Artem Moskalev

# Multiplayer Game Server Software Architecture

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Program in Information Technology

Thesis

05.04.2014

Helsinki Metropolia University of Applied Sciences

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Artem Moskalev<br>Multiplayer game server software architecture<br><br>37 pages<br>5 April 2014 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Internet Software Engineering |
| Instructor(s) | Olli Hämäläinen, Senior Lecturer |


The purpose of the project was to show how large multiplayer server software used to be implemented. The goal of the project was to create working multiplayer game server software for a role-playing game.

The project was carried out in multiple modules, each of those representing an important part of the game server software. As for the platform, upon which the implementation was built, Java was chosen. Multiple frameworks were used in this solution in order to achieve scalability, performance and maintainability required by the project goal.

As a result, the game server software was created. It allowed multiple thousand players to connect to a single point, to play the game online, and to synchronize the state of the characters. Each character was capable of performing location-based functions. The common world was shared among all the player instances. The software supported asynchronous full-duplex server-client communication, and the load of more than one thousand users. The authentication capabilities were also embedded into the game.

The project shows how large complex software systems can be implemented. The server software proves that creating network-based games is a challenging task and requires the knowledge of different fields of computer science.



| | |
|---|---|
| Keywords | multiplayer game, server software, Java network programming |

**Contents**

## Abbreviations and Terms

| | |
|---|---|
| API | Application Programming Interface |
| CRUD | Create-Read-Update-Delete |
| FTP | File Transfer Protocol |
| GC | Garbage Collection |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| I/O | Input/output |
| Java EE | Java Enterprise Edition |
| Java SE | Java Standard Edition |
| JPA | Java Persistence API |
| JVM | Java Virtual Machine |
| LAN | Local Area Network |
| MMO | Massively Multiplayer Online |
| MVC | Model-View-Controller |
| NIO | Non-blocking Input/output |
| OO | Object Oriented |
| ORM | Object-Relational Mapping |
| RDBMS | Relational Database Management System |
| RPG | Role Playing Game |
| SQL | Structured Query Language |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| XML | Extensible Mark-up Language |
| XMPP | Extensible Messaging and Presence Protocol |

# 1 Introduction

At present, computer games are becoming an important part of people's lives. They are played at home, in public transport, and even at work. This popularity is due to the fact that games can be addictive, can help to spend time with pleasure and to communicate with other people. Moreover, some types of computer games can also help in learning, enhancing memory, can improve attention to detail and analytical skills. As a matter of fact, playing games helps some people to socialize in the virtual world, find new friends and improve teamwork. The latter makes some games especially valuable to employers who want to introduce some competition into the work process, or to improve the teamwork of their employees. [1, 91-94.]

Computer games range from single player shooting simulators to multiplayer games, where people struggle to defeat real human opponents or cooperate to complete difficult game missions in the virtual universe. Therefore, massively multiplayer online (MMO) games have to manage large numbers of players at the same time and synchronize them efficiently, which is not a trivial task and can be daunting for game developers. The game genre discussed in the context of the current project is a role playing game (RPG). In such games, players have their own characters which can be eventually upgraded. Nevertheless, to achieve this goal, cooperation with other people or game objects is a priority. When role-playing games are multilayer, they can be attended by thousands of people at the same time. In this case, they fall into the category of MMO RPG. These are very interesting from the point of view of technology, because MMO RPGs deal with algorithms, artificial intelligence, concurrency, networking, throughput and memory issues and other challenging fields of computer science.

Thus, the topic of this project is the architecture of a MMO RPG server. It deals with difficult and interesting concepts of computer science, specifically network programming and concurrency, as well as large software system design. The goal of the project is to design and implement software, which would be capable of handling multiple concurrent players, providing the services for playing the game, persisting user data, and synchronizing the clients communicating with the server. The software is made in collaboration with another student, Yuri Shukhrov, who is responsible for creating a user interface for the game. The sample images of the user interface will be present in this work.

The scope of the project includes connection establishment and network handling capabilities of a multiplayer game server, the protocol used between the server and its clients, basic gaming services, such as the character management and world instance support, and persisting the data of the players. Moreover, the project includes initial security capabilities, game routines, such as fights, and automatic state management (such as login and logout). However, algorithms, artificial intelligence, and testing are beyond the scope of this project. On completion of the software, it will be possible to play the multiplayer RPG through the implemented server software.

The thesis work continues and extends the innovation project on the same topic. The game server software is improved in terms of its design, and the functions it provides. The functions which are added mostly relate to the game and network modules of the server software. The system is also enhanced in terms of software design. It lacks redundant dependencies, and possesses simplified object-oriented (OO) structure. A broad range of messages are introduced to further improve the communication between the server and its clients. There are some other improvements in the system design ranging from new player interaction capabilities to data-driven object design.

Concurrency and handling user data remain the main features of the project. In this part of the project the discussion is continued. The game data storage and retrieval is discussed in more detail in the thesis project and its influence on game performance is roughly evaluated. Moreover, the alternative technologies for the software development are discussed and some light is shed on the limitations and drawbacks of the chosen strategy to developing the software system in question.

# 2 Software Functionality

## 2.1 Game Overview

The main purpose of the project is to create functional game server software for playing a multiplayer RPG. The most interesting feature of the project is that players take actions in turn and see the results of their actions as the text messages. The outcomes of these actions are usually random. In order to achieve a successful outcome, the players must improve their character and develop new abilities. The higher level the player possesses, and the better equipment the player has, the higher his or her chance of winning is.

There are a variety of characteristics that can describe the character. The most interesting part is the hit points, damage, critical, anti-critical, dodge and anti-dodge points of the character. These characteristics directly influence the performance of the character in the fight and can be upgraded. Dodge and anti-dodge are responsible for the ability of the character to avoid and to deliver hits to other players. Critical and anti-critical points influence the ability of characters to deal and to protect themselves from critical strikes. The main set of upgradeable characteristics is shown in figure 1.

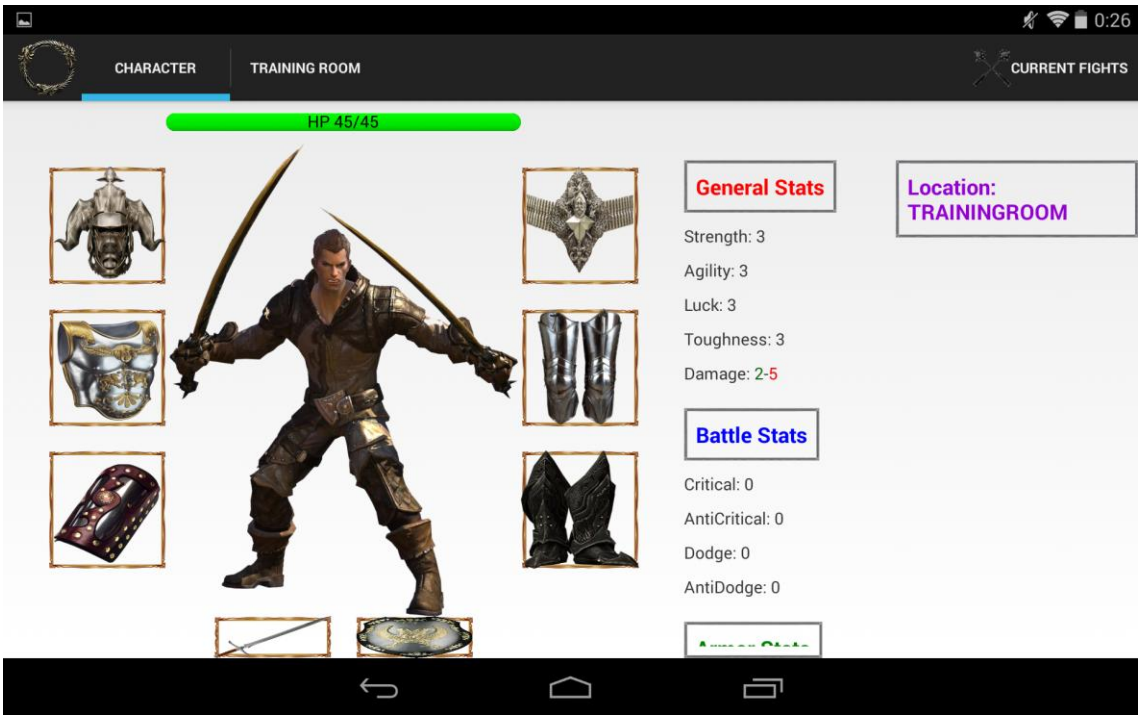


Figure 1. Game interface character screen with statistics

Figure 1 implies that the player can equip his character with items. Items further improve the ability of the character to compete in the fights, boosting damage, hit points and other important characteristics. The game world is built around locations. When the player is first initialized in the game world, his character is situated in the training room location as shown in figure 2.

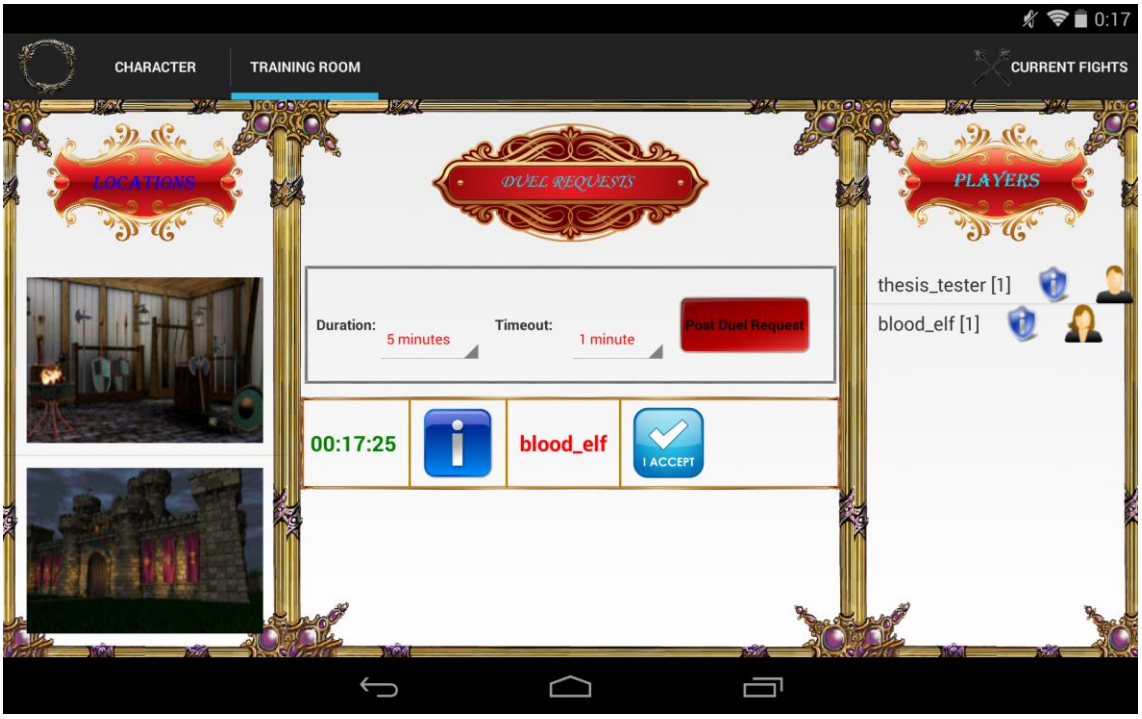


Figure 2. Character placed in the training room location

Every location has a predefined set of functions which a player can utilize. These functions range from location to location, each location having a unique set of functions. As for the training room, the character can create a duel request, which is an invitation to other players to fight. Figure 2 shows the view, which a player has in the training room. If one of the players publishes an invitation to have a fight, all other people in the location can see and accept it. In this case a fight starts.

Figure 3 illustrates a fight between two players. When in the fight, two players` characters are shown opposite each other. Now, they need to deal damage to each other until one of them loses all the hit points. The hit point bars are shown over the heads of the characters. Each player has four body areas to attack another player into, but can choose only one. The player also has four positions to block from which he can choose only two.

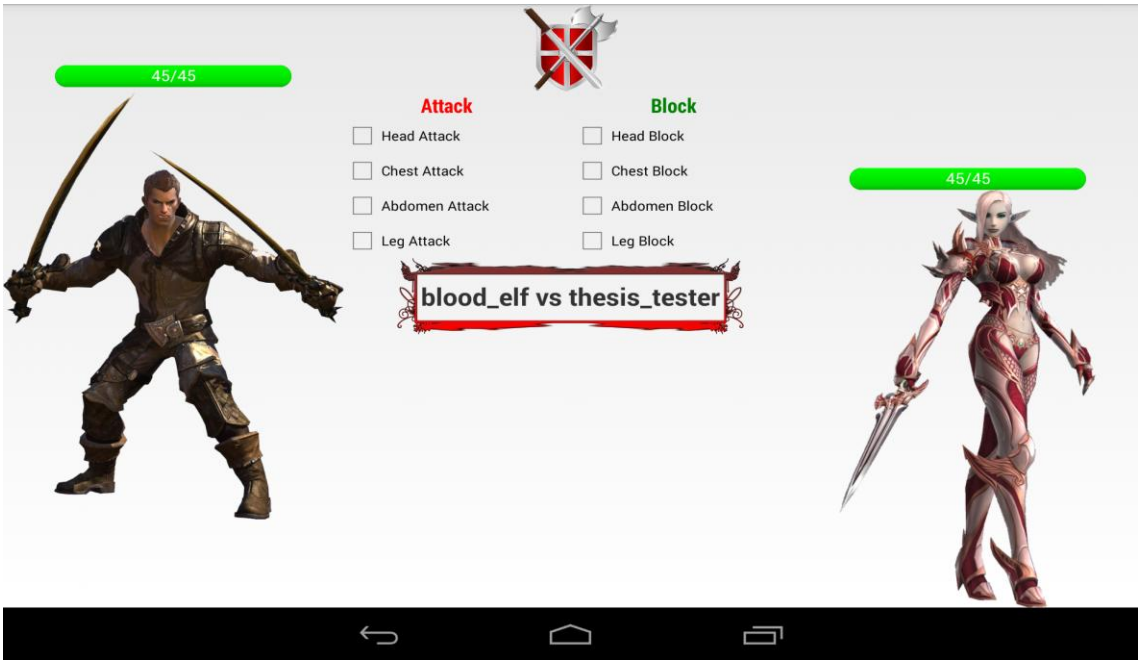


Figure 3. Fight between two players

When the choice is made, the player presses an attack button. The data is stored in the game, and the player begins waiting for the other player`s response as shown in figure 4.

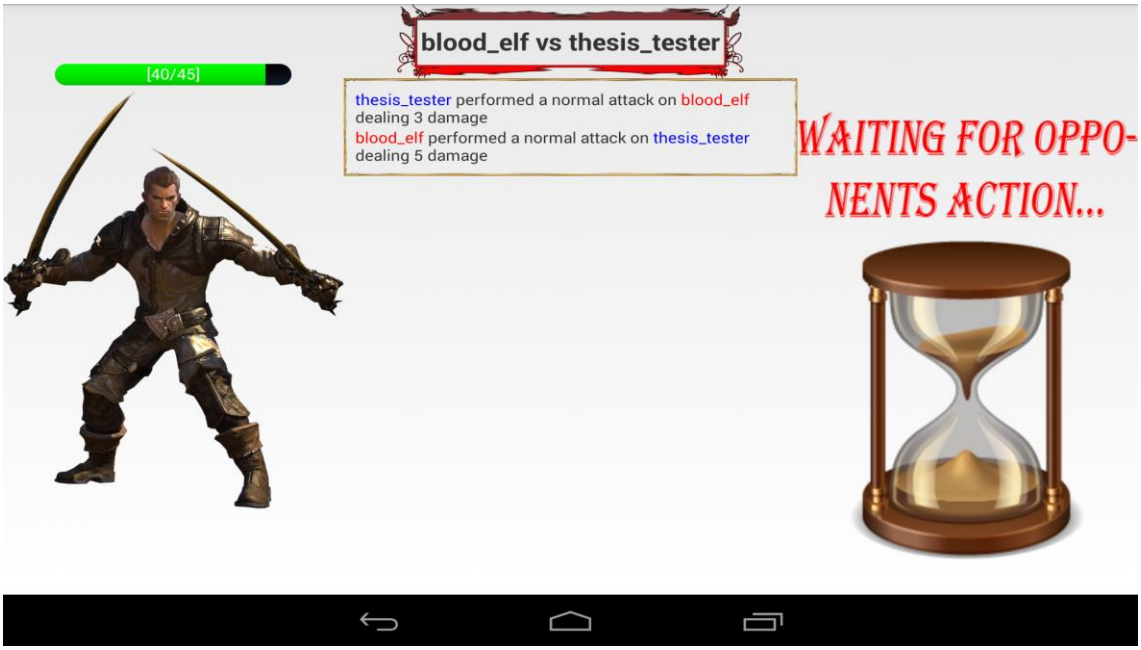


Figure 4. Character awaiting response in the fight

Both players have the same fight functionality, and they do not know their opponent`s moves. In these circumstances, the blocking and damage dealing events are random. Blocking happens when one of the players randomly attacks the body part of the opponent which has been blocked. The damage dealing event occurs when the body part which has been attacked, has not been blocked by the opponent. At the attack, a critical strike might happen (it means that the player would deal twice as much damage) or a dodge event (it means that another character automatically avoids an attack).

This type of fight goes on until one of the character loses all the hit points. In this case a victory is given to one of the players as seen in figure 5.

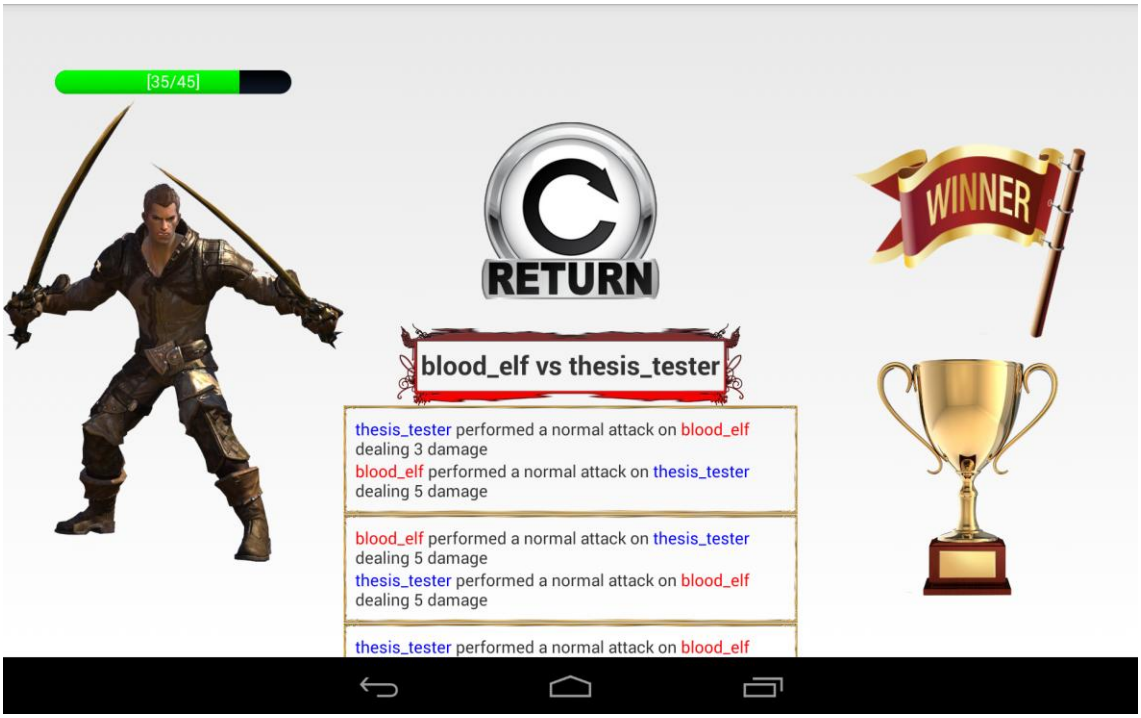


Figure 5. Character winning the fight

As seen from figure 5, there is a log in the middle of the screen. It notifies players about events happening in the fight. All the attacks are documented in it in order for the players to see if their actions have taken any effect.

## 2.2 Server Software Requirements

As long as the server software has to support synchronization of clients with each other and provide data storage service, it needs to have a broad range of features implemented. One of the most important requirements is connected to the networking of such an application. It should be able to:

- Be able to accept incoming requests from clients and establish sockets with them
- Receive data from clients
- Send messages to clients asynchronously
- Have a small memory footprint
- Be able to accept a few thousand clients simultaneously
- Be modular and extensible by design.

The listed requirements relate only to the technical aspect of the game server software. Nevertheless, there is a block of game functions which the software should be able to perform. One of the main functions of the game server software is to perform authentication of players. These functions include:

- Registering new users
- Logging new users in by checking their credentials
- Finding players` characters and linking them with their profiles
- Re-logging players in case the connection is lost
- Signing players out
- Breaking the connection in case it is idle for a long time
- Breaking connections in case malformed requests are received from clients
- Not allowing for concurrent use of one account by multiple players.

When the player has been registered, he needs to perform the game functions. The game server software representing the game world has all the processes running as a global object shared by all the clients and it needs to provide the following game services available to all players:

- Ability to see the characteristics and equipment of the players` characters
- Storing the character in the database after the player signs out
- Ability to perform fights with other players
- Ability to improve equipment and fighting abilities of the character
- Walking around different locations
- Tracking the fight progress by reading logs
- Seeing lists of players available in any location

The listed functions are important in order to begin playing the most basic version of the game. There can be other features implemented in the game, but they generally come down to the listed functionality.

# 3 Theoretical Background

There are a great variety of multiplayer games, MMO game architectures, and concepts, but the basic principles remain the same for all of these software systems. The use of various software architectures and approaches is chosen according to the needs of a particular game. Thus, ubiquitous methodologies and designs form unique server software platforms when combined in a certain way. In order to explain the system architecture of multiplayer game server software, the whole platform is divided into separate layers and parts, which are further investigated in detail. The given approach helps to reduce the complexity of the system as a whole and systematize constituent parts.

## 3.1 Client-Server Communication Protocols

### 3.1.1 Transport Protocol

The primary function of the game server software is client synchronization on a large scale. Therefore, time of message delivery from a client to the server and reliability of such a transmission is critical. Thus, the transport protocol used in the interaction of the server and its clients is crucial and must be taken into consideration.

There are basically two common transport protocols for connecting server software and clients: namely Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Each of them has some limitations and advantages over the other. Their use in a particular situation is influenced by the requirements posed by the software system. If latency is the priority in the game (such as a real-time shooter or sport simulator) then UDP is the right choice. However, if the priorities are reliability, the order of message delivery (like in a chess game between two players) and security, TCP is the obvious choice. UDP is more reasonable to use in small LAN games where the message loss or corruption is improbable. [2, 644.]

TCP is a full-duplex communication protocol between the server and the client, which represents two abstract streams, one of which is from the client to the server and the other is from the server to the client. When the data needs to be delivered, it is written into the outbound stream and read on the other side. When the data needs to be read, the process is the opposite. In addition, when there is a problem with the connection,

the exception will be raised and it can be handled in a reliable way. All the data is received in the right order, with no duplicates. All these benefits of TCP are achieved due to a complex and thorough specification of this transport protocol. TCP packets carry much more supplementary data in the payload in order to provide the listed bonuses, and weigh more than UDP packets. The fact that each retransmission point of the network on the message delivery path has to examine the validity of the information sent, and take steps to prevent data loss or corruption, makes the processing times of TCP packets much higher than those of UDP. [2, 644.]

Unlike TCP, UDP is less reliable. The sent data is not guaranteed to be received by the other side of the communication channel. The pieces of the sent data can sometimes arrive in the wrong order. Moreover, no error will be raised if an exception (such as lost data) occurs. In this case, all the sent data might be lost, and neither the client, nor the server, will know about it. In order to utilize UDP, the game server software has to implement its own software module which would track the errors in the communication process such as broken or lost data packets. The arriving messages have to be rearranged by the receiving software module to put them in the right order. It is obvious that constructing fully-functional UDP enabled game server software is a difficult and non-trivial task. [3, 27-28.]

In the current project, the TCP protocol is used as a primary transport protocol for message exchange between the server and the client. The game is turn-based, which means it tolerates high latency. However, each command that a player issues is important and must not be lost or corrupted. As long as the game server software is accessible for the clients globally, the loss and corruption of transmitted data is very probable along such distances. Therefore, reliability and message delivery order are the main priorities for the game. All the mentioned qualities fully correspond with the TCP transport protocol, and, thus, TCP is used as a transport protocol for message exchange between the server and clients in the current project.

### 3.1.2 Application Layer Protocol

The communication between the server software and the client happens on multiple levels. The highest level of such communication is application communication protocol. There are a number of existing protocols working on this level of abstraction including Hypertext Transfer Protocol (HTTP), Extensible Messaging and Presence Protocol

(XMPP), File Transfer Protocol (FTP) and others. Each of these protocols serves a particular purpose. As for HTTP, it is used for most of the web content on the Internet, allowing people to browse web pages and get access to resources such as documents or music.

As a matter of fact, large distributed software systems such as multiplayer games employ their own custom protocols to deliver messages from the client to the game server and receive the responses in legible format. The application layer communication protocol must be flexible and powerful enough to deliver any type of information required by the game server software or its clients in a definitive yet standardized way. Thus, a custom protocol based on Extensible Mark-up Language (XML) for data presentation has been developed for the project. It features an asynchronous request-response code-based approach for delivering messages from the client to the server software. The server software pushes messages to the client asynchronously when the response is ready, which is possible due to TCP underlying the communication process. The requests from the client to the server always represent an action to perform. They change the state of the player`s character in the game world. The typical request is presented in listing 1.

```
<request command="login">
	<parameters>
		<parameter name="login" value="elf" />
		<parameter name="password" value="male" />
	</parameters>
</request>
```

Listing 1. Example of login request issued by the client

Each request consists of one command which defines an action and zero to multiple command parameters. These parameters carry different name-value pairs which are then parsed by the server software in order to retrieve the details for the received command. In listing 1, the required parameters for the login command are login, and password. These parameters are supplied every time a login command is sent to the server. Next, the server software request processing module decodes the message and acts accordingly. The server can send a response or a stand-alone message to the

client in order to notify it of the changes happening in the game world. The messages from the server software are structured differently than request messages.

```
<systemMessage>
     <code>13</code>
     <text>You have logged in!</text>
</systemMessage>
```

Listing 2. Example of login message from the server

In listing 2, the message from the server about successful login procedure is presented. This message can be sent from the server every time when the login command has succeeded. However, this type of message is returned only as a response to the request. Nevertheless, listing 3 presents a message from the server that can be received without any prior request.

```
<gameMessage>
    <code>301</code>
    <location>TRAININGROOM</location>
    <character level="1" gender="male">elf</character>
</gameMessage>
```

Listing 3. Example of message from server stating that some player has entered the location

All the messages received from the server contain the code which must be interpreted by the client in order to decide what to do with the message. After the code is understood, the message is scanned, and necessary parameters are extracted. Thus, the client can be notified by the server of new updates in the game world and hold the player up-to-date.

## 3.2 Network Module I/O Strategy

The network module is the central part of the server software system and an entry point for the incoming messages to the game world, decoding and storing them in the memory and linking game resources with the communication channels. It is also responsible for establishing and breaking connections to clients, detecting the incoming messages on the channels, starting new concurrent tasks for serving requests, and detecting the periods of idleness of clients.

As long as the main purpose of the network module is to synchronize and connect a large number of clients together, an appropriate strategy for the task has to be chosen. There are several approaches to achieve this. One of them is a “blocking input/output method”. When a socket connection is established, the server needs to read the incoming data through the input stream. When it is done in a blocking manner, there must be a thread for each client on the server. When there is no data transferred, the thread is blocked (doing nothing) in the memory, consuming resources. When the data arrives, the thread immediately proceeds to its execution. The drawback of this approach is that there might be a large amount of clients sending no data. Nevertheless, each thread takes a significant effect on the memory and, thus, the server cannot have more than a few thousand concurrent connections at the same time.

Another approach is the “non-blocking input/output”. This approach utilizes a predefined number of threads which reside in the memory. The presumption is that most of the threads are inactive for most of the time. When new data arrives in the socket, it is stored in a buffer by the operating system [4, 791]. Next, there must be a background thread checking all the buffers of clients in an infinite loop. When it detects that there is some data in one of the buffers, it allocates a new thread from the thread pool. This thread now performs some action with the data received in the buffer. When the processing is over, the thread is returned to the pool where it will reside until the next buffer full of data is detected.

The non-blocking approach can provide a larger number of concurrent connections simultaneously. It is achieved due to the fact that there is a limited amount of threads even though there are a large number of clients connected. Nevertheless, when the latency is the priority, the blocking approach should be chosen. Even though the non-blocking connection management can reduce the number of threads and, thus, the

impact on the memory, the background thread checking the incoming data must check all the buffers in turn, which means there is a small delay between receiving the data and its detection. However, the delay is a few milliseconds, which is an acceptable latency time for most of the game server software architectures. [5, 7-17.]

This project uses the non-blocking approach for serving client requests as long as the latency is not the main priority for a turn-based game but memory is. Moreover, due to the nature of the game, there will inevitably be many clients silently waiting for server responses and, thus, occupying memory. As long as the genre of the game is RPG, the server software will store big amounts of data in memory at any given moment of time. The non-blocking approach allows reducing the memory use by limiting the number of allocated resources to players.

## 3.3 Concurrency

The concurrency issue is applicable to all multiplayer games. This concept spans beyond the networking layer of the multiplayer game server software. When dealing with threads, a few very important concepts come into light: resource safety and access fairness, as well as liveliness and performance of threads.

In a large multiplayer game some of the resources of the game (location, items, trajectory of the bullet) are shared among multiple players that can use them. Nevertheless, while one player is using a resource, the other players to whom the resource is also available, might not wait until the first player completes his work with the resource. As long as the game server software is a highly concurrent environment, many actions happen simultaneously. If these actions modify the same data, an unexpected failure might occur, even leading to serious consequences to the whole system. Such a situation happens when one of the players changes the shared data. If any player uses the resource while it is being changed by another player, the resource might be left in an unexpected state. This fact raises the necessity for the measures to ensure that no incompatible actions happen at the same time. Such measures ensure that the principle of resource safety is implemented. The main idea behind the principle of resource safety is to protect players from unexpected results. However, special techniques need to be used to achieve well performing, but at the same time safe system design. In order to achieve the resource safety, some constraints must be introduced. [6, 5-6.]

One of the ways to protect a resource is such a synchronization mechanism as lock. Locks prevent multiple players from misusing a resource. This mechanism is implemented in many variations in different programming languages. When a resource is protected by a lock, it is important to ensure that the resource is evenly shared among players as long as some of them might not be able to access the resource due to waiting. If this constraint is not fulfilled, one of the players might wait for too long for the server software to respond to his request, because other players have access to the resource in question and the waiting player is never chosen to use the resource.

Moreover, a situation called “deadlock” might happen in the process of synchronizing the access to multiple resources. In this situation, one of the players acquires an access to a resource and needs to access another resource too. However, the second resource is taken by another player who needs the first player's resource in turn. In this situation the two players wait for another resource indefinitely, which causes the whole system to stop working. This potentially dangerous situation must be avoided by all means. [7, 138-139.]

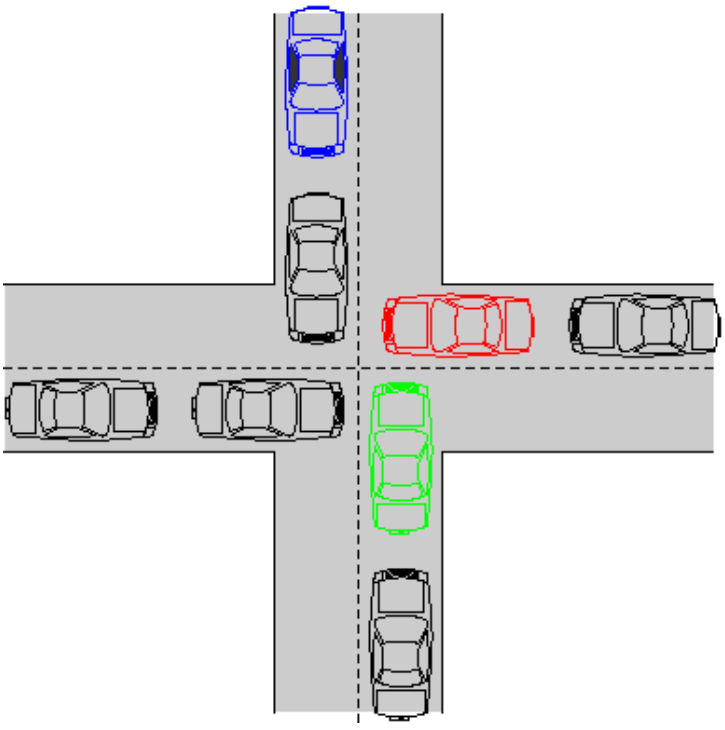

Figure 6. Abstract deadlock illustration

In figure 6, a deadlock situation is described in terms of cars. It is assumed that each road is a resource and each car is the user of a road resource. Each car can move only

forward, and, thus, has to wait until the road in front of it becomes empty. Nevertheless, in this case, none of the cars can move anymore. The roads cross and each driver occupies the road segment (resource) before the junction, waiting for the resource (continuation of the road) to become available. Nevertheless, it will never happen, and the system of cars will never continue moving. Thus, the system is in the deadlock.

The last problem related to concurrency in multiplayer game server software is throughput and cost of synchronizing the access to the resources. The synchronization can improve the performance of the game server software greatly and lower the latency of the responses from the server. However, as long as some resources might not be safe, they must be synchronized. If synchronized properly, only one player at a time might access the resource. As long as the same is true for many players, it might take a lot of time to use the resource by many players sequentially. The mentioned situation represents the safety/performance trade-off because if some of the resources are synchronized, the overall performance of the game server software will decrease. Nevertheless, it is necessary because not synchronizing the access to vulnerable resources might lead to critical errors in the program execution. [6, 6-7.]

The other important issues when dealing with multiplayer RPG server software are database access and memory management, which partly interfere with the already mentioned problems, such as concurrency. This project aims at creating a server software architecture which will be capable of handling thousands of concurrent users, who would be able to play the game online, and addressing the problems of the multiplayer games mentioned in the previous paragraphs.

# 4 Technology Presentation

## 4.1 Software Platform

Multiple technologies can be used to implement the server software for a multiplayer game. Nevertheless, it is important that the chosen technology addresses the issues which an engineer can encounter in the process of software development. There exist a number of software platforms to choose from, in terms of stability, ease of development, speed and maintainability. Nevertheless, the choice for the project is the Java platform. The Java platform includes the Java Virtual Machine (JVM), a rich set of libraries and frameworks and the Java programming language. It is open-source, easy to use and addresses the needs of the project.

There has been some amount of skepticism about using Java as the platform for game development due to concerns about its performance. Java uses heap as the primary data storage during the program execution, with limited capabilities of storing data on the stack. C++ programming does not have this limitation and any data type can be stored on the stack, though this capability is used mostly for small data types, such as arrays or combinations of primitive data types. As long as the access to the stack is faster than access to the heap, C++ might perform better in applications where a large amount of work on primitive data types is done. However, most of the data types in the current project are complex, and the difference between Java and C++ must be negligible.

Java also uses more memory than natively-compiled languages. One of the reasons is that JVM and its processes take up some memory. Another reason is that garbage collection happens in predefined intervals of time, chosen by the JVM user, so that unused objects are kept in the memory until it is cleaned. This feature is the main benefit of Java as well as its drawback, as the programmer does not manage memory directly and relies on the JVM to perform memory allocation and freeing. In figure 7, the process of freeing memory on the heap is shown. The blue area represents the space in the memory taken by the objects, used and unused. The peaks in the graph show, that Garbage Collection (GC) process has successfully run and some memory has been freed. Nevertheless, the peaks do not happen often, and the memory can remain dormant for long periods of time.

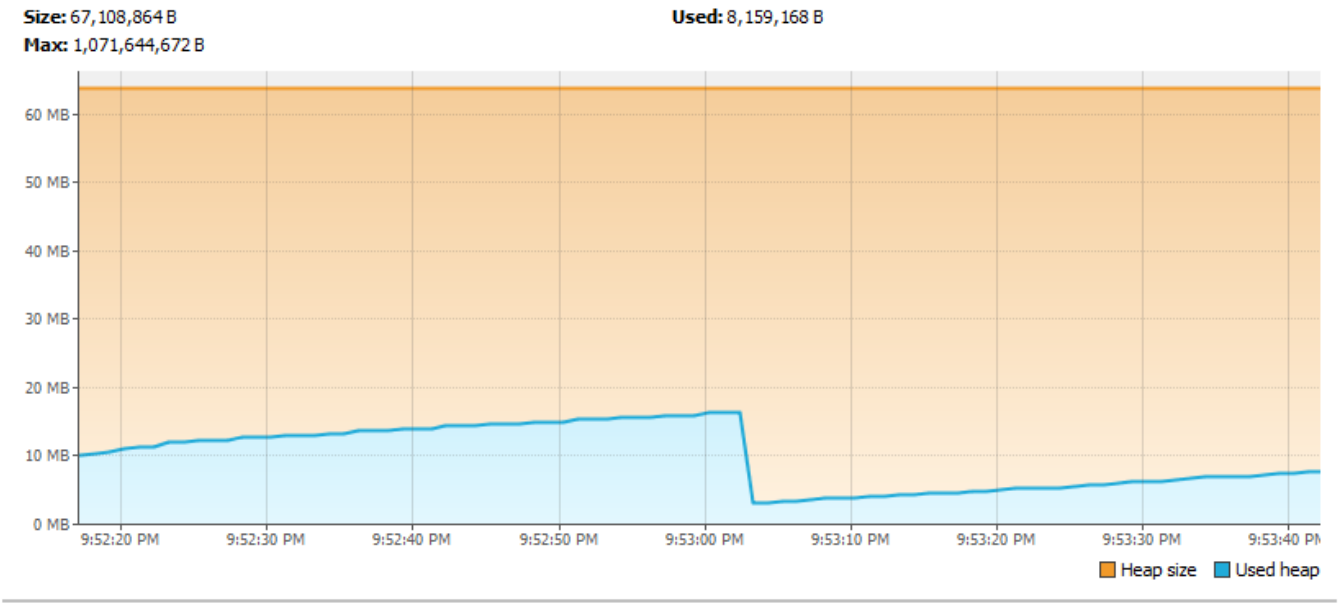


Figure 7. An example of memory, being freed by the GC thread

Another problem, which is posed by the garbage collection, is that while the unused memory is being processed, the virtual machine can stop the execution of a program for a while. Even though the time span is measured in milliseconds, it can be critical for certain programs. Nevertheless, latency is not the priority for this project, so that waiting intervals of up to a quarter of a second are accepted. Moreover, if the performance of the garbage collection is not acceptable, it can be fine-tuned to work in small intervals, thus, not stalling the execution of the program at all. [8, 178-181.]

One of the main benefits of Java platform is the ease of software development with the Java programming language. As long as the memory management is automated by JVM, the number of bugs and mistakes in the program execution drops down dramatically in comparison to natively compiled languages such as C or C++. Moreover, Java as a platform provides a broad set of open-source tools and frameworks, which further simplify the process of software development removing the burden of system programming from the engineer, leaving more time to write business logic of the application. As long as the multiplayer game server software is a large project which requires a considerable amount of work, the approach which simplifies and accelerates the software creation process is preferable. The implementation of this project is also dependent on robust development strategies and tools which can help to write software modules fast. As a result, Java is chosen over C and C++ for the game server implementation. [9, 44-45.]

Java is cross-platform and JVM implementations exist for all major types of operating systems and some hardware systems such as microcontrollers [10, 2]. In terms of this project, it means that Java can be used in conjunctions with the Linux operating system which is free. It makes Java an optimal choice over other rival technologies such as .NET, which requires investments in case the project is continued and used for earning money.

Some other important features of Java as a platform are its scalability and modularity. These qualities allow to reuse the written software and to improve the speed of development even further. The scalability feature of Java stems from the fact that it is very modular. When a new software module is implemented, it can be easily added to the working project continuously without the need to rebuild the whole system completely. In such a way very large and complex systems can be created. This feature is present in many other platforms but Java excels its rivals such as C or C++ at the moment.

## 4.2 Frameworks

After choosing Java as the main software platform, a set of libraries and frameworks are to be chosen in order to further simplify the process of the game server development. One of the most important issues is the communication between the server and the clients. This communication can be performed in multiple ways. The easiest way is a request-response communication pattern, where clients send requests to the server software frequently, querying for the update information. In this scheme, the server receives a request and sends a response back to the client along with the data about the game world state changes. The main drawbacks of this communication type are latency and unnecessary bandwidth consumption as long as many responses from the server contain no meaningful information. If there are many connections at the same time, the server channel might get congested and the latency will increase progressively. In regard to the mentioned fact, common web frameworks (such as Java Enterprise Edition (EE) web layer, Spring Model-View-Controller (MVC), etc.) cannot be used for the project as long as they represent a request-response communication pattern.

Another approach is to use a full-duplex communication channel between the server and a client. In such a way the server itself can notify the client when something important happens in the game world and the number of requests to the server decreases

dramatically. This strategy is more preferable for the current project because the game world changes seldom and it is important not to congest the communication channel of the server by unnecessary requests. The optimal choice is plain Java SE. The connections between the clients and the server are to be created using plain TCP sockets. Java Standard Edition (SE) possesses all the required classes in the distribution packages (java.io, java.net). Nevertheless, the common architecture, given in many books for beginning programmers, where Java TCP sockets are used in a blocking manner, is not applicable to large multiplayer game server software. The main problem is that each connection creates its own thread, and in case there are many connections, the memory footprint is too large for fast and reliable operations of the server software.

One of the ways to bypass the limitation on the number of simultaneous connections is the usage of NIO approach. With this approach, a new thread is not created for each connection to the server but rather a fixed thread pool is used, and, thus, the amount of connections can excess a few thousands or even more, depending on the hardware and frequency of incoming requests. The required classes are placed in the java.nio package of the standard Java SE distribution. Nevertheless, the work with those tools requires considerable programming experience and is hard for inexperienced users. In order to simplify the work with the Non-blocking input/output (NIO) and avoid possible mistakes, the Netty framework has been used in the project. This framework is used to simplify the development of high-performance, concurrent network applications, such as HTTP servers and other software which needs the support of thousands of concurrent users, and is used by many well-known organizations such as RedHat, Twitter, and Apache to name a few. [5, 2-3.]

The network layer of the multiplayer game server software is connected to the game module. The latter is responsible for the operations of the game itself. It stores the state of players and the game world. It also stores and runs the game processes such as fights or other game events. This part of the game server software is written in plain Java SE. As long as this module of the server software is shared among all the players and needs to update its state accordingly, it is highly concurrent and makes a heavy use of the tools from the java.util.concurrent package.

The game server software needs to store players` data frequently, and, thus, it requires a persistence layer which is responsible for the database access. In order to simplify the work with the database, the persistence layer of the game is tightly integrated with

the game module of the software. The Java Persistence Application Programming Interface (JPA) has been chosen for this project to speed up the work with the database and remove unnecessary complexity when storing and retrieving necessary data. JPA is an Object-Relational Mapping (ORM) framework which is a part of the specification of Java EE. It makes a representation of Java objects in the database by translating object fields into relational database table rows. This process is handled by the framework, and, thus, the programmer has to work only with the Java programming language without the need to know Structured Query Language (SQL). However, the Java platform does not restrict the user to utilizing only the predefined implementation of the JPA specification. In this project, the EclipseLink persistence provider has been used. It is a well-established implementation of the JPA specification.

EclipseLink can be used in conjunction with most Relational Database Management System (RDBMS) but the choice for this project is MySQL. It is scalable and fast, and can be used free of charge. It supports the create/read/update/delete (CRUD) set of operations, joined tables and constraints such as primary and foreign keys. These facts make it suitable for use in the project. Moreover, MySQL provides connectors for the Java platform which can be used from the Java programming language (through EclipseLink or directly) to communicate with the DBMS efficiently.

## 4.3 Development Tools

The Eclipse Integrated Development Environment (IDE) has been chosen for work because it automates the majority of development tasks and is very flexible and extensible. It also provides a convenient user interface. The next important feature of Eclipse is its possible integration with Maven. Through the Eclipse repositories, Maven plug-in can be installed through a sequence of short steps. Moreover, Eclipse is tightly integrated with JPA and easy to use to produce the documentation for the project automatically.

Maven is a software management project tool which allows building projects in a uniform way. It simplifies the project compiling, reporting, and packaging. One of the most important features of the tool is that it provides a uniform build cycle, which is always followed stage by stage from compiling to installing the packaged product into the repository. Another important feature of Maven is dependency management. When a new library is to be added to the project, Maven can automatically download it from the

repository online and include it into the local workspace. The main benefit of such an approach is that when there are a great number of dependencies or third-party libraries in the project, Maven can track their versions and update them accordingly. In case a new version of the product is used, Maven detects it and resolves the arising dependencies. It is also tightly integrated with software unit testing tools, and even provides a directory structure to place all the required tests to. Moreover, tests comprise one of the build-cycle stages of Maven and all the following stages after testing are cancelled in case any tests fail. It allows to unit-test software modules in a uniform and reliable way.

# 5 Description of the Software

In this project the server software for a multiplayer RPG has been implemented. The server consists of a few modules which together perform multiple game functions. The most important parts of the software are its networking module (controls the connections between the clients and the server), dispatcher module (connects the game module, authentication module, and the network module), game module (tracks and controls the world state), and system module (which includes the functions for data storage, message creation and other utility functions). There is also a bootstrap module, which loads all the other modules, and which represents the entry point for the software, in the bundle.

All the modules are laid in a tied architecture, where upper layers can communicate only with adjacent layers. Moreover, each module performs a particular function which is unique for this part of the game server software. Each software level communicates only with its counterparts as shown in figure 8.

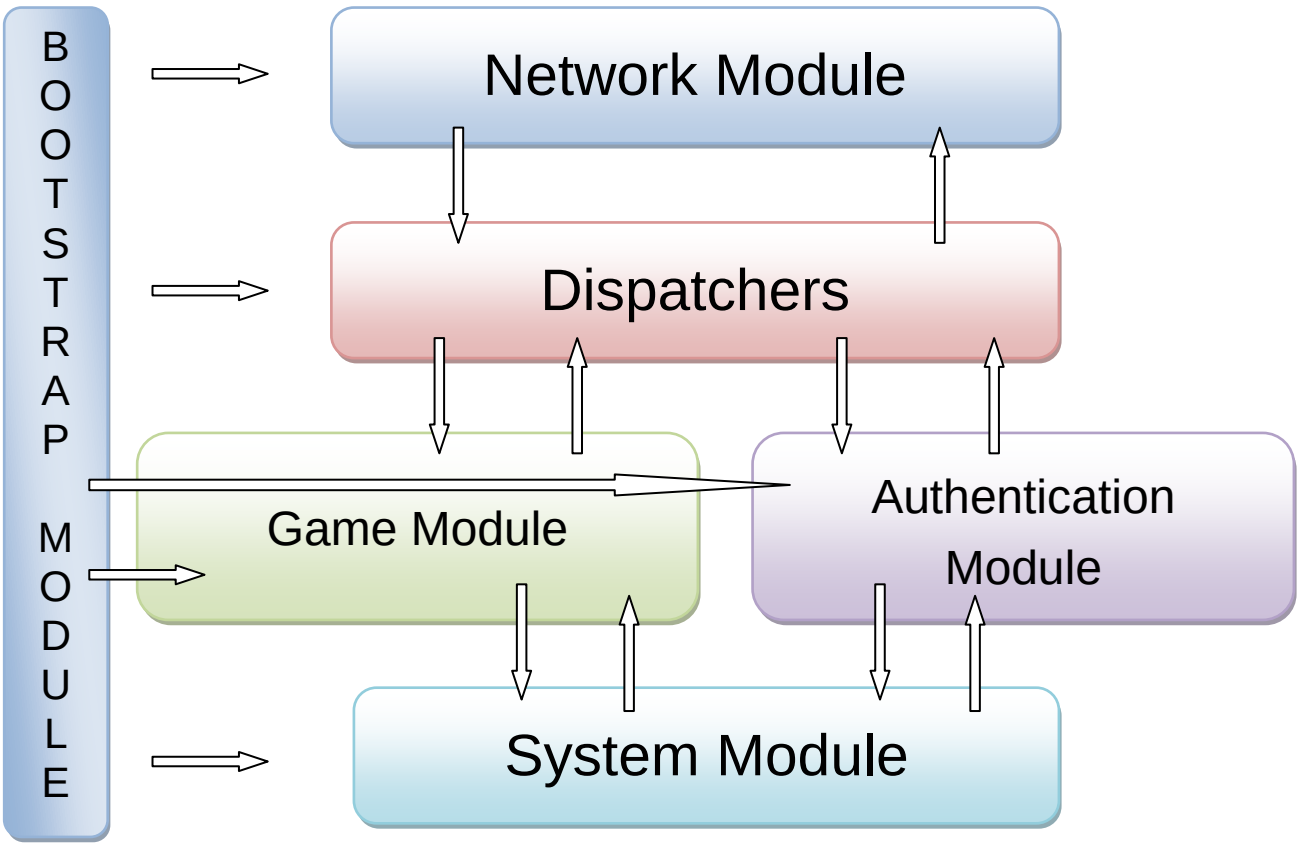


Figure 8. Tiers of the server software architecture

According to the diagram, the bootstrap module spans across all modules because it load all of them. Other modules are organized in a hierarchical manner. The system module is also partially used by all modules. However, it is prevalent in the game and authentication modules as long as they store the character data to the database.

## 5.1 Bootstrap Module

The bootstrap module is responsible for launching all other modules. It is also the module which is the entry point for the software application. When the main class is started in this module, it automatically starts the network module, initializes the game module and tools to work with the persistence (system module). The module also specifies the order in which communication channel handlers are supplied to newly-connected users, and basic Netty server configuration such as thread pools or the port on which the server would listen to TCP connection requests.

The first function that the module performs is bootstrapping the network module. When the user has specified the port on which the application is going to run, the bootstrap module starts the Netty server, allocates the listening accepting socket, specifies what to do on connection event (register decoders and encoders, user session and output dispatcher), initializes the world and its locations, and registers the JPA persistence context (through system module).

Some other functions of the bootstrap module are stopping the server, initializing the logging framework in order to store the information about possible exceptions thrown during the execution of the program and providing the console user interface for starting the server software. The application can be started through a console interface with one simple command. If there are mistakes at the server software startup (no port has been specified), the user will be notified about them in the console. In case there are no critical errors at the startup, the logging framework switches on and starts logging all the following errors and exceptions.

## 5.2 Network Module

The network module is responsible for allocating client connections and managing them. The central part of this module is the Netty framework. It represents a pipeline

which all the incoming messages pass through, and which all the messages, headed for the clients, also traverse. After accepting the connection from the client, the network layer allocates a pipeline for each client. Next, the Netty framework tracks if any bytes from the remote host have been received. In case some bytes have been received from the remote peer, they are dispatched into the user pipeline. If the connection is idle, the pipeline is stored in the memory, not consuming any Central Processing Unit (CPU) time because no thread is managing it until it becomes active again.

On the message path, the incoming and outgoing messages are being changed, to prepare them for the module of dispatchers. Decoders change the incoming requests and pass them to the request dispatcher in the dispatcher module, while encoders receive the messages from the output stream dispatcher. Encoders are used by the game module to distribute messages and transfer them to the remote clients.

The network module possesses multiple encoders and decoders which perform specific network tasks, such as tracking the amount of time for which the client has remained idle (IdleChannelDisconnector). This object is registered up and down the pipe, so that it can track the channel in the direction of requests as well as responses. In case the game client remains silent for a long period of time, it is disconnected from the server. This class also detects any errors in the processing of requests, communication channel disconnects, and special codes, which control the game, in which case it can call the cleanup methods on the player resources. The network module also checks the validity of requests from clients with one of its decoders. If the request is not valid, the connection is immediately closed.

The last decoder sends the request object directly to the dispatcher, where it can be processed. The first encoder in the pipeline is known also to one of the dispatchers, and all the messages from the server to the client are sent directly by the dispatchers` layer into the pipeline. This architecture allows for full-duplex communication between the server and its clients.

As long as the request-response system of the game server software is asynchronous, the sequence diagrams represent the order of actions for the request and the response separately. During the request phase as seen in figure 9, the request is first sent to the ByteToStringEncoder. This part translates the bytes into arrays of readable characters in UTF-16 encoding. Then it sends the character set to the StringToRequestDecoder

which constructs a Java request object out of the given character data. RequestProcessor decides to which dispatcher the particular request should be sent to.

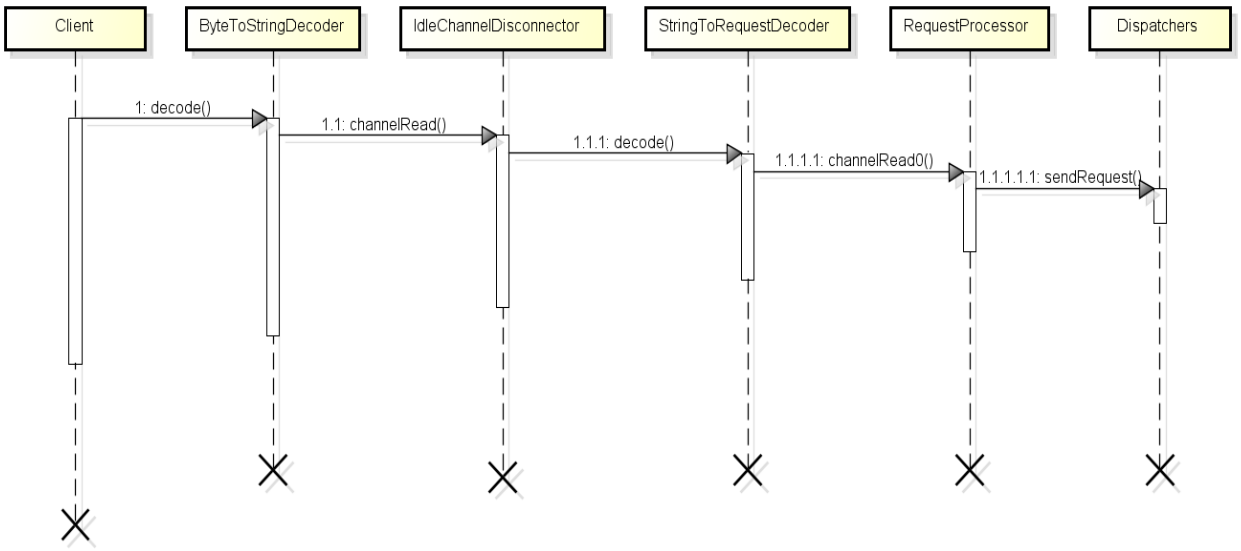


Figure 9. Sequence diagram for network module request processing

When the response is sent to the client asynchronously, one of the dispatchers constructs a response object, and calls the encode method of the ResponseToStringEncoder. This method turns the Java object to the character array. Next, the message is sent to the StirngToByteEncoder, which encodes the response into bytes and dispatches it to the client. This process is illustrated in figure 10.

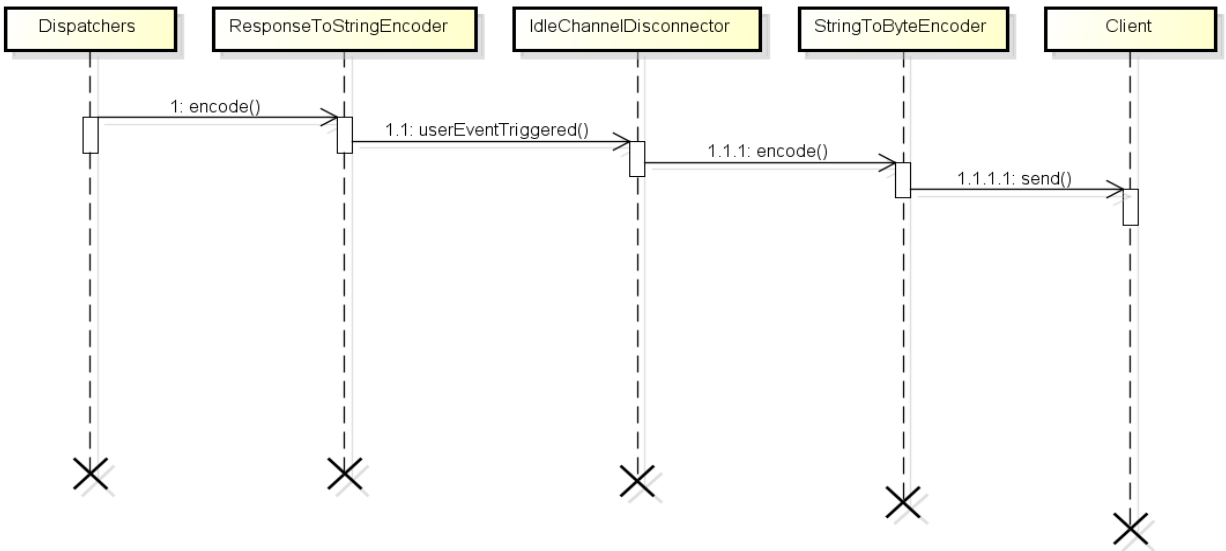


Figure 10. Sequence diagram for network module response processing

The network module also has the hooks to the cleanup routines. In case the disconnection is detected, the network module calls the methods to free the user resources. It also catches multiple exceptions connected to the network layer operation. These errors are logged, and the recovery mechanism is launched. These functions are stored in the IdleChannelDisconnector, which does not do anything on normal request or response processing. Nevertheless, if an exception occurs, or the channel is idle for too long, it automatically starts the mentioned processes.

## 5.3 Dispatcher Module

The dispatcher module is responsible for calling the methods of the game module according to the requests received from the network module. It is situated right between the game world and the connection to the client. When the first request is received from the client, it is always dispatched to the authentication module. If the registration/login has been performed successfully, all the subsequent requests are forwarded to the game module. However, all the dispatchers are allocated through main request dispatcher which has no state but controls the access to dispatchers dynamically. All the requests always flow through the main request dispatcher.

The central concept to the dispatcher module is the player session. The session is allocated already in the network module on connection but some of its fields are not populated yet. When the connection is initiated, the session is given an instance of a new player. Next, the session stores the instance of the output-message dispatcher, which is also created on connection allocation. After this, the session becomes available to all the other dispatchers through the incoming requests. After a successful login, the session is given an instance of the game dispatcher, which stores the in-game character of the player and calls the methods to control the processes in the game world.

A character controller is the dispatcher responsible for the game functions. It has an internal access to the player's session and, thus, to the player's output stream. When initialized, it retrieves the character from the game world and initializes it. In case the first stage is completed successfully, the class registers the output dispatcher for the game character so that all other players can communicate with this particular player. The system of controllers is complicated and shown in figure 11.

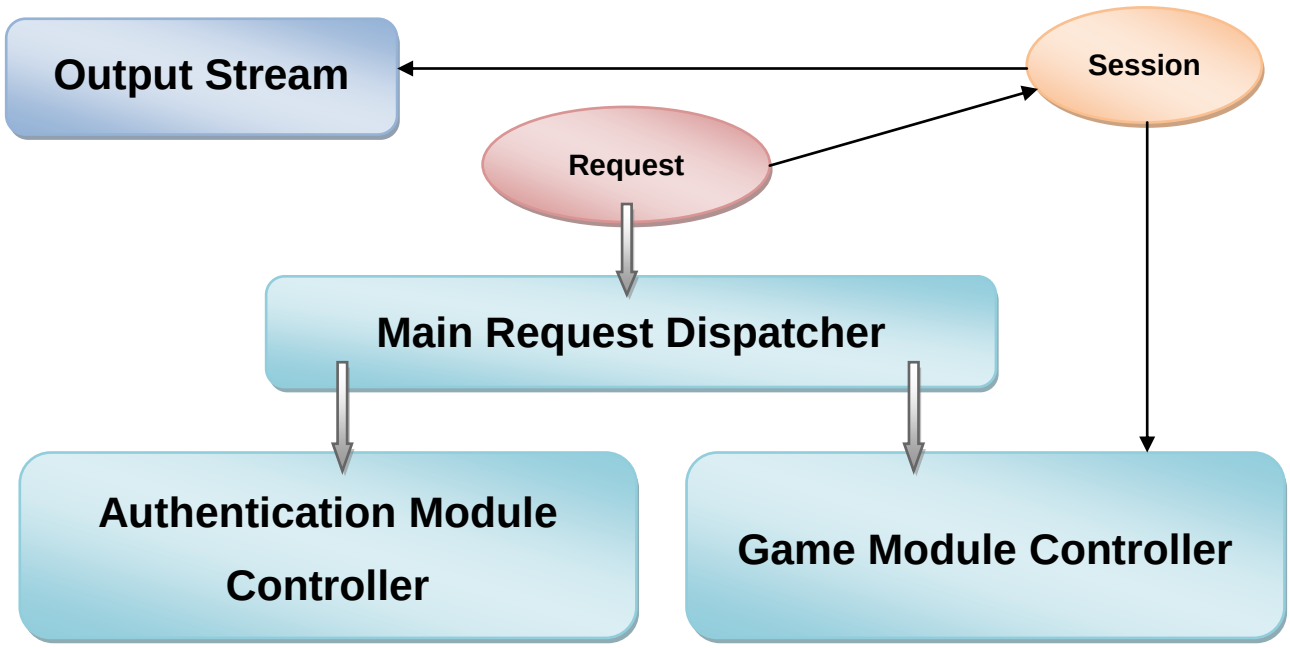


Figure 11. Dispatcher module structure

As seen from figure 11, the dispatchers are combined into a complex structure. Their responsibilities are fully separated but it is often required to use multiple dispatchers at the same time. Output stream represents the dispatcher which is responsible for sending messages to the clients. All the dispatchers have an access to the output stream through the client session. Any request from the user carries a reference to the session instance. The main request dispatcher decides which dispatcher to invoke according to the incoming request, and when the decision is made, the session object is passed over to the dispatcher along with the request.

## 5.4 Authentication Module

The authentication module is separate from the game module because they perform totally different tasks. The purpose of the authentication module is to decide whether a client, trying to connect to the game module, is eligible to do that. The authentication layer uses persistence in order to save and retrieve the player credentials from the database. Moreover, it checks if the user credentials correspond to any other credentials stored in the database. In case the match is found, a signal to the request dispatchers is made in a form of changing the state of the player object inside the player session.

When the first request is received from the main request dispatcher, the authentication module considers it to be either a register request or a login request. In case it is a register request, the module checks if it can register the player. If it can do that, the registration is performed, and the user is automatically logged in. In case a client sends a login request, the authentication module checks if the credentials are correct. If the decision is positive, a new game controller is created and stored inside the user session. The user state changes to “logged” automatically.

## 5.5 Game Module

The game module of the server software is responsible for managing the game world and the players` characters. It has a vast number of functions, and is includes persistence functionality. When the user is authenticated, the following requests are decoded and sent to the game controller by the main dispatcher. The game controller can understand what particular requests mean and invoke them on appropriate game objects.

### 5.5.1 Game Request System

There are a few types of requests for the game: global requests, location requests, character requests, and fight requests. Each type of request performs its functions if the player character is in the right state. The character can exist only at one state at the moment as seen from figure 12.

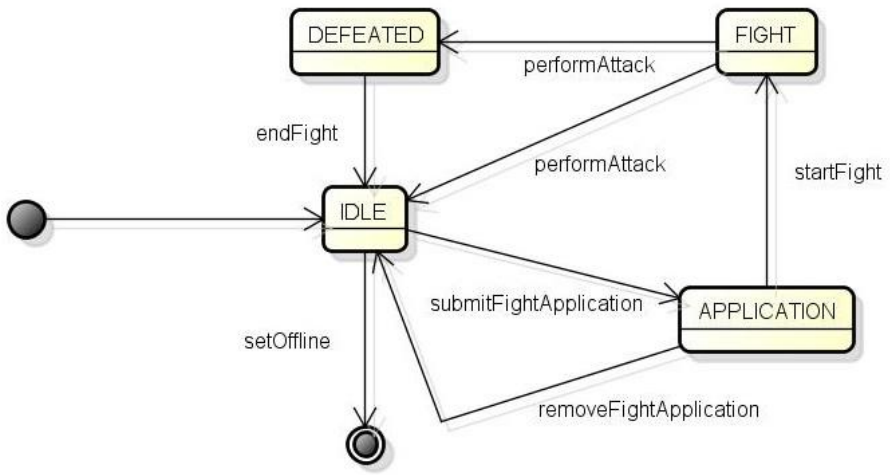


Figure 12. State diagram of the player´s character

The global requests represent the functions which can be accessed at any point in the game, at any time, and, thus, they can be called from any state of the character. These requests include the request to log out or to get the information about another player. Another type of requests is the character requests. They relate to the character changes such as moving to another location, or equipping weaponry, and can be performed only when the character is idle.

One more type of requests is location requests. Each location has a list of fights available at a particular location. As an example, training room has an ability to create new fight applications and start new fights through registered applications. Every location has a command which returns a list of players available at the particular location. Weapon shop has a command which is used to buy new weapons or to list the stock of the shop. All the location command can be issued when the player is in the idle or application states.

The last type of requests is fight requests. These commands can be issued only when the character is in the fight with other players. Fight requests can inform the character to attack another character in the fight, or to use a special ability on the player's allies or enemies. As an example, a special command exists for attacking the enemy in a fight. In case one of the fighters has been idle for too long, a fight-finish command can be sent. This command instantly kills the last remaining enemy in case he is the last one in the enemy team. When the player has been killed but the fight is going on as long as there are remaining players, the player`s state turns into defeated, and he can no longer issue any commands in the fight until it is over. When the fight is over, all players go to the idle state.

### 5.5.2 Game Functions

There are a number of functions that a player can perform in the game. The player's functionality specifies what can be done in the game world. For the game world, three locations have been implemented (training room, barracks and castle) as singletons [11, 170-177]. When entering the location, the player receives the list of players` characters present in the location, and the player will also receive the messages about other characters entering or leaving the location. In the training room, an application can be submitted. Application represents an invitation for other players to fight with the

character, applying for a fight. Any of the characters in the location can accept the fight. In this case, the fight can be started.

The fight represents a step-by-step turn-based action. When the user chooses where to attack his opponent and chooses which body parts he wants to protect, an attack will be issued. Next, the player needs to wait for the response of his opponent. When the opponent responds, the system will calculate the probabilities, and decrease the hit points of the players who take damage. The team where no players are left alive is considered to have lost. When the defeat is registered, the player's state will be changed and he will be returned to the location where he will be able perform non-fight functions.

The player can issue the commands to get to know the information about other players at any time. In case of such a request, its target is located and some of its data is transferred to the inquiring party. The character can also move around the locations of the game world. Every location has its own set of functions available only in that location.
One more feature of the system is the state tracking. When the user logs out, his character is persisted in the database. Nevertheless, if the character is in the fight, he will not be saved until the fight is over. It allows the users to reconnect in case their connections are broken.

# 6 Project Outcome

## 6.1 Project Results

The minimum requirements of the server software for a multiplayer role-playing game were fulfilled during the project. The implementation provided the features to login/register and play the game, and to establish and synchronize communication among multiple clients in a concurrent manner.

The features which belonged to the minimum requirements for the software were all implemented. The business logic of the software was of the utmost importance. Multiple objects (Fights, Locations, etc.) needed complex procedures in order to work with many clients at the same time. As an example, fights required a complex set of timeouts, running in separate threads. This detail posed serious obstacles in implementing the software.

Another difficult implementation feature was that the world had to track the state of active fights, and in case one of the players left the game, his resources had to be cleaned up automatically. As long as players attacked at different times, and many actions depended on the enemy or ally, the synchronization of the fight became difficult. In addition, game message distribution was hard to implement. The messages had to be supplied to the right players at the right time. This fact laid even more restrictions on the logic of player request processing. Nevertheless, the server software was built and it provided the required functionality.

## 6.2 Discussion

One of the most difficult parts of the game server software was the network module. This module served as the basis for the game and its entry point, which resulted in a limited set of technologies which could be chosen for this purpose according to the given requirements. Moreover, the server-client communication model was in the focus of thorough attention from the beginning of the project since it was the main entry point of the server software.

During the process of development, the Java EE request-response model for the client-server communication was rejected because it did not fulfill the requirement stating that all clients had to be immediately notified about the events taking place in the game world. In this model, the requests had to be made frequently in order to retrieve the new information about the world state. Nevertheless, this approach could congest the network bandwidth to the server because many requests would not carry any payload. In order to avoid it, some delay should have been set on requests. However, a pause between requests would inevitably lead to the delay problem, which emerged when the client game state was not synchronized properly with the game world state.

In order to solve the problem, a full-duplex communication link between the client and the server was created. The link worked over TCP-socket and sent data in both directions. The drawback of this approach was that much memory and CPU power was wasted in an attempt to save a thread for each player in the main memory of the game server software. With this technology, the capacity of the server was around one thousand users at most. Nevertheless, a better approach was chosen for the project. The asynchronous non-blocking input/output was used. This technology helped to scale the server software easily, consuming a few times less resources than the original approach with plain TCP sockets and one thread per connection. The introduction of the Netty framework helped to scale the server software and make it meet the requirements.

The other important part of the game server software was the shared world for many players and authentication/reconnection functionality of the implementation. In order to achieve the safe use of shared resources among players, a large number of constraints were introduced in the software. The implementation required considerable amount of research in the field of concurrent programming and design patterns. The problems which were still present in the server software implementation were its shared-resource bottlenecks related to the atomic execution of parts of the program. As long as some objects were prone to concurrency modification exceptions, large chunks of code could be executed by only one thread at a time which resulted in a queue of requests waiting for the execution. In case a guarded object was used by too many clients, the responses could take a noticeable delay before being processed, resulting in high latency for the players. The described problem still needs to be resolved in the future releases of the implementation.

Another interesting part of the implementation was related to the database access. Database operations took a long time to complete and this fact had to be managed in some way by the server software. The solution was to perform the database access as rare as possible. In fact, the database access happened only at the moment when the player was leaving the game. It allowed reducing the number of database operations with the player to two in one session: game character retrieval and game character saving on exit. Nevertheless, the performance of the database operations might become more important in the future because there were many other operations which were going to be implemented, and needed the database access. The performance can be critical in the future. This issue is the main drawback of the software at the moment.

In the project I learned several important concepts related to concurrency and networks. These fields of computer science were especially applicable to the servers of multiplayer role-playing games because they related to the basic problems and aims of this type of software. In the project I designed and implemented the whole system described in the previous chapters and from my point of view, the greatest benefit of it was that I learned to plan and implement large projects and maintain them. In the course of working, I faced a number of serious challenges which I successfully overcame. In the future the project is to be developed further. As a result, the persistence and the concurrency of the game should be improved. As for the game features, new locations and advanced functionality are to be added into the future releases.

Some of the new features might include such location as auction or battle grounds. There are going to be artificial intelligence fighters in the game. In order to make the game more interesting, a set of quests and a broader range of weapons and inventory items could be introduced. In the future, the server software would be distributed over multiple nodes. One of the nodes would provide the game services, the other one would provide authentication services, and the last one would serve static context to clients such as images or sound. A separate server might be introduced for chat and related services. Moreover, the payment system might be viable in case the software was published online. With regard to this, encryption might also be applied to the communication protocol.

# 7 Conclusions

The goal of this project was the creation of server software for a massively multiplayer online role-playing game. All the aims were achieved and all the minimum requirements fulfilled. As a result, a working server software implementation was created.

Among the features implemented in the project were the network module, which connected (reconnects) and disconnected clients from the server, the authentication module, which controlled the login procedures, and the game module which represented the game world with the players` characters inside of it. The technologies used in the project included Java and multiple related frameworks to simplify the work with the databases and to improve the network module. In the process of choice, multiple programming languages and platforms were compared to each other, and the most appropriate one was chosen. The technology of choice was dictated by the necessities of the project and the speed and ease of software development.

During the implementation of the program, multiple obstacles were met. The most difficult ones were connected with the concurrency and persistence. Some of the problems were not fully resolved but only to the level when it did not stop the software from achieving the declared goals. In the future releases of the software, most of the problems were to be eliminated and new features added, so that the project would continue. Overall, the project was an interesting and difficult challenge through which I learned a large amount of new material about advanced computer science concepts such as concurrency, design patterns, or software architecture of large systems. Moreover, during the implementation of the project I acquired skills which could be necessary in my future career. All the above facts make this software unique and it required a solid amount of work.

## References


1. Jeannie Novak . Game Development Essentials: An Introduction, Third Edition. USA, New York: Delmar, Cengage Learning ; 2012.

2. Mike “MrMike” McShaffry and David “Rez” Graham. Game Coding Complete, Fourth Edition. USA, Boston: Course Technology, Cengage Learning; 2013.

3. Elliotte Rusty Harold. Java Network Programming, Third Edition. USA, Sebastopol: O’Reilly Media, Inc.; October 2004.

4. Andrew Davison . Killer Game Programming in Java. USA, Sebastopol: O’Reilly Media, Inc.; 2005.

5. Norman Maurer. Netty in Action, Manning Early Access Program , Version 4 . Manning Publications; Estimated release date: May 2014.

6. Brian Göetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea. Java Concurrency in Practice. Addison-Wesley, 2006.

7. Scott Oaks, Henry Wong. Java Threads, Third Edition. USA, Sebastopol: O’Reilly Media, Inc.; September 2004.

8. Bruce Eckel. Thinking in Java, Fourth Edition. USA, Stoughton, Massachusetts: Pearson Education, Inc.;November 2010.

9. Jonathan S. Harbour. Beginning Java SE 6 Game Programming, Third Edition. USA, Boston: Course Technology, Cengage Learning; 2012.

10. Kathy Sierra, Bert Bates. Head First Java, Second Edition. USA, Sebastopol: O’Reilly Media, Inc.; February 2005.

11. Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, Elisabeth Robson. Head First Design Patterns. USA, Sebastopol: O’Reilly Media, Inc.; October 2004.