NTNU – Trondheim
Norwegian University of
Science and Technology

Tribal Knowledge War

A Location-based Pervasive Knowledge War Game

Adrian Christoffer Norås

Master of Science in Informatics
Submission date: June 2013
Supervisor: Alf Inge Wang, IDI

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

Abstract

In this thesis, we aim to examine the relationship between a game's pervasiveness and its entertainment value, and try to use this relationship to create a game that is both educational and fun to play. It is our hope that a game that is both fun and educational will motivate players actively learn and try to find new knowledge on their own in order to become better at the game.

To examine this relationship, we will develop a prototype of a game that aims to be both fun and educational. This game is called *Tribal Knowledge War*.

We will first conduct a prestudy, where we will find and present information about the Android platform and pervasive games. We will look at what possibilities the Android platform offers for making a pervasive game, and possible ways to implement these. During the prestudy, we will also find and discuss ways to create a good educational game by pursuing some core *aesthetics* in the game: providing a decent challenge, presenting a compelling fantasy, and appealing to the player's curiosity.

After the prestudy we will design and develop a prototype of a game, where the focus will be on competitiveness and winning quiz-duels against close players. A user experiment will then be conducted, where players are invited to play the game against other players, and afterwords fill out a survey where they will explain their feelings and opinions of the game.

In the end of this thesis, the result of the survey will be used to examine to what degree the game was successful as a fun and educational game. These results will show that the game scores high in general usability and is relatively easy to learn how to play. It will also show that the participants found it fun to play, and enjoyed the format of the game. However, while players enjoyed the game, the results suggest that the game does not do enough to stimulate curiosity and motivate further learning. We will use the results and the opinions expressed by players to suggest improvements to the game to fix shortcomings.

Sammendrag

I denne masteroppgaven ønsker vi å undersøke sammenhengen mellom et spills *pervasiveness* og dets underholdningsverdi, og vi skal prøve å bruke denne sammenhengen til å lage et spill som både er lærerikt og morsomt å spille. Målet er å lage et spill som er morsomt og som vil motivere spillerne til å aktivt ville lære og tilegne seg ny kunnskap på egenhånd for å bli bedre.

For å undersøke denne sammenhengen vil vi utvikle en prototyp av et spill som skal prøve å være både morsomt og lærerikt. Dette spillet vil hete *Tribal Knowledge War*.

Først skal vi gjennomføre forhåndsundersøkelser hvor vi vil finne og presentere informasjon om Android-plattformen og *pervasive* spill. Vi vil se på hvilke muligheter Android-plattformen tilbyr for å kunne lage et *pervasive* spill, og mulige måter å implementere disse på. Vi vil også finne og diskutere måter å lage et bra og lærerikt spill ved å sikte på å implementere viktige *estetiske* trekk i spillet: tilby en ordentlig utfordring, presentere spilleren med en engasjerende fantasi, og å appellere til spillerens nysgjerrighet.

Etter forhåndsundersøkelsene vil vi designe og utvikle en prototyp av et spill, hvor fokuset vil være på å konkurrere og å vinne quiz-dueller mot andre spillere som befinner seg i nærheten. Etter utviklingen vil det bli arrangert en brukertest hvor deltagere vil bli invitert til å spille spillet mot hverandre, og etterpå fylle ut et skjema hvor de kan gi tilbakemelding om hva de syntes om spillet.

I slutten av denne oppgaven vil resultatet av brukertesten bli brukt til å undersøke i hvilken grad spillet lyktes i å kombinere å være underholdende med å være lærerik. Resultatene vil vise at spillet fikk god tilbakemeldinger om underholdningsverdi og generell brukervennlighet, og at spillere likte spillets format og utførelse. Derimot vil det også vise at spillet ikke lyktes i stor grad å stimulere nysgjerrighet og oppmuntre til videre læring. Resultatene av testen, kombinert med direkte tilbakemeldinger fra testerne, vil bli brukt til å foreslå forbedringer for å fikse svakhetene til spillet.

Preface

This document is the master thesis of Adrian C. Norås, and concludes my master's degree in Informatics.

In this thesis, I created a prototype of a pervasive duel-based quiz-game for the Android platform, to create a game that is both entertaining to play and can be an effective learning tool.

I would like to thank Alf Inge Wang who was my supervisor for his help on this thesis, and I would also like to thank all the participants in the user experiment, who provided invaluable feedback on the prototype.

Special thanks also go to my family and friends who provided moral support during the entire project, and patiently listened to my frustrated complaining during difficult times.

Trondheim, June 11th, 2013

Adrian Christoffer Norås

Problem description

The project will focus on developing and testing a new type of educational game on mobile phones. The game is a light-weight RPG where players can challenge other players in the same physical location to knowledge-duels. The game can be expanded to utilize other properties of mobile phones, as well as elements from similar games. The game consists of one game server and clients where the game is played. The target platform is Android.

The first phase of the project will be to develop a prototype of the game, including a server and game clients. The second phase is to test the game on players and evaluate the user experience.

Table of Contents

Figure Index

Index of Tables

Part I
Introduction and research

Chapter 1
Project Background

1.1 Motivation

Video games and mobile phones are a big part of today's society. Smartphones have become a very capable gaming consoles, and many users actively use their phones to play games. With so many people carrying around a gaming consoles at almost all times, there are great possibilities to provide a large part of the population with useful applications, such as educational games. Educational games have much potential as a tool to make learning fun, and to motivate its users to seek out knowledge on their own.

Smartphones also create new and exciting possibilities for such educational games, as these devices make it possible for users to bring games with them wherever they go. Most modern devices have many different sensors that can find information about the position of the device and state of the world around it. Sensors such as accelerometer and GPS are easily accessible for applications on the device, and games can let this information influence gameplay in various ways.

Motivated by these new possibilities, we want to create a fun and educational games that utilize some of the sensors of an Android phone to create a pervasive game, that will hopefully make the experience more engaging and valuable.

1.2 Project Goal

The goal of this project is to develop a prototype for an educational and pervasive game, where players that are close to each other can challenge each other to quiz-duels. The prototype should try to incorporate elements and strategies from popular design theories in today's game industry in order to make the game both entertaining and educational. After development of the prototype, it should be evaluated in a user experiment, where participants get the opportunity to play the game against other players, and then give feedback on the overall quality of the game and its educational value. The results of this experiment can be used to see whether a game like this can be relevant to use for educational purposes, whether it would be an enjoyable way for for example students to learn about a subject, and to motivate further learning.

The name of this game will be *Tribal Knowledge War*.

1.3 Reader's Guide

This thesis consists of six parts.

Part I is an introduction to the thesis, where the motivation and goal of this project will be explained, as well as the research questions and method.

Part II is the prestudy where relevant information will be researched and presented, such as related works, information about the Android OS, and various possible solutions for the implementation of the prototype. It will also discuss important terms such as pervasive game, and present various publications concerning game design.

Part III will explain the developer's own contribution in this project. It will explain how the finished game works, what the purpose behind design decisions were, as well as present the implementation of the prototype.

Part IV contains the description and evaluation of the user experiment.

Part V is the conclusion of the thesis, where the answers to the thesis' research questions will be discussed. This part also contains a discussion of possible improvements to the game, and future work.

Part VI contains the appendix where the raw data from the user experiment can be found.

Chapter 2
Research

2.1 Research Questions

The research questions in this thesis all concern how one can make a game that is both enjoyable to play and educational.

2.1.1 Combine fun with educational

The goal of almost every game developer is to make a game that is fun, scary, sad or otherwise engaging in order to give players the desire to play the game for extended amounts of time. A game that is so uninteresting that every player stops playing it after a couple of minutes must be considered a failure. Knowing this, what is the best ways to create a game that players find engaging?

When trying to develop an educational game, the main goal is to teach the player something. What are good ways to make sure that players learn something by playing the game? A game can be educational without being entertaining, but it's hard to imagine players dedicating much time to an uninteresting game unless they are being forced to play it by a teacher or similar.

It's reasonable to assume that the best way to create an educational game, is to combine the two main goals, and try to make a game that is both fun *and* educational, so that players themselves are motivated to play and learn. But what is the best way to do this? What are the most important aspects that makes a game fun, and how can one combine that with making the game educational?

2.1.2 Pervasiveness and knowledge

In today's society, many people have a smartphone of some sort. This means that if a person is close to a group of people, there's a good chance that someone in that group has a device that allows them to communicate with them.

This creates new and exciting possibilities. One can create games that are educational, and that

makes it possible to compare knowledge against other people in the area, both friends and strangers. Would these possibilities make the game more enjoyable? Are people interested in playing against strangers, and would the prospect of winning over others motivate the player to become better at the game?

2.1.3 Research questions summarized

The questions presented in Chapter 2.1.1 and 2.1.2 are the basis for the research questions in this thesis, and these questions are summarized in Table 1.

Table 1: Research questions

RQ1	What makes a game an effective learning tool?
RQ2	What makes a game fun?
RQ3	Can one combine classic concepts from game design with an educational game to create a fun and valuable experience?
RQ4	Does competition make a game more fun?
RQ5	How can one use pervasiveness to increase motivation to learn?
RQ6	Is it more fun to only challenge players in the vicinity in the physical world?

2.2 Research Method

The main research method in this thesis will be the Engineering Method described my Zerlkowitz and Wallace [1]. This method consists of developing and testing a prototype, and based on the results of the tests, go back and improve and refine the prototype until no further improvements are needed.

Development of this prototype will consist of several steps. First, a literature study will be conducted, where the main goal will be to find important and relevant publications on theories of game design in order to find popular and effective strategies on how to make games fun and how to make games educational. The information found during this study will be used to answer many of the research questions, and the strategies will be used in the development of the prototype. Another important goal is to find information about Android OS, server software and various solutions for implementing the prototype.

After research and development, an experiment will be arranged where participants will play the game with other participants and then give feedback on the overall quality and educational value of the prototype. This feedback will be collected through a survey that all participants will receive and fill out after playing the game. The survey will consist of 20 statements, and the participants will give each statement a “score” between 0 and 4, where 0 means that they strongly disagree and 4

means that they strongly agree with the given statement. By analyzing this data, we will get an indication of the general usability of the game, and how much players enjoyed playing it.

In addition to the survey, the developer will also observe the participants while they're playing, and look for anything noteworthy that might be of use in the final evaluation of the prototype, such as behavior while playing or whether they struggle with the GUI.

This feedback will be discussed, and used to suggest changes and improvements to the prototype.

2.3 Development Tools

2.3.1 Eclipse

Both the server and the client was written in Java, and Eclipse was therefore a natural choice of IDE for this project. Eclipse is entirely free and open-source, and it is a very flexible IDE because of its support of plug-ins. Plug-ins can be installed to add practically any functionality to Eclipse, and there are several plug-ins that were used in this project that made development easier.

2.3.2 ADT Plugin for Eclipse

Google has developed a plug-in for Eclipse called ADT (Android development tools). This plug-in makes it easy to develop applications for the Android platform by giving easy access to many of the Android SDK tools, such as providing access to an mobile device through a ADB. This makes it easy to compile and launch an Android application on an external device, and also makes it possible to view logcat-output from the device. ADT also includes customized XML-editors suited to create and edit Android manifest-files and layout-files.

2.3.3 AndEngine

AndEngine is a game engine that is created to make it easy for developers to create 2D games on the Android platform. It is an open-source project started and maintained by Nicolas Gramlich.

The engine is simple to use, and handles the difficult details of creating a game, such as handling user input, creating textures, implementing physics and collision detection, etc.

Use of the engine alters the structure of the application compared to a standard Android application by using its own *SimpleGameActivity*-class instead of the regular Android *Activity.* Developers then create their own *Scenes* that are displayed on screen. A Scene is similar to a canvas, in that sprites and graphics are placed in the Scene at specific coordinates, and are then displayed on screen for the user.

Usually, a developer for the Android platform would have to worry about all the different screen sizes on different Android devices, but AndEngine handles this automatically by detecting the screen size and scaling the graphics appropriately. This should make it much easier to make the

game appear pretty on different devices.

2.3.4 Visio 2013

Microsoft Visio is a flowchart software that makes it easy to draw diagrams that are pretty and easy to understand, and also follows standard UML notation.

2.4 Research process

The process of developing the game will be divided into four main parts: research, design, implementation and evaluation.

During the *research*-part, time will be spent becoming familiar with possible ways to implement the server and client, as well as looking for similar games and concepts to use as inspiration.

The *design* phase will consist of making a general design of the system, such as deciding on functional requirements and how the client and server will interact.

In *implementation*, the client and server will be developed in parallel using an iterative development strategy. The focus will be on implementing limited amounts of functionality at a time, and continuously building on what has been previously implemented, using the general design of the system as a guide.

In the *evaluation* phase, a user experiment will be conducted, in order to find if the goal of making a game that is both fun and educational was reached. In this experiment, participants will be invited to play the game against each other, and afterwords fill out a survey about their feelings on the game. The survey will ask about general usability of the game, and specific questions regarding educational value and enjoyability. In addition to the survey, the developer will observe the players while they are playing the game, and try to look for anything noteworthy in their behaviour that might reveal additional information useful for evaluating the game. The result of this experiment and observation will be used to evaluate the current prototype and suggest changes and improvements.

2.4.1 Use of code snippets and open source software

During development, certain problems will probably be encountered with solutions that are not easily developed alone. If such problems are encountered, discussion boards and forums on the Internet will be used to find common solutions, such as open source software, or simply snippets of code suggested by experienced developers.

Part II
Prestudy

Chapter 3
What makes games fun

One of the goals of the project is to develop a game that is both educational and fun. This chapter will explain some research done on how to accomplish this. This report will return to these concepts later, where they are used to evaluate the different aspects of *Tribal Knowledge War*.

3.1 Mechanics, dynamics, aesthetics

Hunicke, LeBlanc, Zubeck [2] describes a way to examine what they refer to as a game's *Mechanics*, *Dynamics* and *Aesthetics*, and how to use these observations to examine the different aspects of a game, and how the different aspect interact. It suggests looking at a game's *Aesthetics of play* first, to see what experiences the game hopes to give, and then see how the game's dynamics and mechanics should support these aesthetics. Note that the word *aesthetics* is used in a somewhat figurative way in the article, and is meant to describe the desirable emotional response evoked in the player when they interact with the game system.

Table 2 lists the taxonomy of aesthetics suggested in the article.

Table 2: Taxonomy of aesthetics

Aesthetic	Description
Sensation	Game as sense-pleasure
Fantasy	Game as make-believe
Narrative	Game as drama
Challenge	Game as obstacle course
Fellowship	Game as social framework
Discovery	Game as uncharted territory
Expression	Game as self-discovery
Submission	Game as pastime

Sensation refers to a game being pleasurable to experience, for example through having beautiful

music or graphics.

The aesthetic *Fantasy* is about the game presenting the player with an immersive world or setting, one that stimulates the player's imagination.

Narrative refers to the game including an interesting story, where the player cares about the outcome and what happens to the characters.

Challenge is when the game presents the player with obstacles to overcome, where either practice or thorough evaluation of the situation is needed to progress.

Fellowship as an aesthetic is about making the game encourage social interaction, for example through overcoming obstacles as a team.

The aesthetic *Discovery* refers to presenting the player with a game where there is much to discover, for example by having a huge and mysterious world where the player can travel. It can be similar to *Fantasy,* but focuses more on presenting the player with the unknown.

Expression is about providing the player with a way to express themselves in the game, for example by having ways for the player to permanently change the game world through their decisions.

Submission is about a game being simple to play, and not demanding very much attention. A game that has this aesthetic is the kind of game a player plays just to relax and wind down.

Hunnicke, LeBlanc and Zubeck suggests that this taxonomy helps to describe games, and by looking at what aesthetic goals a game pursues, it is easier to examine how the game must function in order to create these experiences for the player.

3.2 What makes things fun to learn

Thomas W. Malone [3] discusses the main characteristics of a good educational computer game, and he organizes these characteristics into three main categories: *challenge, fantasy,* and *curiosity.*

Challenge is explained as “In order for a computer game to be challenging, it must provide a goal whose attainment is uncertain.” This means that a game must neither be too easy or too hard. If a player is certain that they will always reach the goal with minimum effort, the game will quickly become boring. It is equally bad if a player is convinced that they will never reach the goal no matter how hard they try. There has to be a balance where the player must put in effort to reach the goal, and where their effort will be rewarded.

Games that include *Fantasy* show or evoke images of physical objects or social situations not actually present. This could mean many different things, such as setting up a story in the game, where the player has to progress and overcome obstacles to find out what happens. A game that seeks to teach children how to use a map could involve a story about a child being lost in a forest and where they have to use a map to get home. Fantasy can also mean to hide game mechanics behind a setting that disguises their purpose, such as solving puzzles to defuse a bomb that is about to blow up a city.

Curiosity is the motivation to learn, independent of any goalseeking or fantasy-fulfillment. If a game appeals to the player's curiosity, the player would be motivated to play it because they want to

unravel its mysteries. A game where every action causes a simple and predictable reaction would lose its appeal after a while, especially compared to a game where every action can occasionally cause unforeseen consequences. These consequences should not be entirely unpredictable, but rather be the consequences of a system that is deeper and more complex than it originally appeared, so that the player always feels that there is more to discover.

3.3 Competition

Competition can motivate players to learn and perform better as a consequence of introducing a variable difficulty level to the game, but it can also motivate players to improve because they want to be better than everyone else thus introducing a goal for players [3]. Competition is considered to be an effective way to motivate players to learn [4], but one has to be careful because if the game becomes excessively competitive it can have negative effects [5], and make losers lose interest.

Evaluations of other prototypes of competitive quiz games has shown that players seem to enjoy the competitive nature of the game, and that it motivated them to learn [6].

Chapter 4
Similar games and concepts.

In this chapter we will introduce several games that are similar to *Tribal Knowledge War* in some way, and we will also explain what is meant by a pervasive game.

4.1 Pervasive Games

Pervasive games are not easily defined, but a working definition is given by Montola [7]:

> *A pervasive game is a game that has one or more salient features that expand the contractual magic circle of play spatially, temporally or socially.*

Spatial expansion means that the socially constructed location of the game is unclear or unlimited. In contrast to for example traditional board games that are played at the location of the board and players, pervasive games can be played anywhere.

Temporal expansion ties in with spatial expansion, and means that the socially constructed game session is interlaced and mixed with regular life. Just as they can be played anywhere, pervasive games can also potentially be played anytime. Games might lie dormant and then suddenly alert the player into playing them again.

Social expansion means that the boundary of playership is obfuscated. People who are not active players might make a difference to the outcome of the game, either becoming unwillingly involved, or by being made aware of the game being played and deciding to join.

A survey on attitude towards pervasive gaming [8] suggests that such games are an attractive concept to people in general, and that the majority felt that pervasive features would add value to a game. The study also found that simpler game concepts were more attractive than more advanced concepts like augmented reality.

Although player vs player mode was less popular than single player mode, the survey also suggested that people want pervasive games to facilitate competition and communication.

Pervasive games do not have to be video games, and an example of a pervasive game that is not a video game is *The Amazing Race,* a TV game show where several teams compete for a big money prize. The game consists of several "rounds," where in each round the teams travel from location to location, finding and decrypting clues in order to find where to go next, and the goal is to be the first team to arrive at the round's "pit stop." The last team to arrive is eliminated from the game, and the winning team is the one that arrives first at the final pit stop. The locations in the game can be in many different countries, and the clues can be very cryptic and involve the surrounding area, which makes the game fit the definition of a pervasive game.

A good example of a pervasive video game was inspired by *The Amazing Race*, and was developed as part of a master thesis in 2011. This game was called *The Amazing City Game [9]*, and was a game for Android OS. In the game, a person (or team) starts the game on their phone, and they have to go to certain locations in Trondheim to solve puzzles, or to answers riddles and questions. The game takes advantage of many of the sensors in an Android phone, and utilize GPS position and camera etc.

4.2 Knowledge Games

Knowledge games are a popular genre of games, where the goal of the game is to know more than your opponent about a certain topic, mostly general trivia spread across several categories such as geography or literature.

Most knowledge games take inspiration from classic board games such as *Trivial Pursuit*. In *Trivial Pursuit* players throw dice to to move around on the game board, and they have to answer a questions from the category corresponding to the color of the tile they land on. The goal of the game is to reach certain special tiles, where answering the question correctly is rewarded with a piece of cake that symbolizes that the player has "completed" that category. When a player has completed all categories, they win they game if they manage to move to the middle of the board. Figure 1 shows the game board in *Trivial Pursuit,* and Figure 2 shows an example of a card with questions the players have to answer.

Figure 1: Trivial Pursuit's board

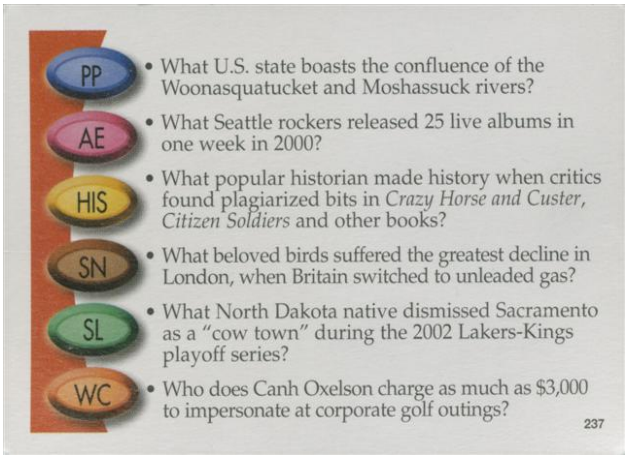

Figure 2: An example card from Trivial Pursuit

Buzz! is a series of quiz-games developed by the company Relentless Software for the Playstation 2 and Playstation 3 [10]. It supports up to four players at a time, and every player gets a custom Playstation-controller shaped similar to the devices participants on game shows on TV use. It has four distinctly colored buttons, and a huge red button at the top. The game has several different game modes, and players use the controller in different ways depending on the current mode. Often, players use the colored buttons to select a certain alternative in answer to a question, and other times the players have to be the first player to press the big red button on top when a correct answer appears.

Figure 3: A Buzz! controller

Figure 3 shows the standard *Buzz!* controller. Figure 4 below shows how an example of how a question is presented to the players. Each alternative corresponds to one of the colored buttons on the controller, and players answer by pressing one of the buttons.

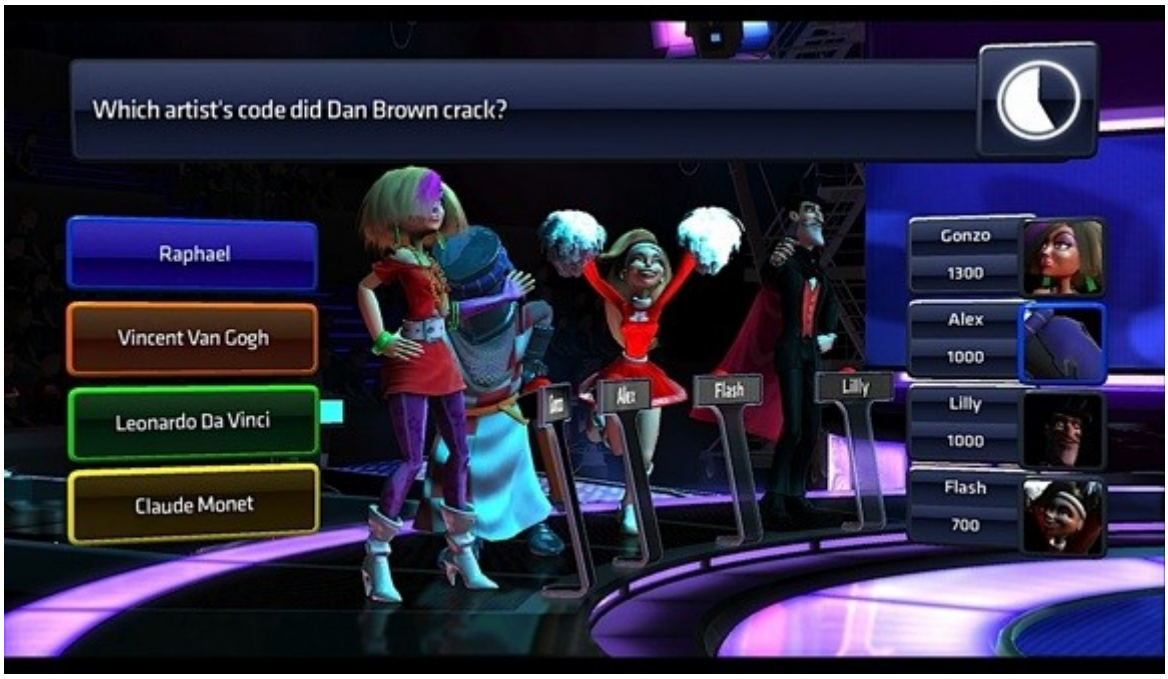

Figure 4: A question during a round of Buzz!

Lecture Quiz is a game concept developed by Alf Inge Wang at NTNU in 2006 [11]. It is a quiz-game designed to be used during lectures in schools and universities, where everyone present can partake in a competitive quiz battle. Players can use a laptop or a smartphone to participate and they only have to register a username on the server running the quiz, and they are ready. Questions are displayed to the entire class, and each player use their device to answer.

Figure 5: A question during a round of Lecture Quiz

Figure 5 shows an example of how a question is presented to an entire class during a lecture. Similar to how Buzz! functions, each player has the colored buttons on their personal device, and answer the question by pressing the appropriate color.

Quiz battle is a game application for Android where players can challenge friends to knowledge-battles. Each battle has a board that consists of a grid of squares, where one square is the start positions, and another is the goal position. Players navigate around by dragging a question-category from the side of the screen to the square they want to move to. They then have to answer a question from that category, and if they answer correctly, they move to that square. Players take turns answering questions, where each turn consists of five questions, and each correctly answered questions grants points. The battle ends when a player places a question on the end-position and answers the questions correctly, and the player with the most points wins the battle.

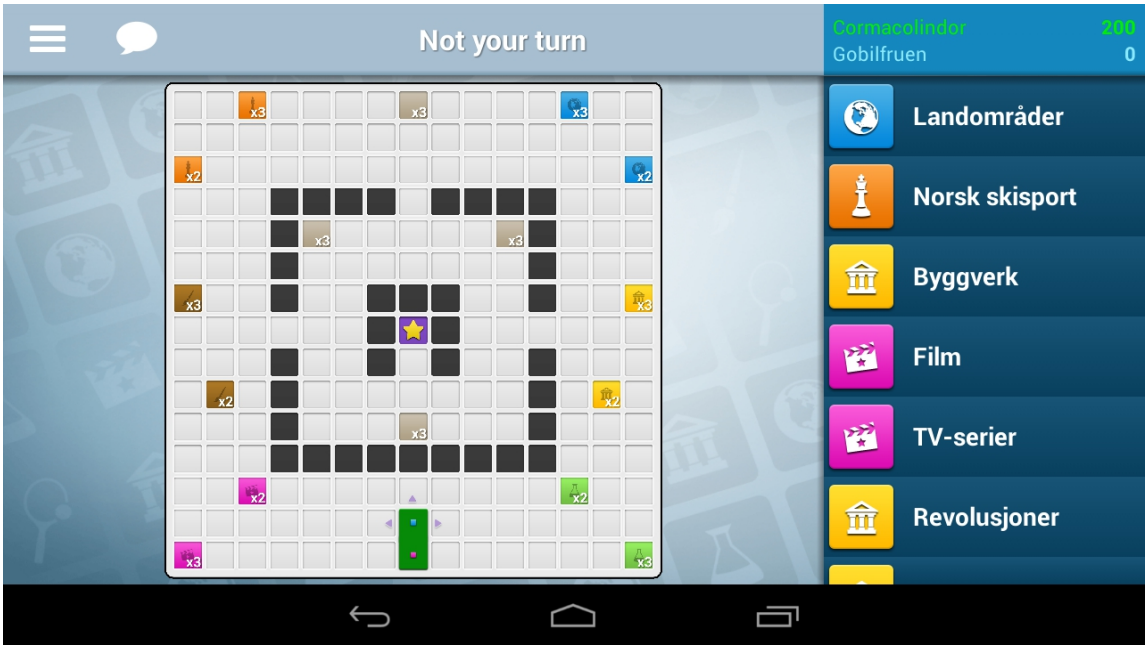

Figure 6: The game board in Quiz Battle

Figure 6 shows the game board in *Quiz Battle,* with the various available categories of questions on

the right side of the screen.

4.3 Duel-based games

Duel-based games are games where a large part of the gameplay is centered around one player challenging another to direct competition, and where the result is that one player wins and the other loses (or it ends in a draw). This is relevant because it's similar to how *Tribal Knowledge War* will work, and inspiration can be taken from these games to make *Tribal Knowledge War* entertaining.

Tetris [12] is a classic video game where a player is given a playfield where blocks are continuously falling. The player can control the blocks, and if they manage to stack blocks in such a way that they fill an entire row in the playfield, the row is removed and the player is awarded points. The goal of the game is to get an as high a score as possible, and it becomes more and more difficult because the blocks fall faster the more points the player has. A player loses if the stack of blocks become so tall that it reaches the top of the playefield.

The game was originally made for the Commodore 64 and IBM PC, but it was the Nintendo Game Boy-version of the game that was released in 1989 that really made the game popular. Since then, many different versions of the game has been made, and several versions have a multiplayer mode. In multiplayer, two (or more) players play at the same time, and whenever a player removes blocks from their own playfield, blocks are added to the opponent's playfield. This makes the duel a constant struggle to stay ahead of the other player, and the goal can be either to reach a certain amount of points, or simply be the last surviving player.

Figure 7 shows a screenshot from a modern version of *Tetris* that supports 4 simultaneous players in multiplayer. While this is not strictly a duel that fits the definition given earlier since more than two players are involved, it follows many of the philosophies and can still be used as inspiration for *Tribal Knowledge War.*

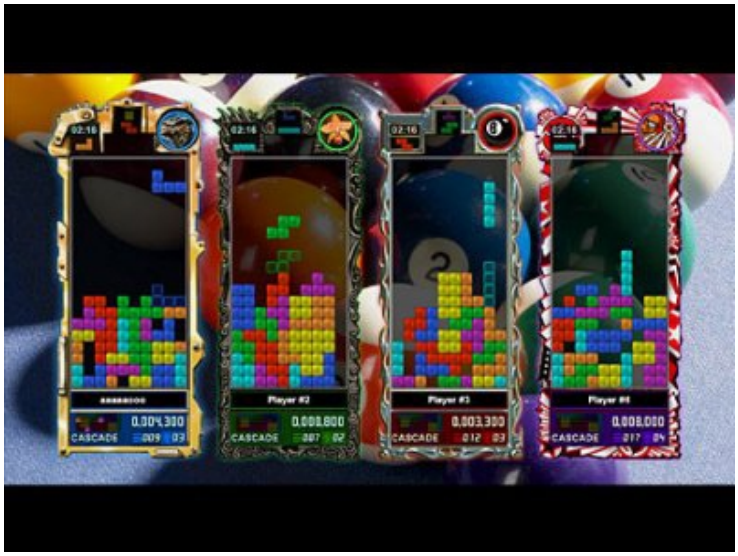

Figure 7: Tetris multiplayer

Super *Puzzle Fighter II Turbo* is a one- or two-player puzzle-game developed by Capcom [13]. In

the game, the two players are given their own playfield and control blocks of “gems” that drop into it. The object of the game is to remove gems as they fall down, but the only way to remove them is to place color-matching gems next to a “crash block” of the same color. This will remove all adjacent gems of the same color and can trigger huge chain reactions. Every time a player removes gems from their playfield, “garbage blocks” are dropped into the opponent's playfield. The garbage blocks can not be removed immediately and can make the game difficult for the opponent. As a consequence, each player tries to be faster than other player and ruin their progress by creating as many garbage blocks as fast as they can.

Figure 8: Battle in progress in Super Puzzle Fighter 2 Turbo

Figure 8 shows a battle in progress. The two blocks on the right-player side with the number 3 inside them are garbage blocks. The number inside indicates that they can not be destroyed before 3 seconds have passed.

Pokémon is a massively popular game series by Nintendo [14], with a strong multiplayer aspect. The player takes the role as a Pokemon-trainer, and the object of the game is to capture and train mystical creatures, and use these creatures to defeat other Pokemon-trainers in duels. This series mostly exists on Nintendo's handheld devices such as the DS and Gameboy, and in the older games, players that were close to each other could challenge each other to duels by standing close to each other and having their devices communicate through a cable or infra-red. This makes those game pervasive in much the same way as *Tribal Knowledge War*. The newer games, however, allow players to challenge each other through the Internet, which makes the location of the players irrelevant.

Figure 9: A Pokémon battle

Figure 9 shows a battle between two Pokémon trainers in the game *Pokemón Fire Red* for the Nintendo Game Boy Advance. *Fire Red* is one of the older games, and two players who wanted to play against each other had to use a cable to physically connect their GBA's toghether.

4.4 A special case: an earlier version of Tribal Knowledge War

Designing an developing a duel-based quiz-game has been an available assignment for a master thesis for many years at the Department of Computer and Information Science at NTNU. A specific earlier version called *KnowledgeWar* [15], made by Sveinung Kval Bakken in 2010 was based on a very similar to concept to *Tribal Knowledge War.*

KnowledgeWar was an application for face-to-face quiz battles, where players were able to challenge other people who was close to them, and both players would be presented with several multiple-choice questions, one at a time. Answering a question correctly would award points, and the player who received the most points would be the winner of that battle.

This game was *pervasive* because the physical location of the player influenced who they could challenge. That version did not however try to bring much gameplay beside the actual quizzes into the game, focusing instead on just implementing a functional quiz-application.

In this project, inspiration was taken from the previous version, but trying to add more game-like elements has been a priority.

Chapter 5
Android platform

Android is an open-source, Linux-based operating system software designed for smartphones and tablets. Android is a very flexible operating system for the user, and a lot of the flexibility comes from the option to install “apps” on the device. “Apps” are applications that extend the functionality of the device, and can be for example an application that lets the user quickly write down notes, or it can be a game meant for entertainment. Apps have access to almost all the features of the phone that the core applications have, so users are free to install custom camera-apps, and phone-apps. Even the onscreen keyboard used for user input can be replaced by installing an app.

Android applications are most often written in a customized version of the Java programming language. It runs on a custom virtual machine called Dalvik Java Virtual Machine (DJVM) instead of the regular Java Virtual Machine (JVM).

Android is the most popular operating system in the mobile market, with a market share of almost 75% in the first quarter of 2013 [16] .

Chapter 6
Possible technical solutions

On modern mobile devices, applications can often be divided into two groups: *web applications* and *native applications*. The line between these groups are somewhat blurred, but they can be described like this: a *web application* is an application that is accessed by users through a network, usually the Internet, and the client is often a standard web browser. A *native application* is a locally installed application that is designed to run in the computer OS, in this context Android OS.

These two kinds of applications have different advantages and disadvantages, and choosing which one to make is therefore an important decision to make in the development of an application.

Studies made on the opinions of various developers show that the development cost and time-to-market are lower for the development of web apps, but native apps most often offer a better experience for users [17].

6.1 Web applications

A web application was defined earlier as an application that is accessed through a network, often using a browser. One big advantage of this, is that the application can be accessed by just about any modern phone, or even a desktop computer. This would make the application almost inherently multiplatform, and would remove the need to write separate applications for for example Android and iOS. The application would as a result be both easier and quicker to develop.

Another big advantage is that such an application would be very easy to distribute and update, because there is no need to distribute a file that must be installed by the user. The user would simply need an URL and they would immediately be able to use the application. And since updating the application would be entirely on the server side, it could be done at any time, and without requiring the user to download any data.

Web applications do however have some drawbacks. While they are platform independent, they are not entirely browser independent. Some browser have small things that make them different from others, and this could potentially make it difficult to make the application appear as desired for every user.

Limited access to phone features is also a drawback. For a long time, web apps could not use any of

the phone's features such as GPS or gyroscope, but this is quickly changing. Today, most functions are accessible by a web application, but not all.

One drawback is the lack of possibility for push-notifications. A web application must be open to be active. If the user closes the web browser, they also close the application. If the application needs to tell the user that something has happened and the user should take action, it has to wait for the user to open the application and check.

Table 3 shows a summary of the main advantages and disadvantages of developing a web application.

Table 3: Advantages and disadvantages of web applications

Web application	
Advantages	**Disadvantages**
Automatically multiplatform	Limited access to phone features
Easier to distribute and update	Limited possibility for push-notifications
	Not browser independent

When developing complex applications, it is often a good idea to use a framework. A framework is an abstraction that provides the user with certain generic functionality, that can be changed or extended by user written code. A developer that uses a framework to develop software can usually not have to worry about low-level details of the implementation, but can concentrate fully on making the system meet their requirements. In the case of game development, it would be desirable to use a framework that makes the handling of graphics easy, and also that automatically handles the game-loop and similar generic game concepts. There are several such frameworks available for the development of web applications.

PaperJS is one such framework. It is an vector graphics scripting framework, and it runs on the HTML5 Canvas [18]. While not strictly meant for game development, it would be a powerful tool for creating nice graphics that would be accessible for every device with a modern browser.

Another possible framework is LimeJS. It is a HTML5 framework, and is good for building fast and native-experience games for basically any modern device [19]. It is open-source, has a decently sized community, and there are several games available on the Apple App Store that are developed using this framework.

A final alternative for framework is CraftyJS. It is an open-source, Javascript-based HTML5 game engine, designed to make simple games with 2D graphics [20].

6.2 Native applications

A native application was defined earlier as a locally installed application that is designed to run on the computer OS, for example Android OS in the case of mobile applications. One immediate advantage of this is that the application has full access to all phone features, like GPS, gyroscope and the ability to run background threads that can initiate actions without user input. This means

that the application can tell the user when some action is required, even if the user does not have the application open.

In contrast to web applications, native applications are automatically single platform, and the developer must make entirely separate applications for every OS they wish to support. This adds to both development cost and time.

Table 4 summarizes the advantages and disadvantages of native applications.

Table 4: Advantages and disadvantages of native applications

Native application	
Advantages	**Disadvantages**
Full access to phone features	Not multiplatform
Push-notifications	Must be installed on each device
Developer experience	Have to distribute updates to every user

As with development of web applications, there are many frameworks available for developing native game-applications.

One such framework is Unity 3D. It consists of a powerful rendering engine fully integrated with a complete set of intuitive tools to create 3D games [21]. It is a commercial product of very high quality, and developers have to pay for a license to use it.

Another framework is AndEngine [22]. It is an open-source game engine designed to make it easy for developers to create 2D games on the Android platform. It is actively developed, has a decently sized community, and offers lot of flexibility in development.

Chapter 7
Possible implementations

As a multiplayer game, one of the basic requirements of the game is that players have to be able to interact with each other, either through chatting or dueling.

There are some very different ways to implement this interaction between players, and it depends on the architecture of the system. With a client-server architecture, it would be natural that players communicate through the server, but with a peer-to-peer architecture, players could communicate directly with each other.

7.1 Peer-to-peer

A peer-to-peer architecture immediately introduces the problem of how players should find each other. Using GPS to find each user's location and then have other users find them based on that position is very difficult using this architecture. In fact, during this pre-study, no way was found to accomplish this. A more technically feasible solution would be to use Bluetooth. Bluetooth is a wireless technology standard for exchanging data over short distances, and practically all modern mobile devices supports it.

Bluetooth usually has range of around 100 meters, and devices with Bluetooth that are within range can find and connect to each other without having to use any external server. It is however, very battery intensive, and without a server, it would not make it possible to make a web application.

7.2 Server

A client-server architecture would require a server for the clients to communicate with, and there are several possible ways to set up this server.

One possibility would be to use Google App Engine [23]. The App Engine is a service offered by Google that lets any developer run web applications on Google's infrastructure. The App Engine is free, and because it uses Google's infrastructure, extremely reliable. It would require no maintenance and it would automatically scale according to its needs. It does however have the big

disadvantage that any sort of server-based permanent storage costs money.

Another possibility would be to set up a web server to run on a local machine for use during development, or rent a server from one of the many websites that offer such services.

One alternative for setting up a private web server is to use server software like Apache Tomcat [24] or Jetty [25]. Apache Tomcat is a open source web server and servlet container, and it provides a HTTP web server environment for Java code to run. One could then use Java Servlets with user-written code to handle requests from clients. These requests could be regular HTTP requests. Jetty is basically identical to Apache Tomcat in function and would therefore be a very viable alternative.

There are alternatives to using Java servlets, and one of these alternatives is Node.js [26]. Node.js is a platform made for easily building fast and scalable network applications. Applications developed for Node.js are written on the server side in Javascript, and can run on an Apache web server because it contains a built-in HTTP server library.

Renting a server from a service on the web would be extremely similar to using a local server, except that it would possibly easier to access for outside users.

7.3 GPS and Location

Continuously keeping track of a user's location is a tricky problem, without a single solution that works best for every case [27]. Android offers two ways to find the location of a device, GPS and Android's Network Location Provider.

GPS is the most accurate, but it does not work well indoors, uses a lot of battery, and it does not return the location of the device as quickly as most users want.

Android's Network Location Provider uses cell tower and Wi-Fi signals to determine the location. This works well both indoors and outdoors, responds faster than GPS, and uses less battery power. However, it is less accurate than GPS and requires that the device is in range of a cell tower or Wi-Fi signals.

7.4 Google Cloud Messaging

Google Cloud Messaging (GCM) is a service offered by Google that makes it possible for developers to send data from their server to a user's Android-powered device [28]. Messages sent this way can only contain a limited amount of data, and they are meant as a way to alert the client that there is new data to be fetched from the server. The maximum payload is 4kb of data, so simple messages like strings of text meant for instant messaging services can be sent directly.

The GCM service handles all aspects of queuing of messages and delivery to the target device, so it's easy to implement into any application. The developer must simply register an account with Google, and create a project in the Google API Console. The service then generates an API Key that the developer must use later. Then, the GCM helper libraries must be installed on the server, and whenever the server needs to send a message, it simply uses these libraries along with the API Key to send messages.

In order for the GCM service to be able to send a message to the target device, the device needs to find its GCM ID, and register this ID on the server. The ID is found by using standard Android libraries in the application on the target device.

Chapter 8
Summary of prestudy

In the prestudy, we have found some important and useful articles about how to make a game fun [3], and how to analyze game elements to examine if the game pursues the correct *aesthetics* [2]. These articles will be used later to design the gameplay of *Tribal Knowledge War*, and also to discuss the final design.

We have also found that there are several important decisions to make, that will influence the architecture of the system, and one of the most important decisions is whether the game should be a web application or a native application. Both have advantages and disadvantages, but both will be well suited for this project. The decision will have to come down to preference, and availability and ease-of-use of frameworks for the chosen solution.

The final decision is to develop the game as a native Android application. This is done because it fits the best with the developer's previous experience, and because this means that the very powerful and flexible framework *AndEngine* can be used. This will make the development easier, and means that more time can be dedicated to designing the game instead of struggling to learn new technology.

We found some possible ways to implement the server-side of the application, and the final decision on this will be presented and discussed in Chapter 12.1.

Part III

Own Contribution

Chapter 9
Description of the final game

In this chapter, the final design and gameplay of the game will be described. This is done to easier be able to explain the choices and changes made during development to the game's design and function.

9.1 Core gameplay

The main gameplay of *Tribal Knowledge War* consists of playing duels against other players where both players have to answer multiple-choice questions, with a time limit on each question. Each duel can have minimum two questions, but maximum five questions. For every correctly answered question, the player receives an amount of points determined by the time left when they answered; more time left means more points received. When both players have answered all questions, the player with the highest amount of points wins the duel.

However, there is also a somewhat strategic game surrounding the duels.

9.2 Players and their “tribe”

In *Tribal Knowledge War* each player plays as the leader of their own tribe of desert nomads. The different tribes are at war with each other, and they fight these wars by having duels where the winner is the tribe with the most trivia-knowledge. In addition to winning simply by having the most knowledge, they also push the odds in their favour by using several silly weapons during duels to make it harder for their opponent to play.

The game has a main resource, *gold*, which is used to buy weapons and buildings. Gold is earned by winning duels, and there are several ways to increase the amount of gold earned.

Each tribe has their own *area,* which in-game is a 5x5 grid where the players can place buildings or *wagons*. These wagons can be placed at each of the four corners of the player's area, and they grant different bonuses during duels, such as more gold awarded for winning, or increased damage. Damage is important because buildings have a limited amount of health, and can be destroyed if the

player loses a duel. Buildings that are placed can also be upgraded by spending gold. Upgraded buildings give stronger bonuses.

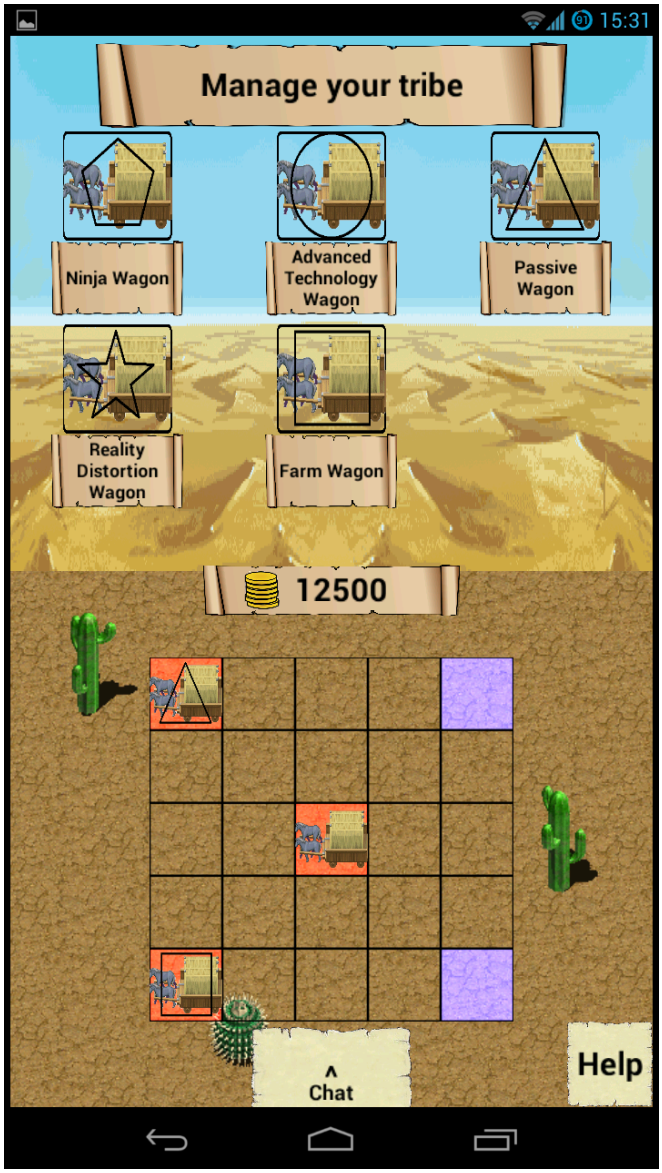

Figure 10: How a player manages their tribe

Figure 10 shows the screen in which a player manages their tribe. The placement of buildings within the player's area matters because when challenging another player to a duel, the attacker decides where to attack the other tribe. The two players' areas will overlap and this overlapping area is the “contested area” in the duel.

9.3 Challenges and the contested area

Figure 11: How to challenge someone

Figure 11 shows the screen where a player challenges another to a duel. The red area in the center of the bottom half is the opponent, and the blue area is the player who is about to send the challenge. The blue player can place the center of their area on any of the blank squares surrounding the red area, and this results in an overlapping are varying in size from 4 squares to 10 squares. Any buildings in the contested area grant bonuses to the duel, and each player's bonuses can be seen in the top left.

Buildings in the contested area can be destroyed if the player owning them loses the duel. A building has a set amount of *health* and if it receives *damage* higher than its health it is destroyed. The damage a building receives is determined by two factors: the amount of questions in the duel and the total damage-bonus from the buildings involved in the duel. Each question means one damage, so a duel with five questions will result in five damage to the loser's buildings in addition to the damage from the winner's buildings.

Gold is earned by winning duels, and the amount of gold is decided by several factors. First, each

square of the contested area is worth 100 gold. So the base gold profit for a duel where the contested area is 4 squares is 400 gold, while the base gold profit of 10 squares is 1000 gold. This base gold profit is modified by the amount of questions in the duel. Every question increases the base gold profit by 10%, so a duel with 5 questions and 10 squares in the contested area will have a gold profit of 1500 gold. Table 5 shows the entire relationship between number of questions, number of squares, and gold profit.

Table 5: Gold profit for all combinations of questions and squares

Questions / squares	4 squares	6 squares	8 squares	10 squares
2 questions	480 gold	720 gold	960 gold	1200 gold
3 questions	520 gold	780 gold	1040 gold	1300 gold
4 questions	560 gold	840 gold	1120 gold	1400 gold
5 questions	600 gold	900 gold	1200 gold	1500 gold

9.4 Duels and weapons

Duels in *Tribal Knowledge War* start at the same time, and they consist of the same questions for both players. Every question is a multiple-choice question, and players have a limited amount of time to answer. After answering a question correctly, a player gets the opportunity to use a *weapon* against their opponent. A *weapon* costs gold, and it makes the next question harder for the opponent in one way or another. It can reduce the amount of time available to answer, or display the text of the question mirrored. A player gets access to weapons through owning specific buildings. Some buildings give access to weapons, and upgraded buildings give access to more powerful weapons.

Although the duels start at the same time for players, the duels themselves are asynchronous, in that players can finish at different times. Weapon-use are also asynchronous, and every time a player uses a weapon, it is placed in a queue in the server. When it's time for the opponent to answer their next question, the game contacts the server and gets the negative effect that's at the front of the queue. Figure 12 shows this process.

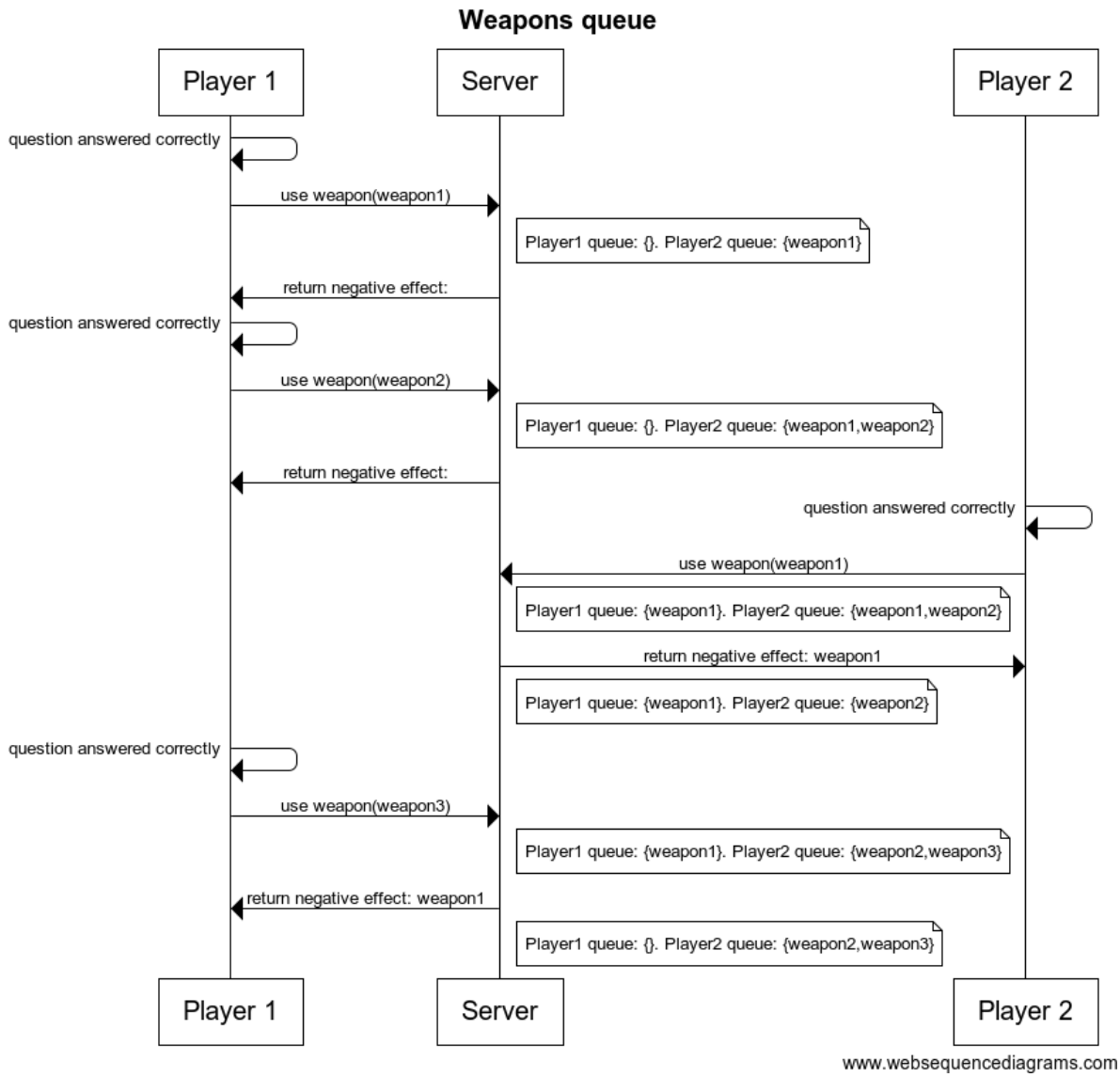

Figure 12 Sequence diagram displaying handling of weapons

Answering questions quickly is doubly rewarded, because the player gets to place lots of weapons in the queue for the opponent while at the same time keeping weapons used against them to a minimum, and in addition, quick questions grant the most points.

9.5 Game flow

This chapter describes the flow of the game, how the game is played, and what the user sees during gameplay.

9.5.1 Game states

Figure 13 shows a state diagram that shows the different states of the game, and what states the

game can transition to from a given state.

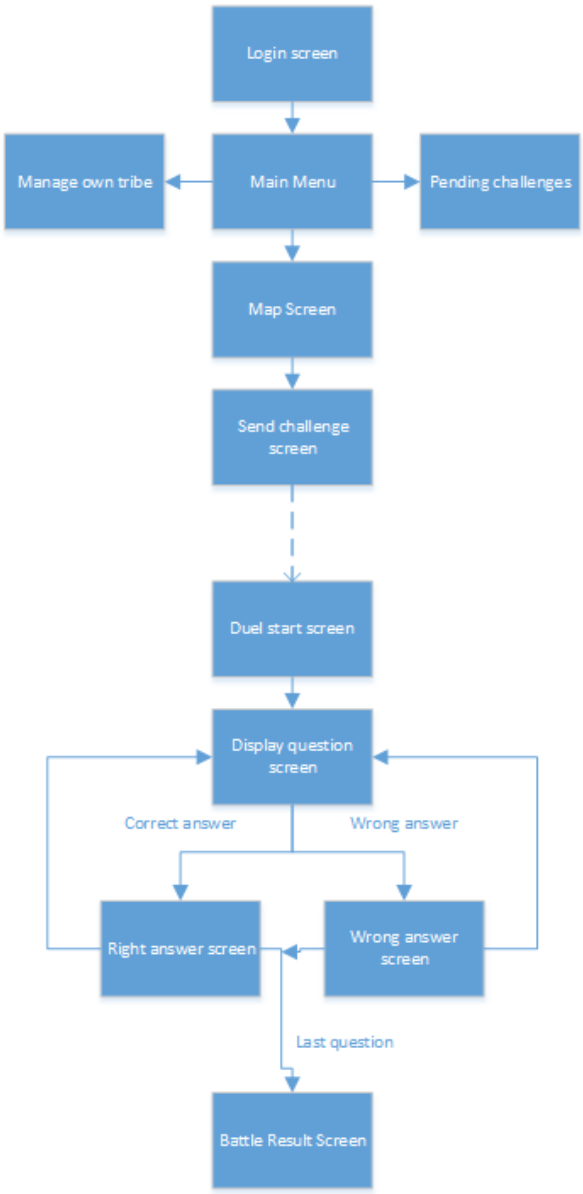

Figure 13 State diagram of Tribal Knowledge War

The *Login Screen* is the screen where the player types in username and password to log in to the game. This screen also has a secondary part where a player can register a username, in case they don't already have one.

The *main menu screen* is the central hub of the game, and consists of three buttons that take you to other parts of the game. There is also a chat available on the main menu screen and all of its subscreens, where players can send messages to everyone playing the game in their vicinity.

Pending challenges is a screen containing a list of all active challenges a player has received. Every challenge creates a notification in the phone's status, but for players' convenience, they can also access them in game.

Manage Own Tribe is where a player reviews their area, places new buildings and upgrades their existing buildings.

The *Map screen* is where a player can view a “map” of the area and the active players close to them. This is not a map displaying the player's physical location, but merely a game-abstraction. It shows a desert landscape, with every close player displayed as their own tribe. A player can challenge other players by clicking on their tribe.

Players that click on another tribe on the map screen is taken to the *Send challenge screen*. In this screen, a player decides the stakes of a duel, what category of questions the duel should have, and how many questions.

After sending a challenge, the *Duel start screen* is displayed until the challenge is either accepted or declined. The player receiving a challenge is shown a screen similar to the *send challenge screen*, but with buttons to either accept or decline, and without any possibility to change stakes or size.

Once the duel starts, both players are shown similar screens. First a screen where the question is displayed on the top part, and with 4 alternatives on the bottom half. Players then click on the alternative they think is the right answer, and the next screen depends on whether or not the answer was correct.

If the player answered correctly, they are shown a screen where they can choose what weapon to use, or not to use a weapon at all. After they have decided, they press a button that takes them to the next question.

If the answer was wrong, the player is shown a screen that informs them of this, and they can click a button to advance to the next question.

When a player has answered all questions, and used their final weapon if the answer to the last question was correct, they are taken to the *battle result screen*. On this screen a player can see whether or not they won the duel, which questions they answered correctly, and which ones they did not answer correctly. The answer to the questions are not displayed here, but a player can press a button next to each question to be taken to a relevant website where they can learn more, and hopefully find the answer. Finally, at the bottom of the *battle result screen* is a button that takes the player back to the *main menu.*

9.5.2 Game Screens

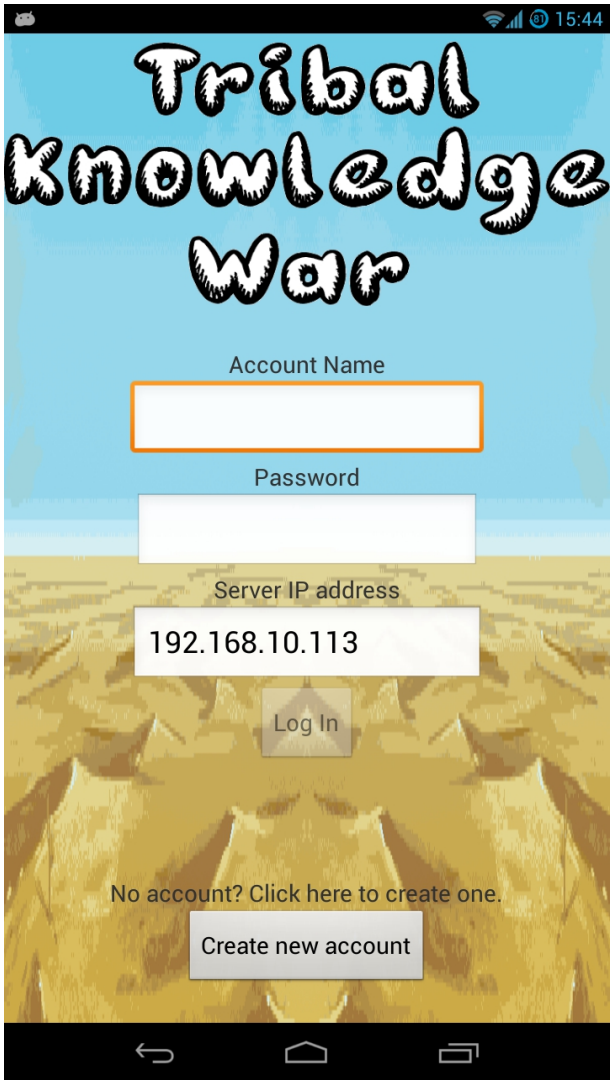

Figure 14: Login Screen

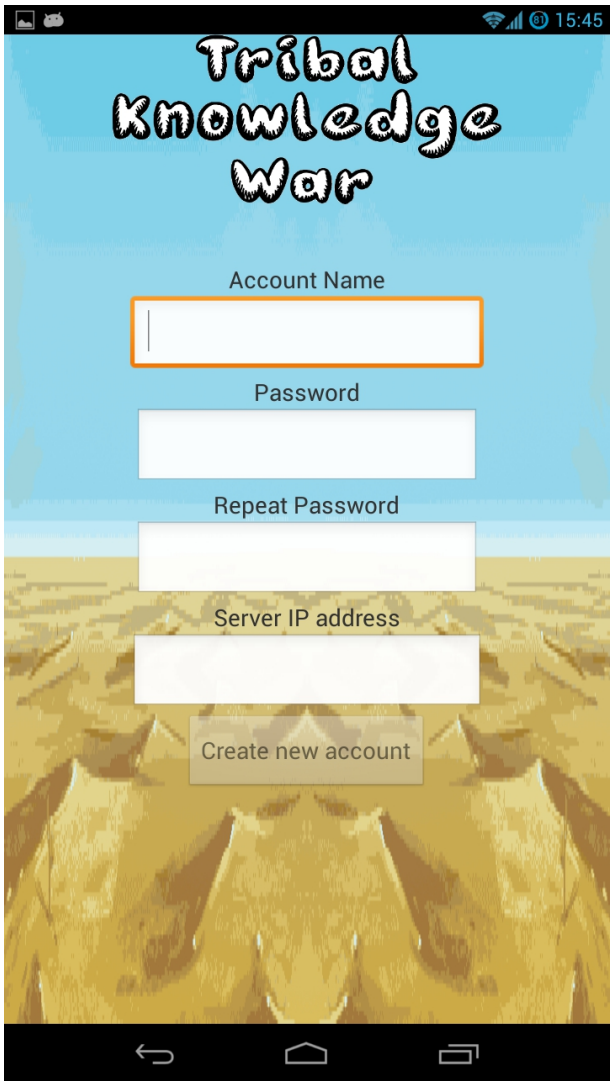

Figure 15: Registration Screen

Figure 14 and Figure 15 show the Login screen and the Register account screen respectively.

The Login screen is the first screen a user sees when they start the application. If the user has registered an account previously, they can type in the name registered and the corresponding password and then press the button marked “Log In”

If the user has not previously registered an account, they must press the button marked “Create New Account,” which will make the screen shown in Figure 4 appear. There, the user can type in the desired Account Name, and the password they wish to use. After filling out the form, pressing “Create new account” will create the account, and the user will be logged in.

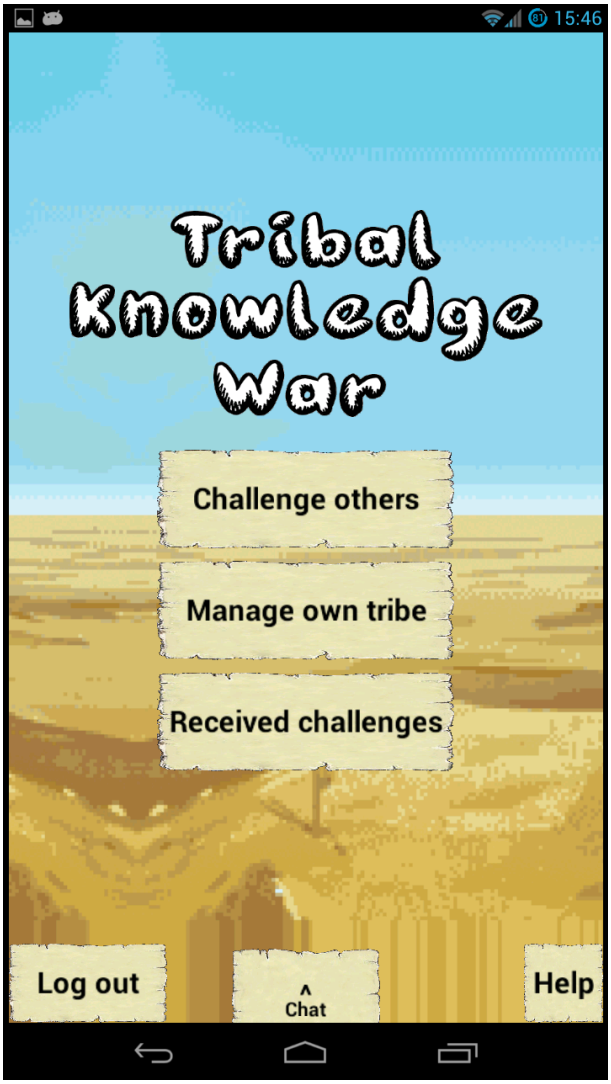

Figure 16: Main Menu

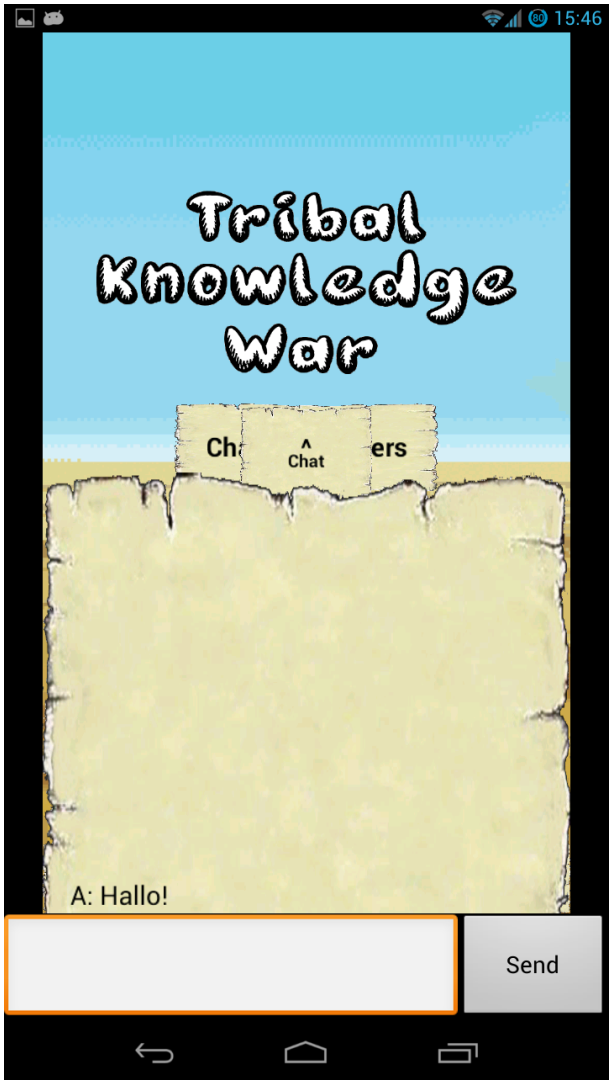

Figure 17: Chat Overlay

Figure 16 and Figure 17 show the Main Menu of the game and the chat overlay.

The Main Menu is the central hub of the game, where the player can access most of the game's functions. Pressing any of the buttons will take the player to the relevant screen.

At the bottom of the Main Menu, there is a tab marked “Chat.” Pressing this brings up the chat-overlay, where the player can chat with other players in the area. Any message written there will appear in the chat for every other player in the player's vicinity.

The chat can be accessed from almost anywhere in the game by pressing the tab at the bottom of the screen. The only time the chat is not accessible is during a duel.

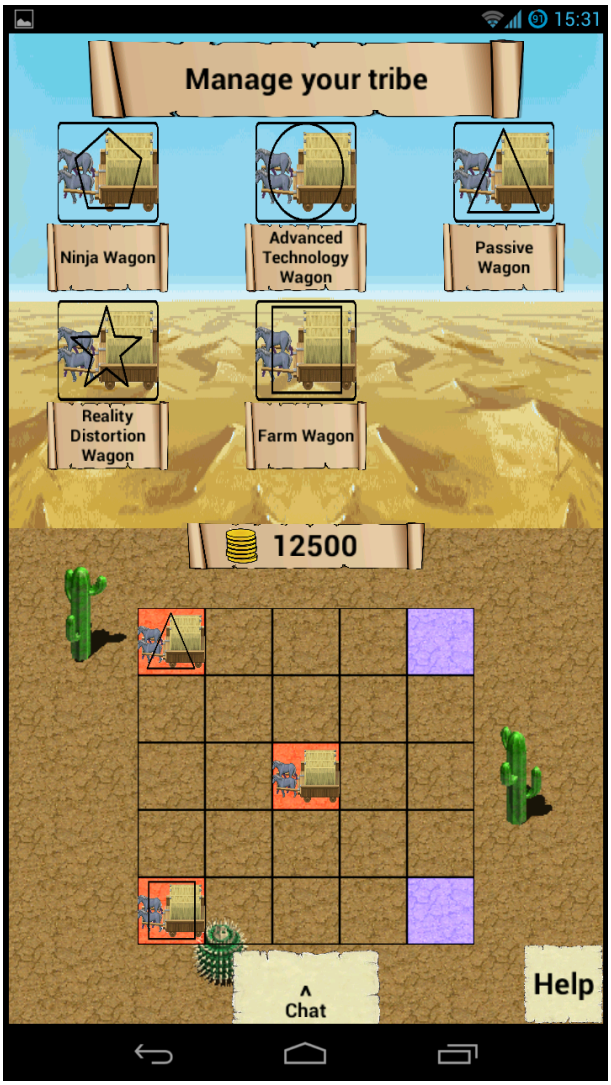

Figure 18: Manage Own Tribe

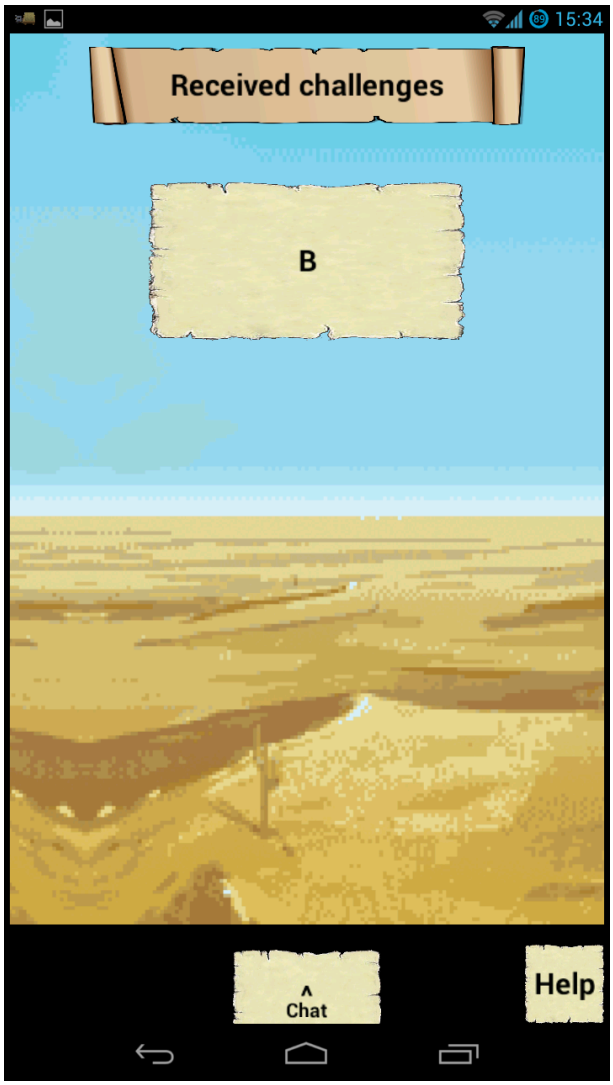

Figure 19: Received Challenges

Figure 18 and Figure 19 show the screen named “Manage own tribe” and also the screen showing received challenges that are waiting to be accepted or declined.

In the Manage your tribe-screen, the player can view their existing buildings, upgrade them, and build new ones. New buildings are placed by dragging them from the top to where the player wants to place it. Players can see information about a building by clicking on it.

Figure 16 shows a challenge that has been received by a player named “B”, but the challenge has not been accepted or declined yet. Accepting or declining challenges can be done in two ways. In addition to creating an item on that screen, a notification is also created in the Android notification bar at the top of the screen when a player receives a challenge. The notification is easiest to use if the game is running in the background when a challenge is received, and the “Received Challenges”-list may be easier to use if the player is actively using the game when a challenge is received.

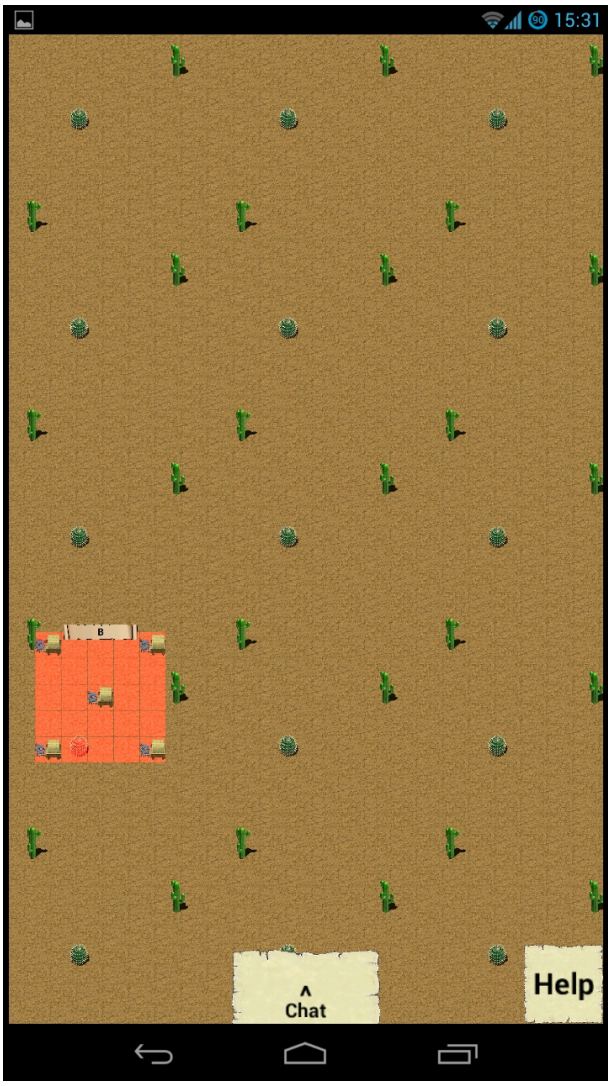

Figure 20: Map Screen

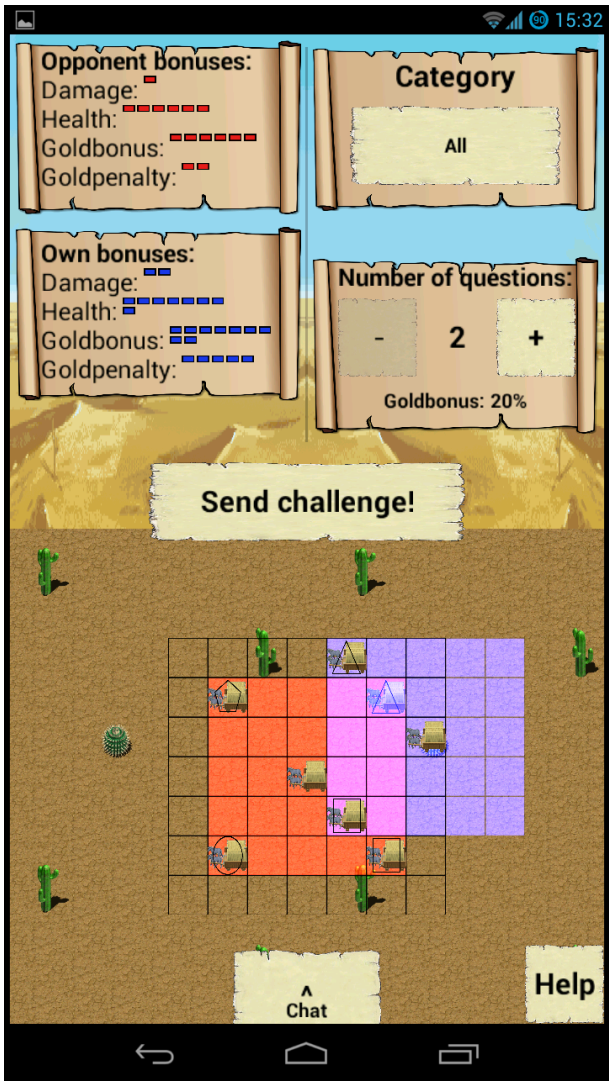

Figure 21: Set up challenge

Figure 20 and Figure 21 show the map where players can see other players in the area, and the screen where the player sets up and sends a challenge.

The map-screen is a game abstraction, and meant to be a more entertaining way of displaying players that are ready to be challenged than a simple list. Other players are placed around the map, and the player can scroll the map around in search of other players. When they find a player they want to challenge, they can press on them and be taken to the screen where they can send a challenge.

Figure 18 shows how a player sets up a challenge. The red square at the bottom of the screen is the area of the opponent, while the blue square is the player's own area. The player decides where to attack from by clicking in the empty part of the grid surrounding the opponent's area. Both the player's and the opponent's bonuses are displayed in the top left part of the screen. On the top right part, the player can choose what category of questions to have in the duel, and how many questions the duel should consist of.

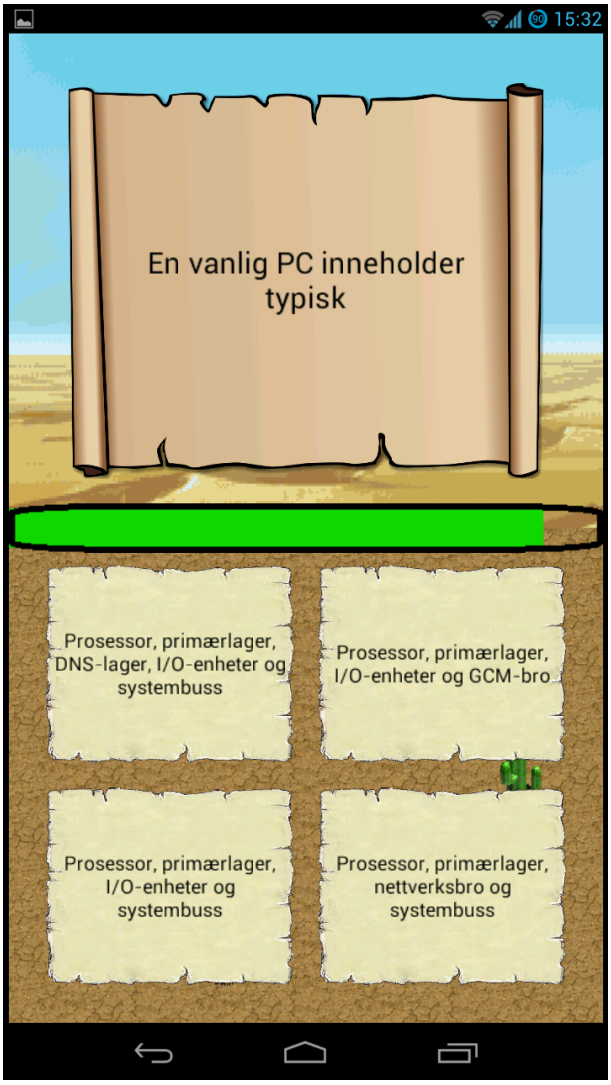

Figure 22: Default question

Figure 23: Choose weapon

Figure 22 and Figure 23 shows the standard way a question is displayed, and also how the player chooses a weapon after answering a question correctly.

The question itself is displayed on the top part of the screen, and the alternatives on the bottom. The green bar in the middle is a visual display of the timer. It ticks down, and when it runs out, the player fails the question automatically.

Figure 11 shows the screen where the weapons available to the player are displayed. Clicking on one of the icons highlights it, and pressing the big button at the bottom will then use the weapon and proceed to the next question.

Figure 24: Battle result

Figure 24 shows the screen that is displayed when a battle is over. On the top part of the screen, the player can see whether they won or lost, and how many points they got and how much money they earned.

The rest of the screen is a list of the question that appeared during the battle, and the green check mark shows that the player answered that question correctly. If a player wants to learn more about a certain topic, they can press the button with a question mark on the right part of the screen, and a browser will open and display a page with background information about the relevant question.

Pressing the button at the bottom takes the player back to the main menu.

Chapter 10
Game design

This chapter consists of two parts. The first one is mostly descriptive explanation of how the game evolved from initial ideas to the finished game, and the second part will be a discussion of why certain choices were made and what the developer hoped to achieve.

10.1 Evolution of game concepts

Competition can be a strong motivating factor for players to improve [6], and by stimulating the sense of competition, the game can hopefully be made more engaging and more fun. Any game that focuses on duels between two players will inherently have competitive aspects, but by adding additional gameplay-mechanics to the quiz-duels the competition will feel more direct.

While just quiz-duels can be entertaining in themselves, it is desirable to give players more reason to play a game. By creating additional gameplay that surrounds the quizzes, players will be motivated to play the game for other reasons than just the desire to answers questions correctly, and that will hopefully make the game more effective as a learning tool**.**

In *Tribal Knowledge War*, one of the first ideas for making the game more interactive, was to give players the possibility to influence each other during duels. Quiz-duels are inherently single-player activities, and simply answering several questions and then get told that you either won or lost does not necessarily feel rewarding or motivating. The inspiration for players to influence each other was taken from puzzle-games that have versus-modes, more specifically Super Puzzle Fighter [13], and Tetris [12]. When a player does something correctly, they are rewarded by being able to give the opposing player penalties, which in the case of Super Puzzle Fighter means adding more blocks to the opponents “board”. This idea was adapted to *Tribal Knowledge War*, and resulted in the introduction of “weapons” to the quiz-duels. The idea is that answering questions correctly gives the player an advantage, and allows them to make the next question more difficult for their opponent. Possible weapons could be: giving the opponent less time to answer the next question, thereby giving them less time to think; manipulating the way question is displayed in some way, for example displaying the text mirrored, and thereby making it very difficult for the opponent to read the question; or giving the opponent some kind of point-penalty, to force them to answer the next several questions more quickly in order to make up the deficit.

As mentioned at the start of this chapter, it is desirable to create an additional layer of game around

the quiz-duels, and the first draft for such a “meta-game” was heavily based on strategy-games like Risk [29] and Civilization [30]. A player would have a territory to defend, resources to control, and buildings to maintain. Since *Tribal Knowledge War* was going to be a pervasive game, the resources would be placed in the world, and players would have to be near these to have control over them. The size of a player's territory would decide how many resources they would be able to control, and winning and losing duels would influence the size of the territory. To more effectively protect a resource, a player would place buildings in his territory, and these buildings would grant additional positional bonuses and give access to the “weapons” mentioned earlier. Buildings would have an “area of effect” so that their bonuses were only granted if it was close to the area of the territory being attacked. Players would physically position themselves in the world in such a way that they could attack other players from specific angles, thereby creating a need for players to strategically place their buildings in their territory to avoid having weak spots that opponents could exploit.

After considering and discussing the implications these ideas would have for the game, several problems were found. Problems with this idea include, but were not limited to:

- Too much complexity for a game without a “win-condition”. Players would probably not feel motivated to put up with the need for micro-managing without any reward outside of in-game currency that can only be used to create more buildings and therefore results in more micro-managing.

- Could be technically difficult to have GPS position be reliable enough to have the attack-angle make sense. GPS can take some time to get an accurate fix on position, especially indoors.

- No natural limit on the size of a player's territory. The size would have to range from ridiculously small to ridiculously large in order to give the system any depth, but that would create problems because the physical world often puts limits on the area in which a player has ability to move around.

- A study looking at people's attitude towards pervasive games [8] suggests that players do not want to play a game that puts restrictions on where they have to be in the physical world in order to play. Having to guard resources in certain locations could be unpopular.

- If the resources were randomly placed or generated in some way by the server, it is possible to imagine a situation where it would be worthwhile for a player to sit at home with the game open, guarding a resource that no one else knows exists. The game should not reward passive play. Alternatively, if the resources were hand-placed by the developer, it would severely limit the amount of places the game could be played.

The general idea of having strategic elements was not entirely abandoned, but many parts were changed.

Instead of having a player's territory increase or decrease, it would be fixed to a 5x5 grid, wherein the player's buildings could be placed. And instead of having the possibility to place the buildings freely inside the area, it would only be possible to place buildings in four different locations, namely the four corners. In order to make sure that the game still involved some kind of strategic choice, there are several kinds of buildings available that grant different bonuses, but a player does not have enough space for all of them. So the player must choose what buildings they think gives

them the biggest advantage.

Part of the reason for the expanding territory that was originally planned, was to give the player a feeling of progression. They would become more powerful by winning more duels, because they would have room for more buildings and therefore have the upper hand against weaker players. When the expanding territory was removed, there was need for another mechanic to give a sense of progression. This mechanic was that players would be able to use gold to upgrade buildings. Upgraded buildings would be more powerful, and give a significant advantage over other players with weaker buildings.

The concept of players attacking others from certain angles were kept, but using the physical location of the player as part of it was removed. Instead, the game uses physical location only to find other players in the vicinity, and a player can challenge any one of those other players to a duel. When challenging another player to a duel, they get to see the opponent's 5x5 grid and what bonuses are granted by the different buildings. The player can then choose to attack from any angle they want, and any buildings involved in the battle risk being destroyed.

The idea of resources, and having them placed in the world was entirely abandoned.

10.2 Discussion on the game's final design

In this section we will try to discuss *Tribal Knowledge War's* design, to look at how it uses elements from game design theory and whether or not these elements are implemented correctly. We will also look at how the game's design supports the guidelines for making educational games fun, and if the attempt to be educational meshes well with the desire to also be entertaining and engaging in its own right.

10.2.1 Mechanics, dynamics and aesthetics

Hunicke, LeBlanc, Zubeck [2] describes a way to examine what they refer to as a game's *Mechanics*, *Dynamics* and *Aesthetics*, and how to use these observations to examine the different aspects of a game, and how the different aspect interact. It suggests looking at a game's "Aesthetics of play" first, to see what experiences the game hopes to give, and then see how the game's dynamics and mechanics should support these aesthetics. Note that the word *aesthetics* is used in a somewhat figurative way in the article.

Following Hunicke, LeBlanc, Zubecks taxonomy of aesthetics, *Tribal Knowledge War* aims to pursue these aesthetics: *Challenge*, *Discovery* and *Fellowship*.

Challenge means "game as obstacle course" and can refer to dynamics such as *competitive play*, the desire to win over other players, and to *overcoming obstacles* the game throws at the player. *Tribal Knowledge War* tries to implements these dynamics to make the game fun. *Competitive play* should be obvious, since the entire game is based around winning quiz-duels against other players, but it also tries to be competitive through giving a way to compare oneself against other players. The ability to upgrade buildings gives a player the ability to feel superior compared to others, for example when looking for opponents to challenge. Weaker players are colored green, while stronger players are colored red. Seeing many red players will hopefully motivate someone to become

stronger.

The other dynamic, *overcoming obstacles*, is implemented as the different weapons that can be used by players during battles. Answering questions quickly, reading text backwards, panning the camera in search for the question, are all obstacles that players must overcome in order to win.

Discovery as an aesthetic is not a main focus, and is therefore not as prominent in the design of *Tribal Knowledge War*, but it is represented by the possibility for players to experiment with different combinations of buildings in order to find an optimal one. Some of the available buildings give access to weapons, while others only grant powerful passive bonuses. This hopefully creates an interesting choice for the player, in that they have to choose between bonuses that are not directly comparable to each other.

Fellowship is a part of the *Tribal Knowledge War's* nature, in that you can only challenge players that are close by. In much the same way as competitive board games are as much about winning as just watching your opponents become annoyed when you pull ahead, *Tribal Knowledge War* aims to create situations where players can laugh with and at each other as they win, lose, or use particularly frustrating weapons at suitable moments. The mechanics that support this, are the nature of the weapons available. They aim to be an annoyance, but not to force a loss on a player.

10.2.2 What makes learning fun?

Having discussed the reasoning for the major design decisions in *Tribal Knowledge War*, it would be wise to also look at educational games, and at what could make these games fun *and* educational. Thomas W Malone [3] discusses the main characteristics of a good educational computer game, and he organizes these characteristics into three main categories: *challenge, fantasy,* and *curiosity.*

Challenge is defined in this way in Malone's article: “In order for a game to be challenging, it must prove a goal whose attainment is uncertain.” *Tribal Knowledge War* has goals on two different levels, both of which are uncertain: the first is the short-term goal of winning a duel, and the second is the more long-term goal of upgrading buildings and becoming more powerful. Both of these goals are uncertain because they depend on the skill of a player's opponents. The long-term goal does not have a clearly defined state of being attained, and it is also possible and likely for a player to lose progress towards this goal by losing duels and thereby losing buildings. Therefore, *Tribal Knowledge War* has two of the characteristics named by Malone, namely *variable difficulty level* and *multiple level goals*.

Fantasy is a part of game if the game “shows or evokes images of physical objects or situations that are not actually present,” according to Malone. He also distinguishes between what he calls *extrinsic* and *intrinsic* fantasies. The difference between them is to what degree they integrate themselves into the game's gameplay. An *intrinsic* fantasy is one where “not only does the skill depend on the fantasy, but the fantasy depends on the skill.” This means that the gameplay is a part of, and makes sense, in the game's fantasy. *Tribal Knowledge War* attempts to have an intrinsic fantasy by being about warring nomad tribes that use quiz-duels to determine power, and who sabotage each other by using dirty tricks. This fantasy is a silly one, but it's an attempt to give player's a sense of why they are answering questions as part of the gameplay.

Curiosity is “the motivation to learn, independent of any goalseeking or fantasy-fulfillment.” The goal of the game-developer should be to evoke the player's curiosity, and a way to do this is to make

sure that the game is neither too complicated nor too simple in regards to the player's existing information. *Tribal Knowledge War* tries to implement this by encouraging players to search for an optimal tactical solution concerning choice and placement of buildings. By making the players ask themselves whether it's better to have many buildings that grant gold bonus or many buildings that do damage, it hopefully motivates players to play the game to experiment and try to find what is the best setup.

Chapter 11
Requirements

11.1 Functional requirements

In this chapter, the functional requirements of the game is presented. Table 6 shows all these requirements.

Table 6: Functional requirements

FR1	The player should be able to register a username/password on the server.
FR2	The player should be able to log in using the username/password.
FR3	The player should be able to see an in-game representation of all players in the physical proximity.
FR4	The player should be able to choose a player in the proximity and send a challenge to a duel.
FR5	The player should be able to decide how many questions are in a duel when creating the challenge.
FR6	The player should be able to set the terms of battle, by selecting “angle of attack” to decide their own bonuses.
FR7	The player should be able to receive a challenge from another player.
FR8	The game should a display a screen showing information about an incoming challenge, and allow the player to accept or decline.
FR9	The game should be able to synchronize duels and start it at the same time for both players.
F10	The game should be able to automatically find the device's GPS position and send latitude and longitude to the server.
FR11	The game should be able to get a list of questions from the server to use in a battle.

Table 6 (cont): Functional requirements

FR12	The game should display a question with four alternatives, and allow the player to select one of them.
FR13	The game should keep track of a timer so that a player only has a certain amount of time to answer a question.
FR14	The game should be able to calculate a score after each question based on time left.
FR15	The player should be able to use a weapon after a correctly answered question to disrupt the opponent.
FR16	The player should be automatically logged out of the game if they have not communicated with the server for a certain amount of time.
FR17	The game should automatically decline a challenge if the player does not accept or decline it within a certain amount of time.
FR18	The game should make it impossible for a player to be challenged to duel while they are already busy playing one.

The functional requirements were decided upon based on an analysis of the design of the gameplay, and therefore most of them are requirements to implement important functions, such as making a working timer so that players only have a limited time to answer questions.

It was decided that the server should keep track of a username/password-combination (FR1 and FR2) in order to give players something to work towards. The game could function without persistent storage of the player's progress, but it was thought that this would not feel as fulfilling to the player.

FR16, FR17 and FR18 are important because there needs to be a protection in place to avoid that players lose money and duels while they are busy or not available in some other way. For example if a player is busy playing a duel against someone, another player should not be able to exploit this by challenging the player to a duel because the challenge will be automatically declined if the player does not finish the current duel quickly enough. This is bad because declined duels counts as a loss for the one that declined.

11.2 Non-functional requirements

Table 7 shows the non-functional requirements of the game.

Table 7: Non-functional requirements

NFR1	The game should be easy enough to learn that it's possible to learn and play within ten minutes.
NFR2	The game should run on Android OS, version 2.2 or later.
NFR3	The device the game is running on must have the ability to find its position through the Android Network Locator.

NFR1 sets a specific requirement for the time needed to learn the game because of the user

experiment that is to be conducted later. Players need to learn the game before they can properly test it, and if the game is too complex, it would be difficult to have players give accurate feedback in a limited amount of time.

Chapter 12
Architecture

12.1 Choice of architecture

Early in the design phase, the it was reasoned that the most relevant architectures would be client/server or peer-to-peer. Because the game was going to be a multiplayer game, it was clear that players would need to communicate with each other, either through a central server or directly with each other. At first, peer-to-peer seemed the most suitable, because it would not make the players dependent on a server that may experience down-time or heavy load. Having players find each other using Bluetooth, for example, would build the pervasiveness of the game directly into its functionality, in that you would only be able to play with someone that was within Bluetooth-range. Using peer-to-peer would also mean that time would not have to be spent designing a server, which could possibly result in more time to develop the gameplay.

However, peer-to-peer does have disadvantages. As mentioned, communication would probably have to be through bluetooth, and using bluetooth on modern smartphones is a very battery-draining activity. It would also be much harder to make sure the available set of questions and answers remained up to date, since they would have to be stored locally on the phone. Therefore, to update the list of questions, every player would need to download an update of the game and install it. One could imagine having issues where two players wanted to play together, but did not have the same version of the game installed and therefore did not have access to the same set of questions.

A client/server architecture was then considered, and that has some advantages over peer-to-peer. The problem of synchronizing the questions is easily solved by storing the questions on the server, and having it deliver them to the client at the start of every duel. Communicating with a server is not as battery-intensive as using bluetooth, because communication with the server can be done over WiFi or through the mobile network. Also, by doing all game-logic on the server, it would be harder for any player to cheat.

There are several disadvantages with a server/client-architecture, however. First and foremost, the server must also be developed and that might take a lot of time away from developing the game itself. The architecture would also create a problem with implementing the pervasiveness of the game, because the server would have to store the location of all the clients and the clients would have to update their position often to have it be relevant. This is a problem because GPS location can be both slow and inaccurate, especially indoors.

After some evaluation, it was decided that the game would use a client/server-architecture. The disadvantages of this architecture was considered to not be as serious as the disadvantages of peer-to-peer, and a client/server-architecture would allow greater control over the game, for example by having the ability to add or remove questions at any moment.

12.1.1 Server

When deciding on server solution, one thing in particular was considered very important, namely that the system had to handle disconnects well. Because the game is played on a mobile phone, disconnects are almost inevitable, for example because of unstable WiFi in public places, or that the phone has to change from a mobile network connection to a WiFi connection. There were two natural options to solve this problem, either make the server stateless, so that a disconnect is unimportant because the client gives enough information every time for the server to do the right thing; or use an already developed protocol like WebSockets, which have built-in protection against disconnects so that it does not have to be taken into account while making the game.

A stateless server would be easier to implement, because it is similar to what the developer already has experience in developing. The server could use Java servlets to handle requests from clients, and the requests could be HTTP requests with a payload containing JSON objects, which are easy to handle both conceptually and technically. A relational database could be used to store all information needed to facilitate communication between clients.

The best solution for using Websockets seemed to using Nodejs as web server. Nodejs allows one to develop a web service by running Javascript code serverside, and it is very popular and with great performance. The main problem with this solution is that the developer in this project has no previous experience using either Javascript or Websockets-technology, so it might take a lot of time to learn how to do even basic things before the development could begin.

Both solutions seemed to be adequate for this prototype, and the choice fell on developing a stateless server with Java servlets and Java code, because this would mean that the development could start earlier. The server software chosen for the web service to run on, was Apache Tomcat. Apache Tomcat was chosen over the alternatives because of the availability of tutorials and other documentation compared to for example Jetty.

12.1.2 Storage

Using a relational database for permanent storage was the most obvious choice, and there are several free and effective database management systems available, for example MySQL. It's easy to set up, offers lot of flexibility, and also handles transactions automatically so there's little need to worry about data integrity. It's also easy to set up a connection between a MySQL database and an Apache Tomcat server. For these reasons, the use of a MySQL database was chosen.

12.1.3 Client

Two main options were considered on how to make the client. The client could either be a native Android application, or a web application. The two options are discussed in Chapter 6.

The choice eventually fell on developing the game as a native Android application. This was considered to be the best option, because it would utilize the experience of the developer best, and it would therefore be reasonable to expect it to lead to the best product.

12.2 Interaction between client and server

Below is a figure showing the physical view of the system.

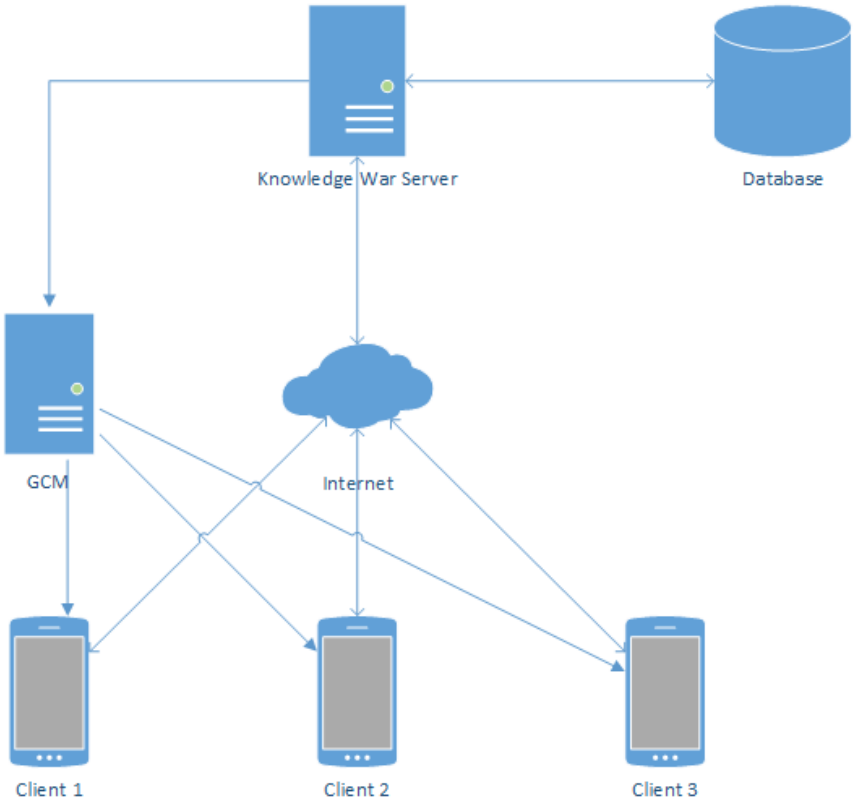

Figure 25: Physical view of system

Figure 25 shows how the client and server communicate with each other. The communication happens either through the web service calls, or through GCM (Google Cloud Messaging). The web service is hosted on an Apache Tomcat web server, and clients requests information from the server using HTTP requests. When the server needs to inform the clients of something, the server sends the message and the client's unique ID to the GMC service, and the message gets “pushed” to the correct device.

All HTTP requests sent by the client are HTTP POST requests. These requests contain a JSON Object in its body, describing what action should be performed by the server. The server is *stateless*, meaning that it does not keep track of the state of the clients in between each request received. For that reason, every request to the server needs to contain all necessary information about the state of

the client for the server to be able to perform the correct action.

Table 8 shows examples of JSON objects being sent from the client.

Table 8: Contents of JSON objects

Scenario	**JSON object**	**Servlet**

Register account	{
	“command” : ”register”,
	“accountname” : ”Player1”,
	“password” : “1234”
	}
		LoginServlet

Send challenge	{
	“command” : “challenge”,
	“challengee” : “Player2”,
	“risk” : “5,2;0-1,0-4;3-4;0-4”,
	“size” : “3”,
	“category” : “0”,
	“accountname” : “Player1”,
	“password” : “1234”
	}
		ChallengeServlet

In the first scenario, the request is sent to the servlet named *LoginServlet.* The servlet reads the body of the HTTP request, and finds that the command is “register” which means that it creates a new entry in the database and inserts the username and password that are included with the request into that entry. In the second scenario, the request contains the command “challenge”, along with the necessary information about the battle and who should receive it. The request also contains the name and password of the player who sent the challenge, so that the web service can check the database and be reasonably certain that the request is sent by a valid player. The server then checks the database to find the rest of the information necessary to notify the other player that they are being challenged.

Figure 26 attempts to show a Logical View of the entire system, with the most important parts of the client and server and how they interact with each other.

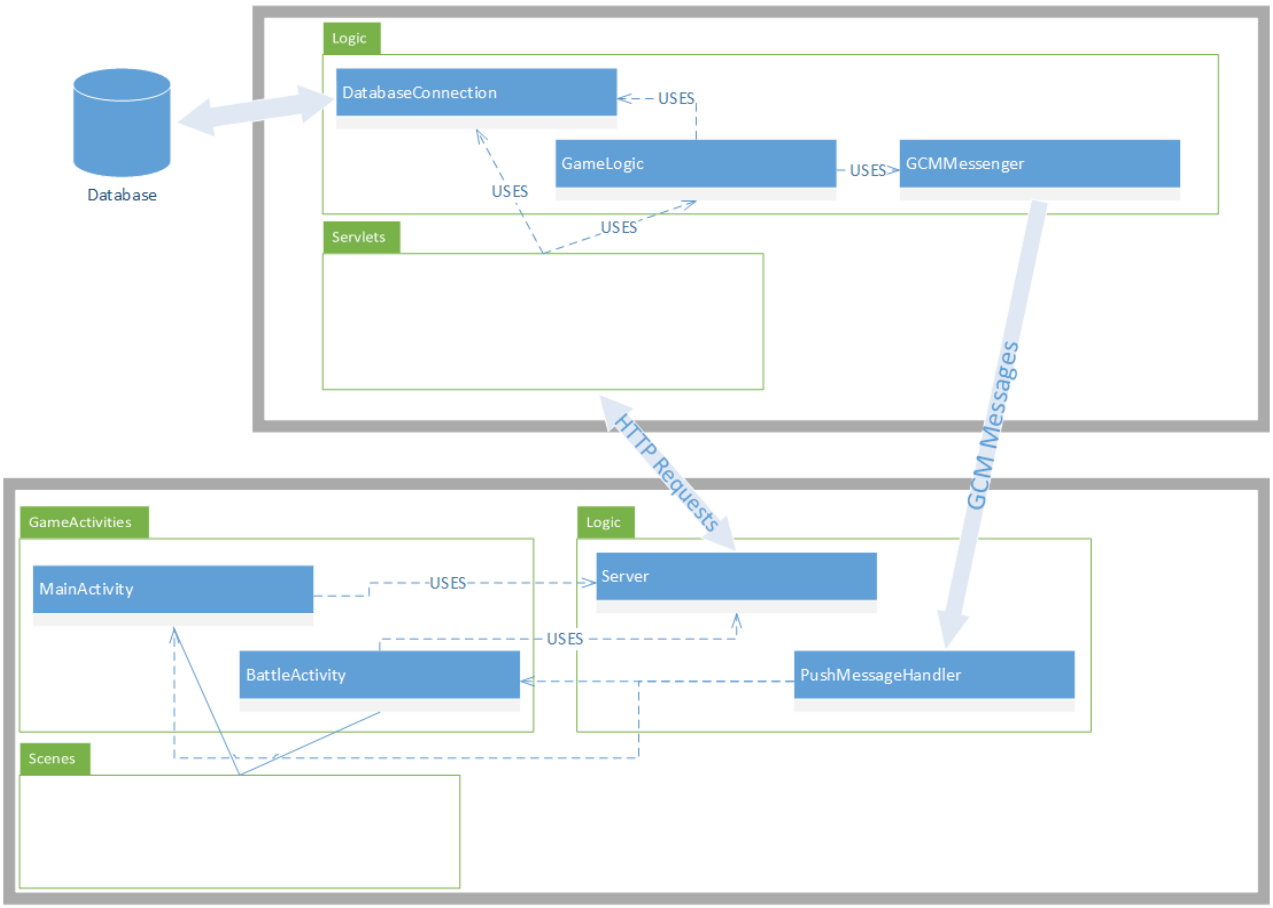

Figure 26: Logical View

The client is inside the frame on the bottom, and the server is inside the frame at the top.

The Activities inside the *GameActivities*-package uses the various scenes contained in the package *Scenes* to display information on the screen.

The various Activities in the client handle all communication with the *Tribal Knowledge War* server through the class called *Server*. The *Server*-class sends an HTTP request to one of the servlets on the server, and the servlet that received the request handles it by using the *GameLogic*-class or the *DatabaseConnection*-class.

If the server needs to alert a player of something, for example if the player is being challenged, the server uses the class *GCMMessenger* to push messages to clients. On the client side, the class *PushMessageHandler* receives and handles the pushed messages.

12.3 Storage

For permanent storage, the system uses a MySQL database. Figure 27 shows the EER-diagram for the database.

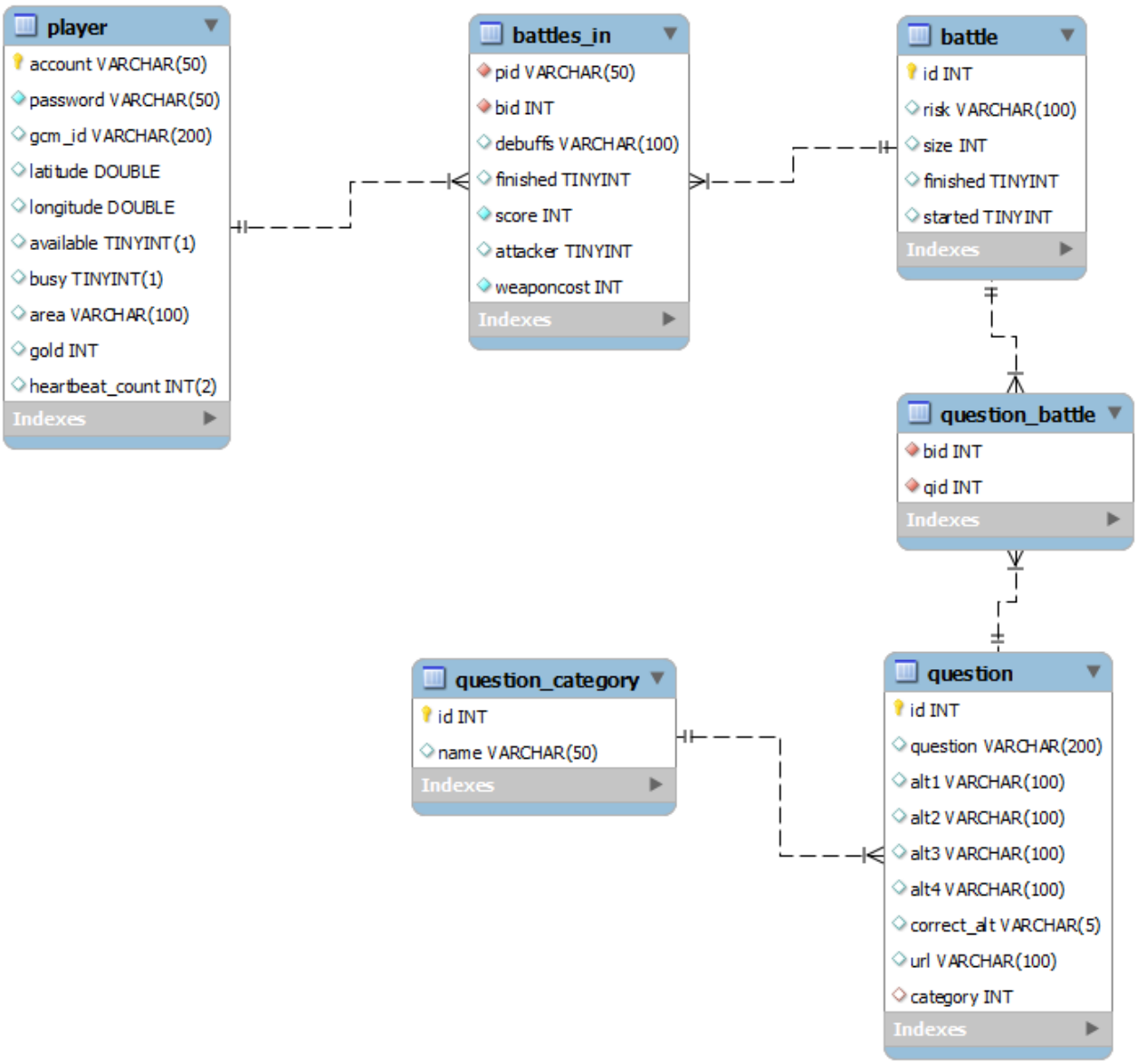

Figure 27: EER diagram for Tribal Knowledge War database

The table *Player* stores information about any player that has registered an account. It contains information needed for the game, such as their coordinates and also the player's *area*. The *area* is the in-game state of a player, and contains information about what *buildings* the player owns, what *rank* those buildings are, and where they are placed. The *area* is stored as a string following a specific format so that it can be easily parsed by the game and server.

The table *Battle* contains information about a specific duel, such as how many questions there are in the duel, and whether it has finished or started. It also contains information about the *risk* of battle. This *risk* is a string-represantation of what are at stake for the players, what part of their *area* is involved and what buildings are at risk of being destroyed if they lose the battle.

Question contains all the questions with its alternatives. Each question also has a URL associated with it, where a player can find more information about a specific question if the player wants to learn more.

Question_categories is the table containing the different categories in which questions can be.

The table *battles_in* stores information about a player's role in a specific battle. Score and amount of gold spent on weapons are stored here, and also whether or not a player has finished. The column *debuffs* contains information about what weapons are used against a player during a duel.

Question_category and *battle_question* are simple tables that associates questions and battles with categories and questions respectively.

12.4 Client

The client is a native Android application, written in Java. The game was developed using a framework called AndEngine (see Chapter 2.3.3). Because of the use of AndEngine, the client has a somewhat different structure than regular Android applications. Instead of using regular *Activities*, a subclass called *SimpleBaseGameActivity* is used. This subclass abstracts away the handling of the game loop and makes the creation and use of textures easy. To display items on the screen, it is necessary to create instances of the class *Scene* (or of a subclass). A scene can be compared to a window or a screen in a game. The main menu, for example, can be one *Scene* and when the player presses on a button to go to another screen, the game changes to another *Scene*. A *SimpleBaseGameActivity* can create several scenes, and switch between them based on user input.

12.5 Handling GPS and finding suitable user locations

Asking for location through use of GPS or Android's Network Location Provider on mobile devices is a tricky subject, because the developer need to find a very delicate balance between accuracy and use of power.

The model suggested by Google for getting best performance is built around maintaining a “Current best estimate” and asking for new location updates when the user initiates an action that would benefit from it [27]. *Tribal Knowledge War's* functionality and dependence on maintaining a good estimate that is stored on a remote server, makes Google's model not ideal. The user seldom initiates any action that would benefit from an updated position, apart from looking for players in the area, but on the other hand, it would be a very common occurrence that *other users* would benefit from a single user maintaining a good location-estimate on the *Tribal Knowledge War*-server.

The fact that players often use the location of other players introduce some specific problems. If the server treats the location as a set of coordinates, latitude and longitude, and responds to requests for “close” players by returning a list of players that are within a specific deviance from the given coordinates, this is very sensitive to inaccurate results. In order for “close players” to be a term that makes sense, the deviance from the user's location can't be too big, but if it's too small, there's a chance that the result is so inaccurate that the server can not find any other players that are close to the player.

Testing and evaluation of the game resulted in the decision that it would be best to focus on using Android's Network Location Provider instead of GPS. The reasons for this is that is most likely that the game will be played indoors, and GPS is not well suited for indoor use. Using the Network Location Provider creates other problems, however, as it is not suited to “track” users. That is, it can not update the location whenever the device moves a certain distance. Using a combination of GPS and the Network Location Provider was deemed unnecessary, because a user's location would

probably considered too inaccurate the moment they stepped into a big building, so it would be meaningless and wasted battery power to accurately track them while outside.

Several implementations were tested during development, and it became necessary to simply find a suitable compromise between accuracy and fast determination of location. It ended up as a relatively naïve implementation, but one that works well for its use.

Every five minutes the client asks the Android Network Location provider for its location, and the result is the device's latitude and longitude given in decimal degrees. The coordinates are uploaded to the server and stored in the database. When the server needs to find a list of players that are "close" to a given player, it uses the latitude and longitude of the player directly.

The server has a defined constant named *CLOSE_PLAYER*, and the server looks for players whose latitude and longitude fall in the interval of the player's coordinates plus-minus this number.

The method has worked well during development and testing, but occasionally created some problems where the location was so inaccurate that it was placed outside the interval created by players right next to the player. It took some amount of testing to find a good balance between size of *CLOSE_PLAYER*, and frequency of updates. Increasing the frequency of location-updates lessened the problem, but unless the location is updated every couple of seconds there will still be a possibility for an extended amount of time where the positions is wrong, and updating very often is not good for battery-life.

Increasing the size of the area of what counts as "close" works well, but it's not a satisfying solution from a developer's perspective.

12.6 Textual representation of player status and battle risk

There are two important concepts in *Tribal Knowledge War* that needs to be stored in the database on the server, namely the player's *area* and a battle's *risk*. Recall that the *area* is the territory a player controls, and it consists of a 5x5 grid with buildings of a certain rank placed in certain locations. A battle's *risk* is the representation of what is involved in a battle, that is, how big the overlapping area is and precisely which squares from the two players' grid are involved.

Both these concepts are converted to relatively simple String-representations and are stored as text strings in the database. By being consistent in the use of separators, it's easy for the client to parse the string extract the information.

The player's *area* are converted to a string that follows this standard:

```
<area-size> : <building> ; <building> ; … ; <building>
```

Area-size is the same for all players, "5x5", and never changes. The reason it exists is because originally, the size of the area was supposed to change as the player earned more money. When that idea was abandoned it was decided that it was best to keep it since it wouldn't have any negative consequences, and it would keep possibilities open.

The *building*-elements in the string follow this pattern:

```
<type> : <rank> : <x-coordinate> , <y-coordinate>
```

Type is a single letter that is different for every type of building. What building is represented by what letter is stored in the game itself. *Rank* is a number that tells what rank the building is, and is a number from 1 and 5. *X*- and *y-coordinate* says where the building is placed in the player's grid. Both of these are numbers from 0 to 4.

An example of a player's area with two buildings would look like this:

```
5x5;D:1:4,0;A:2:2,2
```

Here, the player has a building of type D and rank 1 in the top-right corner, and a building of type A and rank 2 in the middle of the grid.

The *risk* of a battle is more complex, but it follows the same concept of converting it from game concepts to a textstring with different parts separated by semicolons and commas. The risk-string follows this pattern:

```
<placement of attacker in relation to defender>;
        <attacker's involved squares>;
         <defender's involved squares>
```

In order to explain how the *risk* is created, one needs to recall what it is trying to represent in-game. The battle consists of two player, the *attacker* and the *defender*. The attacker sends the challenge, while the defender receives it. It is the attacker that sets up the battle, and they do so by being presented with the defender's area and then placing their own area next to it (see Figure 9). The defender's area is a 5x5 grid that is placed in the center of a 7x7 grid, which means there is an extra line of squares surrounding the defender's area. The center of the attacker's area is placed in one of these surrounding squares. This way, one can give coordinates of the attacker's placement in relation to the defender's area, and this is used for various computations. The defender's top-left square has coordinates (0,0) while the bottom-right has (4,4), so if the attacker places their area in the very top left, they will be placed at the coordinates (-1,-1). If they attack from the very bottom right, they will be placed at at (5,5).

In addition to the placement of the attacker in relation to the defender, the risk also contains information about which squares are involved. The representation of the involved squares follow this pattern:

```
<x1> - <x2> , <y1> - <y2>
```

Because the involved squares always is a rectangle, one can represent it simply by saying which coordinate it starts at and what coordinate it ends at.

Below is an example of the attacker attacking from the very bottom right.

```
5,5;0-1,0-1;3-4,3-4
```

12.7 Model View Controller

The entire system follows a design that's similar to the Model View Controller architecture pattern, where the server is the model, the web service is the controller, and the clients the view. The clients

have very little ability to influence anything regarding calculations of battle result and similar things, they only send messages to the web service about what they want to do, and the web service uses the information found on the server to determine whether or not that action is allowed, and then carry out the action.

If a player wants to build a new building, for example, the client sends a request to the web service describing what building they want to place where. The web service then fetches information about the player from the database, makes sure the player has enough gold, and then places the building, and updates the database. This general procedure is used for almost every action done by the player.

The only place where the MVC-model is not followed, is when it comes to calculating points earned during a battle. The clients calculate the amount of points earned by using information about how much time the player took to answer a question, they include the amount of points in the request sent to the web service when a battle is finished, and the server trusts completely that the amount of points it given is correct.

This is both a break from the MVC-model and a potential security risk. The view itself directly influences the model, and because a client should never be trusted it opens up possibilities for players to cheat by modifying the program to give a false score. However, this was considered to be the best for the game, because it would give the best experience for most players. Giving the server the responsibility for timing each question would be technically difficult because of the stateless nature of the server, and it would also mean that any delay between information being sent from the client to when it arrives on the server would negatively impact a player's score. It would be frustrating for a player to lose battles just because of network issues.

Chapter 13
Implementation

This chapter will describe the architecture of the system and the implementation of the server and client in detail.

13.1 Server

The server consists of four packages. Package *adrian.master.knowledgewar.server* is the root package, and contains several Java servlets that handle interaction with the clients. The other packages are subpackages of the root.

Package *adrian.master.knowledgewar.server.logic* consists of the main classes that do functions that are important for the functionality of the server, such as calculating the result of a duel, or communicating with the database.

Package *adrian.master.knowledgewar.server.gameconcepts* contains the classes that represent different objects in the game, such as a *Player*, a *Building* or a *Battle*.

Package *adrian.master.knowledgewar.server.constants* contains classes that consists of constant values that are necessary for the other classes, such as names of tables and columns in the database, and SQL statements used when accessing the database.

Figure 28 shows the package diagram for the server.

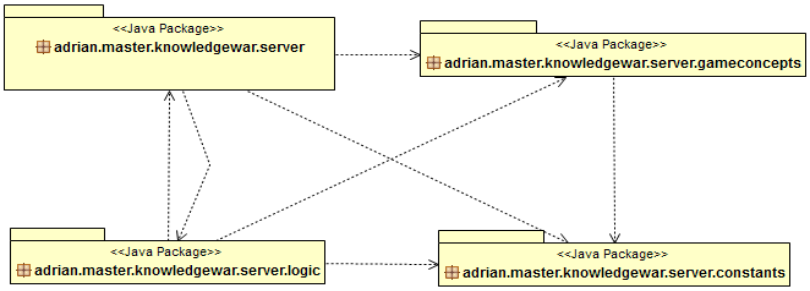

Figure 28: Package diagram for Tribal Knowledge War server

Figure 29: Relationship between single servlet and other important classes

Figure 29 shows the relationship between a single servlet and other important classes. The other

servlets' relationships are very similar, but may use the logic-classes in a different manner. As illustrated in the figure above, *BattleServlet* uses the classes *GameLogic* and *DatabaseConnection* to handle requests from clients. Below is a figure showing an example of such a request. It is a sequence-diagram showing the interactions between the classes when two players finish a duel.

13.1.1 Receiving a request from a client

As mentioned, the server consists of several Java-servlets, and any request from a client is sent to one of these servlets.

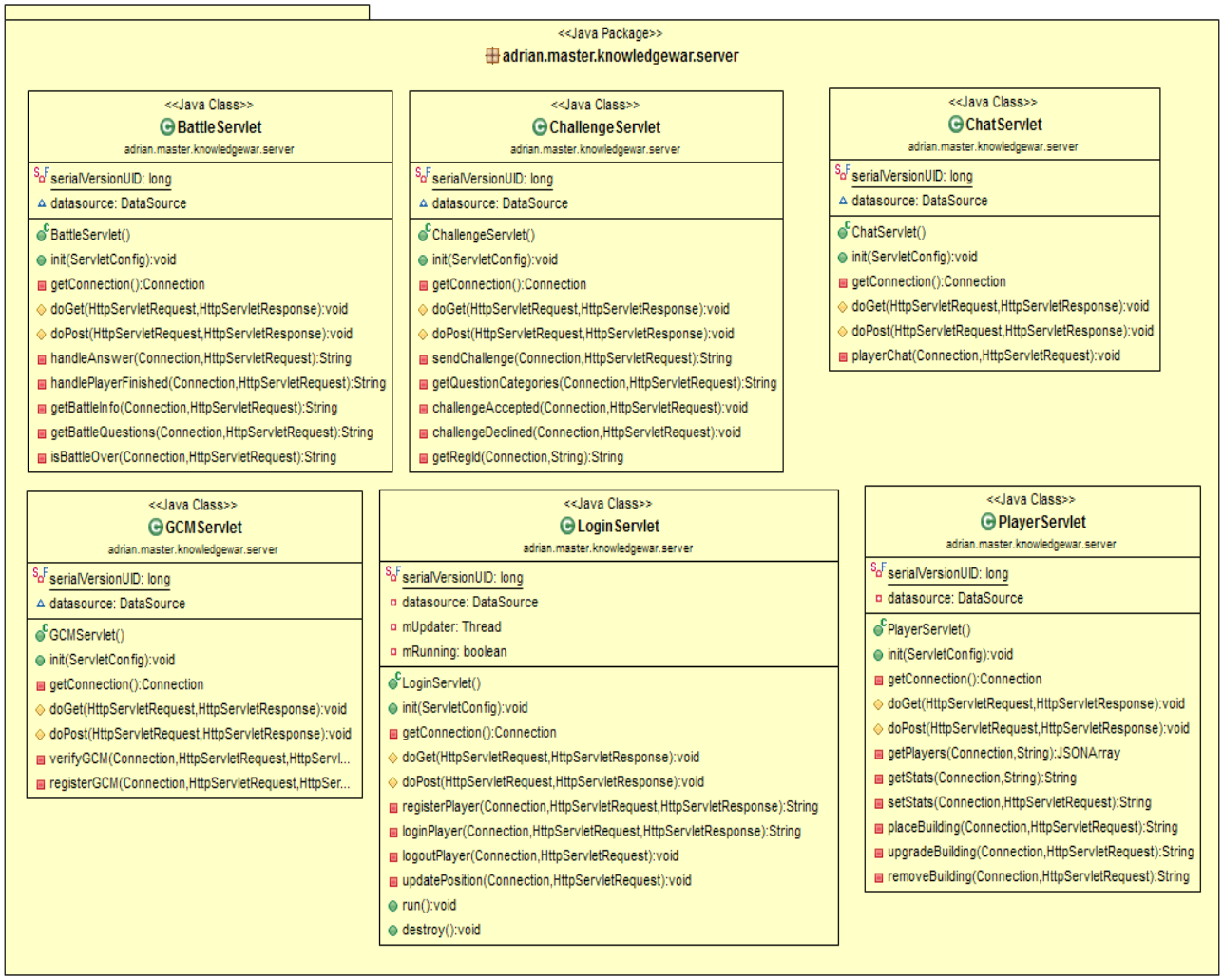

Figure 30: Java servlets

Figure 30 shows the root package, and all the servlets. Each of the servlets handle requests that are related to each other in logical sense. As an example, *BattleServlet* handles all the requests sent during a duel, such as when a question is answered, and when a player finishes.

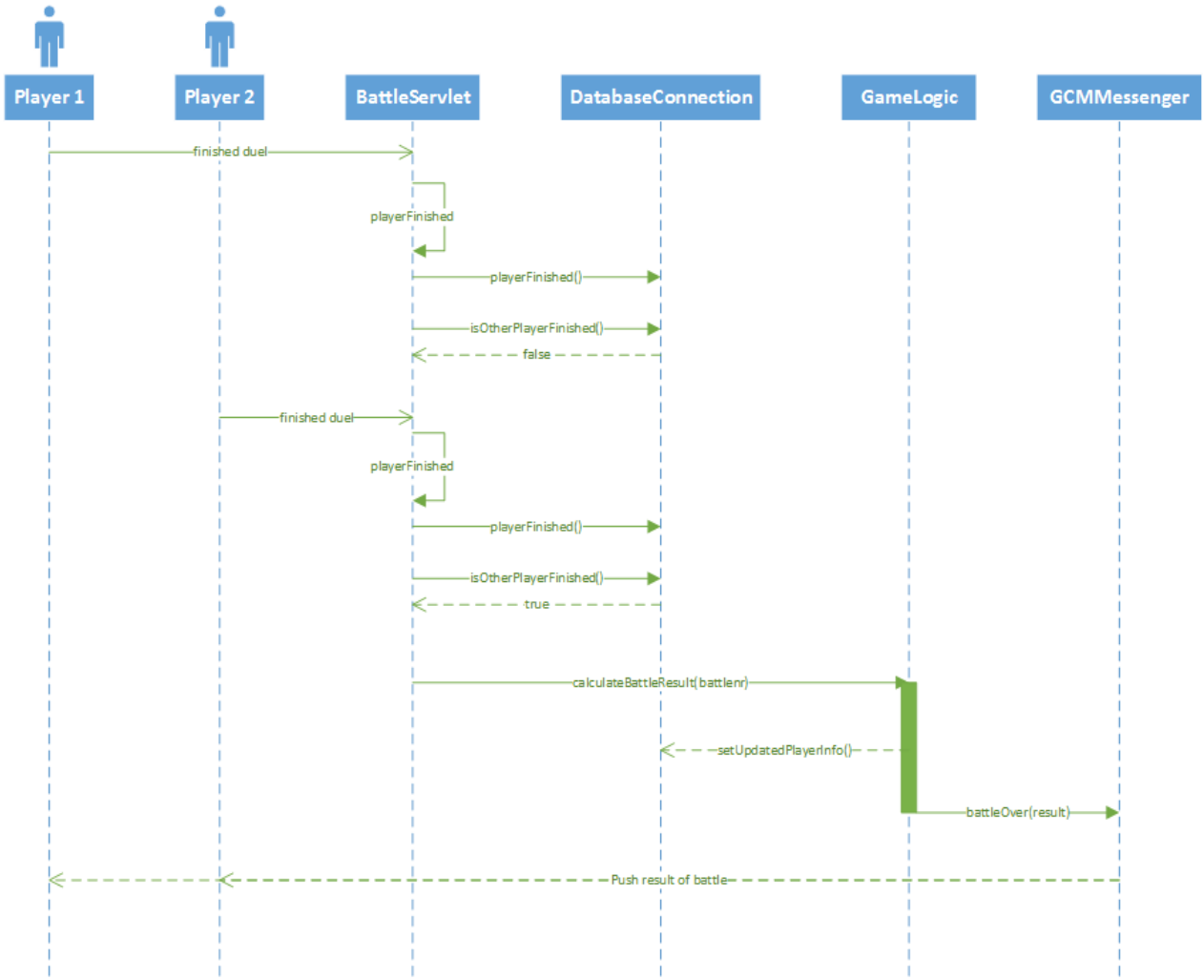

Figure 31: Sequence diagram showing two players finishing a duel

Figure 31 shows how the *BattleServlet* receives a message from both players and how it handles these messages. After marking a player as finished in the database, the servlet checks to see if the other player is finished in the same battle. If they are, the battle is over, and the result can be calculated. If the other player is not finished, the servlet does nothing and waits for the other player to finish. The *GCMMessenger* class uses *GCM* to push messages from the server to the clients, so the clients do not have to continually poll the server while waiting for the result of the battle.

The result of a battle is calculated on the server, and a winner is declared based on the amount of points the clients report to have earned. Players earn points by answering questions correctly in a timely manner, and the player with the most points is the winner.

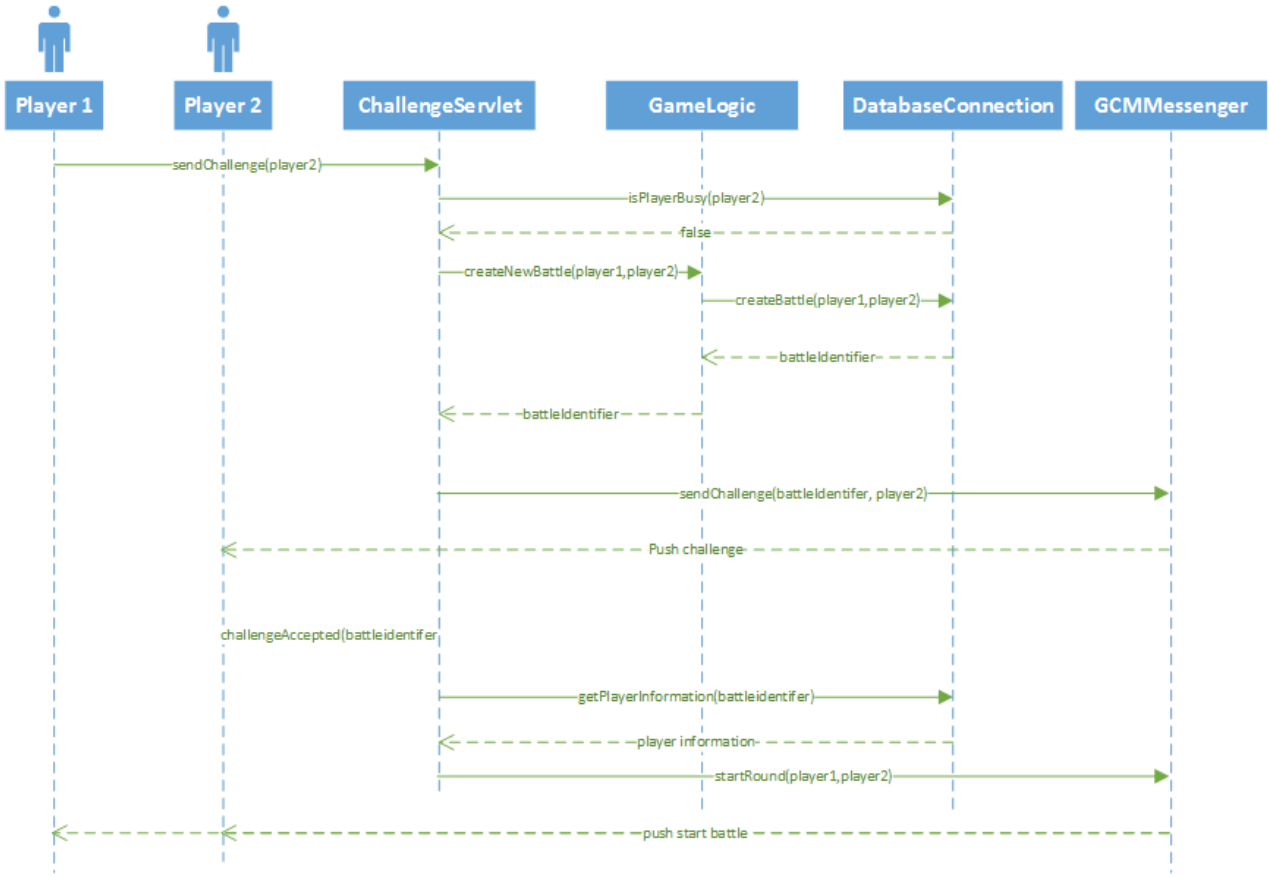

Figure 32: Sequence diagram showing a player challenging another

To demonstrate in what way other servlets use the *Logic*-classes, Figure 32 shows a sequence-diagram describing the sequence of events when a player challenges another player to a duel.

The *ChallengeServlet* receives a request from Player 1 to challenge Player 2 to a duel. *ChallengeServlet* checks to see if Player 2 is marked as busy in the database before sending the challenge. A player is busy if they are currently dueling someone else. It is important to make sure that a player is not busy before sending a challenge, because challenges are declined automatically if a player does not accept or decline it within a certain amount of time, and a declined challenge is an automatic loss for the person being challenged. After making sure that Player 2 is available, the class *GameLogic* uses *DatabaseConnection* to create a new battle in the database, and return the unique identifier for the battle. This identifier is used by both server and the clients to refer to a specific battle in requests.

13.1.2 Receiving a location update and heartbeat

The fact that the server is stateless creates one big problem, namely that it has no way to directly notice that a player is no longer available and mark them as offline in the database. If, for example, a player turns off Wi-Fi on their phone without having mobile data enabled, they would suddenly be impossible to contact for the server, yet they would still appear accessible in game for other players. Since challenges that are not explicitly accepted or declined counts as a loss, it would be problematic if it was possible for other players to send many challenges to a player that is not really present, and cause the unavailable player to lose all gold.

To avoid this scenario, each location update sent by a client, also acts as a heartbeat. As can be seen in Figure 14, the Player-table in the database has a field named *heartbeat_count*. The server runs a background process that increments this field by 1 whenever a certain time passes, but every time the server receives a location update from a player, that player's field is reset to zero. Every time the server increments the field, it also checks its value after the incrementation, and if the value is greater than 3, the field is set to 0 and the player immediately marked as logged out.

13.2 Client

The client consists of 10 packages, 9 of which are subpackages of the root *adrian.master.knowledgewar*. The root package only contains two classes, the *IntentService* that receives messages pushed from the server through GCM, and a *LocationReceiver* that finds the GPS-location of the device.

The package *adrian.master.knowledgewar.activites* contains the regular Android-activities in the game, that is *Activities* that are not subclasses of AndEngine's *SimpleBaseGameActivity*. Not every *Activity* needed to use the possibilities of AndEngine.

All the classes that subclass *SimpleBaseGameActivity* are in the package named *adrian.master.knowledgewar.gameactivities*. These are most the activities in the game, and all of them use the various game-related functions of AndEngine.

Similar to the structure of the server, all classes that represents objects in the game, such as a *Player*, a *Battle* or a *Building* are in the package named *adrian.master.knowledgewar.gameconcepts*.

The package named *adrian.master.knowledgewar.helpscenes* contains subclasses of Scene, and all of them are scenes containing information meant to help explain the game to players. In-game they are accessed by pressing a button marked “Help” on the screen.

Adrian.master.knowledgewar.interfaces is a package that contains interfaces used by various other classes.

The package named *adrian.master.knowledgewar.logic* is similar to the logic-package on the server. It contains several classes containing methods that does different calculations or handles other functions of the game.

The two packages *adrian.master.knowledgewar.scenes* and *adrian.master.knowledgewar.subscenes* both contain the different scenes in the game, from the scene showing the main menu, to the scene displaying a confirmation dialogue when the player wants to remove a building.

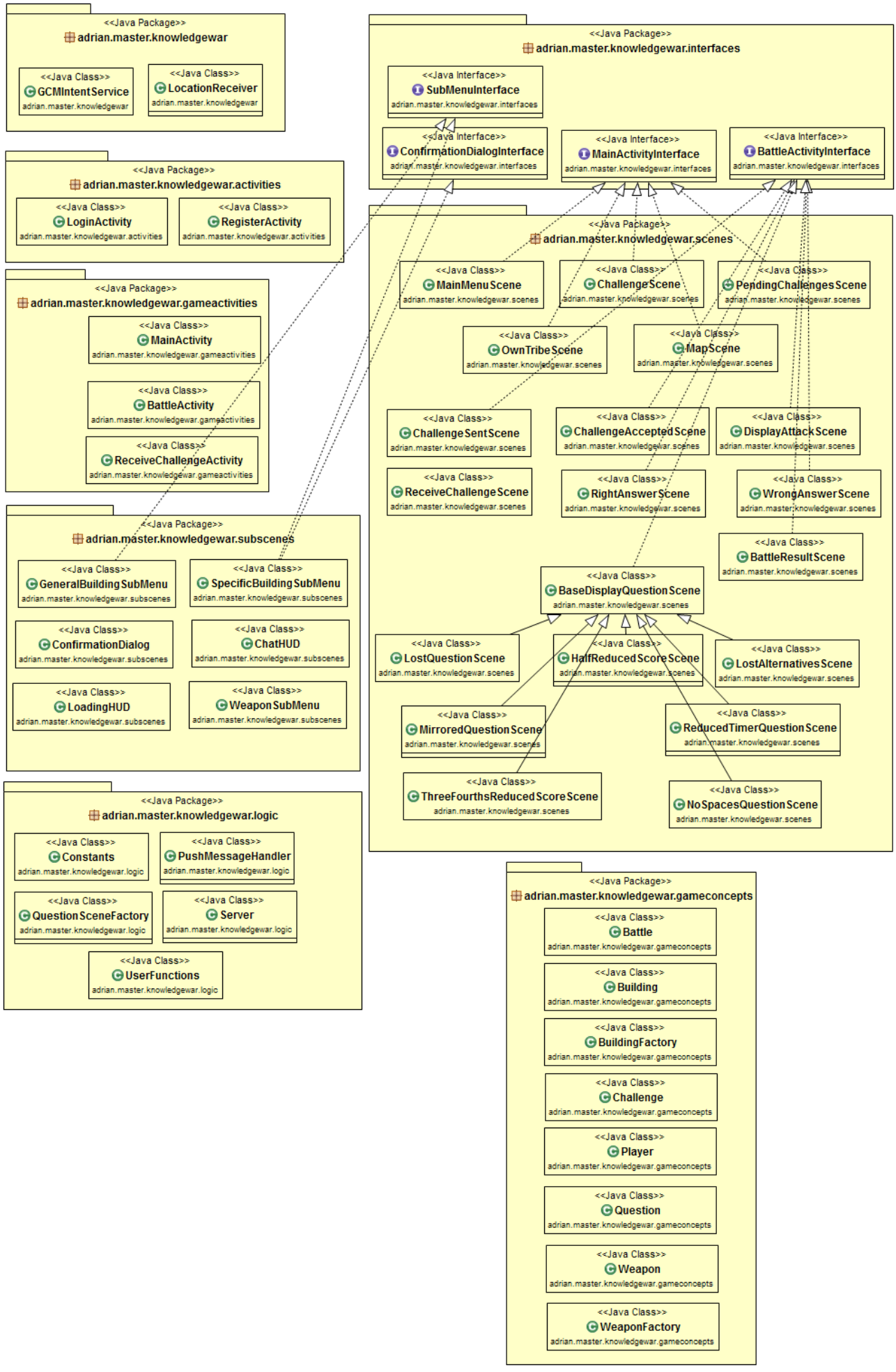

Figure 33: Class diagram of client

Figure 33 shows a Package Diagram of the client, including the most important classes, subclasses and use of Interfaces.

13.2.1 Communicating with the server

Tribal Knowledge War always communicates with the web service by using the class named *Server*. This class contains methods that handles all the different scenarios in which the client needs to send a HTTP-request to the server.

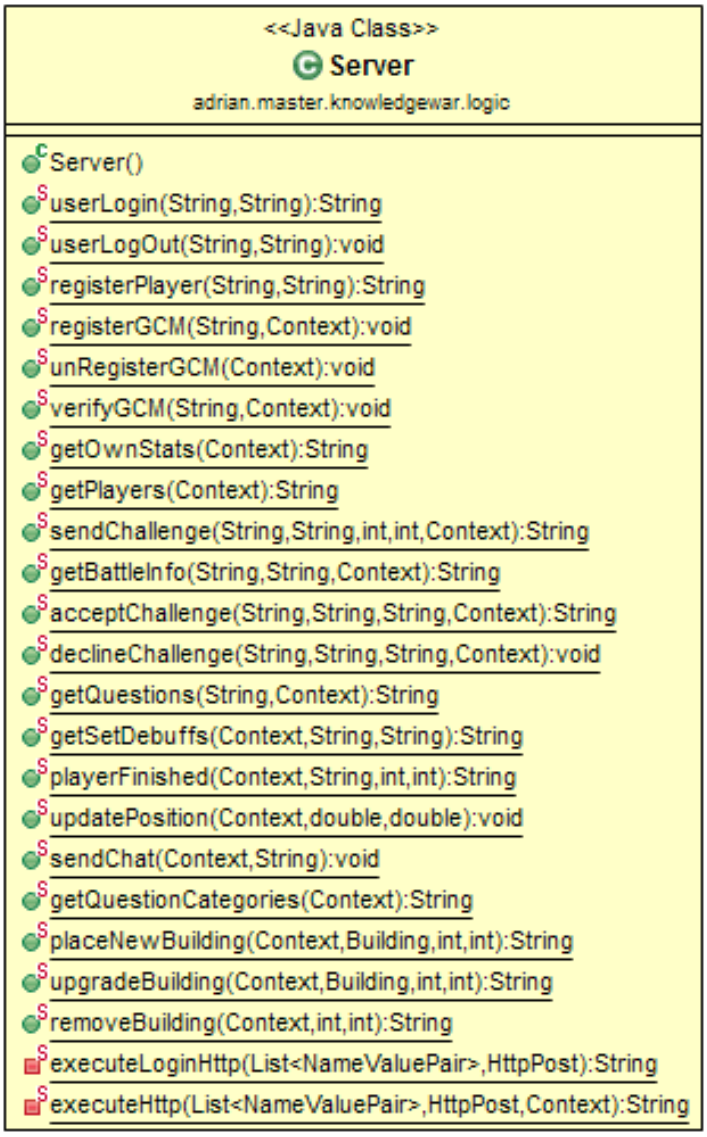

Figure 34: Overview of Server-class

Figure 34 shows an overview of the Server-class and its methods.

In order for any of the Activities in *Tribal Knowledge War* to communicate with the server, it is not sufficient to simply call one of the Server-class' methods. An Android application is not allowed to do “slow” operations in the UI thread, and “slow” operations include network communication, something that is very important for *Tribal Knowledge War*. In order to communicate with the server, the Activity needs to start a background thread that does not block the UI, otherwise the UI would freeze while waiting for a response from the server, something that would be frustrating for

the user. In Android, the easiest way to do this is to extend the Android class called *AsyncTask*, which is a class that makes it easy to do work in a background thread. *AsyncTasks* usually interacts with the UI when it has finished its background work, which usually makes the implementation of such a subclass very specialized. For that reason, it is most practical to make separate private subclasses for each possible task. Each of the main Activities in *Tribal Knowledge War* has several private *AsyncTasks*.

Figure 35 shows the two biggest *Activities*, and their private *AsyncTasks*.

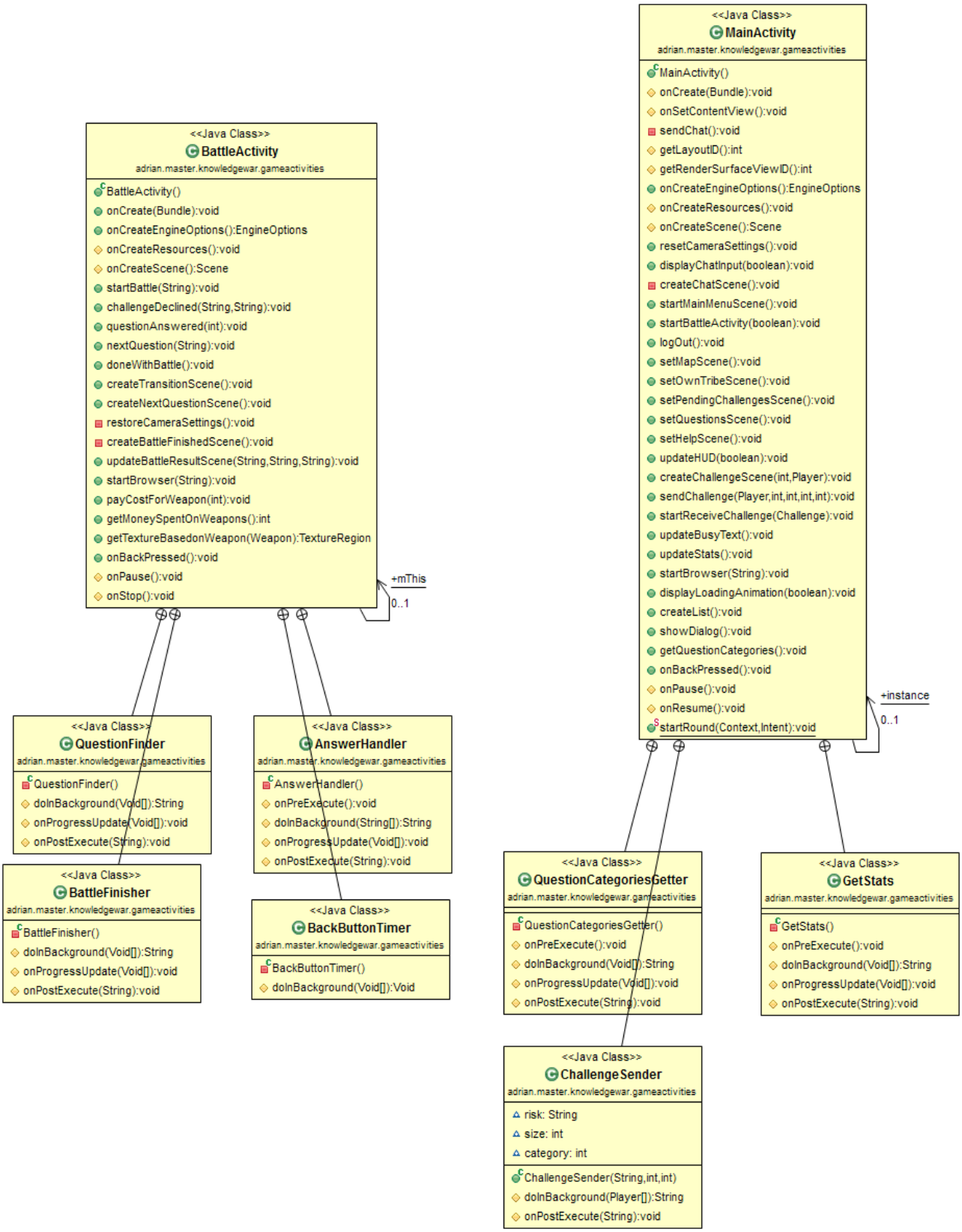

Figure 35: Activities with their private classes

Note that each of private classes has several methods, and one of these methods is called *doInBackground()*. Everything inside that method is performed in a background thread. *BattleActivity* is the *Activity* that is active while playing a duel, and its *AsyncTasks* communicate

with the server to get the questions before a duel starts, to find information about a given battle, and to send an answer to the server and tell the server what weapon the player wants to use. The *AsyncTask* called *BackButtonTimer* is simply a timer that automatically forfeits the battle if the player presses the Back-button on his phone twice within 3 seconds.

Many of the different Scenes also have their own private *AsyncTasks* depending on what the scene needs to do.

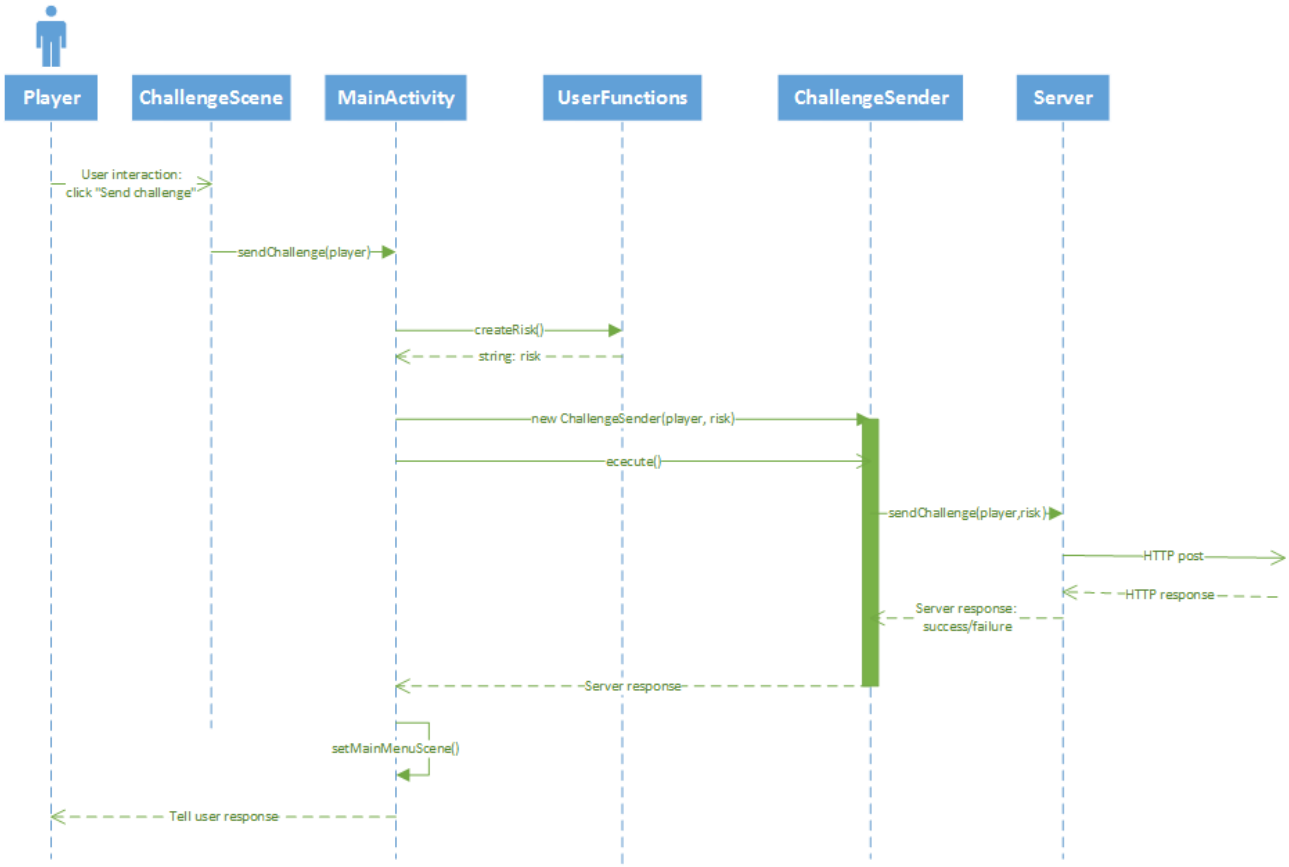

Figure 36: Sequence diagram showing a player sending a challenge

Figure 36 shows the interaction between the classes when a player challenges another to a duel. To send a challenge, a player must be in specific scene in-game, called *ChallengeScene*. After they have decided on the size of the battle and where to attack from, they click the "Send Challenge" button in the UI. The *MainActivity* then creates an AsyncTask-object (the *ChallengeSender*-actor in the diagram) that does work in the background.

The risk-string that is returned from the *UserFunctions*-class is a textual representation of what area is involved in the battle. This text is stored in the database on the server, and used to determine the result of the battle.

13.2.2 Receiving Push Notifications

When the server needs to alert a player about an event, for example that the player is being

challenged to a duel, the message is “pushed” from the server using GCM, as indicated in the physical view of the system. In order for an Android application to receive messages through GCM, several permissions need to be declared in the application's manifest. In addition, the application must implement a subclass of the Android class *GCMBaseIntentService*.

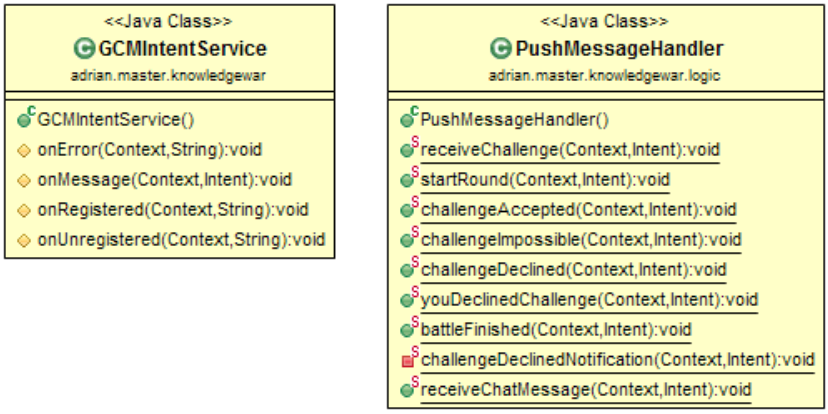

Figure 37: Overview of classes involved with push notifications

Figure 37 shows the two classes involved with handling pushed messages, with their methods. When the device receives a message sent by the *Tribal Knowledge War* server, the OS passes the message to the application, and calls the *onMessage(...)*-method in *GCMIntentService*. The message is not in JSON-format, but much like JSON objects, it can contain several key-value-pairs, and it is used by the applicattion in the same way as the JSON objects shown in table 4 (chapter about Design). The *IntentService* reads the “command” and based on that it calls the appropriate method in the class *PushMessageHandler* in the Logic-package. If the message contained information about a challenge, the *PushMessageHandler* creates a notification that is displayed in the Android status bar along with playing a sound, to alert the player that the game requires an action.

13.2.3 Use of interfaces

In Figure 33 one can see that there are several interfaces used by various classes in the client. These interfaces are very simple, and only have between 1 and 3 methods.

MainActivityInterface are implemented by all scenes that are used by *MainActivity*, and this offers methods that make it easy for the activity to correctly handle the situation if the player presses the back-button on their Android-device. The activity can simply pass the button press along to the current scene, and have that scene handle it. This is important because the default action of the back-button would be to return to the *Main Menu* , but the currently active scene could have a submenu of some sort visible, and the back-button should remove this submenu instead.

The other interfaces serve the same purpose, and are extremely similar in both form and function.

13.2.4 Local Storage

The client side application does not need to store large amounts of data, because all information about a player such as gold and owned buildings is stored on the server and sent to the client on request. However, some values are stored on the client, mostly to make the user experience better, and these values are stored using Android's built in storage for applications called *SharedPreferences*.

SharedPreferences can store simple key-value-pairs, and these values are private for each application, meaning it can not be accessed by other apps. This means that it can be used to store relatively sensitive information without any security-issues.

When a user logs in to the game, the application stores both username and password in the *SharedPreferences*. This makes it possible to automatically log a player back into the game in case the OS has decided to kill the application while it was in the background. It also makes it possible to automatically send the username and password along with any request to the server, and that way have some basic authentication for every action done by a user.

SharedPreferences is also used to store the IP-address of the server that was connected to last, so that if the player always connects to the same server, they do not have to manually input the address every time.

13.2.5 The duels

Duels are handled by the Activity called *BattleActivity*, and as can be seen in Figure 13 in Chapter 9.5.1, it functions as a loop. It displays a question, then a screen saying whether the answer to the question was correct or wrong, then displays the next question, and so on until all the questions have been answered and the result of the battle is displayed.

Between each question, the client needs to contact the server to give information about what weapon the player wants to use against their opponent, and what weapon is being used against the player.

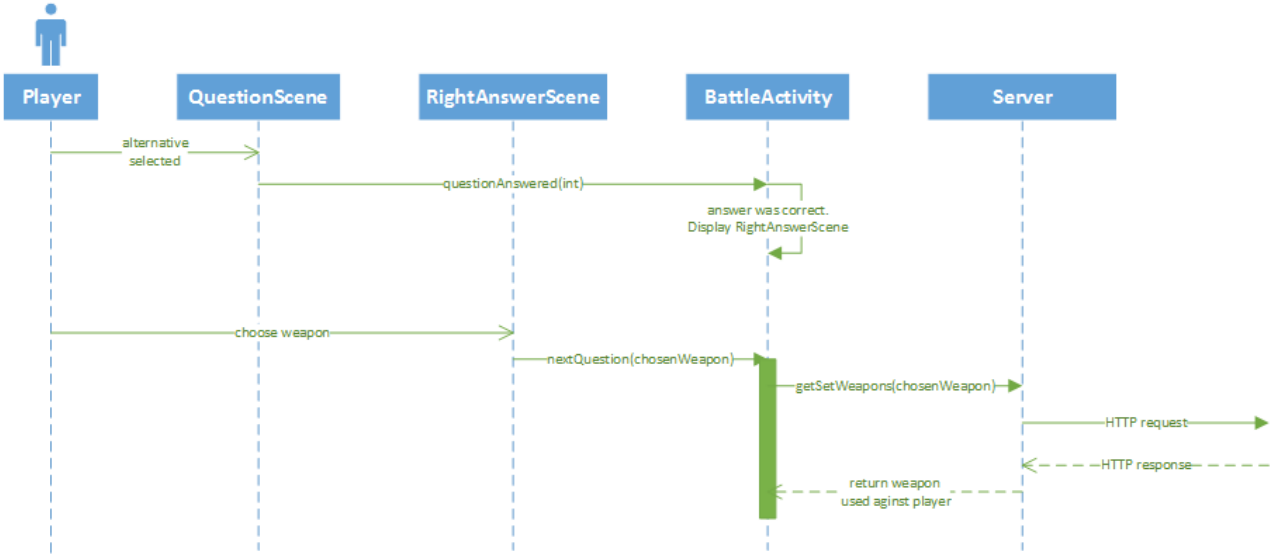

Figure 38: Sequence diagram showing a player answering a question

Figure 38 shows the process of a player answering a question correctly and using a weapon against their opponent.

The way a question is displayed to the player varies greatly depending on what weapon is being used against them. If no weapon is being used, the client displays the standard way to display a question. However, some weapons need an entirely different way to display the question. As an example, the weapon called *High Precision Fan* causes the question to be somewhere else than where the camera starts, so the player has to use their finger to move around on the map in-game and look for the question before they can answer it. Another weapon causes the question to be displayed in the standard way, but simply reduces the amount of time available to answer.

Clearly, the differences between these various ways to display a question are big, yet they still have to accomplish similar things. In order to make the code as simple as possible, and to minimize the amount of identical code, the standard way of displaying a question is done by the class called *BaseDisplayQuestionScene*, and the other specialized scenes are subclasses of this scene.

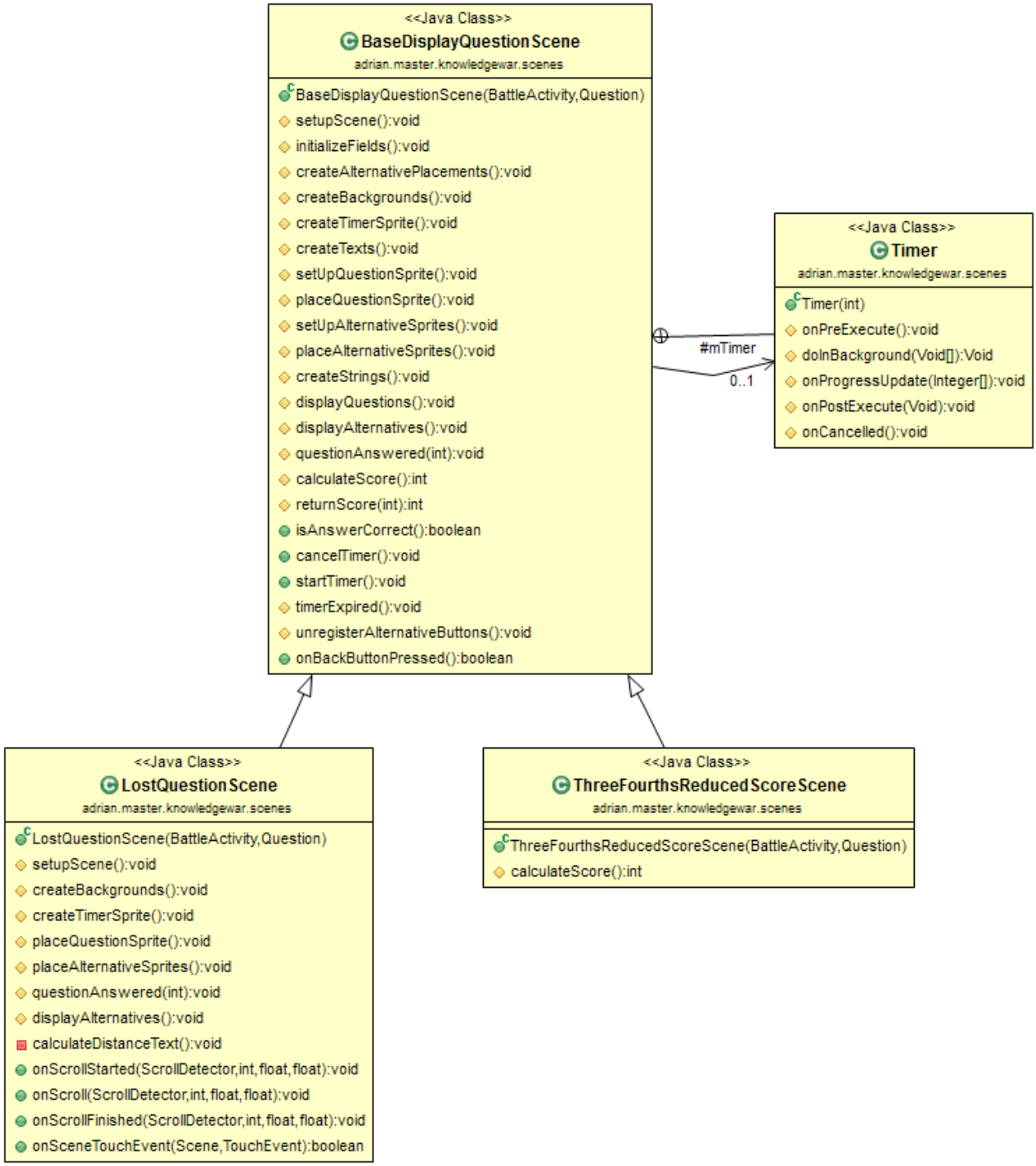

Figure 39: BaseDisplayQuestionScene and two of its subclasses

Figure 39 shows the methods of *BaseDisplayQuestionScene* as well as some of its subclasses. As the figure shows, *BaseDisplayQuestionScene* implements all methods necessary to display a question and its alternatives, as well as having a private class that keeps track of the player's time remaining on the question. Most of the methods are called in the constructor of the object, and sets up the scene the way that is needed:

```
public BaseDisplayQuestionScene(BattleActivity parent, Question question){

	this.mParent = parent;
	this.mQuestion = question;
	initializeFields();
	setupScene();
}

protected void setupScene(){
	createBackgrounds();
	createTimerSprite();
	createTexts();
	setUpQuestionSprite();
	placeQuestionSprite();
	setUpAlternativeSprites();
	createAlternativePlacements();
	placeAlternativeSprites();
	createStrings();
	displayQuestions();
	displayAlternatives();
}
```

Listing 1: BaseDisplayQuestionScene constructor

Listing 1 shows the constructor in *BaseDisplayQuestionScene*. The constructor first sets up two important fields, the *Activity* and the current *Question*. It then calls a method that sets up other fields, such as the different textures, before it sets up the scene itself in *setupScene()*.

The subclasses of *BaseDisplayQuestionScene* all calls the *setupScene(),* but they override the methods necessary to accomplish what it needs to accomplish. One weapon causes the player to receive a lower score if they answer correctly. The scene that is displayed when this weapon is used only needs to override the *calculateScore()*-method, as can be seen in the figure. However, the weapon mentioned earlier that causes the question to be hidden, must override many of the methods in order to set up for example a scrollable map.

BaseDisplayQuestionScene and its subclasses all extend the class *Scene*, so it is not necessary for the Activity to care about what scene is being displayed. Therefore, the Factory Pattern is used to administer the creation of scenes.

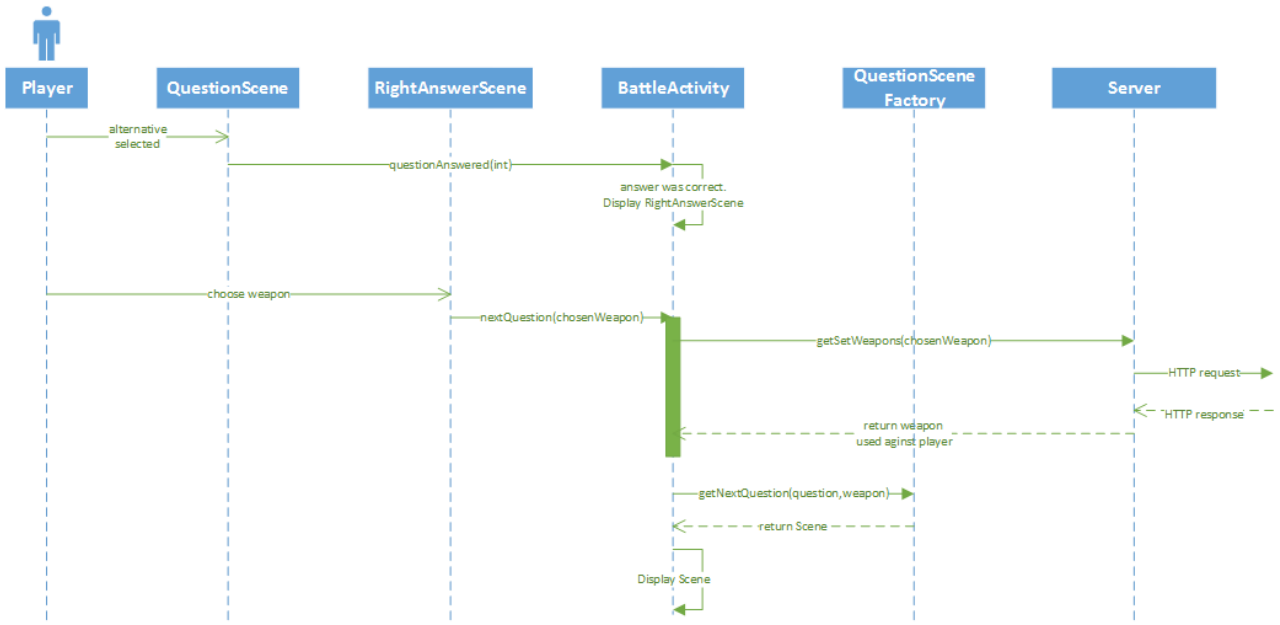

Figure 40: Player answering a question and receiving a negative effect

Figure 40 shows an extended version of Figure 38, that includes how the *Activity* handles the response from the server. The response from the server when using a weapon against the opponent, is a short String that contains information about what weapon is being used. The *Activity* then asks the *QuestionSceneFactory* for the next scene, and the factory returns the appropriate one. The scene is then displayed to the player.

13.2.6 Game Data stored in XML

The are certain aspects of the content in the game that needs to be stored in a permanent manner, and only needs to be read, never altered, to be used by the game. This content includes the names of the various buildings, what they cost to build, what bonuses they give, etc. Similar information needs to be stored for all the weapons. This information could potentially be hardcoded into the game, and stored in a Constants-like class where everything is stored as public *Strings* and accessed that way by the other classes. However, that is not a very elegant solution, and it would make the code bothersome to read and hard to update and expand.

```
<Building>
<Type>B</Type>
<Name>Ninja Wagon</Name>
<Description>Training school for ninjas. Gives
attack-bonuses and the ability to send ninjas on
missions to distract and confuse
enemies.</Description>
<Rank>
<Number>1</Number>
<Value>2000</Value>
<Health>4</Health>
<Global>false</Global>
<GoldBonus>0</GoldBonus>
<DamageBonus>1</DamageBonus>
<HealthBonus>0</HealthBonus>
<GoldPenalty>0</GoldPenalty>
<UpgradeCost>3000</UpgradeCost>
</Rank>
<Rank>
<Number>2</Number>
<Value>5000</Value>
<Health>5</Health>
<Global>false</Global>
<GoldBonus>0</GoldBonus>
<DamageBonus>2</DamageBonus>
<HealthBonus>0</HealthBonus>
<GoldPenalty>0</GoldPenalty>
<UpgradeCost>3000</UpgradeCost>
</Rank>
<Rank>
<Number>3</Number>
<Value>10000</Value>
<Health>6</Health>
<Global>false</Global>
<GoldBonus>0</GoldBonus>
<DamageBonus>4</DamageBonus>
<HealthBonus>0</HealthBonus>
<GoldPenalty>0</GoldPenalty>
<UpgradeCost>-1</UpgradeCost>
</Rank>
</Building>
```

Listing 2 Example from buildings.xml

```
<weapon>
<name>Chicken stampede</name>
<description>
Start a chicken stampede! The opponent has less
time to answer his next question because he has
to get out of the way.
</description>
<attackdescription>
Your opponent has started a chicken stampede!
You have less time to answer the question
because you must get to safety.
</attackdescription>
<class>F</class>
<rank>1</rank>
<cost>100</cost>
</weapon>
```

Listing 3 Example from weapons.xml

Instead, all game data is stored in two xml-files, *buildings.xml* and *weapons.xml,* that are read when the game starts and these are used to create easily accessed Building-objects and Weapon-objects.

Displayed in Listing 2 and Listing 3 are two examples of how information about a building and a weapon is stored in this file.

When the game is started, these two xml-files are parsed, and a list containing Building-objects, and a list containing Weapon-objects are created. These lists are stored in two factory-classes, *BuildingFactory* and *WeaponFactory*, and these classes are used when the game needs an instance of a building or a weapon.

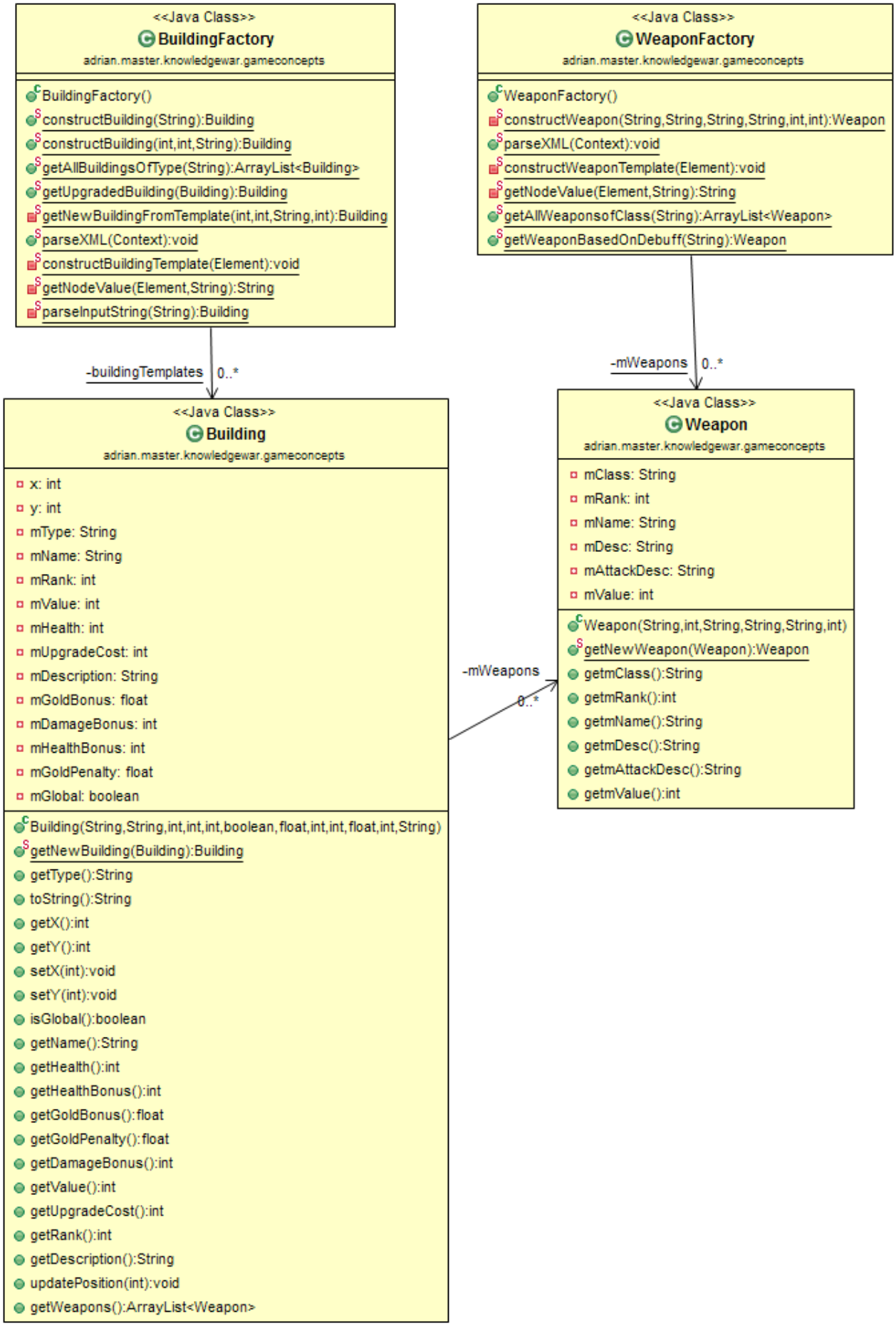

Figure 41: Overview of two factory-classes

Figure 41 shows an overview of the two factory-classes. The two classes keep an *ArrayList* containing templates of all the buildings and weapons respectively. When the game needs an instance of a building, for example when the player is placing new buildings while playing, the game calls the method *BuildingFactory.constructBuilding(String type),* and the *BuildingFactory* creates a copy of the *Building* template-object, and returns this to the game.

Part IV
Evaluation

Chapter 14
User Experiment

In order to evaluate the quality and entertainment value of *Tribal Knowledge War*, a user experiment was run.

14.1 Purpose of experiment

The purpose of the usability experiment is to determine the overall quality and usability of *Tribal Knowledge War*, and also to determine whether or not the plan to create a fun, valuable and educational experience succeeded.

The first part of the questionnaire will be used to determine the general usability of *Tribal Knowledge War*. Having as high usability as possible is important for all applications and systems, because users will prefer to use applications that are easy to learn and use over applications that offer the same functionality, but are more difficult to use. In the case of *Tribal Knowledge War*, a high usability would hopefully mean that users quickly understand how to play the game, and this would hopefully also mean that they would like to play it often.

The second part of the questionnaire will be used to find answers to the research questions in table 1. Having found theoretical solutions to how to make a game both fun and educational, are the attempts at implementing these solutions been successful?

14.2 Description of experiment

A series of smaller experiments were held in the period of May 1st to May 14th, 2013 where the game was made available for small groups of participants that either installed the game on their personal Android device, or borrowed a device with the game already installed, and then played the game against each other.

In total, 7 people participated in groups of 2 or 3 at a time. The participants had very varied backgrounds. Three were graduates in computer science while the other four did not have a technology background. Their previous experience with Android devices were also very varied.

Some had owned devices for a long time, while others did not have one at all, or did just recently get one.

The questions available were mostly IT-related questions, but there were also some questions from other categories, concerning video games, TV shows or other types of popular culture.

14.3 Task List

At the beginning of the experiment, the participants were given a list of tasks they had to complete during the end of the experiment. This was done to make sure that they used all of the most basic functions in the game.

Some of the tasks are a little vague on purpose, to encourage players to experiment with the game, and try to figure out things on their own.

Table 9: User experiment task list

1	Start the application. Register a new account.
2	Enter the “Manage Own Tribe” screen. Build at least one building.
3	Upgrade at least one building.
4	Open the “Map” screen. Challenge another player to a duel.
5	During a duel, use a weapon on the other player.
6	After a duel, check the answer to at least one question.
7	Use the chat to talk to other players.

Table 9 shows the list of tasks the participants were given. The tasks did not have to be completed in order, and the participants were free to do every task more than once. The list only set the minimum amount of things they had to do.

14.4 Questionnaire

After the participants had completed the task list, and felt they had played the game enough, they were given a survey to fill out. The results of the survey will be used to evaluate the usability of the system in accordance with James Brooke's system usability scale (SUS) [31]. SUS is a quick and dirty way to evaluate the usability of a system, and it consists of 10 statements where the respondent marks how strongly they agree or disagree with each statement by using a scale of 1 to 5, where 1 means Strongly Disagree and 5 means Strongly Agree. The results of these questions will be used to calculate an overall usability score for the system.

In addition to the the 10 standard questions of the SUS questionnaire, 10 more questions were added that focused on the specific research questions of this report. These questions are meant to evaluate to what degree the game succeeded in implementing various techniques for making games both educational and fun.

Table 10 and Table 11 show the questions in the questionnaire.

Table 10: SUS statements

	Strongly disagree 1	2	3	4	**Strongly agree** 5
I think I would like to use this system frequently.					
I found the system unnecessarily complex.					
I thought the system was easy to use.					
I think that I would need the support of a technical person to be able to use the system.					
I found the various functions in this system were well integrated.					
I thought there was too much inconsistency in this system.					
I would imagine that most people would learn to use this system very quickly.					
I found the system very cumbersome to use.					
I felt very confident using the system.					
I need to learn a lot of things before I could get going with this system.					

Table 11: Non-SUS statements

	Strongly disagree 1	2	3	4	**Strongly agree** 5
I found this game fun to play.					
I learned something from playing this game.					
I want to learn more about the things I learned.					
I liked the competitive nature of the game.					
I liked playing against other people who are in the same room.					
I liked using “weapons” against my opponent.					
I liked it when my opponent used “weapons” against me.					
I liked the game's setting (desert nomads)					
I found this game engaging.					
I found this game rewarding to play.					

14.5 Questionnaire results

When presenting the results of the 10 first statements in the questionnaire, it is important to note one thing: the results of individual statements are not meaningful on their own [31]. There is a specific way to calculate the result of this questionnaire. First, note that every answer is numbered 1-5. For every odd-numbered statement, subtract 1 from the user response. For every even-numbered statement, subtract the user response from 5. This way, you will end up with a number 0-4 on every statement. Add them together, multiply by 2,5, and you get a single number ranging from 0 to 100. *Tribal Knowledge War* got a SUS-score of 73,9, and that is the only number that is useful in this context. However, he results for individual statements are displayed in Table 12 to give a better view over how the final score was calculated.

Table 12: Results of SUS questionnaire

	SUS contribution
I think I would like to use this system frequently.	**6,8**
I found the system unnecessarily complex.	**5,7**
I thought the system was easy to use.	**7,1**
I think that I would need the support of a technical person to be able to use the system.	**7,9**
I found the various functions in this system were well integrated.	**7,1**
I thought there was too much inconsistency in this system.	**6,4**
I would imagine that most people would learn to use this system very quickly.	**8,6**
I found the system very cumbersome to use.	**7,9**
I felt very confident using the system.	**8,6**
I need to learn a lot of things before I could get going with this system.	**7,9**
	73,9

Table 13 shows the results of the non-SUS statements of the survey. It shows how many of the participants either disagreed, felt neutral, or agreed with a given statement.

Table 13: Result of non-SUS statements

	Disagree	**Neutral**	**Agree**
I found this game fun to play.	14%	0%	86%
I learned something from playing this game.	0%	43%	57%
I want to learn more about the things I learned.	29%	43%	29%
I liked the competitive nature of the game.	0%	0%	100%
I liked playing against other people who are in the same room.	0%	14%	86%
I liked using “weapons” against my opponent.	0%	0%	100%
I liked it when my opponent used “weapons” against me.	71%	0%	29%
I liked the game's setting (desert nomads)	0%	71%	29%
I found this game engaging.	0%	29%	71%
I found this game rewarding to play.	0%	29%	71%

A discussion on this data will follow in chapter 15.4, but one can see here that most players seemed to enjoy the game and found it fun to play. However, many participants did not feel they learned something, and they do not agree that they want to learn more.

14.6 Observations made during experiment

Players seemed to enjoy playing the game, and many participants took actively part in the kind of

friendly competition that *Tribal Knowledge War* tries to encourage: gloating and trash-talking when winning, and whining and complaining when losing. The weapons available to the player contributed to a large part of this, as players who used them often laughed and their opponents often sighed deeply when being used a weapon against.

Some players struggled with some parts of the GUI and finding how to accomplish a certain task. Most of the time they figured it out on their own, but there were some cases where a participant had to ask how to do something.

None of the players bothered spending any time on the *Battle Result*-screen after they had checked an answer to a question once. They mostly just looked at the result of the battle, and then immediately returned to the main menu to play again.

Chapter 15
Evaluation

This chapter will discuss the various choices made during development, such as architecture and choice of framework, and whether these choices negatively impacted the process. It will also discuss and evaluate the result of the user experiment.

15.1 Evaluation of architecture

The decision to make the server stateless so that the clients would have to include all information with every request turned out to have both positive and negative aspects, but the positive outweighed the negative.

The biggest problem with this design was the amount of responsibility it placed on the client, and the amount of extra work required to make sure that the client was taking advantage of the server's stateless nature. The fact that disconnects was a meaningless term in this context because there was no permanent connection, was originally thought to be easy to have work to the developer's advantage, but it was difficult to design both the functionality and the user interface of the client in such a way that it felt intuitive and easy to understand for the user.

As an example, during a duel, if the device lost connection to the WiFi for a couple of seconds, it would not be able to tell the server that it had answered a question. In theory, this is not a problem at all, because the client still has all the necessary information available, it just needs to try to send the message again when WiFi-connection is reestablished. However, how should this be indicated to the user? The client could just try again in the background without telling the user there is a problem, but what if the WiFi cut out completely, and it is therefore impossible to reestablish connection? The client could try in the background for a certain amount of time, and only telling the user if the problem persisted.

In the end, it was decided that the game should assume that loss of WiFi is only temporary, and it tries again and again in the background until it succeeds, and displaying a message on screen that there was a problem contacting the server.

All in all, the decision to make the server stateless was the best and most practical for this particular project, but the game would need additional work to solve some issues before it could be made into a commercial product or similar.

15.2 Evaluation and use of AndEngine

AndEngine was used a the framework for developing the Android application, and this was very good for the project. AndEngine was extremely flexible, and handled all the difficult parts of game-making, such as handling the game loop and making textures, so it was possible to focus on the design and function of the game instead.

However, there were several problems encountered during the development, caused by the rather fast-changing nature of AndEngine, and its lack of proper documentation. The biggest one, and hardest one to find, was one that caused the application to crash at seemingly random times and for random reasons. Quite some time was spent searching different forums to find the reason and how to fix it, and it turned out to be because of the way it handled removing elements from the screen. The details are unclear, but it seems that a race-condition can cause the game to crash unless elements are removed "safely." Code was changed, and the problem disappeared.

Overall, there were very few similar problems, and none of these problems were insurmountable, so using AndEngine was a good decision.

15.3 Evaluation of usability

It is difficult to judge the overall usability of a system with just its SUS-score without any context or systems to compare it with. Bangor, Kortum, Miller [32] has done studies on the effectiveness of SUS as a tool for rating usability, and by evaluating nearly 3500 ratings in 273 studies they have found that the overall mean score for SUS ratings is around 70. *Tribal Knowledge War* got a SUS-score of 73,93, which is slightly above average score.

Bangor, Kortum, Miller also suggests a way to map SUS scores to descriptive adjectives, and *Tribal Knowledge War's* score places it at the lower end of the interval between "good" and "excellent" as description of the overall usability.

15.4 Evaluation of entertainment and educational value

By looking at the second part of the questionnaire one can try to answer the research questions in this thesis. The first three research questions were about how one could combine theories from game design with the concept of an educational game to create an experience that is both fun and educational. During the prestudy, several articles and texts on the subject was found, and the methods suggested in these articles were used in development of *Tribal Knowledge War*. The results of the experiment are somewhat divided.

Most participants felt that the game was fun, with around 85% of the participants either agreeing or agreeing strongly to the statement that the game was fun to play. This is a good result, and suggests that it is a game that many people would enjoy playing. The two main ways that the game tries to encourage players to play the game is emphasizing the competitiveness and making "weapons" fun. Every participant agreed that these aspects of the game were enjoyable, which is a very good result.

However, even if every participant felt that the game was fun, it did not directly translate to any educational value or motivation to learn more. Only 57% of the participants agreed that they learned something by playing the game, and only 28% wanted to learn more after they had finished playing. This is a disappointing result, and it might be a combination of the design of the game and the categories of the available questions. The majority of questions available during the usability test were about general information technology, and that was perhaps not the most suitable set of questions for the participants. Three of the participants have graduated with degrees in Computer Science, and as such probably found the questions very basic. The other four participants had no knowledge of the subject before they started playing, and would perhaps not be interested to learn more because they do not care about the subject at all. Even so, part of the goal of the game was to encourage players to learn, regardless of subject, so the result is not entirely satisfactory.

The purpose of the last three research questions was to find if there was something to gain by making a game pervasive in order to increase players' motivation and desire to play the game. Results from the study shows that participants generally enjoyed the competitiveness of the game and playing against people in the same physical location. This suggests that the pervasiveness increased the enjoyability of the game, but when one looks at these results combined with the game's low score for educational value, it did not sufficiently motivate players to actively learn in order to win against their opponents.

One very interesting result from the second part of the questionnaire is the participants' feelings on using weapons against each other during duels. Every single participant strongly agreed that they liked using weapons against their opponent, but almost everyone also disliked that their opponent used weapons against them. This would mean that the weapons both increased and decreased the players' enjoyment of the game, but considering that most participants agreed that the game was fun overall it probably increased enjoyment much more than it decreased it.

15.5 Feedback from testers after debriefing

Several of the participants had comments during debriefing about their feelings towards the game, and this provided some useful feedback.

Most notably, 3 of the participants were surprised when being told about how the bonuses when attacking were determined, and how one could manipulate the bonuses by choosing where to attack from. This was something they had not realized, and not cared about while playing. This aspect of the game is rather important, so it would be natural for the participants to feel that the game was unnecessarily complex if they felt that this was too difficult to understand. However, this didn't bother them, as they were still enjoying the weapons and the format of the duels. It's hard to directly answer if this impacted how they felt about the game's overall usability, but since *Tribal Knowledge War* got a good SUS-score, it's perhaps possible to explain this by considering the “angle of attack” to be a game-mechanic rather than just functionality. Players often accept that some mechanics are not easy to understand, and perhaps the fact that there was more for them to discover gave the game more depth rather than make it feel overly complex. It would be interesting to arrange more usability tests in order to see if these feelings are common.

For at least one participant, the pervasive nature of the game was a negative thing. They explained that they enjoyed the mechanics of the game, but they would have preferred to be able to play against anyone at any time, and not have to rely on other players being in the same room. They used the game Quiz Battle as an example of a game they enjoy, where the locations of players are

irrelevant.

15.6 Evaluation of observations made during experiment

Many participants had slight problems navigating the menus in the game. The text on buttons were sometimes unclear as to what their function were, and various touch-gestures, for example scrolling the camera when viewing the map, were not very intuitive.

These small problems did not seem to negatively impact the players' enjoyment of the game, and in almost all cases, the players figured out what to do on their own by experimenting. However, this suggests that the GUI would greatly benefit from refinement in order to make it more clear and intuitive.

Another observation made was that most players did not bother to spend any amount of time on the screen that displayed the result of the battle, preferring instead to go immediately back to the main menu and wait for a new challenge or to challenge another player. This almost certainly negatively impacted the educational value of the game, because it is only on the result-screen that players have the ability to check answers to question so they can learn how to answer it correctly next time.

The reason for this could be that the players were not sufficiently motivated to learn more on their own, but it could also be a problem with the way the experiment was conducted. Every group that played against each other consisted of only 2 or 3 people, and perhaps the players felt that they did not want to keep the other players waiting for them to finish checking answers. If the group was big enough that everyone felt certain that everyone else could easily find someone to play with, maybe it would be easier for them to spend some time on the result-screen to check answers and catch their breath.

Part V
Conclusion

Chapter 16
Conclusion

In this thesis, we have studied literature focused on making games educational and fun, and we have used the knowledge from this literature to develop a prototype of a game that should hopefully be both entertaining to play, and educational for its players. After the development of the prototype, a user experiment was conducted to evaluate the quality of the game, and if the goal of making a fun educational game was reached. We are now ready to answer the research questions posed in Chapter 2.1.

In research question 2 and 3 we asked what makes a game fun, and what makes a game an effective learning tool. During the prestudy, good strategies for achieving both these goals were found, and were used during development of *Tribal Knowledge War.* We found that the game had to be challenging, in order for the player to not get bored, and that it should appeal to the player's curiosity and use an intrinsic fantasy.

Research question 1 asked how we can use the knowledge of how to make a game both fun and an effective learning tool to create an enjoyable and educational game. To achieve this, we found a formalism called MDA that made it easy to examine the various *aesthetics* that *Tribal Knowledge War* should aim to achieve. This formalism was used together with the answer to research question 2 and 3 to evaluate the final design of *Tribal Knowledge War* to see if it was a design that achieved what it was meant to do.

In research question 4, we asked: “Does competition make a game more fun?” To answer this question, we found studies that indicated that competition at least motivates a player to play a game, and can also increase a player's motivation to learn from the game. The results from the user experiment on *Tribal Knowledge War* suggests that players did find the competitive aspect of the game to be fun and enjoyable, but that this did not directly lead to them being more interested in learning.

In research question 5 we asked how we could use pervasiveness to increase motivation to learn. Encouraged by studies and literature that suggested that competition increases motivation to learn, we thought to actively use the competitiveness in the pervasiveness of the game. By only being able to challenge other players that are physically close to the player, the competition would hopefully feel more direct and a more important part of the gameplay. This would in turn hopefully result in a stronger motivation to become better at the game than simply challenging strangers over the Internet without ever seeing them or knowing how they are.

Research question 6 asked: “Is it more fun to only challenge players in the vicinity in the physical world?” The answer to this question was meant to provide an answer to whether the attempt at using pervasiveness to motivate learning was successful. Results from the user experiment suggests that it was. A majority of players answered that they enjoyed challenging players in the same room. However, many participants did not agree that they learned something from playing the game, so even if participants enjoyed the pervasiveness, it did not increase their motivation to learn.

Having discussed answers to the research questions, we are ready to discuss whether or not *Tribal Knowledge War* ended up being a both fun and educational game. The feedback received from testers on *Tribal Knowledge War* was very positive, with the majority of players enjoying all aspects of the game, both the gameplay itself and its pervasive nature. However, comments from the participants of the experiments suggests that the game would benefit from a more thorough introduction that explains the in-game mechanics, such as how to manipulate the player's bonuses, better. The game would also benefit from some refinement of the GUI to make it more intuitive.

While players enjoyed the game, results also suggests that the game did not live up to expectations regarding imparting knowledge and motivating further learning. The game in its current state is adequate for entertainment, but would perhaps need additional work to properly facilitate learning.

Chapter 17
Further work

This chapter will discuss in what way it would be natural to continue development of *Tribal Knowledge War*, both from the viewpoint of continuing the current design vision, and making changes in order to enhance the educational value.

17.1 Further development

There are several possibilities for further development of *Tribal Knowledge War*, to make it better suited for use as a learning tool, and more fun and engaging.

The strategic part of the gameplay that consists of constructing different buildings and placing them in the player's tribe can be greatly extended by introducing more penalties, bonuses and weapons granted by buildings. It would require some restructure of code, but creating buildings that give the player a longer time to answer questions would be possibly be a viable idea. New weapons that introduce mini-games similar to the weapons that force the player to navigate around the map looking for the question would also perhaps make the game more fun by introducing more variation to the gameplay.

In order to make the game feel better for the user, creating original artwork to use as sprites and backgrounds should be a high priority before the game is made available to general users. The images used in the prototype is taken from free sources in various places on the Internet, and because of that, the game does not have a very unified artistic direction. Custom artwork would be better able to express the setting of the game, and that would improve the *intrinsic fantasy* for the player.

Although the game got a pretty high SUS-score, the general design of the GUI could use more refining in order to make the flow of the game more intuitive for players. Observing the participants during the test revealed that there were times where they were not sure of what to do and where to click. This did not seem to be a big problem for their enjoyment of the game, but minimizing such problems should always be a priority.

Changes will need to be made in order to make the game better suited to motivate learning. A change that could help this would be to introduce a mechanic that gives rewards in-game every time a player looks for the answer to a question they answered wrong. It would also be beneficial to create a bigger database of questions, with more varied subjects, so that players who do not care about computers could get questions more relevant to their interests.

17.2 More extensive usability tests

In this thesis, we have described a usability test and the result of this test. Although this showed some interesting results, and perhaps gave a small hint of whether the implementation of different design theories was correct, it was a still a very small test with a small number of participants. Making the game available to a bigger audience and gauge their feelings on the game would give a much stronger basis on which to draw conclusions about the game's quality and ability to motivate players to learn.

Bibliography

[1] Zerlkowitz, Dolores. Experimental Models for Validating Technology, *IEEE Computer*, 31(5):23-31. 1998

[2] Robin Hunicke, Marc LeBlanc, Robert Zubek. MDA: A Formal Approach to Game Design and Game Research, *Proceedings of the AAAI Workshop on Challenges in Game AI*. 2004

[3] Malone, Thomas W.. What Makes Things Fun to Learn? Heuristics for Designing Instructional Computer Games. *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, 162-169. 1980

[4] Yu, F. Y. Promoting student learning and development in computer-based cooperative learning, *Proceedings of the International Conference on Computers in Education/International Conference on Computer-Assisted Instruction (ICCE/ICCAI)*. 2000

[5] Kohn, A., *No Contest: the case against competition*, Mariner Books. 1992

[6] Lie-Jie Chang, Jie-Chi Yang, Fu-Yun Yu, Tak-Wai Chan. Development and Evaluation of Multiple Competitive Activities in a Synchronous Quiz Game System. *Innovations in Education and Teaching International 40.1*, 16-26. 2003

[7] Markus Montola. Exploring the Edge of the Magic Circle: Defining Pervasive Games, *DAC 2005 Conference*. 2005

[8] Alf Inge Wang, Hong Guo, Meng Zhu, Are Sæterbø Akselsen, Kenneth Kristiansen. Survey on Attitude Towards Pervasive Games, *Games Innovations Conference (ICE-GIC), 2010 International IEEE Consumer Electronics Society's*. 2010

[9] Sondre Wigmostad Bjerkhaug, Runar Os Mathisen, Lawrence Alexander Valtola. The Amazing City Game, *Master Thesis, NTNU*. 2011

[10] Relentless Software. *http://www.relentless.co.uk/home/*

[11] Alf Inge Wang, Terje Øfsdahl, Ole Kristian Mørch-Storstein. Lecture Quiz - A Mobile Game Concept for Lectures, *Proceedings of the 11th IASTED International Conference on Software Engineering and Application (SEA'07)*. 2007

[12] Tetris. *http://en.wikipedia.org/wiki/Tetris*

[13] Super Puzzle Fighter 2 Turbo. *http://en.wikipedia.org/wiki/Super_Puzzle_Fighter_II_Turbo*

[14] Pokemon official website. *http://www.pokemon.com/us*

[15] Sveinung Kval Bakken. KnowledgeWar - Implementation and Evaluation of a Face-to-Face Mobile Knowledge War Game, *Master Thesis, NTNU*. 2010

[16] Gartner Says Asia/Pacific Led Worldwide Mobile Phone Sales to Growth in First Quarter of 2013. *http://www.gartner.com/newsroom/id/2482816*

[17] Lie Luo. Native or Web Application? How best to deliver content and services to your audiences over the mobile phone, *GIA Industry White Paper 2*. 2010

[18] Paper.js — The Swiss Army Knife of Vector Graphics Scripting. *http://paperjs.org*

[19] LimeJS: A HTML5 game framework for building fast, native-experience games for all modern touchscreens and desktop browsers.. *http://www.limejs.com*

[20] Crafty. *http://craftyjs.com*

[21] Unity 3D - a high quality framework for 3D games. *http://unity3d.com/*

[22] AndEngine. *http://www.andengine.org*

[23] Google App Engine. *https://developers.google.com/appengine/*

[24] The Apache Software Foundation. *http://tomcat.apache.org/*

[25] Jetty - Web server and java servlet container. *http://www.eclipse.org/jetty/*

[26] NodeJS. *http://nodejs.org*

[27] Android Developers: Location Strategies. *http://developer.android.com/guide/topics/location/strategies.html*

[28] Android Developers: Google Cloud Messaging for Android. *http://developer.android.com/google/gcm/index.html*

[29] Risk board game. *http://en.wikipedia.org/wiki/Risk_(game)*

[30] Sid Meier's Civilization. *http://www.civilization.com/*
[31] John Brooke. SUS - A quick and dirty usability scale, *Usability evaluation in industry 189*. 1996
[32] Bangor, Kortum, Miller. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability studies*, 114-123. 2009

Part VI
Appendix

Appendix A:
User Experiment Raw Data

Knowledge War user experiment raw data

	Participant A	Participant B	Participant C	Participant D	Participant E	Participant F	Participant G
SUS1	3	3	1	3	3	3	3
SUS2	2	3	3	2	2	3	1
SUS3	3	3	4	3	2	3	2
SUS4	4	3	3	3	3	4	2
SUS5	3	3	3	3	3	3	2
SUS6	1	3	3	3	3	1	4
SUS7	3	3	4	3	4	3	4
SUS8	2	4	4	2	3	4	3
SUS9	3	4	3	3	4	4	3
SUS10	2	4	4	2	4	4	2
NonSUS1	4	4	1	4	4	4	3
NonSUS2	2	4	4	4	3	2	2
NonSUS3	2	3	1	2	2	3	1
NonSUS4	3	4	4	3	4	4	4
NonSUS5	4	4	2	4	3	4	4
NonSUS6	4	4	4	4	4	4	4
NonSUS7	1	0	1	0	4	1	4
NonSUS8	2	2	4	2	4	2	2
NonSUS9	3	3	2	3	3	3	2
NonSUS10	3	4	2	4	4	3	2