NTNU

Norwegian University of
Science and Technology

Lecture Quiz Extended

An Improvement of the Lecture Quiz Game

Long Tien Tran

Master in Information Systems
Submission date: June 2008
Supervisor: Alf Inge Wang, IDI

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

Problem Description

Lecture Quiz is a game used to provide more active and participant students in lectures. The game is based on a server, a teacher client and a student client.

The goal of this project is to extend the Lecture Quiz game with new functionalities that can enhance the management and gameplay of the system.

Assignment given: 16. January 2008
Supervisor: Alf Inge Wang, IDI

PREFACE

This project is done as the Master thesis at The Department of Computer Science at the Norwegian University of Science and Technology (NTNU) on the Spring Semester 2008. The project is extending the work from the thesis called “Game Enhanced Lectures” written in 2007 [1].

I would like to express my thankfulness to my supervisor Alf Inge Wang for his enthusiasm, advice and kindly help during the project. I am grateful to my wonderful wife, Doan Ha who gives me new inspiration everyday to complete the project. I would also like to thank Thomas and the technical staff at Guru for their support to deploy the game. Thank to Bian Wu, Ole Kristian Mørch-Storstein and Terje Øfsdahl for their willingly to exchange ideas about the project. Thank to all the students in lab ITV363 and my friends at NTNU for their support testing the game.

Table of content

Table of images

Part I: Introduction and Research methodology

1 Introduction

In this section the motivation for conducting the project is described. After that we will discuss the over view of the problems being addressed. At the end is the content for the rest of the report.

1.1 Motivation

Computer games received much more negative critics than positive critics because games are often coined with issue such as violence or sex content. However, there is an emergent shifting from purely entertainment product to educational tool in recent time.

Nowadays, educational games give teachers opportunities to introduce educational and playful elements into school classes. With the support of computer and networking, teachers and students have a chance to experiment an education environment that could not offer by traditional media like social interaction, critical learning, knowledge based communication and effective interpersonal skill. Educational game makes learning fun, easier and faster [2].

At NTNU, some projects have been started to show the benefits of computer game in education just by utilizing the existing equipment like student mobile phones and university network infrastructure. These projects have proved the benefit of using games on class created game prototype and set up the foundation of applying computer games on education [1]. However there are a need to improve these prototypes in terms of game management and gameplay so that it is easy to create new game, add new content and change the gameplay to adapt to different university courses and requirements.

1.2 Problem definition

Lecture Quiz is a game used to provide more active and participant students in lectures. The game is based on a server, a teacher client and a student client. In a master thesis done in 2007 [1], Ole Kristian Mørch-Storstein and Terje Øfsdahl have explored concept of game to be used in higher education and develop a prototype game to further evaluate that concept. Their game has three components; the first one is the server part running on a server. This component connect to a database and provide the services to the others component. The second component is called teacher client, when running the game on a lecture the teacher will use this one to show the game. This one runs on the teacher computer such as a laptop or a PC. The last component is the one running on a mobile phone of students; they will use it to interact with the game.

The purpose of the previous game is about illustrating the concept of using game in the lecture. For the game to be used and accepted in the lecture and contribute to the educational environment, a lot of matters need to be considered and improved.

In this master thesis, the goal is to extend the Lecture Quiz game with new functionalities that can enhance the management and gameplay of the system. The focus is on the game platform and architecture so that the game can be usable, flexible. The concept of the game can be found out in the previous thesis and will be repeated here when appropriate.

1.3 Readers guide

This report has five parts. The first part is the introduction about the purpose and the motivation to conduct this project. The second part describes the current game and elaborates the problems to find out a new solution. The next part investigates the state of the art of the applications currently used in lectures and the technology used to develop such applications. The forth part describe our own contribution; we will propose a new architecture for the game and implement a game using that architecture. The last part presents our evaluation and conclusion about the whole process and outlines the future work.

2 Research questions and Development method

This section presents the problems that will be addressed during the project and the method used to address them.

2.1 Research question

Along with the problem definition described in section 1.2, this section presents the areas that this project focuses on.

The motivation for our project is to extend the game so that it can be applied in the real educational environment. We need to balance between our requirements and time constrain of our master thesis project. The research questions below are carefully selected base on what we suppose the most important factor to enhance the game and what we can do considering our time limitation and our experience.

RQ1: Which features of the previous lecture quiz game can be improved in term of gameplay of the system?

This research question directly addresses the goal of our master thesis. To answer this question, it is needed to investigate into the detail of the previous game, to understand it thoroughly, to find out and evaluate its weakness and strength. With the note that the previous game is not well documented, there is no requirement specification as well as architecture design. So to understand the previous game we must use revert engineering techniques to rebuild game functions and the system structure based on its source code.

RQ2: What architecture is suitable for this kind of game?

The lecture game will run in a distributed environment of different kind of network spanning different platform of hardware and software. Beside that the game may need to change a lot in the future to adapt with the specific requirement. With our knowledge and experience in software engineering we believe that software architecture is main factor that decide the quality of a software system. Having the right architecture for our game is the first step to ensure the enhancement later.

RQ3: Implement a lecture game based on the proposed architecture.

Based on the finding in the RQ1 and RQ2, a new game will be build with some enhancement. The new design will be described in details in this document.

2.2 Development methodology

The lecture game will be developed using both top down and bottom up approach. The top down method is used when coping with unclear defined problems and there is a need to breakdown the problems in to sub problems. For example, to break down the lecture game into three modules including the Game engine, the Master client and Mobile client. The bottom up approach will be used when develop Master client and Mobile client. Some class of these two modules can be coded with out the need of having the detailed design. The classed then will be integrated into the modules.

A well defined software development process is the critical factor to produce reliable software on time and on budget. According to Scott W. Ambler, there are four different categories into which a method could fall into :

1. **Code and fix**. This approach is also known as “hacking”, “hack and slash”, or “no-process at all”. This approach to development is chaotic and often unplanned, or when it is planned the plan is quickly abandoned. Estimates and schedules, when made at all, are rarely met in practice.

2. **Serial rigorous**. Software processes in this category are well defined and often include detailed procedures that developers are expected to follow in a more-or-less serial manner. For example requirements are identified, reviewed, and accepted. The analysis of those requirements is performed, reviewed, and accepted. The design is defined, reviewed, and accepted. And so on. There is room for feedback between phases, although that feedback is provided via a reasonably defined procedure and the changes are then reviewed and accepted as before. Systems are typically delivered on an incremental basis where the releases are on the order of several quarters or years in length.

3. **Iterative rigorous**. Software processes in this category are well defined and often include detailed procedures that developers are expected to apply in an iterative manner. For example requirements may be initially defined at a high-level with the detail

later identified on an as needed basis. Small portions of your system are fleshed out, with software potentially delivered on an incremental basis following short release cycles often on the order of weeks or months. The Rational Unified Process (RUP) and the Enterprise Unified Process (EUP) are examples of an iterative rigorous process.

4. **Agile**. Agile is an approach to software development that is people oriented, that enables people to respond effectively to change, and that results in the creation of working systems that meets the needs of its stakeholders. Software processes in this category are defined at a high-level, often presented as a collection of synergistic practices or philosophies. Feature Driven Development (FDD), Agile Unified Process (AUP), and XP are examples of agile software processes.

In the setting of this project as well as our experience and knowledge, the iterative and incremental method is suitable. Developing the product during a lot of iteration reduce the risks of unclear requirement and the risk of lacking knowledge of project management. Using this method does not require much advance and detailed planning. The software can be refined until it reaches the satisfaction. However, there is a need to customize the iterative process, in this project, the requirement is quite clear. Challenge is in the implementation phases, so the iteration will focus more on that phases.

The method is illustrated in Figure 1:

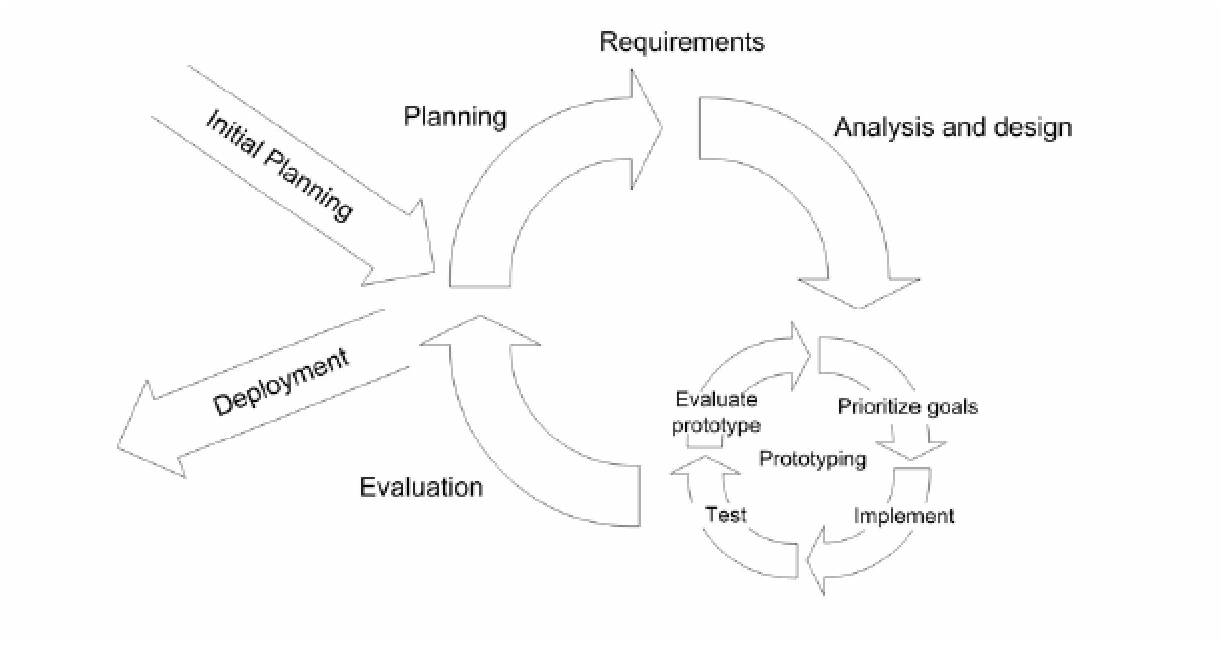


Figure 1. *Iterative software process*

The basic steps include: Planning, Requirement, Analysis, Implementation (prototyping), Evaluation and Deployment.

2.3 Research method

Based on the research areas chosen above, this section will present an appropriated research method to address the research questions. According to Basili, there are three most frequently used research methods in software engineering [2]:

The empirical method: This is a statistical method using to verify a hypothesis through collection of empirical data. This method can e.g. be applied to find out if a new technology is an improvement over older technologies [5].

The mathematical method: this is based on mathematical and formal methods. A formal theory has to be constructed and results are in turn derived from this theory. Empirical data can be used for comparison with the derived results.

The engineering method: This method is defined as "the use of heuristics to cause the best change in a poorly understood situation with in the available resources" ([6] p.28). This one fits into the purpose of this document in its ability to deal with systems of major complexity by consequently finding improvements from self conducted validation methods.

The key point of using the engineering method is about discovery the heuristics that can be used to improve the current situation. Heuristic is difficult to define. According to Koen, heuristic is “anything that provides a plausible aid or direction in the solution of a problem but is in the final analysis unjustified, incapable of justification, and fallible. It is used to guide, to discover, and to reveal” [6]. The heuristics to improve the lecture game in this project is found out by conducting literature research and looking in to the previous source code as well as documents to discover the points that can be improved.

The adaptation of engineering method is illustrated in Figure 2:

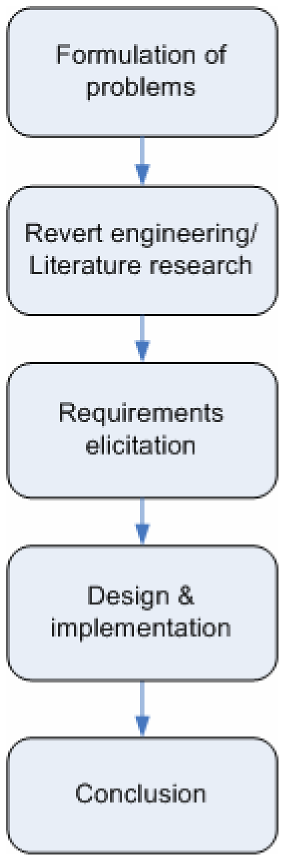


Figure 2. *Research method*

The first step is to formulate the problems. After that the investigation of the previous lecture game to find out the features needed to improve. Then we will specify the requirement for the game. Based on the requirement we will design and implement a next version of game. And finally is the conclusion about the whole process.

Part II: Current solution and Problems elaboration

3 Current solution

In this section, the current state of the game will be investigated in some details to find out which features need to improve, the finding starts with the game server then the teacher client and the mobile client.

3.1 Description

The concept of current game is shown in Figure 3.

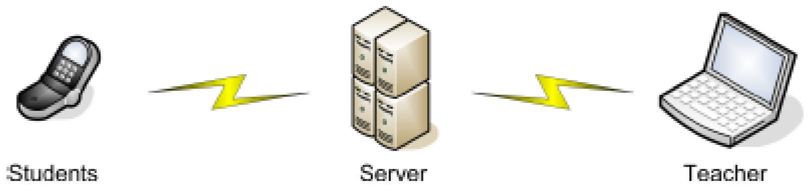
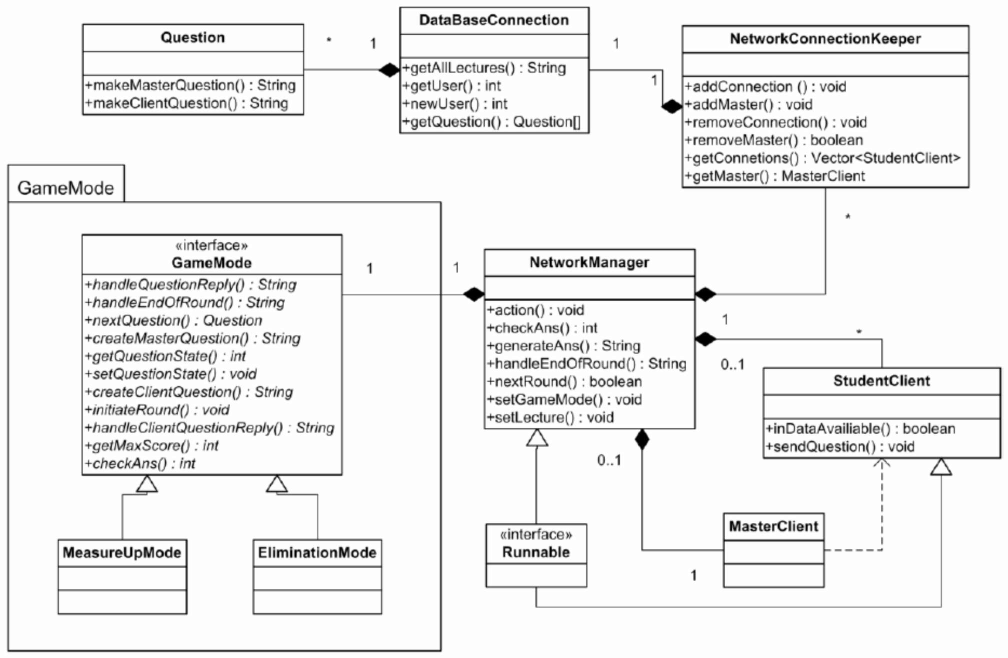


Figure 3. *Current game [3]*

There are three components of the game including Server, Teacher client and Student client (mobile client). In class the teacher uses his computer to control the game which is running on the server. Students use their mobile phone to play the game. The game is showing on the teacher’s computer screen or projecting on a big canvas.

3.2 The game server

The server is the center component of the game; it connects to a central database and gives service to teacher client and student client. Currently only one game can be support by sever at a time. The current game server is very complex and is not documented thoroughly. The architecture of the server is not descriptive and is very difficult to understand. The simplified class diagram in Figure 4 from the last game [1] is the class design for the current game server.



Question
+makeMasterQuestion() : String
+makeClientQuestion() : String
DataBaseConnection
+getAllLectures() : String
+getUser() : int
+newUser() : int
+getQuestion() : Question[]
NetworkConnectionKeeper
+addConnection () : void
+addMaster() : void
+removeConnection() : void
+removeMaster() : boolean
+getConnetions() : Vector<StudentClient>
+getMaster() : MasterClient
GameMode
«interface»
GameMode
+handleQuestionReply() : String
+handleEndOfRound() : String
+nextQuestion() : Question
+createMasterQuestion() : String
+getQuestionState() : int
+setQuestionState() : void
+createClientQuestion() : String
+initiateRound() : void
+handleClientQuestionReply() : String
+getMaxScore() : int
+checkAns() : int
NetworkManager
+action() : void
+checkAns() : int
+generateAns() : String
+handleEndOfRound() : String
+nextRound() : boolean
+setGameMode() : void
+setLecture() : void
StudentClient
+inDataAvailiable() : boolean
+sendQuestion() : void
MeasureUpMode
EliminationMode
«interface»
Runnable
MasterClient
*
1
0..1

Figure 4. UML class diagram of current server

In this diagram, the class names are not descriptive and the consequence is making it difficult to understand the logic of the game. For example the class NetworkManager is responsible for the management of mobile client and master client as well as to manage the communication in the game, so the name Game or GameManager is making more sense than NetworkManager.

Some classes include too much functions and the responsibility division between classes is replicated. For example the NetworkManager class does some functions of Question class. It should be broken down further into smaller class also.

Extending the MasterClient from the StudentClient is not good design decision. MasterClient and StudentClient have similar functions but they have a lot of different features so the good decision is to make a more general class and extend MasterClient and StudentClient from it.

The implementation of current game also makes it difficult to change and customize. Look at the Appendix A.1 about main loop of the previous game. It is an infinity loop to change the state of the game, and do appropriate things in each state. The function is so long (spanning 4 pages), it should be broken down. The using of constant as literature also makes it very difficult to understand and maintain.

3.3 The teacher client

The teacher client is the component that is used by teacher and will run on teacher's laptop or PC. It is responsible for rendering the graphics, playing sound and it is a mean to display feedback. Currently the teacher using JOGL (an open graphics library) for graphics functions. This make the deployment a complicated when require additional library. And this is problems because when comes to deployment the teacher's laptop have different platforms with require different version of library.

As the server, the teacher client is neither documented well enough nor have clear architecture. The teacher client is bad implemented as well.

3.4 Mobile client

This component is a J2ME application running on mobile phone supports java. When the game is playing, player will use this component to communicate with the server to receive questions and submit answers. At the current version, the mobile client is designed to support only one game that runs on the server, so there is no mechanism to choose with game to join in, it just simple connect to the server. Other problem is that there is no way to exit the game than shutdown the mobile. As the others component, the mobile client is not well documented. Beside the class diagram there is a need to know how the client communicates with the rest of the system. What kind of messages is exchanged and so on.

4 Elaborate the problems

The study of previous thesis showing the concept of a game using in the lecture and the investigation also prove the advantage of such a game compare to traditional education method. But there are still a lot of matter needs to be done to create a “real life” game.

In this section, those matters are to be defined.

First of all, it is not practical that one server can host only one game. In a university there are many course and many lectures. So the need of playing many games at the same time on one server is obviously. The current only support only one game at a time and this need to be addressed at the next version of the game. We are going to design a new architecture for the application that can host multi games at the same time.

Secondly, the teacher needs a tool to manage the content of the game. Currently the questions and alternatives are inserted manually into the database. We need to develop a “Game Content Editor” to support teacher in this task.

We can also extend the game by improving the communication protocol. Currently there is no separate class that is responsible for manage communication. The message is exchange and interpreter using several classes. We can improve this by design a class that is mainly responsible for manage data communication.

The last but not least, we can extend the game by adding more game mode. There are currently two modes, measure up mode and elimination mode. The other mode maybe “Group mode” where players are divided into groups and game is playing between groups.

Because of time constrain of the project, we choose to address the first problem which is to extend the game to support multi games, multi players in this project. The architecture we propose also address the third problem. We believe that solving this problem is the important part to make a practical game that can be use in the real lectures.

Part III: State of the art and Technology

5 Market analysis

In this section several applications that can be used in the lecture are presented along with theirs features and the experiences from using these applications. This section is inspired by the previous report [1].

5.1 TVREMOTE Framework

The TVREMOTE framework was designed to allow for student participation in lectures counting hundreds of participants at Darmstadt University of Technology, Germany [7]. The central idea of the framework is supporting interaction such as posting question. The framework has three components including the server, the educator display and student tools like mobile phones.

The system can handle several interaction types. Student can submit message typed in free text. The system features polling of student opinions and electronic question submission. The teacher collects the feedback and reads it from a private display, from which she can select a question for display on a second public screen. The multiple choice quiz provides the teacher with a statistical distribution of correct and incorrect answers. The teacher can also broadcast links and notes that are difficult to copy from projector such as number of URLs. Studies of using the TVREMOTE show that students generally appreciate a short explanation as to why a given answer is correct.

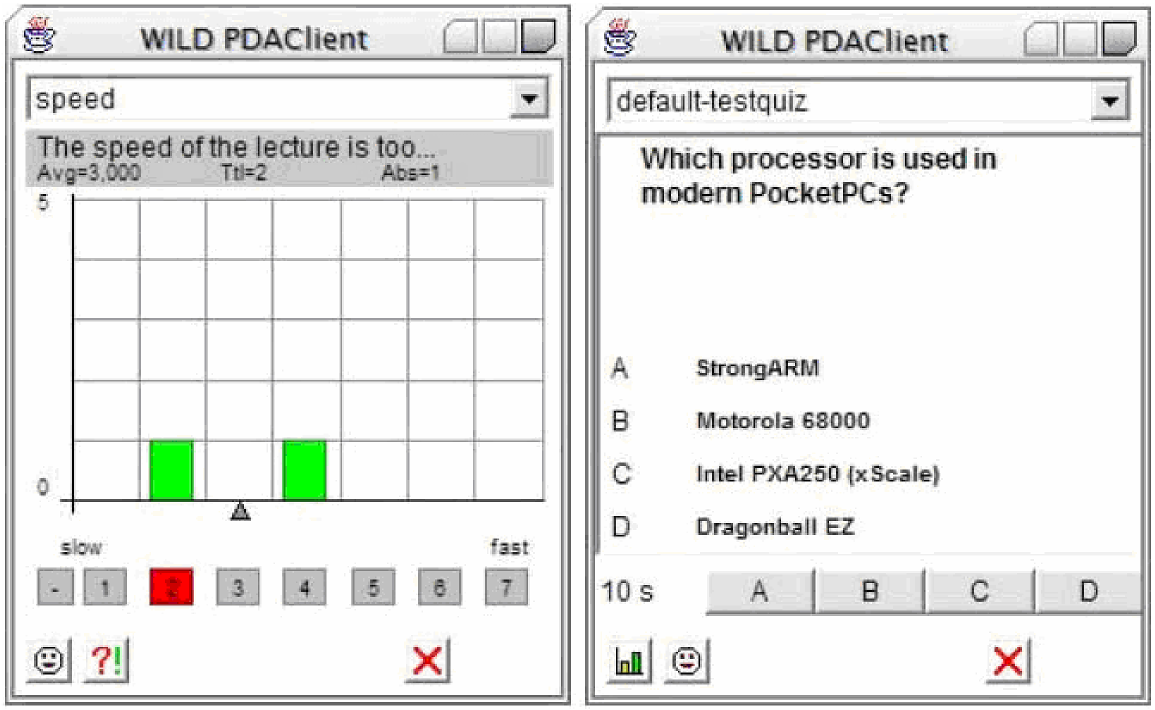
The TVREMOTE framework uses GPRS for data transmission and Bluetooth support is planned as a future feature. Surveys show that students are reluctant to pay for the data transmission fees from sending data over GPRS [7].

5.2 Classroom Presenter

The *Classroom Presenter* system has been used to facilitate active learning at the University of Washington. Students can write notes in blanks in digital slides; the notes can be their questions or problem solutions. The educator hands out a set of Tablet PCs for student use in each lecture. Student notes will be shown on the educator's Tablet PC and can be used for discussion or evaluation later.

5.3 WIL/MA

WIL/MA is an application developed at the University of Mannheim, Germany supporting of digital hand raising, spontaneous comments and multiple choice questionnaires. WIL requires personal Java Runtime environment to run. This means that it is available on laptops, pocket PCs and PDAs. Not many students own pocket PC so the researchers of the project had to buy pocket PC and handed out to the student participants at the beginning of the lecture. The system uses WLAN coverage to transmit data. The teacher receives the student data on his PC and reads it from a private screen. A survey of similar classes where one class used WIL/MA and the other attended traditional lectures showed the learning outcome of using WIL/MA superior [8]. Example of screen shots from student feedback and multiple choices are shown in Figure 5.



The speed of the lecture is too...
Avg=3,000 Ttl=2 Abs=1

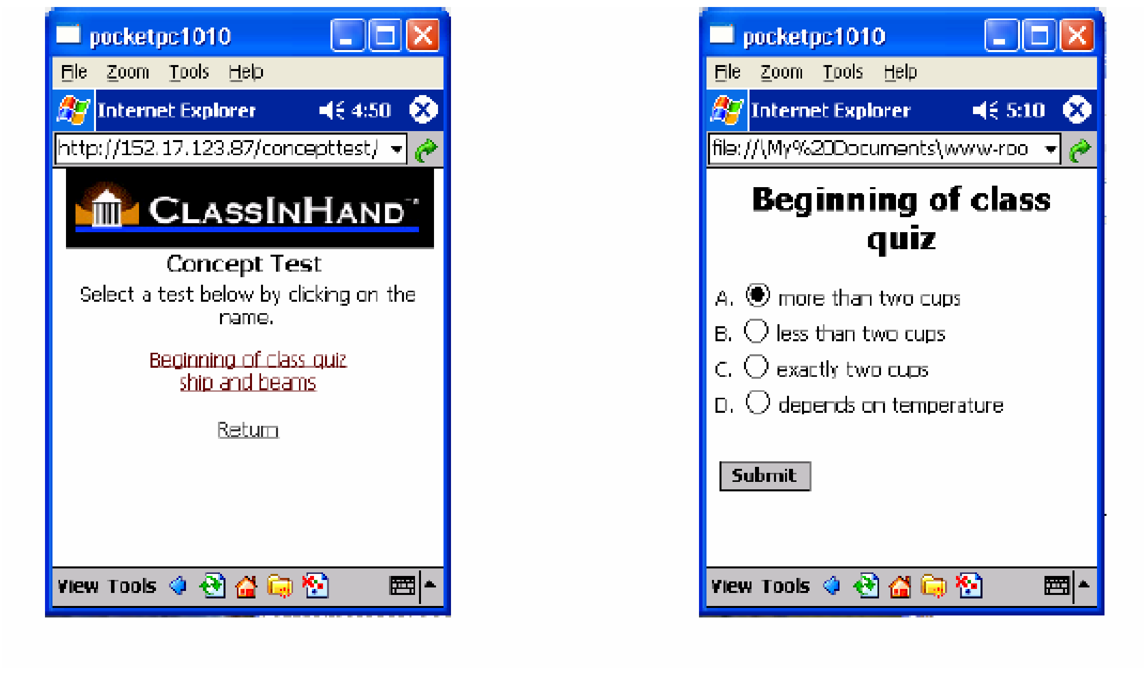
Which processor is used in modern PocketPCs?

A StrongARM
B Motorola 68000
C Intel PXA250 (xScale)
D Dragonball EZ

Figure 5. WIL/MA PDA screen shots [1]

5.4 ClassInHand

ClassInHand is developed at WakeForest University, USA. The basic functions support presentation controller, real time quiz and student/teacher interaction. The system runs on PDAs supporting Windows Mobile 5 or Windows Mobile 2003 for PocketPC. The presentation controller allows teacher using his PDA to control a presentation such as navigating through the slice and read notes. The quiz function allows teacher collecting result submit from students. Currently questions are read verbally. An exam of a question with alternatives is shown in Figure 6. Beside that the application allow student to send text feedback and asking questions.



Concept Test
Select a test below by clicking on the name.

Beginning of class quiz
ship and beams

Beginning of class quiz

A. ● more than two cups
B. ○ less than two cups
C. ○ exactly two cups
D. ○ depends on temperature

Figure 6. ClassInHand PDA screen shot

5.5 Ez ClickPro

EzClickPro is a commercial class room polling application developed by Avrio Ideas for teaching in elementary school. It is commercially available for £3.450 GBP for the maximum set including 100 custom remote controls.

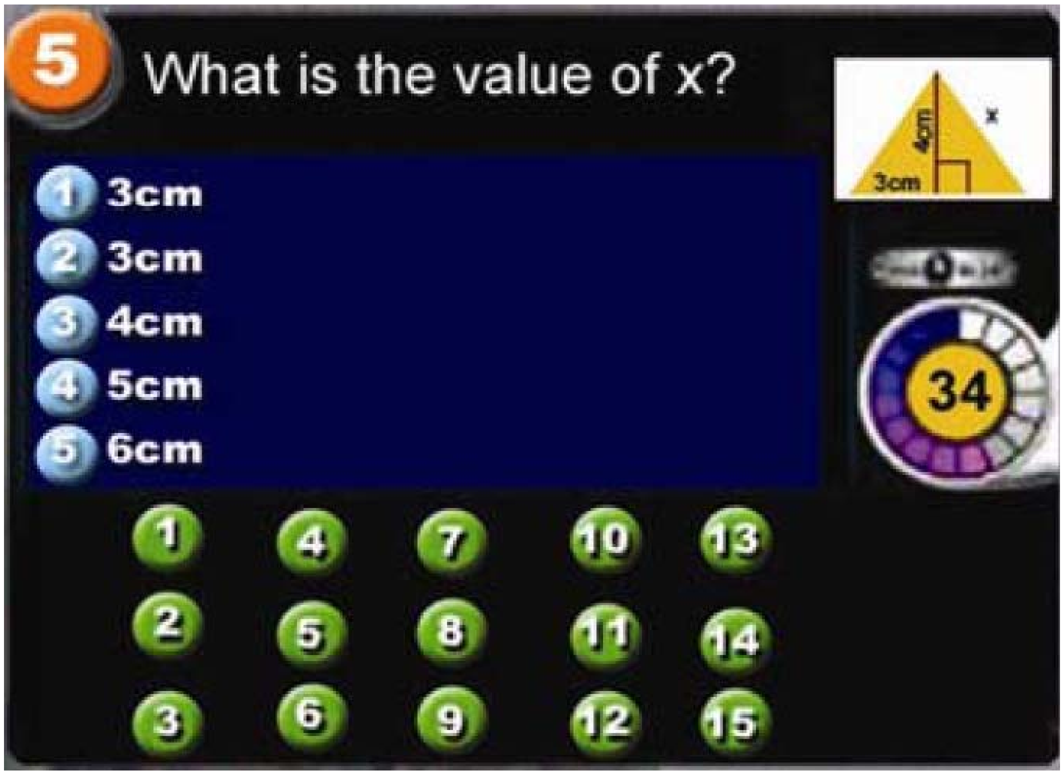


Figure 7. *Screen shot Ez ClickPro*

EzClickPro uses infrared technology and custom produced remote controls, shown in Figure 8, with the teacher running an application on a PC as shown in Figure 7. Multiple choice questions are displayed on a TV or projected on to a canvas, before the students submit their answer using the remote control. Each student is assigned a number that is displayed in a green circle if the answer is correct or otherwise if the answer was wrong. The questions can be presented with elaborating pictures and videos [9, 10]. A new version of the software due 2007, called PowerClip, is integrated into Microsoft PowerPoint.

5.6 Buzz! The Schools Quiz

Buzz was originally conceived as a commercial trivia game for Playstation2 receiving great success in terms of both sales and critics. The game is marked as "the game show in your living room" and comes with special wired handsets called buzzers. There are currently four versions of the game for sale in a multitude of languages, each testing knowledge in different domains such as sports, music and general trivia [11].

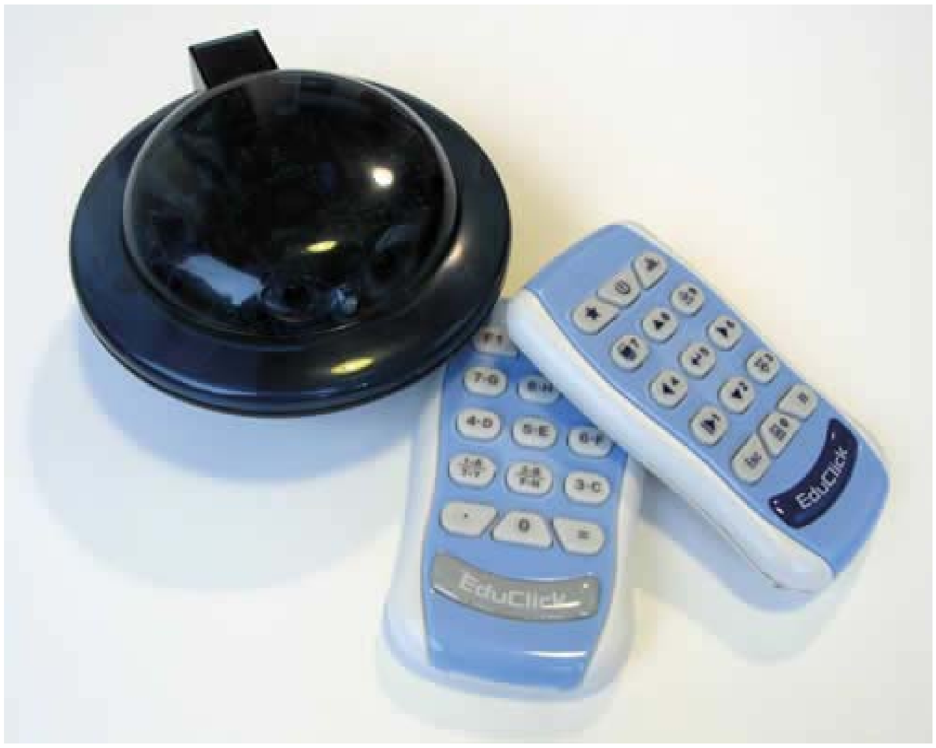


Figure 8. EzClickPro remote controls and sensor

With government funding, a new version of the game, designed especially as a tutoring tool is due released late 2007, named Buzz! The Schools quiz. The game will be marketed towards educational institutions and comes with content covering Stage 2 National Curriculum for primary schools in the UK. A new feature, “Create a quiz” is included to allow tutors to hold revision exercises on a given subject. However, the wired Buzzers are still in use, thus limiting the number of simultaneous players. The content is also limited to the included questions. As a consequence, newer versions are required as the curriculum changes and the game will also be less interesting in countries other than the UK.

6 Evaluation of previous solutions

From Table 1 we see a representation of the features of previous class room software solutions [1]. All the solutions except EZClickPro and Classroom Presenter are tools limited to student participation and student ‑ teacher communication in larger lectures. These solutions have limited or no support for displaying feedback on a projector screen, as a consequence of their function as a communication tool rather than a game. All of the solutions except ClassroomPresenter feature a quiz mode. It seems the quiz is typically intended for the teacher to monitor whether or not the students are paying attention. There are no elements of competition, goals or amusement besides the actual selection of alternatives and the observation of the correct answer. All the solutions except TVRemote depend on hardware being handed out to each student before each lecture.

Two solutions stand out from the class room polling. Classroom Presenter is a powerful communication tool, sharing edited Microsoft PowerPoint slides. But the tool does not support any communication beyond the actual sharing of these slides. Feedback is thus given through the teacher's presentation and never directly to the students. EzClickPro is a pure quiz game where animated feedback is shown directly on a big screen for the students to enjoy.

Table 1. Previous solutions features [1]

Features	TVRemote	Cl. Pres.	WIL/MAX	ClassInHand	EZ ClickPro	Buzz!
Digital student comments	X		X	X		
Teacher info broadcast	X	X	X	X		
Quiz mode	X		X	X	X	X
Public feedback display	(X)	X			X	X
Animated graphics					X	X
No custom HW needs	X					

6.1 Prototype concept comparison

In Part IV, the concept prototype subject to our research is presented. The concept idea is similar to several of the products presented in Table 1. The prototype aims to feature the stimulating gaming experience from “Buzz!”, with animated graphics on a public display. Using mainstream technology, the prototype allows all students and educational institutions to use the system without any hardware of software purchases. This is achieved by only requiring hardware and software that are commonly installed in lecture halls to facilitate usage of the prototype. Likewise for participants, only hardware and software that most students already possess are required. Add‐on functions such as information broadcast and digital student comments are omitted as the prototype is intended to be perceived as a computer game rather than a software tool.

7 Technology

This part presents various technologies available to build different parts of the lecture game. Based on this, the chosen of the technology will be described in Section 8.3.1.

7.1 Server technology platform

This section summarizes the reasons behind the selection of technology platform for the game engine.

7.1.1 Java

The Java is an object oriented, high level language developed by Sun Micosystems. Code written in Java is compiled to byte code, which in turn is interpreted by a Java Virtual Machine (JVM). Early versions of the JVM were considered quite computational inefficient due to the fact that the virtual machine interpreted non - optimized bytecode. But later implementations of the JVM have improved vastly in this regard, and today Java is considered as fast as C and C++ for some platforms. In addition Java is safer to execute [12]. This is due to the JVM and its automatic memory management, thus sparing the programmers of the burden of manual memory management.

One of the key philosophies behind the Java language is platform independence [13]. The bytecode produced by the Java compiler can be run on any implementation of the JVM. Java runtime is available at no cost for a number of platforms, including but not limited to: Windows XP/Vista, Linux, Solaris and Mac OSX [13].

7.1.2 .Net

The Microsoft .NET Framework is a software component that is a part of several Microsoft Window Operating system such as Window Vista or Window server 2008 [14]. It has large library for common classes and manages the execution of program written for that platform. The platform is freely available for download for older versions of the operating system. Other implementations also exist, like Mono for Linux. At the heart of the .NET architecture is the Common Language Runtime (CLR) which is part of the Common Language Infrastructure specification (CLI). The main purpose of the CLR is to make the platform language independent and to provide automatic memory management as well as security features. Code targeted for the .net framework is compiled to Common Intermediate Language (CIL) (previously known as Microsoft Intermediate Language). The CIL byte‑code is then compiled at runtime to machine code [14].

7.2 Mobile Technology Platform

There are several platforms for programming on handheld devices. A short presentation of alternatives is given in this section along with rationale for the choice of platform.

7.2.1 Java 2 Platform MicroEdition (J2ME)

J2ME is Sun's contribution to the wireless market. It is a small scale version of Java designed for smaller and typically handheld devices such as mobile phones, PDAs and other consumer electronics such as car navigation systems. Much like Java a virtual machine interfaces to the specific operating systems, making the actual code portable across any hardware supporting J2ME within a specific configuration. There are configurations for classes of devices such as a configuration for mobile phone, CLDC, and one for more powerful devices such as advanced PDAs called the CDC.

To make the J2ME Runtime Environment complete, a profile constitutes a programming API for the programmer. The only profile available for the CLDC configuration is the MIDP profile. MIDP1.0 is the first released API. An upgraded version, MIDP2.0, supports custom game graphics and multimedia possibilities [15]. Almost every mobile phone being sold at the moment of writing has built in Java MIDP 2.0 support [16].

7.2.2 Microsoft .NET Compact Framework

.NET Compact Framework is a small scale implementation of the .NET frame work optimized and limited to run on small devices. It contains a platform adaptation layer that allows different operating systems on different hardware to run the .NET Compact Framework applications. However, only a few operating systems such as Microsoft CE, Microsoft Pocket PC and Smart phone offer support [17].

7.3 Database solution

There are several competing Database Management Systems (DBMS) on the market today, each powerful and feature rich in their own respects. This part briefly presents the rationale and background for the choice of DBMS.

7.3.1 MySQL

MySQL is a DBMS owned and sponsored by MySQL AB, a Swedish based company. MySQL AB also owns most of the copyrights to the codebase [18]. MySQL is a key component of the LAMP (Linux, Apache, MySQL and PHP) solution stack, which is commonly used to run dynamic websites [19]. The MySQL DBMS comes in two different variants: Community Server and Enterprise Server. Both share the same code base and are released under the GPL. The Enterprise Server, however, is aimed at a commercial marked and has product support and only publicly available source (not binaries). The Community server is released on an unspecified schedule; with binaries being released with every major update free of charge (smaller incremental updates may not have binaries included). As the Community Server is easily available, free of charge and requires no compilation when using the provided binaries for the intended platform, this version has proven itself immensely popular with developers of free software and web sites [19].

7.3.2 Oracle

The Oracle RDBMS (Relational Database Management System) is a commercial, closed source database system which is available on a number of platforms including Windows, Linux, Solaris and Mac OSX.

7.3.3 Other Database Solutions

PostgreSQL is free software, Object ‐ Relational Database Management System (ORDBMS). Its code base is not controlled by a single company, but rather the community which develops and maintains the code [2].

Firebird or (Firebird SQL) is a RDBMS released under the InterBase Public License [21], an open source software license. The application has been in development for over 20 years, starting in 1984 and became open source in 1999.

7.4 Protocols

The choice of protocol is an important design decision, and imposed its own set of constraints on the prototype with regards to robustness and reliability. In the following subchapters we briefly introduce the protocols evaluated for use in this project: TCP and UDP. These are the two protocols that form the core of the Internet Protocol (IP) suite.

7.4.1 Transmission Control Protocol (TCP)

With TCP two peers can establish a connection to one another, having one stream socket at each end. A connection is maintained as the sockets at each end are open. The protocol guarantees in‐order delivery of data between the two parties. TCP controls that no packets has been lost in transmission via sequence numbers on the packets. When packet has been correctly received, TCP sends an acknowledgment of which packets has been received from the sender. In the case of lost or presumably lost packets, the packets will be retransmitted. There is also a checksum in each packet to ensure that the data is not corrupted [22].

7.4.2 User Datagram Protocol (UDP)

UDP does not have any of the mechanisms for reliability which TCP has. There is no sequence numbers for guaranteed in‐order reception of data, nor is there any checking of whether or not the data has arrived properly. These features ensure that UDP is fast and efficient, especially for transmission of large quantities of small data. An example of this can be broadcasting of data. Applications using UDP must tolerate lost or duplicate data [22].

7.5 Communication bearers

The concept to be implemented requires ease of use for large numbers of participants. In this part, we will present a short introduction to the considered communication bearers.

7.5.1 Ethernet over twisted pair

Ethernet over twisted pair cable is standardized as 10BASE‐T, 100BASE‐TX and 1000BASE‐T. The transfer rates are 10 Mbit/s, 100 Mbit/s and 1000 Mbit/s allowing for high transfer rates over short distances, typically 100 meters or less [23]. Data is converted into electrical impulses, which are transmitted over the wires. Ethernet over twisted pair features high transmission rates, low latency and little noise. The limitation of this bearer is the need for the receiver to be physically connected to the sender and the limit on cable length [23].

7.5.2 GPRS, EDGE and 3G

General Packet Switched Data (GPRS) is a packed - switched data service available in GSM mobile networks. Many users share the same communication channel, and transmit data only as needed. This means that the user can be connected to a server for a very long time, yet only pay a small fee if low volume of data is transmitted [24 USA. #22]. The users are charged for data transfer rather than the time they have stayed connected as is the case when using Circuit Switched Data (CSD). GPRS uses one or more times lots to transfer data, where the maximum speed for a regular GPRS slot is 20 Kbit/s. Regular configurations are 3 or 4 bundled slots for up stream data transfer and 1 for downstream. Giving upper theoretical speeds of 80 or 60 Kbit/s upstream and 20 Kbit/s downstream. The more slots used, the more increases the probability of an interrupt caused by CSD needs, such as voice calls, which will cause delays [24 USA. #22].

An enhanced version of GPRS is currently being deployed, Enhanced Data rates for GSM Evolution (EDGE) or Enhanced GPRS (EGPRS). EDGE increases data speeds by increased utilization of the GSM radio signal. EDGE requires compatible handsets and software up grades at the base stations to support the new signal encoding. Maximum speed per slot is 59.2 Kbit/s, giving 236.8 Kb it as a theoretical maximum speed for an EDGE connection using 4 bundled slots, and 473.6 Kbit/s for one using 8 slots [25].

UMTS (3G) is the newest mobile network technology to be deployed in Europe and Norway. UMTS supports up to 384 Kbits/s downstream rates, or 3.6 Mbit/s in High - Speed Downlink Packet Access (HSDPA) enabled networks. The UMTS network has been designed with both voice and data communication in mind, whereas the GSM (2G)networks was originally designed to accommodate voice communication and CSD [26].

7.5.3 WiFi (IEEE 802.11 a, b, g, n)

Wireless LAN, also known as Wi‐Fi, is based on the IEEE 802.11 specifications. Wi‐Fi uses radio waves in the 2,4 GHz spectrum for transmission of data [27]. Wi‐Fi enabled network interface cards is typically bundled with laptops, high‐end mobile phones and PDAs. Wi‐Fi has typically transfer rates of 54 Mbit/second (802.11g) or 11 Mbit/second (802.11b). This is substantially lower than Ethernet over twisted pair [27].

7.5.4 Bluetooth

Bluetooth is a specification for wireless Personal Area Networks (PANs). Bluetooth uses radio waves in the 2,4 GHz spectrum for transmission, i.e. the same spectrum as Wi‐Fi [27, 28]. Transmitters are divided into three classes, where class 1 has a range of 100 meters, class 2 has a range o f10 meters and class 3 has a range of 1 meter. Bluetooth devices are connected in groups, so called piconets, with one master and up to seven active slave devices. Up to other Bluetooth slave devices can be inactive and the master can at anytime activate them, forcing a currently active device to become inactive. Typical Bluetooth applications are file transfer between mobile phones or mobile phones and computers or wireless connectivity between input devices such as keyboards and computers [28].

Part IV: Own contribution

8 Requirement

To specify the requirement of the game, we first describe the over all concepts together with the main parts of the games. Based on that, the detail of how these parts interacting are presented. Finally, the requirements are documented using UML use case diagrams.

8.1 Over all concept of lecture game

The concept of lecture game referred to in this document is mobile multi player quiz game. The game will be used in the lecture where teachers test knowledge of students in a specific area by sending questions to students, collect results, evaluate and display feedback to them on both handheld devices and big screen.

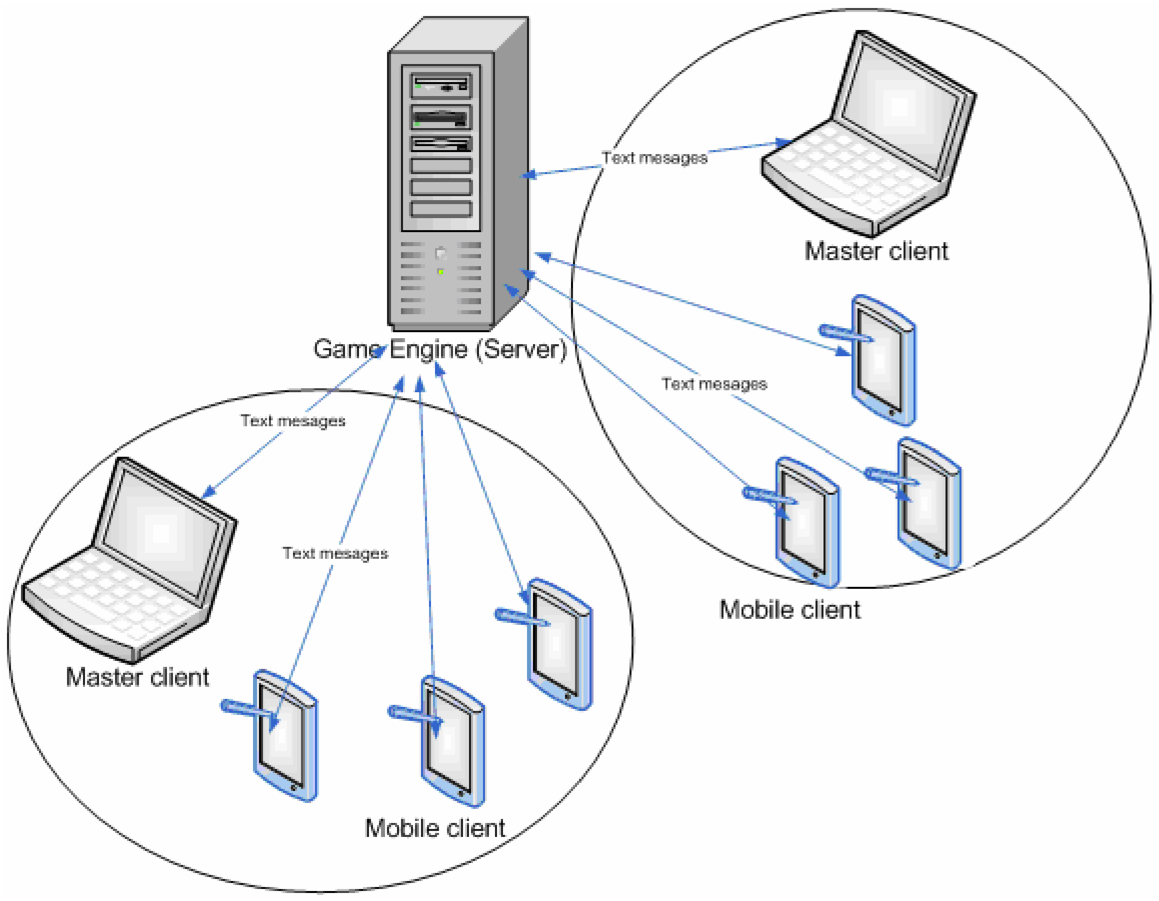


Figure 9. Over all communication

The diagram shows the over all interaction of the system. There are three components running on different platforms. The most important component is the Game Engine running on a server, game engine provide services to Master clients and mobile clients. The game engine has capability to support multiplayer and multi games at the same time. In Figure 9 two examples of games playing concurrently are showing.

The next component is the mobile clients installed on shell phones of users. Before playing, user must connect to game engine (using any networking available Wi-Fi, GPRS). After that, he or she needs to choose with game to join and then waiting for the game to start. On the game, mobile client will receive the questions from the game engine, it then rendering it to player. Player will submit the answers and got feedback when the time is out.

The last component is called the master client which runs on the teacher’s computer like a laptop or a PC. The responsible of this one is to render graphics and displaying feedback to user as well as guideline. In the lecture the teacher computer often connects to a project for a better visualization.

8.2 Detailed description of the lecture game

When the Game Engine (GE) starts it listens for the connection from the Master client and Mobile clients. The Mobile client has to wait to join the game created after a Master client connected successful to GE.

When Master clients connect to Game engine, the user name and password have to be submitted in encoded format to the GE, currently the encoding and decoding functions just return the string itself without doing any thing. The GE authenticates that information with the information about users on its database. It lets the master connected if the authentication process is successful.

After successfully authenticate, the GE is ready to create a new game. The architecture will be opened so that multi Master clients can connect to GE.

At this time a new game has been created and GE is ready for mobile clients to connect to. Mobile client will receive a list of available games; it makes a choice and joins one game at a time. Each time one client connect to the game its information will be sent to the Master client. The game starts when the Master client sends the “Start game” message to the GE. After starting game, no more clients can connect to game.

On the game, the “game story” will progress through a lot of rounds. In each round some things will happened: the GE then will get the a question base on the type of game (for example the level of difficulty) from DB and then send it to the Master client and to some or all of mobile clients (Base on game mode and … add more details later); The clients maybe knock out if it does not answer correctly; the score will be added for correct answer; Sound and graphic will be played and displayed. When time outs the GE got the results, evaluate and send the summary to the master client and mobile clients to display. The GE then moves to the next round and repeats until reaching the last questions.

At the end of the game, GE does the summarization and sends to master client and mobile clients. After that the Master client then can choose to end or restart the game.

8.3 Functional requirement

In this section the functions of the lecture game will be described. There are three components together make the whole system but they are different modules so they will be described separately, when describing one component, the others components will become the actors of that component. UML use case diagram will be used for visualization.

8.3.1 Functional specification of Game engine

The Game engine provides services to both master client and mobile client. The basic functions including verification service, start game, send question, receive answers, send round result and end game. The high level use case model of Game engine in Figure 10 illustrating these functions and showing how the functions are using by its actors.

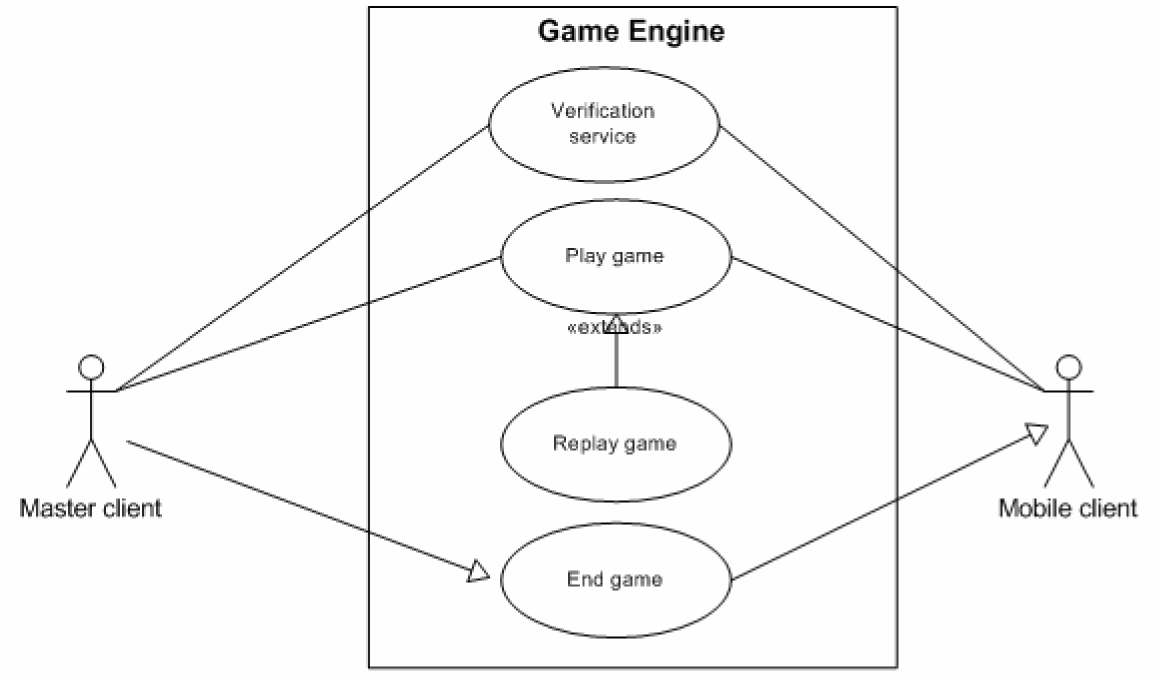


Figure 10. Game engine high level user case

Table 2 to Table 4 describe use cases of game engine in details

Table 2. Verification use case description

Use case	Verification
Brief description	This use case allow client to login to system.
Actors	Master client and mobile client
Preconditions	The client is activated and trying to connect to the game engine
Main flow	The use case begins when master client is activated or mobile client send user name and password to login. If the login request comes from master client, then GE will first send a message to say that logging in is ok and then it tries to create a new game. If the new game is create ok then the game ID will be sent to master client. The GE then delegate the control of communication with the master client to the new game created.

	If the login request comes from mobile client then GE will check user name and password again the information in the database, if the checking function return true, then it lets the mobile client to log in.
Alternative flow	If the game can not be created then a special game id will be sent to master client
Post conditions	If the use case is successful then the client will be logged in the game engine. A new game will be created.

Table 3. Play game use case description

Use case	Play game
Brief description	Start sending questions to clients, evaluate result and render feedback.
Actors	Master client, mobile client
Preconditions	At least one master connected to GE.
Main flow	The use case begins when master client sends the “start” message to GE. GE then will get all the appropriate questions for that game from database. It then iterate through those questions. With each question, GE will send it to master and mobiles. GE collects answers from mobiles, evaluate result and send feedback to clients.
Alternative flow	If master and GE can not exchange messages then game will be terminated.

Post conditions	(none)

Replay game use case

This use case extends of Play game use case. This use case is activated when the game has been played one time, and it allows the game to be played again. Teacher actives by using the menu or just by press some buttons.

Table 4. End game use case description

Use case	End game
Brief description	This use case terminates the game.
Actors	Master client
Preconditions	At least one master connected to GE.
Main flow	The use case begins when master client sends the “exit game” message to GE. GE then will release the resources allocated for the game and terminates send “end” sign to mobile clients.
Alternative flow	(none)
Post conditions	The game ends.

8.3.2 Functional specification of Master client

Figure 11 shows the high level use case model of master client illustrating its functions and showing how the functions are using by its actors.

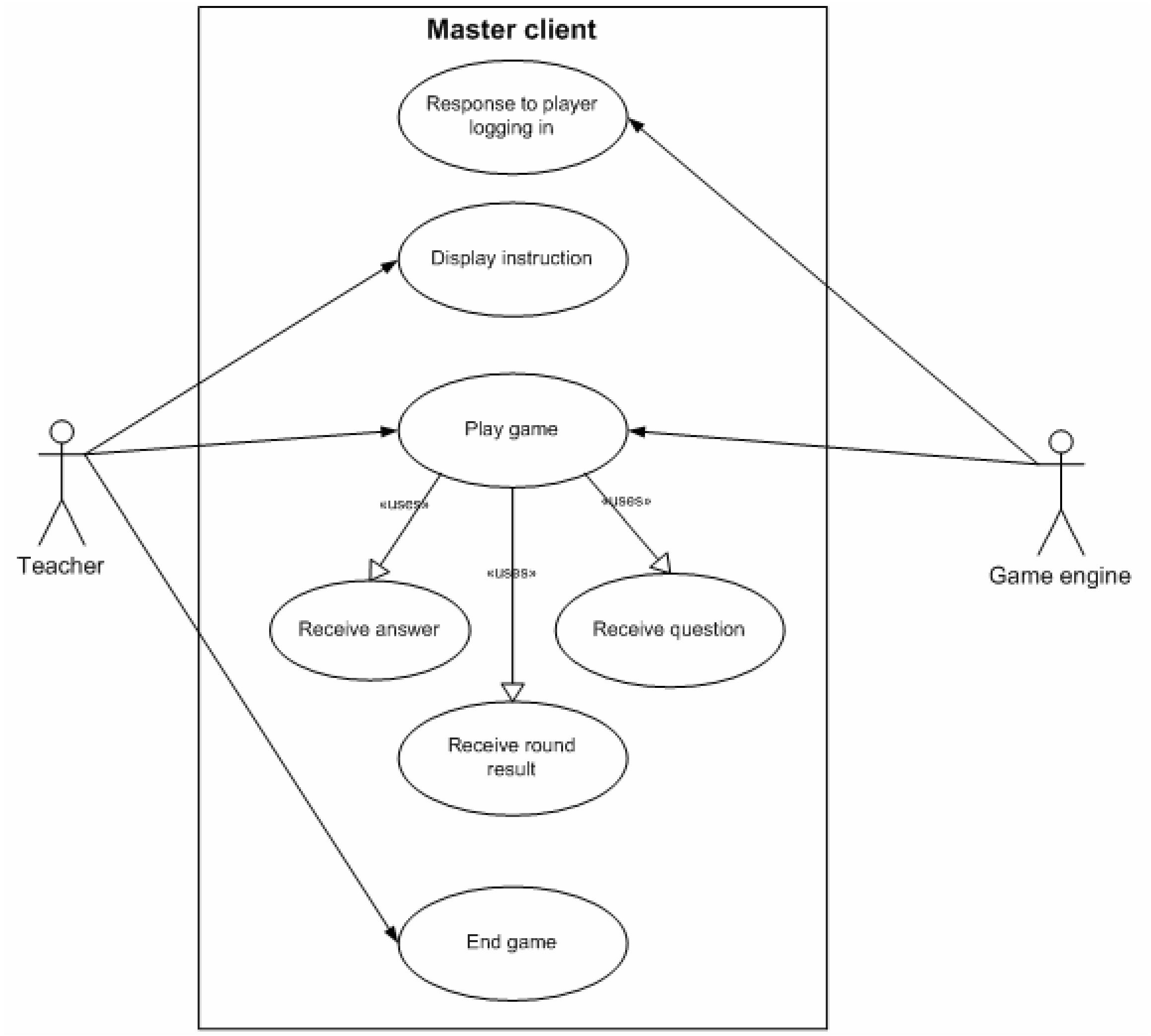


Figure 11. Master client high level use case

Table 5 to Table 8 describes the use case of master client in details.

Table 5. Response to player logging in use case description

Use case	Response to player logging in
Brief description	Update the status of the game when there is mobile join the game
Actors	Game engine
Preconditions	Mobile player choose to join the game

Main flow	The use case begins a mobile client join in the game, the information of mobile will be sent to master for updating.
Alternative flow	(none)
Post conditions	The number of mobile clients will be updated

Table 6. Display instruction use case description

Use case	Display instruction
Brief description	Display short guide of how to play the game
Actors	Teacher
Preconditions	(none)
Main flow	In the game, teacher chooses to display game instruction by choosing from the menu of press The space bar button. Master will change the screen and display the instruction.
Alternative flow	(none)
Post conditions	The instruction will be displayed.

Table 7. Play game use case description

Use case	Play game
Brief description	Start receiving question, feedback from GE and rendering it on screen.
Actors	Game Engine, teacher

Preconditions	The master client is displaying game instruction and teacher chooses to start the game.
Main flow	The use case begins when the teacher choose to start game by press the space bar button. The game will start and the master will receive the first question, display it on the screen together with the count down clock. When time is out, game engine will collect the answers from mobiles evaluate and send the answer to master, master will then do some statistic and display the answer. The game moves on to the next question until reach the last one. At that point, master will receiver the round result form GE and it will display it on screen.
Alternative flow	If master and GE can not exchange messages then game will be terminated.
Post conditions	(none)

Table 8. End game use case description

Use case	End game
Brief description	This use case terminates the game.
Actors	teacher
Preconditions	At least one master connected to GE.
Main flow	The use case begins when teacher press the Escape button. Master will send the exit request to GE to terminate the game.

Alternative flow	(none)
Post conditions	The game ends.

8.3.3 Functional specification of Mobile client

Figure 12 shows the high level use case model of mobile client illustrating its functions and showing how the functions are using by its actors.

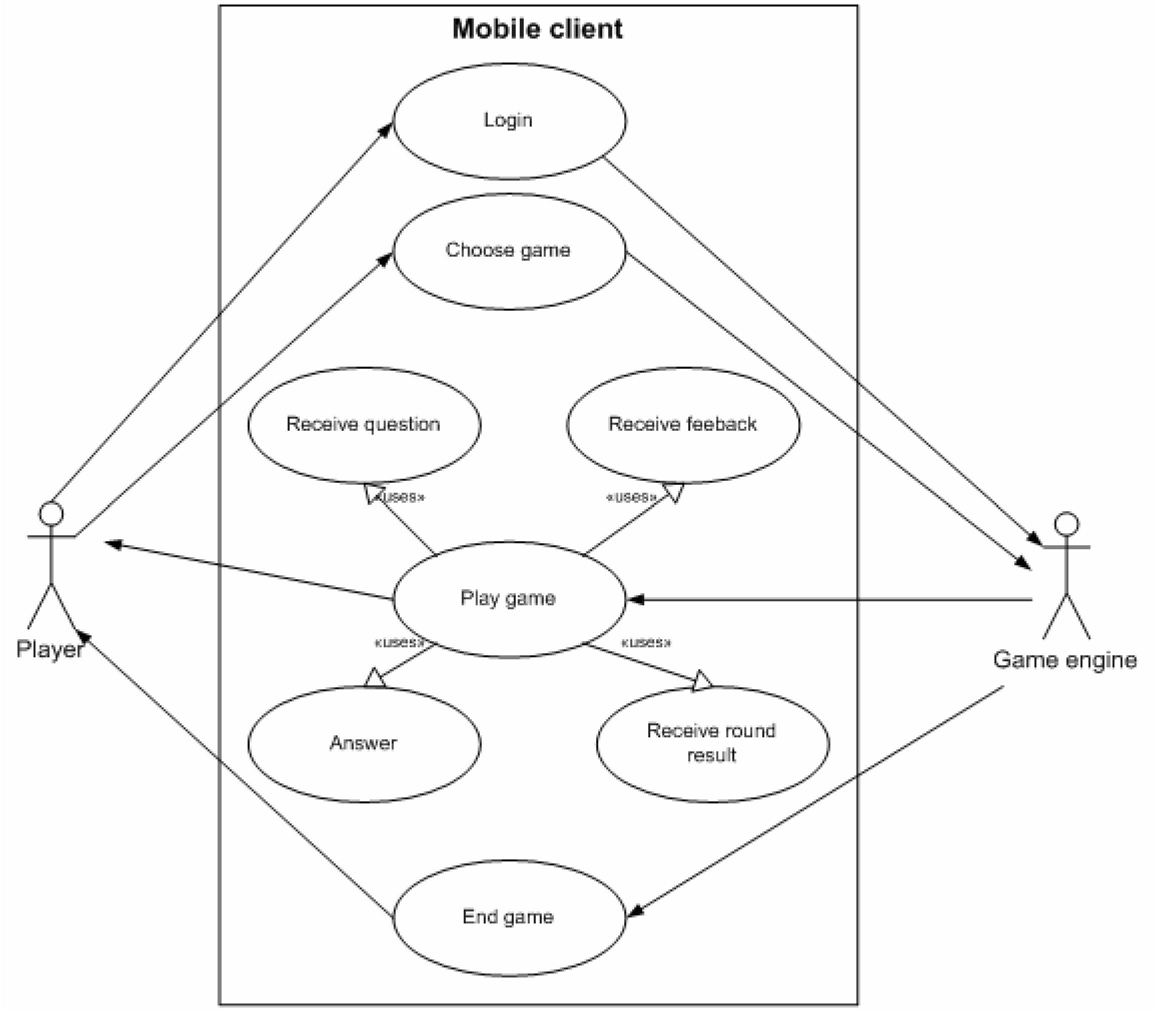


Figure 12. Mobile client high level use case

Table 9 to Table 12 describes the use cases of mobile client in details.

Table 9. Log in use case description

Use case	Log in
Brief description	Allow player to log in GE.
Actors	Game player
Preconditions	Game engine has been started
Main flow	Display the login form Player enter user name and password Send username and password to game engine when players choose “submit” button or similar.
Alternative flow	If game engine is not start then exit game.
Post conditions	(none)

Table 10. Choose game use case description

Use case	Choose game
Brief description	Choose the game to join in
Actors	Player , GE
Preconditions	At least one master connected to GE.
Main flow	Player enter the lecture code that displays on the screen Send the lecture code to GE and wait for feedback The GE will check the code and send feedback to mobile client, if there is code is not exist then mobile client will ask player to enter

	the code again.
Alternative flow	Player can choose to exit game
Post conditions	If the use case is successful then client will join the game.

Table 11. Play game use case description

Use case	Play game
Brief description	Start receiving question, feedback from GE and rendering it on screen.
Actors	Game Engine, game player
Preconditions	The master client is displaying game instruction and teacher chooses to start the game.
Main flow	The use case begins when the teacher choose to start game by press the space bar button. The game will start and the mobile will receive the first question, display it on the screen. And wait for player to submit the answer. When time is out, will display where the player has chosen the correct alternative. The game moves on to the next question until reach the last one. At that point, mobile will receiver the round result form GE and it will display it on screen.
Alternative flow	If mobile and GE can not exchange messages then the mobile will be disconnect from the game.
Post conditions	(none)

Table 12. End game use case description

Use case	End game
Brief description	This use case terminates the game.
Actors	GE
Preconditions	The game engine is shutdown
Main flow	The use case begins when master client sends the “exit game” message to GE. GE then will release the resources allocated for the game and terminates send “end” sign to mobile clients.
Alternative flow	(none)
Post conditions	The game ends.

9 Design

Based on the requirements specified above, the architecture for the game is created. This architecture will be used to find out the classes by breaking down its components into smaller modules. The classes and the communication between them will be documented.

9.1 Proposed architecture for lecture game

The lecture game has a central server and a central database, the server and clients communicate through network. So naturally, the lecture game is based on client-server architecture. How ever the term client- server is very general, there is a need to customize this term. We need to point out which components that client and server contain? We need to decide if it is possible to use the same architecture for different kind of client?

Base on the requirement specified above, the answers for those questions are found. The mobile client and the master client can have the same architecture by using message to communicate with the server. The components and communication between them are illustrated in Figure 13.

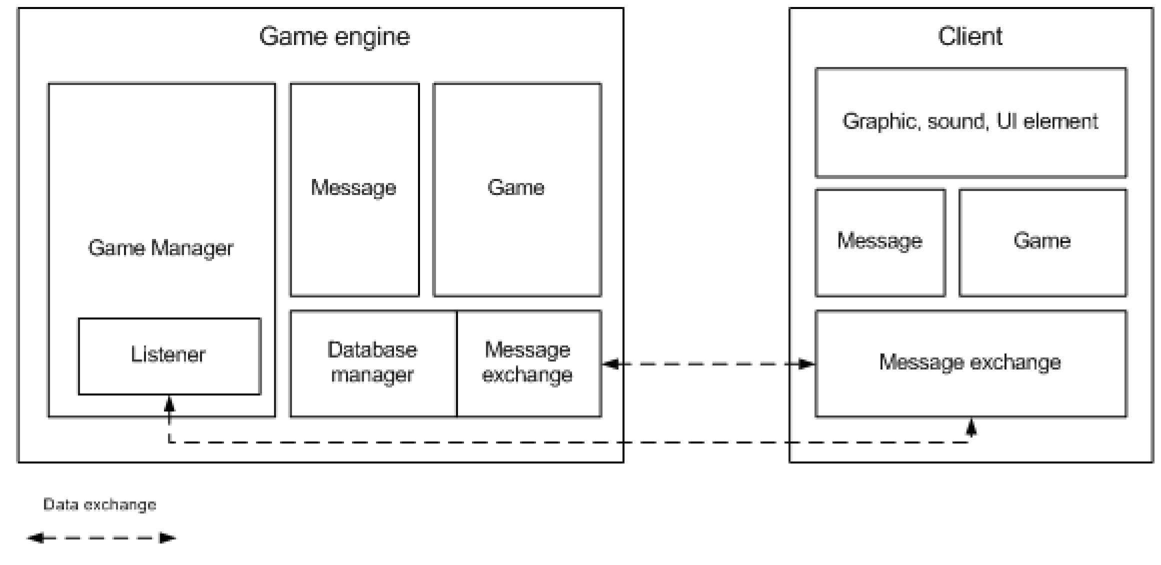


Figure 13. Lecture game architecture

The communication between client and server is handled by the component called “Message exchange” and “Listener”. In the game engine, the “Game Manager” (GM) component is responsible for manage all other component of game engine. It will be activated first and will exist until game engine terminates. Game manager has “Listener” components that listen for connection from client. When a client connects to game engine, GM will create a new Game component. The Game component is responsible for running a specific game (that is the lecture quiz game). After the Game is created, the client will communicate with the game by sending and receiving Messages through Message Exchange components.

When client receives data from Game, its Message exchange component will package that data into a message and pass it to Game component for further processing. The Game component analyzes the message and forwards it to UI component for graphic rendering or playing some sounds.

9.2 Game engine class design

The design of game engine classes is based on the architecture proposed above. Each component from the architecture is implemented by one or more classes. The classes that associate with architecture component have similar name with that component so that it is easy to understand the function of classes. The following diagram in Figure 14 models all classes of game engine, the classes that are less important are only showed with their names. The diagram can be read from the top to the bottom.

The classes are arranged into three packages:

package Core: contains basic classed of game engine

package DataBase: contains classes for manipulate data in a central database

package GameMode: contains classes for implementing different modes of game.

Game manager component from the architecture is associated with GameManager class in Core package. As said above, this class starts the game engine; it contains the main entry function. GameManager class listens for clients using two instants of GameListener class. There are two kinds of listener: MaserListener MobileListener that listens for master client and mobile client accordingly.

Beside the listeners, GameManage has GameList object. As seen in the diagram, GameList and Game havs 1-* relation meaning that the GameManager can manage multi games. Game is added to GameList when there is master client connected to GE.

After having been created, Game object is responsible manage a specific game. Each game associates with one specific master client and n mobile clients. To communicate with master client and mobile client, Game object use two classes MasterCommunicator and MobileCommunicator accordingly. If client send data to Game, the communicator classes will receive it then create a message and call the game to do the actually processing.

The game needs to retrieve questions from database but it does not get the questions directly. Instead it uses the Round class which organizes questions into the round and game will request Round for questions when it needs. Round class uses

DataBaseConnection class to retrieve information from the database.

Each game has a specific mode which specifies how the game is running. The GameMode interface generalize the functions that each specific mode needs to implemented. By using the interface, it is easy to add more modes later with out changing the others classes. This version of game implementing one game mode called measure up mode.

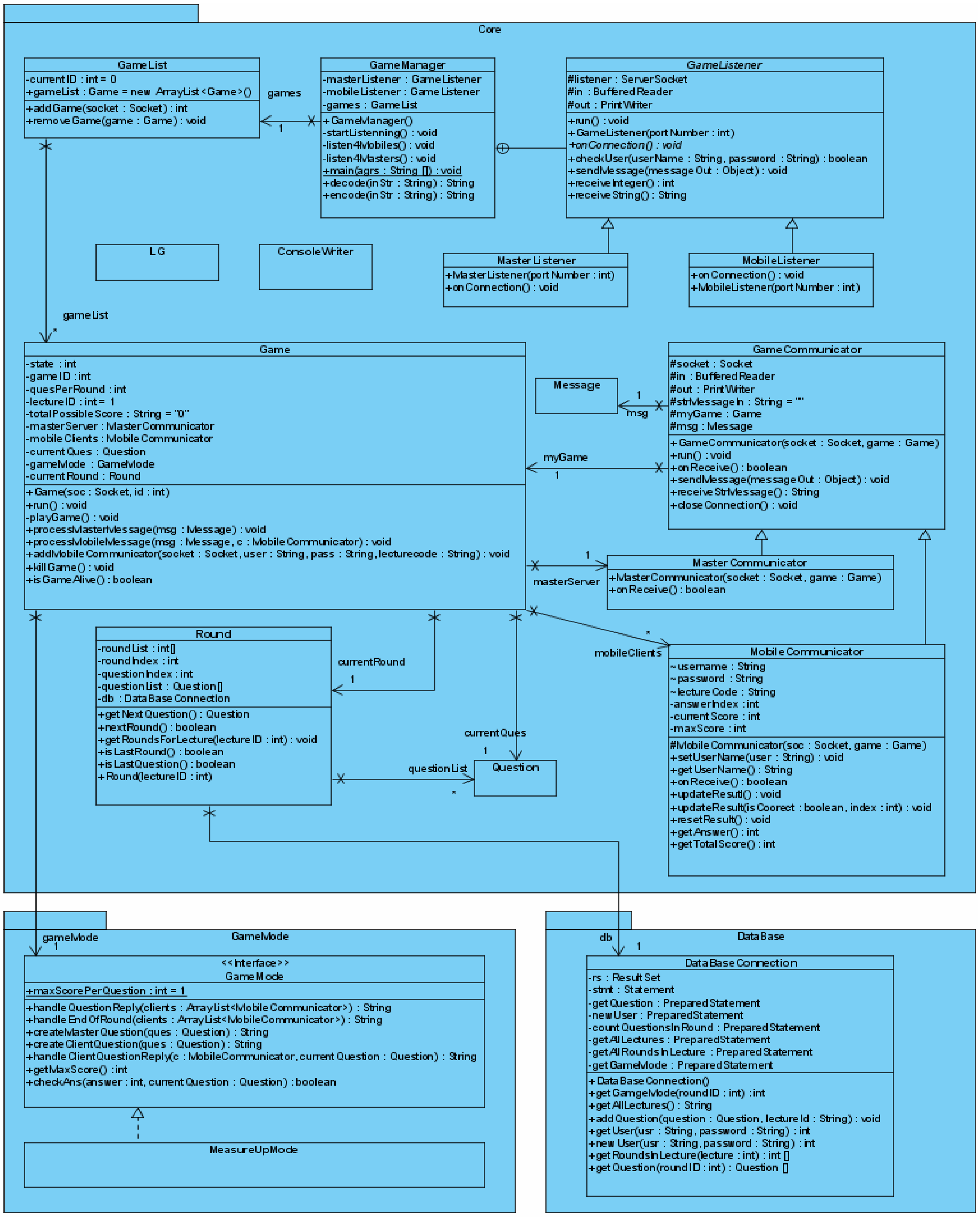

Core
GameList
-currentID : int = 0
+gameList : Game = new ArrayList<Game>()
+addGame(socket : Socket) : int
+removeGame(game : Game) : void
games
1
GameManager
-masterListener : GameListener
-mobileListener : GameListener
-games : GameList
+GameManager()
-startListenning() : void
-listen4Mobiles() : void
-listen4Masters() : void
+main(agrs : String []) : void
+decode(inStr : String) : String
+encode(inStr : String) : String
GameListener
#listener : ServerSocket
#in : BufferedReader
#out : PrintWriter
+run() : void
+GameListener(portNumber : int)
+onConnection() : void
+checkUser(userName : String, password : String) : boolean
+sendMessage(messageOut : Object) : void
+receiveInteger() : int
+receiveString() : String
LG
ConsoleWriter
MasterListener
+MasterListener(portNumber : int)
+onConnection() : void
MobileListener
+onConnection() : void
+MobileListener(portNumber : int)
gameList
*
Game
-state : int
-gameID : int
-quesPerRound : int
-lectureID : int = 1
-totalPossibleScore : String = "0"
-masterServer : MasterCommunicator
-mobileClients : MobileCommunicator
-currentQues : Question
-gameMode : GameMode
-currentRound : Round
+Game(soc : Socket, id : int)
+run() : void
-playGame() : void
+processMasterMessage(msg : Message) : void
+processMobileMessage(msg : Message, c : MobileCommunicator) : void
+addMobileCommunicator(socket : Socket, user : String, pass : String, lecturecode : String) : void
+killGame() : void
+isGameAlive() : boolean
Message
1
msg
myGame
1
GameCommunicator
#socket : Socket
#in : BufferedReader
#out : PrintWriter
#strMessageIn : String = ""
#myGame : Game
#msg : Message
+GameCommunicator(socket : Socket, game : Game)
+run() : void
+onReceive() : boolean
+sendMessage(messageOut : Object) : void
+receiveStrMessage() : String
+closeConnection() : void
1
masterServer
MasterCommunicator
+MasterCommunicator(socket : Socket, game : Game)
+onReceive() : boolean
*
mobileClients
MobileCommunicator
~username : String
~password : String
~lectureCode : String
-answerIndex : int
-currentScore : int
-maxScore : int
#MobileCommunicator(soc : Socket, game : Game)
+setUserName(user : String) : void
+getUserName() : String
+onReceive() : boolean
+updateResult() : void
+updateResult(isCoorect : boolean, index : int) : void
+resetResult() : void
+getAnswer() : int
+getTotalScore() : int
Round
-roundList : int[]
-roundIndex : int
-questionIndex : int
-questionList : Question[]
-db : DataBaseConnection
+getNextQuestion() : Question
+nextRound() : boolean
+getRoundsForLecture(lectureID : int) : void
+isLastRound() : boolean
+isLastQuestion() : boolean
+Round(lectureID : int)
currentRound
1
currentQues
1
questionList
*
Question
gameMode
1
GameMode
<<Interface>>
GameMode
+maxScorePerQuestion : int = 1
+handleQuestionReply(clients : ArrayList<MobileCommunicator>) : String
+handleEndOfRound(clients : ArrayList<MobileCommunicator>) : String
+createMasterQuestion(ques : Question) : String
+createClientQuestion(ques : Question) : String
+handleClientQuestionReply(c : MobileCommunicator, currentQuestion : Question) : String
+getMaxScore() : int
+checkAns(answer : int, currentQuestion : Question) : boolean
MeasureUpMode
db
1
DataBase
DataBaseConnection
-rs : ResultSet
-stmt : Statement
-getQuestion : PreparedStatement
-newUser : PreparedStatement
-countQuestionsInRound : PreparedStatement
-getAllLectures : PreparedStatement
-getAllRoundsInLecture : PreparedStatement
-getGameMode : PreparedStatement
+DataBaseConnection()
+getGamgeMode(roundID : int) : int
+getAllLectures() : String
+addQuestion(question : Question, lectureId : String) : void
+getUser(usr : String, password : String) : int
+newUser(usr : String, password : String) : int
+getRoundsInLecture(lecture : int) : int []
+getQuestion(roundID : int) : Question []

Figure 14. Game engine class diagram

9.3 Master client class design

Similar to game engine, the classes of master client are also derived from the proposed architecture by breaking down the components in client part. The classes and communication between them are capture using the UML class diagram in Figure 15. The diagram could be read bottom up.

In this diagram, MessageExchangeThread class is responsible to communicate with game engine. Game class represents a specific game, it holds all information about state of the game. When MessageExchangeThread gets data from GE, it packages the data into a message and passes the message to Game class for further processing. Base on the kind of message game class will make a call to functions in the Graphics package to do appropriated actions.

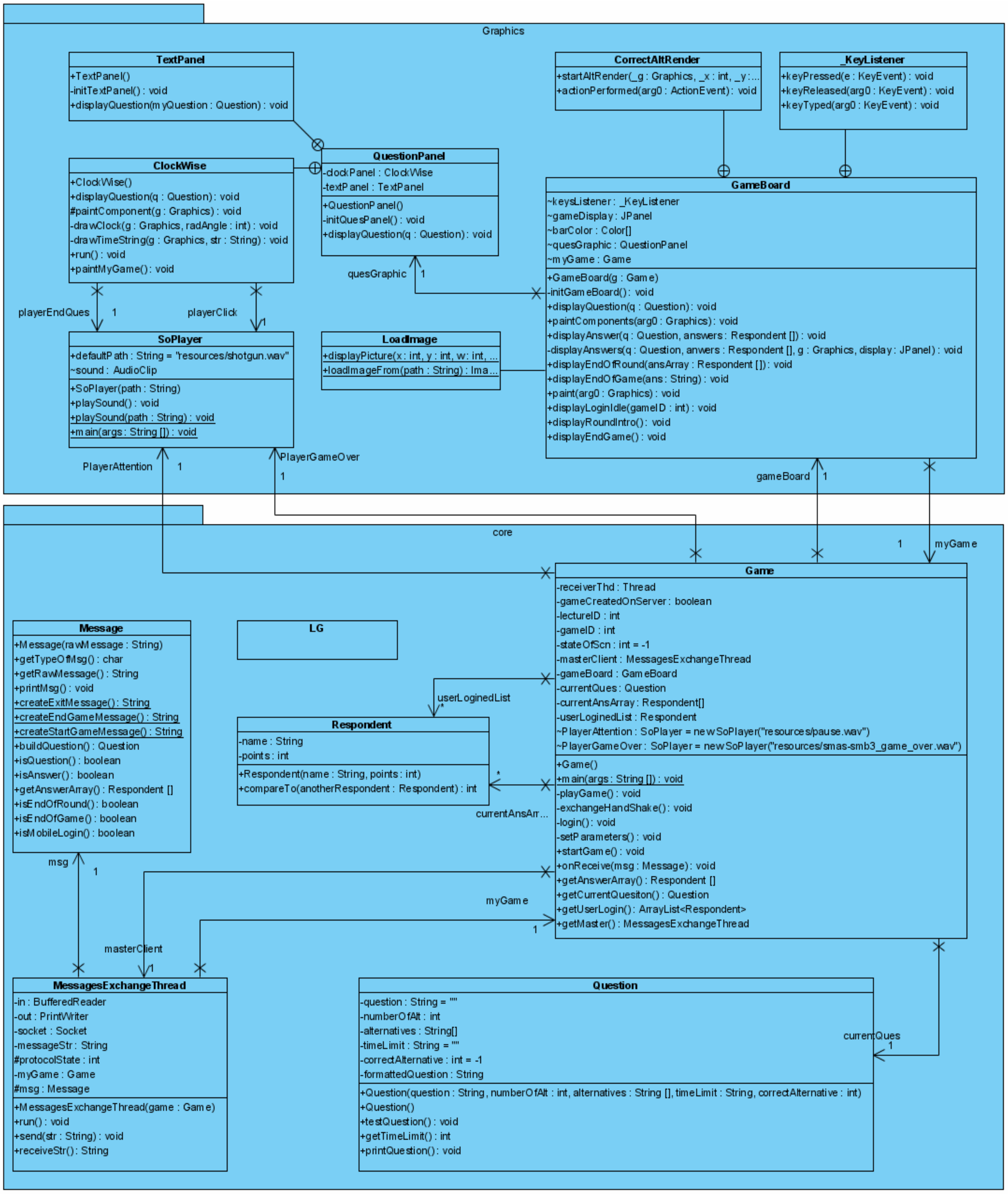

Graphics
TextPanel
+TextPanel()
-initTextPanel() : void
+displayQuestion(myQuestion : Question) : void
CorrectAltRender
+startAltRender(_g : Graphics, _x : int, _y :...
+actionPerformed(arg0 : ActionEvent) : void
_KeyListener
+keyPressed(e : KeyEvent) : void
+keyReleased(arg0 : KeyEvent) : void
+keyTyped(arg0 : KeyEvent) : void
ClockWise
+ClockWise()
+displayQuestion(q : Question) : void
#paintComponent(g : Graphics) : void
-drawClock(g : Graphics, radAngle : int) : void
-drawTimeString(g : Graphics, str : String) : void
+run() : void
+paintMyGame() : void
QuestionPanel
-clockPanel : ClockWise
-textPanel : TextPanel
+QuestionPanel()
-initQuesPanel() : void
+displayQuestion(q : Question) : void
GameBoard
~keysListener : _KeyListener
~gameDisplay : JPanel
~barColor : Color[]
~quesGraphic : QuestionPanel
~myGame : Game
+GameBoard(g : Game)
-initGameBoard() : void
+displayQuestion(q : Question) : void
+paintComponents(arg0 : Graphics) : void
+displayAnswer(q : Question, answers : Respondent []) : void
-displayAnswers(q : Question, anwers : Respondent [], g : Graphics, display : JPanel) : void
+displayEndOfRound(ansArray : Respondent []) : void
+displayEndOfGame(ans : String) : void
+paint(arg0 : Graphics) : void
+displayLoginIdle(gameID : int) : void
+displayRoundIntro() : void
+displayEndGame() : void
quesGraphic
1
playerEndQues
1
playerClick
1
SoPlayer
+defaultPath : String = "resources/shotgun.wav"
~sound : AudioClip
+SoPlayer(path : String)
+playSound() : void
+playSound(path : String) : void
+main(args : String []) : void
LoadImage
+displayPicture(x : int, y : int, w: int, ...
+loadImageFrom(path : String) : Ima...
PlayerAttention
1
PlayerGameOver
1
gameBoard
1
core
1
myGame
Game
-receiverThd : Thread
-gameCreatedOnServer : boolean
-lectureID : int
-gameID : int
-stateOfScn : int = -1
-masterClient : MessagesExchangeThread
-gameBoard : GameBoard
-currentQues : Question
-currentAnsArray : Respondent[]
-userLoginedList : Respondent
~PlayerAttention : SoPlayer = new SoPlayer("resources/pause.wav")
~PlayerGameOver : SoPlayer = new SoPlayer("resources/smas-smb3_game_over.wav")
+Game()
+main(args : String []) : void
-playGame() : void
-exchangeHandShake() : void
-login() : void
-setParameters() : void
+startGame() : void
+onReceive(msg : Message) : void
+getAnswerArray() : Respondent []
+getCurrentQuesiton() : Question
+getUserLogin() : ArrayList<Respondent>
+getMaster() : MessagesExchangeThread
Message
+Message(rawMessage : String)
+getTypeOfMsg() : char
+getRawMessage() : String
+printMsg() : void
+createExitMessage() : String
+createEndGameMessage() : String
+createStartGameMessage() : String
+buildQuestion() : Question
+isQuestion() : boolean
+isAnswer() : boolean
+getAnswerArray() : Respondent []
+isEndOfRound() : boolean
+isEndOfGame() : boolean
+isMobileLogin() : boolean
LG
userLoginedList
*
Respondent
-name : String
-points : int
+Respondent(name : String, points : int)
+compareTo(anotherRespondent : Respondent) : int
*
currentAnsArr...
msg
1
myGame
1
masterClient
1
MessagesExchangeThread
-in : BufferedReader
-out : PrintWriter
-socket : Socket
-messageStr : String
#protocolState : int
-myGame : Game
#msg : Message
+MessagesExchangeThread(game : Game)
+run() : void
+send(str : String) : void
+receiveStr() : String
Question
-question : String = ""
-numberOfAlt : int
-alternatives : String[]
-timeLimit : String = ""
-correctAlternative : int = -1
-formattedQuestion : String
+Question(question : String, numberOfAlt : int, alternatives : String [], timeLimit : String, correctAlternative : int)
+Question()
+testQuestion() : void
+getTimeLimit() : int
+printQuestion() : void
currentQues
1

Figure 15. Master class diagram

9.4 Mobile client class

Mobile client has same architecture as master client, the classes that implements that architecture are presented in the diagram in Figure 16.

In the diagram, the class LeGaMidlet play the role of Game class of master client. The package Graphics of master client is compact into one class which is LegaGameCanvas.

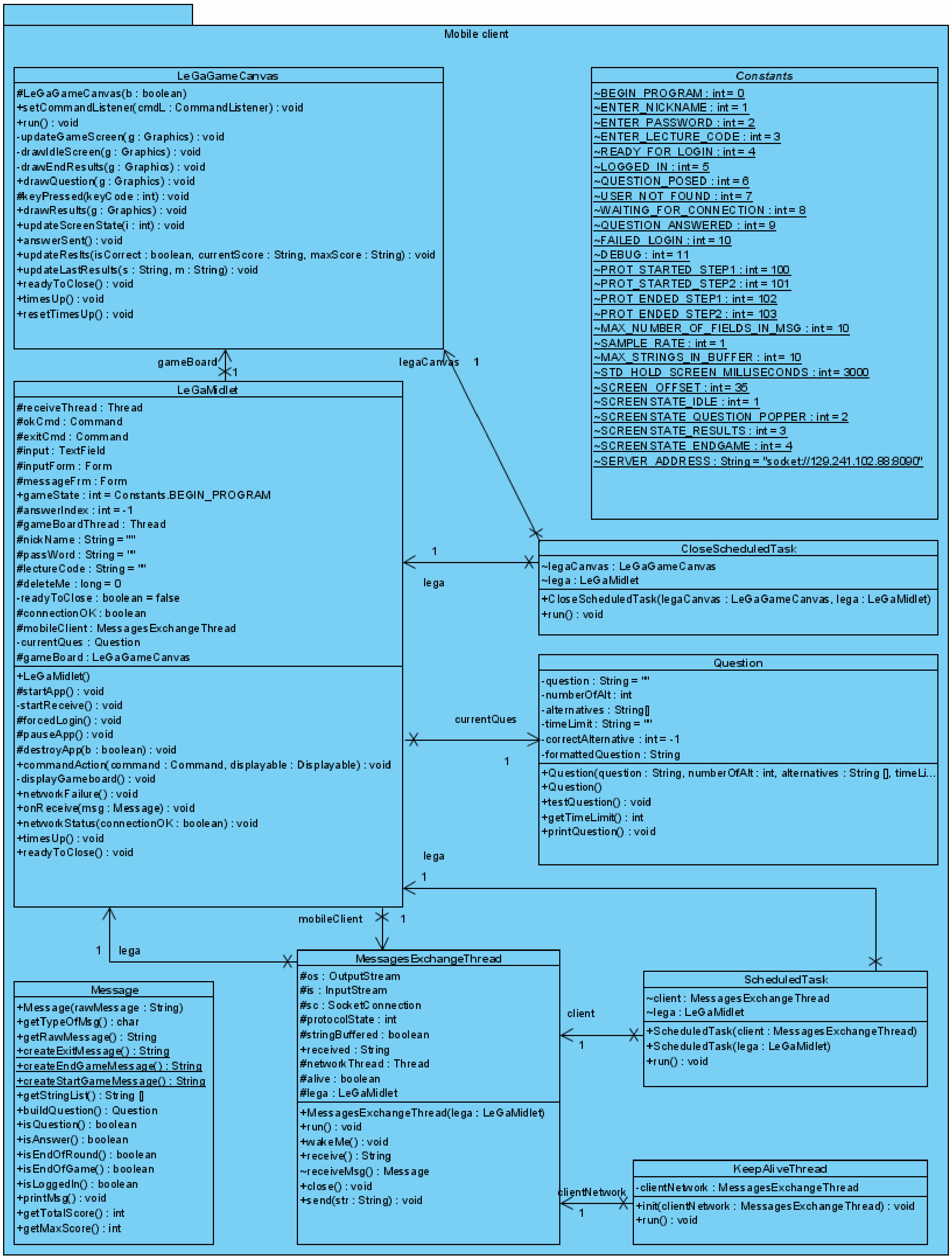

Mobile client
LeGaGameCanvas
#LeGaGameCanvas(b : boolean)
+setCommandListener(cmdL : CommandListener) : void
+run() : void
-updateGameScreen(g : Graphics) : void
-drawIdleScreen(g : Graphics) : void
-drawEndResults(g : Graphics) : void
+drawQuestion(g : Graphics) : void
#keyPressed(keyCode : int) : void
+drawResults(g : Graphics) : void
+updateScreenState(i : int) : void
+answerSent() : void
+updateReslts(isCorrect : boolean, currentScore : String, maxScore : String) : void
+updateLastResults(s : String, m : String) : void
+readyToClose() : void
+timesUp() : void
+resetTimesUp() : void
Constants
~BEGIN_PROGRAM : int = 0
~ENTER_NICKNAME : int = 1
~ENTER_PASSWORD : int = 2
~ENTER_LECTURE_CODE : int = 3
~READY_FOR_LOGIN : int = 4
~LOGGED_IN : int = 5
~QUESTION_POSED : int = 6
~USER_NOT_FOUND : int = 7
~WAITING_FOR_CONNECTION : int = 8
~QUESTION_ANSWERED : int = 9
~FAILED_LOGIN : int = 10
~DEBUG : int = 11
~PROT_STARTED_STEP1 : int = 100
~PROT_STARTED_STEP2 : int = 101
~PROT_ENDED_STEP1 : int = 102
~PROT_ENDED_STEP2 : int = 103
~MAX_NUMBER_OF_FIELDS_IN_MSG : int = 10
~SAMPLE_RATE : int = 1
~MAX_STRINGS_IN_BUFFER : int = 10
~STD_HOLD_SCREEN_MILLISECONDS : int = 3000
~SCREEN_OFFSET : int = 35
~SCREENSTATE_IDLE : int = 1
~SCREENSTATE_QUESTION_POPPER : int = 2
~SCREENSTATE_RESULTS : int = 3
~SCREENSTATE_ENDGAME : int = 4
~SERVER_ADDRESS : String = "socket://129.241.102.88:8090"
gameBoard
1
legaCanvas
1
LeGaMidlet
#receiveThread : Thread
#okCmd : Command
#exitCmd : Command
#input : TextField
#inputForm : Form
#messageFrm : Form
+gameState : int = Constants.BEGIN_PROGRAM
#answerIndex : int = -1
#gameBoardThread : Thread
#nickName : String = ""
#passWord : String = ""
#lectureCode : String = ""
#deleteMe : long = 0
-readyToClose : boolean = false
#connectionOK : boolean
#mobileClient : MessagesExchangeThread
-currentQues : Question
#gameBoard : LeGaGameCanvas
+LeGaMidlet()
#startApp() : void
-startReceive() : void
#forcedLogin() : void
#pauseApp() : void
#destroyApp(b : boolean) : void
+commandAction(command : Command, displayable : Displayable) : void
-displayGameboard() : void
+networkFailure() : void
+onReceive(msg : Message) : void
+networkStatus(connectionOK : boolean) : void
+timesUp() : void
+readyToClose() : void
1
lega
CloseScheduledTask
~legaCanvas : LeGaGameCanvas
~lega : LeGaMidlet
+CloseScheduledTask(legaCanvas : LeGaGameCanvas, lega : LeGaMidlet)
+run() : void
currentQues
1
Question
-question : String = ""
-numberOfAlt : int
-alternatives : String[]
-timeLimit : String = ""
-correctAlternative : int = -1
-formattedQuestion : String
+Question(question : String, numberOfAlt : int, alternatives : String [], timeLi...
+Question()
+testQuestion() : void
+getTimeLimit() : int
+printQuestion() : void
lega
1
mobileClient
1
1
lega
MessagesExchangeThread
#os : OutputStream
#is : InputStream
#sc : SocketConnection
#protocolState : int
#stringBuffered : boolean
+received : String
#networkThread : Thread
#alive : boolean
#lega : LeGaMidlet
+MessagesExchangeThread(lega : LeGaMidlet)
+run() : void
+wakeMe() : void
+receive() : String
~receiveMsg() : Message
+close() : void
+send(str : String) : void
ScheduledTask
~client : MessagesExchangeThread
~lega : LeGaMidlet
+ScheduledTask(client : MessagesExchangeThread)
+ScheduledTask(lega : LeGaMidlet)
+run() : void
client
1
Message
+Message(rawMessage : String)
+getTypeOfMsg() : char
+getRawMessage() : String
+createExitMessage() : String
+createEndGameMessage() : String
+createStartGameMessage() : String
+getStringList() : String []
+buildQuestion() : Question
+isQuestion() : boolean
+isAnswer() : boolean
+isEndOfRound() : boolean
+isEndOfGame() : boolean
+isLoggedIn() : boolean
+printMsg() : void
+getTotalScore() : int
+getMaxScore() : int
KeepAliveThread
-clientNetwork : MessagesExchangeThread
+init(clientNetwork : MessagesExchangeThread) : void
+run() : void
clientNetwork
1

Figure 16. Mobile client class diagram

9.5 Class interaction

In this section we will describe how the class work together to achieve the functions (use cases) of the system. It is not necessary to describe all the use case, instead only the use cases with complicated login behind will be documented. We will use UML notation with sequence diagrams for documenting. The format of some important message that the classes used to communicate will also documented here.

9.5.1 Verification use case of GE

The verification process start when client connect to game engine. If the process is successful then client will login and a new game will be created. The diagram in Figure 17 describes interaction between classes on that process when master client connect to GE.

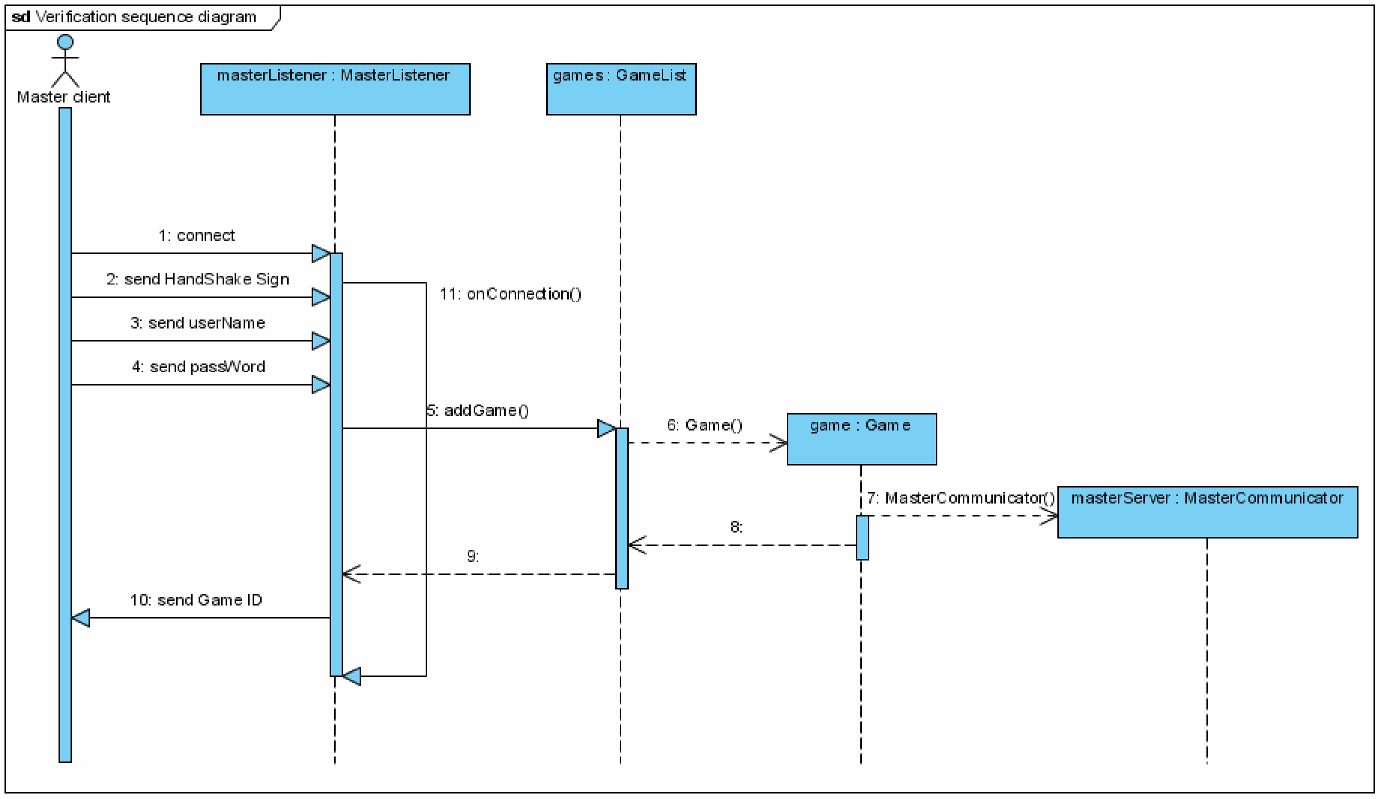


Figure 17. Verification sequence diagram

Right after connection both GE and master client send HANDSHAKE signs to identify each other. After that the client sends user name and password to login.

Server check and send result to client (Currently all user can login.). If the result is ok then server create a new game.

If the new game has been created successful then server sends result to client.

From this time the client will communicate with the game, not game engine. The game thread will run in a separated thread, it runs at the same time with game engine and the listening thread of game engine so that the game engine and create multi game at the same time.

9.5.2 Play game use case (GE)

The interaction between classes to implement the Play game use case is expressed in the diagram in Figure 18.

The classes take part in this use case include GameMode, Game, Round, MasterCommunicator, MobileCommunicator and DataBaseConnection (used by Round class to query the database). And beside that there are two actors, Master client and Mobile client.

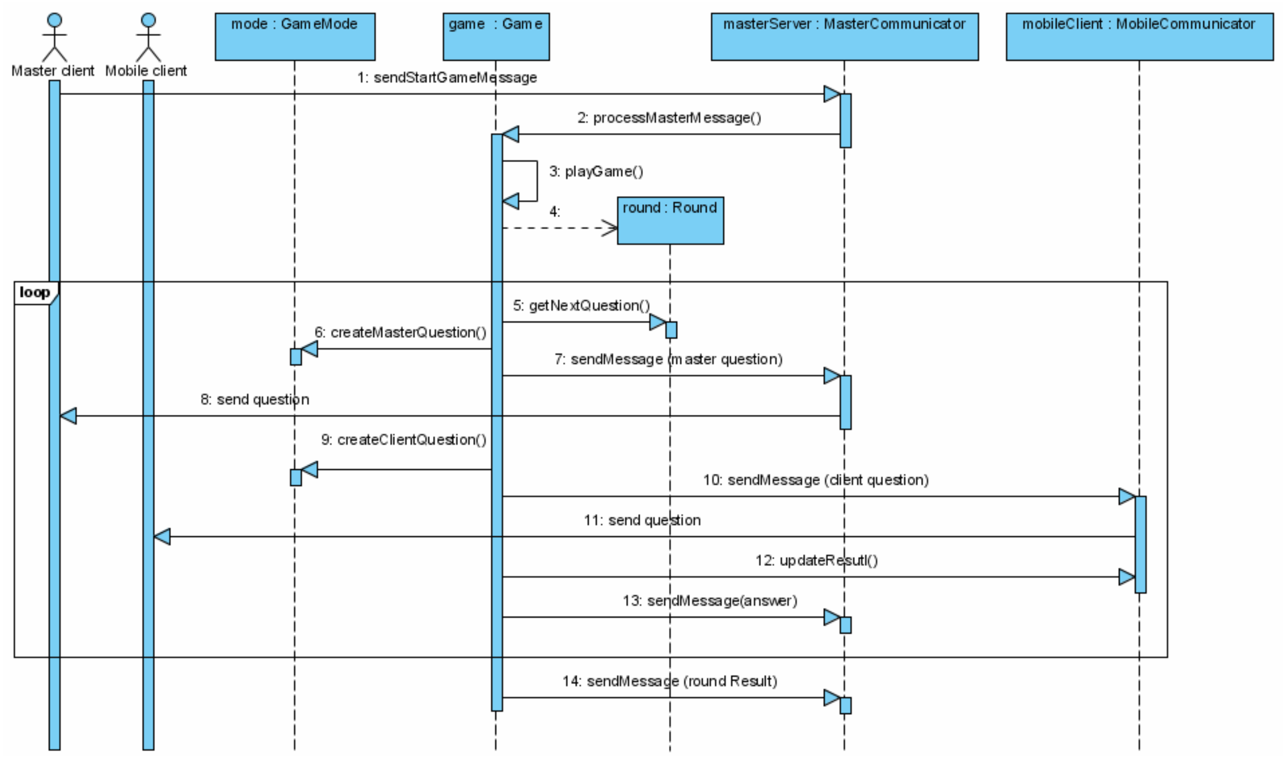

Master client
Mobile client
mode : GameMode
game : Game
masterServer : MasterCommunicator
mobileClient : MobileCommunicator
1: sendStartGameMessage
2: processMasterMessage()
3: playGame()
4:
round : Round
loop
5: getNextQuestion()
6: createMasterQuestion()
7: sendMessage (master question)
8: send question
9: createClientQuestion()
10: sendMessage (client question)
11: send question
12: updateResutl()
13: sendMessage(answer)
14: sendMessage (round Result)

Figure 18. Play game sequence diagram

The master client starts the use case by sending the start game message to masterServer (an instant of MasterCommunicator). The masterServer then send processMasterMessage message to game. In turn game send playGame message to itself. After that a create message will be send to Round class to create a round for the game. Round will contains all questions of the game.

The game then goes to a loop. On each iteration, game will get a question from round, send it to master client and mobile clients. The answers will be collected and send to master client. After the loop, game will summarize the round result and send it to master client.

9.5.3 Message format

Before Creating game, the type of message is recognized by the order of message (the first message is the Handshake sign, the next is the user name, and so on)

After that the message is recognized by the type of message, the first character in the message decides that type. Each type of message is associated with a public static constant in the Message class. The following part describes the formats of Question message and answer message.

- Question message: This is an example of a question message: “7What is a design pattern?|4| A nice hat| A software component| Reusable experience| A design process|30|2”.

    Each message is a string with the first letter is the type of message. In this case the type of message is “7” and it is equal to Message.MSG_QUESTION constant defined in Message class.

    The rest of the string (starts from the second letter) is the data. If there are more then one fields then they are separated by “|”.

    The next field is the question, in this case the question is “What is a design pattern?”, following is the number of alternatives which is “4” in the exam. After that is the list of the alternatives.

    Follows the alternatives is the time period to answer question. In this case, players have 30 seconds to find out the answer.

    The last field is the index of correct answer in the collection of alternatives. The index starts from 0, so in this case “Reusable experience” is the correct answer. Note that in the database the index of alternatives starts from 1 (this presentation associates with the way question is presented to players).

- Answer message: This message is created after game engine has combined answers from mobile players and send it to master client for showing summarization. An example of question is “84|user1|0|user2|2|user3|2|user4|3”.

    The message starts with a letter specifying the type of the message. In this case message type is “8” equals to the constant MSG_ANSWER in Message class.

    The next number is the number of players. There are 4 playes in this case.

The next field is the player's name and his chosen index of alternative. In this case the master client will interpreter that user1 chose alternative 0, user 2 chose alternative 2 and so on.

10 Implementation

This part is will discuss the tools and the environment for coding the classes. The rules for layout the source code are specified. And at the end is policy to deal with exceptions.

10.1 Choosing development environment

When creating a multi users distributed application, there is significant concerns about which technologies are suitable. Costs in terms of risk, development time and the actual price of the technology are some of these factors discussed in the following sections.

10.1.1 Server technology platform

Java and .Net are the two choice of server platform as presented in section 7.1. This section summarizes the reasons behind the selection of technology platform for the game engine between them.

The most important factors when choosing the technology platform for the server is portability. The ideal situation is to have a server application which can run on any platforms. This has several advantages: no need for additional investments in hardware and software to run the game. Also there is the possibility to run the server application on any modern computer and a low threshold for installing and running it. In this light, Java is chosen to build the game engine. In addition to fulfilling the needs for a portable solution, Java has an excellent reputation as a server and middleware platform [29]. There is also a good selection of freely available tools for Java which is not the case for .Net.

10.1.2 Mobile Technology Platform

In this section is the presentation of a rationale choice of mobile technology platforms between J2ME and .Net compact framework.

.NET Compact Framework and J2ME offer the same potential and freedom of implementation in our concept. However, few students own a device supporting the .NET Compact Framework. Availability is an important feature, thus J2ME is the chosen platform of implementation.

To offer a game ‐ like sense to the mobile application, the concept is implemented for MIDP 2.0. It is likely that some students possess a phone that lacks MIDP2.0 support, but this is considered a small loss compared to the increased usability that MIDP 2.0 allows.

10.1.3 Database solution

As presented on section 7.3, there are several competing Database Management Systems (DBMS) on the market today, each powerful and feature rich in their own respects. The two main options for this project are MySQL and Oracle. This part briefly presents the rationale and background for the choice of DBMS.

Portability, price and easy installment, as well as previous experience on our part have contributed heavily towards our decision of choosing the MySQL DBMS as a solution over the other alternatives. All the different database applications mentioned here has readily available Connectors for use with Java which is the project’s development environment. MySQL has abroad user base, and there exist large amounts of tutorials created by the community and other developers, which are freely available on the internet. The version of MySQL used in this project is Community Server 5.0.27 for Win32, which are the newest release containing binaries at this time.

10.1.4 Protocols

The choice of protocol is an important design decision, and imposed its own set of constraints on the prototype with regards to robustness and reliability. As discussed in section 7.4 there are two protocols that can be used in this project: TCP and UDP. These are the two protocols that form the core of the Internet Protocol (IP) suite.

We decided to use the TCP protocol in this project. The reason for this is the inherent robustness and reliability of this protocol. The game concept requires for instance that the student clients installed on the mobile phones and laptop computer shave a persistent connection which the server can use to push out messages. These needs take precedence over the overhead created by TCP in comparison with UDP. We chose to use the same protocol for all the components of the prototype, ensuring interoperability and reducing design complexity.

10.1.5 Communication bearers

The concept to be implemented requires ease of use for large numbers of participants. In this part, we will consider the pros and cons of different communication bearers and how can we use them in our project.

Since the development framework used in this project will be Java and TCP/IP, which implies the use of socket programming, the data bearer is essentially transparent to the applications per se. This, however, does not mean there is not a preferred or expected data bearer for the different components of the prototype. The server will typically be a stationary workstation or a server computer which is always on and always connected to the internet. This warrants the use of Ethernet over twisted pair, as it is a stable and widely available communication bearer on university campuses. The teacher client will typically connect to the server from a class room. Both Wi‐Fi and Ethernet over twisted pair are suitable for this, as these communication bearers are usually available in a class room. In some cases we imagine 3G, EDGE or even GPRS can also be used if neither Wi‐Fi nor Ethernet over twisted pair is available. The amounts of data to be transferred between the server and the teacher clients are miniscule, so the bandwidth will not be an issue.

The student clients will be run by the students, situated inside the lecture hall, on either a mobile phone supporting J2ME or a laptop computer. This requires wireless connectivity; preferably GPRS, 3G or Wi‐Fi, as most new mobile phone support one or several of these communication bearers. As UMTS (3G) will increase its market penetration, we must expect a reasonable amount of student clients will be connected over UMTS in the future. Some newer handsets also support Wi‐Fi, and in lecture halls with Wi‐Fi coverage, we will not be surprised if the students choose to connect over this communication bearer, as it will be free of charge for the users.

10.2 Naming convention

In this section we discuss the naming rules for classes, objects, instances, methods and variables. It is important to make names as descriptive as possible when implementing. Because the code might be used and expanded by others later on, it is more important that code is easy to understand, than easy to write. The implementation is divided into three main implementation modules:

10.2.1 Classes

For naming of classes, the following rules shall be applied:

- ➢ The different parts of the name shall start with a capital letter.
- ➢ The name shall be descriptive and easy to understand.
- ➢ Abbreviations used as a part of the name shall consist only of capital letters.

10.2.2 Instance of class

For naming of instances, the following rules shall be applied:

- ➢ The name shall start with a lowercase letter.
- ➢ The name shall star with an abbreviation of the object owner or attributes.
- ➢ The next part shall star with a Capital letter, the name shall represent the class which the object belong to.
- ➢ The last part is an option, shall with a Capital letter and with an abbreviation of the class.

10.2.3 Variables

For naming of variables the following rules shall be applied:

- ➢ Variable names shall be given a name that reflects what kind of value they contain.
- ➢ Variable names can either use lowercase or mixedCase. mixedCase is used if the name is composed by several words, with each separate word starting with an upper case letter such as longNameForVariable.
- ➢ Variable names should preferably be self-explanatory. The exception is if the variable is often referred to or is a counting variable.

10.2.4 Methods

For naming of regular methods the following rules shall be applied:

- The name shall start with a lowercase letter.
- The name shall start with a verb.
- Other parts of the name shall start with a capital letter.
- The name shall be descriptive and easy to understand.

10.3 How to deal with error

Errors or exceptions should be deal with as soon as possible when it happened. In the layers of application, some time the error happened at low level, if have enough information to process then process it right at that time. If the errors need more information to process, then forward (throw) it to the next level to process.

11 Deployment

This part describes the process of packaging, distribution and installation of different parts of the software and library across different hardware systems. The part of game engine and database will be deployed on a server or a workstation that always connect to internet while the master client is installed on a PC. The last part will runs on a shell phone that enable Java.

11.1 Deployment of game engine

Both of source code and binary of game engine are delivered. In this section we are going to describe how to compile and run the game engine.

11.1.1 Compiling the source code

The environment using to compile game engine source code is Eclipse version 3.2. Game engine use additional library **Conector/j Driver mysql-connector** for managing database. The process of compiling game engine is documented in Appendix A.2

11.1.2 Install the database

The DataBase Management System (DBMS) used in the game is MySQL, for instruction of to set up the database see Appendix A.5

11.1.3 Installation of GE and configuration

GE is delivered as a .jar file and a configuration file. The configuration file is a .txt file with 4 lines. The first two lines are the username and password to access the database. The next 2 lines are the port numbers where the GE listens for connection from master client and mobile clients accordingly. Modify the configuration file to with the appropriate information.

The computer that hosts GE can be a server or a work station. The server must have a fixed IP address or hostname that can be seen form internet. Server uses socket with specific ports to communicate with clients. Make sure there ports are not blocked (for example by some firewall programs).

The game has been tested with Dell E1405 laptop connecting to internet at NTNU campus. At NTNU, computers often have Cisco VPN client installed with a built in firewall that always turn on by default, it should be turn off for the game engine to run.

GE is installed just by copying the two file and put in the same directory. After that one can start the game engine just double click on its .jar file. Before running game engine, JVM must be installed; the version required is 5.0 or higher. JVM can be downloaded form http://www.java.com/en/download/index.jsp.

11.2 Deployment of master client

This section describes the process to compile and run the master client

11.2.1 Compiling the master client

The environment using to compile master client is Eclipse version 3.2. Master client is built with pure java API, so there is no need to install additional library. The process of compiling game engine is documented in Appendix A.3

11.2.2 Installation of master client and configuration

Master client is delivered as a .jar file. In addition there are a configuration file and a graphic file. The configuration file contains the address of game engine and the port that game engine listens for connection.

The computer hosts master client needs to have JVM version 5.0 or higher. This computer need also connect to the game engine. There are a number of connection options available such as wireless network, twisted pair cable or event the internet. In the test of game master client connected to game engine in a LAN.

11.2.3 User guide

To activate teacher client, double click the .jar file. The first screen of game will be shown with a logo. The game then waits for clients to join as in Figure 19.

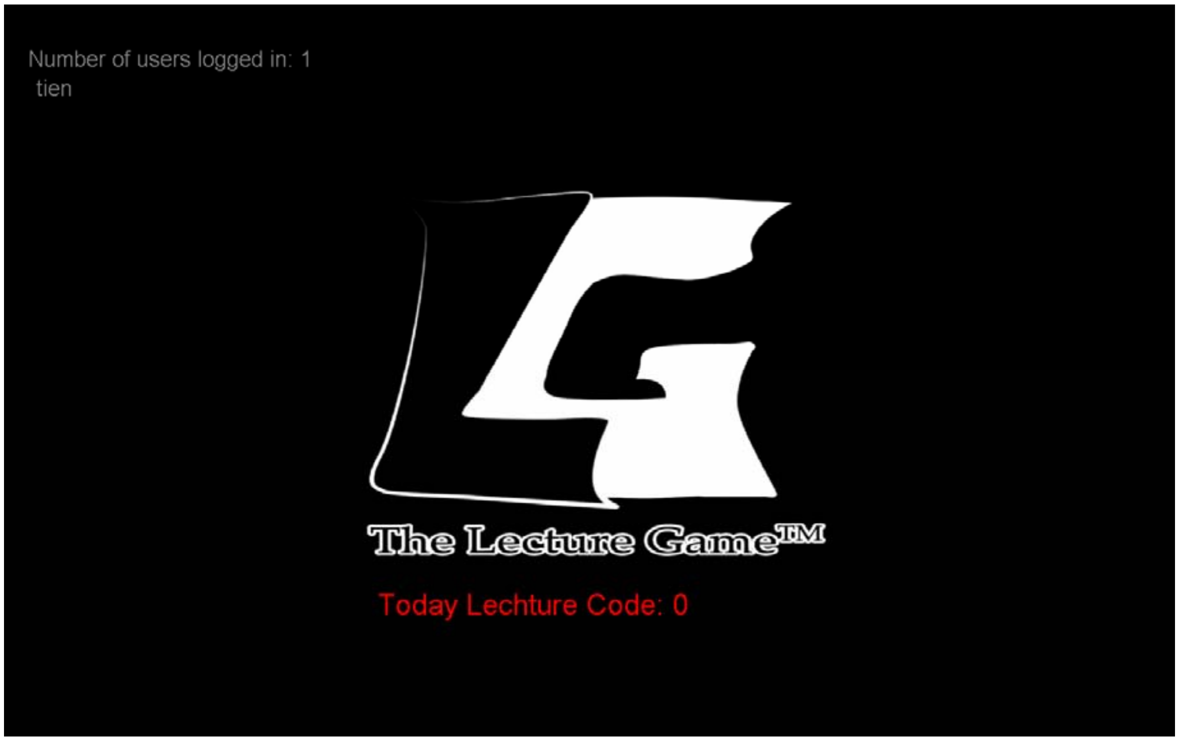


Figure 19. Waiting screen

To move to the next screen press the space bar. The game introduction will display as in Figure 20.

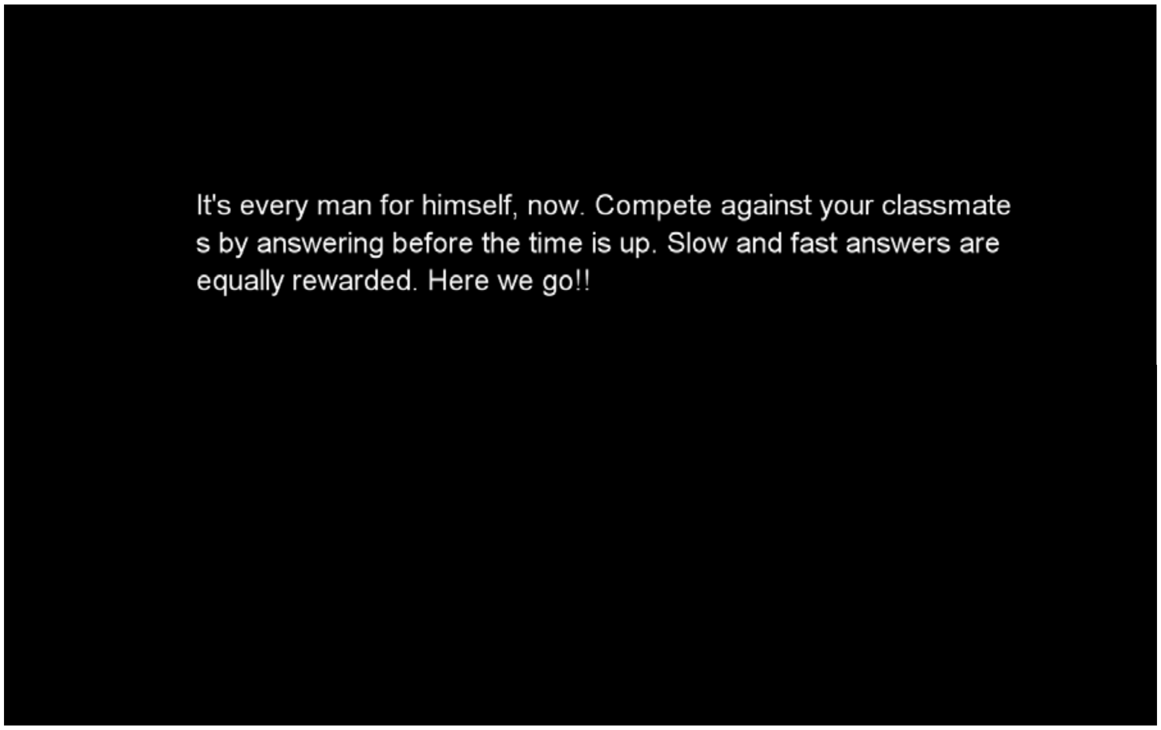
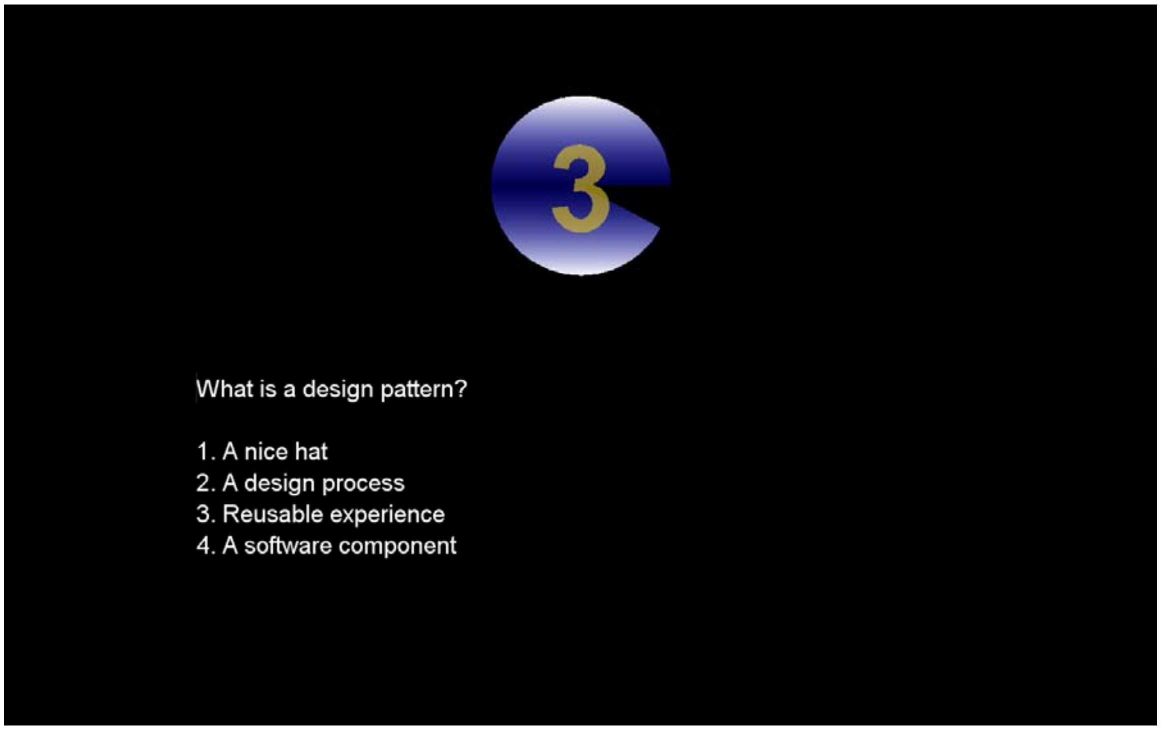

It's every man for himself, now. Compete against your classmate
s by answering before the time is up. Slow and fast answers are
equally rewarded. Here we go!!

Figure 20. *Introduction screen*

To start playing game, press space bar again. The game then will iterate through all the questions. An example of a question is shown on Figure 21



What is a design pattern?

1. A nice hat
2. A design process
3. Reusable experience
4. A software component

Figure 21. *Question is displayed on the teacher's PC*

When the game finishes, press Escape to exit game, press space bar to play again.

11.3 Deployment of mobile client

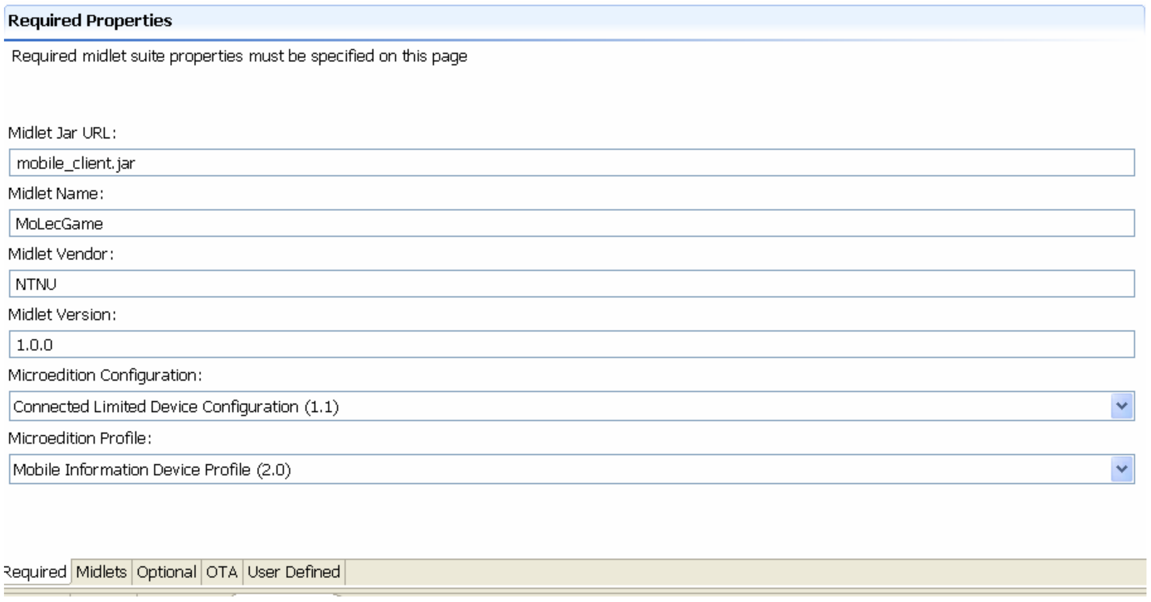
In this section we present how to package the mobile client and how to deliver it using a WAP site.

11.3.1 Package the mobile client

The mobile client runs on a shell phone that supports MIDP 1.0 specification. This section gives instruction to package the mobile client using Eclipse, for the guide of compiling mobile client see Appendix A.4.

To create the .jar and .jad files for shell phone, following these steps;

Open the MobileClient project in Eclipse, then open mobile_client.jad file. Make sure the configuration is the same as in Figure 22 below



Required Properties

Required midlet suite properties must be specified on this page

Midlet Jar URL: mobile_client.jar

Midlet Name: MoLecGame

Midlet Vendor: NTNU

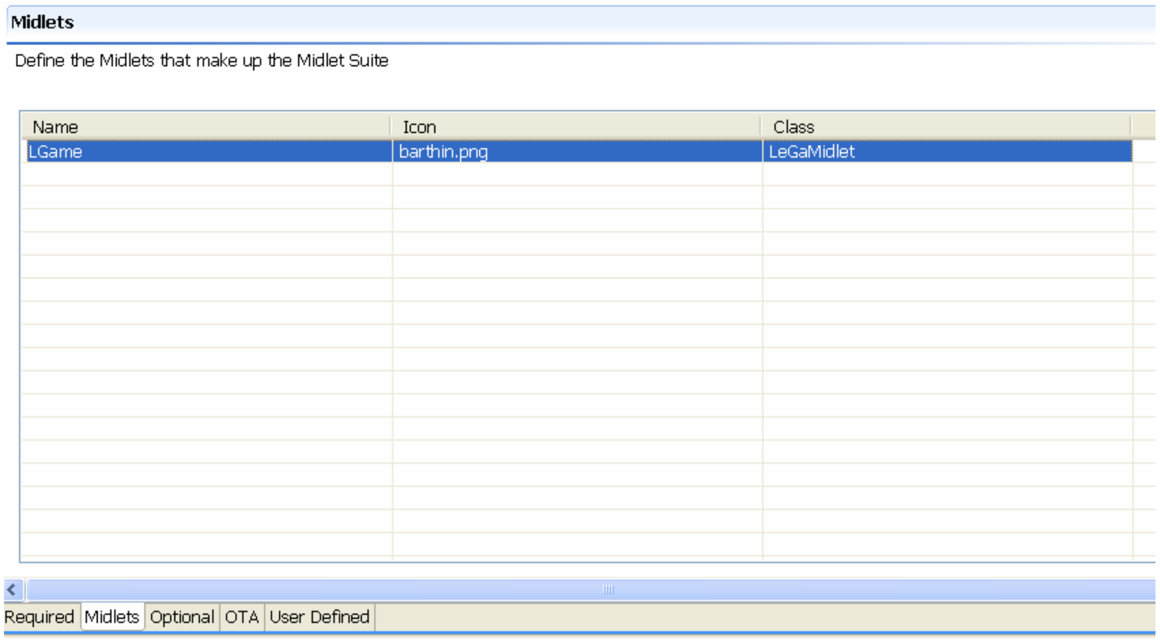
Midlet Version: 1.0.0

Microedition Configuration: Connected Limited Device Configuration (1.1)

Microedition Profile: Mobile Information Device Profile (2.0)

Figure 22. Configure the Jad file

Change to the Midlets tab, click “add” button to add a midlet, make sure to enter the information as in the Figure 23


Midlets

Define the Midlets that make up the Midlet Suite

Name	Icon	Class
LGame	barthin.png	LeGaMidlet

Figure 23. Configure the midlet class

The last step is to create the jar file. On the package explorer view (if it does not show, then activates it by choose Window → Show View → Package Explorer), right click on the project name and choose J2ME → Create Package as Figure 24.

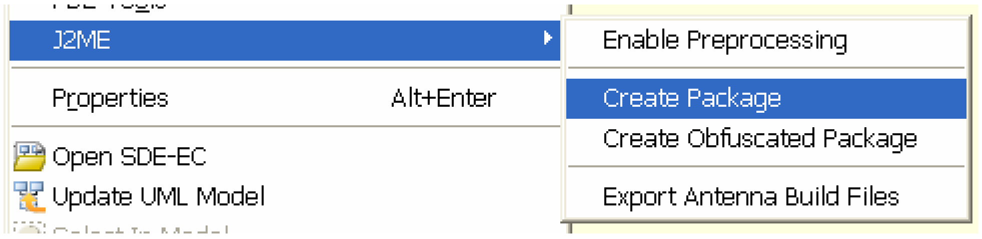


Figure 24. Package mobile client

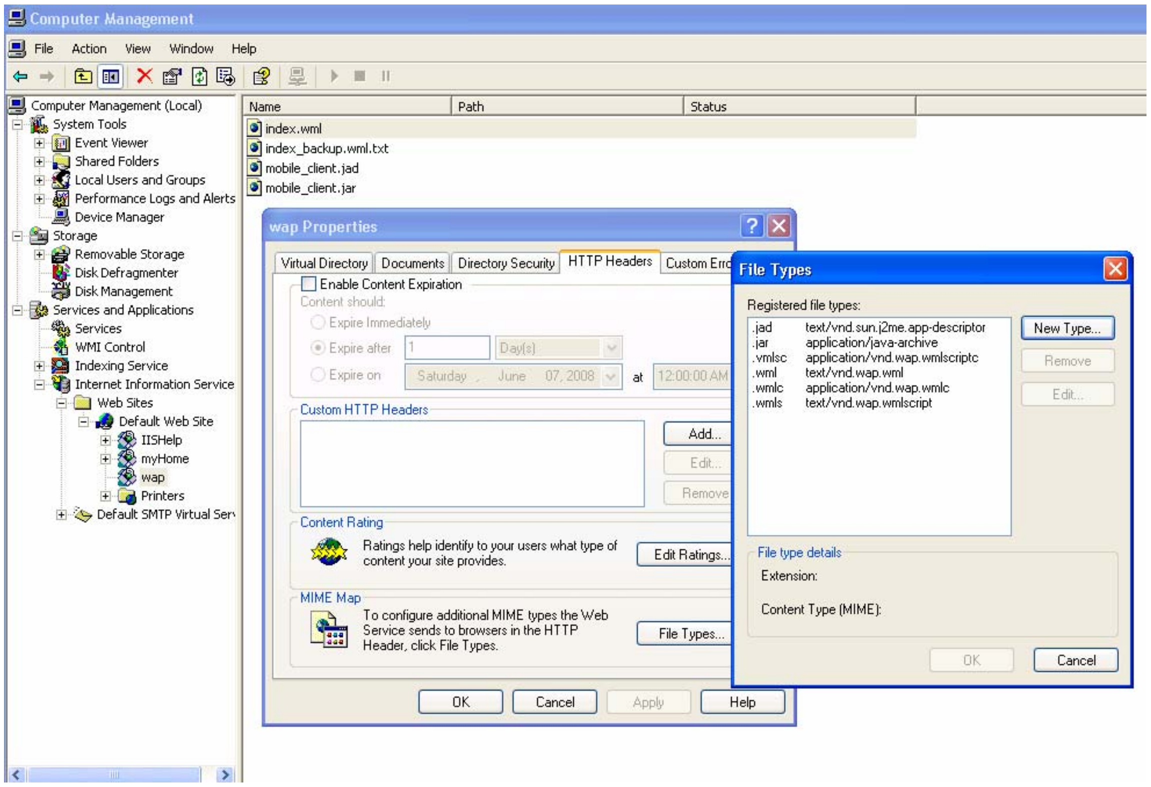
Make sure there is no emulator is running otherwise there will be an error. If everything is ok then Eclipse will create 2 files mobile_client.jar and mobile_client.jad in “deployed” folder in the project directory.

11.3.2 Delivery of mobile client part

This section will give instructions of how to deliver the mobile package using WAP site. It is possible to use any web server that supports WAP. In this project we use IIS from Microsoft. Some knowledge of create a web site using IIS is assumed. In the following parts we give instructions on how to configure the website so that IIS can understand the WAP.

Make sure your computer has IP address or a domain name that can be pinged form internet. This condition is important because you are going to put the jar file there for shell phones to download using GPRS.

Create a virtual directory using IIS, to configure this directory as a WAP site, change the mime type as Figure 25:



Name	Path	Status
index.wml		
index_backup.wml.txt		
mobile_client.jad		
mobile_client.jar		

Enable Content Expiration
Content should:
Expire Immediately
Expire after 1 Day(s)
Expire on Saturday , June 07, 2008 at 12:00:00 AM

Custom HTTP Headers

Content Rating
Ratings help identify to your users what type of content your site provides.
Edit Ratings...

MIME Map
To configure additional MIME types the Web Service sends to browsers in the HTTP Header, click File Types.
File Types...

Registered file types:

.jad	text/vnd.sun.j2me.app-descriptor
.jar	application/java-archive
.vmlsc	application/vnd.wap.wmlscriptc
.wml	text/vnd.wap.wml
.wmlc	application/vnd.wap.wmlc
.wmls	text/vnd.wap.wmlscript

File type details
Extension:
Content Type (MIME):

Figure 25. *Configure mime type in IIS*

Create an index.wml file that contains entry for the .jar and .jad you want to deliver, a simple example is:

<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"

"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>

```
<card title="Download">

<p>Download the Jar file and istall on your phone:</p>

<p>

<a href="mobile_client.jar">mobile_client.jar</a>

</p>

<p> <a href="mobile_client.jad">Download (JAD)</a> </p>

</card> </wml>
```

If every thing go well then you can download the jar file in a GPRS enable shell phone by go to the wap site http://url_of_your domain/name_of_vitual_directory/index.wml such as http://129.241.103.165/wap/index.wml .

Part V: Evaluation and Conclusion

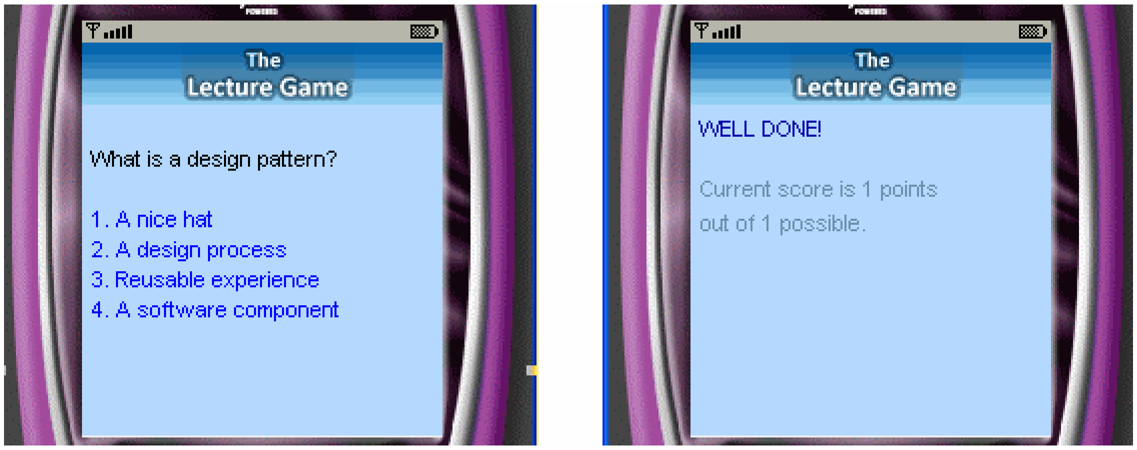
12 Evaluation

This section presents the evaluation of our major findings during the project. We will also present here our experiment during the development process.

12.1 Evaluation of new implementation

It is very challenging to implement a game that spanning through different platforms and distributed environment. That is why we are happy to end up our project with a run-able game. We have conducted several tests on the game to make sure it running correctly.

The user test has been conducted on the real environment at NTNU campus with 3 mobiles phone and 3 emulators running on 2 laptops and a PC. Figure 26 shows how a question and a feedback are sent during the game and how they are shown on a mobile phone's screen. Figure 27 and 28 show the question and feedback projected on a big canvas. All the functional requirements are implemented and run correctly.



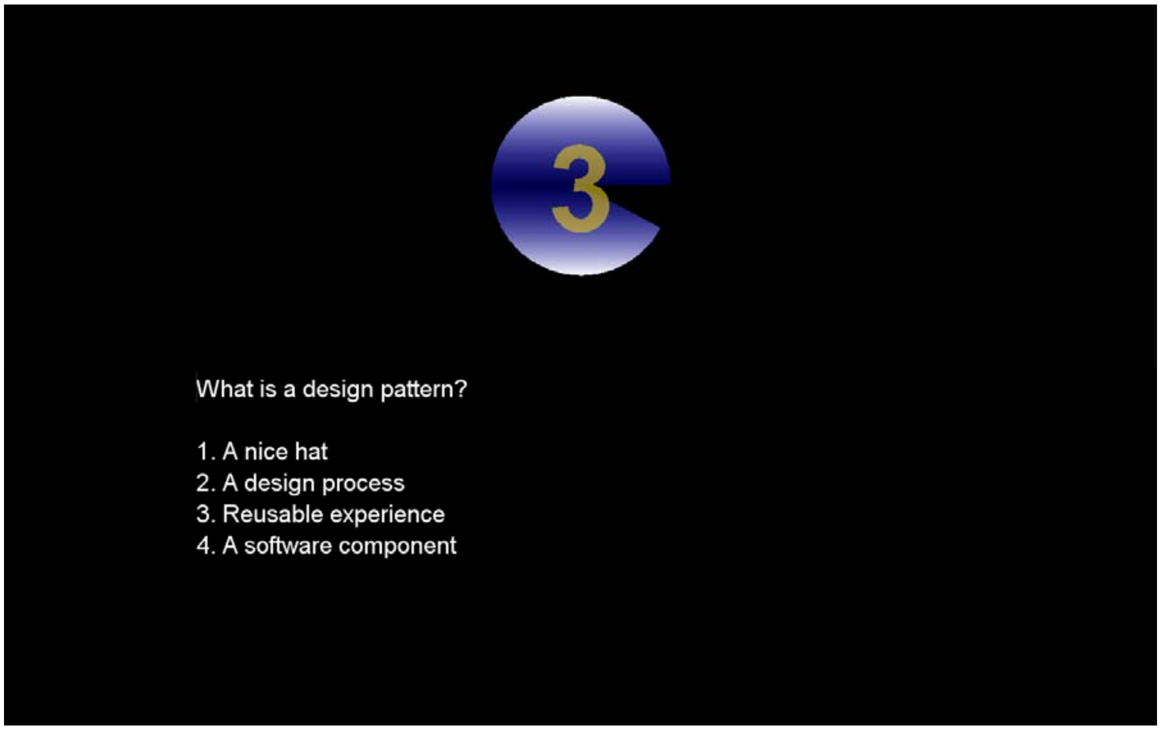
What is a design pattern?

1. A nice hat
2. A design process
3. Reusable experience
4. A software component

WELL DONE!

Current score is 1 points
out of 1 possible.

Figure 26. Question and feedback on mobile phone


What is a design pattern?

1. A nice hat
2. A design process
3. Reusable experience
4. A software component

Figure 27. *Question showing on big canvas*

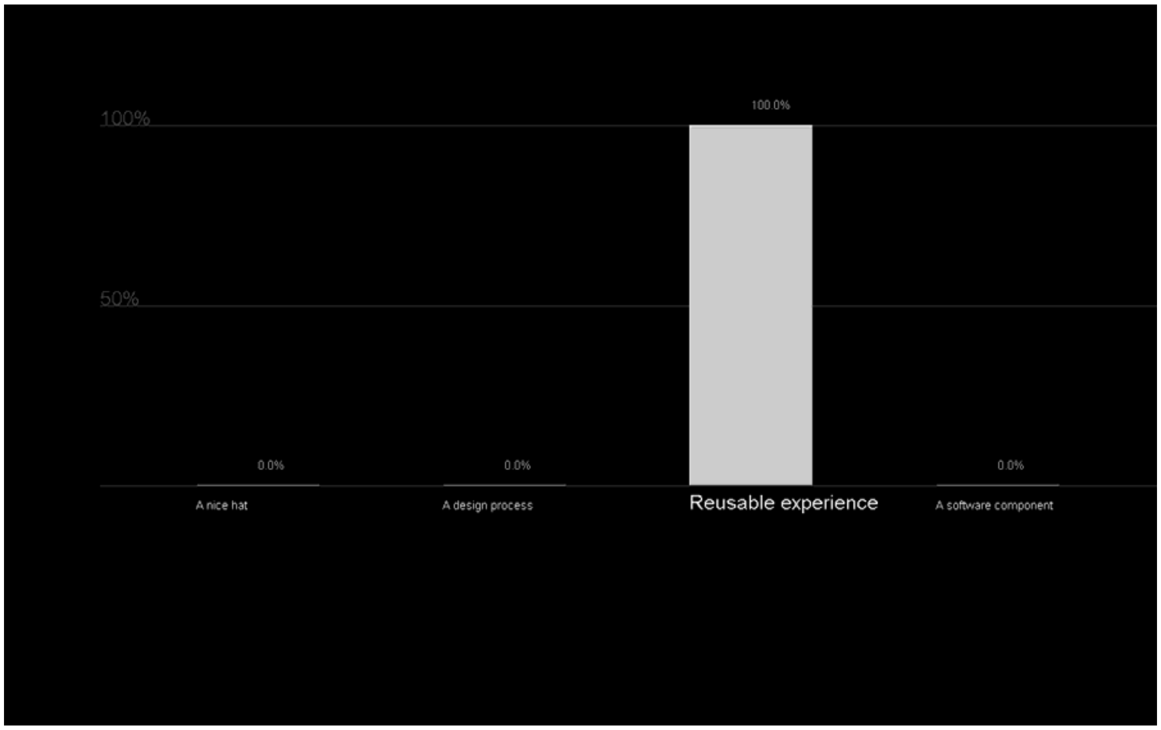


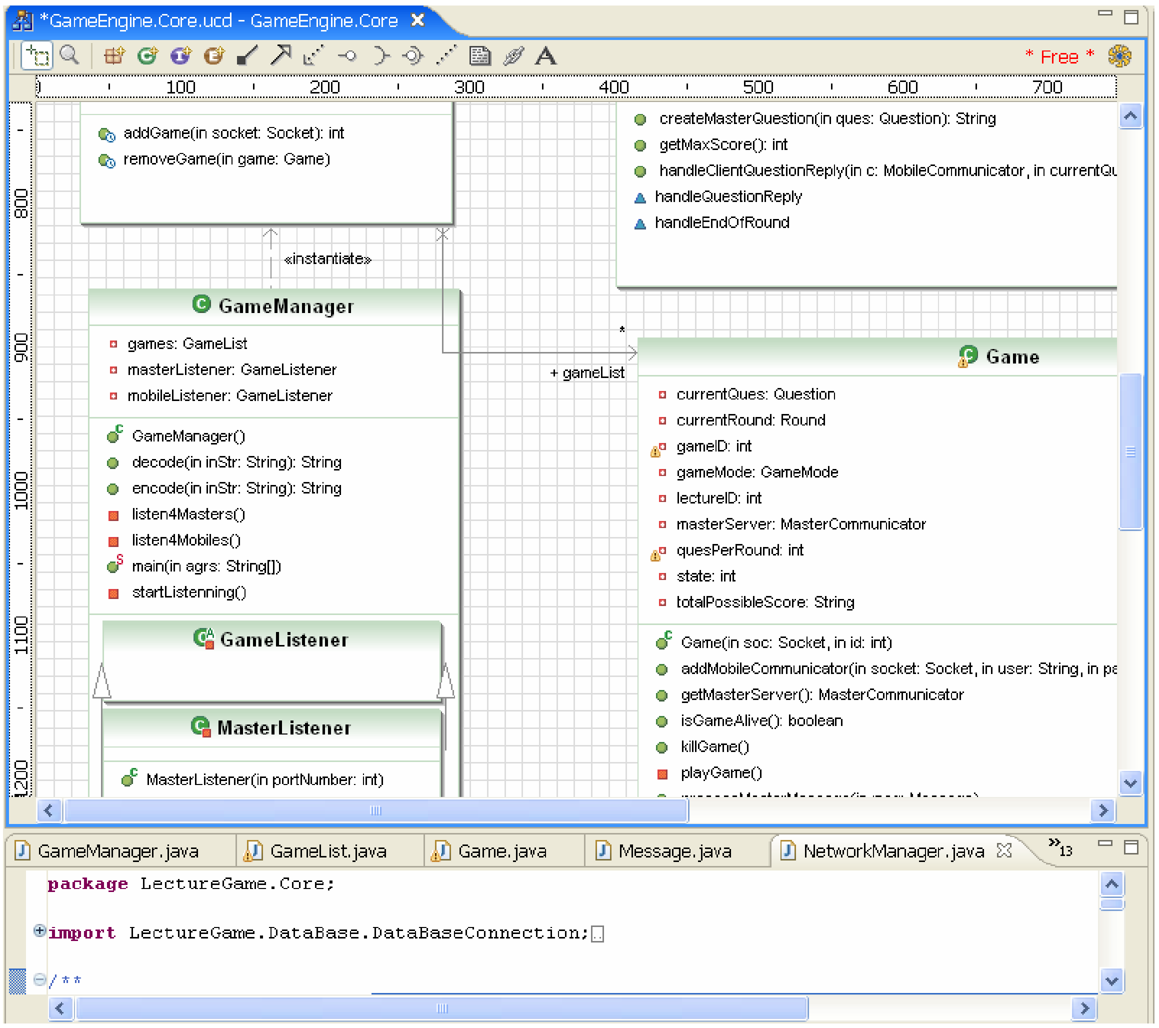
Figure 28. *Feedback on big canvas*

In our performance test of the game within the development environment, a normal PC can server up to 10 games at the same time. The PC in the test is a laptop running WindowXP with the Intel T2300 processor 1.66MHz and 1Gb RAM. Also the PC activates a master client and several mobile simulators. All the functions of game engine, master client and mobile client are correct according to the specification in section 8.1.3.

We experience a problem of delay when logging into the game engine from mobile phone during the daytime. On the daytime it may need up to 2 seconds to log in, meanwhile at night the logging process happens immediately. But such a problem does not affect the experiment to play game because after the connection, data transmission speed is fast enough, no delay is significant.

12.2 Development process

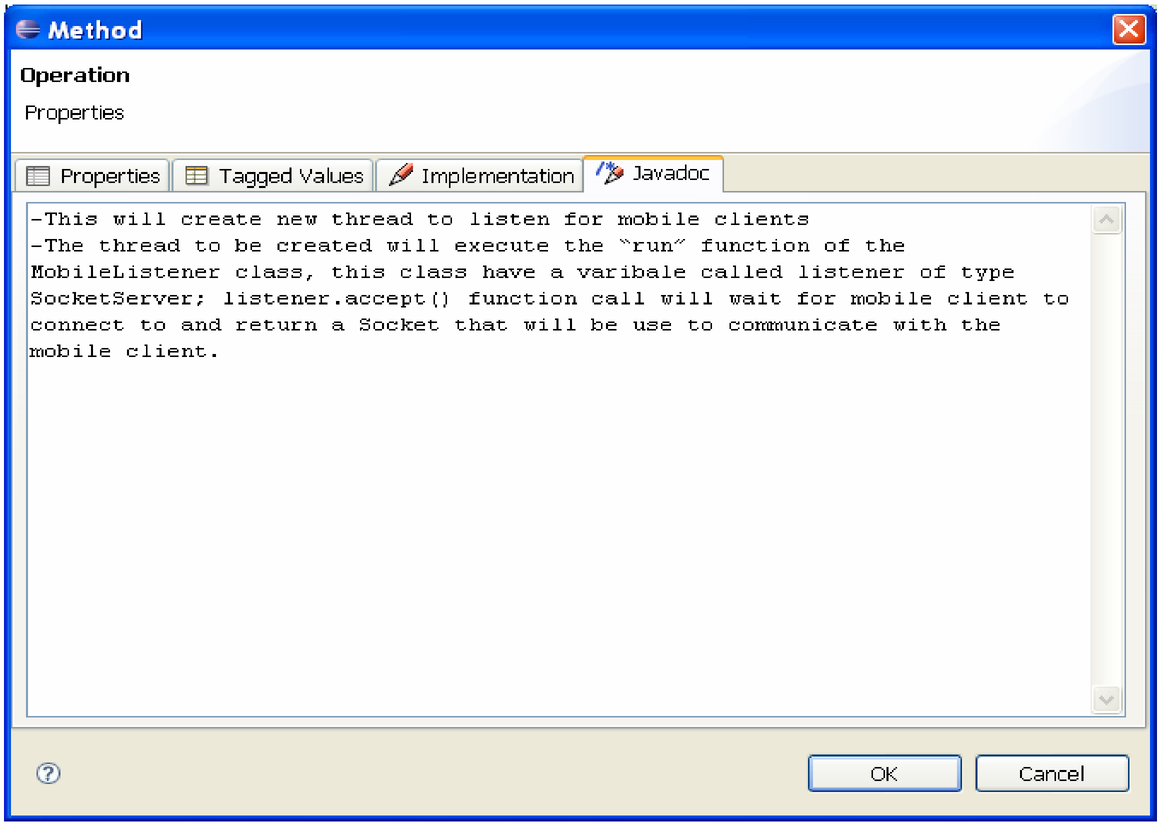
We use iterative method to develop our game. This is an important factor leading to the successful of our project. As we image from the beginning of the project the implementation is very challenged so that we put more focus and made several iterations on the prototype phase as on Figure 1 (at the begin of the report). In our project the first iteration is to define and implement the class of main modules which are the game engine, the master client and the mobile client. The class is first implemented with the name of its methods together with methods' description and attributes. The methods then will be refined gradually to implement their functions. The refined process uses the top down approach.


*GameEngine.Core.ucd - GameEngine.Core
* Free *
addGame(in socket: Socket): int
removeGame(in game: Game)
«instantiate»
GameManager
games: GameList
masterListener: GameListener
mobileListener: GameListener
GameManager()
decode(in inStr: String): String
encode(in inStr: String): String
listen4Masters()
listen4Mobiles()
main(in agrs: String[])
startListenning()
GameListener
MasterListener
MasterListener(in portNumber: int)
createMasterQuestion(in ques: Question): String
getMaxScore(): int
handleClientQuestionReply(in c: MobileCommunicator, in currentQu
handleQuestionReply
handleEndOfRound
+ gameList
Game
currentQues: Question
currentRound: Round
gameID: int
gameMode: GameMode
lectureID: int
masterServer: MasterCommunicator
quesPerRound: int
state: int
totalPossibleScore: String
Game(in soc: Socket, in id: int)
addMobileCommunicator(in socket: Socket, in user: String, in pa
getMasterServer(): MasterCommunicator
isGameAlive(): boolean
killGame()
playGame()
GameManager.java
GameList.java
Game.java
Message.java
NetworkManager.java

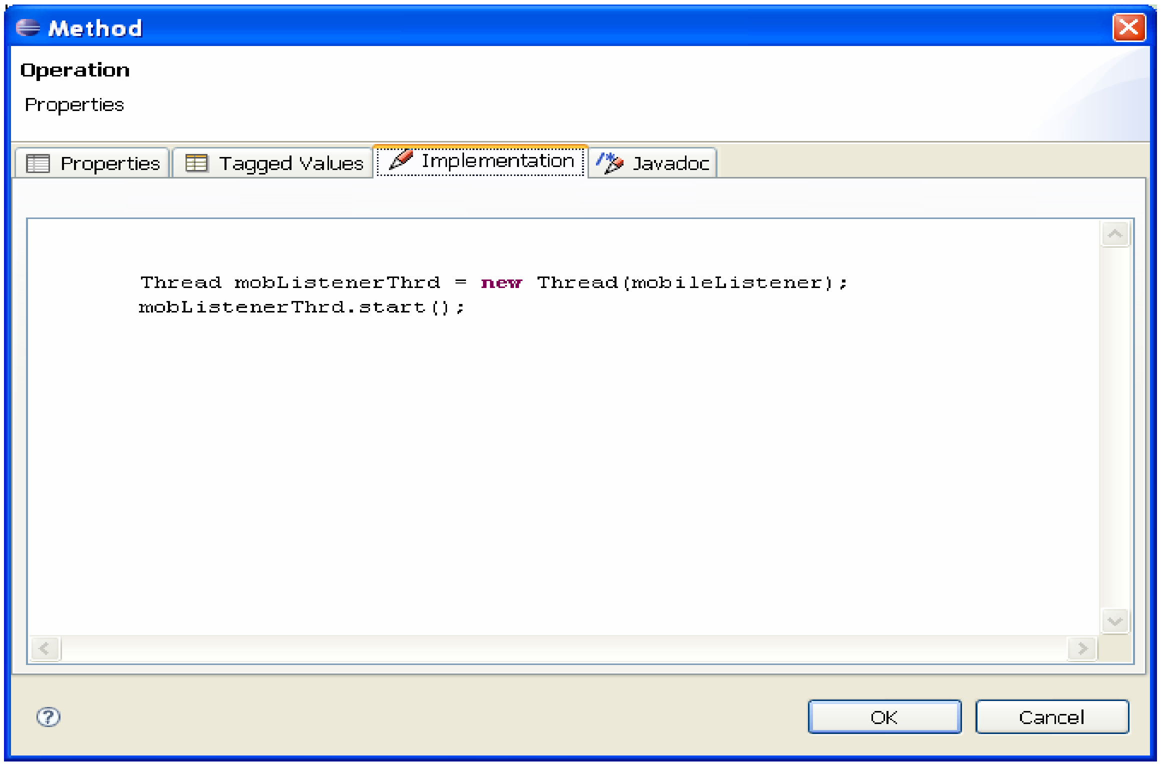
```
package LectureGame.Core;

import LectureGame.DataBase.DataBaseConnection;

/**
```

Figure 29.   Create classes with Eclipse UML

Our approach is supported by using a case tool call “Eclipse UML” in Eclipse environment. The tool allows us to implement the class visually and then it can generate the code later. We think that the tool like this is very helpful for iterative process with top down approach. First we use the tool to create classes visually as in Figure 29, and then we add methods, properties. The tool can be used to describe what the method does and how it can do that. We also can use the tool to implement the method and it will generate code later. Figure 30 shows how we have done the refined process in our project.



```
-This will create new thread to listen for mobile clients
-The thread to be created will execute the "run" function of the
MobileListener class, this class have a varibale called listener of type
SocketServer; listener.accept() function call will wait for mobile client to
connect to and return a Socket that will be use to communicate with the
mobile client.
```



```
        Thread mobListenerThrd = new Thread(mobileListener);
        mobListenerThrd.start();
```

Figure 30. *Method refined process starting with descriptions and then implementation of listen4Mobiles() method.*

We do not only use the top down method but also the bottom up method. The bottom up method is used when there is some class that we can start coding immediately. For example, a class to play some sound in our game is the kind of class can be implemented without waiting for detail design. The bottom up method is often useful in projects that have more than one programmer, but it also works in our project with only one programmer; in the way that it helps to create other new classes when we integrate the low level classes.

We experienced several problems during the implementation process. The first problem was the limitation of some free version and this caused some conflicts. For example the Eclipse UML that we used to implement classes and to generate the code did not work with subversions that we used to manage version control. We have tried to overcome this problem by disconnecting from subversion every time we use Eclipse UML. Other example of tool problem came from a tool that we used to reverse engineering; we used this tool to create the class diagrams, because it is a trial version so we could not use its export function. Our solution was to use a capture screen tool to copy class diagrams.

We also faced a problem to debug the game in only one laptop. The game has three parts that runs on different platforms. To debug the game we need to run all the parts. In our project we addressed this problem by two solutions. The first is to run three instant of Eclipse simultaneously; each instant has a separate work space. The second solution is to use an Emulator to emulate the mobile phone.

In our project we decide to use Java APIs for drawing graphics. Figure 31 is an example of graphics that we built in our game showing a count down clock. We have tested the game on a Macbook and a Window computer. By using Java APIs, our application can run on any platform that supports Java. Compare to the previous game this is another improvement feature because the previous game does not work on a Macbook. The challenging when using native Java APIs for graphics is that we have to spend a significant amount of time to build all the animations, graphics rendering, and text rendering functions from scratch.

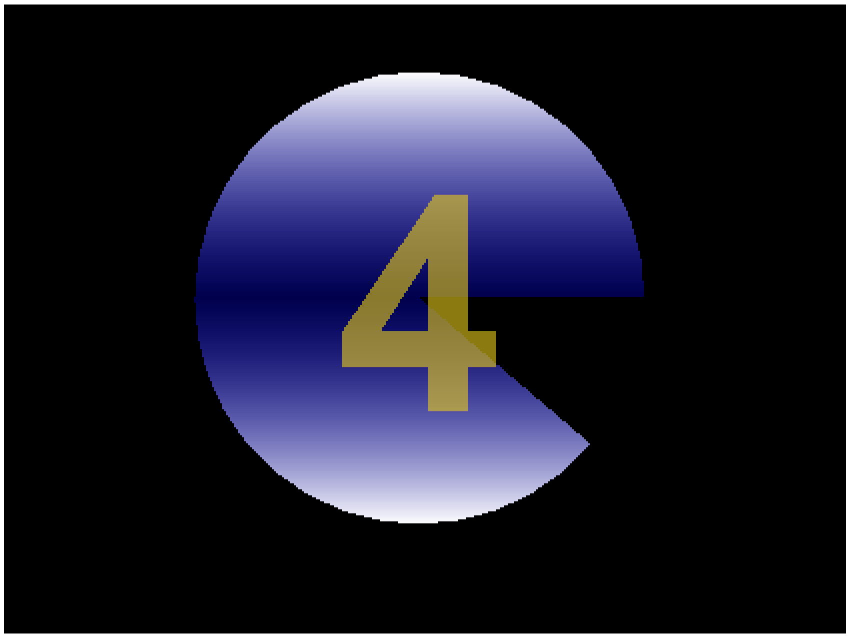


Figure 31. *An animation building with native Java APIs*

12.3 Team work

Having a good team work was an important factor contributing to the success of our project. During our project, meetings were conducted weekly. We used artifacts like papers, pencils, and computers for exchange information and making decisions. Some of papers are shown on Figure 32.

Every meeting we discussed what we had done and came up with the new ideas and work for the next week. Having been working together and meeting every week helped us to make the necessary improvements, to set new goals, to build and manage work plan and to reach the final goal of the project.

Figure 32. *Team work artifacts during project*

13 Conclusion

Based on the findings in the project, the answer for our research questions is presented as follow.

RQ1: Which features of the previous lecture quiz game can be improved in term of gameplay of the system?

We have analyzed the document as well as diged into the source to evaluate the current game. The results have been documented in part II. There are a number of improvements we can make to the current game and we choose to address two of them in this version of the game which are supporting multi games at the same time and enhance the communication layer.

RQ2: What architecture is suitable for this kind of game?

We have proposed and implemented an architecture for the multi games. The architecture is based on client-server architecture. We have decided to use the same architecture for mobile client and master client to simplify the development. The new architecture dedicates a component to manage the data communication. This guarantee the easily changes to adapt to new requirements in the future. The architecture is not only easy to understand but it also can be implemented. The new version of the game is built on the architecture showing that it is suitable for supporting multi games.

RQ3: Implement a lecture game based on the proposed architecture.

Base on the finding in RQ1 and RQ2, we have tailed the architecture and implement a new version of the game. The requirements, design, implementation, and deployment processes have been documented carefully in our report. All the functions of the game have been implemented and tested.

Compare to the previous the new version supports multi games, has clear defined architecture. The design document is also an important part of deliverable in our project.

14 Future work

Supporting multi games is the first step to make a practical game to be accepted and used in the real life. How ever, as the evaluation in section 4 we can do further work as the summarization below

Game content editor

Currently the questions are putted into the database manually using DBMS. This is not practical for every teacher. So there is need for a tool for managing the creation of lecture and input questions, review game result later. The tool also supports the selection of questions for a quiz, create new lecture based on the old ones. The tool may provide statistic information to teacher and other system in schools or universities.

Communication protocol

In our implementation we have a dedicated component for managing data communication. This component exchange data in text format which is not a standard format for data communication anymore. In the future we can improve the communication using XML format. Currently J2ME MIDP client and server can communicate using SOAP protocol. There are two implementations available: the implementation from Sun called JSR172 and kSOAP [30]. We encourage customizing and using these protocols in the new version of the game for better data interchange. The change should not cause any modification on other layers because of modular design.

Add more game modes

We can extend the game by adding more game modes. Because of time limit, we just create one mode in the new version of the game but the architecture is open for adding more game modes. The other mode maybe “Group mode” where players are divided into groups and game is playing between groups.

Reference

1. Ole Kristian Mørch-Storstein; Terje Øfsdahl, *Master thesis Game Enhanced Lectures*. 2007

2. Wiki, http://en.wikipedia.org/wiki/Educational_game last access on January 26, 2008.

3. Scott W. Ambler, *Choose the Right Software Method for the Job*. 2006, http://www.agiledata.org/essays/differentStrategies.html.

4. V.R. Basili, The Experimental Paradigm in Software Engineering. in Experimental Software Engineering Issues: Critical Assessment and *Future Directions*. 1993, Dagstuhl Castle, Germany: Springer Verlag.

5. A.I. Wang, Using a Mobile, Agent-based Environment to Support *Cooperative Software Processes*. in IDI. 2001, NTNU: Trondheim. p. 407.

6. B.V. Koen, Toward a Definition of the Engineering Method. in ASEEIEEE Frontiers in Education Conference. . 1984. Philadelphia.

7. H. Bär, E. Tews, and G. Rössling, *Improving Feedback and Classroom* Interaction Using Mobile Phones. 2005.

8. Lecturelab: Uce Servers & Clients, *WIL/MA, Wireless Interactive* Lectures in Mannheim. http://www.lecturelab.de/. Access on 3rd June 2008.

9. A.I. Educlick. 2004 Limited, *http://www.avrio.co.uk/shop/educlick.htm*. Access on 3rd June 2008.

10. Aclass Technology, *http://www.aclasstechnology.com/*. Access on 3rd June 2008.

11. Government Backs "Buzz! For Schools", http://www.mcvuk.com/news/25249/Government-backs-Sonys-Buzz-for-schools Access on 3rd June 2008.

12. C.M. And C. Lengauer Pancake, High-performance Java: Introduction.

Communications of the ACM. 2001. 44(10 (2001)): p. 98-101.

13. Java Technology, *http://java.com/en/about/*. Access on 29 June 2008.

14. Wiki, .NET Framework
http://en.wikipedia.org/wiki/Microsoft_.NET#Microsoft_.NET. Access on 2 June 2008.

15. S. Helal, Pervasive Java, in IEEE Pervasive Computing Magazine. 2002.

16. Midp Java Phones, http://www.club-
java.com/TastePhone/J2ME/MIDP_mobile.jsp. Access on 2nd June 2008.

17. C. Neable, The .NET Compact Framework, in IEEE Pervasive Computing
Magazine. 2002.

18. Mysql Wiki, *http://en.wikipedia.org/wiki/MySQL*. Access on 2nd June 2008.

19. Lampware, *http://www.lampware.org*. Access on 2nd June 2008.

20. Postgre Sql Faq, http://www.postgresql.org/docs/faqs.FAQ.html. Access on 2nd
June 2006.

21. Interbase Public License V 1.0,
http://www.firebirdsql.org/index.php?op=doc&id=ipl. Accesson 2nd June 2008.

22. S.H. Jeong, QoS support for UDP/TCP based networks. Computer
communications. 2001. 24(1): p. 64.

23. K. Azadet, Gigabit Ethernet over unshielded twisted pair cables. 1999.

24. S.N.S. Haggman, GPRS performance estimation in GSM circuit switched
services and GPRS shared resource system. in Wireless Communications
and Networking Conference. . 1999. New Orleans, LA, USA.

25. A.F.S.M.F.M.H. Olofsson, EDGE: enhanced data rates for GSM and
TDMA/136 evolution. Personal Communications, IEEE. 1999. 6(3): p. 56-66.

26. A. Samukic, UMTS universal mobile telecommunications system:
development ofstandards for the third generation. IEEE transactions on

vehicular technology. 1998. 47(4): p. 1099.

27. B.P. Crow, IEEE 802.11 Wireless Local Area Networks. IEEE communications magazine. 1997. 35(9): p. 116.

28. J. Haartsen, Bluetooth-The universal radio interface for ad hoc, wireless connectivity. Ericsson review. 1998. 3(1): p. 110.

29. K.E.M. Martin Karlsson, Erik Hagersten, David A. Wood., Memory system behaviour of Java-based middleware. In the ninth International Symposium on High-Performance Computer Architecture. 2003.

30. Mobile SOA: Service Orientation on Lightweight Mobile Devices *http://www.firebirdsql.org/index.php?op=doc&id=ipl*. Accesson 2nd June 2008.

http://ieeexplore.ieee.org/iel5/4279552/4279553/04279753.pdf?tp=&isnumber=&arnumber=4279753, Access on 5th June 2008.

Appendix

Appendix A.1 The Main loop function in previous game

```
private void action() {


    master = connectionKeeper.getMaster();


    if ((master != null || connections.size() != 0)) {
      //if(true){


      // ****** NO CONNECTIONS ********
      String res1;
      if (serverState == LG.STATE_NO_CONNECTIONS) {
        serverState = LG.STATE_CONNECTED;
        ConsoleWriter.debug("Setting serverstate " + serverState, 3);
      }


      // ****** CONNECTED ********
      else if (serverState == LG.STATE_CONNECTED) {


        if (master != null) {
          if (master.inDataAvailiable()) {
            master.wakeme();
        }
    }

    for (int i = 0; i < connections.size(); i++) {
        StudentClient c = connections.get(i);
        if (c != null && c.inDataAvailiable()) {
            c.wakeme();
        }
    }
}

// ****** PUSH QUESTIONS ********
else if (serverState == LG.STATE_PUSH_QUESTION) {



    currentQuestion = gameMode.nextQuestion();
    if (currentQuestion == null) {
        serverState = LG.STATE_END_OF_ROUND;
        ConsoleWriter.debug("Setting serverstate " + serverState, 3);
        return;
    }
    totalNumberOfQuestions++;
    ConsoleWriter.debug("inne i STATE_PUSH_QUESTION", 3);
```

```
            // sjekker at vi har et spørsmål.

            // henter ut spørsmålet som skal sendes til klientene, og lager
            // samtidig spørsmålet som skal sendes til masteren. forskjellen ligger i at riktig
            // alternativ sendes med spm til master.
            if (gameMode.getQuestionState() ==
LG.QUESTIONSTATE_FINAL_IN_ROUND && currentRoundIsLastRound()) {

gameMode.setQuestionState(LG.QUESTIONSTATE_FINAL_IN_LECTURE);
            }

            String masterQst = gameMode.createMasterQuestion();

            if (master != null) master.sendData(masterQst);

            for (int i = 0; i < connections.size(); i++)
               if (connections.get(i).loginStatus()) connections.get(i).sendData("A");

            try {
               //her settes delayet som skal føre til at mobilene er mest mulig synchet.
               Thread.sleep(2000);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
            String clientQuestion = gameMode.createClientQuestion();

            for (int i = 0; i < connections.size(); i++)
                if (connections.get(i).getAllowedToAnswer())
connections.get(i).sendQuestion(clientQuestion);

            serverState = LG.STATE_WAITING_REPLY;
            ConsoleWriter.debug("Setting serverstate " + serverState, 3);

        }

        // ****** WAITING REPLY ********
        else if (serverState == LG.STATE_WAITING_REPLY) {
            //setter maks totalScore mulig å oppnå
            totalPossibleScore += gameMode.getMaxScore();

            try {
                Thread.sleep(currentQuestion.getTimeLimit() * 1000 + 1500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            for (int i = 0; i < connections.size(); i++) {
                StudentClient c = connections.get(i);
                if (c.inDataAvailiable()) {
                    // gir tillatelse til at connectionen kan svare; permit so that the connection
can response
                    c.wakeme();
                }
            }
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();  //To change body of catch statement use File | Settings |
File Templates.
            }
            res1 = gameMode.handleQuestionReply();
            if (master != null) master.sendData(res1);
            int gameModeQuestionState = gameMode.getQuestionState();

            if (gameModeQuestionState == LG.QUESTIONSTATE_NOT_FINAL)
serverState = LG.STATE_CONNECTED;
            else serverState = LG.STATE_END_OF_ROUND;

            ConsoleWriter.debug("Setting serverstate " + serverState, 3);
        }

        // *** END OF ROUND ***
```

```
else if (serverState == LG.STATE_END_OF_ROUND) {

    res1 = gameMode.handleEndOfRound();
    if (master != null) master.sendData(res1);
    nextRound();

}

// *** END OF LECTURE ***
else if (serverState == LG.STATE_END_OF_LECTURE) {
    String res = handleEndOfRound();
    if (master != null) master.sendData(res);
    serverState = LG.STATE_CONNECTED;
    makeClientsDie();
    ConsoleWriter.debug("Setting serverstate " + serverState, 3);

}

// *** LOAD NEW LECTURE ***
else if (serverState == LG.STATE_NEW_LECTURE) {
    resetStatistics();
    nextRound();
    serverState = LG.STATE_CONNECTED;
    ConsoleWriter.debug("Setting serverstate " + serverState, 3);
```

```
            }

            // ****** PING ********
            else if (serverState == LG.STATE_PING) {

                for (StudentClient c : connections) c.triggerPing();
                serverState = LG.STATE_CONNECTED;
                ConsoleWriter.debug("Setting serverstate " + serverState, 3);

            }

        } else if (connections.isEmpty() && master == null && serverState !=
LG.STATE_NO_CONNECTIONS) {
            ConsoleWriter.debug("Waiting for Connections", 3);
            serverState = LG.STATE_NO_CONNECTIONS;
            ConsoleWriter.debug("Setting serverstate " + serverState, 3);
        }
    }
```

Appendix A.2 Compiling game engine

We are going to present here how to use Eclipse SDK to compile the game engine, but any java compiler can be used to compile the code.

Tools preparation

- Download Eclipse SDK, an open source java compiler at http://www.eclipse.org/platform, the version used is 3.2

- JDK version 1.6

- Windows XP

Steps to compile GE

- Create a folder and name it "C:\LECTURE_GAME" or whatever you want. This will be used as your working space in Eclipse.

- Copy the source folder GameEngine that is delivered with this document (this folder contains GameEngine project and source code) into the folder you created above.

- Download Conector/j Driver mysql-connector-java-5.0.8.zip and unzip to the fold "\jre1.6.0\lib\ext". You can find it under the Java home directory.

- Now open Eclipse (eclipse.exe), at the first time, the screen will look like in Figure 33. You need to choose "Workbench" to continue.

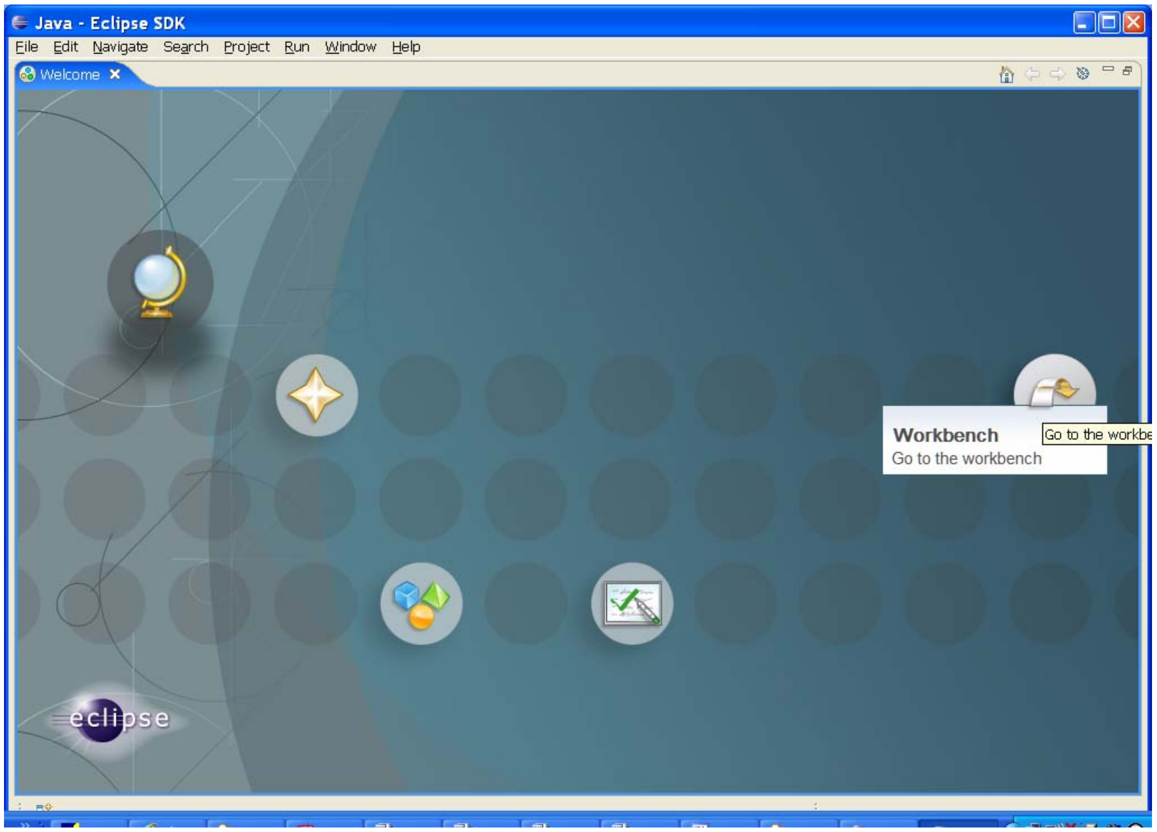
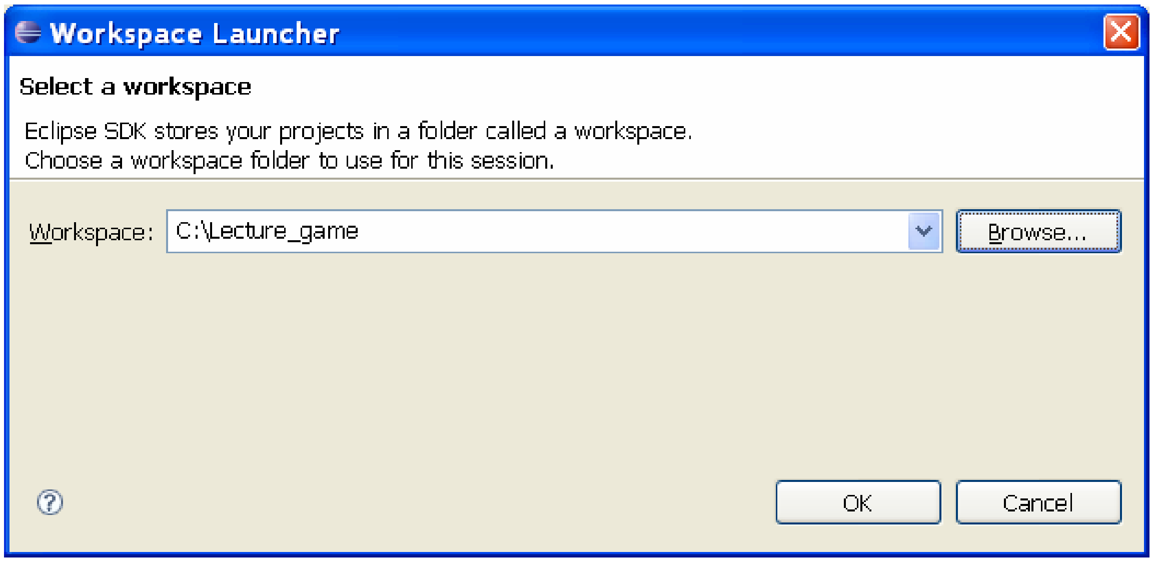


Figure 33. *Open workbench*

- Now you need to specify the "work space" in Eclipse, to do that choose File → Switch Work space and Browse to choose the "LECTURE_GAME" directory you have created.


Workspace Launcher

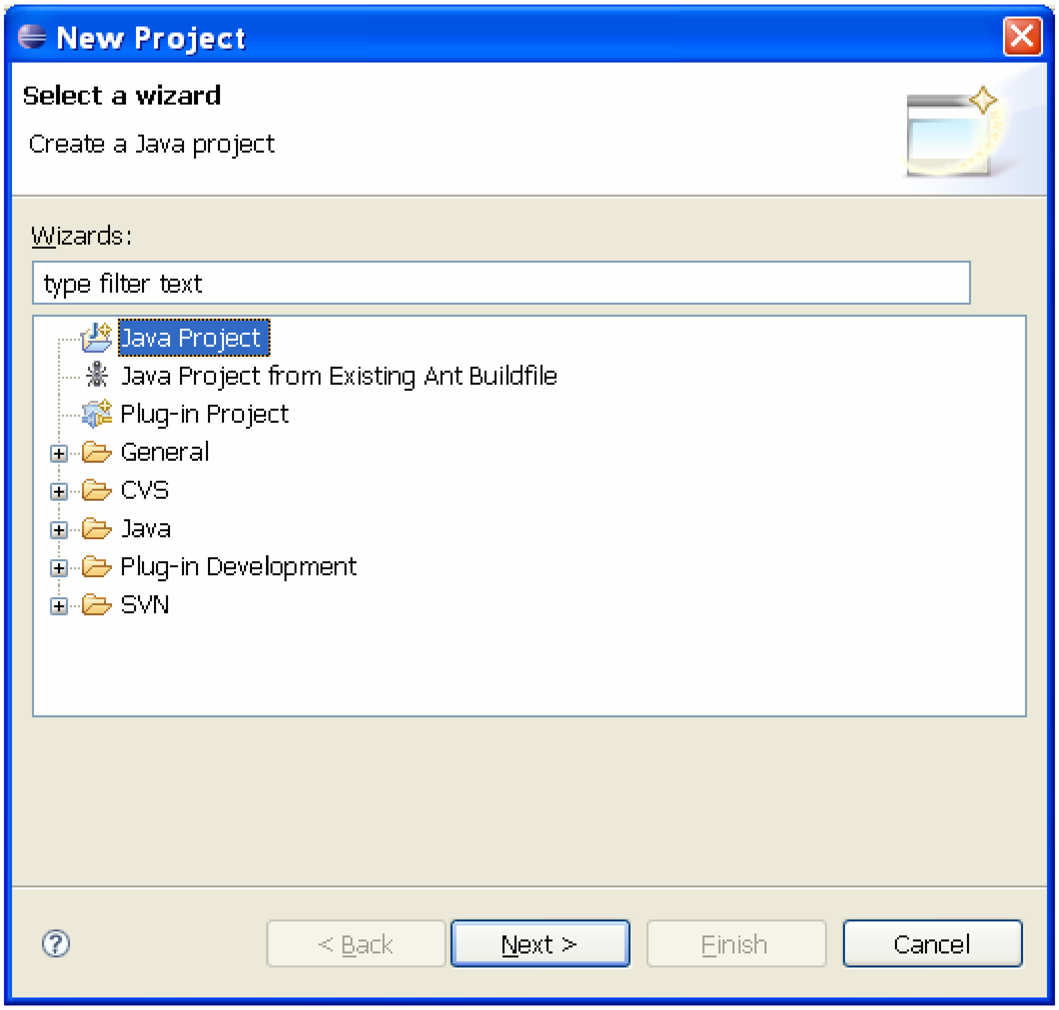
Select a workspace

Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Lecture_game

Figure 34. *Choose workspace*

- The next step is to create a Eclipse Project, choose file → New → Project. Choose java project and click Next, type “GameEngine” as the project name, click finish. A project will be opened in the workbench and Eclipse will automatically load all the source files in to that project.


New Project

Select a wizard

Create a Java project

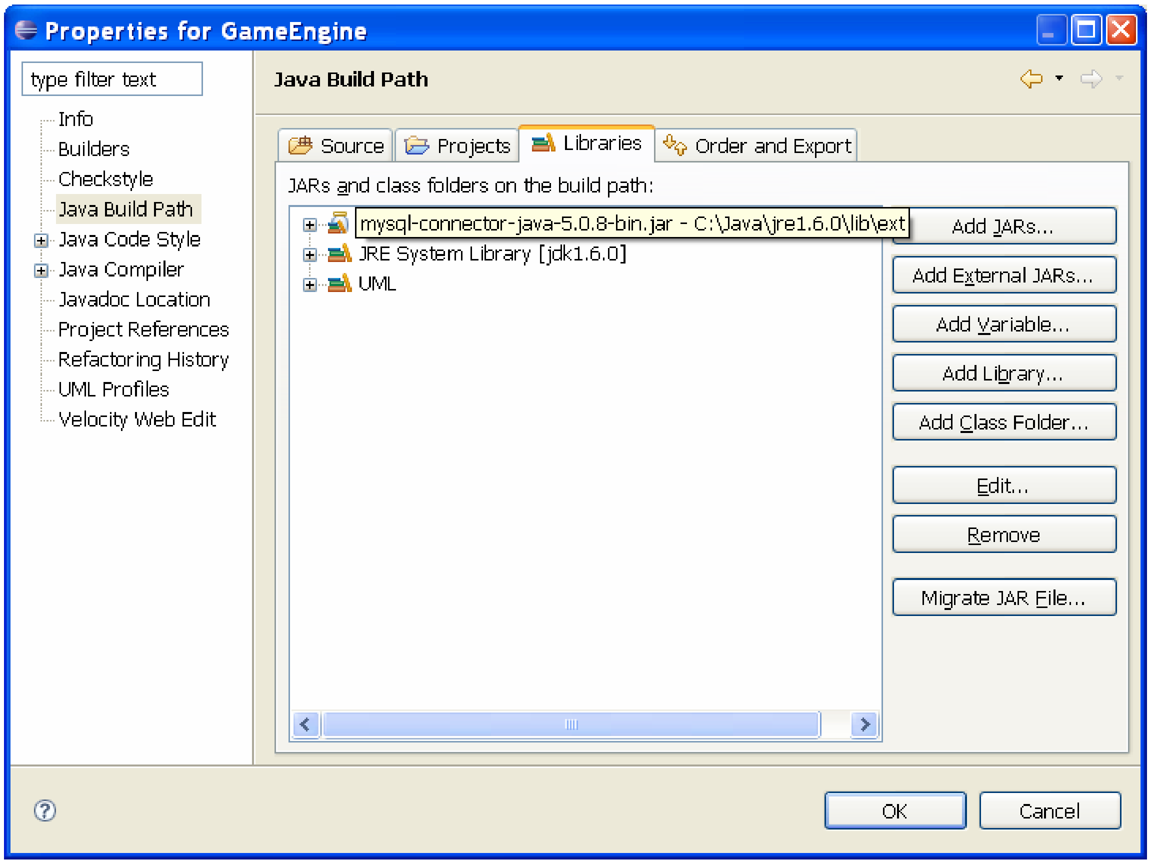
Wizards:

type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- CVS
- Java
- Plug-in Development
- SVN

Figure 35. *Choose type of project*

- The last step is to configure the project, right click on the project name and choose Properties. Go to Java Build Path and choose Libraries tab. Add an external Jar file from the mysql-connector you have just copied above. You may need to remove and add again if it is already exist. The configuration of libraries is the same as Figure 36


Properties for GameEngine

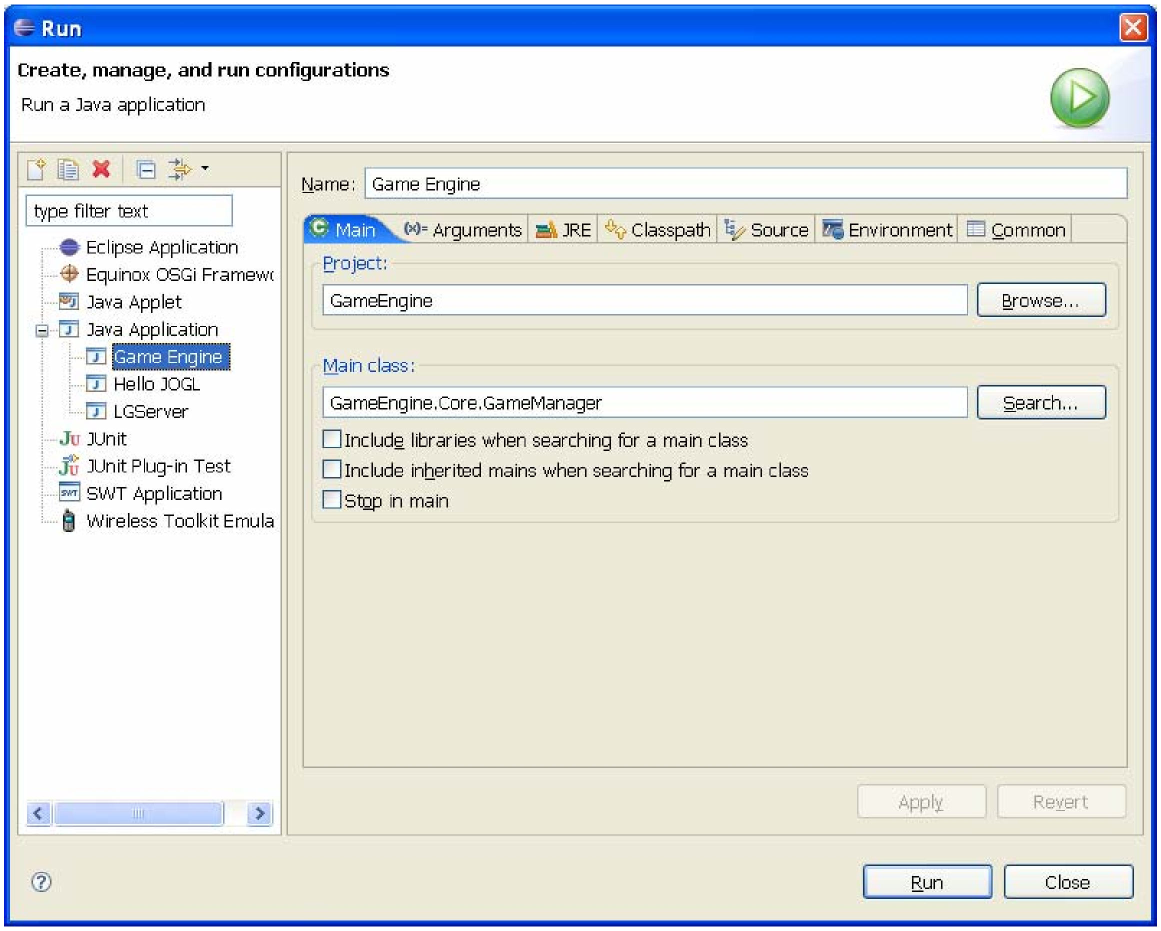
Java Build Path

JARs and class folders on the build path:

mysql-connector-java-5.0.8-bin.jar - C:\Java\jre1.6.0\lib\ext
JRE System Library [jdk1.6.0]
UML

Figure 36. Configure the library

- To build and run the application click button. At the first run, you may need to specify the main class, fill the main class text box with "GameEngine.Core.GameManager" and then click "run" button.



Run

Create, manage, and run configurations

Run a Java application

Name: Game Engine

Project: GameEngine

Main class: GameEngine.Core.GameManager

- [ ] Include libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Figure 37. Configure the run

If everything goes well then the game engine runs and print the line start listening for master client and mobile client on the debug window.

Appendix A.3 Compiling master client

To compile the master client, prepare the environment the same as for Game engine, for detail refer to the section Tool preparation above.

Follow these steps to compile master client

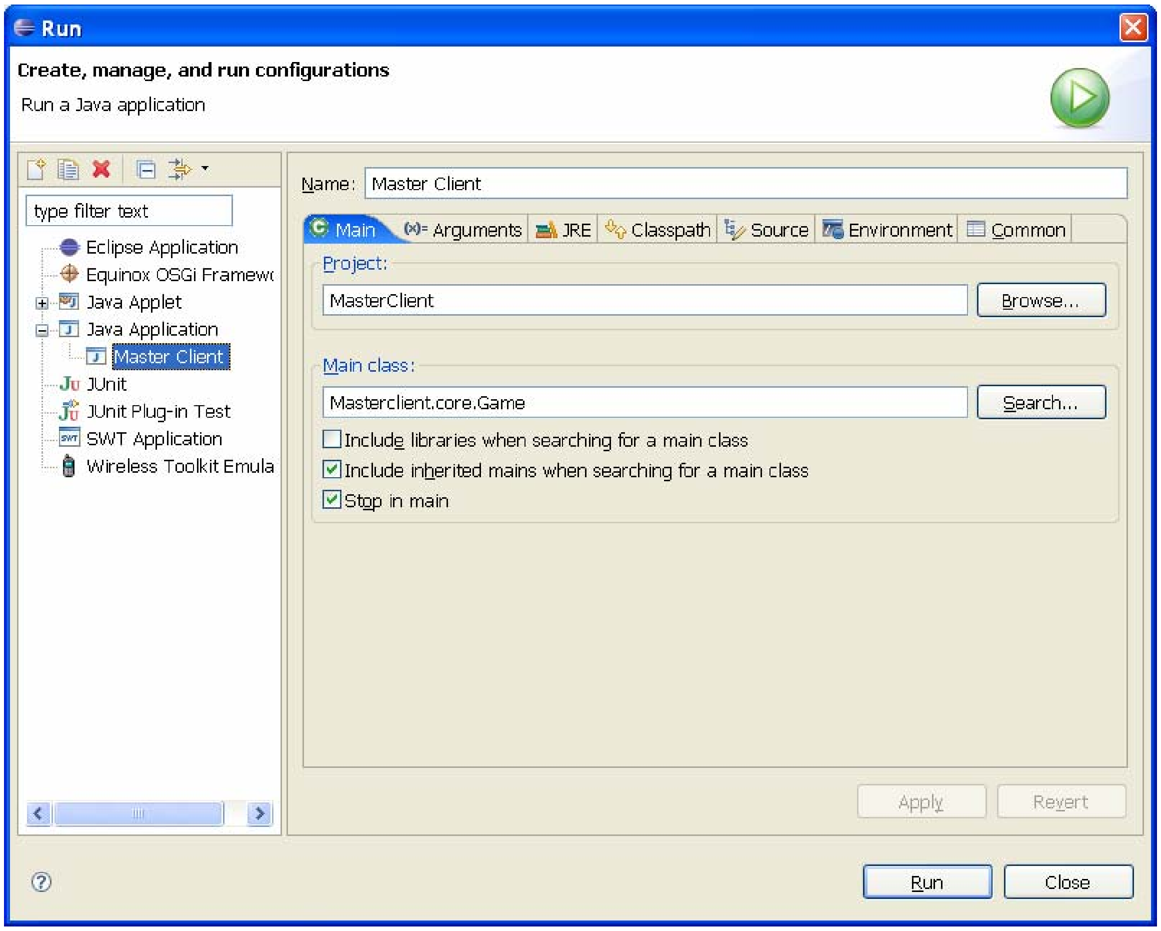
- Create a folder C:\Lecture_Game\Master_Space

- Copy the folder MasterClient from the CD delivered with this document (this folder contain the project and source files of master client) into C:\Lecture_Game\Master_Space

- Run other instance of Eclipse and change work space to C:\Lecture_Game\Master_Space.

- The next step is to create an Eclipse Project, choose file → New → Project. Choose java project and click Next, type "MasterClient" as the project name, click finish. A project will be opened in the workbench and Eclipse will automatically load all the source files in to that project.

- To build and run the application click button. At the first run, you may need to specify the main class, fill the main class text box with "Masterclient.core.Game" and then click "run" button.


Run

Create, manage, and run configurations

Run a Java application

Name: Master Client

Project: MasterClient

Main class: Masterclient.core.Game

☐ Include libraries when searching for a main class

☑ Include inherited mains when searching for a main class

☑ Stop in main

Figure 38. Configure the run of master client

If everything goes well then the master client will run and show the first screen.

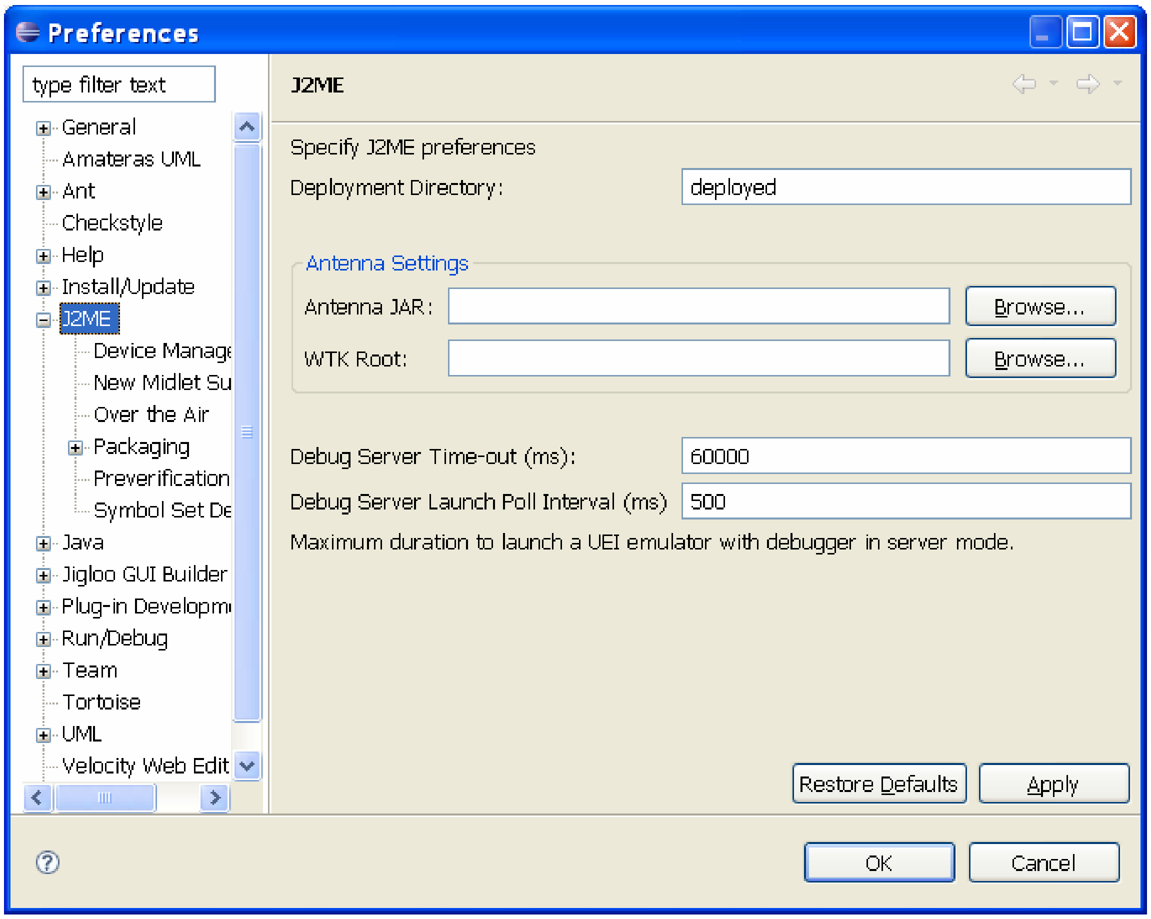
Appendix A.4 Compiling mobile client

Tools preparation

Beside the tools that specify above for compile game engine, download and install J2ME Wireless Toolkit 2.1_0 from http://java.sun.com/products/sjwtoolkit/download-2_1.html (There is a need to register, you can login with account longtien/longtien)

Install EclipseME, a plugin for building J2ME application in Eclipse. Go to http://www.eclipseme.org/docs/installEclipseME.html for instruction of how to install EclipseME.

Configure Eclipse and EclipseMe, go to Window → Preferences and make sure the "Debug Server Time-out" and "Debug Server Launch Poll Interval" are the same as Figure 39



Specify J2ME preferences

Deployment Directory: deployed

Antenna JAR:

WTK Root:

Debug Server Time-out (ms): 60000

Debug Server Launch Poll Interval (ms) 500

Maximum duration to launch a UEI emulator with debugger in server mode.

Figure 39. Configure J2ME

Step to compile mobile client

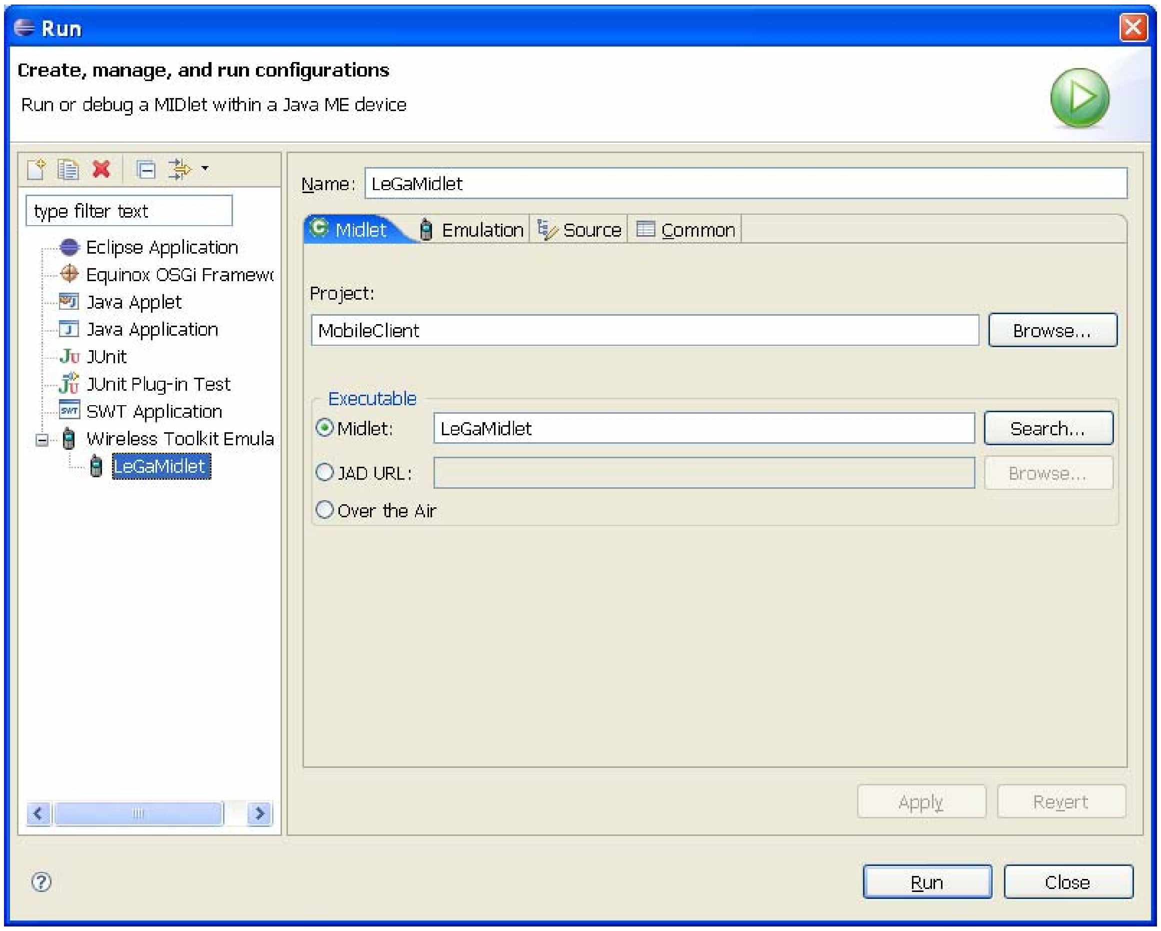
- Create a folder C:\Lecture_Game\Mobile_Space

- Copy the folder MobileClient from the CD delivered with this document (this folder contain the project and source files of mobile client) into C:\Lecture_Game\Mobile_Space

- Run other instance of Eclipse and change work space to C:\Lecture_Game\Mobile_Space.

- The next step is to create an Eclipse Project, choose file → New → Project. Choose java project and click Next, type "MobileClient" as the project name, click finish. A project will be opened in the workbench and Eclipse will automatically load all the source files in to that project.

- To build and run the application click button. At the first run, you may need to configure the run,create new Wireless Tooket Emulator as Figure 40.



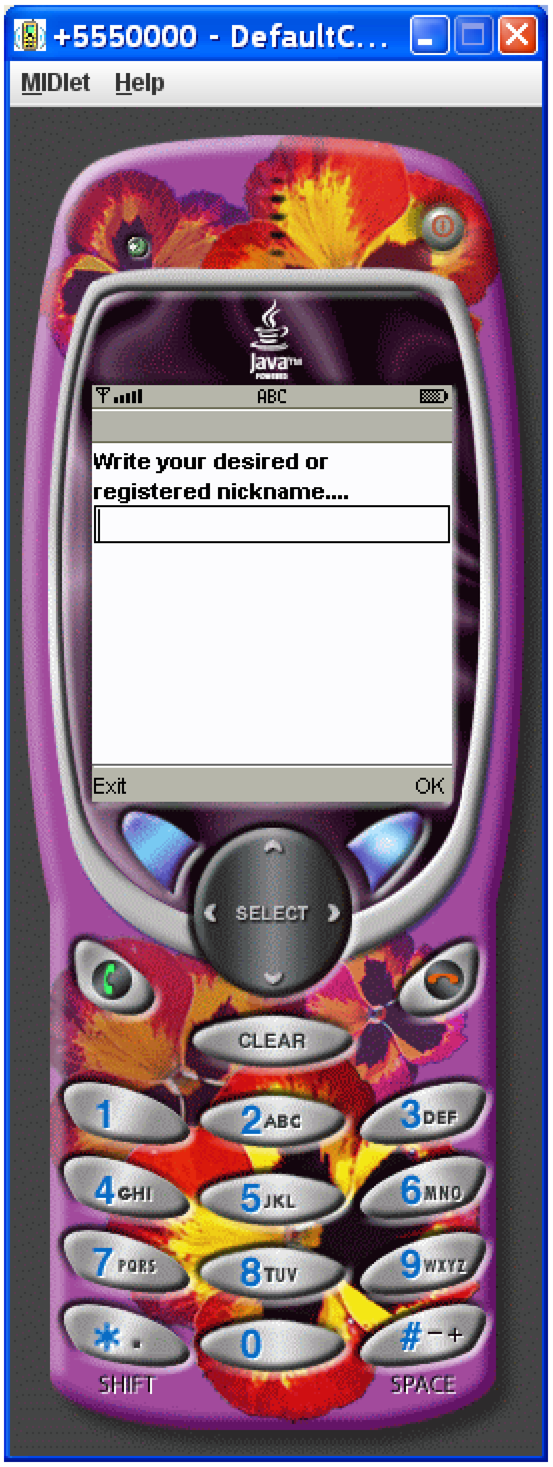
Name: LeGaMidlet

Project: MobileClient

(● Midlet: LeGaMidlet
○ JAD URL:
○ Over the Air

Figure 40. Configure the run for mobile client

If everything goes well then the mobile client will run and show the first login screen as in Figure 41.



Write your desired or registered nickname....

Exit OK

Figure 41. *Log in screen*

Appendix A.5 Setup database

To set up the database, following the steps below:

- Install MySQL database and create

- Create an account with username/password is lecturegames/ntnu

- Run the following SQL statements to create the database for lecture game

```
-- MySQL Administrator dump 1.4
--
-- ------------------------------------------------------
-- Server version       5.0.24-community-nt



/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
```

```
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;



--
-- Create schema test
--

CREATE DATABASE IF NOT EXISTS test;
USE test;

--
-- Definition of table `course`
--

DROP TABLE IF EXISTS `course`;
CREATE TABLE `course` (
  `courseID` varchar(7) NOT NULL,
  `name` varchar(45) default NULL,
  `description` varchar(45) default NULL,
  PRIMARY KEY  (`courseID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
```

```
-- Dumping data for table `course`

--


/*!40000 ALTER TABLE `course` DISABLE KEYS */;

INSERT INTO `course` (`courseID`,`name`,`description`) VALUES

 ('tdt4210',NULL,NULL);

/*!40000 ALTER TABLE `course` ENABLE KEYS */;



--

-- Definition of table `lecture`

--


DROP TABLE IF EXISTS `lecture`;

CREATE TABLE `lecture` (

  `lectureid` int(10) unsigned NOT NULL auto_increment,

  `course` varchar(7) NOT NULL,

  `description` varchar(45) default NULL,

  PRIMARY KEY  (`lectureid`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--

-- Dumping data for table `lecture`

--
```

```
/*!40000 ALTER TABLE `lecture` DISABLE KEYS */;
INSERT INTO `lecture` (`lectureid`,`course`,`description`) VALUES
 (1,'tdt4210',NULL);
/*!40000 ALTER TABLE `lecture` ENABLE KEYS */;



--
-- Definition of table `player`
--

DROP TABLE IF EXISTS `player`;
CREATE TABLE `player` (
  `username` varchar(20) NOT NULL,
  `password` varchar(20) NOT NULL,
  `registrationdate` datetime NOT NULL,
  `deprecated` int(10) unsigned NOT NULL,
  `isAdmin` tinyint(1) NOT NULL,
  PRIMARY KEY  (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `player`
--
```

```
/*!40000 ALTER TABLE `player` DISABLE KEYS */;
INSERT INTO `player`
(`username`,`password`,`registrationdate`,`deprecated`,`isAdmin`) VALUES
 ('a','a','0000-00-00 00:00:00',0,1),
 ('Terje','sesam','0000-00-00 00:00:00',0,1);
/*!40000 ALTER TABLE `player` ENABLE KEYS */;



--
-- Definition of table `question`
--

DROP TABLE IF EXISTS `question`;
CREATE TABLE `question` (
  `questionID` int(10) unsigned NOT NULL auto_increment,
  `question` text NOT NULL,
  `roundid` int(10) unsigned default NULL,
  `timelimit` int(10) unsigned default NULL,
  PRIMARY KEY  (`questionID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `question`
```

```
--


/*!40000 ALTER TABLE `question` DISABLE KEYS */;
INSERT INTO `question` (`questionID`,`question`,`roundid`,`timelimit`) VALUES
 (1,'What is your name?',1,30),
 (2,'What decide sw architecture?',1,30);
/*!40000 ALTER TABLE `question` ENABLE KEYS */;



--
-- Definition of table `question_alternative`
--

DROP TABLE IF EXISTS `question_alternative`;
CREATE TABLE `question_alternative` (
  `alternativeID` int(10) unsigned NOT NULL auto_increment,
  `questionid` varchar(45) NOT NULL,
  `alternative` varchar(80) NOT NULL,
  `isCorrectAlt` tinyint(1) NOT NULL,
  PRIMARY KEY  (`alternativeID`,`questionid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `question_alternative`
```

```
--

/*!40000 ALTER TABLE `question_alternative` DISABLE KEYS */;
INSERT INTO `question_alternative`
(`alternativeID`,`questionid`,`alternative`,`isCorrectAlt`) VALUES
 (1,'1','No name',0),
 (1,'2','Software quality',1),
 (1,'3','After requirement specification',1),
 (2,'1','You have a name',1),
 (2,'2','software function',0),
 (2,'3','The first phase',0),
 (3,'1','Pupy',0),
 (3,'2','sw company',0),
 (3,'3','In the Design phase',0),
 (4,'1','Catty',0),
 (4,'2','sw designer',0),
 (4,'3','When deployment',0);
/*!40000 ALTER TABLE `question_alternative` ENABLE KEYS */;


--
-- Definition of table `round`
--
```

```
DROP TABLE IF EXISTS `round`;

CREATE TABLE `round` (

  `roundid` int(10) unsigned NOT NULL auto_increment,

  `lectureID` varchar(45) NOT NULL,

  `gametype` int(10) unsigned NOT NULL COMMENT '10 for measure up, 11 for
elimination mode',

  `lecture` int(10) unsigned default NULL,

  PRIMARY KEY  (`roundid`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--

-- Dumping data for table `round`

--


/*!40000 ALTER TABLE `round` DISABLE KEYS */;

INSERT INTO `round` (`roundid`,`lectureID`,`gametype`,`lecture`) VALUES

 (1,'1',10,1);

/*!40000 ALTER TABLE `round` ENABLE KEYS */;






/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;

/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
```

```
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
*/;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
\
```