LAUREA
AMMATTIKORKEAKOULU
Uuden edellä

Creating a mobile MMO game

Mikkilä, Miso

2014 Leppävaara

Laurea University of Applied Sciences
Leppävaara

Creating a mobile MMO game

Miso Mikkilä
Business Information Technology
Bachelor's Thesis
September, 2014

Laurea University of Applied Sciences
Laurea Leppävaara
Business Information Technology

Abstract

Miso Mikkilä

Creating a mobile MMO game

Year 2014 Pages 57

The term MMO stands for "Massively Multiplayer Online" games and they are real-time multiplayer games. The game type has become popular on PCs in the last ten years. The biggest games can have thousands of simultaneous players and millions of registered players. The cost of creating the games has grown to match Hollywood movie budgets. There are many reasons for the rising costs. The games have become bigger, players demand more and better content, the 3D-graphics have become almost photorealistic and games are developed for multiple platforms. Mobile MMO games exist but none have clearly established themselves like the top MMO games on PCs.

Is it possible for a small team to create a mobile MMO game without financing? The goal of the thesis is to find out what makes MMO game development so expensive and to create a simple proof-of-concept mobile MMO game with minimal resources. It aims to identify the necessary design-choices that will make it possible for a 1-3 person team to develop a mobile MMO.

Analysis of MMO game development showed that content was the most expensive part of game development. In order to keep the cost of the content low, design choices needed to be made throughout the whole game design process: graphics style, animations, sounds and the game type. Other factors that directly affected the cost were the technology choices, scalability, architecture and the hosting platform.

Cost was the main factor when making the design-choices for the proof-of-concept game. The graphics will be 2D based instead of 3D. The gameplay will be tactics-based instead of action-based. The game world will be room-based and have room-based graphics. Only free, open source technologies with good community support were used. Scalability was only considered when choosing the technology but not yet in the development of the proof-of-concept game.

By adhering to the above decisions a working base for a proof-of-concept game could be created in about 2 months. Based on the work it was estimated that the game base could be expanded to a production-ready MMO game release for two platforms (Android and HTML5) with a total workload of 6 months.

Keywords MMO, mobile game, real-time, online game, game design, game development, multiplayer, cross-platform, cloud platform, game graphics

Laurea-ammattikorkeakoulu **Tiivistelmä**
Leppävaara
Business Information Technology

Miso Mikkilä

MMO-mobiilipelin suunnittelu ja kehittäminen

Vuosi 2014 Sivumäärä 57

Lyhenne MMO tulee sanoista "Massively Multiplayer Online" ja sillä tarkoitetaan reaaliaikaisia monen pelaajan pelejä. Pelityyppi on noussut suosioon PC-alustalla viimeisen 10 vuoden aikana. Suurimmissa peleissä voi olla tuhansia yhtäaikaisia pelaajia ja miljoonia rekisteröityneitä pelaajia. Pelien kehityksen hinnat ovat kasvaneet Hollywood-elokuvien tuotantohintojen kokoluokkaan. Nouseviin hintoihin on useita syitä. Pelit ovat kasvaneet, pelaajat vaativat enemmän ja parempaa sisältöä, pelien 3D-grafiikat ovat muuttuneet lähes fotorealistisiksi ja pelejä tehdään usealle alustalle. Mobiileja MMO-pelejä on olemassa, mutta ne eivät ole tehneet läpimurtoa suurimpien PC-puolen MMO pelien tapaan.

Onko pienen tiimin mahdollista luoda MMO-peli ilman rahoitusta? Tämän opinnäytetyön tarkoitus on selvittää mikä MMO-pelien kehityksessä on kallista ja kehittää yksinkertainen MMO-mobiilipelin prototyyppin minimaalisilla resursseilla. Tavoitteena on tunnistaa valinnat, jotka mahdollistavat MMO-pelin kehityksen mobiilialustoille 1-3 hengen tiimillä.

MMO-pelikehityksen analyysi osoitti, että sisältö on kehitystyön kallein osa-alue. Jotta sisällön hinnan saa pidettyä alhaisena, joudutaan tekemään valintoja pelin suunnittelun jokaisella osa-alueella: grafiikka-tyyleissä, animaatioissa, äänissä ja pelin tyylissä. Muita suoraan hintaan vaikuttavia seikkoja ovat teknologiavalinnat, skaalautuvuus, arkkitehtuuri ja käytetty palvelinalusta.

Hinta oli tärkein valintoja ohjannut kriteeri prototyyppipelin suunnittelussa. Grafiikat ovat 2D eikä 3D. Pelaaminen on stategia- eikä toimintapohjaista. Pelimaailma perustuu huoneisiin ja huonekohtaiseen grafiikkaan. Vain ilmaisia, avoimeen lähdekoodiin perustuvia teknologioita hyvällä yhteisötuella valittiin käytettäväksi. Skaalautuvuus huomioitiin ainoastaan teknologiavalinnoissa, mutta ei prototyyppipelin kehityksessä.

Edellämainituilla valinnoilla pystyttiin luomaan pelin prototyyppi noin kahdessa kuukaudessa. Työn perusteella arvioitiin, että tuotantovalmis mobiili MMO-peli voitaisiin kehittää ja julkaista kahdelle alustalle (Android ja HTML5) yhteensä noin 6 kuukauden työllä.

Asiasanat MMO, mobiilipeli, reaaliaikainen, online-peli, pelisuunnittelu, pelikehitys, moninpelattava, monialusta, pilvialusta, peligrafiikka

Table of contents

1 Introduction 7
1.1 Subject and objective 7
1.2 Background 7
1.2.1 MMO games 7
1.2.2 Popular MMO games 8
1.2.3 Technology challenges 8
1.2.4 MMO game costs 9
1.3 Motivation and objectives 10
1.4 Requirements 11
1.4.1 General requirements 11
1.4.2 Cost 11
1.4.3 Technology requirements 11
1.5 Research approach 12
1.6 Approvals 13
1.7 Definitions 14
2 MMO game design analysis 15
2.1 Cost structure 15
2.2 Game design 17
2.3 Game content 20
2.4 MMO game type 22
2.5 Architecture 22
2.5.1 Networking 22
2.5.2 Server architecture 24
2.5.3 Client architecture 25
2.6 Technology 26
2.6.1 Client-side technologies 26
2.6.2 Server-side technologies 27
2.6.3 Servers and hosting 28
2.7 Implementation choices 29
2.7.1 Game design 29
2.7.2 Technology stack 29
2.7.3 LibGDX 30
2.7.4 SockJS client 31
2.7.5 Vert.x 31
2.7.6 OpenShift PAAS cloud platform 31
3 Implementation 32
3.1 Architecture 32
3.2 The game server 32

3.3 Server design 33
3.3.1 Incoming connections 34
3.3.2 Game commands 35
3.4 Client 36
3.4.1 Connecting the client 36
3.4.2 User interface 37
4 Creating a working game demo 38
4.1 Creating a game area 38
4.2 Creating rooms 38
4.3 Creating players 40
4.4 Creating NPCs 41
4.5 Testing the game 43
4.5.1 Log-in 43
4.5.2 Entering the game 44
4.5.3 Moving around the game area 44
4.5.4 Encountering NPCs 45
4.5.5 Multiple online players 46
4.5.6 Test verdict 47
5 Research contributions 48
5.1 Applicability of the results 48
5.2 Reusability of intruduced solutions 48
5.3 How could the result be improved even further 49
6 Conclusion 50
6.1 Does the game implementation fullfill the original requirements? 50
6.2 Summary 51
References 52
Images 56
Tables 57

1 Introduction

This document is a bachelor's thesis written for the degree programme in Business Information Technology at Laurea University of Applied Sciences.

1.1 Subject and objective

The subject of the thesis is "Creating a mobile MMO game". The goal of the thesis is to identify what typically makes MMO game development expensive and create a simple proof-of-concept MMO game using designs and technologies that enable the game development at a low cost. It should result in a game-prototype that can be expanded further to cost-effectively create and publish a mobile cross-platform MMO game.

1.2 Background

1.2.1 MMO games

MMO games are games where many players play in a shared virtual world. The virtual world is open and already exists when a player logs in and continues to exist when a players logs out of the game. MMO comes from "Massive Multiplayer Online". MMO games support large amounts of simultaneous (at least hundreds but mostly thousands) players and are played on the internet. They allow the players to interact with each other in the same virtual game-world.

MMO games are real-time multiplayer games, although not all real-time multiplayer games are MMO games. For example realtime multiplayer racing, arcade and first-person-shooter games are not MMO games. Those games "exist" for the time it's played as opposed to MMO games where the game-world exists before players log in and continues to exist after they log off. MMO games are all real-time multiplayer games and the problems and design issues that apply to MMO also apply to smaller real-time multiplayer games (in smaller scale). It is debatable if a MMO game with "only" 10-100 players can be called a MMO game. For simplicity this thesis will use "MMO" for all games that fit into the category – be it a smaller single-server MMO game or a huge MMO game like "World of Warcraft". Ultimately, most smaller multiplayer games want to grow their game-world and player-base to become a "massive multiplayer online" game.

MMORPG type of games combines MMO games and the popular RPG (Role Playing Game) genre. Most MMO games are role playing games (RPG) and therefore actually "MMORPG" games but they are mostly called by the shorter "MMO" form.

Webopedia defines MMORPG this way:

”MMORPGs are online role-playing multiplayer games which allow thousands of gamers to play in the game's evolving virtual world at the same time via the Internet.” (Webopedia)

Key aspects of MMO games are that there are a large number of players that play in the same, shared, virtual world and they interact with each other. Depending on the game they could be playing with each other or against each other.

This thesis tries to produce a proof-of-concept mobile multiplayer game. On mobile devices the MMO games have not established themselves like the games on desktops. According to Peterson (2014) the largest potential of MMO games is mobile. He argues that there are hundreds of millions of tablets and over a billion smartphones and that the number is still climbing.

1.2.2 Popular MMO games

MMORPG were popularized in the 90:s by Ultima Online and a few years later by World of Warcraft. Currently (in 2014) there are multiple popular MMO games for PCs that have millions of registered players. These include (in no specific order) titles such as World of Warcraft, Star Wars – The Old Republic, Final Fantasy (at least XI and XIV), League of Legends, Neverwinter, World of Tanks and Elder Scrolls Online – just to name a few. In an article from gameindustry.biz Matthew Handrahan (2014) quotes research (Superdata Res. 2014) numbers which show that the most popular subscription-based MMO in 2013 was the game “World of Warcraft” with a 36% market share and a worldwide revenue of 1041 million US dollars.

The number of players is not the only big number about the biggest MMO games. The MMO game “World of Warcraft” was reported (The Game Reviews 2009) to have 5.5 million lines of code, to be running 13250 servers with a total of 75000 CPUs and to use 113 Terabytes of RAM. To achieve those numbers and keep the gaming experience real-time, the backend needs to have a special architecture.

1.2.3 Technology challenges

For software and game developers, MMO games have been a popular subject since the beginning. One of the reasons is likely the fact that creating a MMO game has always been considered to be a difficult task from the technology-perspective. Developing MMO games requires expertise in many areas – from the client to the server and the network connecting them.

An article about MMO game database guidelines writes “Developing an MMOG server requires expertise with client/server architecture, network protocols, security, and database design” (Lee 2003). The article continues to describe the challenges: “The server must be able to handle and verify a large number of connections, prevent cheating, and apply changes (bug fixes or added content) to the game. A system for recording the game's data at regular intervals, without stopping the game, is also important.” These listed requirements are still valid but the games have grown from 2003 (which were mostly PC games) to a number of new devices and platforms: smartphones, tablets, game-consoles, web browsers. They are bigger and have better graphics.

Compared to single-player games there are multiple aspects in MMO games that make the technology challenging:

1. Real-time communication. MMO games require real-time communication over the internet between the game server and the player’s client-applications.
2. Game-engines must scale to large numbers of simultaneous players (possibly thousands) without slowing down.
3. Distributed system. Traditional games only consist of one game process. MMO games contain (possibly) thousands of client processes that are connected to the game-server(s) which also handle multiple processes.
4. Traffic. Game-messages must be relayed from thousands of clients to thousands of clients. The network traffic must have a certain design and be optimized in order to not congest the network.
5. Security. Single-player games do not need security but a MMO game has interfaces open to the internet. The servers communicate and store player information which can contain sensitive (payments, game assets, login info etc.) information.

Creating a MMO game for mobile devices adds one more difficulty because the internet-connections over mobile networks are almost guaranteed to occasionally loose connections and reconnect at varying speeds and latencies. When mobile devices roam they might at one time be connected to a fast 4G network and a few minutes later the connection might have dropped to a 2.5G network. In-between the connection might have dropped for a few seconds while a new connection was negotiated by the device.

1.2.4 MMO game costs

Nowadays the biggest MMORPG games have budgets similar to Hollywood movie productions. According to Peterson (2014) the development budgets have become “so immense” that they can “become dangerous”. As an example he mentions the game-company “38 Studios” that received a $75 million loan (Gamespot 2014) for game development but “crashed and burned”

in 2012 while developing project Copernicus, a MMO game. Another example of the high MMO game costs was “Star Wars – The Old Republic” that was released in December 2011 and reportedly cost $200 million to develop (Fritz & Pham 2012). The Elder Scrolls Online, launched in April 2014, was rumored to have already exceeded the $200 million cost in January 2014 before it was released (Sharkey 2014).

In an article on www.GameBreaker.tv Lucas Jordan wonders if you can even develop MMO games for less than 1 million dollars. Jordan writes that MMO development is a “massive money sink” and that it can be “incredibly destructive” if the game is not successful. He thinks that it’s a very risky business and places a massive amount of stress on developers and publishers (Jordan 2013).

He looks at a new MMORPG project on crowd-funding website Kickstarter which is trying to raise 750.000$ for game development and is skeptical about the project succeeding. He “would like to see the game succeeding” but concludes: ”Unfortunately, the $750,000 mark just doesn’t seem realistic and I’m concerned they’ll run out of money long before development is complete, but I truly hope they prove me horribly wrong” (Jordan 2013).

The initial game development is not the only cost for multiplayer online games. After they have been released they need to be maintained and developed. If they get a lot of players they need to be scaled up to multiple servers. Even though World of Warcraft was already released in 2004 when game budgets were considerably smaller than games today, Blizzard - the company behind the game, disclosed that the four-year upkeep (without development costs) on the MMO game was $200 million (Plunkett 2008).

1.3 Motivation and objectives

The motivation for this thesis is to research the technologies and build a base on which a 1-3 person team can build a multiplayer game for mobile platforms. The results will be used by a small startup company to launch a simple medieval-fantasy aspired multiplayer indie-game without initial funding. The game is planned to be created using sweat equity and minimal financial resources.

This thesis will weigh the requirements (and realities!) of creating a low-budget mobile/cross-platform MMO game and create a prototype game with a design and architecture that fits the predefined requirements best.

The thesis will produce:
1. A design with documented design decisions for a mobile MMORPG game.
2. A proof-of-concept prototype of a multiplayer MMORPG game.

The game-content (game areas, monsters, quests, graphics, sound effects, music etc.) is not in the scope of the thesis – only creating the game architecture and a simple game-prototype to validate the functionality of the created framework. At best the prototype created for this thesis can be used with only minimal changes as the base for a future multiplayer game that could launch at the end of 2014. Hopefully also other small game-companies and indie game-developers will benefit from the research done for this thesis.

1.4 Requirements

This section lists the predefined requirements for the multiplayer game for which this thesis will be created. Some of the requirements come from the startup company that will publish the multiplayer game, some are created by technical choices, some are just ”common sense” but the main requirement will be cost and resources.

1.4.1 General requirements

The game should be real-time multiplayer game where game content can be added incrementally without technical expertise. The game architecture should perform well enough to handle over 100 simultaneous players on one server instance and it should be fast enough so that games implemented on it are perceived as good enough by modern usability (or playability) standards.

1.4.2 Cost

The primary requirement for the multiplayer game is that the cost for the content, for hosting and for the development should be minimal. It should be possible to attain all graphical, audio, animation etc. content for less than 1000€ and the game should run with initial server-costs less than 100e per month after the game has been releases. The main resources available for the game creation are sweat-equity mostly in the form of software development.

1.4.3 Technology requirements

The game should use a cross-platform framework that targets at least Android mobile devices (both smartphone and tablet) and modern web browsers (with HTML5 support). All other supported platforms (iOS, Windows Phone, Windows 8, desktops) are considered a bonus but not necessary. The programming languages that can be used in this project should be preferably Java, secondary JavaScript and only as a third choice other programming languages.

1.5 Research approach

The research approach for this thesis is constructive research which is a widely used research approach in software engineering and computer science.

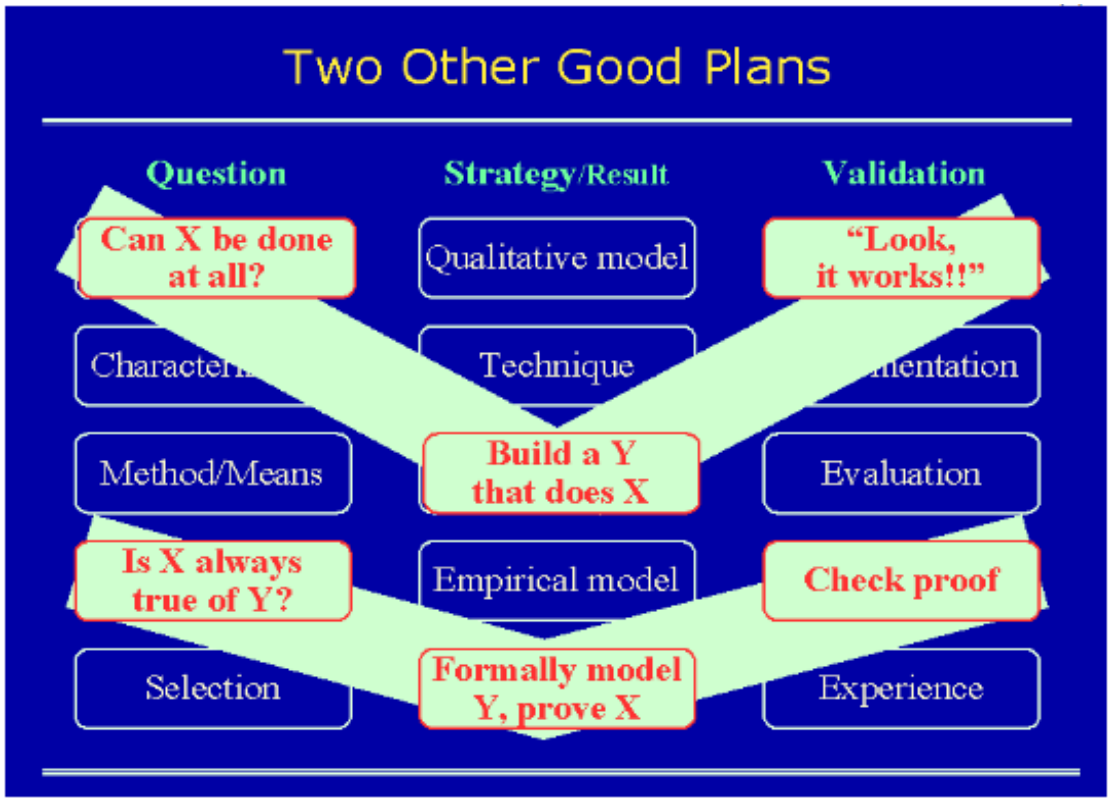

Illustration 1: Two plans for constructive research

In the book "Model-Based Reasoning in Science and Technology" there is a chapter on constructive research and info-computational knowledge generation. It describes the idea of the constructive research method like this:

"The idea of Constructive Research, is the construction, based on the existing knowledge used in novel ways, with possibly adding a few missing links. The construction proceeds through design thinking that makes projection into the future envisaged solution (theory, artifact) and fills conceptual and other knowledge gaps by purposefully tailored building blocks to support the whole construction. Artifacts such as models, diagrams, designs, plans, organization charts, system designs, algorithms and artificial languages and software development methods are typical constructs used in research and engineering. Constructivist solutions are designed and developed and not in the first place discovered,[...]" (Magnani, Carnielli, Pizzi, 2010, 360).

A typical 6-part breakdown of the constructive research process as defined by Mäntylä (2001):

1. Find a practically relevant problem
2. Obtain an understanding of the topic and the problem
3. Innovate, i.e., construct a solution idea
4. Demonstrate that the solution works
5. Show theoretical connections and research contribution
6. Examine the scope of applicability

Using the above 6-part break-down for this thesis:

1. The problem: How can you create a mobile MMO game with limited resources?
2. Examine existing literature, articles, discussions and tutorials about MMO game design, development and technologies.
3. Design and implement mobile MMO game base on the achieved knowledge.
4. Create some simple content to form a proof-of-concept game-demo and test it.
5. How does the result compare with solutions? Has our design choices solved problems that others have had? How could it be improved further?
6. Compare the working demo against the original requirements. Is it good enough?

Initially the thesis sets out to analyze what information is available about developing multiplayer games, identify technologies, weight different design-choices and evaluate technologies that can be used. This is done by reviewing literature, expert presentations and software development forums and multiplayer-game websites. The analysis should result in a variety of technologies, tools and design choices that can be used to create the game. After implementing a core game-architecture and game engine a prototype-game will be created on top of the existing implementation. After that the game-prototype can be used to validate if the implementation adheres the pre-determined requirements.

The main construction of the project-based thesis will be the implementation a MMO game-engine and a working demo. This document will contain the design choices that have been made during the construction.

1.6 Approvals

Prior to publishing this thesis written agreements have been made with the startup-company with agreement on the following issues:

- Required paperwork for creating a bachelors thesis for a company.
- Agreement that source code and information included in in the thesis is public and can be reused without any restrictions.

1.7 Definitions

MMO - Massively Multiplayer Online.

MMOG - Massively Multiplayer Online Game.

RPG - Role Playing Game.

MMORPG - Massively Multiplayer Online Role Playing Game.

RTS - Real Time Strategy (game).

NPC - None Player Character. A computer controlled game-character.

AI - Artificial Intelligence.

HUD - Heads-up-display. The information shown on top of the game screen with the players portrait, hit points, etc. The term comes from the fighter-planes that have information projected directly onto the cockpits glass so that the pilots do not need to lower their gaze to see the displayed information.

F2P - Free to play. A game that is free to play but has usually other in-game monetization methods.

API - Application Programming Interface.

RIA - Rich Internet Application. Applications that work in the browser but are characteristically very much like traditionally desktop applications. RIA applications function inside the same page by dynamically updating the page instead of reloading complete HTML pages.

GWT - Google Web Toolkit. A framework for creating RIA applications. GWT allows the applications to be coded in Java and be compiled to optimized JavaScript.

WebSocket - A technology introduced in HTML version 5 which allow web applications (clients) to establish a socket-like permanent bi-directional connection to the server.

Zone - A geographical area in a multiplayer game is often called a zone. For example a city, a forest etc.

PAAS - Platform as a service. A cloud-platform that already has a server-framework in place so that usually only the source-code or binaries need to be deployed.

IAAS - Infrastructure as a service. A cloud-platform that only has a server + operating system installed where the user can decide to install any software.

RDBMS - Relational Database Management System.

2 MMO game design analysis

MMO game architecture has been a popular subject from the beginning. There is ample research and literature in software engineering on how to design MMO games designs at a high-level design level but there is not much about the actual technologies or the architecture at a lower level.

Even though this thesis is about creating a mobile MMO the literature review will be conducted mostly on PC-based MMO games. This is simply for the reason that PC-based based MMO games are the considered the “traditional” MMO games and on mobile platforms MMO games have not clearly established themselves yet. There is a lot of information available on the PC-based MMO games. That will be used as a guideline and at the end in the “design choices”-section the thesis will try to summarize what it will take to make everything fit into the original requirements of creating a mobile MMO while keeping the cost low.

2.1 Cost structure

The average budget of commercial MMO game launches are in millions of euros. In order to drive down the costs for developing and launching a MMO game you first need to know what the actual costs consist of. This section tries to break down the total cost to see what it actually consists of.

Mark Kern from mmorpg.com says that in modern MMO games the game content is what is most expensive and that zone creation is about 70% of an MMO’s total cost (Kern 2014). He argues that “After all, you are talking about detailing something like 5-10 square miles of land (or more). Everything has to be placed by hand: mountains, roads, caves, dungeons, valleys, trees, rocks, points of interest, themed/unique monsters and even grass!”

Kern also details other areas of cost like the features, programming, art and world design. Other areas that affect the total costs are 2D art, animations, user interfaces, client/server engineering, database architecture, and the network infrastructure (Jordan 2013) . In addition to these there are other, often smaller expenses worth pointing out like the story or script, music tracks, sound effects, technology licenses, and server hardware.

The game graphics is usually the biggest part of the content. The costs for game graphics is an area that can be broken down into multiple parts. Graphics can be used in so many different ways and levels. The process for creating 3D graphics is described like this:

”The front-end (or client) component of a commercial, modern MMORPG features 3D graphics. As with other modern 3D games, the front-end requires expertise with implementing

3D engines, real-time shader techniques and physics simulation. The actual visual content (areas, creatures, characters, weapons, spaceships and so forth) is developed by artists who typically begin with two-dimensional concept art, and later convert these concepts into animated 3D scenes, models and texture maps." (Luna 2006)

In an article called "The Spiraling Cost of MMOs" Jeff Francis writes about the ever-rising costs for MMO games:
"In the past, things were a great deal simpler, and, as such, you only needed a small team or an individual to create a game. Modern technology and its resident issues dictates a much larger development team than the past. The biggest factor in the cost of developing an online game is manpower. The more people you have to hire, the more expensive it will be. Gamers are another reason why MMO costs continue to climb. As technology has improved, we gamers demand more from our games. We want better graphics, full orchestra soundtracks, and voice acting in our MMO games. All of this costs money to fashion as game companies have to hire lots of people to work years to create the games. Factor in the wide range of video and sound cards that the game has to comply with, then the cost continues to rise." (Francis 2014)

He sees the amount of platforms as another factor for the rising costs: "In the past, game companies focused on crafting games for the PC, but that has changed. Companies look to port their games not only to the Mac, but also the various consoles, such as the Xbox One and PlayStation 4. All of this raises the development cost of the game as it has to be adapted to the new platforms. Smaller online games are also porting themselves to smartphones and tablets. All this means that code has to be rewritten and that graphics and sounds need to be adapted as well. Naturally enough, this all costs money." (Francis 2014)

A summary of the costs of developing a modern MMO game therefore include development, acquisition or work on the following parts:

- Graphics
- 2D-art
- 3D-modelling
- Physics engine
- Animations
- Sound
- Music (sound-tracks)
- Sound effects
- Font-end programming
- User interface
- 3D-engine programming

- Server/network programming
- Game engine
- Client-server communication
- Database
- Game content
- Monsters, items, weapons, characters, skills, spells
- Zone creation
- Stories and scripts
- Servers and hosting (bandwith etc.)
- Technology licenses
- Support for multiple platforms

The above only lists costs identified for the development of the game. The costs occurring after the development like maintenance, marketing, sales and hosting are not included.

On the other side, Blizzard has shown that even after 10 years (and beyond it's prime) a MMO game can still be a billion dollar business (Superdata Res. 2014).

2.2 Game design

Image 1: Screenshot from 'Age of Conan MMORPG game

The standard for PC-based MMORPG games these days is to create almost photo-realistic 3D worlds. Those require detailed 3d-models for everything from a single straw of grass to the mountains and cities. Illustration 2 shows a screenshot from the game "Age of Conan" (2014).

There are however other ways to implement graphics. A 2D game is simpler and faster to implement and only requires a fraction of the graphical assets compared to a 3D game. There are tile-based 2D games (which are at the moment popular in mobile MMORPG games) and there are room-based 2D games. Many tile-based games use a view from upper left or right to make it look more 3D-like instead of completely flat tiles. Illustration 3 showing a tile-based mobile game “TibiaMe MMO” from Google Play store, a game which has 500k-1M downloads at the time of writing.

Image 2: A tile-based game 'TibiaMe MMO' for Android

An example of a room-based 2D-game is the mobile game “Avatar Fight” from Google Play store, which has 1M-5M downloads at the time of writing.

Image 3: Room-based 2D-game 'Avatar Fight'

The screenshots of the room-based games might look similar to 3D-games but the rooms have not been dynamically generated – they are only static 2D-images (they can use animations) that have been put in front of a background image.

Nearly all 3D-games and many 2D games also use a physics engine which will simulate gravity to make the animations look more real and do things like collision detection of game objects etc. These are only required in the games that include an arcade/action game element. A MMO can be created like a real-time strategy (RTS) game without the arcade game elements and without a physics engine.

Animations are used in both 3D and 2D games. They can be made to be very simple using only 2-3 static images or they can be complicated real-time-calculated 3D-model animations. Animations can be left out of the game altogether. If they are left out altogether then the graphics could be as simple as showing a static image in different rooms in the game with the details described in text.

It is possible to create multiplayer games without graphics. The oldest ancestors of today's MMORPG games where multiuser dungeon games (”MUD”) that were completely text-based. They usually ran on Unix-based mainframe computers and were played using telnet and even gopher connections.

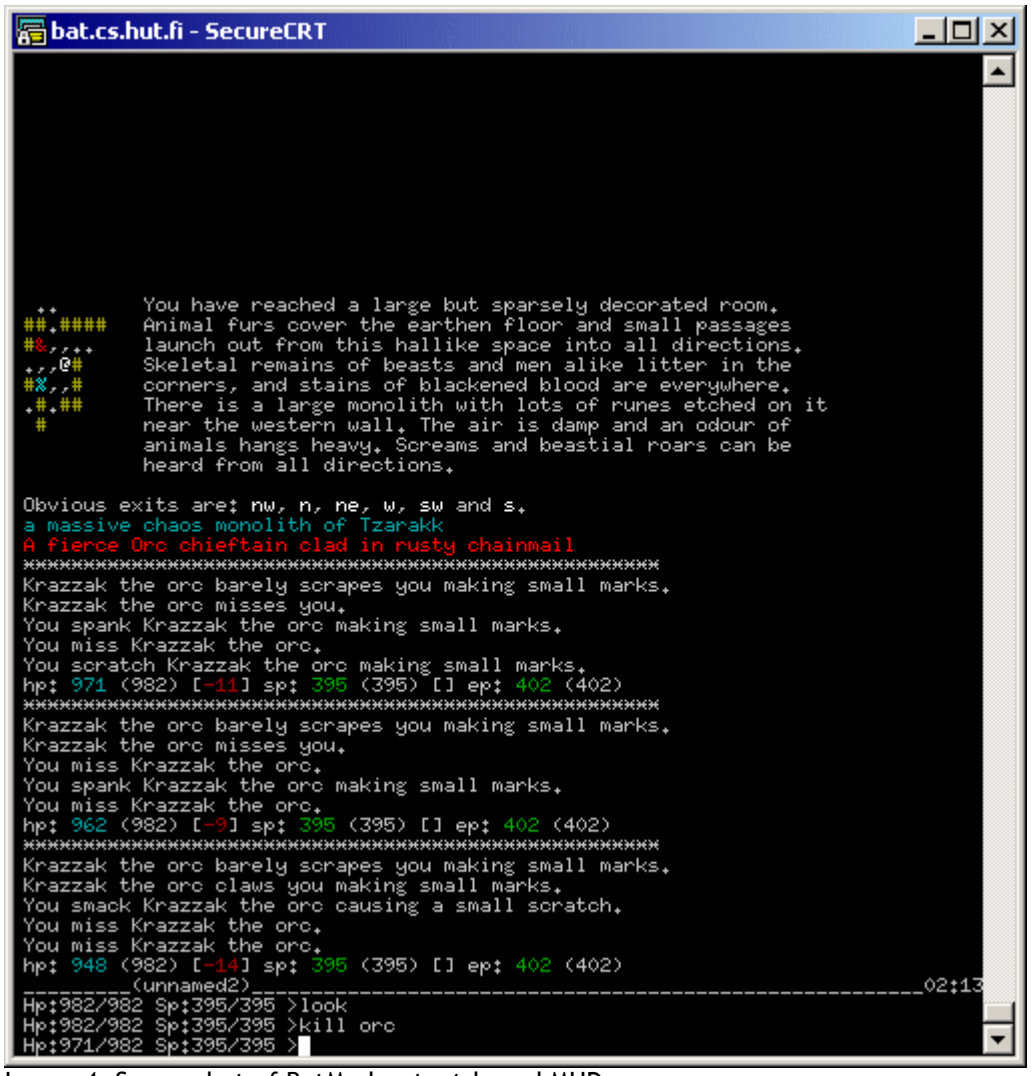

Image 4: Screenshot of BatMud, a text-based MUD game

Even without any graphics MUD games are usually fast-paced and offer good playability using text-input. Even though there are many active MUD games running (which have hundreds of simultaneous online players) they are nowadays very niche-games even among multiplayer gamers.

2.3 Game content

Modern games require a lot of content like stories, images, music and sound effects. They are very costly to create since they will require a lot of working hours and expertise. There are websites offering free graphics, soundtracks and sound-effects that can be used in games (see table). Free game content resources come with a multitude of licenses which needs to be taken into account. The Creative Commons licenses are usually the least restrictive. There are also game-specific licenses like the ”Open Game License” (OGL) which have a more viral-licensing nature. Using material licensed under OGL forces you to license everything else in

your game under the OGL license. This is very similar to the GPL licenses used for open-source software.

SOURCE	CONTENT TYPE	COMMENT
www.gutenberg.org/	Public Domain literature	Many public domain fantasy, adventure, horror and sword & sorcery novels that can be used as a base for the games story.
openclipart.org	Graphics resources	Mostly black & white sketches.
game-icons.net	Icons	A collection of over 1300 icons.
www.iwozhere.com/SRD/Gallery.html	Monster images	Free monster images (Note the license – uses the Open Game License)
opengameart.org/content/epic-fantasy-music	Fantasy music	Medieval and fantasy music.
opengameart.org/content/50-rpg-sound-effects	RPG sound-effects	A collection of sound effects.
opengameart.org	All kinds of game art	A website dedicated to all kinds of free game-art.

Table 1: Sources for free game content

Using free game graphics and sounds will save a lot of time and money and it is probably the only way for a small indie-production without financing to add graphics and audio cost-effectively to a game.

One important thing to note is that, even though this review mostly looks at PC-based MMO games, the target is still to create a game for mobile devices. There are two obvious differences between a mobile (smartphone) and a PC. Those are:

1. Screen size
2. User input devices

A 24-inch PC-monitor has 16-fold the amount of screen estate that a 4-inch mobile phone has. The PC:s are usually controlled with a mouse and keyboard while the mobiles use a touch-screen. This makes the game-graphics, especially the animations, a lot less significant on the mobile phones. The user will not be able to discern the fine details of animated graphics and the users finger will hide part of the screen when the screen is touched. The latter should be taken into account when dragging the avatar or other game-objects across the screen. It will be easier and maybe even better for usability to move a simple indicator (for example a circle) around the drag-route instead of animating the avatar or game-object while it's moving. Another option is to delay the animation so that it can be viewed after the finger has been lifted from the screen – this however will work better in turn-based games and might not be possible in real-time games.

2.4 MMO game type

There is also a classification of MMO games that separates them into “Sandbox” games or “Themepark” games. The type of the game will affect the amount of content and complexity that is necessary for the game. In sandbox-games the players continuously develop their surroundings (the game world) and the changes are permanent. They might buy a plot of land, build a castle or a farm and with fields and build their own virtual business there. Other people will see these buildings and they might buy crops from the farmers. When a house is burned down there will only be ruins left in that spot. Themepark-games have game-areas (big forests, dungeons, cities etc.) that have been created by the game developers and the players can “take a ride” on them. In theme-park games the areas stay the same and permanent changes cannot be done. The only permanent changes are to the player characters: Their experience points, items, wealth etc.

One game-website described the difference of the two game-types like this: “MMO’s can roughly be divided into two categories; those that strive to entertain the players and those that strive to let the players entertain themselves” (Lorehound 2011). It names “World of Warcraft” as an example of Themepark-type MMO and “EVE Online” as a sandbox-type of MMO. The article stresses that one requirement for the sandbox-type of MMOs is that they need a to give the players a “big toolbox” to create the content of the game and says that the steep learning curve is a problem (compared to the themepark-type games). From the developer-perspective it’s clear that traditional themepark-style games are easier to create.

The sandbox-style games have received popularity in the recent years and many MMOs are mixed and include elements from both types of games.

2.5 Architecture

This section is about the architecture of MMO games. The mobile + web platforms that we are targeting for our game have not been around for long but there is ample literature for desktop-based MMO games since that has been the only viable platform for MMO games before the smartphone era.

2.5.1 Networking

The first decision about multiplayer game architecture is how to implement the game networking. There are two ways to implement networking: The authorative implementation using client-server architecture as in illustration 2 and the non-authorative implementation using P2P architecture (Bevilacqua 2013) as seen in illustration 3.

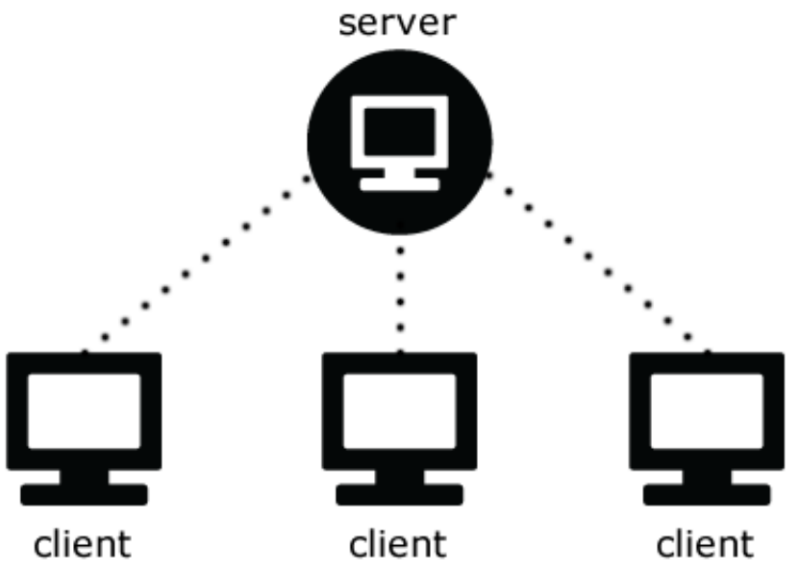

Illustration 2: Authorative implementation using client-server architecture

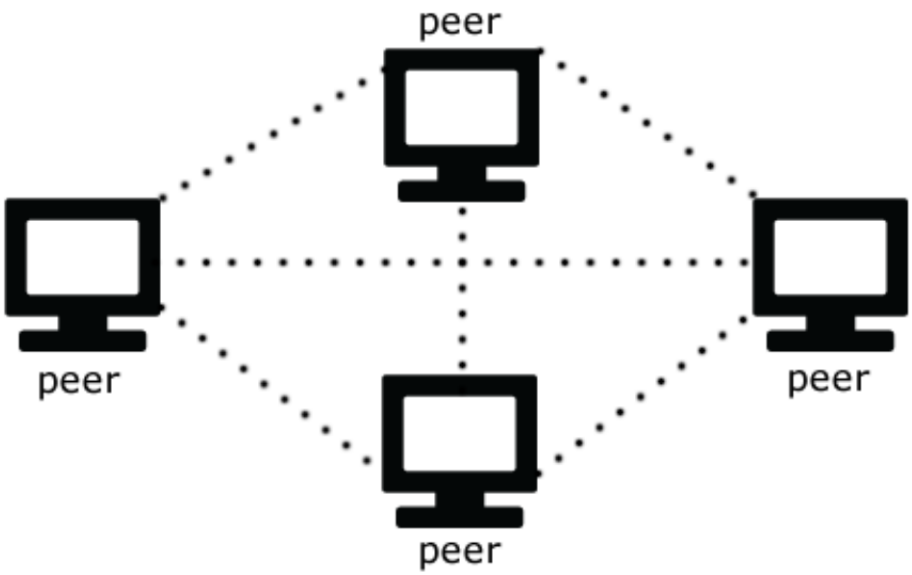

Illustration 3: Non-authorative implementation using P2P architecture

The P2P approach has some advantages like saving bandwith and computing power (Aasen & Johannessen 2009) it would require a lot of extra network-programming work and cheating prevention mechanisms. It is also poorly suited for mobile and browser based MMO games. The authorative architecture is the commonly used architecture for MMORPG games. It uses a central server which runs the game-engine and the main role of the client application is to show game events to the player and take game input (using keyboard/mouse/touch-screen).

2.5.2 Server architecture

The server architecture of MMO games is something that is not very well documented. The server architecture is key to achieving scalability in massive concurrent player numbers. The big MMORPG games have managed to build a working architecture but it is not an easy task. In the Fall of Dominion (a MMO game in open beta-testing at the time of writing) developer blog the author writes:

”Many MMO companies protect this knowledge and keep it to themselves for several reasons, mostly because developing such games is a very expensive enterprise and why should anyone get this knowledge without paying the price they had to.” (A Journey Into MMO Server Architecture 2013)

The article offers some high-level design suggestions for achieving scalability like using multiple game-servers, multiple proxy-servers, an asynchronous sync-server, separate login-servers and custom caching. It says that changes to players or other objects can be made quickly in-memory on a synch server and the server then takes care of applying all the changes to the database – regularly, but it doesn’t need to happen immediately.

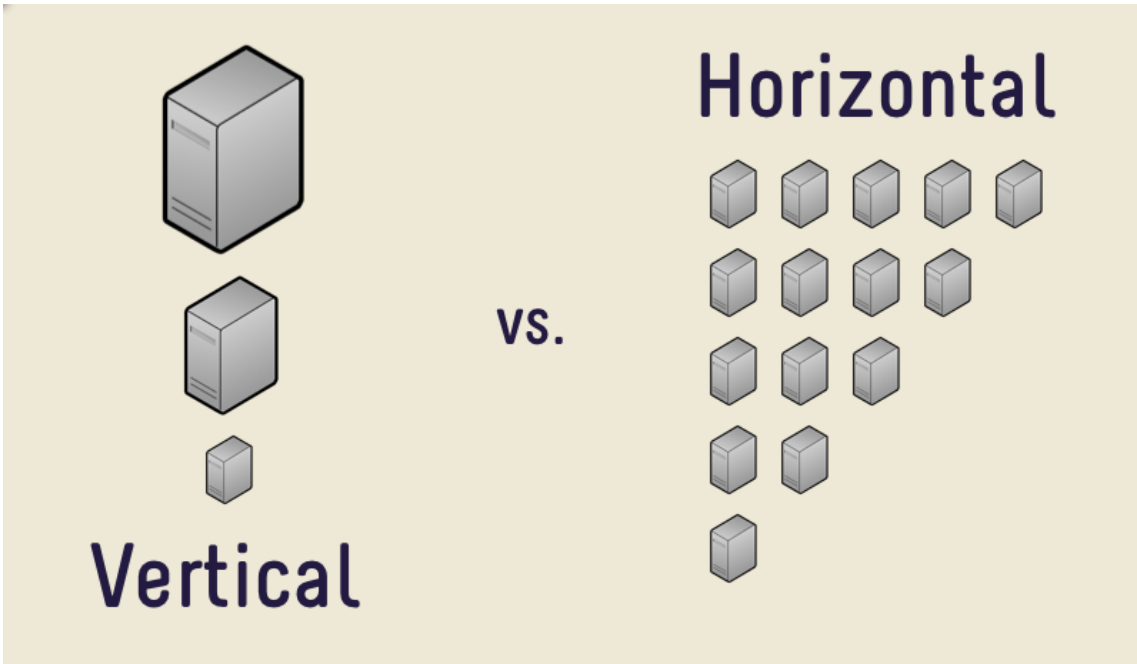

Illustration 4: Scaling vertical means bigger servers, horizontal means more servers

MMO games should strive for horizontal scalability (”scale out”) instead of scaling vertically (”scaling up”) since it has many advantages when the number of players grow. Scaling out

means adding more servers while scaling up means using bigger and more powerful servers (PC-Freak 2012). Even with a one-server setup you can make the future scalability simpler by keeping in mind the some of the best practices for horizontal scaling from the beginning: Decouple the services and partition the application (Shalom 2010). One of the most important things to take into account upfront is the database: Avoid using the traditional relational database model. The NoSQL approach will by nature provide better scalability at the cost of giving up referential integrity and transactions.

Jnan Dash, a "RDBMS veteran from IBM and Oracle" writes in a Zdnet-article: "I know from experience that as a database grows in size or the number of users multiplies, many RDBMS-based sites suffer serious performance issues." He continues: "Many commercial RDBMS products offer horizontal scaling (clustering) as well, but these are bolted-on solutions and can be very expensive and complex. If an organization is facing such issues, then it should consider NoSQL technologies, as many of them were designed specifically to address these scale (horizontal scaling or scale-out using commodity servers) and performance issues." (Dash 2013)

Since MMO games are a popular topic for programmers and game-developers a lot of discussions can be found on the subject. For example on reddit.com (Reddit 2014) there is a long discussion about MMO game architecture with some MMO developers sharing their thoughts. The main conclusion that can be drawn from these discussions is that there is no single "correct way" for the architecture.

2.5.3 Client architecture

In the authorative ("central server") multiplayer game architecture the client application is not processing the game-state since it would make it prone to cheating on the client-side. Instead the client's role is to connect to the server and do the following things:

Receive information about what is happening to and around the player.
Show (visually, audible, text based) whatever the game-server sends to the player about the game-state.
Get the players input-commands (movements and actions in the game) and send them to the game-server.

The client architecture at the implementation-level is mostly dictated by the choice of client-side technology and the best practices for the chosen development framework. There are however some design patterns that are useful for every programming language and framework when creating user interfaces. Some of the most useful ones include the observation pattern, the model-view-presenter model, the command pattern, the singleton pattern and the use of an eventbus.

2.6 Technology

The following links provide information about current technologies that could be used to implement our mobile + web based MMO game. The is no single piece of literature that would instruct how to use them together to create a MMO game framework - the information about the technologies come in bits and pieces. Even literature for creating mobile MMO games is pretty much none-existent. The main reason is that we are targeting platforms that have only been ”mainstream” for a few years. The other reason is that the life-cycle of technology-frameworks is very short and the currently most used technologies might only have existed for a year or two.

2.6.1 Client-side technologies

The main requirement for the client technology was that it's cross-platform and supports at least Android and HTML5 web applications. The primary programming language was Java.

There are many cross-platform game frameworks available. When looking at stable frameworks that have already been used for multiple commercial games – then there are some aspects that are most important for selecting the client framework:

- Language: What programming language is used for development of games
- Target platforms: On which platforms can the games be published
- Is the framework 2D or 3D oriented? Some have good support for both.
- What is the license: open or proprietary, free or commercial?

Framework	Language	Android	HTML5	Other platf.	2D / 3D	License
LibGdx	Java	Yes	Yes	iOS,Desktop,BB ,WP8(coming)	2D & 3D	Apache 2
Cocos2D	JavaScript, C++	Yes	Yes	iOS,Desktop, W8, WP8, etc.	2D	MIT
JMonkeyEngine	Java	Yes	No	Desktop, iOS (coming)	3D	BSD
Unity	JavaScript, C#	Yes	Yes	iOS,Desktop, BB,etc.	2D & 3D	Proprie-tary
Monogame	C#	Yes	No	iOS,Desktop, W8, WP8	2D & 3D	Proprie-tary
PlayN	Java	Yes	Yes	iOS, Desktop, Flash	2D	Apache 2

Table 2: Cross-platform mobile game frameworks

In addition to game-frameworks there are RIA frameworks that can be used to create web applications. Web applications can be wrapped, packaged and published into native application using cross-platform HTML5-frameworks such as the widely used “PhoneGap”-framework. Technologies that could be used to create simple HTML5 based games (even though they are

not game frameworks) include GWT, jQuery + JavaScript, Angular.js + JavaScript, Vaadin and many more JavaScript-frameworks.

2.6.2 Server-side technologies

The most important requirements (see requirements section 1.4) for the server-side framework are:

- Preferably Java-based
- Support for WebSocket
- Can be scaled in the future
- Performance

The communication between the game server and the game-clients (Android, Html, and Desktop) should use WebSocket, a relatively new technology often associated with Html5. The WebSocket protocol was standardized RFC 6455 in 2011 (IETF 2011)] and is being standardized by the W3C at the time of writing (W3C 2014). WebSocket enables the client to keep and open, bi-directional connection between the client and the server.

Platform	**Language**	**WebSocket**	**Scaling**	**Other**	**License**
Node.js	JavaScript	Using Sockets.io API	Using cluster-module	Good API support.	MIT
Vert.x	Polyglot (Java + jvm languages)	SockJS support inbuilt	Inbuilt multi-instance & clustering support	Inbuilt eventbus across cluster	Apache 2
JavaEE app-server	Java	In JavaEE 7	Multithreaded and clustering	Very widely used	Multiple - from open to proprietary

Table 3: Some server platforms that can be used for game server development

This thesis will use the Vert.X server platform for the following reasons:

- It supports java (but also other languages as it's a polyglot platform).
- It supports WebSocket (and SockJS) directly.
- It is very high performance and lean (only a few jars/megabytes).
- It is reactive (event-based) which supports real-time data and the bidirectional nature of the client-server game.
- It is based on a server-side eventbus for message transmission.
- It scales out horizontally (including the eventbus) which will make scaling out the application easier in the future if necessary.
- Has a shared-data API that works across Vert.X server instances (in the future across the whole cluster).

2.6.3 Servers and hosting

There are many cloud-based hosting solutions available. The requirements for a hosting provider for the game were:

- Primary Java
- Low cost
- Vert.x or WebSocket support

Cloud	Provider	Language	WebSocket / Vert.x	Other
AWS	Amazon	Java + other	Yes / Yes	Vert.x support on IAAS
Azure	Microsoft	Java + other	Yes / Yes	Vert.x support on IAAS
App Engine	Google	Java, Python	No / No	
Heroku	Salesforce.com	Java + other	Yes / Yes	
OpenShift	Red Hat	Java + other	Yes / Yes	
NodeJitsu	NodeJitsu	Node.js	No / No	
CloudFoundry	VMWare	Java + other	Yes / Yes	
HeroCloud	Idea Fabrik	HeroScript	not needed	Game-cloud for 3D games
GameSparks	GameSparks	JavaScript	not needed	Game-cloud

Table 4: Cloud-based gameserver or PAAS platform hosting providers

Comparing pricing, performance or even the deployment is very difficult as it would be comparing apples to oranges. A server instance on one cloud is not the same as another server instance on another cloud. For example Red Hat calls their server instances a “gear” and describes them like this: “A gear is a resource-constrained container that runs one or more user-specified software stacks, also known as cartridges. Each gear has a limited amount of RAM and disk space.” Red Hat has 3 different “gear” sizes of which the small gear has 512MB of RAM memory available. The pricing models are different for each cloud provider. Some have a very fine-grained pricing model that will charge for database-calls while others will only charge for uptimes and bandwidth. The real cost and performance can only be measured by deploying the same software on multiple clouds and seeing the performance and price when they are put through the same use.

Choosing a cloud platform comes down to checking if the platform supports the required programming language, if it supports other required technologies (like WebSocket or the Vert.x platform), if they have adequate documentation to get started and if they offer some free server-space to develop and experiment on.

This thesis will use Red Hats OpenShift cloud platform for the following reasons:

- Supports Java-based applications
- Support for Vert.X server applications
- Support for WebSocket
- Free when using a max of 3 small gears (“small server instances”)

2.7 Implementation choices

This section describes the selected design choices and technologies of the proof-of-concept cross-platform multiplayer game.

2.7.1 Game design

At the center of the multiplayer game for this thesis is the requirement for a low-cost game where adding content is fast. This applies to all aspects of the game creation: development, content, maintenance and hosting. The following design-choices were made to best achieve that goal:

- Use room-based 2D graphics since it's easiest and fastest to implement. A mobile game is mostly played on a ~4 inch screen and fancy smooth animations are not as important as they would be on big-screen desktop-games.
- Use a combination of high-quality background images + foreground images so that only little artwork is required but the game can still look good and be very visually pleasing.
- Leave out all arcade-game-elements and thereby avoid extra work on physics engines, collision detection etc.
- Use static image for all graphics. They can be acquired for a low cost. Static images can later be replaced one-by-one with animated characters, backgrounds etc.

2.7.2 Technology stack

Based on the literature review the proof-of-concept game that will be created will use the following technology stack:

Client Application	**Client Programming Language**	Java (JDK 6)		
	Technologies used for target platforms	Android: Android SDK	HTML5: GWT	Desktop: Java2D
	Client Framework	LibGDX		
	JSON (de)serializing	LibGDX JSON parser		
Connection	**Client Connection**	SockJS-client (JavaScript API)		
	Messaging Protocol	WebSocket + fallback protocols		
	Message Format	JSON		
	Server Connection	Vert.x SockJS ServerSocket		
Server	**JSON (de)serializing**	Vert.x JSON Processor		
	Server Platform	Vert.x		
	Cloud Platform	OpenShift		
	Server Programming Language	Java (JDK 7)		

Table 5: Technology stack used for the proof-of-concept game

Both client and the server will use Java as the implementation programming language. The server side uses the Java 7 SDK because Vert.x relies on the high-performance NIO2 networking class introduced in Java 7. The client side will however need to use Java6 SDK because the Android SDK only started supporting Java 7 at API level 19 (Android 4.4 “KitKat”). Using Java 7 on the client side would therefore make the Android-version of the game incompatible with all Android versions older than 4.4.

2.7.3 LibGDX

The LibGDX website describes the framework as a “Desktop/Android/BlackBerry/iOS/HTML5 Java game development framework” (LibGDX 2014) and cites the following 6 features on the front page:

- Cross Platform: Publish your games on Windows, Mac, Linux, Android, iOS, BlackBerry and HTML5, all with the same code base.
- Open Source: LibGDX is licensed under Apache 2.0 and maintained by the community. Contribute today!
- Feature Packed: Comes with batteries included. Write 2D or 3D games, let LibGDX worry about low-level details.
- Super Fast: Heavy emphasis on avoiding garbage collection for Dalvik/JavaScript by careful API design and the use of custom collection.
- Documentation: Learn LibGDX inside out on the Wiki, study the Javadocs, or read a third-party tutorial. Learn from example code and demos.
- Community Support: Get great support from a big and growing community of game and application developers.

LibGDX was selected as the client-framework because:

- It uses Java.
- It targets Android & HTML5 but also other platforms like iOS and desktop OS:s.
- It is a mature technology with a wide user base and good documentation.
- It can be extended by platform-specific code relatively easily.

Additionally the announcement for the LibGDX version 1.2.0 release had a note at the end: “Oh, and we'll look into letting you run your games on Windows Phone 8.1 officially as well. Some folks already managed to get things running using our WebGL backend :)” (LibGDX v120 2014). This means that Windows Phone 8.1+ support can be expected soon in the coming releases.

2.7.4 SockJS client

The SockJS client webpage describes the API like this:

“SockJS is a browser JavaScript library that provides a WebSocket-like object. SockJS gives you a coherent, cross-browser, JavaScript API which creates a low latency, full duplex, cross-domain communication channel between the browser and the web server. Under the hood SockJS tries to use native WebSockets first. If that fails it can use a variety of browser-specific transport protocols and presents them through WebSocket-like abstractions. SockJS is intended to work for all modern browsers and in environments which don't support WebSocket protocol, for example behind restrictive corporate proxies.” (SockJS 2014)

This game will use the SockJS WebSocket emulation for the client-server connection because using pure WebSockets will not work in many circumstances. In those cases a pure WebSocket-based solution will fall flat. WebSocket emulation will however be able to negotiate a new connection protocol and has the ability to fall back to other mechanisms like long-polling. The Vert.X server platform supports the SockJS API server implementations out-of-the-box making SockJS is the logical choice when using Vert.x.

The messages that are transported via SockJS are JSON-strings. Even though JSON is “JavaScript Object Notation” it is nowadays considered as language agnostic and widely supported on all platforms and programming languages.

2.7.5 Vert.x

The Vert.x website describes the platform like this: “Vert.x is a lightweight, high performance application platform for the JVM that's designed for modern mobile, web, and enterprise applications.” (Vertx 2014)

Vert.x is a polyglot platform and supports 6 programming languages: Java, JavaScript, Ruby, Groovy, Python and Clojure.

2.7.6 OpenShift PAAS cloud platform

Red Hat calls OpenShift “The Open Hybrid Cloud Application Platform by Red Hat” and writes: “Host your applications in the public cloud. OpenShift Online automates the provisioning, management and scaling of applications so that you can focus on development and creativity.” (OpenShift 2014)

OpenShift supports 3 low-performance virtual servers (“small gear”) for free. Developers can pick predefined modules (“cartridges”) to be installed on those virtual servers when creating an application on the cloud platform. Cartridges include for example JBoss, Vert.x, Python, Node.js, MongoDB, Roby, MySQL, Jenkins, etc.

3 Implementation

This section describes the implementation of the game. First some general issues about the architecture are described, after that implementation details for the server and the client.

3.1 Architecture

In the authorative architecture the game-server will be the only place that holds the game-state and evaluates the game-actions. The client’s main tasks are to display what the game server tells it “has happened” and to take user input and send them to the server.

3.2 The game server

The game-server handles the game objects. The game will use the following game-objects:

- Player: A human controlled GameCharacter in the game. Players are represented by an image of the character.
- NPC (“NonePlayerCharacter”): A computer-controlled GameCharacter in the game. NPCs are represented by an image of the NPC.
- Item: An item. Items can be in rooms or be carried by characters (players & NPC).
- Area: A collection of rectangular rooms.
- Room: A room that can contain game characters (both player and NPC) or items. Rooms have a background image.

The Players and NPCs or almost similar with the only difference being that Players are controlled by commands that come from the client applications as opposed to NPC:s commands that come from pre-defined scripts. The game-server does not distinguish between the players or NPC:s.

The one thing that makes a game alive is the “tick”. The game server receives a “tick” message every 2 seconds and that “tick” will make the computer-controlled NPC characters live: The tick will trigger their pre-defined command-scripts to be executed. The game-server will tick as long as it’s running and make sure the game progresses.

The “tick” is sent by the games “pacemaker” every few seconds (see illustration 5).

```
private void startTicking() {
    System.out.println("Start ticking now!");
    v.getVertx().setPeriodic(TICK_INTERVAL, new Handler<Long>() {
        @Override
        public void handle(Long event) {
            JsonObject json = new JsonObject();
            json.putNumber("TICK", turn);
            v.getVertx().eventBus().publish(BusMessage.TO_GAMESERVER,json);
            turn++;
        }
    });
}
```

Illustration 5: The game-tick is communicated to the game-server over the eventbus. By changing the tick-interval the pace of the game can be made faster or slower.

3.3 Server design

Scalability is not a requirement for the game-demo created for this thesis as the target is to create a single-server multiplayer game that can handle 100+ concurrent players. It is however beneficial to keep future scalability in mind when designing the server and choosing technology.

For scaling the GameState object needs to handle data correctly. It is used to alter the actual game-data (the “state”) and also used to fetch the latest version of the game-objects. In order to make the server scalable we need allow multiple GameServer and GameState instances to be started on the server-side. If the game-data simply existed locally in memory then the servers would soon be out of sync and the data would be different on each server instance. This is where the Vert.X shared data comes to play: It allows storing data to and retrieving data from multiple game-server instances and guarantees that the data stays the same on all instances.

The Vert.X documentation states: “Sometimes it makes sense to allow different verticles instances to share data in a safe way. Vert.x allows simple java.util.concurrent.ConcurrentMap and java.util.Set data structures to be shared between verticles.” (VertX Java Manual 2014)

The manual says that at the moment the data in of the shared data mechanism only shares data in the server instances of a single JVM but not across the full cluster. This is about to change in a future version as the Vert.X documentation on the shared data states: “Currently data can only be shared between verticles in the same Vert.x instance. In later versions of Vert.x we aim to extend this to allow data to be shared by all Vert.x instances in the cluster.”.

3.3.1 Incoming connections

When the server starts up it will first start the game-server (and start the ticks), then it will fire up http-servers that start to listen to incoming client connections. The game-server uses the Vert.X platforms SockJS server implementation to accept incoming bidirectional connections. The SockJS server will store the connections (for sending messages back) when they connect and forward all incoming messages from the clients to the incomingMessage() method.

```
SockJSServer sockServer = vertx.createSockJSServer(httpServer);
sockServer.installApp(new JsonObject().putString("prefix", super.getPath()), new Handler<SockJSSocket>() {
    public void handle(final SockJSSocket s) {
        final String connectId = s.writeHandlerID();
        addConnection(connectId, s);
        s.dataHandler(new Handler<Buffer>() {
            public void handle(Buffer data) {
                incomingMessage(connectId, data.toString());
            }
        });
        s.exceptionHandler(new Handler<Throwable>() {
            @Override
            public void handle(Throwable event) {
                event.printStackTrace();
                dropConnection(connectId);
            }
        });
    }
});
```

Illustration 6: The SockJS accepts connections and forwards incoming messages.

The communication from the server back to the client similarly straightforward.

```
SockJSSocket socket = super.getConnection(connectionId);
if(socket!=null) {
    try {
        socket.write(new Buffer(message));
    } catch(Exception exe) {
        dropConnection(connectionId);
    }
}
```

Illustration 7: Communicating from the server back to a client (identified by connectionId)

Messages coming from the game-server will be written to the corresponding clients SockJS connection. The connectionId must be mapped to the correct connected player. On errors the connection will be dropped and the client will try to reconnect (if it's still running).

Sometimes connections will drop without the server getting notified about it. Those connections need to be cleaned up. For this purpose the clients will send a heart-beat every 30 seconds. If a client has not sent a heartbeat for 40 seconds its connection will be dropped and cleaned up.

3.3.2 Game commands

When the game-server has parsed a message from a game-client and identified who is doing what – then it will execute the actual command. Commands from the none-player characters ("computer controlled characters") are of course not coming via client-connections. NPCs issue commands that from their command-scripts. The same (below) code will handle both commands from NPCs and Players - since both NPCs and Players are subclasses of the Game-Character class. The execution of the commands (depending of course on the command) will alter the game-state and in the process fire of objects called "visuals".

```
@Override
public void evalAction(long turn, GameCharacter character, GameCommand command, GameState gameState, ArrayList<Visuals> visuals) {
    String currentRoomSid = character.getRoomSid();
    String areaSid = character.getAreaSid();
    Area parentArea = gameState.getArea(areaSid);
    switch(direction) {
        case LEFT:
            if(parentArea.hasLeft(currentRoomSid)==false) {
                Visuals vis = new Visuals();
                vis.setScope(null, null, character.getSid()).setAction(Visuals.ACT_ECHO);
                vis.setMsg("You cannot go left.");
                visuals.add(vis);
            } else {
                String leftRoom = parentArea.getLeftRoom(currentRoomSid);
                Room newRoom = gameState.getRoom(leftRoom);
                boolean done = gameState.changeCharacterRoom(character, newRoom);
                // X leaves left
                Visuals visOldRoom = new Visuals();
                visOldRoom.setScope(null, currentRoomSid, null);
                visOldRoom.setSubject(character.getSid()).setAction(Visuals.ACT_LEAVE).setDirection(Visuals.DIR_LEFT);

                // Update the location now!
                Visuals changeRoomVis = new Visuals();
                changeRoomVis.setScope(null, null, character.getSid()).setAction(Visuals.ACT_CHANGE_ROOM)
                    .setSubject(newRoom.getSid()).setDirection(Visuals.DIR_LEFT);

                // X enters from right
                Visuals visNewRoom = new Visuals();
                visNewRoom.setScope(null, newRoom.getSid(), null);
                visNewRoom.setSubject(character.getSid()).setAction(Visuals.ACT_ENTER).setDirection(Visuals.DIR_RIGHT);
                visuals.add(visOldRoom);
                visuals.add(changeRoomVis);
                visuals.add(visNewRoom);
            }
            break;
```

Illustration 8: Example of the move-left action being executed.

As seen from Illustration 8 the evalAction-method will do the actual changes to the game-state. The GameState object is the only object that can change the game-state. That includes characters locations, hit points, etc. It handles the changes in such a way that nothing is overwritten or out-of-sync. After the command has changed the game-state (in this case moved the character from one room to another) some Visuals-objects will be created: One visual for the character leaving the old room, one for the player's room-graphics to be updated and one for the player arriving in the new room. The visuals are what are being graphically "played back" in the client applications. The visuals are things that players and NPCs can observe or see. Each Visual has a scope that will be used by the MessageDispatcher class to distribute them to the correct "addresses". The three scopes for visuals are:

Area-scope: Everyone (player and NPC) in the given area will see the visual. This could be lightning-strike during a thunderstorm in one of the game-areas in the game. NPCs could react to the lightning and the game-clients could visualize a lighting strike graphically to the players.

Room-scope: Everyone in the same room will receive the visualization. For example a player might say: “Hello all” and all players and NPCs in the room will receive it.

Character-scope: Only the defined character (or npc) will receive the visuals. This can be used to send events to npcs and players that nobody else receives. For example the player might receive a message from a friend: “Freddy Friend tells you ‘Howdy, how are things going?’”. In the above move-left example a “You cannot go left” message is echoed to the player if he tries to move left in a room where there is no exit on the left side.

3.4 Client

Most of the client-code is in the core-package and will work in all client-platforms. The only platform-specific code is the connection-class that handles the SockJS connections to the server.

When creating a LibGDX project for example in the Eclipse IDE – it will create one “core” project and one project for each target-platform. This game was specified to target Android, HTML5 and desktop platforms. Additionally there is a project that contains the common objects that are used on both the server and the client-side (since there is Java on both client and server sides!).

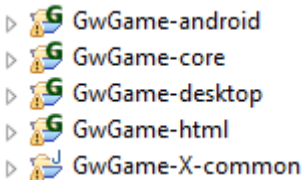

Illustration 9: Eclipse projects for this game

These are the 5 projects used for the client-application. In addition to these there is another project for the server implementation.

3.4.1 Connecting the client

In the core API LibGDX only supports basic http-client get and post calls so that using any other connection method (such as SockJS) will require implementing them using platform-specific code.

To create the platform-specific connection-classes the core-package requires a ServerConnector interface what will be implemented and used for each platform-specific project separately.

Implementing the ServerConnector for the HTML5 based project was straightforward since it could use JavaScript version of the SockJS client by importing the script from http://cdn.sockjs.org/sockjs-0.3.4.min.js and using it by calling the JavaScript-functions directly from the GWT code using GWTs JSNI (JavaScript Native Interface) functionality.

Implementing the ServerConnector for the Android platform was more problematic. There was no SockJS client to be found that would work on Android. Using WebSocket directly could have worked but was problematic since there was a problem getting the OpenShift based server to accept WebSocket connections. The pure WebSocket-based version would also not have any fallback methods for failed connections. What did the trick was adding a hidden WebView (basically a full browser) into the Android-application and loading a local webpage that was using the same SockJS-JavaScript version in the WebView. After adding some bridge-methods between the WebView and the native Android code the WebView was able to call the native code and vice-verse. The Android-connection was now working too.

Implementing the ServerConnector for the desktop platform was again more problematic. Using the JavaScript-based SockJS API with a hidden browser might have worked but since desktops were not required platforms the ServerConnector could use pure WebSockets. The WebSocket-version could still not connect to the OpenShift-based server but since the desktop-version was only used during development it was enough that it connected to the local game-server.

3.4.2 User interface

The client UI uses a model-view-presenter (“MVP”) design and uses multiple model classes:

- A MainModel class that knows about the users connection-state and contains the two sub models PlayerModel and RoomModel
- A PlayerModel that contains the players information, stats and carried items.
- A RoomModel that contains the characters and items in the room.

As opposed to usual MVP design the changes to the model will not fire off changes to the view. All changes to the view will be fired by the incoming Visuals-objects from the server. The following shows how the animation for the room-transition is done when a room-change visual is received:

```
private void changeRoom(Visuals v) {
    final Group currentRoomActor = stage.getRoomGroup();
    RoomInfo roomInfo = this.controller.getMainModel().getRoomModel().getRoomInfo();
    ArrayList<CharacterInfo> roomCharacters = this.controller.getMainModel().getRoomModel().getRoomContentCharInfosWithoutHero();
    ArrayList<ItemInfo> roomItems = this.controller.getMainModel().getRoomModel().getRoomContentItemInfos();
    if(currentRoomActor==null) {
        // No previous room - just set the new background!
        Group newRoomActor = stage.createRoomActor(roomInfo,roomCharacters,roomItems);
        stage.setRoomGroup(newRoomActor);
    } else {
        // Do room-transition animations
        final Group newRoomActor = stage.createRoomActor(roomInfo,roomCharacters,roomItems);
        currentRoomActor.addAction(VisualActionFactory.moveToDirectionAction(currentRoomActor, v.direction));
        SequenceAction seq = new SequenceAction();
        seq.addAction(VisualActionFactory.moveToCoordinatesAction(newRoomActor, v.direction,0,0));
        seq.addAction(new RunnableAction() {
            @Override
            public void run() {
                stage.setRoomGroup(newRoomActor);
            }
        });
        newRoomActor.addAction(seq);
    }
}
```

Illustration 10: Room-changing transition when receiving a room-change visuals-object from the server.

4 Creating a working game demo

In order to create a working game-demo the game will need to have some content. It will need a game-area with rooms and at least one player.

4.1 Creating a game area

The game must have a game-area with at least 1 room where the players start their game. The best way to create an area would be to create an area/room editor that can be used to create new game-areas but for a single test a hard-coded area is good enough.

```
/**
 * Load Area from DB
 */
public Area loadArea(String areasid) {
    Area a = null;
    switch(areasid) {
        case A_CITY:
            a = new Area("city", "City", "The capital city", "The capital city located between a river and a mountain-range
                    new String[] {R_FOREST_1,R_FOREST_2,R_CITY_ARENA, R_CITY_TEMPLE, R_CITY_CENTER, R_CITY_SHOPS, R_CITY_GUI
            break;
        default:
            System.out.println("ERROR: GameDAO could not load area: "+areasid);
            Thread.dumpStack();
            break;
    }
    return a;
}
```

Illustration 11: The hard-coded test-game-area called "city" only has 10 adjacent rooms.

4.2 Creating rooms

The rooms have an id, name and description. Each room has also a background image that is shown to players when they are standing in the room.

```
public Room loadCityRoom(String roomsid) {
    Room r = null;
    switch(roomsid) {
        case R_CITY_CENTER:
            r = new Room(A_CITY, R_CITY_CENTER, "City Center", "The busy marketplace at the City Center", "A lot of people are going after their b
            break;
        case R_CITY_ARENA:
            r = new Room(A_CITY, R_CITY_ARENA, "Arena", "The Great Arena for Fighters", "Arena blablabla..","rooms/city1.jpg");
            break;
        case R_CITY_GUILD:
            r = new Room(A_CITY, R_CITY_GUILD, "Guild", "The Guild of a Thousand Arts", "Guild blablabla..","rooms/city5.jpg");
            break;
        case R_CITY_SHOPS:
            r = new Room(A_CITY, R_CITY_SHOPS, "Shops", "The shopping district", "The shopping district blah..","rooms/city4.jpg");
            break;
        case R_CITY_TEMPLE:
            r = new Room(A_CITY, R_CITY_TEMPLE, "Temple", "The Temple of the Gods", "Temple blablabla..","rooms/city3.jpg");
            break;
        case R_FOREST_1:
            r = new Room(A_CITY, R_FOREST_1, "Forest", "Somewhere in the forest", "You are in the middle of a green forest.","rooms/forest1.jpg");
            break;
        case R_FOREST_2:
            r = new Room(A_CITY, R_FOREST_2, "Forest", "Somewhere in the forest", "You are in the middle of a green forest.","rooms/forest2.jpg");
            break;
        case R_FOREST_3:
            r = new Room(A_CITY, R_FOREST_3, "Forest", "Somewhere in the forest", "You are in the middle of a green forest.","rooms/forest3.jpg");
            break;
        case R_FOREST_4:
            r = new Room(A_CITY, R_FOREST_4, "Forest", "Somewhere in the forest", "You are in the middle of a green forest.","rooms/forest4.jpg");
            break;
        case R_FOREST_5:
            r = new Room(A_CITY, R_FOREST_5, "Forest", "Somewhere in the forest", "You are in the middle of a green forest.","rooms/forest5.jpg");
            break;
        default:
            System.out.println("ERROR: GameDAO could not load room: "+roomsid);
            break;
    }
    return r;
}
```

Illustration 12: There are 5 city-rooms and 5 forest-rooms in the test-game. Their actual order is defined in the parent area.

Below are two examples of the background images used for the game-test.

Image 5: Background image of a game-room in the city

Image 6: Background image of a game-room in the forest

4.3 Creating players

The player characters will be hard-coded for the test just like the area and the room's objects. In a real game the players would start the game with a character-editor where they could choose the character's name and looks.

```
/**
 * Load Player from DB
 */
public Player loadPlayer(String sid) {
    Player p = null;
    switch(sid) {
        case PLAYER_INGTAR:
            p = new Player(PLAYER_INGTAR,"Ingtar","Ingtar from the Borderland guards.","A tough looking soldier with a face that looks like it's carved from stone. C
            break;
        case PLAYER_CONAN:
            p = new Player(PLAYER_CONAN,"Conan","Conand the barbarian.","A savage cimerian barbarian with steelern muscles and the speed of a panther.", R_FOREST_1,1
            break;
        case PLAYER_GANDALF:
            p = new Player(PLAYER_GANDALF,"Gandalf","Gandalf the White","An old sorcerer wearing a white robe and a wooden staff.", R_FOREST_1,100,1);
            break;
        case PLAYER_SONJA:
            p = new Player(PLAYER_SONJA,"Sonja","Red Sonja the Warrioress","A warrioress with fiery red hair wearing chainmail and a long sword.", R_FOREST_1,100,1);
            break;
```

Illustration 13: The test-players are hard-coded for the game-test.

Image 7: Player images used for the game

Image 8: Player portraits used in the games HUD

4.4 Creating NPCs

The next step is to create a NPC - a computer controlled none-player-character. This is the code used to create a “Bard”, a simple NPC that wanders around the city singing songs:

```
public static NonePlayerCharacter createBard() {
    ArrayList<GameCommand> commandScript = new ArrayList<GameCommand>();
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_LEFT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_LEFT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SAY,null,"What is your favorite song?"));
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SAY,null,"Let me sing you about a mermaid!"));
    commandScript.add(new GameCommand(GameCommand.COMMAND_LEFT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"To yon fause stream that, near the sea, Hides mony an elf and plum, And ri
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"The day shines clear--far in he's gane Whar shells are silver bright, Fish
    commandScript.add(new GameCommand(GameCommand.COMMAND_RIGHT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"Whan, as he laved, sounds cam sae sweet, Frae ilka rock an' tree; The brie
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"Gowden glist the yellow links, That round her neck she'd twine; Her een wa
    commandScript.add(new GameCommand(GameCommand.COMMAND_RIGHT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"The smile upon her bonnie cheek, Was sweeter than the bee; Her voice excel
    commandScript.add(new GameCommand(GameCommand.COMMAND_RIGHT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_SING,null,"Trallallallallallaa..diibadaabaa.."));
    commandScript.add(new GameCommand(GameCommand.COMMAND_LEFT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_RIGHT));
    commandScript.add(new GameCommand(GameCommand.COMMAND_JUMP));
    NonePlayerCharacter npc = new NonePlayerCharacter(BARD, "A bard singing in the streets", "bard", "A bard playing and singing in the streets
    npc.setStats(50, 500, 200, 360, 50000, 15, 15, 15, 15, 15, 10);
    return npc;
}
```

Illustration 14: Creating a bard NPC for testing the game.

This is the image used for the bard:

Image 9: The (only) NPC in the proof-of-concept game: The Bard

The bard will simply execute one game-command every time the game server “ticks”. The ticks will come with a pre-defined interval (for example every 3 seconds) and are essentially what make the game “tick”.

The game-engine does not make a distinction between player characters or npc characters. They are treated equally. If a character leaves a room the game-engine doesn’t know if the

command originates from a humans keyboard or the npcs so called “command script” that gets executed on every tick.

In addition the command-script (which is run on every tick) there is also another place that can be used to bring npcs alive: their responses to visuals. Visuals are the objects that are sent to the player’s client-application so that they can “show” what is happening in the game. Just like players, all npc characters will also receive visuals about events happening in the same room. NPC characters can react to the visuals by analyzing them and issuing suitable responses. This way the npcs can become interactive. They can listen to what other characters in the room are saying and for example respond to certain keywords. They can look at players arriving in the room and decide to attack them. This way they can get a simple AI.

4.5 Testing the game

4.5.1 Log-in

After the client is started and manages to establish the connection to the server it will present a “log-in screen”. The test-game doesn’t have player registering. Instead the game will just present buttons for choosing one of the pre-defined player characters that have been hard-coded on the server-side.

Image 10: The "login screen" only has buttons for choosing one of the predefined players.

4.5.2 Entering the game

After choosing the players character the player is dropped into the middle of the players starting screen (in this case a forest-room).

Image 11: The player 'Ingtar' has entered the game.

4.5.3 Moving around the game area

The player can swipe left/right to move the character around the rooms. When using a desktop or browser the left/right keyboard keys will also work. When moving the player will remain in the middle and the background image slides away and the new background appears and slides into place.

Image 12: Room transition when player moves right.

4.5.4 Encountering NPCs

The bard is the only NPC in the test-game. It wanders around, jumps now and then and sings songs. You can see the bard arrive in your room (it slides in from left or right) and do some quick jumps occasionally.

Image 13: The bard standing in the same room as the player.

The test-game does not support fights but in a real game the NPCs could be creatures that could attack the players: dragons, demons and other monsters.

4.5.5 Multiple online players

When multiple players arrive online they will all see each other and whatever the other player is doing when they are in the same room. In the screenshot below 4 new players have arrived online making it 5 players in the same room.

Image 14: A total of 5 players standing in the same room.

The player's character always appears at the middle of the screen and the others (both players and NPCs) will stand around the player. This is only possible because the game does not have a concept of location other than the room. All characters in the same room can interact with each other no matter who is "next to whom". For example they can see each other; they could chat or attack each other. This is especially visible in the screenshot below which shows all 5 players screens (all in the same game-room) at the same time. The player characters always see themselves in the middle (check top-left portraits to see who's game-screen it is).

Image 15: The game-screens from 5 players (in the same room) viewed at the same time.

4.5.6 Test verdict

Since this is a game very early in development phase (and thus still far away from early access or beta versions) there is still little functionality to test. Whatever was completed during development of this game-framework does work however.

The game worked well on the Android-platform. Even on a low-end Android smartphone model with Android 2.2, 512MB or RAM and a 800x480 resolution screen. It also worked well on 10 inch tablet with an Android 4.0, 1GB of RAM and a 1024x768 resolution.

The game also worked HTML5-based in the latest versions of the Firefox, Chrome and Internet Explorer browsers. The screen resolution for the browser version is always fixed (when testing a screen-resolution of 1280x720 was used) but that should not be a problem as it's possible to deploy multiple versions onto the webserver and let the user choose the preferred resolution.

The desktop-version was only used for development and was not part of the requirements. It worked when connecting to the local game server using WebSocket but it didn't work when trying to connect to the OpenShift-cloud based game server. That is because the Vert.x cartridge on OpenShift does not yet support WebSocket and the server was deployed using the Vert.x cartridge. The desktop-version only tries to connect using "real" WebSocket instead of the SockJS WebSocket-emulation. The browser and Android version use the SockJS emulation

and manages to negotiate another transfer protocol. That is why the connections is still working. WebSocket on OpenShift are at the moment only supported by the custom cartridge but Vert.x cartridges are hopefully getting WebSocket support too in the future.

5 Research contributions

5.1 Applicability of the results

The results show that mobile MMO game development is possible for a small team without financing. In two month it was possible to create the core for a MMO game with working messaging and support for two platforms: Android and Web (HTML5). The created messaging functionality was finished and could be used as such for a production-ready game release. It offered private message relaying to players, message relaying to game-rooms (to game-objects inside based on the room) and message relaying to game-ares (message relaying to game-objects based on the area). The main functionality that is missing from the created proof-of-concept is a combat system and game content.

The key for creating a MMO with little resources is the word “mobile”: the screen estate on smartphones is a lot smaller than for PC-based MMO games. The graphics will not require all the details and animations that PC-games use. There is a lot of competition on PC-based MMO games as the game-type has a more than a decade long history. There are also tens of MMO games that have been created, maintained and marketed with 100M+ euro budgets. A MMO game for PCs without high-quality graphics and animations will have a hard to establish itself on the market.

Mobile MMO games are a game-type that still have to “reinvent themselves”. There are many mobile games that describe themselves as MMO games but the fact that their game-styles are very much different from game to game tells that there is no clear mobile MMO game type yet. Translating the same concept that is used on PC-based games does not work. Their learning curve is too steep, they have too much content, they are too big. They should be made for short game-sessions instead of the hours-long game sessions that are typical for PC-based MMO games. While creating a MMO-game for PC:s and game-consoles is very difficult to accomplish without a budget in millions of euros – creating a mobile MMO game on a low budget is possible and a matter of doing the right game-design choices.

5.2 Reusability of intruduced solutions

The technology stack that was used in for this game is very much reusable as such. It will shorten the research and development time for individuals and teams that plan to create a mobile MMO or a mobile multiplayer game using Java-based technologies. Using Java on both

the server and client-side was beneficial in this development project. When using the same programming language on the server and client-side, you can use the common domain objects for both sides which will make the codebase smaller, clearer and easier to maintain.

The full source-code for the game is not available as its still work in progress. It might be released at a later point.

5.3 How could the result be improved even further

On the game-content site it was missing a few features that would make it a real game:

- Fights or a combat-mode
- Skills or spells
- Areas with large maps
- Player character creator

On the technical side it was also missing a feature technology-wise that would be required for a published game:

- Persisting the player's game-state to a database.

A production-ready game would require some tools for creating and maintaining content. Hardcoded rooms and monsters are not maintainable and therefore special editors would be required for the game-objects:

- Area/map/room editor
- NPC (monster) editor
- Game Item editor

Were these above issues solved or added to the game-platform then it would be technically ready to be published on the mobile MMO-space. Another matter is if the content, story and game type, visual appeal etc. are good enough to attract players to make the game popular.

6 Conclusion

6.1 Does the game implementation fullfill the original requirements?

The created multiplayer-game prototype is a simple but a solid base for a production-ready MMO game. Without major work on scaling-out the game servers it should be able to handle a few hundred simultaneous players in a single server. With a more powerful server more than a thousand simultaneous playes should be possible. It was not possible to fully test this but as the server is very lightweight and WebSocket is a very efficient communication protocol, the server should easily be able to handle hundreds of online-players at the same time. It provides a platform with good technology that can be used to built on. By extending and scaling the created platform (as described in section 5.3.) it could become a mobile & web based MMO that runs thousands of players simultaneously. The requirement was to create a game base that could handle 100+ players and could be expanded. That requirement is fulfilled.

An estimate for finishing the game and releasing the MMO game and both Android and web (HTML5) platforms is provided in table 6.

Current proof-of-concept game	2 months
Combat-mode, some skills/spells	6 weeks
Player character creation/registering	2 weeks
Area/Npc/Item editors	3 weeks
Persist player states	1 week
Add game content	2 weeks
Testing with minor fixes	2 week
Publish game to Android/website	1 week
TOTAL	**~6 months**

Table 6: Estimate for the schedule of a production-ready mobile MMO release based on the proof-of-concept game.

According to this estimate the game would need an additional 4 months of full-time work for one person to produce a simple ready-for-production mobile MMO game.

The estimate is for a single-server setup. Scaling the game server to support thousands of online players would require several months more work since it would most likely require scaling out the game servers horizontally. This would require partitioning the game server by for example dedicating whole game-servers (or clusters) to run single game-areas of the game. This would distribute the load among the servers and maximize the performance by keeping the content of one game-area “close” so that the players and game-objects interacting with each other (in the same game-area) would be on the same server or cluster. The workload for scaling out to multiple game-servers would likely be a months and require evaluating new hosting providers and deployment strategies also.

6.2 Summary

The game server for a real-time multiplayer game is different than application servers in traditional enterprise web applications. At the core is the message flow between the clients, the game objects and the game server. It's a little like the messaging for a chat-room server – only a lot more complicated. Messages need to be evaluated, routed, duplicated, transformed or ignored based on small game-internal-details like the players or monsters skills or statistics.

Looking at the proof-of-concept game it is clear that the message-flow is and will be one of the most challenging aspects of MMO game development. It was also the area that took the most time as easily half of the time was spent on making the message-flow work: the design, implementation and the challenge to make it work on all platforms (Android, HTML5, desktop). The current player,room and area based messaging will work for a simple game but with added content the messaging will need to become content-aware. When new content is added it might introduce new rules on message flow. That is an aspect which will keep messaging a challenging part of the game-server development.

The work produced for this thesis will hopefully be published in the form of a great medieval mobile MMO game in the first half of 2015.

References

A Journey Into MMO Server Architecture. 2013. A Journey Into MMO Server Architecture. Accessed 26 August 2014.
http://www.mmorpg.com/blogs/FaceOfMankind/052013/25185_A-Journey-Into-MMO-Server-Architecture

Aasen & Johannessen. 2009. Hybrid Peer-to-Peer Solution for MMORPGs. Accessed 26 August 2014.
http://www.diva-portal.org/smash/get/diva2:347749/FULLTEXT01.pdf

Bevilacqua, F. 2013. Building a Peer-to-Peer Multiplayer Networked Game. Accessed 26 August 2014.
http://gamedevelopment.tutsplus.com/tutorials/building-a-peer-to-peer-multiplayer-networked-game--gamedev-10074

Dash, J. 2013. RDBMS vs. NoSQL: How do you pick? Accessed 26 August 2014.
http://www.zdnet.com/rdbms-vs-nosql-how-do-you-pick-7000020803

Francis J. 2014. The Spiraling Cost of MMOs. Accessed 8 September 2014.
http://mmo-play.com/mmo-blog/spiraling-cost-of-mmos

Fritz, B. & Pham, A. 2012. Star Wars: The Old Republic - the story behind a galactic gamble. Accessed 26 August 2014.
http://herocomplex.latimes.com/games/star-wars-the-old-republic-the-story-behind-a-galactic-gamble

Gamespot. 2014. Footage of Defunct 38 Studios' MMO Project Copernicus Emerges. Accessed 26 August 2014.
http://www.gamespot.com/articles/footage-of-defunct-38-studios-mmo-project-copernic/1100-6421062

Handrahan, M. 2014. The Old Republic earned $165 million last year. Accessed 26 August 2014.
http://www.gamesindustry.biz/articles/2014-07-18-the-old-republic-earned-usd165-million-last-year-report

IETF. 2011. The WebSocket Protocol. Accessed 26 August 2014.
http://tools.ietf.org/html/rfc6455

Jordan, L. 2013. What Can We Learn From MMO Development Costs? Accessed 26 August 2014.
http://www.gamebreaker.tv/mmorpg/can-learn-mmo-development-costs

Kern, M. 2014. Up to 90% of MMO Real Estate is Wasted. Accessed 26 August 2014.
http://www.mmorpg.com/showFeature.cfm/loadFeature/7559/Up-to-90-of-MMO-Real-Estate-is-Wasted.html

Lee, J. 2003. Relational Database Guidelines For MMOGs. Accessed 26 August 2014.
http://www.gamasutra.com/view/feature/131216/relational_database_guidelines_for_.php

LibGDX v120. 2014. libGDX 1.2.0 released. Accessed 26 August 2014.
http://www.reddit.com/r/gamedev/comments/28srpd/libgdx_120_released

LibGDX. 2014. LibGDX official webpage from Bad Logic Games. Accessed 8 September 2014.
http://libgdx.badlogicgames.com

Lorehound. 2011. The Merits of "Sand Box" versus "Theme Park" MMO's. Accessed 26 August 2014.
http://lorehound.com/news/the-merits-of-sand-box-versus-theme-park-mmos

Luna, F. 2006. Introduction to 3D Game Programming with DirectX 9.0c. Wordware Publishing Inc.

Mäntylä, M. 2001. Methods for software business and engineering research. Accessed 26 August 2014.
www.soberit.hut.fi/~mmantyla/work/Research_Methods/Constructive_Research/constructive_research.ppt

Magnani, L. & Carnielli, W. & Pizzi, C. 2010. Model-Based Reasoning in Science and Technology. Springer Berlin Heidelberg.

OpenShift. 2014. OpenShift by Red Hat official webpage. Accessed 8 September 2014.
https://www.openshift.com

PC-Freak. 2012. What is Vertical scaling and Horizontal scaling. Accessed 5 September 2014.
http://www.pc-freak.net/blog/vertical-horizontal-server-services-scaling-vertical-horizontal-hardware-scaling

Peterson, S. 2014. MMORPGs: Time to level up. Accessed 26 August 2014.
http://www.gamesindustry.biz/articles/2014-02-26-mmorpgs-time-to-level-up

Plunkett, L. 2008. How Much Has WoW Cost Blizzard Since 2004? Accessed 26 August 2014.
http://kotaku.com/5050300/how-much-has-wow-cost-blizzard-since-2004

Reddit. 2014. Interested in MMO server architecture. Accessed 26 August 2014.
http://www.reddit.com/r/gamedev/comments/1w746u/interested_in_mmo_server_architecture

Shalom, N. 2010. Scale-out vs Scale-up. Accessed 26 August 2014.
http://ht.ly/cAhPe

Sharkey, M. 2014. Rumor: The Elder Scrolls Online Cost $200 Million to Make. Accessed 26 August 2014.
http://www.gamefront.com/rumor-the-elder-scrolls-online-cost-200-million-to-make

SockJS. 2014. GitHub sockjs / sockjs-client website. Accessed 8 September 2014.
https://github.com/sockjs/sockjs-client

Superdata Res. 2014. US digital games market update: July 2014. Accessed 26 August 2014.
http://www.superdataresearch.com/blog/us-digital-games-market

The Game Reviews. 2009. World of Warcraft by the Numbers. Accessed 26 August 2014.
http://www.thegamereviews.com/article-1515-World-of-Warcraft-by-the-Numbers.html

VertX Java Manual. 2014. Java API Manual. Accessed 26 August 2014.
http://vertx.io/core_manual_java.html

Vertx. 2014. Vertx official webpage. Accessed 8 September 2014.
http://vertx.io

W3C. 2014. The WebSocket API. Accessed 26 August 2014.
http://dev.w3.org/html5/websockets

Webopedia. 2014. What is MMORPG? Accessed 8 September 2014.
http://www.webopedia.com/TERM/M/MMORPG.html

Age of Conan. 2014. More Media – Screenshots. Accessed 8 September 2014. http://www.ageofconan.com/media/screenshot#list

Illustrations

Illustration 1: Two plans for constructive research 12
Illustration 2: Authorative implementation using client-server architecture 23
Illustration 3: Non-authorative implementation using P2P architecture 23
Illustration 4: Scaling vertical means bigger servers, horizontal means more servers 24
Illustration 5: The game-tick is communicated to the game-server over the eventbus. 33
Illustration 6: The SockJS accepts connections and forwards incoming messages. 34
Illustration 7: Communicating from the server back to a client (identified by connectionId) 34
Illustration 8: Example of the move-left action being executed. 35
Illustration 9: Eclipse projects for this game 36
Illustration 10: Room-changing transition when receiving a room-change visuals-object from the server. 38
Illustration 11: The hard-coded test-game-area called "city" only has 10 adjacent rooms. 38
Illustration 12: There are 5 city-rooms and 5 forest-rooms in the test-game. Their actual order is defined in the parent area. 39
Illustration 13: The test-players are hard-coded for the game-test. 40
Illustration 14: Creating a bard NPC for testing the game. 42

Images

Image 1: Screenshot from 'Age of Conan MMORPG game 17
Image 2: A tile-based game 'TibiaMe MMO' for Android 18
Image 3: Room-based 2D-game 'Avatar Fight'..... 18
Image 4: Screenshot of BatMud, a text-based MUD game 20
Image 5: Background image of a game-room in the city..... 39
Image 6: Background image of a game-room in the forest 40
Image 7: Player images used for the game 41
Image 8: Player portraits used in the games HUD..... 41
Image 9: The (only) NPC in the proof-of-concept game: The Bard 42
Image 10: The "login screen" only has buttons for choosing one of the predefined players.43
Image 11: The player 'Ingtar' has entered the game..... 44
Image 12: Room transition when player moves right. 45
Image 13: The bard standing in the same room as the player. 45
Image 14: A total of 5 players standing in the same room. 46
Image 15: The game-screens from 5 players (in the same room) viewed at the same time.47

Tables

Table 1: Sources for free game content 21
Table 2: Cross-platform mobile game frameworks 26
Table 3: Some server platforms that can be used for game server development..... 27
Table 4: Cloud-based gameserver or PAAS platform hosting providers 28
Table 5: Technology stack used for the proof-of-concept game 29
Table 6: Estimate for the schedule of a production-ready mobile MMO release based on the proof-of-concept game. 50