easy to input. The questions should either be formulated such that players of various ages are able to answer them, or different questions could be created for different age ranges.

A variation of the quiz game could be to have a set of questions offered to the player where the answers to the questions would be found by gathering information from the museum. These questions could also be tied to the location of the player such that when the player came in the vicinity of a museum display, they got questions related to that display. The players would then wander around the museum, and when they came in the proximity of a exhibition information about that exhibition would appear inside the game, and a quiz with questions about the specific exhibition would also appear where the player could answer the questions. This could encourage the players to examine the museum more thoroughly since the player would be able to answer more questions when finding more exhibitions. The game would have to track the location of the players, which could be done in various ways, e.g. a WLAN zone could be established around each exhibition, and when the player entered the WLAN zone (s)he would receive the information and questions about that zone.

A problem for this sort of game is firstly that not every child in a school class has a mobile phone. This could be solved by dealing out mobile devices at the entrance to the museum, but this could be costly for the museum, and a problem of stolen mobile devices could arise. The problem of theft could be solved by registering the devices, and keeping track of which person borrows which mobile device.

7.1.2 Haunted House

This concept uses a real life location to form the basis of a game, and uses the mobile device as a tool in the game, making a bridge between a traditional computer game and real life game. An example of this could be to use a *haunted house* as a basis for a murder mystery by setting up a scary house with various artifacts and introduce a murder, that the players will have to solve. This could either be a competition between the players, or it could be a collaborative effort. The concept is similar to the first game in the *Resident Evil* series, created by CAPCOM, where the player investigated mysterious events in an seemingly abandoned house, except blending a real life location with a computer game, and being multiplayer. A mobile device could be used to show the map plan of the house, indicating which rooms the player has visited and show clues found by the player. By incorporating sensors in the house and sending the information to the mobile device, the map could show in which room and on which floor the players were at.

The mobile devices could also be used for other purposes such as getting clues sent to the mobile device, which would then be used to further solve the mystery, and perhaps the clues could also give further details on where the players should go next. The clues could be given either as textual information, audio logs or as video logs. Video and audio logs could give the impression that the house was actually occupied and give a more realistic experience. Actors would have to record and perform these logs. *BioShock* and *Doom 3* are modern games that use video and audio logs to tell the story of the game, in addition to more traditional full motion video sequences with digitally acted performances.

The mobile devices could also be used more directly in the story by forcing the players to send information to a server in order to advance the story, e.g. by having to answer riddles to open closed doors. The doors, remotely controlled by the game server, would then open if the player answered the question correctly.

The player could also be forced to answer questions or riddles to control the lighting in certain areas of the house. Certain areas of the house could be covered in darkness, but when the player answered certain story related questions, the area would be either fully or partly covered in light. This light could uncover new clues, artifacts or details about the story which was otherwise hidden in the darkness.

Physical world puzzles could also be incorporated into the game. The player could be forced to solve a physical world puzzle, e.g. by having to move certain blocks into the right position, or press buttons in the correct order. The player could then get information that help to solve the puzzles via the mobile device.

Clues in the game would have to be scattered around the house, and the players would have to actively look for them and use them in some sort of game mechanic to advance the story. The mobile device could be used to store information about items gathered, and notes found. Sensors could be incorporated into important game objects to make sure this happened properly. The player could use the notes found to for instance send a password to the server. This password would then be used to unlock new clues or riddles which the player would have to use/solve to advance the story.

This sort of game supports both single player and multiplayer gaming. The multiplayer aspect could either with be a group helping each other to solve the mystery, or a competition between the group members to see who solves the game the quickest.

When designing such a game there are several problems and difficulties that must be overcome. The first thing is of course the technical solution for combining the real life location with the mobile devices, senors, and the game play. Secondly a client server system should be set up, game software must be made, and limitations on mobile devices come into play. Also the length of such a game must be carefully considered, since not everyone will have time to for instance use 4-5 hours in such a game. A solution could be to make different kinds of mysteries with different lengths, or to incorporate some kind of save system that would save the state of the game and the players could then return to the game on another occasion. This would then open up other problems such as the real life objects that would have to be considered, e.g. if a riddle involves transferring an object to another room then the object should be in that room when the players returns. This would have to be carefully considered when designing the actual game play. Another problem with the saving system could be data storage, since if the game becomes popular a lot of information may be needed for storage; however magnetic disc storage is becoming increasingly less expensive over the years and this could be a non-issue. Another issue would be the cost of creating the real life house, the cost of the software and hardware development and the cost of running the game.

7.1.3 Real life event game

The idea behind this concept is that the game uses real life data to govern the data used in the game, the game state,and the gameplay of the game. E.g. using real life data from the stock indexes in a stock exchanging game, or use weather reports to determine how the weather in a game behaves. This could be used to control the wind in a game about sailboats. It could also be used in a farming game where the weather reports would be used to show the in game weather. The in game weather could then be used to decide how good the harvest would be, when the farmer should collect his(her) corn, or when the farmer should gather his(her) animals because of rain.

7.1.4 Capture Point Domination

Capture Point Domination is a game concept which utilizes the location of the players in the gameplay. The basic idea is that a predetermined amount of virtual markers are placed on a map representation of the physical world. The map displays the actual physical world with streets and buildings, and the virtual markers set by either the players, or randomly placed by the game server, are capture zones which the players must catch in order to win the game. The players of the game are divided into teams, and the map could display the other players on your team, as well as your own position. The object of the players then becomes to move around in the physical world while the position and location of the players are recorded using a location technique. The players would then have to move in the vicinity of the capture zones in order to capture the zones.

By implementing an item system in the game, the players could use items to get advantages over the other players. Some items could be attack items, such as booby traps, while other could give beneficial effects, such as being able to see the players of the other team for a brief period of time.

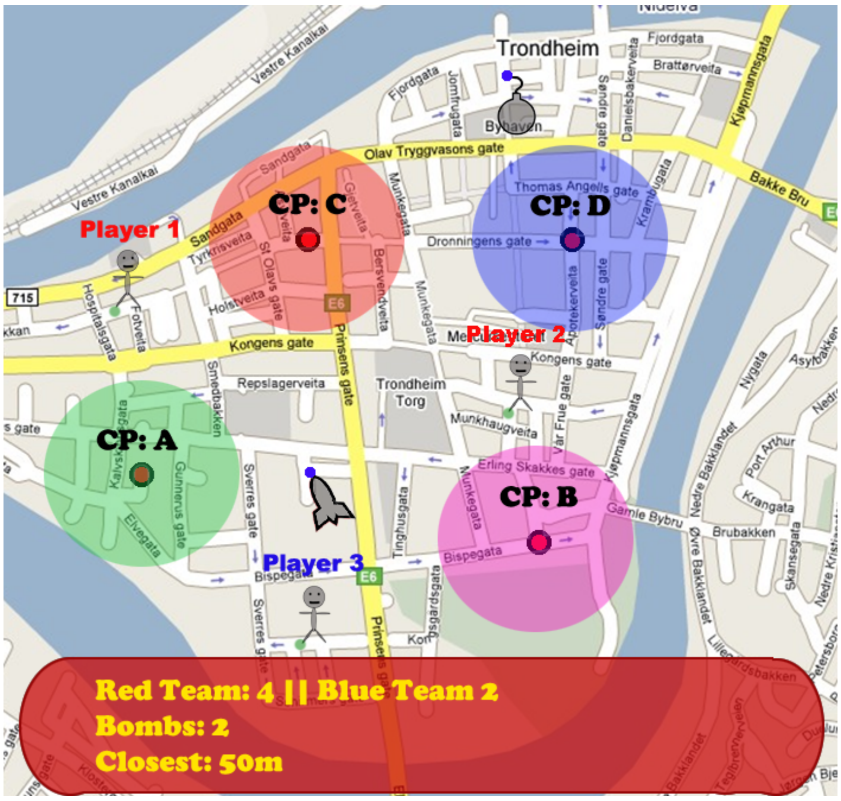


Figure 7.1: A concept image of the *Check Point Domination* game concept

The game could support several gameplay modes, where each mode has different objectives and different game mechanics. One mode could for instance be a mode where the winner is the team that had control over the most capture zones after a predetermined amount of time. The players could then decide which mode they wanted to play before starting the game. The game would then have several options which altered the game, and which the players would have to decide upon before starting the game.

Figure 7.1 shows a concept image of the game view of *Capture Point Domination*. The game window shows the map representation of the center of Trondheim, with street names and some building information. The map has four capture points, CP:A, CP:B, CP:C and CP:D. The center of the capture points are marked on the map with a red dot with a black dot surrounding it. The colored circles surrounding the center of the capture points are the capture zones of each capture point. The player does not have to stand in the center of the capture zone to take over the zone as it is enough to be inside the colored zone around the capture point to take the zone[1]. The other players are shown on the map as well, and they are represented with a player avatar on the map with the name of the player displayed over the avatar's head. The exact location of the player on the map is displayed as a green dot on the map, and the avatar is there merely to make it easier to see the players. The color of the name of the player indicate which team the player is on, and in this case there are two teams; the red and the blue team. While the map in this figure shows players from both teams, this would not be beneficial for the competitive aspect in a game implementation, and thus in a implementation of the game the map should only show players that are on the same team. There are also two items shown on the map. These are represented by an image showing the property of the item, and the exact location of the item is shown by the blue dot next to the image. The players must be in the proximity of the blue dot to get the item.

The two different items on the map are a bomb and an artillery strike. These are just two of many items that might be implemented in the concept. The red part on the bottom of the game screen is a information part of the UI, showing such information as the score of the two teams, how many items the player has, and finally the length in meters to the closest capture point. This is just an idea of which information could be displayed in the information field, and the information should be changed.

7.1.5 Pervasive Battleship

The idea behind the concept Pervasive Battleship is to create a version of the classic game *Battleship* using location technology on a mobile device. For each game there are two teams of players, where the objective of each team is to kill the players of the other team. The team that kills all the players on the other team wins. The players move around in the physical world, and the physical world is represented as a map on the

[1]The different colors of the circles are random, and are not game related but chosen to differentiate the different capture points

mobile device. The map does not show the positions of the players of the opposite team, but it should show the players position, as well as the positions of the other players on the team. The players would not be forced to stay immobile in one position, but would instead be free to move around within the area of the physical world used by the game.

The player attacks other players by choosing to attack a spot on the map. This could be done by moving a marker around on the map with a keypad, or by entering the coordinates directly in a text box. The attack would then consist of a virtual bomb, which would be launched and explode on the position chosen by the player. Every player of the opposite team which reside within a predetermined radius of the explosion is killed, and is out of the game for the remainder of the game round. The game could allow the players to choose for instance to play three rounds, where the winner would be the best out of the three rounds. This option would be available to the players at the start of the game, and the players should have an opportunity to vote over the game options. Other options available to the players could be for instance to decide whether the bombs used by the players could kill the player which sent it or not. This could create situations where the player knew that (s)he was in a zone with a enemy, for instance because (s)he saw the physical player, and the player would then choose to sacrifice (her-)himself in order to kill the enemy.

The game should have restrictions on player attacks, i.e. the players should not be allowed to send bombs limitlessly as this could create situations where the players simply bombed every location possible on the map, causing the game to be to easy. This could be solved by only allowing one player to attack other players at a time, using a turn-based structure. When a turn is given to player, the player is able to attack other players, and after the player has finished his(her) turn by attacking a position, the turn ends and a new player is able to attack. The turn system would only affect the possibility for players to attack with bombs, while the availability to move around does not depend on the turn, as this would be difficult to incorporate in a physical environment.

7.2 MMO Framework

The idea behind this concept is to describe a concept for multiplayer mobile games that can be used for a MMO[2], but that also have other aspects which can give a single-player experience as well as other modes. The basic idea is to split the games using this concept into several distinct parts which use the same character data, where the characters are created by the players. The gameplay is separated into 3 distinct parts; the single player portion, the multiplayer portion, and the "free roaming play". All of the parts are connected through the player character, or avatar[3], and the player uses the same avatars

[2]Massively Multiplayer Online game, games with many simultaneous players(in the region of hundreds, or thousands)

[3]A term used about a players character, i.e. the game character controlled by the player in the game's virtual world

in the three game parts, and the avatars are updated across the three modes.

The player data, i.e. the different data about the players avatars, must be stored on a central database, controlled by a game server. The idea is that a player creates one or several avatars which are used in the various game modes. The concept does not require the game to be a RPG, so the avatar does not have to have complex statistics, but as a minimum the avatar should be able to represent the player in the virtual world. The games implementing this concept should use a client server architecture, since the games rely on a central database containing sensitive player data, at least in the setting of the game, and the client server architecture also gives greater control of login control, and makes it easier to control the player mass in a market situation.

The single player portion of the game is possible to implement as either a pure offline mode, which makes the game a single player game, or to make the single player portion of the game online, which enables the player to use the single player portion of the game as a training facility for his(her) avatar. The idea is that the player creates an avatar in the single player portion of the game, and the avatar can then be trained in the single player portion of the game to improve the avatar, or simply to give the player a fun experience. The data about the avatar must be stored on the game database in the online mode, while the data can be stored locally in the offline mode. The offline mode is conceived to be similar to the game *Tamagotchi* as the player trains and controls the avatar around, but the player is unable to take the avatar into multiplayer.

The multiplayer portion of the game can consist of several different gameplay modes, such as a simple virtual chat environment where the players are represented by their avatars, or as a arena combat style of game where the players pit their avatars against other players' avatars. The player uses the avatars trained in single player in the game modes of the multiplayer portion of the game, but the game can also allow the avatars to be improved and trained in the multiplayer part.

The free roaming part of the game is a simulation environment where the players place their avatars, and then leave the game server to simulate their behavior. This is supposed to be a voluntary game mode, well suited for casual gamers who do not have a lot of time to play, as these players can place their avatars in the simulation environment, and still have their avatars update and improve themselves. The mode could give information to the players in form of SMS messages with for instance reports about the daily behavior and experiences of the avatar, or inform the player that their avatar is in a combat situation, giving either the opportunity to take the avatar out of the free roaming mode, or to engage the enemy in combat with direct control over the avatar. The free roaming mode can also include several different views of the simulations, such as allowing the players to see what their avatars are doing on a website. The player should be allowed to give directions to their avatars upon delivering them to the simulation environment. *Final Fantasy XII*, created by Square Enix, uses a system called gambits, which act as

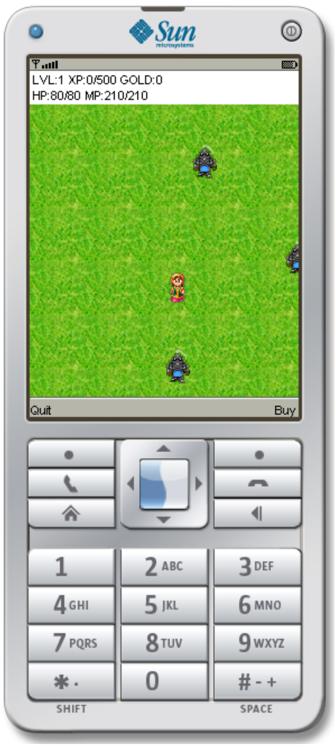


Figure 7.2: Screenshot of a prototype JavaME implementing a verison of the MMO framework

AI routines and give the players the option to automate certain actions by for instance saying that "when my character's health is below 30%, cast a healing spell". By allowing the player to give AI routines to the avatar in the free roaming mode, the player can be more certain that the simulation will choose the correct action in dangerous avatar situations.

The suitability for this concept for a mobile phone will depend on the complexity of the game implementing the concept, but the main problem with games using the concept will be network bandwidth and latency, but again this will depend on the implementation of the concept.

The **MMO Framework** may be suited for several games and game types. Figure 7.2 shows a screenshot of the prototype game built in the depth study by using a simplified version of this framework. The game is a multiplayer RPG (not an MMO) where the player can create an avatar that can either be a magician or a soldier, and the player then use this avatar to kill monsters in order to gain experience and gold. The prototype created a single player part, a multiplayer part, and a simulation environment, and it showcased a simple implementation of the framework, even though it did not function 100% properly since the projet group suffered some problems(the single player and simulation portion of the game functioned, but the multiplayer portion was not fully functional due to time constraints and implementation problems).

The previous example is one of many games that could be created by using this concept, and below three ideas that might be used to create a game with the framework are listed and explained briefly. All of the ideas look at ways to use the single player, multiplayer and free roaming mode.

7.2.1 Robot Fighters

Robot Fighters is a game where the players create their own robots and battle other players' robots in arena style battles, either one versus one, or many versus many. In the single player part of the game, the players can customize their robots and train against a computer controlled opponent in a arena fight. The player gains resources, e.g. virtual money, in the multiplayer part of the game which can be used to modify and improve their robots even further in the single player part. Players could face off against other players in multiplayer either in tournaments which could be events started by the development team and gave higher rewards than the standard fights, or they could fight against either random opponents, or chosen opponents in non-tournament games. Each win could give a standard amount of resources to the winner. The game could also allow other players to place virtual bets on the fights, such that players could get resources even when not winning fights, however they would loose the money they bet if the fighter they bet on lost the fight.

The parts of the robots that could be improved by the players could be weapons, which could either be ranged weapons, such as rifles and hand guns, or close combat weapons such as pneumatically(in order to convey a sense of realism in the game) controlled axes and spikes. Other improvements could be robot armor to decrease damage, improved motors to increase power and speed, etc. The combat in Robot Fighters would be real-time combat where the players moved their robots around virtual arenas and used the weapons attached to the robot to attack the opponents. The robots would have a health bar that would reset after each fight. Each attack would do damage to the opponent depending on the armor of the opponent, the attack power of the weapon on the robot, as well as various other factors which could be applied to the fights, e.g. how blunt weapons work on metal casings etc.

The free roaming mode could be used as a way for the players to gain items for their robots. The players would send their robots into a arena fight simulator which spawned random battles against computer controlled adversaries, and if the robots won they would get random items generated by the server. This would work as another way of gaining items besides spending virtual money gained in the multiplayer fights on items. The items could then be used in the single player mode to improve the robots. The player would set certain parameters that would decide the robots behavior, which fights the robot should fight, how often the player wanted the robot to fight, the fighting pattern of the robot, and more.

7.2.2 SpaceShip Wars

SpaceShip Wars is similar to Robot Fighters in that the players can improve their avatars, in this case their space ships. The player can create one or several space ships which the player then can improve. The ship improvement element consists of modifying and improving various ship components such as armament, ship armor, ship engines, giving the ship new abilities, and more. The game would however not be a arena fighter like Robot Fighters, but instead be a open virtual world, i.e. the universe, which the players could explore. The virtual world of the multiplayer would consist, apart from the players, of several planets where the players could dock their space ships and get quests, talk to NPC's, buy and sell items, and more. The universe would be full of various monsters and space pirates that the player could kill for money and items. The NPC's on the various planets could also give quests revolving around these monsters, or the quests could be deeper and more involved. The game could also give players the option of focusing on trading. The players could be tasked by NPC's to travel trade routes with wares, or the players could make their own trade routes. Resources would spawn randomly throughout the universe which the players could gather, and there would be trade professions that utilized the various resources to craft items. There could for instance be weapon makers, ship makers, etc. The combat in the game would be real-time with RPG elements such that the statistics of the player space ships would decide the strength of the ships in combat. The player could also choose not to partake in the more combat focused elements, but instead focus on the trade aspects as described above.

The single player part of the game could consist of the player improving their ships and train in combat against computer controlled monsters. The training would not give any negative effects upon death, and neither give any benefits upon victory, but instead give the player training for the multiplayer component of the game. The ship improvement could consist of using money and items gained during multiplayer to enhance their space ships. The space ships would have several statistics that governed the attributes of the ships, such as health, strength, speed, cargo capacity, etc., and the ships could be improved by gaining new items.

The free roaming mode would simulate ship movement in the universe, and would have the players' space ships fight against computer controlled space ships. The reward for winning against the monsters could be random resources and money, while the penalty for dieing could be a lowered resource generation rate for some time, e.g. for 5 minutes, during the multiplayer mode. This penalty could accumulate forcing the player to make strategic decisions about the simulation.

7.2.3 Massive Football

The Massive Football game would be a game where the players would have control of virtual football[4] team. The player would get control of a football team that they would have to train, and face other players in football matches. The players could choose whether or not they wanted to be part of virtual football leagues, or whether they instead wanted random matches with other players. The virtual leagues could be based on the player locations, or they could be randomized with a team limit for each league, e.g. the leagues could only have 20 teams. The matches in the leagues could either be organized by the players in the leagues to match their real life schedules, or the server could set time spaces where the players would have to play against each other, e.g. the players could be forced to play against each other within two days of the time limit, otherwise the game ends with a tie, or a loss to the team that did not show up to the game.

The winner of the matches would depend on the players on the virtual teams, i.e. their statistics such as shooting skills, tackle power, etc., as well as the tactics laid out by the player, much like the *Championship Manager* game series. The matches themselves would be simulated based on these statistics, and they could also be animated. The player would have to train his(her) team, and make sure that (s)he had enough players in case of injuries, and the game could also allow the players to buy players from other teams, or trade their athletes with other players athletes. The single player part of the game would consist of the player training the athletes of the team, planning strategies against the next opponent, and allowing the player to buy or trade new athletes. The player would gain money from simulated ticket income from virtual visitors of the matches, and this income would be controlled by the skill and position of the players team in the leagues, if the player had joined a league. This income could then be used to buy new athletes.

The free roaming mode could be a simulation environment where the player could test new strategies and new players in either preparation of the next game, or just for fun. The results from this simulation could be used to alter strategies in the next match, and to check which players would be most suited to use in the next match.

7.3 Mini game collection

The idea behind this concept is to create a game lobby for players where they can interact with each other and also play games against each other. The lobby should allow players to chat with each other, either in private chat rooms, or in public chat canals. It should also list games available for the users of the lobby to play. The users of the lobby should be able to invite other users to game sessions, where they play the games in multiplayer against each other. The lobby should have a list of active users, from which users can choose users to play against, however the lobby should also offer the possibility to join games without inviting other users, or being invited by other players, using a

[4]Not American football, but soccer

matchmaking functionality where the server waits for incoming players until the server limit is reached and the game commences.

The chat rooms or canals offered in the lobby could help players to decide which game they wanted to play through discussion, or the users could simply chat about topics unrelated to the games. The lobby should also include a buddy list, where the users can place other users. The buddy list then acts as a list of the friends the user have made during the time spent in the lobby and playing the games, or it could include friends known prior to using the game lobby. The lobby could also have a rating system for the games where the players rated the games so that the other players could get an indication about the different games. The lobby could even support a system where the players could write small reviews of the games where they wrote their views about the game, which could further help the players find new and exiting games.

The games featured in the game lobby should be short games, hence mini games, and they should offer short playing sessions, e.g. ranging from 5-20 minutes. By having shorter playing sessions it is easier for players to start playing the games, as the games steal less of their time. As for the genre of games the lobby could offer there would be little to no restriction; however games with short game sessions and simple gameplay may be better suited for such environments. The games offered could be such games as multiplayer puzzle games, or multiplayer miniature golf, pool, etc.

The games offered by the game lobby could change over time. The lobby could offer the games as separate downloads where the connecting point would be the lobby itself. The players could then download only the games they wanted to play, and they could download and play new games when they were available. Each game would be charged separately, but the lobby itself would be free to download, and only act as a social hub and as an enabler of multiplayer gaming sessions. The lobby could also offer trial games, or demos of games, where the players could download a limited version of the games for free, and if they liked it, they could buy the full version of the game.

7.4 Turn based multiplayer RPG mechanism

The idea behind this concept is not so much to create an entire game, but rather to describe a game mechanism that could be used in multiplayer RPGs for the mobile phones. The concept uses turn-based mechanics for the combat, while the other aspects of the game can be real-time, e.g. such as player interaction, chatting, and wandering around the game world. The idea is to use a system similar to the one used in traditional Japanese RPGs, such as the earlier *Final Fantasy* games. By incorporating a variation of one of the system used in such games the player can be free to wander the game world and interact with other players in real-time, but when the player enters combat, which can be invoked by the player by walking into a computer controlled enemy or another player, or by selection of an action from an in-game menu option, the player and the enemy(either computer controlled or controlled by another player) is then taken to a combat screen which is only visible to the player and the enemy, and the combatants start a turn based fight.

The combat itself can be implemented in various ways, but the main idea is that the combatants get to attack each other in turns. By using player statistics, such as agility or speed, the game can use these statistics to decide which combatant should get to attack first, how many attacks each combatant can do before the turn is over, and when there are more than two players; decide how many turns the combatant must wait after its turn is over. The combat can be implemented in a fantasy style, a modern, or a futuristic style, and the damage output can be based on statistics of both the equipment of the character, and the character statistics.

The outcome for the loser of the battle will depend on the game that implements it. Certain games will kill off the looser and give a penalty for loosing, while other games will simply give a beneficial effect to the winner. The system could also be used as a duel, where both combatants agree to fight each other, usually for fun, or for showing off, and the winner only gains the ability to brag about the win to the other combatant, while the looser can only loose his(her) pride.

By having a turn-based system, the problem with network latency and low bandwidth can be neglected as the combat then only relies on the turns, and not on the time aspect of the combat. This system may also not require much computation making it well suited for a mobile phone multiplayer game.

7.5 2D Multiplayer Shooter

The idea behind this concept is to create a 2D multiplayer shooter for mobile phones. The perspective of the game should be that of for instance the old *Super Mario Bros.* games from Nintendo.

The game could incorporate one or more of the multiplayer modes found in other shooters, such as deathmatch where there are no teams, and the score of the player is decided by how many other players (s)he kills, team deathmatch, which is a version of deathmatch, but where the players are divided into teams and the score of the team is the accumulated scores for each player, where the score for each player is decided by how many other players (s)he kills. Other modes could include such modes as domination, which creates one or several points on the map which the players must stand on in order to gain points. The points are captured by the players by standing on them for a predetermined time, and when captured the points generate points for either the player in situations where there are no teams, or for the team when teamplay is activated, and the player or team with the most points win(this could either be done by having a point limit where the winner is the first team or player who reaches the limit, or incorporate a time limit, where the winner is the one with the most points generated after the time has run out).

The different modes could differ in whether they allowed players to respawn or not. When players are allowed to respawn in a round, there is some times a timer incorporate such that either the player is generated after a certain time after death, or there could be a independent timer which had players respawn at predetermined time intervals. In modes where players do not respawn there are usually rounds involved, where each round lasts until either a certain objective is reached, e.g. by planting a bomb and having it explode(as is done in certain *Counter Strike* maps), or when a team kills all of the other players of the other team. The winning team of the round then increases its points, and the winning team overall is the team with the most points after either a preset time, or after having reached a point limit.

The player would be in control of a character on screen, and the character would use weapons to kill his opponents in order to gain frags[5]. The game could support several different weapons, both ranged and melee style of weapons(i.e. non-ranged weapons). The weapons could either be modern day such as assault rifles, more medieval style of weapons with with crossbows and swords, or more futuristic with lasers and energy weapons.

One problem that could arise in the mobile platform is that the player could find it difficult to aim his(her) weapon at the enemy because of the limitations found in many mobile phone keypads, e.g. by not having a mouse to steer the aiming reticle. This problem would have to be solved in order for the game to be fun to play. One solution would be to slow down the pace of the game to make it easier for players to aim and fire their weapons at enemies, similar to the game style of for instance the *Worm* series of games, where the players take turns controlling weapon-holding worm characters. This could either be done by level design, i.e. by designing the levels in such a way that encounters between two or more players gives the players time to react to the situation. It could also be done like in the *Worm* games by implementing a turn based system, but that would alter the gameplay significantly.

The maps[6] in the game should have strict boundaries as to which areas of the map the player could access. The maps themselves could consist of various platforms for the players to jump around on, buildings, or nature artifacts, such as rocks, mountains and rivers.

A 2D shooter multiplayer game could be well suited for mobile phones. The graphics do not have to be a problem as it is a 2D game, and the only problem besides the before mentioned keypad problem, would be that of network traffic. The effect of the network traffic problem depend on the network technology offered by the users mobile

[5]A term used by many gamers to describe the points gathered by killing opponents

[6]Each round in the game is played on a certain map, which acts as the battleground for the multiplayer sessions

phones, and this could create differences in both latency and network speed among users, giving users with better mobile phones an advantage. There are however ways to avoid some of the problems[Powers, 2006], and such solutions should be used to make the game more playable.

Chapter 8

New game concepts

This chapter describes the new game concepts conceived in this project. These concepts where created for this project alone, and where not previously conceived in the depth study performed in TDT 4570 [Nøsterud, 2007]. Some of the concepts are described generally, while others are described more thoroughly, however all of the concepts in this chapter are not complete game ideas, but merely display the core gameplay concept and ideas that can be used by games implementing the concepts.

8.1 Strategy game

Strategy games are usually difficult to create for mobile game platforms, especially the more complex real-time strategy games found on PC's. The problem lie in the complexity of the games and the limitations of the systems. There are however strategy games for mobile platforms, albeit in a less complex form than that of its PC counterparts. Turn based strategy games could be well suited for mobile platforms due to the relatively low network traffic needed for these games. This section looks at some concepts that could be used for mobile multiplayer games.

8.1.1 Tower defense

Tower defense: Wrath of Gods created by New Edge[New Edge] in JavaME is an example of a strategy game adaption of the popular tower defense sub-genre for mobile JavaME compatible platforms. The game does not have networked multiplayer however, but it does have hotseat, i.e. the possibility for two or more players to play against each other on the same mobile phone. The tower defense genre has been popularized by Flash games on the Internet, and the game by New Edge is an adaption of the common features of tower defense games. These games focus on tasking the player with placing out certain protective objects on a virtual battleground, which could for instance be guard towers that shoot arrows at intruders, or hired mercenaries with machine guns that shoot incoming hostiles. The games usually have waves of enemy intruders which the player must fend off in order to survive and win the game. The player is usually rewarded with money for killing the intruders, as well as for defeating each wave. The money gathered can then be used to upgrade the troops and equipment of the player, buy new troops, or in some games buy special powers and abilities that has beneficial effects on the player, e.g. by giving the player damage spells (s)he can throw at the enemies.

Even though most of the tower defense games found on the Internet are singe player games, the concept itself could be made into a relatively simple multiplayer strategy game for mobile platforms with either competitive or cooperative multiplayer. Common for both the cooperative and the competitive mode would be the strategic elements of the game, and the strategic game mechanics of the game. The main element of the game would be placing defenses against waves of attacks, either from the computer controlled component, or a human component. To flesh out the gameplay, other elements could be added to the games. The players could allowed to create not only defense structures and immobile defense troops, but also vehicles and troops that could be used to attack the opponent. The acquisition of defense structures and attack units would require virtual money, or to add more complexity to the game; one or more of several resources which the player must collect in the virtual game map. Many strategy games have resource nodes in the virtual maps which the player must control in order to gain the specific resource. The main source of in-game money would be that of defeating the opposing forces, which would grant a small sum of money for each defeated unit, but the games could also have certain areas on the virtual game map that would give additional supply of money when controlled by a player. Such control points are usually important strategic points in strategy games. The virtual game world would be represented by several virtual maps, where each map is the battleground for the specific match between the components of each round. In each round the different opponents would be placed in different areas of the map, and each player is given a start sum which can be used to buy troops and defenses. The defensive structures are then placed by the player on the map, using a tile based system where each structure may have special requirements on placement, such as not being able to be placed in forests, or requiring more tiles than the standard tile number. The round would then commence and the combatants would send their units to attack and defend against each other.

The actual combat should be turn based and each combatant would have a certain available options for each turn available, e.g. the possibility to move their units, or buy new defense structures and units. There would also be limitations on each turn such as units only being able to move a certain distance each turn, and limitations on the number of units and structures that could be bought each round. In each turn the combatant is able to buy units and structures and place them and order strategic movement of units and issue attack orders. The units able to move around the maps would move in a similar fashion to that of the structure placement, i.e. by using a tile based system where each unit would be able to move a certain amount of tiles each round. The combat would be done by matching the combat statistics of the different units, where the one with the most units and the most powerful units would be more likely to win. Each unit would have several sub-units which together constituted the actual unit represented on the game map, i.e. each unit on the game map would have X number of sub-units inside it, making for small troops for each unit. Units within the reach of the defense structures would be attacked by the defense structure one time each round. The game would also use rounds in addition to individual turns. When each combatant finishes its turn, the round is over and a new round begins. In the beginning of each round the combatants would be rewarded with the resources from the control points controlled on the game map, and the each round restriction, such as structure restriction, would have its counters set to zero again. The winning conditions for each match could vary, and could for instance be the last surviving combatant, or each match could have certain conditions that must be met in order to win, for instance to destroy a specific structure found in the starting area of each combatant.

The difference between cooperative and competitive game modes for this concept is the opponent of the players. In the cooperative game mode, the players work together to fend off the incoming computer controlled forces. In traditional tower defense games the objective is usually to survive and defend against attacks on your base, and the cooperative mode is well suited for this. In the competitive mode the opponent for each player would be another player, and the pure defense mode would be difficult to implement. At the very least it would have to be implemented as a round based concept where for each round one team is attacking, and one team is defending against the attacks. The mode described in this section have a different structure where each player can both attack and defend in each match. Both of these modes could however be implemented into the game as different modes of gameplay.

8.1.2 Tanks

Albeit not purely a strategy game, the Tanks concept sport turn-based gameplay with strategic elements. The concept is based on the old concept of opposing tanks which have to eliminate the other tank by firing at them. The players must then take into consideration such elements as the distance to their opponent, the angle of the attack, and the force of the attack. *Pocket Tanks* created by Blitwise Productions[BlitWise Productions] for Windows is a modern version of the classic concept, shown in Figure 8.1 where the two tanks controlled by the two players are placed on opposite sides of the map.

This concept uses the basic ideas of the classic concept and is based on turn-based gameplay, where each player has a limited time for each turn to complete different in-game tasks. Each player have the possibility to have more than one unit though, and the winner of each match is the one that has destroyed all of the units of the opponent. The players earn gold by destroying enemy units, and for winning each round. The money gained in previous rounds can be used to buy improvements and reinforcements for the next round, and thus the games should have relatively short playing times for each round, in order to support consecutive rounds of gameplay for the players. The players starts with a predetermined amount of gold, and can use the gold to buy units before

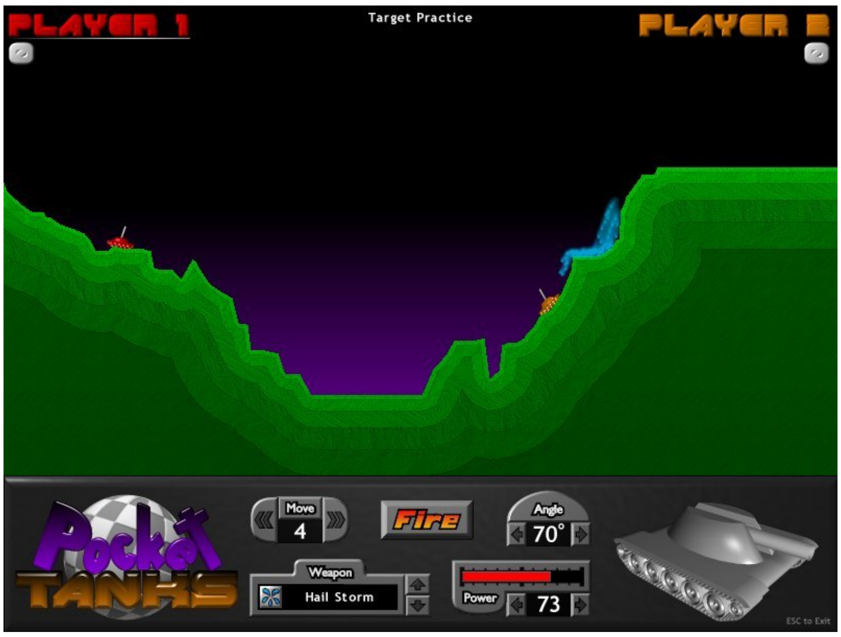


Figure 8.1: Screenshot of Pocket Tanks

each round starts. This is done through a pre-round screen where the players can buy improvements for their units as well as the units themselves. When every player is ready to start the round, the round starts with placing the players units at random locations and the players must then navigate the game level and destroy the other players' units. An alternative to the turn-based mechanic described here, could be to use real-time as a basis, but then add mechanics to slow the pace of the game down. The features could for instance be to add relatively large reloading times betweens each units attack, such that each player attacks with one unit, and after the attack has to wait for instance 50-60 seconds, where in the meanwhile the player has the opportunity to attack with his(her) other units.

The games implementing this concept should have several tank units, with different abilities and behavior for each unit, and different costs depending on the strength of the unit. The player should however have control of the angle and force of each units attacks, unless the attack does not need to; e.g. a homing missile. The units should have either different weapons, or have different movement abilities. Each unit should be able to move, but the movement should be limited for each turn either by having a time limit for each turn, or have a maximum distance each unit can move, which could be a differentiator between units. The units could also be allowed to make more complex movement than strictly forward and backward movement, such as jumping, or the game could have items that granted the player the ability to for instance teleport the different units around the maps, or allow units to being airlifted to other locations by airplanes. Each unit should have a unique weapon, unless its movement pattern and abilities are relatively unique compared to units with similar weaponry, and the weapons could have different effects, such as standard tank missiles, homing missiles, cluster bombs, mines, air strikes, etc.

The games implementing the concept would be best suited with a 2D viewpoint in the game, and not overhead view, but instead have a side view of the combat. A problem with mobile devices may be the size of the screens, as for instance most mobile phones have screens with higher height than width. A solution to the problem could be to implement a mini map of sorts that is placed in the uppermost area of the screen. This would allow the maps to be larger than the actual screen size of the mobile, as the mini map would help the player keep track of enemy units. The upper half of the screen could also then have additional information such as information about all of the players units, and for instance a feature where the players could have a chase camera of the projectiles fired by the tanks. This would work by adding a chase camera to the projectile which then followed the projectile from the barrel of the tank, all the way to the final destination. This feature, along with the mini map, would help the player in targeting off-screen enemies and could function in such a way that it only activated the chase camera when the projectile of the tank targeted areas not currently shown on screen. The maps could have different terrain in the levels, and could also have natural obstructions such as stones and mountains which would force the player to move around the obstacles in order to gain a better position to attack their opponents. In more complex games, the terrain could be deformable and destroyable, such that the gunfire of the tank units would deform the terrain, creating several strategic possibilities, as well as potential difficulty with movement for the units. As described above, the games could also support in-game items such as teleportation devices, or crates that grants the player who obtains them more gold. The different items could appear at random locations on the map and at random or fixed time intervals, and the first player that got to the item would get it.

8.2 Puzzle game

Puzzle games are usually relatively low in complexity in terms of graphics and performance requirements for systems running the games, and thus puzzle games are well suited as mobile games, and there are already many puzzle games available for mobile systems. Currently though, most puzzle games for mobile systems are single player games, but the added functionality of multiplayer could add new depth to the puzzle games as well as support competitive and social interaction between the players.

Puzzle games revolve around solving puzzles, which come in many forms. Games like *Tetris* is a game with puzzle elements and revolves around placing bricks to gain points, but a game like *Portal*, which can be described as a first person puzzle game, is also a puzzle game where the player solves 3D environmental puzzles using a portal gun[1]. The

[1] A gun that creates an enter and exit portal which can be placed on certain areas. The player can then move through the portals.

Tetris style puzzle games are usually not suited to have multiplayer where the players share the same puzzle area, so multiplayer games similar to Tetris usually have separate puzzle areas for each player. The multiplayer sessions then usually revolves around getting the highest score, or finishing first. Certain elements could however be used to improve the social aspect of the games, and some game incorporate similar techniques, such as adding actions that could alter the state in another players game. This could for instance be done by having in-game items which affects other players. The effects of such items could for instance be that a opposing player could have his(her) screen covered in virtual ink, making it difficult to see, make the screen of another player shake violently, making it difficult to see what's going on in the screen, resizing another player's puzzle area, adding elements to another player, which in Tetris would mean adding more blocks, and there are many other possibilities. The rest of this section is devoted to a concept of a multiplayer platform-puzzle game.

8.2.1 Platform-puzzler

The basic concept behind this idea is a multiplayer 2D puzzle game, where the objective for the players is to move through a in-game map which has several obstacles the players must overcome in order to proceed to the next map and puzzle challenge. The concept incorporates platforming techniques in that the players are able to move and jump around the map. Other elements may also be used as navigation utilities, for instance teleporters, jump pads, grappling hooks, and other means of transporting a player to a different location.

The social aspects and interaction in the game can be implemented in three different ways, at least, all of which alter the way the game is played and how each level should be created and which utilities and abilities each player should have access to. The extreme variants of the three ways discussed here are the purely cooperative and the purely competitive game style, which are also discussed in Section 8.3 about the mystery solving game concept, but the concept could also incorporate both cooperative and competitive gameplay. The difference between the various approaches to the social interaction lie in how the players work together to reach the final destination in each level in the game, and how much cooperation is needed in order to overcome the various obstacles in the game levels.

In the completely competitive approach the players are competitors and compete against each other to complete the level with the fastest time, and with the most points at the end of each level. In order to sustain the interest of the player, the game should be tailored to the competitive nature of the players, and give the players a reason to want to win the game. Giving out points for completing objectives and having high score lists are some measures that can be used to peak player interest, but there are other techniques that can be used as well. As suggested earlier the players could for instance have the possibility to gather items that alter the game state of other players, such as placing traps that immobilize the player that steps on it, or items that generate additional obstacles in the game levels, for instance oil stains that make the floor slippery, or a wall that slows other players down. The level design would have to take into account that the players will not cooperate in overcoming the obstacles and thus the puzzles must be solvable by a single person, and the obstacles must be passable by a single person as well.

The opposite direction of complete competitive gameplay is completely cooperative gameplay. In this setting the objective for the players is still to get through the different obstacles and puzzles to reach the end of each level, but now the players are forced to cooperate in order to solve the puzzles. That is, the puzzles and obstacles are designed such that more than one person is required to solve them. The players are then forced to work together in order to get through the level and onto the next. This will require a different strategy than that of a completely competitive game. The score system is one thing that is affected by the cooperative nature off this approach, as generally each player will try to get as high a score as the player is capable off, but now the score of each player is dependent on at least one other player, and the score system itself must also be taken into consideration, i.e. how the points for solving puzzles and clearing obstacles are divided among the players that complete them. By giving each player that completes each level in the same game round the same amount of points, there is little competitiveness between the players of the game, but in a game focused on cooperative gameplay this might not be a problem. A solution to this problem could be to have online high score rankings where the score for each level is uploaded on a global ranking site so the players can see which team of players is the best. This could further spur cooperative interaction between the players, since they are forced to work even better together in order to get a higher score. The other important thing to consider when having a cooperative game is the level and puzzle design. With the focus on players cooperating in solving problems, the puzzles and obstacles in the levels can be designed around that concept and may be designed in such as way that they require more than one player in order to be solved. There are several possibilities in making such puzzles and obstacles. One approach would be to make obstacles in the level that specifically require more than one player to overcome by using their abilities and wits. These challenges would be physical in nature, i.e. requiring physical action in the virtual game world such as jumping, and could for instance be obstacles such as large walls which require the players to stand on top of each other in order for the player on top to jump over the wall. That player would then perhaps press a switch on the other side of the wall that opened a door in the wall to let the other players through, lower the wall in the ground, or raise additional platforms for the other players to jump on. The obstacles could also be tackled in a different manner, by for instance giving each player various abilities that are needed to solve the puzzles. These abilities could for instance give the players the ability to toss their teammates over obstacles, to operate teleportation devices which require more than one person to operate, operate canons in which one of the players must enter carrying a rope which is tied to a firm location on the canon's end, and then being blown across a chasm and finally tying the rope on the other end of the chasm so the other players can cross the chasm using the rope. Figure 8.2 shows a concept image of a puzzle in which

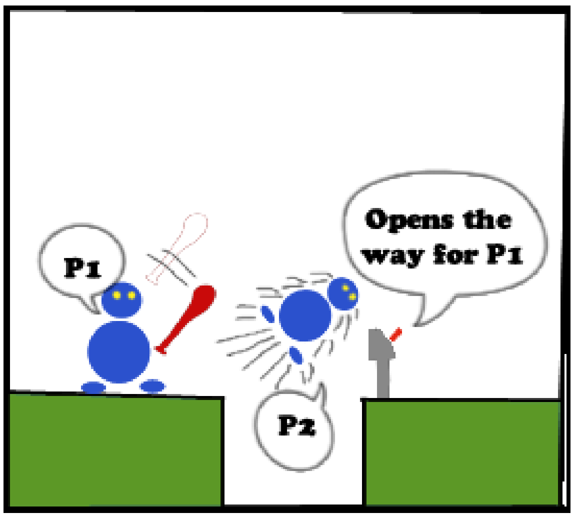


Figure 8.2: A concept image of a possible puzzle for the Platform Puzzler

player 1(P1) and player 2(P2) are faced with a chasm which they can not simply jump over. Therefore P1 uses his(her) club to hit P2, which then is hurled over the chasm by the force of the impact with the club. When P2 lands on the other side of the chasm, his(her) attention is drawn to a console with a large red button on it, and upon pressing the button, a bridge is created that spans the chasm, enabling P1 to cross the chasm as well.

The puzzles and obstacles could also be based on other factors than the virtual physical obstacles discussed above. The players could for instance be placed in various areas of the level, physically separated from each other, where each player push button and pull on levers in order to open doorways and paths for other players. Each player is then unable to finish a level alone, since the player must rely on other players to open the path for him(her). The buttons and levers could be used to open doors, raise and lower platforms, turn the light on and off(in situations where the puzzle depends on the level of light in the room), turn on and off wind(if a player is placed in a balloon, the wind will allow the player to be blown across the room), as well as many other possibilities. One challenge that comes from such a design is that the players may be unaware that they are supposed to help each other, and thus be frustrated when unable to solve the puzzle, open the door, etc. The game must therefore be aware of this issue, and incorporate in-game hints that allows the players to understand that they must wait for other players to overcome the obstacle or complete the puzzle in front of them. Simply allowing the players to send messages to each other could also help the players, but in puzzles where for instance the time aspect is important, such as when a player has raised the ceiling, which is full of nails and spikes, to the top position, and the other player must run across the room in order to not be crushed by the ceiling which shortly after having been raised to its top position comes crushing down again. In such situations the game should give in-game hints to the player that something or someone has raised the ceiling and that it's time to move in order not to be crushed by the ceiling.

The two ways of creating the puzzles and obstacles in the game discussed so far, are the extreme variations of competition and cooperation between players, where in one the players do not cooperate at all, while in the other they are forced to cooperate in order to finish the level. A better solution might be to incorporate elements of both and create an environment where the players are able to chose either to help each other, or to treat the other players strictly as competitors. Each level could for instance have several possible routes to the exit, where each route either require more than one player, or can be done alone. The points for solving each puzzle and obstacle should then be different for the different paths, in order not to make one of the paths more preferable than the other. The single player path could for instance give less points for each obstacle, but because of the nature of the puzzles be easier and faster to perform, while the multiplayer paths could give more points, but at the same time would require more time thus balancing the different paths. In such a scenario the player should not be penalized in choosing either of the paths, but should instead be encouraged to try different possibilities out of curiosity and for the fun of it.

Another possibility would be to have the players work together similarly to the completely cooperative style, but give the players the possibility to abandon the teamwork and instead finish the level alone. The idea is that at each level the players start out on the same team and the puzzles and obstacles require teamwork between the players to solve and overcome, and the players get a relatively large amount of points where each player receives the same amount. Each player however has the option of turning against the other players at which point the level's structure changes and becomes a completely competitively structured level and it is then every player for him(her)-self. The player which then finishes first gets more points than the rest of the players. This structure creates tension between the players since the players never knows if or when a player turns against the team. In the situation where one player abandons the group, all of the players will get less points than they would have if they solved the puzzles as a group, but the first person to finish the level will receive more points than the other, thus forcing the players to consider how they want to play each level. The players who do not abandon the group should not be penalized though, and they should have an equal chance of finishing the level, that is, a player probably would choose a strategic point to go over to the solo mode, but the rest of the player should still have the possibility to finish the level, which is why the structure of the level at that point should be slightly altered to allow the rest of the players to finish the level. The players could also be given items at that point, which could help remove some of the issues of fairness some players may experience upon such a changing point in the game. These items could be some of the items mentioned above, such as traps, ropes that can be thrown on players to slow them down, creating walls in front of people, teleportation devices, etc. As an interesting twist, these items could be given to the part of the players that did not abandon the group, as a way of hunting down the player that left the group, but the items should however be usable against all players.

The three different directions for the game discussed above are not without their flaws. Some of the problems are mentioned above, but there are also other problems. One of the problems stems from the screen size of mobile phones, which is generally relatively small with screen resolutions of 180x250 being in the normal range of most mobile phones. The resolution then limits the number of players that can be shown simultaneously on screen without them cluttering the screen, making it impossible for players to actually play the game. A solution could be, at least for the completely competitive direction, to create a separate level for each player so that the players are not on the same screen, however this solution might be less enjoyable for the players to play as it might feel less like a multiplayer game, and more like a single player game where the score of each player is shown at the end. Another problem with having to many players is the puzzles themselves, for the cooperative puzzles. Creating puzzles requiring 5 peoples, when there are 4 players in the game will be a problem. The solution might be to create puzzles that can scale in proportion to the amount of players, so if for instance the puzzle is getting over a wall, the wall will lower if there are fewer players in the game than originally planned. A limitation in the number of players on each level would also help with the problems.

8.3 Mystery solving game

The gist of this game concept is that several players must unravel a mystery either in competition against each other, or as a cooperative activity where the players must help each other to solve the mysteries in the game. The initial idea of the concept was that of a detective story where the players must solve a murder mystery in for instance an old mansion. The players would upon entering the game session be informed about the murder, and during the course of the game be given clues that help the players toward solving the mystery.

The game itself would be a multiplayer game running on mobile platforms with JavaME. The virtual game world should be represented in either 2D or 3D, and the mansion, or any other location used in the game, as well as the story, should be presented to the player using either in game graphics or pre-rendered animations, e.g. movies or pre-made images forming animations. The initial thought of the project members went to "flashy" cartoon graphics, to create a loose and more humorous atmosphere for the game.

There are several game types which could be used to present the murder mystery to the players, and the game type used for the game would decide how the game play and game mechanics should be implemented in the game, as well as how the game is played by its users. The game could for instance be implemented as an adventure style point-and-click game, as a more quiz or trivia style of game, as a mini game collection style game, or as a more action oriented adventure style game, more similar to the *Zelda* games from Nintendo, than the point-and-click adventure style games like *Monkey Island*. The main part of the game, regardless of game style, should however be to uncover the mystery behind the murder by uncovering clues about the murder and finally revealing the murderer. Each of the clues available for the players would be handed out to the players by awarding them as prices for completing subtasks within the game. These subtasks could for instance be competitive puzzles, mini games, or individual puzzles. The players should also be awarded a game score, or points, for completing these subtasks, in order to support competitiveness between the players. The atmosphere should as mentioned above be more to the lighter side, and it could for instance be a theft instead of a murder for the players to unravel.

The multiplayer aspect of the games should allow for several players to either compete against each other in solving the mysteries, or help each other solving the various puzzles and tasks of the games. The games implementing the concept should consider whether a competitive or a cooperative approach should be taken, or implement game mechanics that support both. In a situation where the players are all situated in the same room for instance, the players could talk to each other in order to solve the problem, but if the game does not accommodate cooperation inside the virtual game world and reward the players for cooperating, there is not much incentive for the players to do so. As mentioned above, the players should be given clues about the solution of the murder mystery or main puzzle, and the players could be given individual clues by for instance giving the players a random clue after successfully completing a subtask within the game, or if the subtask itself enables generating clues relevant to the subtask itself, give the players one of the clues associated with the current subtask. The game should give the players the option to view and go over their clues at any time, e.g. by giving the players a clue inventory list where all the clues collected by each player are listed.

The final part of any of the games implementing the concept should consist of solving the main mystery of the game, i.e. the murder mystery or main puzzle of the games. The main mystery of the game should reward the most points out of all tasks in the game. All of the players should be given a chance to solve the mystery. In a more action oriented game the solution could rely on winning a mini game for instance, or in a more adventure and puzzle oriented game, the players would be allowed to enter their murderer candidate, selected after reading through the clues gathered throughout the various sub tasks of the game. In games relying on the players to use the clues gathered to select a murderer, a problem would arise to players who did not earn any clues, thus creating a situation where such players would have no other option than to guess. In such cases the game could as a minimum give the players a small amount of standard clues for each murder case. In order to support both competitive and cooperative gameplay, the games could give the players the option to share one or more of their clues with the other players, where the player could select to reveal his(her) clues to all of the competitors, or only a selected few of them. The players chosen to receive other players clues then would have to either accept the clues and be forced to share his(her) own clues, or deny the offer from the other player(s). The game could also allow the player to trade clues. The players that choose to share their clues would share the points for solving the murder mystery or puzzle, given that the answer produced by the players is correct. Such a game mechanic would force the players to consider whether or not to play the game cooperatively or not. Having too few clues would give the player a good incentive to cooperate with another player in order to gain the necessary information to solve mystery, while the shared point strategy would create a situation where competitive players would think twice about cooperating with other players, as this strategy might cause the other player to win the game overall, even if the added information from the clue sharing would lead to the player solving the mystery.

This game mechanic could also differentiate players located in the same room, versus players that are not collocated. Players that are collocated have the option to communicate with each other in the real world where the players that are not collocated may not be given such options. In a mixed game, i.e. a game with both collocated and non-collocated players the collocated players could have an advantage by working together both to solve the subtasks as well as discussing the main mystery. One tactic for the collocated could even be to solve the mystery out of the game, but choose not to share their clues in game, thus gaining more points by giving the same answer without point loss. Problems like these are not unique to this concept, but it could be a problem nonetheless and should be remedied in the game.

8.3.1 Adventure style

The murder mystery could be presented to the players as a adventure style of game. Adventure style games usually involve solving puzzles, and exploring and investigating the environment in order to for instance gain items needed in puzzles that will unlock new areas of the game. The adventure genre is usually divided into several genres where the two main genres are the ones focused on action, like the *Zelda* games for the Nintendo console systems, while the more traditional adventure games like the *Monkey Island* series, *Grim Fandango*, etc., are less focused on action, and focus more on puzzle solving and exploration.

The games implementing this concept could be either a traditional adventure style game, or a more action oriented adventure game, or a mix between the two genres. With a setting of a murder mystery happening within a mansion, a adventure game implementing the idea could focus on exploring the mansion to gain clues about the murder, interrogating the relevant suspects to the crime, and solving puzzles related to the environment and the game. There are several possibilities for implementing these features in a game.

One way to do it is to force the players to solve puzzles in order to progress through the mansion and to gain clues about the murder. The mansion could then be divided into several rooms where the players would have to solve the puzzle in each room to progress the next room and finally arriving in the final room, for instance the room where the murder took place, where the players are tasked with identifying the identity of the murderer by using the clues gathered throughout the game session, or simply guess. The players could be rewarded with clues by solving the puzzles in each room. The puzzles themselves could be puzzles relevant to the different rooms, i.e. the current environment of the player int he virtual world, or be a more general puzzle, e.g. a tower of Hanoi style of puzzle.

The actual structure of the room progression could be changed according to the wanted level of cooperation between the players:

Competitive focus With a competitive focus on the game the players would try to get more points than the competitors by solving more puzzles and gaining more clues. One way to organize the rooms in order to support competitiveness would be to have each of the players going through either the same rooms, but not have each players actions affect the other players, i.e. by having a separate instance of the mansion rooms for each player, and the players then would gain clues when solving the puzzles in the rooms and advancing to the next room. The main focus then would be to solve the puzzles the fastest in order to gain access to the final room first and have the most time to solve the puzzle. The players actions could also be made to have impacts on the other players by having each player be in the same room at the same time always, but the first player to solve the puzzle in the room would gain the clue and also open the door to everyone, thus being the only player to gain access to the clue in that particular room.

Cooperative focus In a cooperative environment the players would help each other in solving the puzzles and in order to gain access to the clues and the next room in the mansion. This could be done either by having the players be in the same room solving puzzles that requires more than one person to solve and execute(e.g. by having environmental puzzles that requires the strength of more than one person to solve). The game could also be structured in such a way that the players would have to help each other in order to gain access to new rooms. This could be done by having the puzzle in the player's current room not open the next room for the player when solved, but instead open a door for another player. The players would then not be located in the same virtual rooms, but instead be located in various rooms throughout the mansion where the puzzle in one room opens an exit in a completely separate room.

The games could also focus less on puzzles and more on other factors, such as exploration and investigation, and more action oriented puzzles. The players could be forced to investigate the environments they are faced with, e.g. within a virtual room in the virtual mansion, and find clues in the environment itself instead of gaining them through secondary tasks such as puzzles, i.e. the players could be forced with investigating and exploring the different rooms in the mansion in order to find evidence that support their theories and strengthen their belief that one of the murder suspects must be the murderer[2]. The player should be given the option to store thoughts and clues gathered through this investigation in-game, for instance by allowing the player to store notes in an notebook or inventory system in the game.

With a more action oriented approach to the adventure style, the players could be forced to solve more environment and spacial oriented puzzles, i.e. by for instance pushing levers and solving puzzles involving moving objects on screen instead of solving brain teasers. This approach could also be implemented differently according to the social and collaborative focus of the game. The players could as previously mentioned be forced to work together to solve the spacial puzzles by for instance having one lever pushed by a player open a door for another player. The other player then in turn would have to solve a spacial puzzle in order to open a new way for the first player. The players could also be placed in the same room as before solving spacial puzzles requiring more than one player. In more competitive focused games the players could as suggested above try to solve the spacial puzzles faster than the other players in order to gain more time in solving the final murder, but the game could also allow for more direct interaction with between the players by for instance having objects that the players could place in the other players room in order to slow them down.

The actual moving between the different rooms of the mansion could also be done in different ways. In the more action oriented approach for instance the players could have avatar representations of themselves on-screen which they would have to move around using the keypad of their mobile devices, while in the more traditional adventure style of games, the players would not necessarily have to have an on-screen avatar to control, but could instead be given a simpler interface where they where given animated images and had a limited way of interacting with the images in order to solve the puzzles, or gather clues from though investigation and exploration, in order to progress through the game and through to the next room.

8.3.2 Quiz/Trivia style

Another game genre that could work well with the concept is that of the quiz or trivia genre, as a version of the quiz concepts in Section 8.4. The overall atmosphere and main goal of the games implementing the concept would still be a murder mystery, and the various questions asked in the games should tie in to the games by for instance having the correct answers to questions reward the player with a clue about the murder, and the questions themselves may even relate directly to the plot of the murder mystery, thus indirectly leading the player toward the identity of the murderer; i.e. since the questions relate directly to the plot, a correct answer to a question regarding events in the plot make the players aware of that specific part of the plot, while wrong answers will tell the players that either a previous assumption made by the player is incorrect, or that parts

[2]The investigation could be conducted for instance in a similar fashion as for instance the TV series *CSI* where the investigators look through the crime scene in order to find evidence and clues about the murderers identity.

of the plot can be ruled out as irrelevant to the overall plot.

The questions asked throughout the game could also be unrelated to the plot of the murder mystery. The murder mystery would then be the background of the game, and correct answers to the questions would reward the players with clues relevant to the murder. The final task for the players would then be to solve the mystery utilizing the clues gathered through the question answering to solve the mystery. The way the questions would relate to the background story of the game could be that of tasks given by characters withing the game world. The mansions ins the games could be inhabited by several non-playable characters which would act as question givers within the game. These characters would then give out clues to players answering questions correctly. In the case of questions that are related to the background story of the murder mystery, the characters could for instance have their own suspicions around either circumstances regarding the murder, the environment in the virtual game world(e.g. suspicious changes in the mansion that could be related to the murder), or suspicious persons in the mansion. For the concept of questions unrelated to the story the characters could for instance give questions as a means of challenging the players by only giving clues to the players who really deserve, i.e. the players who can answer their questions correctly.

As for the multiplayer aspects of the games, the questions presented to the players could be either given to the players individually, or to all of the players simultaneously. In either case, the correct answer would lead to a clue be given as a reward, but the difference between the two options lie in that for individual questions, each player has the possibility of gaining the clue, while for simultaneous player questions there can only be one winner of each question, and only one player will be given a clue for each question. The competitive aspect of the games can also change between the two different options. For the individual questions the competition between the players is less, since each player is given equal possibilities to answer each question, and each player has the possibility of gaining the clue from the question. When the question is given to all of the players and only one player can receive the clue, there is a higher tension between the players, and a higher incentive to answer the question correctly since the players can loose out on clues, which could ultimately leading to the players not being able to solve the murder mystery.

8.3.3 Mini game style

The mini game style is somewhat similar to the adventure style discussed above in that it revolves around the players progressing through separated parts of the game, and in each segment, or part, the players have the chance of gaining a new clue in the murder mystery. The difference between the two styles is that while the adventure style was more focused on puzzles, the mini game style focuses more on mini games. The mini games can be any mini game suited for the game, and it could even include puzzle elements such as that of the adventure style. The games using the mini game style could be implemented using the room structure as explained in the adventure style section, where in each new room the players enter, there is a new and unique mini game that the players must play and win in order to gain new clues. The final challenge for the players would still be that of solving the murder mystery though.

As with the quiz style discussed above, the mini games could be created to be relevant to the setting of the virtual mansion, the environment within the mansion, or be more generic standard mini games. The mini games themselves could also be presented to the players in a similar fashion as discussed in the quiz style with non-playable characters giving out clues through challenging the players to mini games where the winner of the mini game is rewarded with the non-playable characters clue. *Buzz! Junior Monster Rumble* for the Sony Playstation 2 is an example of a mini game collection that has several mini games, but adds them to a context where the different mini games themselves fit into the environment of the virtual game world, in the game's case a castle inhabited by monsters. The *Buzz* game incorporates several well known mini games, and some new ones, and changes them to fit into the new context of a monster world, and since it is a game targeted toward a younger audience it inhabits a lighter and more humorous atmosphere throughout the game. A similar approach could be used in this concept by incorporating general and new mini games into a setting where a murder must be solved. The added element of clues gained through winning the mini games and the overall goal of solving the murder mystery makes the concept more unique than simply a mini game collection.

As with the quiz and adventure style, the mini game style could either have the mini games separate for each player, or have all of the player fight against each other in the mini games, or a mix of both. The players could for example walk through the virtual mansion in separate paths and encounter a new mini game in each entered room, but at certain points in the mansion the paths of the players intertwine allowing the mini game in the particular room to be fought amongst all of the players.

8.4 Quiz game

The core idea of this concept is to create a multiplayer quiz game for mobile devices, where the players players get questions and answers them on their mobile devices. The game should support several players where the players are given the same questions, which depending on the game mode, could have several answer alternatives, and the players should be given points in order to compete against each other. The points could be divided according to several factors, for instance depending on the time the players used to answer the questions, where the players that answered correctly first gets the most points. The player with the most points after the game has finished, is the winner. The questions could stem from one of several categories, with for instance questions from music, sports, history, geography, etc. The game could either keep the questions in each game round separate, i.e. by only asking questions from a certain category, or it could ask questions from different categories. The questions would however not necessarily be traditional questions where each question is about remembering facts, but the game could also have small tasks or brain teasers, where the player must solve small problems. The game could still offer the players answer alternatives though, and the teasers could be mathematical problems, logical puzzles, and other small tasks.

The questions are the crucial part of the quiz game, but they could be presented to the players in different ways, and the game could support several modes where each mode alters the gameplay slightly to create variation in the game. The modes could differ by several factors, in for instance how points are divided among the players with correct answers, whether the players receive negative points for answering falsely, whether the time and order of the correct answers should factor into the point giving, and whether the modes should change the gameplay more substantially. The players could then before each game round decide which game modes should be used in the game round, and how many questions there should be in each mode. The modes could be played in succession, and the points for each player would be accumulated and the points of all the players would be compared to each other at the end of the round, and the game could have high score lists that keeps track of the points. The *Buzz!* series for the Sony Playstation 2, a game which utilizes special buzzer controllers to answer trivia questions, offers several different game modes the player can choose from, and some of these game modes would fit the mobile platform. Below some of the modes in the *Buzz* games, and how they would fit, or could be fitted to a mobile quiz game:

The Point Builder is a standard mode where the players are presented with a question with answer alternatives and a time limit of say 20 seconds for each question, and all players are awarded the same score if they answer correctly, with no penalty for answering incorrectly. This mode would fit well as a default mode, and fits the mobile platform nicely.

Fastest Finger is similar to the point builder mode, in that the players are given a question with answer alternatives, and every player with the correct answer gets points, but now time is a factor as well, and the order in which the players answered the questions would factor into how many points the players would get. It would be more difficult to implement due to network latency in mobile networks, but the problem could be worked around.

Spin is a mode where the players are presented with a roulette wheel with several different genres of questions, for instance different sports, and the players in turn start an arrow that spins around the different genres and the player tries to stop the arrow at his(her) preferred genre. The gist of the idea is to have several genre of questions and have each player in turns decide the genre for the next question. This could be well suited for the mobile platform, depending on the selection technique is implemented, which would not have be made in the roulette spin way as done in *Buzz*. The questions themselves could be presented and scored in the same way as for instance point builder, but the main point behind the mode is the selection of different genre of questions, and how this is presented to the players.

Expert is a mode where each player select his(her) preferable genre among several, and has to answer as many questions correctly as possible within a time limit, where each correct answer raises a score bar, and each incorrect answer lowers the score bar. After every player has finished answering questions in his(her) category, the points are divided according to the score bar, where the highest scoring player gets the most points. The mode would be well suited on a mobile platform, as it has turn based mechanics for presenting the questions.

Point Stealer is similar to the fastest finger mode in that here time is a factor. The first player to press the button will get the chance to answer the question, and if the player answers correctly, the player then is able to steal points from another player, where the winning player chooses which player to steal from. As with the fastest finger mode, timing issues and network latency might be a problem here, but as a social mode and mobile mode it fits well.

Risk adds gambling to the questions. Before each question the players are shown the category of the question, and are asked to bet points among several options. If the players then answer the question correctly they gain the amount of points they bet, and if they answer incorrectly they loose the amount of points they bet. The mode is well suited for mobile phones, since time is not a crucial factor as with the fastest finger mode. The mode could also be expanded to allow the players to not only bet on themselves, but also bet on the other players.

Pass the Bomb is a mode where one player gets a question which acts as a bomb. The player must then answer the question given to (her)him before a timer goes off. If the timer goes off the player loses points, but if the player answer correctly the bomb is passed on to the next player, in a round robin fashion, where the round continues until the bomb is blown up. On a mobile phone the game could use vibration to notify the player about the bomb, for instance by vibrating when the player gets the bomb, and increase its vibration until finally the bomb explodes, which causes the vibration to reach its maximum, which could be followed by an animated explosion in the game and sound effects. The mode is well suited for mobile devices and using the different functionality of the mobile devices could create a more immersive experience for the players.

Last man standing is a mode where all of the players are given the same question, where each question has answer alternatives. If a player answer incorrectly on a question, the player is out of the round, but if the player answers the question correctly, the players continues to the next round where a new question is given to the remaining players. The mode continues until there is only one player left. The mode is well suited for mobile devices, but if there are few players the mode made be less fun than if there are more players. A solution to this problem could be to introduce lives to the mode, where each player has for instance three lives, and for each incorrect answer the player looses one life. When the player has lost all of his(her) lives, the player is out.

Quiz master is a mode where one of the player is the quiz master, and is the one that is deciding which questions should be asked, and gives the questions to the other players. The quiz master does not get points for answering questions, but the quiz master role can be shifted among the players. The quiz master could also be given the ability to decide how many points the players should get for answering the questions correctly, as well as deciding the mode for each round or question, as the quiz master mode is based around the questions, and not on which mode they are presented to the players. The questions given to the players in this mode, could either be those of the standard game, or the game could allow the quiz master to give the players user created questions. These questions would be created by the users. The game should have a question creator editor in which the players could create questions, and possibly upload them to an online server where other players could download them. Players could then share questions with each other, with the goal of creating a community around the game, possibly by supporting a web page with user created questions, high score lists, and forums. The user created questioned could also be extended to fit the other modes of the game, so that the players could decide whether to use the standard questions within the game, or user created ones.

In the modes and gameplay mechanics suggested above, the main component is the questions. The difference between the modes lie mainly in the presentation of the questions to the player, as well as whether the time aspect ties into the game mode, and the different question categories the game can draw its questions from. This structure, with different modes changing the presentation of the questions to the players, are well suited for short play sessions which is important for certain mobile gamers, but it may lack the potential of engaging the players to play for longer play sessions, and the only goal for the players is that of getting more points than the other players, aside from the psychological effect of showing ones prowess and wits to the other players. By adding additional gameplay mechanics on top of the core gameplay, i.e. the questions, the game can be more immersive for the players, as well as give more defined goals to the players.

There are several gameplay mechanics which could suit being structured around question asking and answering, such as implementing a classic board game structure where the game world consists of a game map which is divided into several separate sectors which are interconnected in a way as to form a pattern, e.g. a circular ring of sectors with an center section similar to the game board in the game *Trivial Pursuit*. The players, which are represented on the game map by for instance in-game avatars, must then move around the different sectors, and the movement could for instance be determined by a server controlled dice throw where the number on the dice would decide how many sectors the players could move. After having moved to the location decided by the dice, the players are then asked questions, and if they answer correctly, they then gain an advantage in the game. The advantage would depend on the game implemented, but it could for instance be an additional move by dice throw, that the players would be given points for answering correctly, or as in the trivia board game *Trivial Pursuit* that the players be given an object vital to the game progression. The players could also be divided into different teams, where each team moves as one avatar on the game map, and the players on each team are asked the same questions and must then together come to the solution of the question. The game could give the players various features to help with communication between team members, such as text based chatting, and giving the players a majority vote system in which each player on the team votes for the most probable answer to the question in that players mind. The game would be played turn-based, where in each turn the players or the player teams would move according to a server controlled dice throw. The goal of the game would then either be to traverse all of the sectors in the game map before any of the other players or player teams, or as discussed above, gather all of the vital game objects.

The approach taken above could also be widened to include elements from other genres than board games and quiz, such as for instance the roleplaying genre. In such an approach the player would be in control of an avatar which they moved around the game map, which could be divided into interconnected sectors such as described above. The avatar would have health and experience points associated with it, and the player would gain experience points by going into battle with other entities on the game map, or with other players. The battles would be quiz based where the players involved in the battle would compete against each other by answering questions, perhaps by using one of the modes discussed above. The health of the player would decide the outcome of the battle, as wrong answers could for instance lower the health of the player. The game could also have levels, where the experience points gained from winning battles would be used to give additional health points, and add special abilities, such as for instance the ability to reduce answer alternatives in questions asked to oneself, or increase the answer alternatives of the question asked to the combatant in a battle. The goal of the game could be one of several things, such as gaining the most experience points within a time limit, gaining the most in-game objects within a round, where the game map would incorporate several in-game objects which would be won by entering the sector the object resided in and winning against the opponent residing in the sector.

A Quiz game would suit the mobile platform well, since it does not require complex graphics, and the network data needed to be send would be relatively low. The mobile phone could also add features to the game, such as vibration mentioned above. A problem could be that of deciding who answered the question the fastest, and keep track of time events. This could be avoided by designing the game modes in such a way that time and synchronization is a non issue, but that could limit the game modes available.

8.5 Concept summary

This section compares the concepts described in this chapter, and gives a indication of the most prominent concept to use as a prototype in the project.

The main focus of this project is social gaming, and thus the social aspect of the concepts, and how well they support social interaction between the players is important, where social interaction range from cooperative gameplay, to in-game text message possibilities, and even as far as in-game character gestures can be considered social interaction, as they can be used to convey the feelings and emotions of the player.

Competitiveness is also important for games, where it gives players an incentive to play the game, however competitiveness can in some cases hinder social and cooperative gameplay. When competitiveness is the focus of a game, the cooperative and social experience for the players may be less of a focus, and certain elements of competitive games can hinder cooperability between players, as for instance in MMORPG games where competitive player-versus-player combat can be a nuisance to players who simply want to experience the game world in a social player group without being attacked by other players. However, the competitiveness can be incorporated as to support cooperability and social gameplay. Consider for example team-based games, where the players are grouped into teams who then compete against each other in some form. In such a situation more often than not the team that is best organized and is able to work together against the opposing team has an advantage against a team of players who do not know how to work together as a group. In such a situation the game supports and rewards and social interaction between indirectly by not forcing cooperative gameplay on the players, but the players that are able to work together as a team will have an advantage over the others. When placing such gameplay into a situation where there are other players around the team-based gameplay not participating in the competition between the teams, as described in the MMORPG example above, the competitive nature of the gameplay can be intrusive to the players not participating in it. Therefore competitiveness in this project is not as important as social gameplay and cooperativeness.

Being able to implement the concept as a prototype game is also important for the project, and thus the implementation complexity is also important when considering the different concepts. The implementation complexity for this project will depend on factors such as network latency and bandwidth limitations, general problems relating to the mobile environment, and the complexity of the concept and how difficult the concept is to realize into a game when considering development time and project size. The implementation complexity is then equally important as the social aspect of the concept, as a concept with extreme social features, but with highly complex implementation is unlikely to make it into a working prototype.

8.5.1 Concept Comparison

Below the different concepts described in this chapter are discussed in respect to the three problem areas described above; social gaming, competitiveness, and implementation complexity.

Quiz game The quiz game concept is well suited for social gameplay, as is evident in the *Buzz* game series for the Sony Playstation 2, and the concept can be implemented in several different to support different aspects of social interactivity. The different modes suggested in the concept description shows some of the potential to alter the gameplay of the concept, and the question answering game mechanic can be used as part of a larger concept which uses the question mechanic as a part of the gameplay, for instance as a trivia board game, or as suggested in the concept description, as a quiz with role-playing elements incorporated. The concept is also well suited for competitiveness, as the concept can offer social gameplay in a competitive setting. The players competes against each other in answering questions, and for instance in a situation where all of the players are collocated in the same room, the experience become more social as the players can interact outside of the game and see the reactions of the other players after in-game events. The complexity of the concept will differ based on how many different modes is supported, and whether the concept utilizes for instance the RPG-quiz approach, which would require more work. The network latency and bandwidth are not that big of a problem for the concept, as the data needed to be transfered between the players is relatively low, however when time is used as a defining factor in the gameplay mode, synchronization issues may arise which must be solved.

Mystery solving game As described in the concept description, the concept can be implemented in several different ways, and some of the ways are more tailored toward competitiveness, while certain ways of implementing it focuses more on cooperability. However the competitiveness is not intrusive to the point of not allowing social interaction between the players, and it can thus can give the players an social experience with friendly competitiveness between the players. The specific concept that enforce cooperative problem solving between the players force social interaction upon the players. and may feel intrusive to certain players, in a different way than that of imposing competitive gameplay upon the players, but for the context of this project, the social behavior between the players in such a situation would be interesting. The implementation complexity however would be relatively great for any of the different ideas of the concept, where the gameplay itself could potentially be to difficult to implement within the time space of the project. The network bandwidth and latency needn't be big problems because of the non real-time gameplay of the concept, while when implemented using time constraints, as discussed in the quiz concept case, synchronization might become an issue.

Platform-puzzler As with the mystery solving concept, the platform-puzzler concept can be implemented in several ways, however the completely cooperatively gameplay direction described in the concept description is the most interesting in the context of this project, as it has social gameplay as it main focus. The concept gives rise to questions such as the difference between playing cooperative games in a collocated or non-collocated situation from the players perspective. Competitiveness is not the main focus in the completely cooperatively direction, but competitive elements can be added to the game, such as Internet leader boards. The implementation complexity need not be big for the concept, but the complexity of the concept lies in the gameplay mechanics used in the game, i.e. the different puzzle solving tools given to the player. Implementing a portal gun system as seen in *Portal*, created by Valve as a part of *The Orange Box*[3], would lead to a more

[3]See http://www.whatistheorangebox.com/ for the game description of *The Orange Box*

Concept	**Social**	**Competitive**	**Imp**	**Sum**
Quiz	5	3	3	11
Mystery solving	3	3	1	7
Platform-puzzler	5	3	5	13
Tower defense	1	5	3	9
Tanks	1	5	3	9

Table 8.1: Table showing a comparison between the different concepts in this chapter quantified.

complex game than implementing a simple crate pushing game. Network latency and bandwidth need not be problems either, but again this would depend on the puzzle solving game mechanics implemented in the game.

Tower defense The tower defense concept is at its core a competitive turn-based concept, with an added mode with cooperative play. Both modes offer a social experience for the player, where the cooperative mode may be more focused on the social part. The implementation complexity of the concept vary on the gameplay elements added, but can develop to be relatively complex the more strategic elements and mechanics are added to the concept.

Tanks The main focus with the tanks concept, as with the tower defense is competitive gameplay, where two or more players control several units that fight each other for victory on the battlefield. The social aspect of the concept lie in the competition between the players, and the in-game interaction between the players. The implementation complexity of the game is relatively moderate, with bandwidth and latency not being big issues because of the turn-based combat system planned in the concept, while more elements, units and strategic gameplay mechanics add to the complexity of the concept.

8.5.2 Concept choice

Table 8.1 shows a quantified comparison of the different concepts described in this chapter. The different concepts are compared according to the competitiveness, the social aspects, and the implementation complexity. Each of the three differentiators are given a weight and summed up, where the highest final sum may be best suited for this project. The weights consist of a number which is either 1, 3, or 5, where 5 is more suited for this project, and 1 is not well suited for this project, i.e. regardless of category 5 is the most positive weight, and 1 is the most negative weight. The final sum is simply the three numbers combined into a sum. The weights are based on the discussion above, as well as the intuition of the project members. The Table shows that the platform-puzzler may be the best suited concept for this project, at least when using this weight system. The project members also find the concept to be interesting, and that it has possibilities to be used to investigate interesting questions regarding social gaming. Therefore this concept is chosen to be implemented as a prototype for the project.

Chapter 9

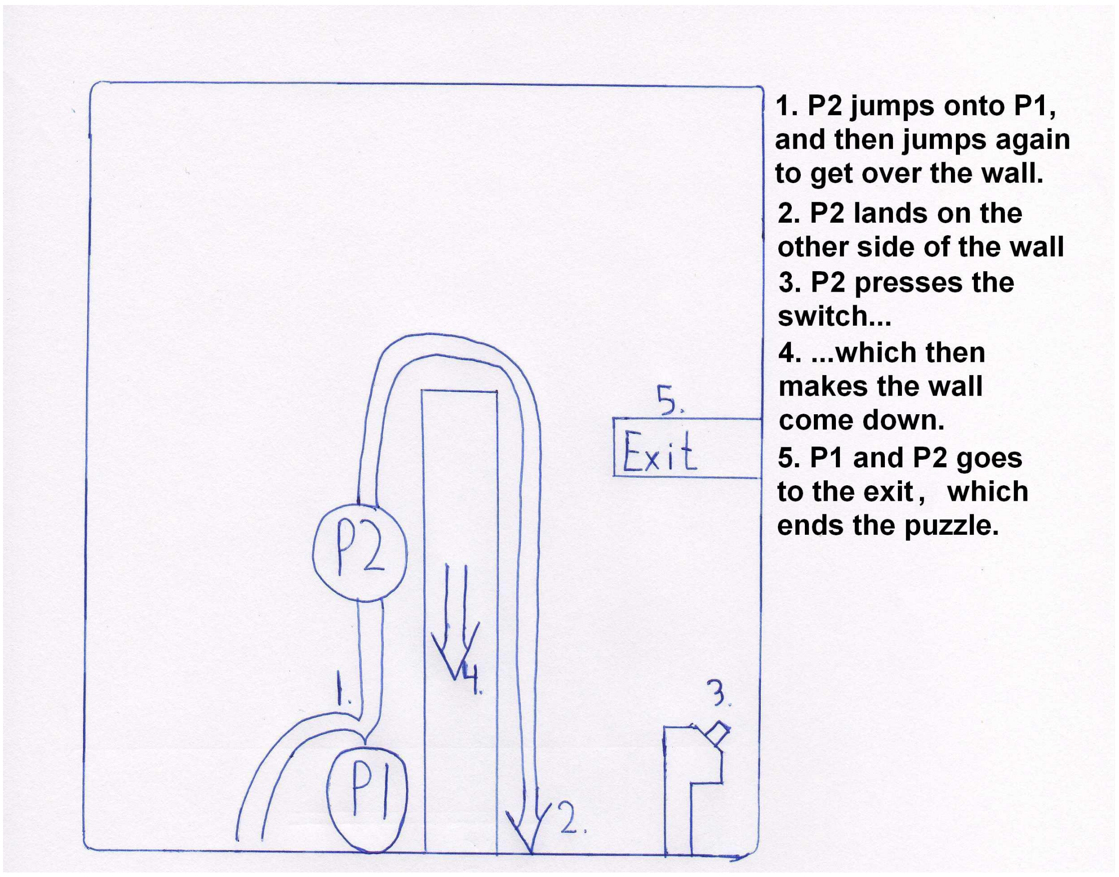
Prototype: Platform-puzzler

This chapter describes the prototype game developed during the project. The prototype game developed during the project is based on the *Platform-puzzler* concept described in Section 8.2.1. This chapter first describes the game, its setting and the game rules, and then describes the requirements, the game design, and the implementation of the game, before finally describing the testing of the prototype.

9.1 Game concept design

Platform-puzzler is a 2D side scrolling networked multiplayer platform game with puzzle elements where two or more players solve puzzles together in the same level. The prototype game focuses on the more cooperatively concept found in Section 8.2.1 in which the players are forced to work together in order to solve the puzzles, i.e. the puzzles are designed and implemented in such a way that they require at least two players in order to be solved. Each player have control over an in-game character that moves through the game. The game world is divided into several levels that the players go through sequentially. Each level consist of one or more puzzles that are based around environmental challenges, such as overcoming obstacles, and opening locked doors, and each level has an exit that leads to the next puzzle level, and all of the players must go through the exit for each level to end.

Figure 9.1, 9.2, 9.3, and 9.4 show the basic mechanics the prototype should support and how some of the puzzles could be created. These figures act as scenarios that show the basics of the game and what the prototype game should support and implement. Figure 9.1 shows the first puzzle scenario with the most basic functionality. The Figure shows the players, here named P1 and P2, as simplistic characters which are able to move forwards and backwards as well as being able to jump in the 2D side scrolling environment. The puzzle in the scenario revolve around the wall, which is to high for a player to jump over without help, and a switch that lowers the wall. To get to the switch the players must help each by having one of the players jump on top of the other in order to be able to jump high enough to be able to get over the wall. When the player gets over



1. P2 jumps onto P1, and then jumps again to get over the wall.
2. P2 lands on the other side of the wall
3. P2 presses the switch...
4. ...which then makes the wall come down.
5. P1 and P2 goes to the exit, which ends the puzzle.

Figure 9.1: Puzzle scenario for the prototype

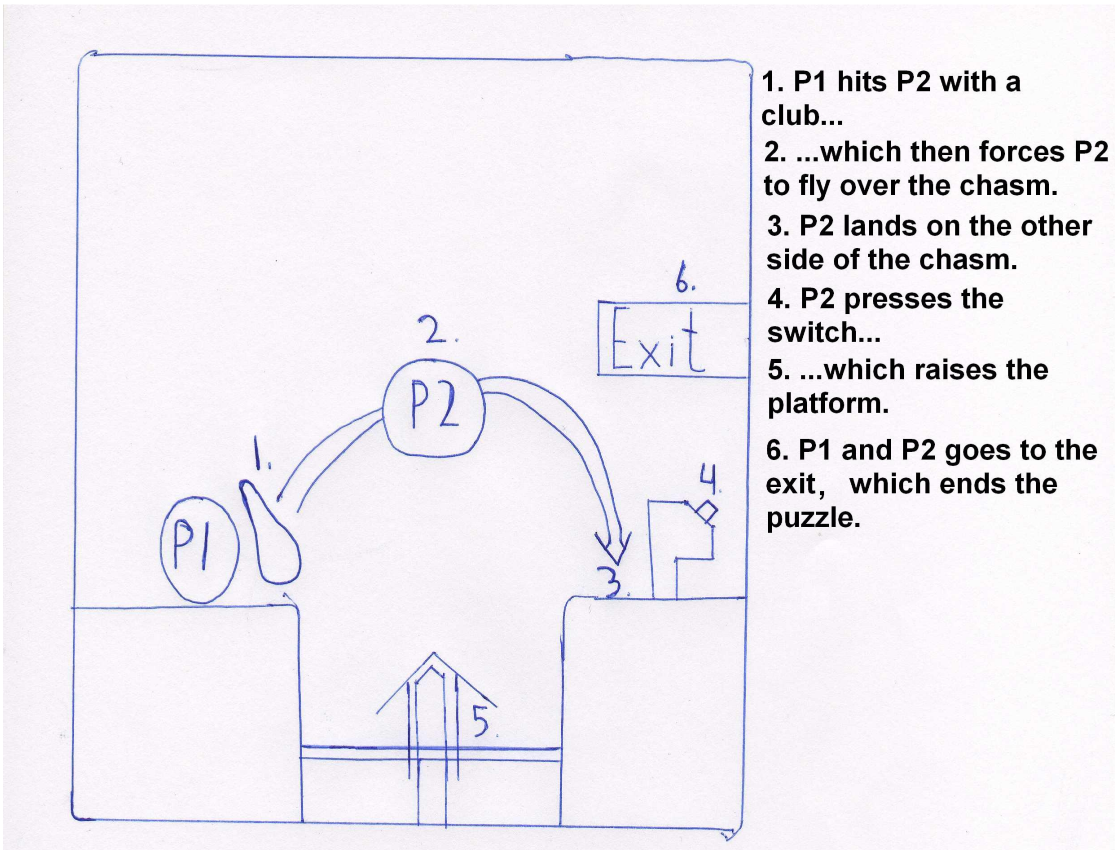
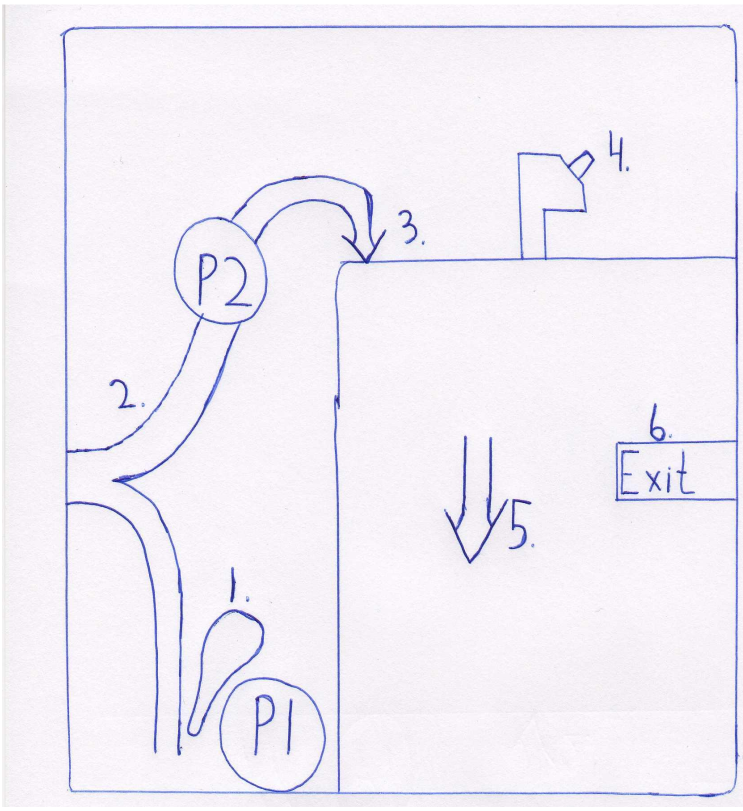

1. P1 hits P2 with a club...
2. ...which then forces P2 to fly over the chasm.
3. P2 lands on the other side of the chasm.
4. P2 presses the switch...
5. ...which raises the platform.
6. P1 and P2 goes to the exit, which ends the puzzle.
P1
P2
1.
2.
3.
4.
5.
6.
Exit

Figure 9.2: Puzzle scenario showing movable platforms and player throwing



1. P1 hits P2 with a club, which forces P2 to fly into the wall.

2. P2 bounces back from the wall...

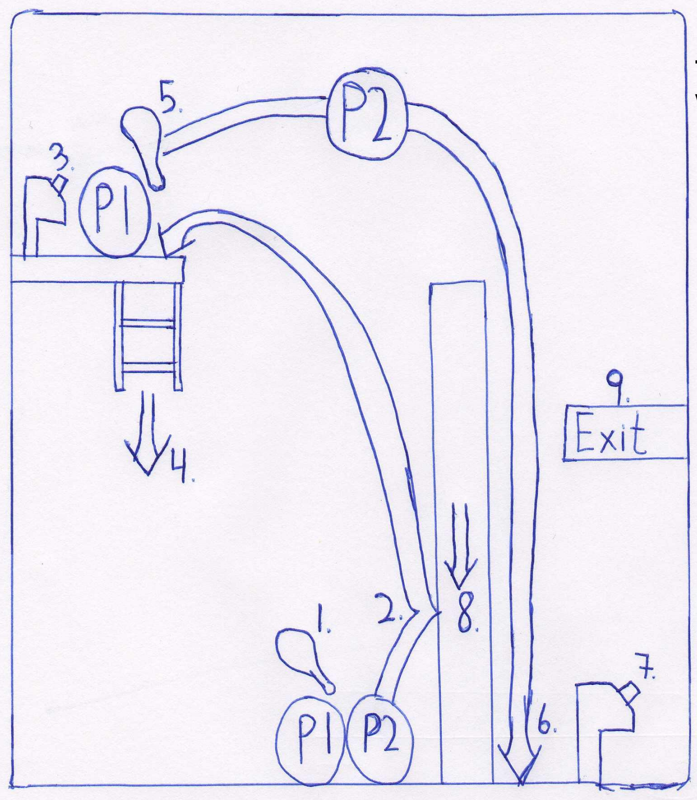
3. ...and lands on top of the platform.

4. P2 presses the switch...

5. ...which lower the entire platform, which in turn makes the exit reachable.

6. P1 and P2 goes to the exit, which ends the puzzle.

Figure 9.3: Puzzle scenario showing wall bouncing



1. P1 hits P2, which forces P2 to fly into the wall and...
2. ...bounces back out and flies onto and lands on the platform.
3. P2 presses the switch which...
4. ...lowers the ladder so that P1 can climb to the platform.
5. P1 hits P2 with the club again, which forces P2 to fly over the wall.
6. P2 lands on the other side of the wall and...
7. ...presses the second switch which...
8. ...lowers the wall.
9. P1 and P2 go to the exit, which ends the puzzle.

Figure 9.4: Puzzle scenario showing a more complex puzzle

the wall, that player then presses the switch which lowers the wall such that the other player can get past the wall as well. Then as the last part of the puzzle the players cross the exit together. This implies several game mechanics, where the first one being that the player has control of in-game character that is able to move in several directions, as well as having additional abilities, where in this scenario jumping and jumping on top of another player are crucial to the puzzle.

Another implication of the scenario is that the environmental puzzles consist of, apart from the player abilities, in-game objects that can have properties. The objects in this scenario firstly show that the objects can be of any shape, and that they can be static, or be able to move, such as wall. The switch also implies that the game should have a trigger system, where a player can trigger an event, which leads to an action. The player can trigger the action either directly by pressing a button when near a in-game switch, such as the switch panel in the scenario, or more indirectly by walking over an trigger that resides within a specific area of the game map. The trigger/event mechanic then becomes a object property, since an object can be assigned a trigger that creates an event when invoked. This event in turn leads to an action, which could either alter the environment, such as lowering the wall in the scenario, or alter the properties of objects, such as for instance making a object climbable.

Figure 9.2 shows a new scenario that introduces a new player ability, and thus a new game mechanic. The puzzle itself involves a chasm that is to big for the players to able to jump over, even if one player stands on top of the other. On the other side of the chasm stands a switch that raises a platform such that the players are able to get over the chasm, and go to the exit. The new mechanic allows a player to force another player to fly through the air, in the scenario represented by a club that is controlled by a player so that the player can hit the other player with the club, and the other player then flies over the chasm enabling the switch to be pressed. The mechanic itself could be presented as something different than a club, but it should have a visual in-game representation to make it more understandable.

Figure 9.3 shows another scenario that utilizes the club mechanic discussed above, and adds an additional mechanic that work together with club mechanic to create new puzzles. The puzzle itself in the scenario involves a high platform that has a switch on top of it, but the platform is to high for a player to jump on top to, as well as being to high also if a player stands on top of another player. The switch lowers the platform such that the players are able to go to the exit. In order to reach the switch, the players are forced to use the new mechanic which allows a player to bounce off of walls and objects after being hit by the club mechanic. In the game a player should bounce off of walls or objects after being hit by the club mechanic, but it should be restricted to vertical faces such that a player does not bounce off the ground.

Figure 9.4 shows a scenario that has two main elements to it. The puzzle utilizes the mechanics introduced above and introduces a new property to objects, which is the property that allows player to move vertically on objects, which in the scenario is represented by a ladder. The puzzle involves a wall that is to high to reach by jumping, jumpoing from another player, or by using the club mechanic on the left back wall. The solution to this is to use the club mechanic to launch a player into the wall, and the player then bounces on top of the platform above. On top of the platform is a switch that lowers the ladder such that the player below can climb the ladder in order to reach the platform as well. The final part of the puzzle involves using the club mechanic again which then launches a player over the wall, which enables that player to press the switch that lowers the wall such that all the players can cross the wall and reach the exit.

The game mechanics discussed above are the basic building blocks of the puzzles in the game, and as the fourth scenario shows, these mechanics can be utilized together in order to create new puzzles which involve more than one core mechanic in order to be solved. Another idea not specifically mentioned in the scenarios is to have the players start in different locations in the puzzle level, but the players must still work together in order to solve the over all puzzle in the level such that they get to the exit together. This could be done by for instance having switches that open doors or raises platforms not for the player that invokes the switch, but for another player in another area in the level, which in turn may have a switch that opens a new path for the first player.

9.2 Requirements

This section will look at the various requirements for the prototype. The requirements are created from the description of the prototype game in this chapter, as well as from the description of the *Platform-puzzler* concept found in Section 8.2.1. The focus of the prototype is to get a working prototype game of the *Platform-puzzler* concept, using JavaME and tailored toward mobile phones, that can be used further in the project, so the functional requirements are the most important ones, while non-functional requirements such as security, performance and modifiability are not as important, but may still contribute to the system. All requirements are numbered using "FR-#" for functional requirements, where FR stands for functional requirements, and # is the number of the requirement, while the non-functional requirements are numbered using "NFR-#", where NFR stands for non-functional requirements, and # is the number of the requirement.

9.2.1 Functional requirements

The functional requirements describe the functionality the prototype game must offer to the player.

FR-1 - Player In the game each player should controll an in-game character by using keypad buttons.

FR-2 - Character movement The character controlled by the player should be able to move forward, backward, and be able to jump, thus requiring the acquisition of 3 keypad buttons that support these player abilities.

FR-3 - Action triggers The prototype should incorporate a trigger/event system where action triggers cause an event that leads to an action. The action triggers should be invoked by players by pressing a button, thus requiring a keypad button to support this action.

FR-4 - Events An event should be trigger by an action trigger and the event should be sent to the receiver of the event which can then act on the event.

FR-5 - Actions An action should be triggered by an event and should either change the properties of in-game objects, or alter the state of the prototype game.

FR-6 - Objects The in-game world should be populated by in-game objects. The objects should be represented in the game by graphical artifacts or images.

FR-7 - Object properties A in-game object should have one or more properties that affect the behavior of that object, and the interaction between the player character and the object.

FR-8 - Object property, player collision An object should either allow a player character to move through the object, or not.

FR-9 - Object property, trigger An object should be able to have an action trigger associated with. The player triggers the action triggers, and the object creates an event appropriate for the wanted action. The action trigger could either be triggered by a player key press, or by setting up an area that triggers an event when a player enters it.

FR-10 - Object property, event/action An object should be able to receive an event, and instigate the proper action based on the type of event.

FR-11 - Object property, vertical character movement An object should either allow an player to move vertically on the object, e.g. similarly to walking a ladder, or not. Vertical player movement would require tow keypad buttons representing vertical ascending and descending movements.

FR-12 - Object property, static An object should either be static, or be able to move along a predetermined path. The movement could either be continuous, or be triggered by an event.

FR-13 - Multiplayer The prototype game should not have any single player component, thus requiring multiple players.

FR-14 - Game world The game world should be presented to the player as a 2D side scrolling world. The player should be able to control his(her) character in the world and solve puzzles along with other players. The game world should be divided into several separate, but interconnected levels in which each level consist of one or more puzzles. The completion of all the puzzles in a level should lead to the exposure of the level exit. A level exit should lead to either a new level, or when there are no more available levels, an screen informing the player that there are no more puzzle levels available in the current game session.

FR-15 - Puzzles The puzzles should consist of environmental puzzles and be created using one or several objects. The solution to the puzzles should require the players using character abilities, and utilizing the object properties.

FR-16 - Character abilities A player character should have one or more abilities that should be utilized when solving puzzles.

FR-17 - Character ability, club mechanic A player should be able to use a club mechanic in which the player uses a club that has the ability to force other players to fly a predetermined distance. The club should be represented in the game via either graphical artifacts, or as an image, and the player should be able to invoke the club mechanic by using a keypad button.

FR-18 - Character ability, wall bouncing A player that is affected by the club mechanic should be able to bounce off of objects. This should however only apply to vertical player-object collisions, e.g. after flying into a wall.

FR-19 - Character ability, player-on-player jumping A player should be able to jump on top of another player in order to gain an elevated vantage point.

9.2.2 Non-functional requirements

These requirements describe requirements that describe qualities about the prototype not necessarily related to the functionality offered by the prototype game.

Network

These requirements relate to issues regarding network.

NFR-1 - Client server The prototype should use a client server architecture.

NFR-2 - Multiplayer player amount The prototype game server should allow for at least two simultaneous player connections while running, thus the puzzles should be designed and implemented in a way that require two players in order to solve them.

NFR-3 - Client server disputes In circumstances where latency or bandwidth cause state data to differ from client and the server, the server's state data is to be trusted.

NFR-4 - Client disconnection If a player disconnects from a game session, the remaining players in the session should be notified about the disconnection as well as be given a warning that the game session is closing, after which the game session must close. The game session must close because the puzzles require two players to solve them, and in the situation where there is only one player left in the game session, that player can not continue solving the puzzles since (s)he would require assistance from another player in order to continue[1].

Usability

These requirements relate to the usability of the prototype, i.e. how easy it is for the players to use and understand the game.

NFR-5 - Puzzle understanding A player team in a game session should be able to understand how a puzzle works, and thus be able to solve it, in less than 5 minutes, with an additional 5 minutes added if the players are not co-located in the same room.

NFR-6 - Puzzle completion After having understood a puzzle, the players should be able to solve the puzzle and reach the exit in less than 10 minutes, with an additional 5 minutes added if the players are not co-located in the same room. This applies to all of the puzzles such that no puzzle should take longer than 10 minutes to solve.

NFR-7 - Game understanding A player should be able to join a multiplayer game within 5 minutes after having started the game for the first time, not counting in network delay and shortage of players to join the game. The player should also be able to understand all of the controls in the game and be able to understand and move the in-game character within 5 minutes of starting his(her) first multiplayer game session.

9.3 Game Design & Implementation

The prototype is, as described previously in this chapter, based on the *Platform-puzzler* concept found in Section 8.2.1, as well as the description of the prototype discussed earlier in this chapter. The prototype game is a multiplayer game where two or more players control an in-game avatar, which in the prototype game is represented as a red ball. The game consists of two main components; the client application and the server application[2]. The client application is developed in the Java Platform, Micro Edition(Java ME) utilizing MIDP 2.0 and CLDC 1.1. Each player has a separate client

[1]A solution to the problem could be either to allow disconnections and reconnections such that players could join on-going game sessions, or in the case where there are two or more players remaining after a disconnection, the game session could continue as normal.

[2]The source code for both the client and the server along with the binaries for both are located in the zip file attachment of this report

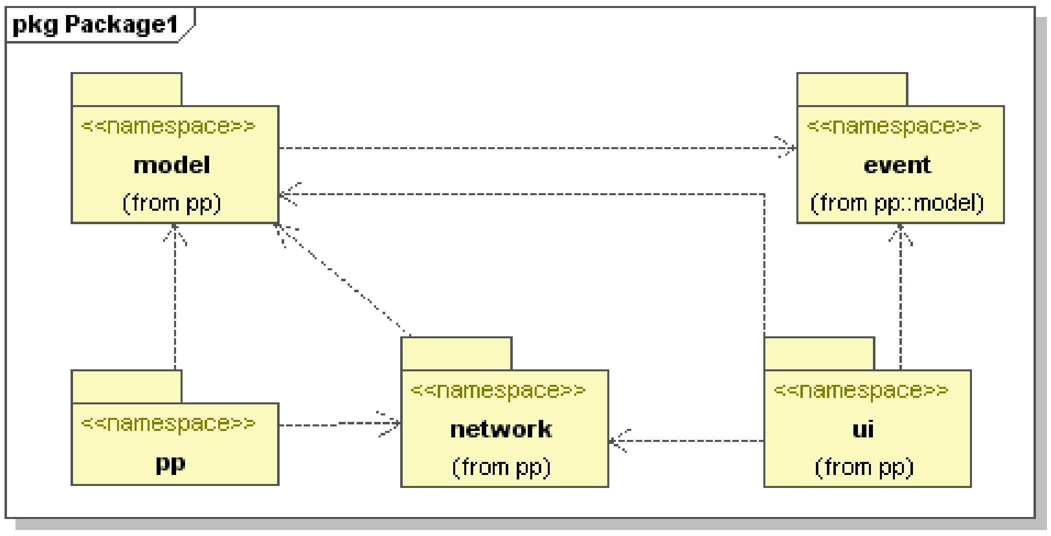
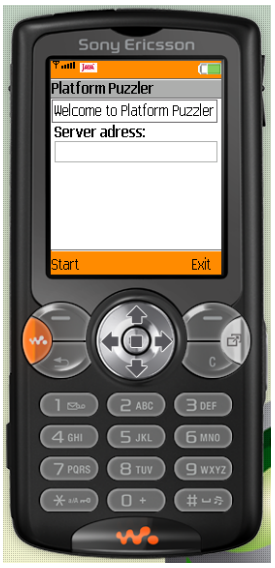


Figure 9.5: Package diagram of the client application

application running on his(her) mobile device, and the client and server communicates through a TCP/IP connection, where for the mobile device this will typically be wireless communication where the mobile device of the player dictates bandwidth and latency. The server is created in Java SE 1.5, and for each game session there are one server that support two or more players. The server can run on any platform that has a network connection and support Java SE 1.5.

9.3.1 Client

The client part of the prototype is, as discussed above, a separate application created in Java ME, and it consist of the packages shown in Figure 9.5. The top level package of the package hierarchy of the client application is the **pp** package. This package contains the MIDlet class, i.e. the class that is first started when the application is run, as well as the other classes in the client application. The other packages in this system are **pp.model**, **pp.model.event**, **pp.ui**, and **pp.network**. The **pp.model** package handles the classes in the client that incorporates classes representing the player, and player actions, the levels in the game world and generating these levels, as well as the objects in these levels and their properties. **pp.model.event** is a sub package of **pp.model** and it has classes that are involved in the event trigger system used in the game. The package **pp.ui** on the other hand contains the classes that handle the user interfaces of the client application, i.e. how the game is represented to the users. The final package is the **pp.network** package, and it contains classes that handle communication with the server. The client communicates with the server via TCP/IP sockets, and the package contains classes for sending and receiving messages, as well as classes for defining and handling incoming and outgoing messages.



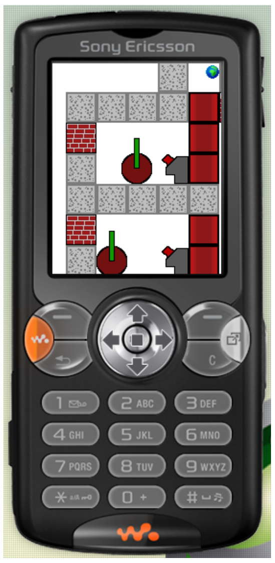
Platform Puzzler

Welcome to Platform Puzzler

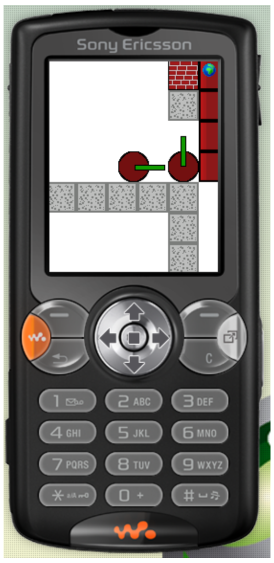
Server adress:

Start Exit

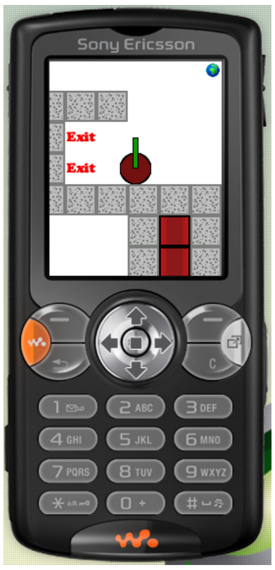
(a) The start screen of the application, where the player input the server IP address



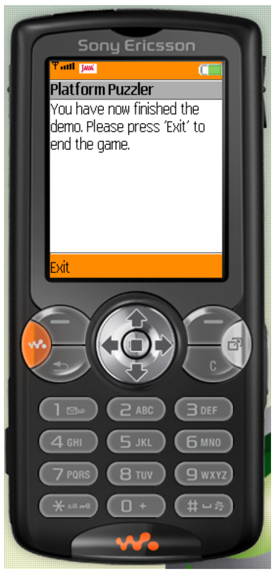
(b) Two players who are at their starting locations in the current level



(c) One of the players try to apply their club to the other player



(d) Here the exit of the current level is shown



Platform Puzzler

You have now finished the demo. Please press 'Exit' to end the game.

Exit

(e) This is the screen that the players face after having finished the prototype levels

Figure 9.6: Various screenshots from the prototype game client running on Sony Ericsson Wireless Toolkit emulator

Main class,the game MIDlet

The main class of the client application is the **PlatformPuzzler** class, which lie in the **pp** package. Figure 9.7 shows the class diagram of the **PlatformPuzzler** class. **PlatformPuzzler** inherits from the **MIDlet** class of MIDP and acts as the starting screen for the player. Figure 9.6(a) shows this class as it is displayed in the emulator used to test the application. The graphical user interface created by the class informs the player that (s)he is playing **Platform-puzzler**, as well as displaying one text box for user input, and two command buttons. The text box is used by the player to enter the IP-address of the server. The two commands are "Exit" and "Start" commands. The "Exit" command simply exits the game, returning to regular interface of the platform running the client application, while the "Start" command attempts to connect to the server using the IP-address entered by the user. If the IP-address is not valid, or no IP-address is entered, nothing will happen and the user must enter a valid password, but if the IP-address is valid, the **PlatformPuzzler** class sends a message to the server, requesting that a player should join the server session. Upon receiving an acknowledgment of the request message from the server, the **PlatformPuzzler** class creates a new **GameWindow** object, which contain the main game loop, as well as a new **Player** object that handles player information.

Model

Figure 9.8 shows the classes in the **pp.model** package. The package contains, as discussed briefly previously, objects that contain information about and handle the actions of the levels of the game world, the generating of these levels, the in-game objects that populate the levels, as well as a player class.

Vector2D is a class that represents simple 2 dimensional vectors, i.e. it is a class containing two variables(which are integers in this class) **x** and **y**. The class also implement some elementary vector operations such as vector addition, the scalar product(or dot product), and multiplication of the vector with a scalar. This class is used in the prototype to represent positions of the player and level objects, as well as being used as speed vectors indicating the speed of the player. The positions of the player and the objects are global such that position **(x,y)** is located in the same point in the game world for every client connected to that game world. The speed vector of the player contains the relative speed of the character controlled by the player, as the speed itself depends of the time between each main game loop execution, as well as being pulled down by gravity.

The **Player** class represents the user of the prototype game, as well as other clients connected to the same game server. Each **Player** object has two vectors, as discussed above, where one vector is the global position of the player while the other vector is the speed of the player, both in the x- and y-direction. The player class has several methods that handles these vectors, such as updating the position of the player. In the graphical user interface each player is represented as red ball. This red ball is represented in the

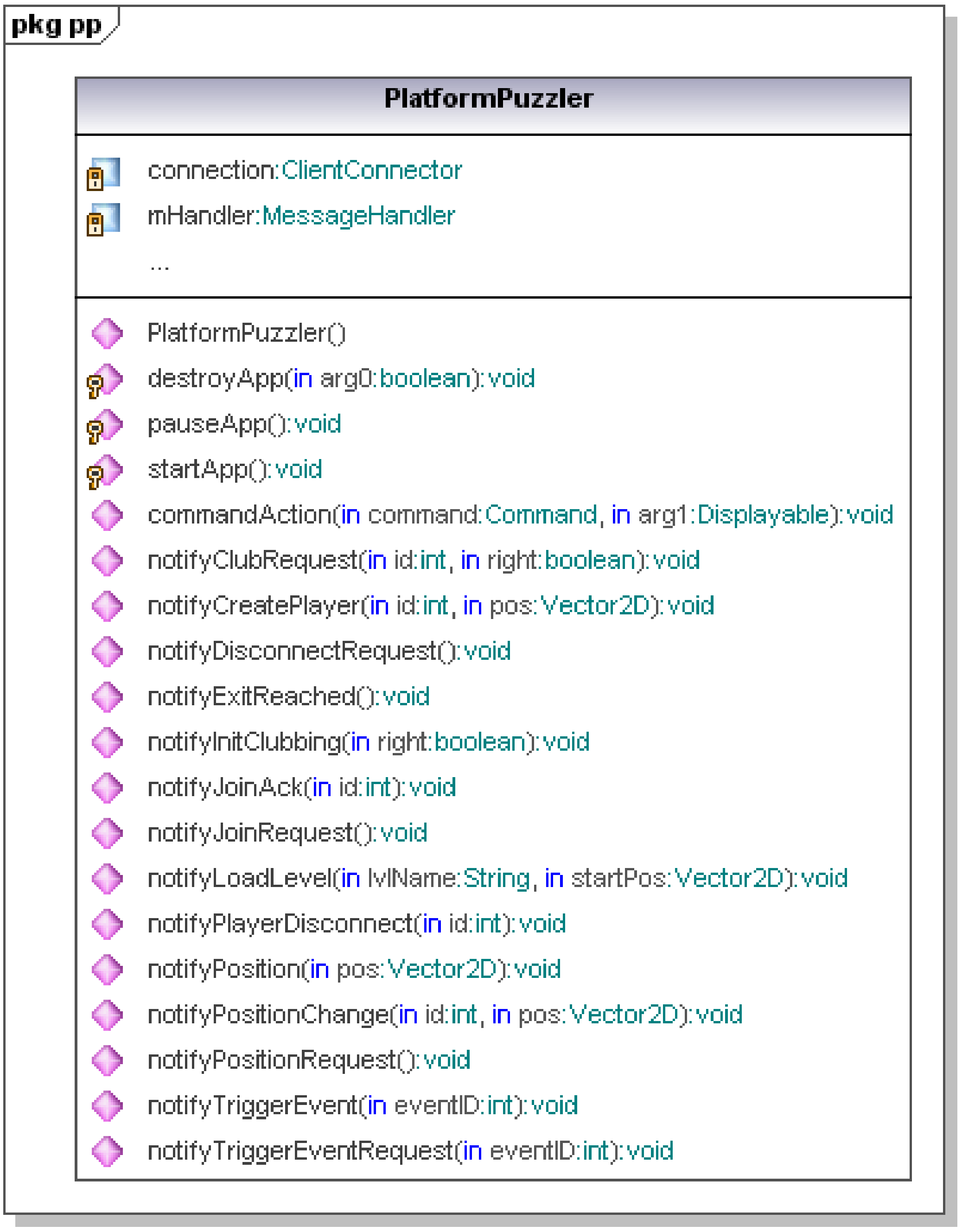

pkg pp
PlatformPuzzler
connection:ClientConnector
mHandler:MessageHandler
...
PlatformPuzzler()
destroyApp(in arg0:boolean):void
pauseApp():void
startApp():void
commandAction(in command:Command, in arg1:Displayable):void
notifyClubRequest(in id:int, in right:boolean):void
notifyCreatePlayer(in id:int, in pos:Vector2D):void
notifyDisconnectRequest():void
notifyExitReached():void
notifyInitClubbing(in right:boolean):void
notifyJoinAck(in id:int):void
notifyJoinRequest():void
notifyLoadLevel(in lvlName:String, in startPos:Vector2D):void
notifyPlayerDisconnect(in id:int):void
notifyPosition(in pos:Vector2D):void
notifyPositionChange(in id:int, in pos:Vector2D):void
notifyPositionRequest():void
notifyTriggerEvent(in eventID:int):void
notifyTriggerEventRequest(in eventID:int):void

Figure 9.7: A class diagram of the classes in the **pp** package

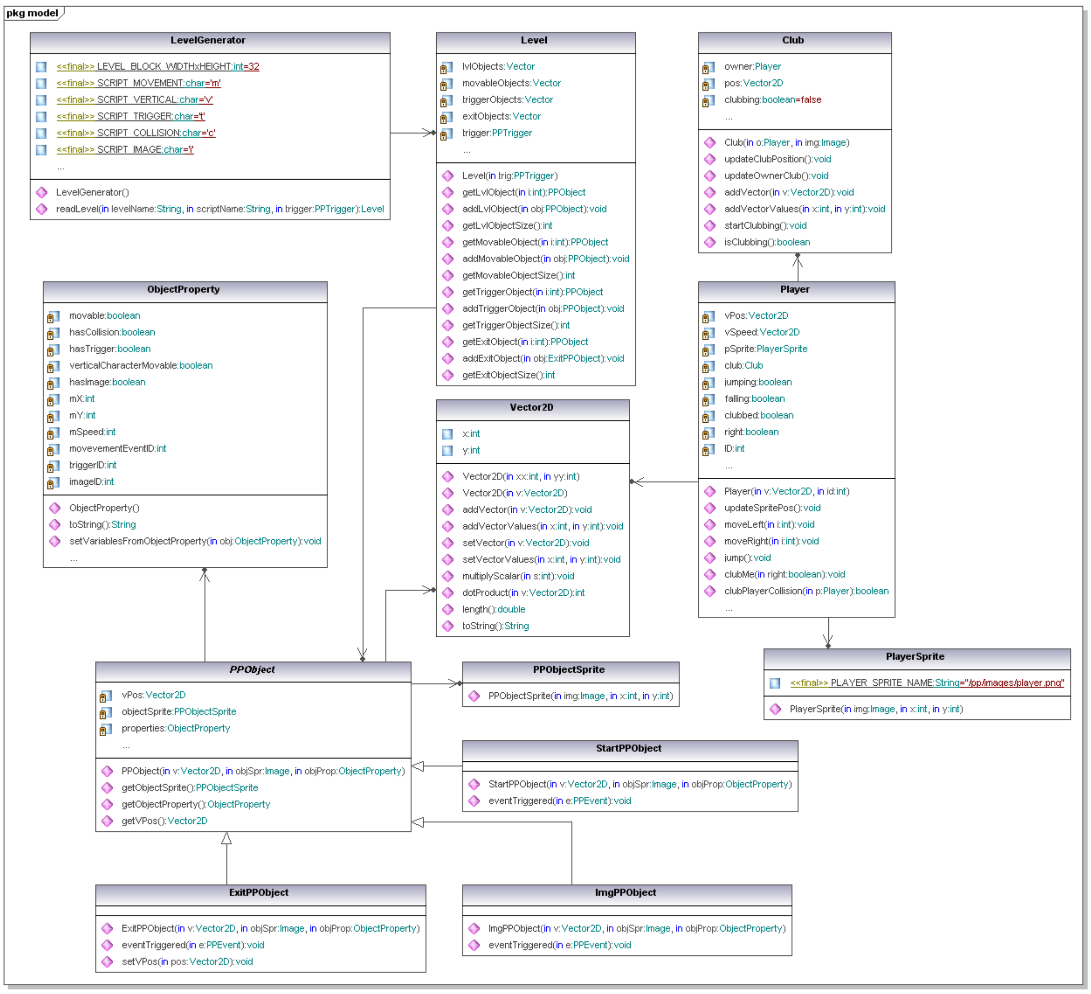

pkg model
LevelGenerator
<<final>> LEVEL_BLOCK_WIDTHxHEIGHT:int=32
<<final>> SCRIPT_MOVEMENT:char='m'
<<final>> SCRIPT_VERTICAL:char='v'
<<final>> SCRIPT_TRIGGER:char='t'
<<final>> SCRIPT_COLLISION:char='c'
<<final>> SCRIPT_IMAGE:char='i'
...
LevelGenerator()
readLevel(in levelName:String, in scriptName:String, in trigger:PPTrigger):Level
Level
lvlObjects:Vector
movableObjects:Vector
triggerObjects:Vector
exitObjects:Vector
trigger:PPTrigger
...
Level(in trig:PPTrigger)
getLvlObject(in i:int):PPObject
addLvlObject(in obj:PPObject):void
getLvlObjectSize():int
getMovableObject(in i:int):PPObject
addMovableObject(in obj:PPObject):void
getMovableObjectSize():int
getTriggerObject(in i:int):PPObject
addTriggerObject(in obj:PPObject):void
getTriggerObjectSize():int
getExitObject(in i:int):PPObject
addExitObject(in obj:ExitPPObject):void
getExitObjectSize():int
Club
owner:Player
pos:Vector2D
clubbing:boolean=false
...
Club(in o:Player, in img:Image)
updateClubPosition():void
updateOwnerClub():void
addVector(in v:Vector2D):void
addVectorValues(in x:int, in y:int):void
startClubbing():void
isClubbing():boolean
ObjectProperty
movable:boolean
hasCollision:boolean
hasTrigger:boolean
verticalCharacterMovable:boolean
hasImage:boolean
mX:int
mY:int
mSpeed:int
movevementEventID:int
triggerID:int
imageID:int
ObjectProperty()
toString():String
setVariablesFromObjectProperty(in obj:ObjectProperty):void
...
Vector2D
x:int
y:int
Vector2D(in xx:int, in yy:int)
Vector2D(in v:Vector2D)
addVector(in v:Vector2D):void
addVectorValues(in x:int, in y:int):void
setVector(in v:Vector2D):void
setVectorValues(in x:int, in y:int):void
multiplyScalar(in s:int):void
dotProduct(in v:Vector2D):int
length():double
toString():String
Player
vPos:Vector2D
vSpeed:Vector2D
pSprite:PlayerSprite
club:Club
jumping:boolean
falling:boolean
clubbed:boolean
right:boolean
ID:int
...
Player(in v:Vector2D, in id:int)
updateSpritePos():void
moveLeft(in i:int):void
moveRight(in i:int):void
jump():void
clubMe(in right:boolean):void
clubPlayerCollision(in p:Player):boolean
...
PlayerSprite
<<final>> PLAYER_SPRITE_NAME:String="/pp/images/player.png"
PlayerSprite(in img:Image, in x:int, in y:int)
PPObject
vPos:Vector2D
objectSprite:PPObjectSprite
properties:ObjectProperty
...
PPObject(in v:Vector2D, in objSpr:Image, in objProp:ObjectProperty)
getObjectSprite():PPObjectSprite
getObjectProperty():ObjectProperty
getVPos():Vector2D
PPObjectSprite
PPObjectSprite(in img:Image, in x:int, in y:int)
StartPPObject
StartPPObject(in v:Vector2D, in objSpr:Image, in objProp:ObjectProperty)
eventTriggered(in e:PPEvent):void
ExitPPObject
ExitPPObject(in v:Vector2D, in objSpr:Image, in objProp:ObjectProperty)
eventTriggered(in e:PPEvent):void
setVPos(in pos:Vector2D):void
ImgPPObject
ImgPPObject(in v:Vector2D, in objSpr:Image, in objProp:ObjectProperty)
eventTriggered(in e:PPEvent):void

Figure 9.8: A class diagram of the classes in the **pp.model** package

code by a sprite in the class **PlayerSprite**. The sprite class handles the image representing the player, as well as supplying functionality for checking collisions between the sprite of the player, and other sprites; a functionality which is inherited from the **Sprite** class, the class that the **PlayerSprite** extends.

The club mechanic, which is discussed earlier in this chapter, is supported in the **Club** class, which both contain the position of the club, as well as the graphical representation of the club in-game by extending the **Sprite** class and using an image to represent the club. The player and the club can be seen in Figure 9.6(b) which shows two player where the red balls are representations of the players, and the green "stick" is the graphical representation of the club. The club is controlled via a keypad button which the user presses to initiate the club mechanic. The game then shows a simple animation of the player attempting to beat another player with his(her) club. The collision detection between the club and the other players depend on whether the club is being actively used, or whether it is idle, i.e. when the club mechanic is not initiated by the player. If the club hits another player while not being idle, that other player is then thrust forward and upward by the impact.

A part from before mentioned functions, the **Player** class also have methods for controlling the state of the player, such as keeping track of whether the player is falling or jumping. Other methods updates the speed vector of the player depending on the direction and the action the player wants to undertake, e.g. jumping require speeds both in the x- and y-direction, while going right in the coordinate system of Java only requires a speed vector that has a positive x-direction.

The **Level** class represents the level structure of the game world, where the world is divided into one or more levels, where each level is populated with level objects. The **Level** class acts as a level object container in the code, where it stores the different level objects in lists depending on the type of level object. The **PPObject** class is the representation of level objects. This class contains and handles the global position of the level object in the game world, which is represented as a **Vector2D** object, the graphical presentation of the level object, which is handled by a **PPObjectSprite** object, and it also has a **ObjectProperty** object which contains the properties of the object. The **PPObjectSprite** contains the image of the level object and it supplies a collision detection method inherited from the **Sprite** class, which in the game is used to check whether the player is colliding with the level objects. The **ObjectProperty** class contains the different properties of the object, as discussed previously in this chapter, and it decides how the game treats the object. The properties of the object decides whether the object has a graphical representation or is translucent, whether the object can move or not, and whether the object has the property of being able to trigger an event. An event is triggered if the player is standing inside an object that has an action trigger, i.e. if the player is colliding with the object, and if the players has pressed the keypad button that indicates that the player wants to trigger the event. After the button is pressed the game then checks whether or not the player is colliding with an object that has an action trigger, and if so an event is then created.

In the **pp.model** package there are three sub classes of the **PPObject**, each with special properties not found in the **ObjectProperty** class. **ExitPPObject** is a level object that represents the exit of the level. Figure 9.6(d) shows the graphical representation of the **ExitPPObject**. If the player collides with a **ExitPPObject** the game either loads the next level, or loads the exit window of the game if there are no more levels to load. The **StartPPObject** represents the starting position of the player in a level. This level object has no graphical representation, and only indicates in a level the positions the player can start in after a new level has been loaded. The **ImgPPObject** is the standard level object do not have any special properties, thus only having the properties described in the **ObjectProperty** class.

The **LevelGenerator** class creates a new level through its **readLevel(...)** method. Each possible level in the prototype game is saved as two text files inside the JAR of the prototype game, where one of the files is a textual representation of the level where each level object in the level is represented as a character. Each of the textual representation files create a single level in the game. The **LevelGenerator** then reads through a file character by character and creates level objects with positions relative to the position of the character in the text file. There are two special characters that are treated differently than regular characters; 's' and 'e'. 's' indicates that the level object created in that position should be a **StartPPObject**, while 'e' indicates that the level object should be a **ExitPPObject**. The other of the two files representing a level, is a simple script file that creates **ObjectProperty** objects. Each line in the script file creates a **ObjectProperty** object, and each of these properties are identified by an id. This id is used in the **LevelGenerator** to tie **ObjectProperty** objects to level objects. The **readLevel(...)** method first reads the script file creating the **ObjectProperty** objects, and after that it reads through the textual representation file. Each character in the textual representation file is handled as an id. This means that if the character of the specific level object is equal to a **ObjectProperty** identifier, that **ObjectProperty** is then tied to the level object. Each **ObjectProperty** can apply top several level objects such that the script file does not need to contain a **ObjectProperty** for each level object in the level. Also, the characters 's' and 'e' cannot be used in the script file as identifiers since they have special meanings in the textual representation files.

Each **Level** created by the **LevelGenerator** class then contains the different level objects created by the **LevelGenerator**. The different level objects, i.e. **ImgPPObject**, **StartPPObject**, and **ExitPPObject**, are kept in separate lists in the **Level** object in order to simplify collision detection for instance when the player is checking to see whether (s)he is colliding with an **ExitPPObject** since the list of **ExitPPObject**'s is usually smaller than the list of all level objects in the level.

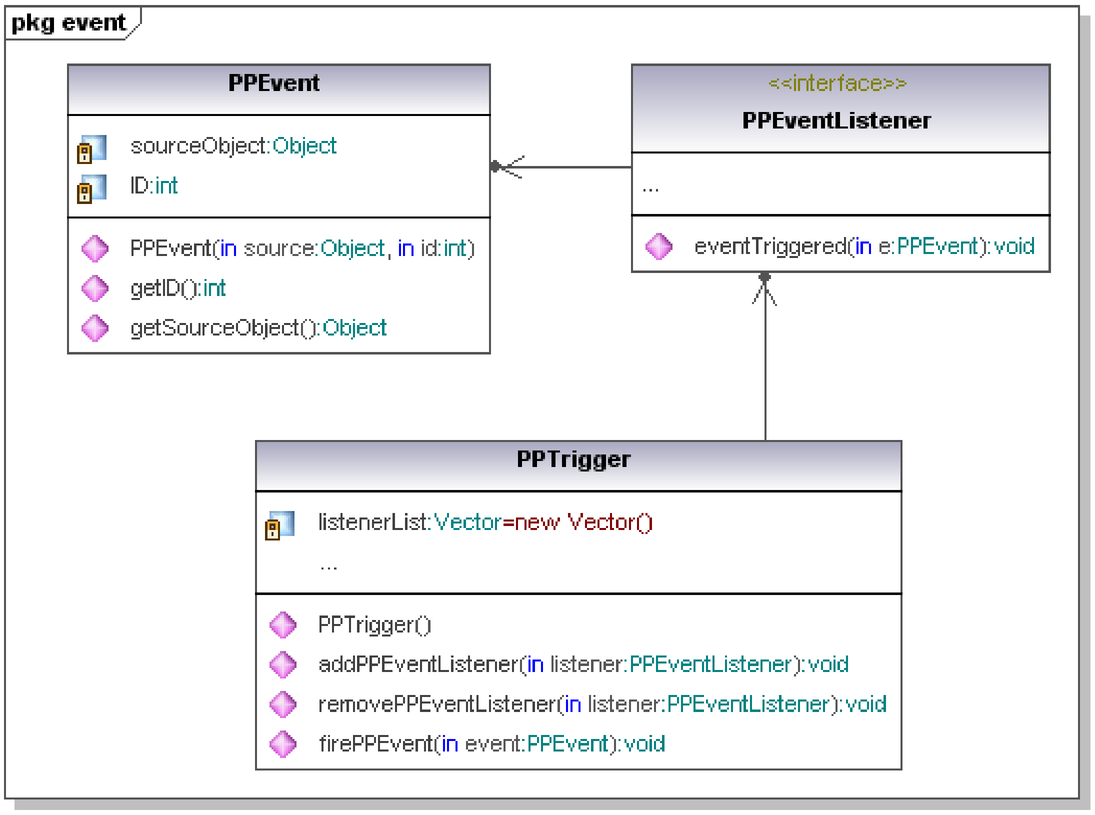

pkg event
PPEvent
sourceObject:Object
ID:int
PPEvent(in source:Object, in id:int)
getID():int
getSourceObject():Object
<<interface>>
PPEventListener
...
eventTriggered(in e:PPEvent):void
PPTrigger
listenerList:Vector=new Vector()
...
PPTrigger()
addPPEventListener(in listener:PPEventListener):void
removePPEventListener(in listener:PPEventListener):void
firePPEvent(in event:PPEvent):void

Figure 9.9: A class diagram of the classes in the **pp.model.event** package

Event

The **pp.model.event** package is a sub package of the **pp.model** package, and its classes implement the action trigger/event system which is described previously in this chapter. There are three classes in this package, **PPEvent**, **PPEventListener**, and **PPTrigger**. Figure 9.9 shows the class diagram of these classes. The classes of this package implement a simple version of the event notification pattern, where an object can be notified about an event. An object that implements the **PPEventListener** interface and registers to a **PPTrigger** object is able to listen to notifications about events. Classes implementing the **PPEventListener** must implement the **eventTriggered(...)** method. This method is called by a **PPTrigger** when a situation in the game has created a new **PPEvent**. The **PPTrigger** class has a list of event listeners that it notifies when another class calls the **firePPEvent(...)** method. Any **PPEventListener** object that has registered to the specific **PPTrigger** is then notified about the new event, and they receive an **PPEvent** that gives an indication about what event has occurred in the game. The **PPEvent** class has a identifier that is used to distinguish between events, as well as a variable that tells the listeners what the original object of the event creation was. The identifier of the event is used in the game to distinguish between the level objects that created the event.

Each level object can listen to incoming events, and the identifier is used to decide whether or not the listening level objects should respond to the event. In the prototype game the player has a keypad button tied to the trigger mechanism such that when the player presses the button, the game checks to see whether or not he player is colliding with an level object that has the property of being able to trigger events. If the player has collided with such an object, the game takes the trigger id of the trigger object and by using a **PPTrigger** object, calls the **firePPEvent(...)** method thereby notifying the listening level objects about the event. If any of the listening level objects are listening to the specific identifier of the original trigger object, the listening object performs the action that is tied to that specific identifier.

UI

Figure 9.10 shows the class diagram of the classes in the **pp.ui** package. This package contains the different graphical user interface screens that are used in the prototype game, except the starting screen which lie in the **pp** top package. The **EndWindow** is the screen that pops up after the player has entered an exit are of a level, and there are no more levels to load. The screen informs the player that the prototype game is at its end, and that the player should exit the game. The **HelpWindow** is a screen that is meant be a help screen for the player where (s)he can read how the game is controlled, and how some of the mechanics of the game work, but this screen is currently not functioning correctly, and the player is unable to invoke the screen in the game, even though the class still exists in the source code.

GameWindow is the class that contains the main game loop in the prototype game, as well as the screen on which the user is playing. The class inherits from the **GameCanvas** class which is found in MIDP, giving the **GameWindow** rendering possibilities, as well as button press pulling, i.e. the ability to pull the super class information about button presses. The **GameWindow** class is thus responsible for both rendering the graphical output to the player, as well as controlling the game logic. **GameWindow** keeps both a **Player** object, which is the client running the application, as well as a list of the other players connected to the same server. The **GameWindow** class uses these to keep track of and initializing player movements and actions after the player has invoked these actions by pressing buttons on the keypad, or after having received messages from the server requiring updates to either the main player, or the players in the player list. The **GameWindow** displays and keeps track of the current level, after having received a message from the server telling the **GameWindow** which level it should load.

As mentioned above the main game loop is also controlled by the **GameWindow**. In each round of the loop it executes methods that check to see whether the player has pressed any buttons on the keypad, and if so it takes the appropriate actions such as for

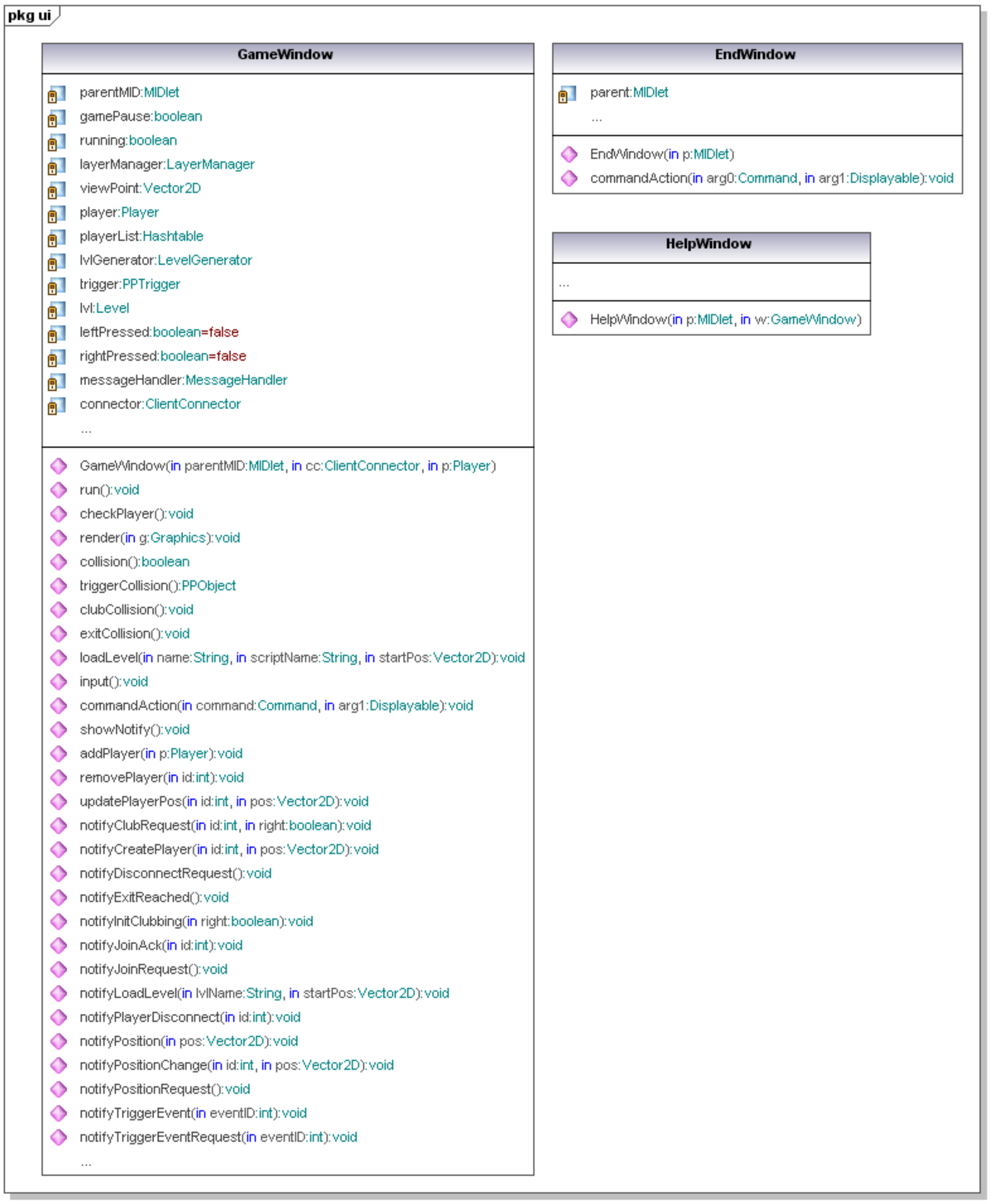

pkg ui
GameWindow
parentMID:MIDlet
gamePause:boolean
running:boolean
layerManager:LayerManager
viewPoint:Vector2D
player:Player
playerList:Hashtable
lvlGenerator:LevelGenerator
trigger:PPTrigger
lvl:Level
leftPressed:boolean=false
rightPressed:boolean=false
messageHandler:MessageHandler
connector:ClientConnector
...
GameWindow(in parentMID:MIDlet, in cc:ClientConnector, in p:Player)
run():void
checkPlayer():void
render(in g:Graphics):void
collision():boolean
triggerCollision():PPObject
clubCollision():void
exitCollision():void
loadLevel(in name:String, in scriptName:String, in startPos:Vector2D):void
input():void
commandAction(in command:Command, in arg1:Displayable):void
showNotify():void
addPlayer(in p:Player):void
removePlayer(in id:int):void
updatePlayerPos(in id:int, in pos:Vector2D):void
notifyClubRequest(in id:int, in right:boolean):void
notifyCreatePlayer(in id:int, in pos:Vector2D):void
notifyDisconnectRequest():void
notifyExitReached():void
notifyInitClubbing(in right:boolean):void
notifyJoinAck(in id:int):void
notifyJoinRequest():void
notifyLoadLevel(in lvlName:String, in startPos:Vector2D):void
notifyPlayerDisconnect(in id:int):void
notifyPosition(in pos:Vector2D):void
notifyPositionChange(in id:int, in pos:Vector2D):void
notifyPositionRequest():void
notifyTriggerEvent(in eventID:int):void
notifyTriggerEventRequest(in eventID:int):void
...
EndWindow
parent:MIDlet
...
EndWindow(in p:MIDlet)
commandAction(in arg0:Command, in arg1:Displayable):void
HelpWindow
...
HelpWindow(in p:MIDlet, in w:GameWindow)

Figure 9.10: A class diagram of the classes in the **pp.ui** package

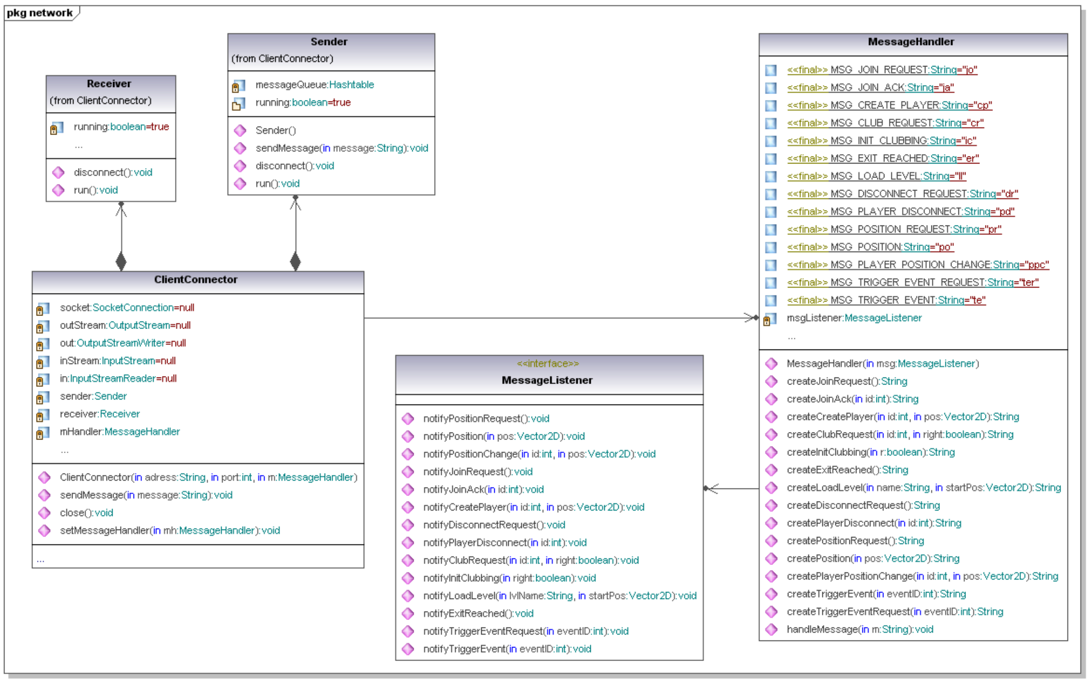

pkg network
Receiver
(from ClientConnector)
running:boolean=true
...
disconnect():void
run():void
Sender
(from ClientConnector)
messageQueue:Hashtable
running:boolean=true
Sender()
sendMessage(in message:String):void
disconnect():void
run():void
ClientConnector
socket:SocketConnection=null
outStream:OutputStream=null
out:OutputStreamWriter=null
inStream:InputStream=null
in:InputStreamReader=null
sender:Sender
receiver:Receiver
mHandler:MessageHandler
...
ClientConnector(in adress:String, in port:int, in m:MessageHandler)
sendMessage(in message:String):void
close():void
setMessageHandler(in mh:MessageHandler):void
...
<<interface>>
MessageListener
notifyPositionRequest():void
notifyPosition(in pos:Vector2D):void
notifyPositionChange(in id:int, in pos:Vector2D):void
notifyJoinRequest():void
notifyJoinAck(in id:int):void
notifyCreatePlayer(in id:int, in pos:Vector2D):void
notifyDisconnectRequest():void
notifyPlayerDisconnect(in id:int):void
notifyClubRequest(in id:int, in right:boolean):void
notifyInitClubbing(in right:boolean):void
notifyLoadLevel(in lvlName:String, in startPos:Vector2D):void
notifyExitReached():void
notifyTriggerEventRequest(in eventID:int):void
notifyTriggerEvent(in eventID:int):void
MessageHandler
<<final>> MSG_JOIN_REQUEST:String="jo"
<<final>> MSG_JOIN_ACK:String="ja"
<<final>> MSG_CREATE_PLAYER:String="cp"
<<final>> MSG_CLUB_REQUEST:String="cr"
<<final>> MSG_INIT_CLUBBING:String="ic"
<<final>> MSG_EXIT_REACHED:String="er"
<<final>> MSG_LOAD_LEVEL:String="ll"
<<final>> MSG_DISCONNECT_REQUEST:String="dr"
<<final>> MSG_PLAYER_DISCONNECT:String="pd"
<<final>> MSG_POSITION_REQUEST:String="pr"
<<final>> MSG_POSITION:String="po"
<<final>> MSG_PLAYER_POSITION_CHANGE:String="ppc"
<<final>> MSG_TRIGGER_EVENT_REQUEST:String="ter"
<<final>> MSG_TRIGGER_EVENT:String="te"
msgListener:MessageListener
...
MessageHandler(in msg:MessageListener)
createJoinRequest():String
createJoinAck(in id:int):String
createCreatePlayer(in id:int, in pos:Vector2D):String
createClubRequest(in id:int, in right:boolean):String
createInitClubbing(in r:boolean):String
createExitReached():String
createLoadLevel(in name:String, in startPos:Vector2D):String
createDisconnectRequest():String
createPlayerDisconnect(in id:int):String
createPositionRequest():String
createPosition(in pos:Vector2D):String
createPlayerPositionChange(in id:int, in pos:Vector2D):String
createTriggerEvent(in eventID:int):String
createTriggerEventRequest(in eventID:int):String
handleMessage(in m:String):void

Figure 9.11: A class diagram of the classes in the **pp.network** package

instance moving the player to the left. Each execution of the game loop also checks for collision between the player and the game world. Specifically it checks to see whether the player collides with any of the level objects in the current level that have the property that enables them to be collidable, as well as checking to see whether the player collides with an exit level object, and whether the club of the player collides with another player after the player has activated the club mechanic. Lastly the game loop renders the current frame by using a **LayerManager** to keep track of all of the sprites in the level, i.e. the level objects, the players, and the clubs of the players. After having invoked all of the actions above, the game loop sleeps for a short period; a period which takes into account the time spent for each execution of the loop in order to try to create a stable frame rate.

Network

Figure 9.11 shows the class diagrams of the classes inside the **pp.network** package. The source code in this package is based on the code produced in the depth study in TDT-4570 [Nøsterud, 2007], but it is slightly changed in order to accommodate the changes to the underlying game concept, as well as changes to the source code utilizing this package.

The main components of the **pp.network** package is the **ClientConnector** class with its inner nested classes, and the two classes responsible for the message handling on the client side. The **MessageListener** class along with the **MessageHandler** handles the creation, parsing, and notification of messages to and from the server in the client. The two classes form a simple implementation of the observer pattern in that classes wishing to listen to incoming messages from the server implements the **MessageListener** interface as well as creating a **MessageHandler** object in order to parse the incoming messages, as well as create the out going messages. The **MessageListener** has several methods that notifies the implementor class which types of messages the sever has sent the client. In going messages first go through the **MessageHandler** which parses the messages looking for the information found within the message, and then calls the correct notifying method in the **MesasgeHandler** depending on the message content. The **MessageListener** interface then informs, or notifies, its implementor about the message, and the implementor then takes the appropriate action to the message. The **MessageHandler** is also used to create valid messages with the correct message syntax used in the prototype game, thereby defining the syntax.

The **ClientConnector** handles the communication between the client and the server through a TCP/IP connection. The class has a **MessageHandler** object that it sends the incoming messages to in order to parse them, and it also contains the different Java streams used to send and receive messages from the server. The **ClientConnector** has two nested inner classes, **Sender** and **Receiver**, and these classes run in two separate threads. The **Sender** thread sends out going messages to the server and uses a message queue in order to receive more than one outgoing message at a time. When the **Sender** thread is not sending messages it is sleeping and waiting for the **ClientConnector** to notify the **Sender** that there are new messages to be sent to the server. This happens by classes calling the **sendMessage(...)** of a **ClientConnector** object. The **Receiver** thread handles incoming messages from the server, and it sends each message to the **MesasgeHandler** so that the message can be parsed. The **Receiver** runs in a loop that for each iteration of the loop attempts to read a new message from the server stream, and if the stream contains a message the **Receiver** reads the stream message for message and sends each complete message to the **MessageHandler**.

9.3.2 Server

The server package is, as discussed above, a separate application from the client application, and it is not created in Java ME either, but instead in Java SE and can thus run on a PC's, or any other platform able to run Java SE applications. The server contains two main packages which are the **pps.model** package, and the **pps.server** package. Figure 9.12 shows the package diagram of the server application. The **pps.model** package contains a class than handles server specific level information, i.e. information relevant for the server to know about the levels, as well as a 2 dimensional vector class. The **pps.server** package contains classes that handle the communication and connection with the clients, the server logic, and the message handling which consists of handling incoming messages and parsing them, as well as creating outgoing messages with the correct syntax.

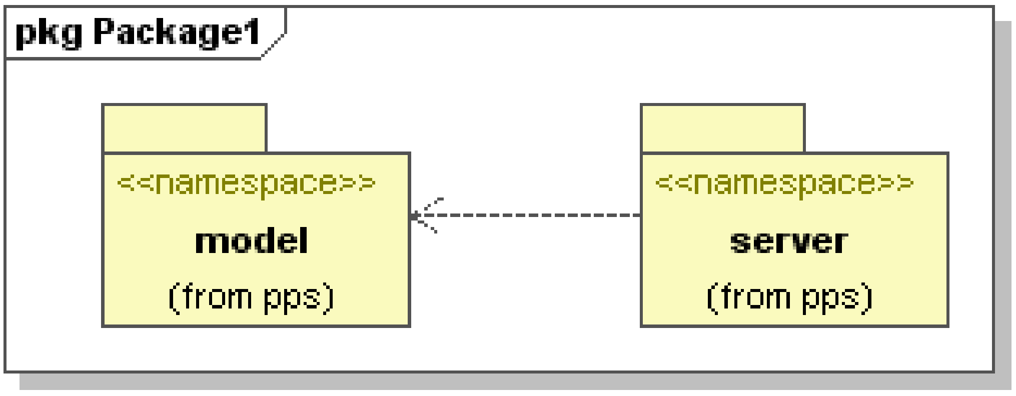


Figure 9.12: Package diagram of the server application

Model

Figure 9.13 shows the class diagram of the classes inside the **pps.model** package. The package contains the two classes **Vector2D** and **LevelCoordinator**. The **Vector2D** class is the same class as the vector class in the client application. It supports simple vector math and is used in the server to keep track of the locations of the players, as well as being used in the **LevelCoordinator** class to keep track of the starting points in each level.

The **LevelCoordinator** class handles level specific information in the server application. The main responsibilities for the class are to keep track of the current level for the server, appointing start locations to each individual player at the start of each level, as well as keeping track of the current level and deciding which level should be loaded next. The server has a description file that decides the level order, i.e. in which order the levels should be loaded and played by the players. The server also has copies of the level description files which are found in the client application(but not the script files found in the client application), and these textual description files are used to find the starting points of the level, as each level may have more than one starting point. The **LevelCoordinator** firstly reads in the order of the levels in the system and sets the current level to the first level in that file, and then it finds the starting points in that level. When a new level should by loaded after a player has entered an exit level object, the **LevelCoordinator** finds the next level and resets the starting points to fit the new level. If there are no more levels to load, the server sends a message to the client saying that it should load its end screen, i.e. quit the main game loop and initialize a **EndWindow** object.

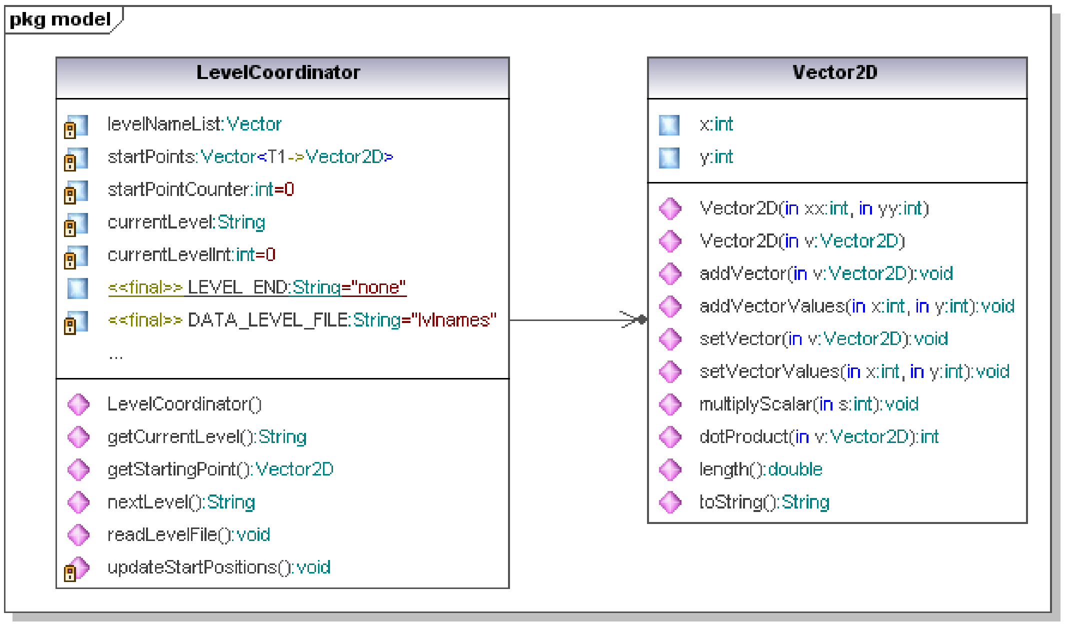

pkg model
LevelCoordinator
levelNameList:Vector
startPoints:Vector<T1->Vector2D>
startPointCounter:int=0
currentLevel:String
currentLevelInt:int=0
<<final>> LEVEL_END:String="none"
<<final>> DATA_LEVEL_FILE:String="lvlnames"
...
LevelCoordinator()
getCurrentLevel():String
getStartingPoint():Vector2D
nextLevel():String
readLevelFile():void
updateStartPositions():void
Vector2D
x:int
y:int
Vector2D(in xx:int, in yy:int)
Vector2D(in v:Vector2D)
addVector(in v:Vector2D):void
addVectorValues(in x:int, in y:int):void
setVector(in v:Vector2D):void
setVectorValues(in x:int, in y:int):void
multiplyScalar(in s:int):void
dotProduct(in v:Vector2D):int
length():double
toString():String

Figure 9.13: A class diagram of the classes in the **pps.model** package

Server

Figure 9.14 shows the class diagram of the classes in the **pps.server** package. The classes in this package are similar to the ones in the **pp.network** package on the client application. The source code of this package is also based on the depth study performed in TDT-4570 [Nøsterud, 2007], but it has been modified to fit the **Platform-puzzler** prototype. The biggest changes are in the messages and the server logic.

The **MessageHandler** class and the **MessageListener** class are similar to the same classes found in the **pp.network** package in the client application, and they work in the same way where an object implements the **MessageListener** interface in order to listen to incoming messages from the client which are parsed by the **MessageHandler**. The **MessageHandler** also similarly as in the client application define and controls the message syntax, and is used to create valid messages.

ServerConnector is the main class in the server application. It controls the server logic in the prototype game, and it implements the **MessageListener** interface in order to be informed about incoming messages from the clients. The **ServerConenctor** runs a loop that listens to incoming connections from new clients, and after a new client has connected, it creates a new **PlayerSession** object that keeps track of that particular client. Each client thus has a **PlayerSession** object and the messages received from the clients are identified by their **PlayerSession** object. Each **PlayerSession** has two

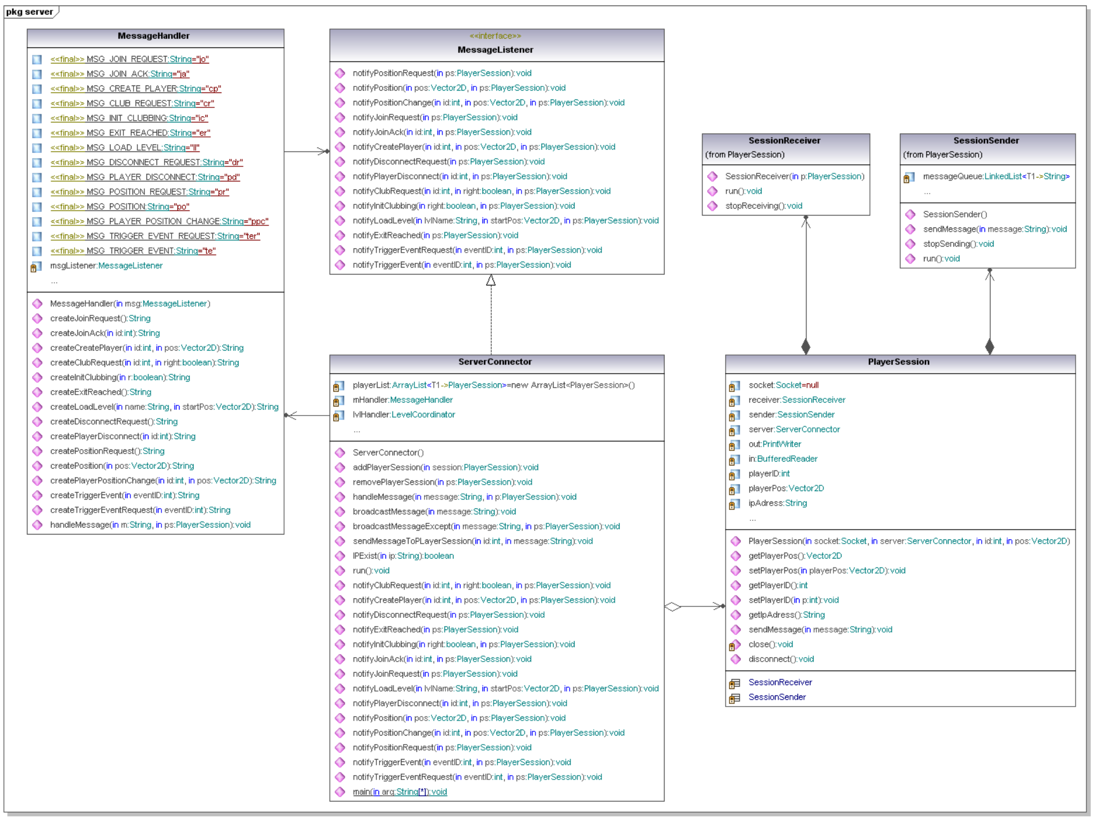

pkg server
MessageHandler
<<final>> MSG_JOIN_REQUEST:String="jo"
<<final>> MSG_JOIN_ACK:String="ja"
<<final>> MSG_CREATE_PLAYER:String="cp"
<<final>> MSG_CLUB_REQUEST:String="cr"
<<final>> MSG_INIT_CLUBBING:String="ic"
<<final>> MSG_EXIT_REACHED:String="er"
<<final>> MSG_LOAD_LEVEL:String="ll"
<<final>> MSG_DISCONNECT_REQUEST:String="dr"
<<final>> MSG_PLAYER_DISCONNECT:String="pd"
<<final>> MSG_POSITION_REQUEST:String="pr"
<<final>> MSG_POSITION:String="po"
<<final>> MSG_PLAYER_POSITION_CHANGE:String="ppc"
<<final>> MSG_TRIGGER_EVENT_REQUEST:String="ter"
<<final>> MSG_TRIGGER_EVENT:String="te"
msgListener:MessageListener
...
MessageHandler(in msg:MessageListener)
createJoinRequest():String
createJoinAck(in id:int):String
createCreatePlayer(in id:int, in pos:Vector2D):String
createClubRequest(in id:int, in right:boolean):String
createInitClubbing(in r:boolean):String
createExitReached():String
createLoadLevel(in name:String, in startPos:Vector2D):String
createDisconnectRequest():String
createPlayerDisconnect(in id:int):String
createPositionRequest():String
createPosition(in pos:Vector2D):String
createPlayerPositionChange(in id:int, in pos:Vector2D):String
createTriggerEvent(in eventID:int):String
createTriggerEventRequest(in eventID:int):String
handleMessage(in m:String, in ps:PlayerSession):void
<<interface>>
MessageListener
notifyPositionRequest(in ps:PlayerSession):void
notifyPosition(in pos:Vector2D, in ps:PlayerSession):void
notifyPositionChange(in id:int, in pos:Vector2D, in ps:PlayerSession):void
notifyJoinRequest(in ps:PlayerSession):void
notifyJoinAck(in id:int, in ps:PlayerSession):void
notifyCreatePlayer(in id:int, in pos:Vector2D, in ps:PlayerSession):void
notifyDisconnectRequest(in ps:PlayerSession):void
notifyPlayerDisconnect(in id:int, in ps:PlayerSession):void
notifyClubRequest(in id:int, in right:boolean, in ps:PlayerSession):void
notifyInitClubbing(in right:boolean, in ps:PlayerSession):void
notifyLoadLevel(in lvlName:String, in startPos:Vector2D, in ps:PlayerSession):void
notifyExitReached(in ps:PlayerSession):void
notifyTriggerEventRequest(in eventID:int, in ps:PlayerSession):void
notifyTriggerEvent(in eventID:int, in ps:PlayerSession):void
SessionReceiver
(from PlayerSession)
SessionReceiver(in p:PlayerSession)
run():void
stopReceiving():void
SessionSender
(from PlayerSession)
messageQueue:LinkedList<T1->String>
...
SessionSender()
sendMessage(in message:String):void
stopSending():void
run():void
ServerConnector
playerList:ArrayList<T1->PlayerSession>=new ArrayList<PlayerSession>()
mHandler:MessageHandler
lvlHandler:LevelCoordinator
...
ServerConnector()
addPlayerSession(in session:PlayerSession):void
removePlayerSession(in ps:PlayerSession):void
handleMessage(in message:String, in p:PlayerSession):void
broadcastMessage(in message:String):void
broadcastMessageExcept(in message:String, in ps:PlayerSession):void
sendMessageToPLayerSession(in id:int, in message:String):void
IPExist(in ip:String):boolean
run():void
notifyClubRequest(in id:int, in right:boolean, in ps:PlayerSession):void
notifyCreatePlayer(in id:int, in pos:Vector2D, in ps:PlayerSession):void
notifyDisconnectRequest(in ps:PlayerSession):void
notifyExitReached(in ps:PlayerSession):void
notifyInitClubbing(in right:boolean, in ps:PlayerSession):void
notifyJoinAck(in id:int, in ps:PlayerSession):void
notifyJoinRequest(in ps:PlayerSession):void
notifyLoadLevel(in lvlName:String, in startPos:Vector2D, in ps:PlayerSession):void
notifyPlayerDisconnect(in id:int, in ps:PlayerSession):void
notifyPosition(in pos:Vector2D, in ps:PlayerSession):void
notifyPositionChange(in id:int, in pos:Vector2D, in ps:PlayerSession):void
notifyPositionRequest(in ps:PlayerSession):void
notifyTriggerEvent(in eventID:int, in ps:PlayerSession):void
notifyTriggerEventRequest(in eventID:int, in ps:PlayerSession):void
main(in arg:String[*]):void
PlayerSession
socket:Socket=null
receiver:SessionReceiver
sender:SessionSender
server:ServerConnector
out:PrintWriter
in:BufferedReader
playerID:int
playerPos:Vector2D
ipAdress:String
...
PlayerSession(in socket:Socket, in server:ServerConnector, in id:int, in pos:Vector2D)
getPlayerPos():Vector2D
setPlayerPos(in playerPos:Vector2D):void
getPlayerID():int
setPlayerID(in p:int):void
getIpAdress():String
sendMessage(in message:String):void
close():void
disconnect():void
SessionReceiver
SessionSender

Figure 9.14: A class diagram of the classes in the **pp.server** package

threads that take care of the communication between the server and that particular client. These threads are the nested inner classes **SessionReceiver** and **SessionSender**, and they function similarly to the **Sender** and **Receiver** classes of the **pp.network** package. **SessionSender** has a message queue where it saves outgoing messages that the server intends to send to the client, and while there are no messages in the message queue the **SessionSender** waits idly until it is notified about new messages that should be sent to the client. The **SessionReceiver** runs in a loop where for each iteration of the loop it tries to receive new messages from the client. If there are new messages to the server these messages are sent to the **MessageHandler** which in turn parses them and gathers the information found within the messages, and notifies the **ServerConenctor** such that it can take the appropriate action to the message. Each **PlayerSession** is given an identifier used to distinguish between the different clients, and it also keeps track of the position of the player that is associated with that particular **PlayerSession**.

A part from connecting to new clients, the **ServerConnector** also handles the server logic. It periodically sends out requests to the clients about their player positions, and it also takes the appropriate actions depending on the incoming messages from the clients, e.g. when a client wants to initiate a club mechanic on another player, or when a player has entered an exit object in a level and a new level should be loaded by the clients. The server's logic is typically based around deciding the appropriate action based on incoming messages from clients, informing the server about the current state of the game levels, and the server then sends out the information to the correct clients which then uses the information to conduct the appropriate client action.

9.3.3 Client-server interaction

The actual transmission of messages between the server and the clients happen through TCP/IP and there are dedicated packages that in both the client and server applications that control the network and flow of messages. The package **pp.network** for the clients, and **pps.server** for the server, handles the network in the two applications, as discussed above.

Figure 9.15 shows a sequence in which a generic message is sent from a client to a server. To send a message to a server, the client must first create a **ClientConnector** object with the correct IP address to the server. The **ClientConnector** then creates two threads that handle sending and receiving messages to and from the server. The **ClientConnector** object can then be used to send messages to the server, as is shown in the Figure. Here an object has created an **ClientConnector** object, and uses it to send a message to the server. Because the **MessageHandler** defines the message syntax in the prototype, which is discussed more thoroughly above, the **ClientConnector** must use a method in the **MessageHandler** that creates a message that has the correct message syntax for the message type the client wants to send. The prototype game has several different message types, and the **MessageHandler** has a method for each of the method types which creates a valid message of that type. In Figure 9.15 the client

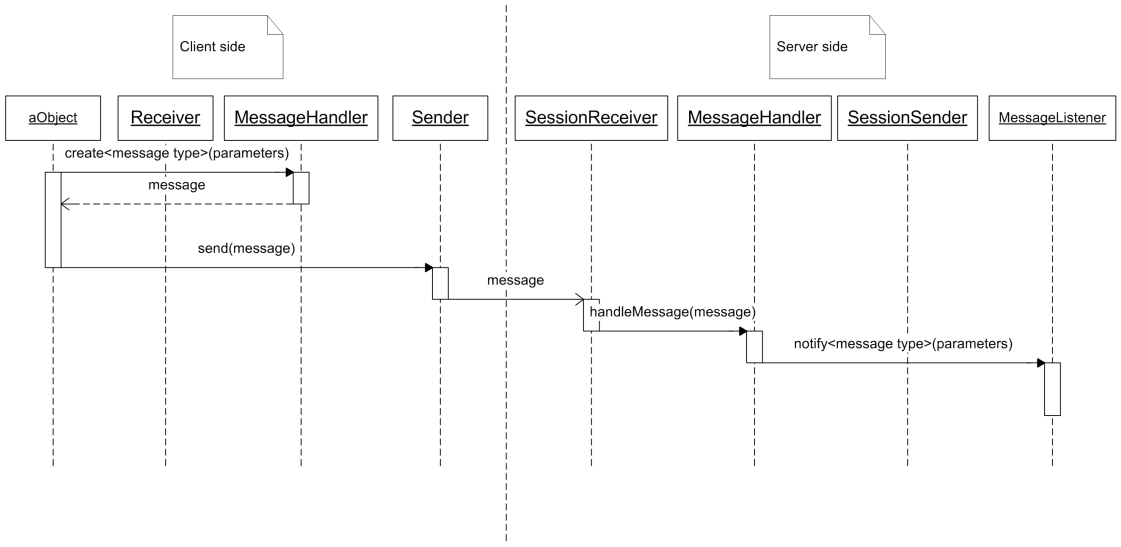


Figure 9.15: Sequence diagram showing a client sending a message to the server

is sending a message that where the message is not specified, so it could be any valid message. The client then uses the **Sender** object created by the **ClientConnector** to the server. In the server the class **PlayerSession** is similar to the **ClientConnector**, which is described in the previous section, and the **SessionReceiver** object created by the **PlayerSession** receives the message sent by the client. The message is then sent to the **MessageHandler** which validates it and parses the information found within it. If there are any classes that are registered as **MessageListener**'s to the **MessageHandler**, these classes are notified about the incoming message. Just like the **MessageHandler** has a method for the creation of valid messages for each message type, the **MessageListener** has a notify method for each message, and calls the methods based on the type of message. On the server, the **ServerConnector** is the only listener, and thus it ultimately receives the finally parsed message, and decides which actions must be taken. While Figure 9.15 showed a client sending a message to the server, Figure 9.16 shows the server sending a message to a client. Since the classes handling the message sending and receiving are similar for both the client and server applications, the process is similar between the figures. The **ServerConnector** starts by getting a valid message from the **MessageHandler** and sends it to the client using the **SessionSender** object created by the **PlayerSession**. A difference between the client and the server is that the server has several connections where each connection corresponds to a client, represented by a **PlayerSession** object, whereas the client only has one connection; the server. The server thus has the additional work of deciding which client to send the message to. On the client side the message is received by a **Receiver** object, and the message is forwarded to a **MessageHandler** that parses the message and notifies the object listening to the connection. Another difference between the server and client is that on the server side, the **ServerConnector** is the only object implementing the **MessageListener** interface,

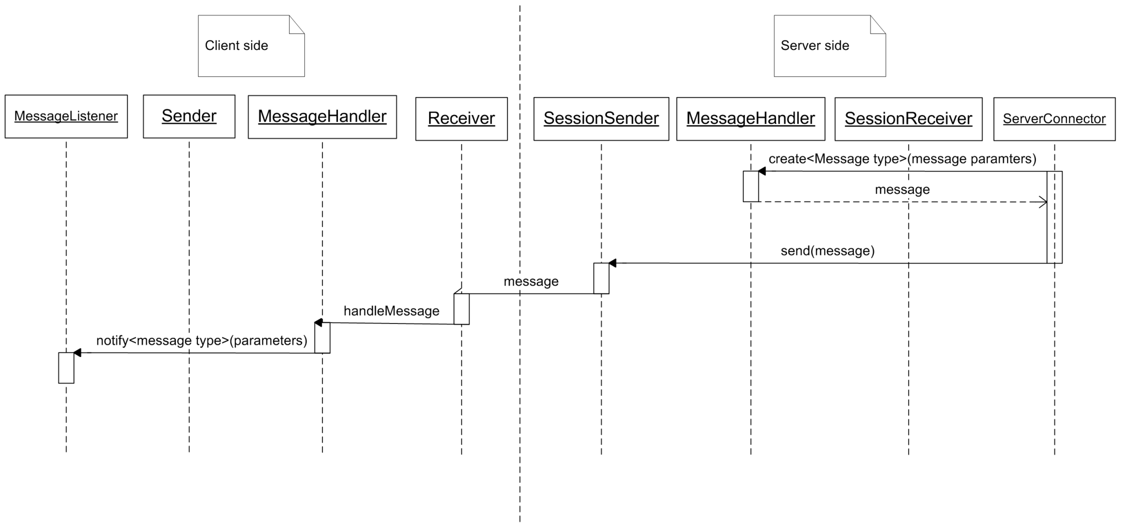


Figure 9.16: Sequence diagram showing the server sending a message to a client

```
<message prefix>:<message part 1>,<message part 2>,...,<message part n>|
```

Figure 9.17: The basic syntax of the messages in the prototype

while on the client side there are more than one object implementing the interface, and they react differently to the messages types. Generally though the **GameWindow** class handles most of the logic regarding incoming messages.

Message structure

The diagrams explained above show how the basic connection between the client and server are handled in the prototype, however they do not show the structure of the messages. Figure 9.17 shows the basic syntax of a message. Each message is a Java String object, sent on the TCP/IP connection as bytes, and parsed in the **MessageHandler**, and the message consists of a message prefix that shows what type of message it is, e.g. a message telling the server that a client wants to join the server. The rest of the message(the tail of the message) is separated with an ':' character, and the tail of the message contains from 0 to n message parts. These message parts contains variables that are parsed. Each variable is separated by an ',' character, and they are not identified by a String such as the message prefix. The message parts must thus be put in the correct position, and be of the correct type(e.g. a integer or floating point number) in order to be read correctly. This is however not a problem since the **MessageHandler** defines the syntax, and it both creates and parses the messages. Each message is ended by a '|' character; this in order to separate the messages when reading them. Figure 9.18 shows the entire message syntax with all of the different message types. The syntax is described using BNF(Backus Naur form). Here a message can be one of the different message types and each message type can have 0 to n variables where the variables are either an integer, a Java String, or an Java boolean. The different message types are:

Join request which is used when the player wants to join a server

Join acknowledgment which is sent by the server to inform the client that the client can as well as assigning an ID to the client

Create player which is sent by the server to notify the client that there are another player that is playing on the server

Club request which is sent by the client when it wants to initiate a club mechanic on another player

Initiate clubbing which is sent from the server to the client that should initiate the clubbing mechanic on itself, after the server has received a club request from another client

Exit reached which is sent by a client when that client has reached the end of an level, and it leads to the server trying to load a new level and sending the message **Load level**

Disconnect request which is sent by a client when it wants to disconnect from the server, which leads to the server sending a **player disconnect** message to the other players informing the other players that a player has quit the server

Position request which is sent by the server to request the position of the player, which leads to the client sending a **position** message which contains the information about the position of the player, i.e. the x and y position

Player position change which is broadcast to all of the players to inform the players that the player with the specified position has moved

Trigger event request which is sent by a client after having pressed a level object that has an trigger, and it leads to the server broadcasting a "trigger event" message which informs all of the players that an event with the specified ID should be created and sent to the listeners

Connecting to the server

Figure 9.19 shows a sequence diagram describing the process of a client connecting to a server, although it is in a simplified form where the message handling sequences shown in Figure 9.15 are left out. The connection is established by sending messages between the client and server, where they react to the messages and possibly send new messages to each other based on the type of message received. The first message of the diagram is the player wanting to start the game. When the game starts, a **PlatformPuzzler** object is created, and to connect to the server the player has to enter the IP of the server to connect

```
<message>                ::= <join request> | <join acknowledgment> |
                             <create player> | <club request> |
                             <initiate clubbing> | <exit reached> |
                             <load level> | <disconnect request> |
                             <player disconnect> | <position request> |
                             <position> | <player position change> |
                             <trigger event> | <trigger event request>
<join request>           ::= "jo"  ":" "|"
<join acknowledgment>    ::= "ja"  ":" <player id> "|"
<create player>          ::= "cp"  ":" <player id> "," <player x-pos> ","
                             <player y-pos> "|"
<club request>           ::= "cr"  ":" <player id> "," <right> "|"
<initiate clubbing>      ::= "ic"  ":" <right> "|"
<exit reached>           ::= "er"  ":" "|"
<load level>             ::= "ll"  ":" <level name> "," <player x-pos> ","
                             <player y-pos> "|"
<disconnect request>     ::= "dr"  ":" "|"
<player disconnect>      ::= "pd"  ":" <player id> "|"
<position request>       ::= "pr"  ":" "|"
<position>               ::= "po"  ":" <player x-pos> "," <player y-pos> "|"
<player position change>::= "ppc" ":" <player id> "," <player x-pos> ","
                             <player y-pos> "|"
<trigger event>          ::= "te"  ":" <event id> "|"
<trigger event request> ::= "ter" ":" <event id> "|"
<player id>              ::= <integer>
<player x-pos>           ::= <integer>
<player y-pos>           ::= <integer>
<right>                  ::= true | false
<level name>             ::= <string>
<event id>               ::= <integer>
<integer>                ::= Java integer
<string>                 ::= Java String
```

Figure 9.18: The definition of the message syntax used in the prototype

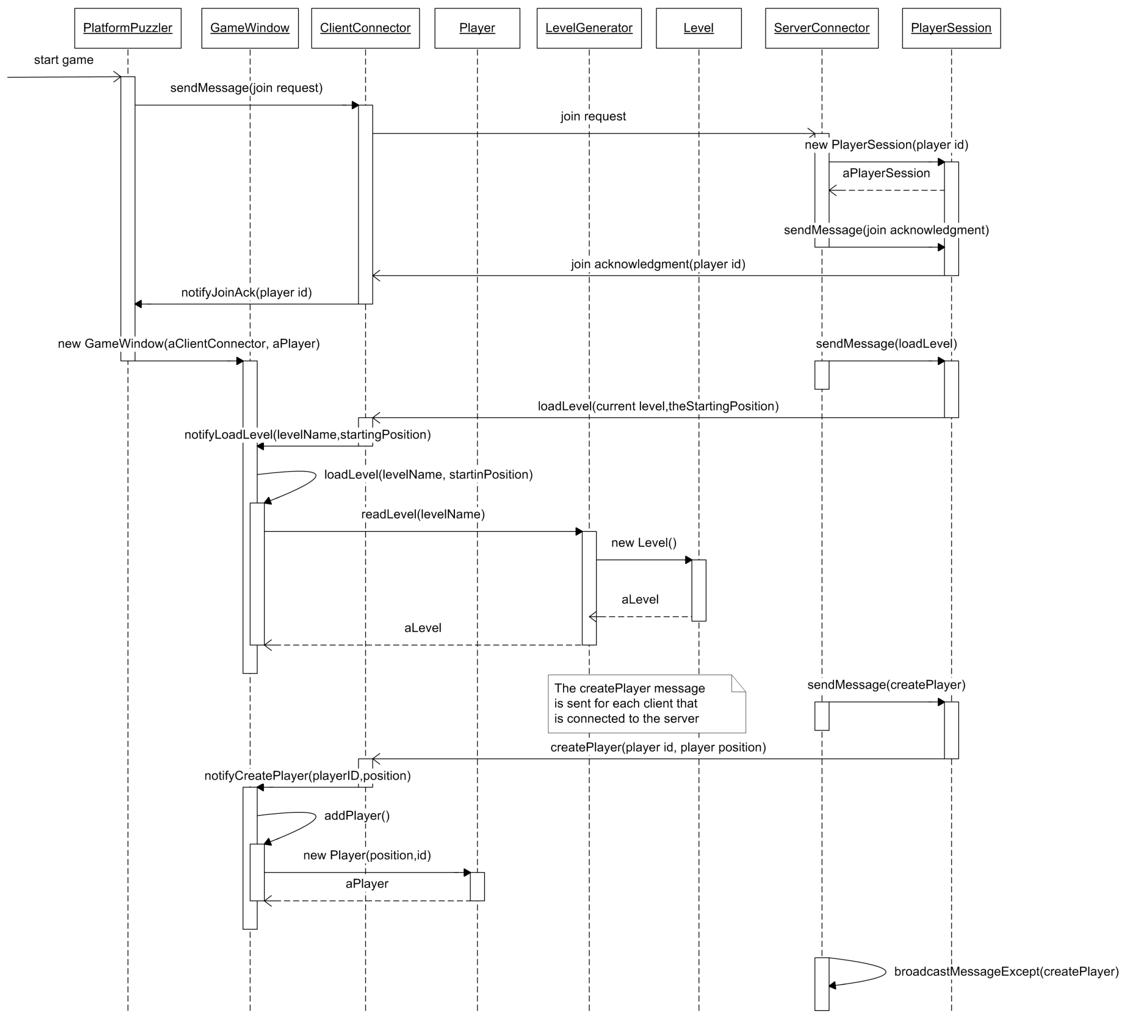

PlatformPuzzler
GameWindow
ClientConnector
Player
LevelGenerator
Level
ServerConnector
PlayerSession
start game
sendMessage(join request)
join request
new PlayerSession(player id)
aPlayerSession
sendMessage(join acknowledgment)
join acknowledgment(player id)
notifyJoinAck(player id)
new GameWindow(aClientConnector, aPlayer)
sendMessage(loadLevel)
loadLevel(current level,theStartingPosition)
notifyLoadLevel(levelName,startingPosition)
loadLevel(levelName, startinPosition)
readLevel(levelName)
new Level()
aLevel
aLevel
The createPlayer message is sent for each client that is connected to the server
sendMessage(createPlayer)
createPlayer(player id, player position)
notifyCreatePlayer(playerID,position)
addPlayer()
new Player(position,id)
aPlayer
broadcastMessageExcept(createPlayer)

Figure 9.19: Sequence diagram showing the process of a client connecting to a server

to, and press the connect button as shown in Figure 9.6(a). A **ClientConnector** object is then created and it is used to send a "join request" message to the server. The server in the meanwhile is listening to incoming client connections, and creates a **PlayerSession** object when the client has connected. The **PlayerSession** object then notifies the **Serverconnector** about the "join request" message, and the **ServerConnector** then sends a "join acknowledgment" message back to the client via the **PlayerSession**. The **PlatformPuzzler** is then notified about the message from the server by the **ClientConnector** (via the **MessageListener** interface described previously in this Section), and the **PlatformPuzzler** then creates the main game window that will render the game, run the game logic, and receive the messages from the server as of that moment instead of the **PlatformPuzzler**. **GameWindow** is that class, and it creates a new **Player** object with the identifier brought with it from the server in the "join acknowledgment" message.

In the diagram, the messages from the server are sent after the appropriate actions to the previous message are taken on the client side, however in reality the messages from the server are sent one after the other in succession and then handled in turns by the client, but in order to keep the diagram more readable the different server messages are divided more sequentially. The next message is the "load level" message, which is sent by the **ServerConnector** via the **PlayerSession** to the **ClientConnector**, which then notifies the **GameWindow**. This message contains both the name of the level to be loaded by the client, as well as the starting position for the player object representing the player in the game. The **GameWindow** then calls it internal method **loadLevel(...)** which loads the level with the specified name using a **LevelGenerator** which returns a **Level** object, and finally the method sets the starting position of the player and centers the camera, or viewpoint, around the center of the player. The **Club** object and the sprite associated with that object is also placed relative to the player.

The next message is a "create player" message. The server will send one of this message for each of the players connected to the server, excluding the client itself, and the client then creates player object locally and draws them on-screen. The **ServerConnector** sends the "create player" message, which also contains the position of that player and the identifier of that player, via the **PlayerSession** to the **ClientConnector** which notifies the **GameWindow** about the message. The **GameWindow** then calls the method **addPlayer(...)** which in turn creates a new **Player** object with the identifier and position contained in the message from the server. The player object is then added to the list of players and added to the **LayerManager** which handles the graphics and drawing of sprites. The position of the **Club** object of that player is also updated relative to the position.

The final action performed of the server is to notify the rest of the clients that a new client has joined the game. The **ServerConnector** thus broadcasts a "create player" message to all of its **PlayerSession** objects, except the client that joined, informing them that a new client has joined the game with a specified identifier, and that the client

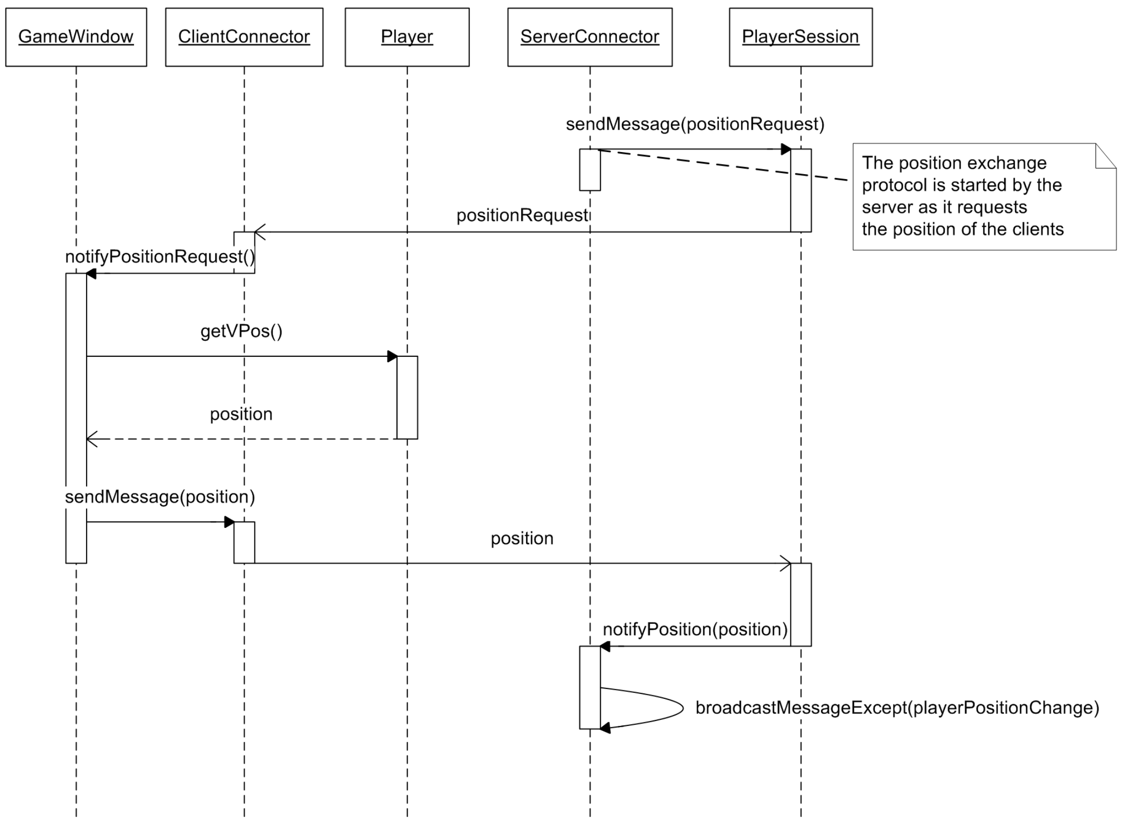

GameWindow
ClientConnector
Player
ServerConnector
PlayerSession
sendMessage(positionRequest)
The position exchange protocol is started by the server as it requests the position of the clients
positionRequest
notifyPositionRequest()
getVPos()
position
sendMessage(position)
position
notifyPosition(position)
broadcastMessageExcept(playerPositionChange)

Figure 9.20: Sequence diagram showing the server requesting the position of a player

should be drawn in their game worlds.

Updating positions

The **ServerConnector** keeps track of the positions of the players connected to the server, and the positions are stored in the **PlayerSession** objects. The way that positions are handled in the prototype is that the players are free to move around as much as they please, and that he server periodically requests the information about the position of each of the players connected to the server. Even though the players move locally by using speed vectors, the prototype does not use prediction of player movement, and does thus not send the speed vectors of another client to a client, but instead sends the most current position received by the server. This can lead to the observation of discrete movement for the player, but since the focus of the prototype is on the social side, this issue is not as important.

Figure 9.20 shows the sequence of the server requesting the position of a client. The **ServerConnector** has a thread that periodically requests the position information of its clients. The **ServerConnector** sends a "position request" message via the **PlayerS-**

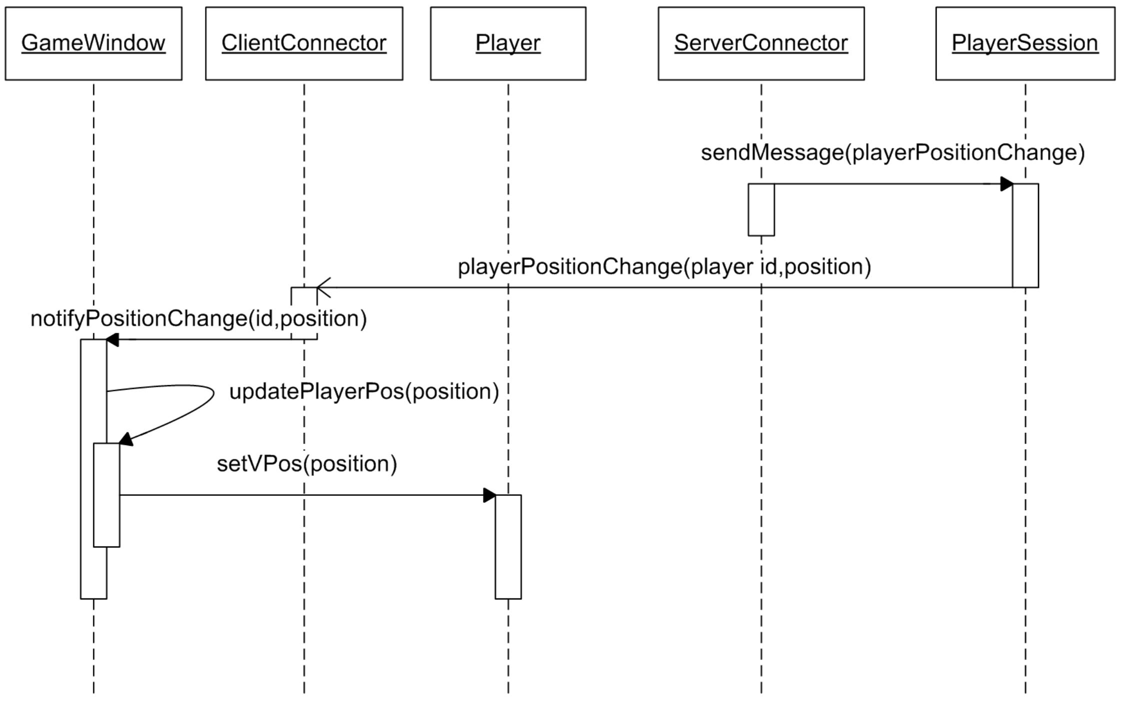

GameWindow
ClientConnector
Player
ServerConnector
PlayerSession
sendMessage(playerPositionChange)
playerPositionChange(player id,position)
notifyPositionChange(id,position)
updatePlayerPos(position)
setVPos(position)

Figure 9.21: Sequence diagram showing how the client handle a "player position change" message

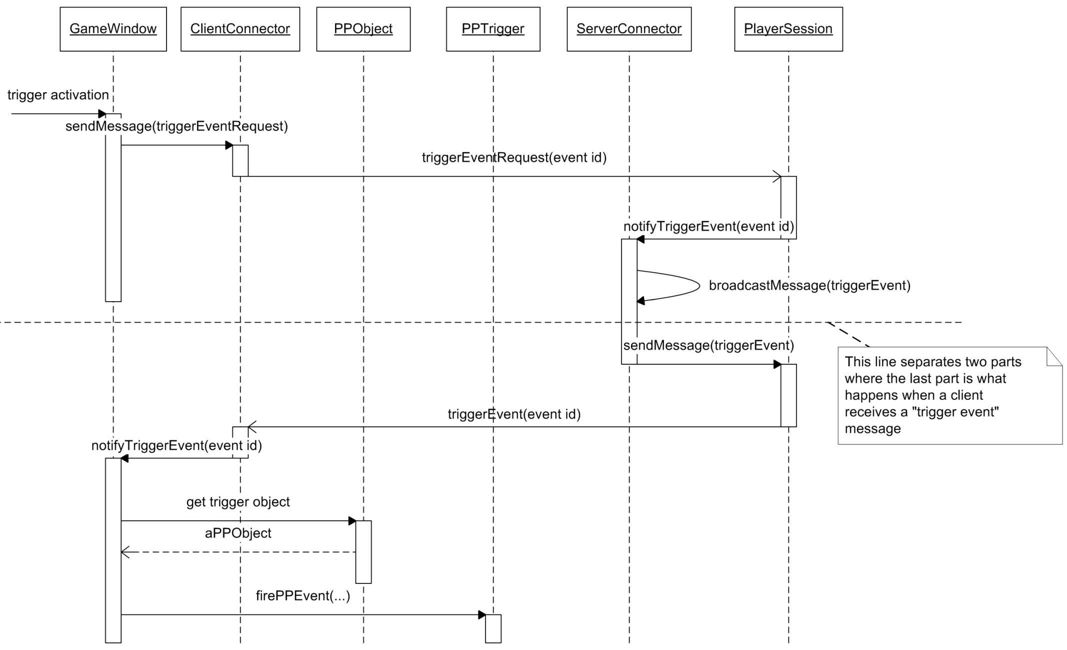

GameWindow
ClientConnector
PPObject
PPTrigger
ServerConnector
PlayerSession
trigger activation
sendMessage(triggerEventRequest)
triggerEventRequest(event id)
notifyTriggerEvent(event id)
broadcastMessage(triggerEvent)
sendMessage(triggerEvent)
This line separates two parts where the last part is what happens when a client receives a "trigger event" message
triggerEvent(event id)
notifyTriggerEvent(event id)
get trigger object
aPPObject
firePPEvent(...)

Figure 9.22: Sequence diagram showing how events are handled by the client and server

ession representing the client to the **ClientConnector** of that client. The **ClientConnector** then notifies the **GameWindow** that the server wants to know the position of the player. The **GameWindow** then retrieves the position information from its **Player** object, i.e. the object representing the client, and it sends a "position" message back to the server via the **ClientConnector**. This message contains the position information about the client, and it is received by the **PlayerSession** of that client on the server side, and the **ServerConnector** is then notified. The final action of the **ServerConnector** is to broadcast a "player position change" message to the rest of the clients, excluding the requested client. This message contains the new position of the client that moved.

Figure 9.21 shows how the client handles a "player position change" message. The **ServerConnector** broadcast was shown to broadcast the message above, and it uses the **PlayerSession** to send the "player position change" message, which contains the new position of the client with the specified identifier. This message is sent to a **ClientConnector**. Here the client is any of the clients connected to the server. The **ClientConnector** then notifies the **GameWindow** about the message, and the **GameWindow** then calls the **updatePlayerPos(...)** method, which updates the position of the player with the identifier retrieved from the message. The position of the **Club** object of that client is also updated.

Event system

The event system here means the system that handles the events and triggers related to level objects. An event is triggered when a player presses the corresponding button on the keypad that attempts to activate an trigger. If the player is in the vicinity of an level object that has an trigger, the client notifies the server about this. The client does not however trigger this event locally yet, but instead waits for the response from the server. The **GameWindow** sends a "trigger event request" via the **ClientConnector** to the server. This message contains the identifier of the trigger, and is received by the **PlayerSession**, which in turn notifies the **ServerConnector** about the message. The **ServerConnector** does not check the validity of the trigger id in the prototype, but instead broadcasts a "trigger event" message to all of its clients. This message contains the identifier of the event that should be triggered, and tells the clients that they should trigger this event locally.

The part of the diagram that is below the dotted line describes what happens for a client that receives the "trigger event" message. The **ServerConnector** sends the message via the **PlayerSession** associated with the client, and the **ClientConnector** of that client receives the message and notifies the **GameWindow** about the message. The **GameWindow** then uses the event identifier retrieved from the message to find the level object that has the trigger with that identifier. This object along with the event identifier itself is used by the **PPTrigger** to fire an **PPEvent** which is sent to all classes that implement the **PPEventListener** interface. The objects that are then listening to the exact identifier of the event then performs the action related to that identifier as specified in the script file, i.e. the script file associated with the current level.

Club mechanic

The club mechanic is, as described in the previous Section, activated when the player presses the button on the keypad assigned to that action. The client then starts the clubbing animation on the client side, however the animation is not shown for the other clients in the prototype. The main game loop checks to see whether the player's club is colliding with another player for each round of the loop, however it only checks this if the variable **clubbing** is set true in **Club** object of the **Player**. If the **GameWindow** finds that there is an collision with the club of the player, and that it also has initiated the club mechanic, the **GameWindow** sends a "club request" message to the server via the **ClientConnector**. This message contains the identifer of the player that the client has hot with its club, as well as the direction the client was moving when it hit the player, in order for the player to fly off in the right direction. The message is received on the server side by the **PlayerSession** associated with the client, and the **ServerConnector** is notified about the message. Currently in the prototype the validity of the club request is not checked, due to the fact that it is a prototype, and that the social aspect of the game is the most important part, however it could be a point of improvement in future upgrades of the prototype to check for the validity of the club request. The **ServerConnector**

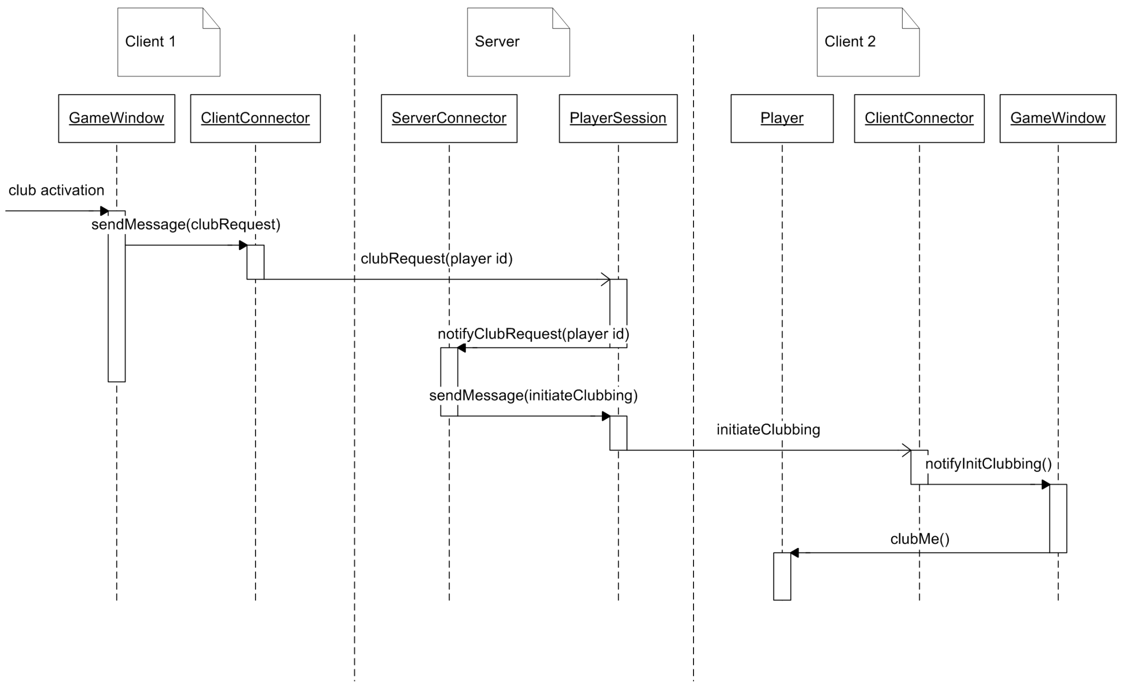


Figure 9.23: Sequence diagram showing the club mechanic

then sends a "initiate clubbing" message to the client that has the identifier retrieved from the message the server got from the first client. In the diagram the two clients are separated by the dotted vertical lines. The "initiate clubbing" message is then sent to the second client, and it is received by the **ClientConnector** of that client. The **GameWindow** of the second client is then notified about the message, and the **Player** object of the **GameWIndow** then activates the club mechanic on itself by calling the method **clubMe()**. The **Player**'s speed vector is then updated to be a standard value specified in the source code to be the default value of the club mechanic. This speed vector is reliant on the direction of the first client, and the x-component of the speed vector is set to go in the direction that the first client was moving in when clubbing the second client.

9.4 Testing

This section looks at the testing performed in this project[3]. The prototype game was tested on both emulators, and on the test phones, however the multiplayer was only only tested on emulators, due to network constraints on the test lab used. The test phone was a Sony Ericsson W810i, which is described in Section 4.2.1. The emulator used for testing were mainly the emulator created by Sony Ericsson and released in the Sony

[3]The README.txt file which is provided in the zip file attachment to this report explains how to play the prototype game and setup the server.

ID	Description	Imp	Impl
FR-1	Player	H	F
FR-2	Character movement	H	F
FR-3	Action triggers	H	F
FR-4	Events	H	F
FR-5	Actions	H	F
FR-6	Objects	H	F
FR-7	Object properties	H	F
FR-8	Object property, player collision	H	F
FR-9	Object property, trigger	H	F
FR-10	Object property, event/action	H	F
FR-11	Object property, vertical character movement	M	P
FR-12	Object property, static	M	P
FR-13	Multiplayer	H	F
FR-14	Game world	H	F
FR-15	Puzzles	H	F
FR-16	Character abilities	H	F
FR-17	Character ability, club mechanic	H	F
FR-18	Character ability, wall bouncing	M	N
FR-19	Character ability, player-on-player jumping	M	N
NFR-1	Client server	M	F
NFR-2	Multiplayer player amount	H	F
NFR-3	Client server disputes	M	P
NFR-4	Client disconnection	L	P
NFR-5	Puzzle understanding	M	F
NFR-6	Puzzle completion	M	P
NFR-7	Game understanding	M	P

Table 9.1: Table showing which of the requirements where implemented in the game

Ericsson SDK 2.5.0.2. The emulator in the Sun Java Wireless Toolkit was initially used, but the Sony Ericsson emulator had better performance, especially when the multiplayer code was added to the prototype, and the emulator was also more similar to the test phone(since the phone was a Sony Ericsson mobile phone).

The test process used in the project involved three parts: to continuously and incrementally implement code, then compile the prototype, and lastly to test for errors. It was easier to test the prototype after each increment than to created test plans and check for errors and bugs at the end of the development process. This was partly due to the small size of the project group, which made it easy to keep control of the code and thus make it easier to test for errors after each increment, but the relatively low implementation complexity of the prototype game also enabled the project group to use this method.

Table 9.1 shows a list of all the requirements for the prototype game, both functional and non-functional requirements, and it gives an indication of how important each requirement is, based on the description of the *Platform-puzzler* concept, and the description of the prototype game. The **Imp** column, or importance, shows this, and rates the requirements from low(L) to high(H), with medium(M) as the middle value. The **Impl** column shows whether or not the requirement was implemented into the prototype game, and gives three different degrees of implementation of the requirement in the game; namely not implemented(N), partly implemented(P), and fully implemented(F).

In addition to tests performed by the project group, the prototype game was also tested on users not part of the project group. This test looked more on the social side of the prototype than on the technical, and it is described in Chapter 11. There the prototype is also evaluated, but the requirements with "not implemented" and "partly implemented" values in Table 9.1, are further discussed below:

FR-11, Object property, vertical character movement: This requirement said that a object should allow a player to move vertically on it. This is however not supported at all in the prototype. The property was scrapped from the prototype due to difficulty implementing it, as well as time constraints within the project.

FR-12, Object property, static: This requirement said that a object should be able to move continuously. This requirement is only partly met since an object can be static, and they are able to move, but hey are not however able to move continuously from point A to point B, and back to point A in a loop, as was the intention of the requirement. This was due to the difficulty such a functionality would bring to the development with the time constraints of this project.

FR-18, Character ability, wall bouncing: This requirement stated that a player should bounce off of walls when it is being affected by the club mechanic. This is not supported at all in the current version of the prototype. It was however tested during development, but it was scrapped since it did not function as planned. This was due to the fact that the project team had difficulties with the collision detection, and thus an additional functionality that requires alteration of the collision detection routine could lead to more troubles than it was worth when considering the time frame of the project.

FR-19, Character ability, player-on-player jumping: This requirement specified that the players should be able to jump and stand on top of each other. This is not working in the prototype. The project team had as stated above difficulties with the collision detection, and this forced this specific functionality not to be implemented in the prototype.

NFR-3, Client server disputes: In the current prototype the server has not enough knowledge of the clients in order to fully support this requirement.

NFR-4, Client disconnection: The disconnection process is currently not functioning properly in the prototype, and disconnections can some times work, but other times they may cause the server to stall, and the clients that are left when another player disconnects may also experience problems after such an disconnect. The concrete source of the problem was not found, but it could partly be related to the way messages are handled in both the server and the client.

NFR-6, Puzzle completion: During the testing of prototype with testers not from the project group, the testers struggled to complete the game, and could not finish the level within the time limit specified in the requirement on the first run when they were not co-located. In the second run when they were co-located however they finished the puzzles well within the time limit. This is more thoroughly explained in Chapter 11.

NFR-7, Game understanding: During the testing of the prototype with testers not from the project group, the testers had some problems with the controls, and did not fully understand them within the time period specified in the requirement, however this could also be because of the prototype running on an emulator on a computer and not a mobile phone, thus the requirement is partly fulfilled.

All in all the prototype is functioning well enough to be used in the project, and the errors described above does not completely break the game, or make it unplayable. In a future version however, these faults should be investigated and improved upon.

Part IV

Evaluation

Chapter 10

Technology evaluation

This chapter looks at how the technologies used to develop and run the prototype has affected the game, both in runtime and how the technologies have affected the development. Only technologies that have had a noticeable effect on the development of the game are evaluated here.

10.1 Emulator

As described in Section 9.4 the project group initially used the emulator found in the Sun Java Wireless Toolkit. This emulator functioned well in the early parts of the prototype development, before the multiplayer network code was added. The addition of the network code lead to the emulator performing poorly: to the point that it could not read the player input. The network code allowed the client application to connect to the server application, which both where running on the same computer, with the emulator running the client application, but the game became unplayable since the players where unable to move due to the poor performance of the emulator.

The network code used in the client and server applications were similar to the code used in the previous project of the group, the depth study performed in the course TDT-4570 [Nøsterud, 2007], and the project group had similar problems with the network in that project. However, since the code itself did manage to connect the client and server, the project group wanted to try another emulator to check to see whether the problem lie with the emulator, or the network code itself. Since the test mobile phone was a Sony Ericsson phone, the emulator found in the Sony Ericsson SDK 2.5.0.2 was tested with the prototype game. The Sony Ericsson proved to perform better on the computer used to develop and test the prototype(which was the same computer used in the previous project) than the Sun emulator. With the Sony Ericsson emulator the game became playable, and the emulator could read keypad presses, allowing the testers to play the game. This indicated that the problem experienced in the depth study was mainly due to the emulator being used in that project. The discovery of this lead to the Sony Ericsson emulator being used for the remainder of the development period.

10.2 Network

The server and client communicated using messages of strings translated by a specific class in the game which controlled the syntax and message format, and the communication was done using TCP/IP. Because of the design of the levels used in the prototype, only two players were needed to solve each level, meaning that there were no reason to test the prototype using more than two players connected at once. The firewall and router setup on the computer used for testing the prototype however did not allow for the test mobile phone to connect to the server(which was running on the test computer). The prototype game was thus only tested on the emulators. The game did thus not experience any bandwidth or latency problems related to the network, and such problems can only be found by further testing. This issue however was not thought to be critical to the project group, since the focus of the project was more on the social side of the prototype, than on the performance.

10.3 Test computer

As mentioned above, the test computer used in test and development process could not be used to test multiplayer with the test mobile phone. The project group also believes that the emulator problem discussed above with the Sun Java ME emulator could lie equally with the test computer. The Sun emulator did not read keypad input well when the network code was added, but it performed well before this code was added. The Sony Ericsson emulator did also struggle on the test computer when the test computer was forced to run two separate emulators and the server. The game was still playable, but it was considerably slower than when running simply one emulator. This slowdown in performance when running two clients at the same time lead to the development team having to force certain constrictions on the network code. The most severe of these restrictions were to only update the player positions once or twice each second. This was due to the amount of messages that needed to be sent and parsed when the player position updating interval was lower than this, and as a result of this the emulators performed poorly. The project team acknowledges that the prototype code itself could cause some of the problems, but was unable to test this problem more thoroughly in the time period of the project. Further testing could find the source of the problem, be it the emulator, or the network code.

Chapter 11

Prototype evaluation

As described in Chapter 9, the prototype game *Platform-puzzler* is a multiplayer 2D side scrolling platform game, with focus on cooperative gameplay and puzzles.In this chapter the prototype will be evaluated, along with the test that was performed on the prototype.

11.1 Prototype testing and results

Section 9.4 describes the test process used in the development phase of the project, and shows how the prototype adheres to the requirements created for the prototype. This Section describes an additional test performed by the project group with the help of the supervisor. In this test people outside of the project group were brought in to test the game. The testers had no previous knowledge of the game, and had not played the game prior to the test session.

The test was conducted at NTNU, and two testers were found on campus. The test session consisted of two smaller play sessions, where each session were followed by the testers answering a questionnaire about the prototype game. The two testers were the same for both play sessions. It should be noted that since there were only two testers, the answers to the questionnaire and the information found during testing may not be generic, or statistically valid, but they may show indications on areas that should be further investigated, or even show areas that may need further research. Bugs and errors in the prototype game found during the testing though are valid regardless of how many testers there were, and these bugs should be fixed in future improvements of the prototype.

Figures 11.1; 11.2, and 11.3 show the testing environment, along with the two testers. The test was, as previously discussed, performed in two play sessions, where the difference between the two sessions were that in the first test the testers were not playing the game in the same room, something which is shown in Figure 11.1 and 11.2, while in the second and last test the two testers were playing the game in the same room, which is shown in Figure 11.3. The difference for the testers then was found in the communication avail-

Figure 11.1: One of testers playing the game alone

Figure 11.2: The other tester playing the game

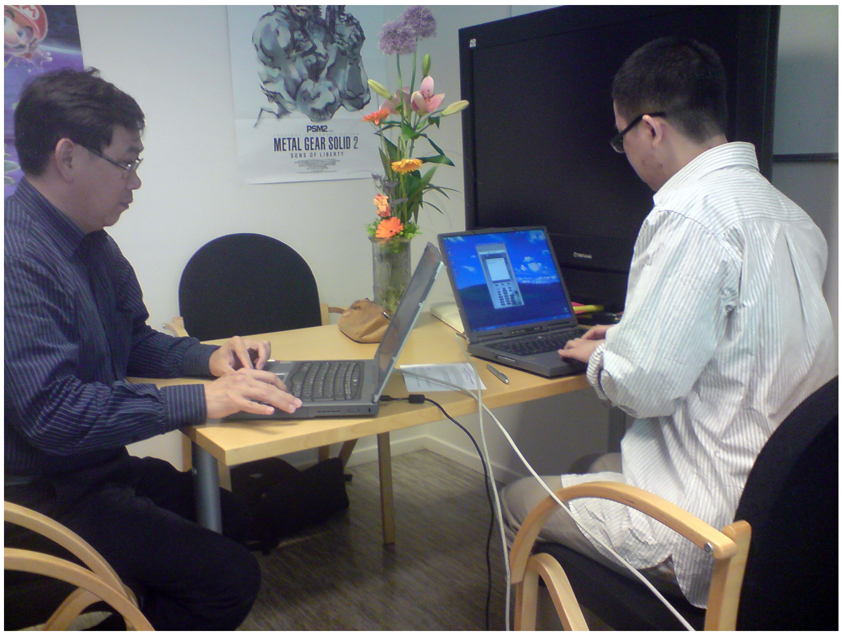


Figure 11.3: The testers are playing the game co-located

able for each tester with the other tester. When the two testers where not located in the same room they could not communicate verbally or textually with each other, since this functionality is not present in the prototype, but in the session were they where playing together in the same room they were allowed to talk to each other and assist each other with the game. During the testing the project group was interested in observing how the two testers played with each other, and whether they worked together as a team to solve the puzzles in the game. The project group was especially interested in observing whether there were any differences in how the two testers played the game when they were located in different, versus when they were located in the same room. The interaction between the players, both in the game and in the real world where then interesting in that respect.

The actual testing was performed, as discussed above, at NTNU June 3, 2008 using two laptop computers provided by the project supervisor. The two laptop computers where both outfitted with the newest version of Java as well as the emulator provided in the Sony Ericsson SDK 2.5.0.2. One copy of the prototype game ran on each laptop computer, and each prototype game ran on the Sony Ericsson emulator. The server, which used Java SE 1.6, was run on one of the laptop computers. The two computers where both located in the same network, and the difference between the two computers where then the hardware in the computers themselves(however they were both capable of running the prototype game well), and that one computer was simultaneously used as both client and server, although this did not bring any advantages in the prototype game to the tester running this computer other than slightly less network latency. The network latency between the two clients running the prototype game was at times moderately high, but it did not interfere to much with the actual testing, and the reason for the latency is not known as it could be either the network, the computers themselves(due to firewall settings etc), or the network code in the prototype game.

Before the first play session started the testers were briefed about the game, and told how to control the game as well as what the objective of the testing session was and what the prototype game was about. The two testers were then allowed to start, and they first tested the game by playing it in separate rooms. During both play sessions the two players were observed and timed to see if there were any differences between the two play sessions. In the second session the two testers were located in the same room and were allowed to communicate and help each other.

During both of the play session the two testers played the prototype game *Platform-puzzler* together in a level which was specifically created for this test. In this level the two players are forced to work together in order to solve all of the puzzles, and in order to reach the exit. There are several puzzles in the level, and one exit. All of the puzzles can all be solved using the basic game mechanics previously discussed in this report, namely by jumping, by pressing buttons, or panels, in the game trigger events, or by using the club mechanic. For the test there was only one level, and that level was used for both play sessions. The project group acknowledges that might lead to results that are affected by the fact that for the second play test the testers are already familiar with the level and the puzzles within it. The focus of this test however lies in observing and trying to understand how the players work together cooperatively, and whether there are any differences between playing it in the same room or in separate rooms. The project group thus believes that the results from the test can still give indications of the true nature of the prototype game. The results from the test are discussed later in this chapter.

After each of the play sessions the testers answered the questionnaire, which is further discussed below.

11.1.1 Questionnaire

The questionnaire in Table 11.1 was given to the two players at the end of each play session. The testers where given the questionnaire along with a short description on how to fill out the questionnaire. The questions themselves ask the testers what they thought about the prototype game itself, the concept behind it, specific gameplay mechanics within the game, and the social experience of the prototype for the testers.

The questionnaire is structured as a table with two columns and one row for each question. The questions itself is presented in the left column and is given a identifier on the form of 'Q#', where 'Q' stands for question and '#' is the unique number of the questions. The right column is used by the tester to answer the question. In the questionnaire there are three different ways of answering the questions, where each question has one answer form. Most of the questions have the numbers 1 to 5, i.e. '1 - 2 - 3 - 4 - 5', where 1 corresponds to 'strongly disagree', 2 corresponds to 'disagree', 3 corresponds to 'neutral', 4 corresponds to 'agree',and 5 corresponds to 'strongly agree'. The testers answer these questions by writing under the number that they feel best corresponds to their opinion. The second question type has a simple 'positive' or 'negative' answer option to the tester, which are used to indicate how the game affected them. Lastly there are some questions that have blanc answer fields. In these fields the testers are encouraged to write in free form the answer to the question, and about their opinion on different elements of the game.

The results from the two play sessions, both the observations of the project group and to the questionnaires, are discussed in the next section.

Question	Alternatives
Q1: I found that playing the game with another person enabled us to cooperate and work together to solve the puzzles	[1 - 2 - 3 - 4 - 5]
Q2: I think this concept is suitable for a mobile platform; for instance on a mobile phone	[1 - 2 - 3 - 4 - 5]
Q3: I would like to play a finalized game based on this concept	[1 - 2 - 3 - 4 - 5]
Q4: I found the puzzles suitable for a multiplayer game	[1 - 2 - 3 - 4 - 5]
Q5: I think the game would be more enjoyable to play if it contained more competitive elements, for example with a high score list	[1 - 2 - 3 - 4 - 5]
Q6: I found it easy to work together with another player to solve the puzzles	[1 - 2 - 3 - 4 - 5]
Q7: I think the experience of playing the game would change for the worse if I was to play this game alone	[1 - 2 - 3 - 4 - 5]
Q8: I found that the game supported cooperative gameplay well	[1 - 2 - 3 - 4 - 5]
Q9: Did you find that the game forced you to work together with another player, and if so, did you find it to be negative or positive	[Negative - Positive]
Q10: I thought the controls were easy to understand and use	[1 - 2 - 3 - 4 - 5]
Q11: I found the puzzles in the game easy to understand and solve	[1 - 2 - 3 - 4 - 5]

Continued on next page

Table 11.1 – continued from previous page

Question)	Alternatives
Q12: I found that the game mechanics available to me(i.e. the club mechanic, being able to jump, and being able to pull triggers) were suitable for a cooperative multiplayer game	[1 - 2 - 3 - 4 - 5]
Q13: Did you encounter any serious flaws, or bugs in the game; and if so, what where they?	
Q14: Did you see any room for enhancements to the game, either in terms of player abilities(like the club mechanic) or puzzle design; and if so, what where some of your ideas?	
Q15: If you found that there were elements of the prototype game you would want to change, or that there where elements missing, which elements would you want to change/add in order to improve a future version of the prototype game?	

Table 11.1: Questions to the testers ranging from questions about the game concept to the game mechanics

11.1.2 Results

As discussed above the test session consisted of two play sessions where two testers tested the game firstly in different rooms, then lastly in the same room. After each play session the testers answered a questionnaire. The observations of the player behavior as well as the game behavior are discussed below, along with general observations in the test sessions along with the results of the questionnaire.

First play session: different rooms

In the first session the two testers where placed in different rooms, as shown in Figure 11.1 and 11.2, with no way of communicating with each other, neither vocally nor textually. During the test the testers seemed to be hampered by the lack of communication options, and they had problems solving some of the puzzles. On puzzle in particular was troublesome for the players, since it requires that the two players work together and utilize the club mechanic to throw one player on top of a platform that can only be reached by using this mechanic. Initially the players had trouble understanding exactly how to solve the puzzle, but after some time they understood that they had to use the club mechanic. Then the problem became that they did not know which player should use his club, and which player should stand still and wait to be clubbed.

The group made an interesting observation during this particular puzzle where one of the player started to jump up and down on top of a platform. This platform was indeed the platform that the project group had designed to be used as a launching pad using the club mechanic of that puzzle. The player that jumped up and down repeatedly wanted to indicate to the other player that the other player should hit him with the club. This action indicated to the observers that because of the lack of communication between the players, the players themselves tried to created a way of communication by using the available game mechanics, similar to the freeform socialization described in Section 5.2. The player tried to create a means of communicating with the other player by using the tools given to the player, even though the project group themselves had not thought of the possibility of this specific game mechanic being used in such a way.

The other player though did not seem to fully understand the intention of the player creating this communication method though, and the players still had problems solving this puzzle. They did however manage to solve the puzzle, but they used relatively longer time than the project group had envisioned the puzzle to take.

Another problem discovered during this session was that the players sometimes had problems knowing where the other player was. The game allows the two players to move out of each others viewpoints, and it has no indicators showing a player where the other player is when this happens. This did at times make it harder to solve some of the puzzles in the game since they did not know whether to wait for the other player, or go look for him. A problem related to this was that because the players did not know initially how to solve the puzzle, they had to investigate and explore the level in order to figure out how to get to the next part of the level. When the players then were unable to communicate with each other they could not tell the other player whether they were exploring, or standing somewhere in the map waiting for them, and with no indication of where the players were it got even worse.

The testers used 24 minutes from the game was started until the game was stopped. The testers did not manage to finish the level due to a bug that lead one of the players to get thrown out of the map. Instead of restarting the level the project group decided to move the testers to the same room and start the second part of the test instead. This was due to the players being close to the exit, and because the play session had taken much longer than first anticipated by the group.

Second play session: same room

During the second play session the testers were located in the same room and were allowed to talk to each other and help each other with the game. The two testers immediately started communicating with each other. This communication was used to let each other know how they should solve each puzzle, where they should stand in relation to each other when using the club mechanic, in which direction the club mechanic should be used in each specific puzzle, deciding who should use the club and who should be clubbed, and generally giving advice and helping each other. Here the vocal communication was used to solve that arose in the first play session(and which was discussed previously): that of deciding who should use the club and who should be clubbed, and in which direction to club each other. When they could not talk to each other they were forced to use in game mechanics to try to hint to the other player that they wanted to use the club mechanic, or simply use the club randomly. Now when they had were able to talk they could communicate with each other and decide how to best solve the issue. The two testers talked to each other in every puzzle. After each action they performed they also informed the other tester about it, e.g. if they pressed a button in the game they would inform the other tester about it and thus they both knew what the other tester was doing.

Even though they were able to talk to each other, they still struggled with the same club mechanic puzzle in which they struggled in the first play session. They did however solve it much quicker since they were able to discuss with each other how best to solve the puzzle, and they agreed upon where they should position themselves in the level, and who should use the club. By communicating with each other they seemed to solve the puzzle much faster than they did in the previous play session, but for the project group it also indicated than in future work with the prototype this puzzle should be modified in order to make it easier to solve, especially since the problem the testers struggled with was not how to solve the puzzle, but where exactly to position themselves in order to use the club mechanic exactly correct. They understood relatively fast that the club mechanic should be used for this puzzle, but the actual solving of the puzzle proved harder due to the design of the objects in the level: the character of the players repeatedly hit an object in the level due to not standing exactly on the correct starting position when using the club mechanic.

Another observation made by the project group was that the testers would occasionally ask where the other player were in the level. This happened when the two players were so far apart from each other that they could not see each other in the game. This problem was also present in the first play session indicating that this is a problem that should be looked into for the project group. In terms of the second play session it was not a big problem since it could be solved by communication between the testers, but for situations where communication is unavailable this can be a relatively big problem, as previously discussed. This could be solved by having a visual indication of where the other players are when they are can not be seen on screen in the game. Arrows could for instance be used where the arrow would point in the direction of the player and be shown on the edge of the screen. This arrow would then indicate in which direction the other players are relative to the player, and also to some extent whether the other players are located higher, or lower than the player. Such an arrow would however not give an exact position of the other players. To give players the exact location of other players a mini map system could be used where in for instance one of the upper corners of the screen there is a small representation of the level, and all of the players could for instance be represented on this level representation as red dots. This mechanic is used in many games, but it would take space away from the player in which the player can not see the level.

The size of the mini map must also be large enough to enable the players to recognize the level representation, and differentiate the different players from each other in the mini map. This could mean that the map had to be so large as to be obstructive to the player.

The testers used 3 minutes to complete the test level in the second play session. That is 21 minutes faster than during the first session. Even when taking account for the fact that in the second play through they knew the controls better as well as the puzzles, this is still so much faster that it indicates that the addition of the possibility of communicating properly with each other helped the players a lot. This was also indicated from the testers where after the second test one of the testers said that after this test he was now much better able to answer the questionnaire, and especially the questions about the social experience of the game and the cooperation of the players. The project sees this a strong indication that there is a distinct difference in playing games in the same room versus playing in different room, at least in games where communication tools are not present.

General observations

The project group made observations that were not strictly related to the difference between the location of the testers in the play sessions. One observation relates to the way messages are handled by the server. Messages sent to the server by player 1 before player 2 has joined are not sent to player 2 when (s)he joins. In the play sessions this showed itself when a tester pressed a button in the level that caused a action trigger to trigger an event which was sent to the server, and the appropriate action was performed for this player. The other tester though had not yet entered the game, and when he entered the game the action that had occurred on the first player did not occur for the second player thus making inconsistencies between the two game worlds. The first player was then forced to press the button again in order to trigger the event for the new player. This is in reality a rather big error, especially when the players are not located in the same room and thus not able to tell the other player about it. It could be solved by storing important messages, like event messages, in a list on the server and use this list when a player joins to send the information so that the state of the game world for the new player is not different from the states of the other players in the game.

Another observation made was when one of the testers exclaimed *"I did not see the exit!"*. This occurred when one of the testers had entered the exit of the level, thus ending the demo. The players that said this was in fact nowhere near the exit when this happened, but the game still ended with a message saying that a layer had reached the exit and that the game was over. This was in fact intended by the project group so that the players that did not enter the exit would not have to go to the exit, but instead be brought to the new level instantly. For the player that went through the exit this was not a problem, but for the other player this was a strange experience, and had he not known that it was the end of the session and the level he would not have understood what had happened. This could be solved by giving visual clues or text when one of the players enter the exit, but it could still be strange for the players. Another solution would be to not load the new level until all of the players had entered the exit. This however would not have been possible with the current test level as it only allows one player to reach the exit. A re-design of the level would thus be necessary in order to support this.

The testers had more problems with the controls than the project group had anticipated. This could stem from the fact that the game was run on a laptop computer using an emulator and that the button layout thus became quite awkward. The project group could however have explained the controls better, and perhaps given the testers either a help screen in the game that described the controls and the basic mechanics of the game, or as a real world instruction paper that explained the basics of the game to the testers. On of the testers had problems with the controls for at least half of the first play session. How the controls work on a real mobile phone though can not be judged by this test since the difference between the laptop computer and a mobile phone is severe. The button layout also differs from mobile phone to mobile phone since the manufacturers can choose how to implement certain Java ME keypad actions since Java ME has generic action description in MIDP.

A problem observed by the project team was that of the graphical representation of a trigger in the game world. These triggers generate events, as described previously, and in the test level there are several such trigger objects which open up doors. These trigger objects are presented graphically as a panel with a red button on them. They do not however have separate graphical representations for when they are not pressed, versus when they are pressed. This had the consequence that the players would occasionally try to press a trigger several times and since in the current version of the prototype such triggers can only be triggered once, nothing happens when they do this. This happened especially often when they reached a trigger that they did not press themselves(even though some times they also attempted to press such triggers), and also when they reached a trigger object that had previously been pressed by another player when they were not close enough to this trigger that they could see the action that corresponded to the event triggered by the object, e.g. they had not seen that a door had been opened by the other player pressing this trigger object.

A solution to this problem could be to have a visual difference between the two states that such a trigger object can be in. For instance when the object has not been triggered, the red button on the panel could stand further out from the panel than when it has been triggered. When a player then triggers it the red button could be pressed in. Visual clues that were clear could also be used, e.g. by destroying the trigger object when a player presses it, or for instance have the trigger object look like it is malfunctioning by for instance animating it so that there seems to fly electric sparks out of it. Audio clues could also be used for the trigger objects such that the object plays a different when a player can trigger it, versus when the player can not trigger it. Thus the player can relate a sound to a door opening, and may thus conclude that when another sound than this is played when (s)he presses the button, the trigger object has already been triggered.

The project group observed several bugs during the test sessions, some of which were known in advance. The first bug that was observed was that when a player walks a object that has moved from one position to another in a level, the player will get stuck on the edge between a non-movable object and the object that previously moved. This was known in advance and is caused by a fault in the collision detection. A solution to this problem was not found during the development and it was thus left in the game.

Another bug that was found during the test sessions was not known by the developers. The bug happened when a player was jumping and was hit by another player with the club. When this was done in certain areas of the level the player that was hit with the club was thrown out of the bounds of the level. The player was then falling endlessly and could not come back to the inside of the level, thus forcing the player to quit the game, and it could also make the level unsolvable for the other players since the level requires more than one player in order to be solvable. This was what happened at the end of the first play session, and cause the session to be ended. A solution to this problem could be to redesign the level by setting up high wall around the level such that a player could not jump out of the level. Another solution could be to check whether the player is within the boundaries of the level, and if the player then falls out of the boundaries of the level reset him(her) to for instance the last position of the player before (s)he fell off the level.

The difference of skill level between the two testers in the test sessions was relatively high, were one of the testers had played games before, while the other did not have much experience in playing games. This fact did indeed show itself during the test sessions were the more skilled player understood the concepts of the game quicker, and understood better how to control the game and how to use the different game mechanics available to the testers(i.e. the club mechanic, the character movement, and the ability to trigger certain objects). It also showed itself in that the more skilled player took more control of the session, especially in the second play session, and that tester tried to take control of the situation and navigate the two testers through the puzzles. The difference of skill between the player became less apparent as they became more familiar with the controls and when they became more able to understand how best to use the game mechanics available to them though. Nonetheless it shows a interesting social behavior of players that are more skilled than other players where they sometimes attempt to take control of the situation, perhaps because they feel that in that way they stand a better chance of completing the objective at hand. This behavior could be interesting to investigate further in future research.

Another observation was made by the project group that involved the difficulty of understanding how to solve the puzzles. This was especially evident in the first play session where they could not discuss how to solve the puzzle. At certain points in the level the testers seemed unclear about how to solve the puzzles. This is to be expected however since they are not fully able to understand how to use the game mechanics properly after having just started to play the game. The project group however could still do a better job as to indicate how to solve the puzzles. This could be done subtly in the game, for instance with painted arrows that indicate the route the testers should take, or basic drawn out instructions on how to solve a specific puzzle, for instance by drawing a club poster next to a puzzle that requires the player to use his(her) club. Another solution could be to measure how long the players take to solve a puzzle. If the players then use more time than had been initially intended by the developers, the game could give visual or audio clues to the testers, making it easier for them to understand the solution to the puzzle.

The problem could also stem from the fact that the level was designed in such a way that it was to difficult to understand for new players. It may be difficult for a new player to understand that (s)he must use a specific game mechanic to solve a puzzle if (s)he has never used this mechanic to solve a puzzle before. The developers should then possibly introduce the game mechanics to the players in a less difficult way by for instance introducing only one new gameplay mechanic for each level, and at the start of each level explain how to use the mechanic, and by guiding the player through the level indicate to the player in which situations (s)he should expect to use this mechanic. By introducing the game mechanics in such a way, the developers could create increasingly complex puzzles where in the final levels all of the game mechanics would have to be used together to solve the puzzles. In these more complex levels then the player would have a better chance of understanding how to solve the puzzles since they had experienced similar situations previously in the game.

Questionnaire

Because the two testers answered a questionnaire after each play session there are four answer sets. The interesting part is that there are two ways of looking at the questionnaires. The first ways is to simply look at the answers to the questions in order to gain some insight into how the game works, but the questionnaires can be checked for differences between the two play sessions thereby trying to see whether the opinions of the testers changed in the two different situations. Due to the low sample size however, no true statistic results can be drawn from the questionnaires, but they may show areas that can be further investigated(as discussed previously). The questions in the questionnaire from Table 11.1 themselves are discussed sequentially below:

Q1: This question asked the testers whether they found that playing the game with another person enabled them to cooperate and work together to solve the puzzles. All four answers gave this question a 5, or strongly agree, thus indicating that the testers felt that playing the game with another play was a cooperative experience, and that this did not change between the sessions.

Q2: This question asked the testers whether they thought that the concept was suitable for a game on a mobile platform. All of the testers gave 5 as an answer, thus indicating that they indeed thought this to be the case. It also indicated that the two different environments they played the game in did not change this fact.

Q3: This question asked the testers whether they would like to play a finalized game based on this concept. All of the answers were a 5, and they did not change between the play sessions. This indicates to the project group that the concept for this game may be worth continuing and perhaps take some of the points found during this test session and enhance the game further.

Q4: This question asked the testers whether they found the puzzles suitable for a multiplayer game. On this question there was a slight difference between the answers of the two play sessions. In the first play session the average of the answer was 4.5(i.e a 4 and a 5), and in the second the play sessions the average answer was 5(i.e. two 5's). Although the difference between the two play sessions is small, and that a 4.5 still means that the testers agree with the question, it still shows that the testers found the puzzles slightly more suitable for a multiplayer game after having playing he second time. The reason behind this though could be both because they understand the game better the second time they played it, or that the communication in the second play session made it easier to solve the puzzles.

Q5: This question asked the testers whether the though the game would be more enjoyable to play if it contained competitive elements. As with the previous question this question slightly different answers between the two play sessions. In the first session the average was 4.5, and in the second the average was 5. Firstly this indicates that both the testers after both play sessions feel that the game would be better or more fun if there had been competitive elements in it. This was not the focus of the prototype, but the project group acknowledges that competitive elements could make the game more fun, or bring more incentive for the players to play the game, and make it more rewarding for the players as well. This is something that could be added in future iterations of the game. The second interesting observation was that the average answer score changed between the two play sessions, and that it changed to be a higher score in the second play sessions. This may indicate that when the testers play the game in separate rooms the fact that they are unable to communicate easily makes the game more difficult, and that they then do not require as much competitive elements in the game, while in the second session where they play in the same room the communication options makes it easier to them and thus the need for competitive elements gets stronger. These are just thoughts though since the changed was very small, and the results from both play sessions indicated that the testers would have liked to have more competitive elements in the game.

Q6: This question asked the testers whether they found it easy to work together with another players to solve the puzzles. The difference between the answers after each play session for this question was higher than that of the previous questions. For the first play session the average score was 3.5. This shows that even though the testers were able to play the game, they had problems working together with another player to solve the puzzles when they were located in separate rooms. This was also observed by the project group during the play session. The score for the second play session was 5, thus indicating that when the two testers were in the same room they found it easier to work together and solve the puzzles. This indicates to the project group that there is in fact a difference between the two situations, at least in this game.

Q7: This question asked the testers whether they thought that the experience of playing the game would change for the worse if they had to play the game alone. The average score for the question was 5 for both play sessions. This indicates that the prototype game adheres to the focus of the project group for the game, i.e. to create a prototype game where the focus was on cooperative multiplayer gameplay.

Q8: This question asked the testers whether they thought that the game supported cooperative gameplay well. The average score for this question was 5 for both play sessions, and like the previous question(Q7) it thus indicates that the prototype game adheres to the main focus the project group had when developing the prototype.

Q9: This question asked the testers whether they thought that the game forced the testers to work together and if so whether they found it to be positive or negative. The answers to the question from the testers were all 'positive' thus indicating that the game indeed forced the two players to work together, which was the aim for the project group. It also indicated that the testers found this to be a positive experience, which could further lead the project group to believe that the concept behind the prototype game is sound.

Q10: This question asked the testers whether they though the controls were easy to understand and use. Here the answer scores differed the two play sessions again with the average score of the first play session being 3.5(where the scores were 3 and 4), and the average score of the second session being 4(where the scores where 4 and 4). This firstly indicates that the testers did not fully agree with the controls being easy to use and learn, thus meaning that the controls were not perfect. It could also indicate though that the project group had not explained the controls to the testers good enough, which was previously discussed in the observation part of this section. Secondly it indicates that the testers found it easier to control the game in the second play session. This is most probably because the testers had become more used to the controls by the second session, and thus it was easier for them to control the game then.

Q11: This question asked the testers whether they found the puzzles in the game to be easy to understand and solve. Here the average scores of the questions differed between the two play sessions. For the first sessions the average score was 3.5(with a 3 and 4), where for the second sessions the average score was 4(with two 4). This indicates that the testers found the puzzles to be slightly easier to solve during the second play session. This could be because of the communication that occurred between the two players in the second session, which was observed by the project group as well. It could also indicate that the puzzles themselves are not perfect since the average score for both sessions were below 5. Even though the testers mostly agreed with the puzzle4s being easy to solve, there might still be room for improvement in the design of the puzzles in future work with the prototype game.

Q12: This question asked the testers whether they found the game mechanics available to them were suitable for a cooperative multiplayer game. For both play sessions the average score of the question was 4.5(with a 4 and a 5). This indicates that the testers found the game mechanics to be suitable for a cooperative multiplayer game, but also that there is room for improvement in future work with the prototype since none of two average scores were 5.

Q13: This was an open question that asked the testers to write down any flaws or bugs they found while playing the game. One of the testers wrote about the bug involving jumping and using the club mechanic on a player that is jumping, which was previously discussed previously in this section. This further indicates that this is a bug that should be fixed.

Q14: This was an open question that asked the testers to write about their thoughts of enhancements to the game in terms of player abilities, or in puzzle design. One of the testers suggested that the club mechanics to be changed to also allow for clubbing other players vertically into the air. Currently the club mechanic can only be used to club players left or right. The project group acknowledges that this could indeed add to the game, and it would give possibilities to create new puzzles around the club mechanics. These new puzzles would be similar to the old puzzles revolving around the club mechanic in nature, and they would thus feel similar to the players.

Q15: This was an open question that asked the testers whether they found any parts of the prototype game that they wanted to change, or whether there were anything they would like to add to the game in order to improve it. One of the testers suggested that the developers could add monsters to make it more difficult, and more interesting. These monsters would then be controlled by the server and the players would have to either bypass them, or disable them in some fashion, for instance by using the club to beat them. The project group acknowledges that this could add more incentive to the player for playing the game, similar to adding more competitive elements previously discussed, but for this project this was not the main focus. The focus of this project, as previously discussed, was more on the social and cooperative side of the game. In future iterations of the game however, server controlled enemies would be a great addition that could make the game more exciting to play, and an element of difficulty not directly tied to puzzles.

11.2 Prototype evaluation

The finished prototype game is a 2D side scrolling platform game with puzzle elements with a focus on cooperative multiplayer where two or more players play in the same level and help each other to finish the level and continue to either the next level, or the end of the game. The main focus for the group was to create a prototype that allowed the group to see how players can work together in a cooperative game, and whether such a game can be made for a mobile game.

The prototype game in its final state seems to have reached the goals of the project group, at least in terms of it being a game that requires two or more players to work cooperatively together in order to finish the game. The observations made during the testing session and the questionnaires answered, which were discussed in the previous section, also indicate that the game itself has reached the goals of the project group, and it allowed the project group to test some of the questions of the project group. During the test session the game also indicated that there were differences in the experience of players that play the game in the same room versus players that play the game in separate rooms. This may indicate both that there exist a difference in how players play games when they are in different location situations, which should be further investigated, but it may also indicate that the games themselves should be created with this in mind. The project group could for instance attempt to implement tools in the games that helps the players to overcome the differences. Another issue that could be further investigated is whether the players themselves play games differently when they are located in the same room, versus when they are located in separate rooms. Will a player for instance be more helpful when (s)he is located in the same room as another player, and does that mean that a player is less likely to help a player if they are not located in the same room? These and similar questions could lead to further research in the area.

From a technical viewpoint the game could have performed better. Even though it was only meant to be a prototype, and that the focus was not on the technical side, or on the performance of the game. The project group found some bugs during the test sessions, which are described in the previous section. There were also game mechanics and features that the project group wanted to add to the game, but were not able to; either because of time constraints or because of implementation problems(which is discussed more thoroughly in Section 9.4). These problems did however not make the game unplayable, and it was still able to give the project group an insight into the problems that it wanted to look into, such as the cooperative part of the game.

The project group was also unable to test the prototype on a real mobile device, such as a mobile phone, but used a emulator instead to test out the game and the concept. Since the focus of the project was not performance though, this is not a big problem since the emulator still allowed the project to test out the game. In future work with the game however a real world test situation using a mobile network and a real server should be considered since it will help with creating a better performing multiplayer game in terms of network capacity.

Based on the test sessions and the evaluations in this chapter the project group claim the prototype game to be conceptually sound since it followed the concept behind the *Platform-puzzler*, and it allowed the project group to investigate and research the cooperative and social nature of mobile games within the context of this project.

Part V

Summary

Chapter 12

Conclusion

This chapter seeks to answer the research questions identified in Chapter 2, as well as concluding the work done in the project and the findings of the project.

12.1 Answers to research questions

1. How does player co-location(i.e. an environment where two or more people are located in the same room) affect the experience of playing games?

This research question tries to look at co-location and how it affects the experience of playing games.

(a) Are some game concepts more suited for co-located environments than others?

As the test session of the prototype game created for this project shown, which was discussed in Chapter 11, there are at least some game concepts that are easier to play when the players are located in the same room. When the prototype game was tested with testers from outside the project group the project group observed differences between the situation where the testers were located in the same room, versus when they were located in the same room. This indicates that some game concepts are more suited for co-location than non co-location. Then you can also argue that games in for instance the first person shooter genre may not gain much from enforcing co-location since these games are typically more played in non co-located situations than co-located situations. The answer to the question then becomes yes, since a game that is purposely created to be played co-located will most likely be better at it than games that were not created specifically for this.

The test session of the prototype game also shows that games that were created to enforce cooperative multiplayer, where the players are forced to help each other to finish the game, will work better in a co-located situation than a non co-located situation when the games do not support communication within the game. The observations made by the project group showed this quite clearly since the testers completion time became much smaller during the play session where they were located in the same room, so much so that the difference seemed more likely to come from this change in co-location rather than the skills of the testers themselves increasing. If a game has good communication tools included in the game however, this might change. This was not shown in the tests done in this project however, and can not be fully concluded here.

(b) Can some game concepts affect the experience of playing the game differently depending on whether the players are co-located or not?

During the test session discussed and described in Chapter 11 the testers where asked questions regarding the nature of the game after having played it. The testers answered these questions after both sessions giving the project group the opportunity to look for differences between the two play sessions[1]. The answers the testers gave after each of the test sessions showed that the testers found the it easier to play the game and work together with another player when they were located in the same room, and also that they found the puzzles to be easier to solve and understand when they were in the same and were able to talk and discuss with each other the solutions to the puzzles, and how the players should work together to solve them. The observations made by the project group during the test sessions also showed that the testers seemed to solve the puzzles easier when they were located in the same room.

This indicates that there are in fact concepts that can affect the experience of playing games differently depending on whether the players are located in the same room, versus being located in different rooms. Due to the small sample size(being only two testers) though this may not be a general fact, but at least for this concept and in these testing conditions the difference was quite clear to the project group.

2. How do gameplay mechanics affect social gaming?

This research question tries to see whether the gameplay mechanics in a game can affect the social experience of the game.

(a) Can social gameplay mechanics in games affect the experience of the player?

The social archetypes and game mechanisms discussed in Section 5.2, which were identified by Richard Bartle and Shannon Appelcline, shows that the availability of social mechanisms in a game may help a player archetype express his(her) interest. The socializers discussed by Bartle for instance may find it very difficult to socialize with other players if there are no gameplay mechanics available in the game that support socialization(which could be as simple as a chat system with textual messages). The different gameplay mechanisms can also affect each other. A player that is for instance only interested in fighting against other players in a game, i.e. the killers, may be a nuisance

[1]In the play sessions one session was performed with the testers located in different room, whereas in the other sessions the players were located in the same room.

for an explorer, i.e. a player that likes to explore the game world, since the game might allow the killer player type to kill the explorer. Here the social mechanism that allows a player to kill other players is affecting another player in a negative way. The gameplay mechanisms can thus affect the players both positively and negatively, and the developers must then strive to create a balanced game where the players have mostly positive experiences, but that they also ensure this does not feel restricting to the players. In the previous example for instance the game could have certain areas that players were allowed to kill other players, like it is done in *World of Warcraft*.

(b) What game mechanics can be used to make mobile games social?

In Section 5.2 several mechanisms suggested by Appelcline are discussed. Appelcline suggests for instance agreeable battles in which to players agree to fight each other, which was discussed in the previous research question as well. Other mechanisms include such mechanisms as having auctions were players can sell and trade virtual game items, and having player votes where the players votes for game rules, which maps to play on in shooter games, etc.

The mobile games that were investigated in Chapter 6 also showed some of the social mechanisms that exist in some games today. Some of the games in the chapter have online leader boards with player rankings where the players can enter the scores they achieved in the games and thereby test their skills against other players. Another feature found in some of the games were buddy lists which allowed the players to save a profile for the players that the meat during the multiplayer game sessions. These buddy lists can then be used to keep track of players, to send instant messages between the players in the list, or to send game invites directly to the players on the list. Some of the games also had an emote system that allowed the players send taunts to the players which could for instance show the emotions of the player, or be used to gloat.

These are just some of the gameplay mechanisms that can be implemented into games to make playing the games a more social experience for the players.

3. What game genres are most suited for mobile collaborative games?

In Section 5.3 several different game genres are discussed, and the social aspect of them are also discussed. Generally there are not many genres that can have a form of collaborative element to them at all. Some first person shooters for instance have a coop mode where two or more players can play together either through the original single player mode, or through a special dedicated coop mode. First person shooters are usually considered to be competitive and often have mutliplayer modes where the players compete against each other and kill each other for points. They also however have team based modes where they work together as a team to kill players on other teams, or to complete in game objectives such as planting bombs in the game *Counter Strike*. This shows that while a genre like the first person shooter genre which is considered to be a highly competitive genre in multiplayer, still has elements of collaborative and cooperative gameplay in them.

The other genres discussed in the same section also have, or at least have the opportunity to have, elements that support collaborative and cooperative gameplay. Team based modes is very common in many multiplayer games, and in such modes the players have an advantage if they are able to work together. Whether they actually want to cooperate with the players on their team is another matter completely though. Since all of the game genres have elements of collaboration in their multiplayer in some form it is difficult to say that a game genre is more suitable than another. It may even be easier to look at differences between individual games, and what the vision of the people that created the games had. A game could be meant to have only single player for instance, or that it was intended not to have collaborative elements at all. A single game does not equate to the whole genre though, and even though not all games in a genre have collaborative elements, does not mean that games in the genre could not, or should not have collaborative elements. There are therefore no genres that are completely unsuited for collaborative games, and rating the genres on how collaborative they are might not even be as interesting as looking at individual games.

12.2 Conclusion

The goal of this project is to explore new game concepts for mobile and social games where the games should either be real-time or asynchronous mobile multiplayer games where several players interact using the mobile network. In addition a prototype that enabled the project group to help test some of their research questions was also to be built, and the focus of the project was slightly more on the social side of gaming than on the technical or performance side.

The prototype , *Platform-puzzler*, was conceptually sound as it allowed the project group to test some of their research questions, and during the test sessions described in Chapter 11 it showed that it enabled the project group to test the cooperative nature of the game, and how players are affected when they are located in the same room versus when they are located in different rooms. The prototype game indicated to the project group that there in fact is a difference between the two situations, and that this difference should be considered when developing cooperative games.

Even though the prototype had some bugs, and that it lacked some of the mechanisms originally planned for the game, it still echoed the idea of the underlying concept that social games can be created on a mobile platform, and that it can enable cooperative gameplay between players.

The other concepts discussed in this project also show the possibilities of games on mobile platforms. Mobile games have unique experiences, but also unique possibilities by enabling mobility to factor into the gameplay, as well as fact that the different technologies found in the mobile devices can be used to enhance the gameplay and experience of playing the game further. Mobile games also utilize and incorporate social gameplay mechanics into them, and there are still more areas where social mechanisms can improve the experience of the players in multiplayer games, as well as new mechanisms that can be implemented in games. The concepts described in Chapter 7 and Chapter 8 show only some of the possibilities with mobile and social games.

Chapter 13

Further work

This chapter looks at improvements in the form of ideas of new functionality as well as improvements to the existing game, that the project group feels will lead to a better prototype game.

13.1 Improvements to Platform-puzzler

The first step in the future work with the prototype game would be to look at the bugs found during the test session discussed in Chapter 11, as well as look at how the game deviated form the requirements found by the developers, which is described in Section 9.4. The bugs should be fixed, but the deviations from the requirements are not so clear. Some of the deviations stem from features originally planned that were left out of the prototype game, such as players bouncing off walls. If these features were to be added the game would have to be further tested to see whether they fit the game, and whether they function properly. The project team experienced some problems with the collision code and this would also have to be fixed in order to implement the features that were left out.

The performance of the game in multiplayer should also be tested further so that it performs better, has less latency and perhaps also require less bandwidth. Further tests to see how the game performs on mobile networks should also be performed since this was not done during this project due to time constraints.

A problem that was found during the play testing was that the testers found the puzzles in the game to be somewhat difficult to understand, especially when they played the game in different rooms. A solution to this problem was discussed in Chapter 11 where some of the solutions discussed were to create introduction levels that introduced each new game mechanic to the player such that the player could become more acquainted with the game mechanics, and thus find it easier to use them in more difficult puzzles. Having in game clues was also discussed, such as signs that explain certain puzzles. Having arrows that show where the other players are located when they are not on the players screen was also discussed, since this was a problem for some of the testers.

The test level itself should also be evaluated, especially since one of the puzzles proved to be difficult for the testers to complete due to a slight design error by the project group. New levels could also be created that introduced increasingly more difficult puzzles. If new gameplay mechanisms where to be introduced the developers would also have to create new levels that contained puzzles that used these mechanics. The levels could also be tailored to suit more than two players, and even require more than two players.

The testers wrote on one of the questions in the questionnaire that they would like to have more competitive elements added to the game, both to make the more interesting as well as increasing the difficulty. One of the testers suggested adding monsters to the game. These monsters could for instance try to stop the players from completing the levels. The player could for instance use his(her) club to hit the monster, or a new gameplay mechanic could be introduced. Borrowing an idea from the *Super Mario* games the players could jump on the enemies to remove them from the game for instance.

Other competitive elements could also be added, and these elements could incorporate non violent elements such as a score system. The game could add scores either such that the players gained individual scores thereby increasing the competition between the players, or the score could be team based with an online high score list that ranked the score of the different teams. Individual scores would add a competitive element to the cooperative nature of the game thereby making the players both team mates and opponents. This could lead to a more exciting experience for the players where they could decide to get for instance sabotage the cooperative puzzles in an attempt to get more points for (her-)himself.

The score itself could come from for instance in game objects that the players would have to collect, similar to the coins found in *Super Mario* games. The score could also come from other sources, e.g. from killing monsters if monsters were included to the game. The players could also gain points from completing puzzles and finishing the levels. The levels could then have different scores where some of the scores were individual whereas other points were cooperative points which were given to all the players that completed the objective bound to it. This could carry out to level design as well by for instance having multiple paths though a level were one path required teamwork from two or more, where another path could require less teamwork, but have a higher amount of objects that gave individual points. The players would then have to choose whether to play the game cooperatively, or whether they wanted to be more individual. The points given for each of these paths should therefore be carefully chosen as to encourage both cooperative players and individual players.

13.2 Further research

Additional testing of the prototype could give further insight into social gaming, and cooperative gaming. New ares of interest could be cooperative versus competitive gaming, and how best to incorporate the two into games and how they affect players, and whether they affect players differently. If the prototype was to be updated with some of the improvements suggested in the previous section, additional testing would also help in testing the new game as well as give further insight into the research questions of this project as well as potential new research questions.

The test sessions discussed in Chapter 11, and the indications found during both the observation made during the play test sessions by the project group, along with the answers from the questionnaires, should also be investigated further. A larger study could also be performed. Since the test session only had two testers the results found during the test session was mainly indications, and a larger study could help give more definite answers to the questions raised in this project, along with the observations already made during the first test session.

13.3 Improvements to the other concepts

The other concepts described in this report could also be further investigated and prototyped to identify new functionality suited for the concepts as well as discovering the value of the concept itself, and whether the concept is worth considering. Some of the concepts may give additional insight into the questions raised in this project, as well as give additional insight into social and mobile games. Some of the concepts conceived and described in this report could also be used together, and use components of each other to form new concepts.

Chapter 14

Recommended readings

This chapter notifies the reader of articles or books read by the project team which will help interested readers in further understanding the source material and resource material found during the projects lifetime. The following list shows some of the sources of information used in this project.

Programming Wireless Devices With the Java 2 Plattform Micro Edition, Second Editio by Roger Riggs et al. is a book that describes the Java ME architecture in a easy-to-understand way. It describes the various parts of Java ME, and describe in more detail the API's of CLDC and MIDP, and show the differences between the various versions of the API's. The book also shows many relevant examples on how to use the various parts of the CLDC and MIDP framework and describe how to create MIDlet's using MIDP.

HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS by Richard Bartle is an excellent article discussing the various social behavior of people who play MUDs, or Multi User Dungeons. The article discuss the normal behavior of such players as found by the author, which himself has been involved in the creation of various MUDs.

Social Gaming Interactions, Part One-Three by Shannon Appelcline further builds on the social behavior described by Richard Bartle in his article, and gives a list of functionality and features ideal for developers to utilize in order to create possibilities for players to engage in social interaction. The article builds on the information found by Bartle, and describes several features utilizing the various social behavior patterns discovered by Bartle.

Bibliography

[Abowd, 1997] Gregory D. Abowd, Anind K. Dey, Gregory Abowd, Robert Orr & Jason Brotherton, 1997 *Context-awareness in wearable and ubiquitous computing* Last visited 11 of October 2007 at http://www.cc.gatech.edu/fce/pubs/iswc97/wear.html

[Anderson] Christoffer Andersson *Mobile Positioning - Where You Want To Be!* Last visited 11 of October 2007 at http://www.wirelessdevnet.com/channels/lbs/features/mobilepositioning.html

[Appelcline,2003-1] Shannon Appelcline, 2003-11-20 *Social Gaming Interactions, Part One: A History of Form* Last visited 18 of October 2007 at http://www.skotos.net/articles/TTnT_/TTnT_136.phtml

[Appelcline,2003-2] Shannon Appelcline, 2003-12-04 *Social Gaming Interactions, Part Two: Competition* Last visited 18 of October 2007 at http://www.skotos.net/articles/TTnT_/TTnT_137.phtml

[Appelcline,2003-3] Shannon Appelcline, 2003-12-18 *Social Gaming Interactions, Part Three: Cooperation & Freeform* Last visited 18 of October 2007 at http://www.skotos.net/articles/TTnT_/TTnT_138.phtml

[Apperly, 2006] Thomas H. Apperley, University of Melbourne, SIMULATION & GAMING, Vol. 37 No. 1, March 2006 6-23 *Genre and game studies: Toward a critical approach to video game genres*

[Bartle, 1996] Richard Bartle, 1996, *HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS* Last visited 18 of October 2007 at http://www.mud.co.uk/richard/hcds.htm

[Basili, 1992] Victor R. Basili, *The Experimental Paradign in Software Engineering*

[BeforeCrisis.net] *Before Crisis.net* Last visited 7 of October 2007 at http://www.beforecrisis.net/

[Belcher, 2006] James Belcher, 27 of January 2006 *Mobile Gaming is Taking Off* Last visited 15 of October 2007 at http://www.imediaconnection.com/content/8022.asp

[BioShock] 2K Boston and 2K Australia, 2007, http://www.2kgames.com/bioshock

[BlitWise Productions] BlitWise Productions. Last visited 20 of February 2008 at http://www.blitwise.com/ptanks.html

[Bluetooth] *Bluetooth.com* Last visited 5 of March 2008 at http://www.bluetooth.com/Bluetooth/Technology/

[Braude, 2001] Eric J. Braude, 2001, *Software Engineering An Object-Oriented Perspective*

[BREW] *Quelcomm BREW* Last visited 14 of October 2007 at http://brew.qualcomm.com/brew/en/

[Chau, 2006] Fiona Chau, 16 of September 2006, *Mobile gaming aims for mass market* Last visited 15 of October 2006 at http://www.telecomasia.net/article.php?id_article=1744&page=5

[Cheok, 2004] Adrian David Cheok, Kok Hwee Goh, Wei Liu Farzam Farbiz, Siew Wan Fong, Sze Lee Teo Yu Li, Xubo Yang, 2004, *Human Pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing*

[Cheok,2006] Adrian David Cheok, Anuroop Sreekumar, Cao Lei, Le Nam Thang, 2006 *Capture the Flag: Mixed-Reality Social Gaming with Smart Phones*

[CybStickers] *CybStickers - nå kan du lage digitale helleristninger med mobilen!* last visited 7 of October 2007 at http://www.sintef.no/content/page1____6785.aspx

[Deus Ex] Eidos Interactive Ltd., 2000, http://www.eidosgames.com/games/info.html?gmid=109

[Doom] id Software, 1993, http://www.idsoftware.com/games/doom/doom-ultimate/

[Doom 3] id Software, 2004, http://www.doom3.com/

[Drane, 1998] Christopher Drane, Malcolm Macnaughtan, Craig Scott Computer Systems Engineering University of Technology, Sydney, 1998 *Positioning GSM Telephones*

[Duke Guide] Duke Nukem 3D Arena User Guide *Duke Nukem Arena 3D* Last visited 14 of October 2007 at http://cache.getitnow.edgesuite.net/imgs/appmedia/4117.pdf

[Ericsson] *Technology, We develop the innovations and establish the standards that lead the industry*, Last visited 18 of October 2007 at http://www.ericsson.com/technology/tech_articles

[Ericsson, Edge] White Papir, Ericsson, February 2007, *THE EVOLUTION OF EDGE* Last visited 18 of October 2007 at http://www.ericsson.com/technology/whitepapers/3107_The_evolution_of_EDGE_A.pdf

[Forman, 1994] George H. Forman, John Zahorjan, University of Washington, 1994 *The Challenges of Mobile Computing*

[Fox. 2002] David Fow, Roman Verhovsek, 2002 Addison Wesley *Micro Java Game Development*

[Fuglem, 2004] Ingebrigt Fuglem, Tore Worren, Steinar Brede, 2004 *De nye lokasjonsteknologiene for tjenesteproduksjon.*

[Game Boy] *Game Boy micro* Last visited 11 of October 2007 at http://www.gameboy.com/

[GameSpot] *GameSpot.com* Last visited 12 of October 2007 at http://www.gamespot.com

[Gears of War] Epic Games, 2006, http://gearsofwar.com/

[Gibson, 2006] Bruce Gibson, 2006 *Press Release: Casual Gamers and Female Gamers to Drive Mobile Games Revenues Over the $10 Billion Mark by 2009* Last visited 15 of October 2007 at http://www.juniperresearch.com/shop/viewpressrelease.php?id=19&pr=16

[GPS] *Global Positioning System Serving the World* Last visited 9 of October 2007 at http://www.gps.gov/

[Hallberg, 2003] Josef Hallberg, Marcus Nilsson, Kåre Synnes, Luleå University of Technology / Centre for Distance-spanning Technology Department of Computer Science and Electrical Engineering, 2003 *Positioning with Bluetooth*

[IGN Duke Nukem Review] Levi Buchanan, June 15, 2007 *Duke Nukem Arena 3D Review* Last visited 14 of October 2007 at http://wireless.ign.com/articles/796/796848p1.html

[iHobo] *A Guide to Computer Game Genres* Last visited 13 of October 2007 at http://ihobo.com/gaming/genres.shtml

[iPhone] *Apple - iPhone* Last visited 8 of October 2007 at http://www.apple.com/iphone/

[Java ME] *The Java ME Platform - the Most Ubiquitous Application Platform for Mobile Devices*, Last visited 3 of October 2007 at http://java.sun.com/javame/index.jsp

[Jegers, 2006] Kalle Jegers, Mikael Wiberg, 2006, *Pervasive Gaming in the Everyday World*

[JVM] Tim Lindholm, Frank Yellin *The Java Virtual Machine Specification* Last visited 4 of October 2007 at http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html

[J2ME Building Blocks] *J2ME Building Blocks for Mobile Devices* Last visited 4 of October 2007 at http://java.sun.com/products/cldc/wp/KVMwp.pdf

[Kolo, 2004] Castulus Kolo, Timo Baur, November 2004 *Living a Virtual Life: Social Dynamics of Online Gaming* Last visited 18 of October 2007 at http://www.gamestudies.org/0401/kolo/

[Krikke, 2003] Jan Krikke, IEEE Computer Graphics and Applications, Jan/Feb 2003, *Samurai Romanesque, J2ME, and the Battle for Mobile Cyberspace*

[Lindley, 2003] Craig A. Lindley, October 3, 2003 *Game Taxonomies: A High Level Framework for Game Analysis and Design* Last visited 13 of October 2007 at http://www.gamasutra.com/features/20031003/lindley_01.shtml

[Liz, 2004] Joze Liz, June 2004 *Nintendo Releases GBA Sales Milestones* Last visited 11 of October 2007 at http://www.pgnx.net/news.php?page=full&id=4968

[Luukainen, 2007] Editor: Sakari Luukkainen, *Topical Evolution Paths of Mobile Multimedia Services* , p.53-63, Anssi Vanhanen, *MOBILE GAMES BUSINESS*

[MachineWorks Northwest] *MachineWorks Northwest* Last visited 14 of Ocotber 2007 at http://www.machineworksnorthwest.com/

[Microvision] *MILTON BAILEY'S MICROVISION (1979-1981) THE GAMEBOY WASN'T THE FIRST* Last visited 11 of October 2007 at http://ryangenno.tripod.com/sub_pages/Microvision.htm

[MobileTracker, 2005] *Total mobile subscribers top 1.8 billion* Last visited 7 of October 2007 at http://www.mobiletracker.net/archives/2005/05/18/mobile-subcribers-worldwide

[New Edge] *New Edge, Tower defense: Wrath of Gods 2007.* Last visited 18 of February at http://www.newedge.ru/en/ourgames/td.html

[New Super Mario Bros.] Nintendo 2006, http://mario.nintendo.com/

[N-Gage] *N-Gage* Last visited 11 of October 2007 at http://www.n-gage.com/

[N-Gage QD] *N-Gage QD Game Deck | Gaming Deck | Gaming Phone | Gaming Device | Accessories* Last visited 11 of October 2007 at http://www.n-gage.com/en-R1/gamedeck/ngage_qd/

[Nintendo DS] *Nintendo DS Lite revolutionizes hand-held gaming with dual screens, touch screen technology, built-in PictoChat software and more.* Last visited 8 of october at http://www.nintendo.com/systemsds

[Nintendo Wii] *Wii.Nintendo.com - In-Depth Regional Wii Coverage* Last visited 8 of October 2007 at http://wii.nintendo.com/

[Nøsterud, 2007] Øivind Nøsterud, Alf Inge Wang(Main-supervisor), Anne Marte Hjemås(Co-supervisor), autumn 2007. *Mobile and social video games.* Depth study in the course TDT4570 - Game Technology at NTNU.

[P990i] *P990i - Oversikt - Sony Ericson* Last visited 8 of October 2007 at http://www.sonyericsson.com/spg.jsp?cc=no&lc=no&ver=4000&template=pip1&zone=pp&pid=10336

[Pelkonen, 2004] Tommi Pelkonen, February 2004 *Mobile Games E-Content Report 3 an integrating report by ACTeN Anticipating Content Technology Need*

[POTCM] *mDisney Studios - Pirates of the Caribbean Multiplayer game* Last visited 21 of October 2007 at http://www.plundertheport.com/

[Playstation Portable] *PlayStation.com - PlayStation Portable - About PSP* Last visited 11 of October 2007 at http://www.us.playstation.com/PSP/About

[Powers, 2006] Michael Powers, November 2006, *Mobile Multiplayer Gaming, Part 1: Real-Time Constraints* Last visited 15 of October 2007 at http://developers.sun.com/mobility/midp/articles/gamepart1/

[Rashid, 2006] Omer Rashid, Ian Mullins, Paul Coulton, Reuben Edwards, 2006, *Extending Cyberspace: Location Based Games Using Cellular Phones*

[Riggs, 2003] Riggs, Taivalsaari, Peursen, Huopaniemi, Patel, Uotila, 2003, *Programming Wireless Devices With the Java 2 Plattform Micro Edition, Second Edition*

[Soldat] Michal Marcinkowski, *Soldat* Last visited 16 of October 2007 at http://www.soldat.pl/

[Tanenabum, 2003] Andrew S. Tanenbaum, 2003, Pearson Education Interantional, *Computer Networks, Fourth Edition*

[Tetris Multiplayer] *Tetris | EA Mobile Games | Mobile Games* Last visited 14 of October 2007 at http://www.eamobile.com/Web/Catalog/US/en/game/mobile/ProductDetailOverviewView/product-24171

[Tibia ME] Tibia Micro Edition. Last visited 14 of Ocotber 2007 at http://www.tibiame.com/home/?language=en

[Undercover 2] *Undercover 2: Merc Wars* Last visited 21 of October 2007 at http://undercover2.com/main.php

[Verizon] *Duke Nukem 3D Arena* Last visited 14 of October 2007 at http://products.vzw.com/search_games.aspx?id=search_games&appSearchText=duke+nukem&appSea

[Verizon V CAST] Verizon V Cast. Last visited 14 of October 2007 at http://products.vzw.com/index.aspx

[Verizon Wireless] Verizon Wireless. Last visited 14 of October 2007 at http://www.verizonwireless.com/b2c/index.html

[Wang, 2006] Alf Inge Wang, Dr.Ing Thesis, February 5. 2001, *Using a Mobile, Agent-based Environment to support Cooperative Software Processes*

[Wang, 2005] Alf Inge Wang, Carl-Fredrik Sørensen, Steinar Brede, Hege Servold, Sigurd Gimre, 2005 *The Nidaros Framework for Development of Location-aware Applications*

[Wang, 2006] Alf Inge Wang, Michael Sars Norum, Carl-Henrik Wolf Lund, 2006, *Issues related to Development of Wireless Peer-to-Peer Games in J2ME*

[Wikipedia] *Wikipedia*, Last visited 18 of October 2007 at http://en.wikipedia.org/

[Williams, 2004] Christopher Williams, Mark Burge, 2004 *MIDP 2.0 Changing the Face of J2ME Gaming*

[Wolf, 2000] Mark J. P. Wolf, 2000 *Genre and the Video Game* Chapter 6 of *The Medium of the Video Game*

Part VI

Appendices

Appendix A

Installation and execution guide

This appendix describes how to install and play the prototype game. All folder destinations are destinations to the zip file that was delivered with the report.

A.1 How to start the client

The client is a Java ME application and can be started either by starting the game in an emulator, or transfer the files in the "platform puzzler/bin/client/" folder on to your mobile phone, i.e. the PlatformPuzzler.jad and PlatformPuzzler.jar, where the jar file is the actual game and the jad file is the description file of the game. When the game is transfered to the mobile phone, go to the folder that it has been transfered to and install it on your mobile phone. This will require that the mobile phone is able to connect to the server via mobile network however, and it has not been tested by the project group.

Depending on your emulator you may in some emulators start the game by double clicking the jad file. For other emulators however refer to their own descriptions on how to start MIDlet applications. The project group used the emulator that can be downloaded on Sony Ericsson's web sites[1](Please note that this will require Java SE 1.6 or higher), and the project group can only vouche for this emulator since no other emulators were tried.

A.2 How to start the server

The server requires Java SE 1.5 or higher in order to run. To start the server simply go to the "platform puzzler/bin/server/" folder and double click on "run Server.bat" which will start the server.

[1]http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

A.3 How to play the game

Install the server and start the server on any device able to run Java SE 1.5 applications and have a network connection. Please note that the update rate of the position requesting from the server can be updated in the .bat file by altering the number displayed after the java call. It is set to 250 as default which means that it sends a position request to all of the clients on time every 250 milliseconds(roughly).

Then start the mobile game by either using a emulator or run it from your mobile phone. Then you enter the IP address of the server to join the server. One server is required per game and the game is most suited for two players at the moment; however the server and an emulator can be run on the same computer. The controls will differ between the emulator/phone you use, but on the standard emulator from Sony ERicsson the 4 and 6 are left and right movement, 5 is jumping, and the A is used to press trigger objects while B is used to use the club. The game require two players in order to solve the first level, and currently there is only one level in the game.