Project 14: Design och konstruktion av grafeditor

User Requirements Document

Project submitter: Swedish National Defence College

Project Group nr 20 Aliquid

Simon Lindholm, Erik Lindström, Andreas Linn, Martin Mützell, Johannes Olegård, Anton Olsson, Viktor Palmkvist, Daniil Pintjuk, Frans Tegelmark, Sean Wenström.

- Version number and document status
- Version 2.0.2
- Date of printing 2014-04-30

Abstract

STRATMAS v7.5 is simulation software intended for decision aid in crisis situations like war and natural catastrophes. It needs improvements to be usable and a new name to convey its difference to other STRATMAS projects. Infrastructure such as road, waterpipes and communications are needed to simulate more advanced countries, war zones and evacuation scenarios. All these features are best represented as graphs and as such need the same kind of abstractions and GUI representations. This is what we will implement in this project.

Table of Contents

Document Change Record

- 2014-02-20: Created document. Added structure, comments, TODOs and some text.
- 2014-03-02: Pretty much everything, version bump 0.1.0
- 2014-03-04: Finishing touches. version bump 1.0.0
- 2014-03-29: Added feedback as comments and fixed them
- 2014-04-28: Changed some required requirements to nice-to-have
- 2014-04-28: Version bump 2.0.0
- 2014-04-28: Added GUI description etc
- 2014-05-15: Version bump 2.0.2, small fixes after response

1 Introduction

1.1 Purpose

The purpose of this document is to clarify what this version of the software based on STRATMAS 7.5 is supposed to do and how it should work. Who will use it? How will they use it? What “features” does it need to have? Where will it run? Can it run there? And so on. It aims to be a document to clarify the users needs and divide them into manageable chunks. It will be used between the customer and programmers to be sure that what they are doing matches the customer and users expectations.

So this document is basically a detailed description of what we are going to build and how it should work. Think of it as an informal contract between the project group and the client.

1.2 Scope of the software

STRATMAS v7.5 is a client-server simulation software that visualizes and simulates a warzone catastrophe area.
This project intends to extend STRATMAS v7.5, with a “graph editor” to support infrastructure such things as roads, power lines, water pipes, etc.

1.3 Definitions, acronyms and abbreviations

Here is a list of words that appear in this document and short descriptions of what the intended meaning is.

- STRATMAS: STRATegic MAnagement System. The name of the different simulation software developed at the Swedish National Defence College. There have been many different versions of STRATMAS (not necessarily related to each other except in concept) and this project intends to extend one of these versions (v7.5).
- Scenario - Version of the simulation, often a country or crisis zone.
- JOC - Joint Operation Command, the proposed users of this software.
- Agent - A major mobile entity in the simulation e.g military platoon or terrorist milita.

- Graph: A graph is a set of “nodes” and “edges”. A node is any object and edges describe how these nodes are connected. Graphs are very generic and can be used for describing virtually anything.
- Node - Abstract, connection points for edges, e.g water pumps or road split.
- Edge - Abstract, connections between nodes, e.g the roads, waterpipes or powerlines.
- Graph editor: A graphical user interface for creating and editing graphs (here: for STRATMAS).
- GUI - Graphical user interface, the visual representation of the software and simulation
- Server - The simulator part of the software, calculates the outcome of a scenario for the client to display
- Client - The GUI part of the software, connects to the server
- XML - eXtensible Markup Language. A way of formatting text-data so that it is as easily readable and modifiable by people as by machines. XML is used to for storing simulation parameters for scenarios.
- TacLan - TACtical LANguage. An XML-based language for describing the parameters used in the simulations to specify simulation scenarios.
- Java - A programming language used in this project; used for the client.
- C++ - Another programming used in this project, used for the server.
- Cell - Small part of the simulation world. The simulation world is divided into a grid of cells.
- XSD - XML Schema Definition, an xml file defining how other xml files are structured. Used for verifying data keeping it consistent and compatible with the program(s).

1.4 References

[1] Full system description
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/system-description.pdf?raw=true
[2] Model details
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/reference/stratmas-models.pdf?raw=true
[3] Client documentation
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/StratmasClient/client_user_manual.pdf?raw=true
[4] Server documentation
https://github.com/sootn/aliquid-stratmas/blob/master/doc_v7_5/StratmasServer/server_user_manual.pdf
[5] Taclan2 xsd
https://github.com/sootn/aliquid-stratmas/blob/master/schemas/stratmasProtocol/taclan2sim.xsd

1.5 Overview of the document

This sections attempts to provide a birds-eye view of the document tailored for specific roles. See also the table of contents. This document is divided into three major sections: 1 Introduction, 2 General Description and 3 Requirements. The first contains a summary, the

second gives a more general description of the project and the third contains details. As such most parties will want to read the second section to get a overview and then dive into specific parts of the third according to role.

The Customer and End-users are will probably be most interested in section 3.1 which basically contains the “features” of the software.

Developers and technical roles will probably have to read all of section 3. If only the technical aspects are of interest, see section 3.2.

2 General Description

2.1 Product Perspective

STRATMAS v7.5 is not in active use, but the idea is to support the so called Joint Operation Commands in their decision making. The JOC plans military intervention in foreign countries. We want them to be able to test different scenarios and get an idea of what could possibly happen that they hadn't thought of beforehand. Today the JOC use analog maps, whiteboards and clear film. One of the main reasons STRATMAS is not used today is lack of simulation capabilities of important infrastructure like water supply, roads, communications, etc. All these come down to the lack of graph abstractions in the client and server software. Today all major agents are moving in straight lines, at the same speed over mountains and through rivers, as if flying (although transportation means like helicopters would be rare in reality).

2.2 General Capabilities

This project focuses on allowing the user to create and add things to scenarios, things that can usually be represented as graphs. As such functionality to to do this is required on top on other UI capabilities and the server needs to understand and use these features. These features are graphs based on nodes and edges. Agents will be able to use edges marked as roads and rivers to travel faster and the state of "cells" will be modified by edges and nodes marked as waterpipes, communication and electricity.

With simulations on infrastructure the JOC can reason about more important strategic positions and how and what to protect. These infrastructures are incredibly important not only in war but in other crisis as well.

The the client wants a fast graphical user interface in order to simulate and visualize their plans. The plans include troop movement in order to counteract enemy attacks or natural disaster of different kinds. The interface needs a facelift to present the simulation in a more approachable way to convince user to use the software. Making minor enhancements like implementing a more native look and feel with java's existing libraries is easy compared to the improvement in user experience.

As infrastructure is a huge topic and far more advanced than we can ever implement there is a lot of interesting "nice to have" features which are possible to implement, but take time. Like area nodes of mobile phone towers, communication delay, lack of information and miscommunication. Features that we won't have the time to implement but if time would allow it we could, or at least prepare the capabilities for future development of such features.

2.3 General constraints

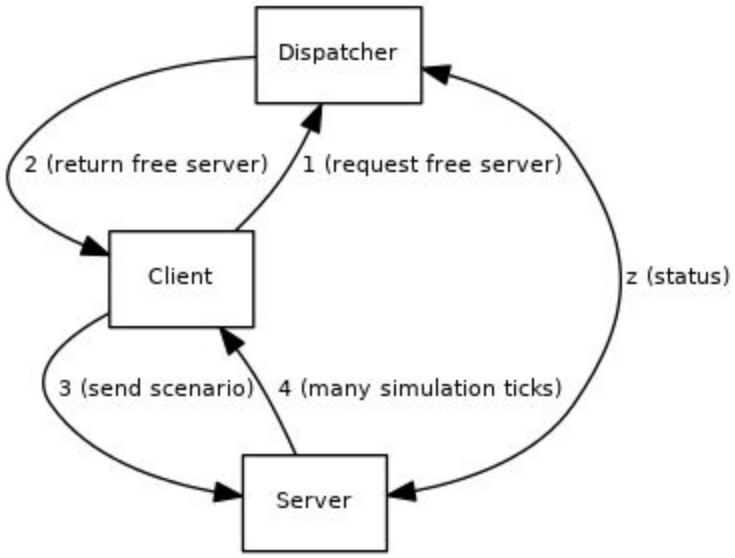
The simulator is meant to aid the JOC in their decision making and, given a set of parameters, show what *could* happen and maybe inform the user about a possible scenario or important detail he or she forgot or missed in their analysis.

Our customer has a lot of great ideas how to improve on STRATMAS but as time is limited we will focus on simulation and visualization of infrastructure.

Adding infrastructure will require a lot of changes to GUI and the underlying implementation of agent behaviour. We have decided together with the customer not to make this process too advanced and to avoid things like AI or other decision making apart from e.g. simple pathfinding.

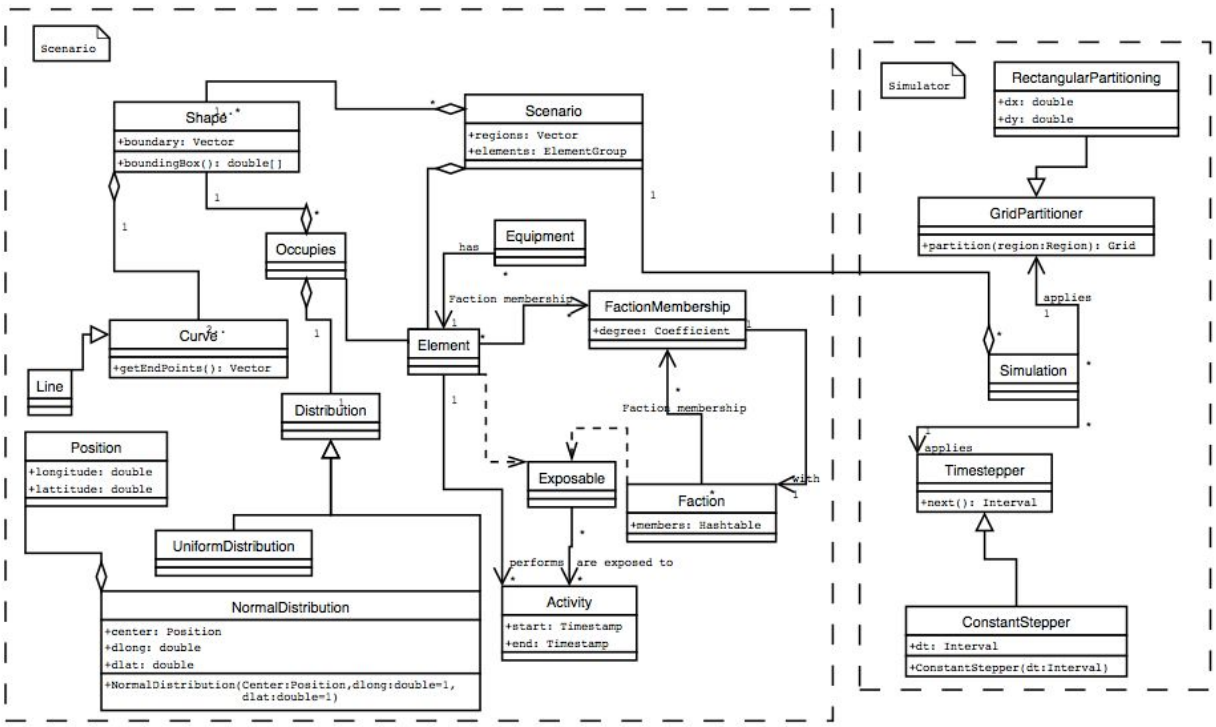
All scenario files and data sent between client and server is defined by the TacLan protocol using xml schema files. The software also contains a “dispatcher” that keeps track of alive servers and their burden and helps allocate a server then a client wants to simulate a new scenario.

Here is a view of a network situation:



The dispatcher starts first. When the server starts it registers itself with the dispatcher. The dispatcher then starts to continually ask the server if it still is online. When the client starts it may connect to the dispatcher. When the client wants to simulate a scenario, it asks the dispatcher för a free server. The client can then connect to the server, send the scenario and start simulating. Simulation is done one “tick” at a time, i.e. a short duration is simulated each time. The client asks for ticks, and the server simulates each tick and returns the results.

Information sent between the server and client is in the TacLan protocol, including abstract shapes of process variables and agents like groups of soldiers or important activities of factions in the crisis. See [2] for a description of parts and Taclan is defined in XSD [5].

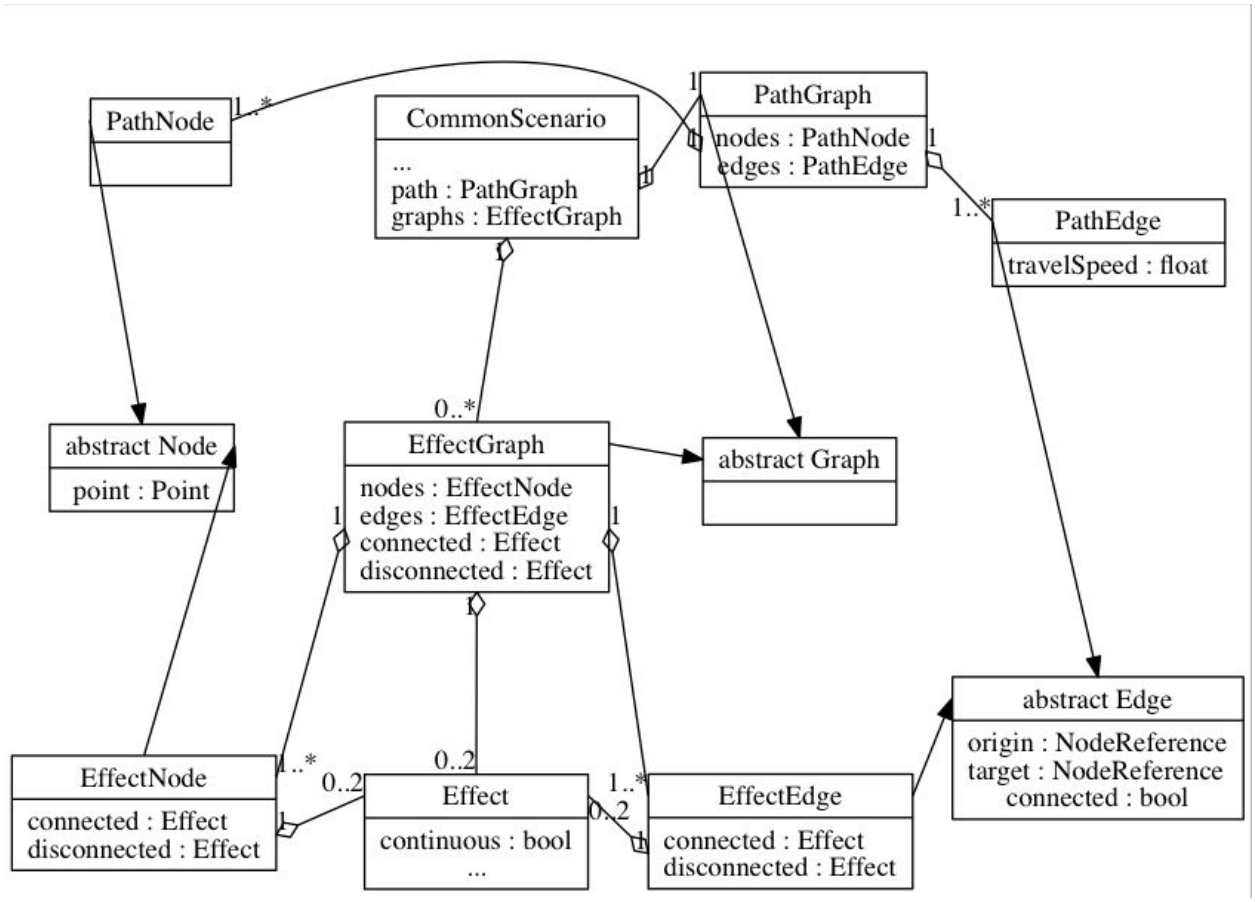


This schema is a schematic view of the data model employed in the stratmas client to represent the TacLan protocol from the XML sent from and back the server. For more information take a look in the Full System Description [1].

The “top” of the data is the Simulation (center right). The Simulation contains everything the server needs to know in order to simulate. The simulation contains three parts: GridPartitioner, TimeStepper and Scenario. The GridPartitioner explains how the simulation-world is divided into cells. The TimeStepper explains how the simulation is divided into “ticks”, i.e. small units of time that the server can simulate one at a time. Both GridPartitioner and TimeStepper each have only one implementation each: RectangularPartition (the grid consists of rectangles/squares) and ConstantStepper (each tick has the same length). The Scenario contains the simulation world; this includes Shapes and Elements. Shapes are basically polygons on the map, used to describe areas. Elements may be a lot of things, e.g. MilitaryUnits. Units can do things (Activities) belong to Factions, occupy space, etc.

Most of the simulation- and scenariodetails are not of interest in this project, as we are not going to change them, but rather add things. See below.

Our addition to the system can be described by our additions to TacLan, which is shown in brief below:



Graphs consist of two things: nodes and edges (between those nodes). We want to implement to kinds of graphs: PathGraphs and EffectGraphs. PathGraphs are basically roads, i.e. Units can travel along PathEdges between PathNodes to speed up travel. EffectGraphs are used to distribute “Effects” across the areas along the graph. An EffectGraph could represent a system of waterlines, powerlines, etc. An Effect might be “lack of water”, “lack of power”, “watrepolution/desease”, etc. For example cutting an EffectEdge may lead to EffectNodes down the line being disconnected from the net. If the graph represents powerlines, this means that the areas around the cut-off EffectNodes lose power, which in turn may lead to decreased quality of life among the residents of those areas.

2.4 User characteristics

The proposed end users is the members of the Joint Operation Command. We don’t expect these the be particularly technical and as such the GUI needs to be easy and responsive and most details hidden. Writing XML by hand is out of the question, every feature needs to be available from the GUI. Compilation and advanced installation processes will be hard for these user to follow, a good installation process is crucial and will be fairly easy to create for the java client and the c++ server with the use of pre-existing solutions. We can trust a sysadmin to

install the server. We could include the server in the client distribution and run it in the background, computers today are way faster than they were in 2006 and our tests show that you don't really need server hardware to run the simulation. This may be a viable option.

The proposed end users, the Joint Operation Commands know little to nothing about computers. As it stands today starting the server from the commandline is out of the question. As inexperienced computer user we do not assume they will even read the documentation given to them. But hopefully they will try to right click something and find the menus and buttons we introduce. As there is no more installation of the client except downloading and double clicking the java jar file (if the admin have correctly installed the latest java) the client should be a breeze to at least start up.

The sysadmins do need to have basic know how of Linux or Cygwin to install and run the command line-only server.

2.5 Assumptions and dependencies

We assume that simulation that we got are at least a good approximation of reality and that the theory behind it have got some serious research done.

We also assume the simulations that will be done are small and our pathfinding and state changing algorithms are fast enough to be run on the modern machines that the JOC have access to.

2.6 Operational environment

The Swedish National Defence College mainly uses a well defined standard environment of Windows machines, but also Apple Macintosh OS X and Linux servers like Ubuntu Server LTS. As such the software should have support for all these environments. We target Java 1.6 on the assumption that all machines are able to install and run java 1.6 programs.

2.7 GUI

As our task is to implement a new UI in an old environment and our focus is on keeping consistency with the rest of the system. We reused much of the existing classes and tools in the GUI when creating our new interactive elements, while trying as hard as possible to behave in the same way as the rest of the system. Even if these may not be the same interactions as if we implemented the system from scratch. Small parts of the applications can't behave in a different way than the rest of the system. The user will avoid that part or miss features when things do not behave as the rest of the system. For these reasons we made the choice to implement adding and removing edges and nodes with right click menus, just like inserting everything else in the system and left double click to change properties in a new window. All of these interactions are inspired and behave exactly like activities, events, camps and agents in the rest of the system.

Right click on node -> menu with choices: remove node, add edge, connect to new node

Double click on node -> properties
Right click on edge -> menu with choices: remove edge
Double click on edge -> properties
Right click in empty space -> menu with choices: create node.

We made the decision to have round nodes because everything else in the system are squares, making the nodes visible even if there is a lot of clutter on the screen. Just like it is important to have the same interaction as the rest of the system we distinguish our parts with these round features to signal that they have a different meaning. Even if the node is on the same spot as another symbol you can still see the node. Green and blue complements each other, a good reason to have green edges on the already blue background.

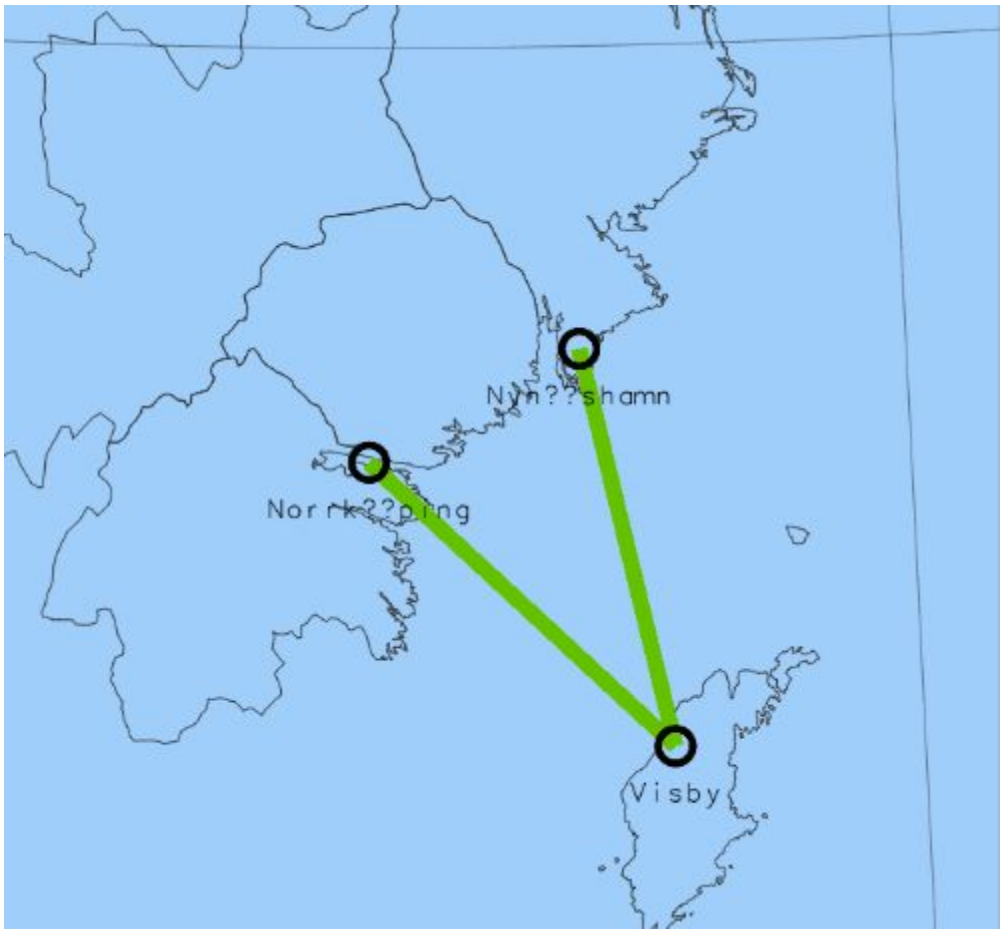


Fig: Simple scenario to simulate the Gotland Ferry moving people from and to Visby.
Here the gui implementation is not yet simulated on the server.

3 Specific Requirements

In this section and its subsections, we present specific requirements and features that will be, may be, and won't be in the final product. Each requirement consists of a table listing the following attributes:

Identifier: a short name of the requirement.
Requirement Description: a short description of the requirement.
Justification: a explanation why this requirement does/doesn't exists.
Need: "required", "nice-to-have" or "unrealistic". How likely it is that the requirement will be in the final product. Required means the it must be in the final product. Nice-to-have means that it could be in the final product, if there's time. Unrealistic means it is deemed unlikely to implement in the given timeframe.
Priority: "high", "low" or "medium". High if the requirement needs to be done early, low if it is acceptable for it to be finished late. Medium is somewhere in between. This attribute has to do with the order the requirements might be implemented.
Stability: whether or not the requirement is likely to change in the future.
Source: who requested/suggested/created the requirement.
Verifiability: explanation on how to verify that the requirement (1) is in the design (2) works as intended.

3.1 Capability requirements

This section and its subsections contain functional requirements, i.e. what the final product should be able to do.

3.1.1 Supported types of Graphs

The following should be possible to express in the simulation (using graphs).

3.1.1.1 Roads

Identifier	1-GRAPH_ROADS
Requirement Description	Should be able to (1) create "roads" in the GUI, and (2) display them in the GUI, and (3) simulate them in a meaningful way in a scenario. Units should e.g. use path-finding along roads.
Justification	The existing linear movement across the simulation world is far from reality, with agents following roads the location of the agents will be more "true" to what could actually happen.

Need	required
Priority	high
Stability	stable
Source	Client
Verifiability	We are able to simulate a agent e.g platoon moving across the map following a road created through the GUI. The road needs to diverge and the agent takes the shortest path to its goal.

3.1.1.2 Water, tele and power lines

Identifier	2-GRAPH_LINES
Requirement Description	Should be able to (1) create water, tele and power lines GUI and display them in GUI and simulate them in a meaningful way in a scenario. Should be able to (2) simulate water-contamination and their consequences in disease spread and other static changes of cells in the simulation.
Justification	The health and quality of life is very dependent of access to clean water and power. And these properties are one of the main goals of the simulation.
Need	required
Priority	high
Stability	stable
Source	Client
Verifiability	We are able to simulate a city's dependencies of water and power. With the lack of these see how the city's cells degrade in quality of life.

3.1.1.3 Communication and information spread

Identifier	3-GRAPH_COMS
Requirement Description	Simulate the “worldview” (what the agent perceives the state of the simulation to be) of every agent and update it by simulating communication through organisation structure, known information, scanners, internet, radio, miscommunication, use of communication, etc.
Justification	This would enable simulation of more realistic agent behaviour.
Need	Unrealistic
Priority	N/A
Stability	unstable
Source	Client
Verifiability	N/A

3.1.2 Map

Identifier	4-MAP_TOP
Requirement Description	The software simulates various things and does it fast. As such it should be possible to view these results easily and see the changes as they happen. The map should have the following features: 1. The map should have many views (as it has today) to filter and view different things happening in the world. 2. The map should be instantiable, i.e. multiple map-windows should be able to exist individually at the same time. 3. There should exist a view to show military units from multiple factions at the same time. 4. The map-window should be controllable in its own window as well. This includes mouse-wheel zoom and click-and-drag

	to adjust camera position and orientation. Also it should be easy to change the map-view.
Justification	The map is probably the most important feature in the client since it best “visualizes” the simulation. The client showed us a “command-central” room with many screens an emphasised the importance of being immersed in the information. More map-windows would more effectively use this environment and to ease to control of multiple windows each window should control it self (not entirely by a separate window, as it is now).
Need	required
Priority	high
Stability	stable
Source	Anton and Johannes
Verifiability	Verified when every feature marked with “required” is verified.

3.1.2.1 map views

Identifier	5-MAP_VIEWS
Requirement Description	The map should have map views (as it has today) to filter and view different things happening in the world.
Justification	This application is intended to be used in a environment with many screens to immerse the user in the information and maybe inspire.
Need	nice-to-have
Priority	medium
Stability	stable
Source	Customer

Verifiability	When different views can be chosen for a given map window.

3.1.2.2 Instantiable Map Views

Identifier	6-MAP_MOREMAPS
Requirement Description	The map should be instantiable, i.e. multiple map-windows should be able to exist individually at the same time.
Justification	There is a need to be able to simultaneously overview different areas and track different variables and units because of the size and scope of the simulation.
Need	nice-to-have
Priority	medium
Stability	stable
Source	Johannes
Verifiability	Should be able to create several map windows and configure them independently to show different aspects of the simulation.

3.1.2.3 view units from different factions

Identifier	7-MAP_UNITS
Requirement Description	There should exist a view to show military units from multiple factions at the same time.
Justification	When observing a battle or event it is useful to see all involved parties at the same time.
Need	nice-to-have
Priority	medium
Stability	stable
Source	Johannes

Verifiability	Can configure the map view to show units from multiple factions at the same time.

3.1.2.4 Control map view

Identifier	8-MAP_CONTROL
Requirement Description	The map-window should be controllable from its own window. This includes mouse-wheel zoom and click-and-drag to adjust camera zoom and position. Furthermore it should be easy to change the map-view.
Justification	Improved usability on machines with a single monitor.
Need	nice-to-have
Priority	medium
Stability	stable
Source	Johannes
Verifiability	When the mouse-wheel can be used to zoom and the map can be moved by clicking and dragging the mouse on it.

3.1.3 Improved scenario editor

Identifier	9-EDITOR_SCENARIO
Requirement Description	The existing scenario editor is unusable. It's intended purpose to edit the scenario xml-files used to describe what to simulate. It existed as such the scenario editor abstracts the user away from the actual files. The current editor is little more than an xml tree editor and has multiple bugs that hinder usage. Improve editor so that the map can be used for placement of objects.
Justification	XML is tedious to write, especially for big scenarios. XML is used (rather than any

	binary format) because it's extendable and so that "power users" should be easily able to do debugging and advanced things. But most users will not want to go into such depth and effort to understand and use the software. This is why the scenario editor is needed.
Need	nice-to-have
Priority	medium
Stability	unstable
Source	Aliquid
Verifiability	The graph editor should have the following features: 1. Create a graph. need: required priority: high 2. Edit a graph. Move nodes around, remove and add nodes and edges. need: required priority: high 3. Save graph to a scenario. need: required priority: high 4. Open graphs from scenarios. need: required priority: high 5. Edit arguments contained within nodes of the graph. need: required priority: high 6. Save presets of argument values of nodes.

3.1.4 Simulation Playback

Identifier	10-EDITOR_PLAYBACK
Requirement Description	It should be possible to save and playback simulated scenarios with the same result.
Justification	It would be useful to able to go back in time and view the intermediary states between past simulation events.

Need	nice-to-have
Priority	low
Stability	stable
Source	Aliquid
Verifiability	If a user can run, save and playback a scenario then this feature is complete.

3.2 Constraint requirements

This section contains non-functional requirements, i.e. what limits the final product must be kept in. Note that this may include both “musts” and “must-nots”.

3.2.1 Rebrand and rename

Identifier	11-REBRAND
Requirement Description	Change the name of the project to something else.
Justification	The name STRATMAS is stuck on a lot of different software that does different things. To imply what the software is and that it is different the customer wants to change it.
Need	required
Priority	high
Stability	stable
Source	Client
Verifiability	The product has a new name and a new logotype

3.2.2 Update dependencies

Identifier	12-DEPENDENCIES
Requirement Description	Newer versions of Boost, Java binding for OpenGL (JOGL) and Xerces. Will include some changes to some methods using deprecated api’s.

Justification	Stable libraries makes stable programs, less bugs and better performance.
Need	required
Priority	high
Stability	stable
Source	Aliquid
Verifiability	The server and client compiles and runs with the latest versions of of all dependencies.

3.2.3 More native look and feel

Identifier	13-NATIVE_GUI
Requirement Description	The client uses java ugly default interface widgets, but we can change that to a more native-ish look.
Justification	The default java widgets will throw users off when they see it. A more normal, and native look to the operating system will make the user feel at home.
Need	nice-to-have
Priority	high
Stability	stable
Source	Andreas
Verifiability	The clients GUI uses native ish (its java) buttons and other widgets. The buttons should compare and look almost the same as the ones the user are used to on their operating system.

3.2.4 Switch automatic build program for the server

Identifier	14-BUILD_SYSTEM

Requirement Description	Switch out the old build system for cmake.
Justification	The old build environment assumed old hardware was used and that everything needed to recompile after every change. CMake is platform agnostic and has a lot of automation features. Better build environment makes faster development and happy programmers.
Need	required
Priority	high
Stability	stable
Source	Anton
Verifiability	The server builds without errors

3.2.5 Performance

Identifier	16-SPEED
Requirement Description	The client UI should “feel” responsive when not doing actions that require IO or network operations.
Justification	The focus of the project is supposed to be on visualization, rather than exact correctness. As such the software should be easy to work in, and lagging and waiting would make the user frustrated.
Need	nice-to-have
Priority	high
Stability	stable
Source	Client
Verifiability	After a UI click a response should arrive within 1/2 second when run on a KTH ubuntu lab computer.