Final Report Assessment

for

HavNet

Group Name:	Lavin (No. 15)
Project name:	Physics network simulation tool
Course:	Software Engineering (DD1393)
Product/Process:	HavNet/Scrum
Client:	Avalanche Studios AB

Prepared by:

Niklas Kamateros	kamate@kth.se
Edvin Von Platen	edvinp@kth.se
Björn Lindqvist	dfsafd@kth.se
Ahmet Sezer	ahmets@kth.se
Edin Suta	esuta@kth.se
Mohammad Javad Ahmadi Amin	mjaa2@kth.se
Danilo Catalan Canales	daniloc@kth.se
Marko Kocic	kocic@kth.se
Nils Hammar	nilsham@kth.se
Joakim Ericsson	joakie@kth.se

Table of Contents

1. Scope

This document serves as a formal report of the project HavNet by Lavin, for Avalanche Studios, below called *the client*. The report will showcase achieved milestones, risk assessments and the overall result of the project with regards to project stakeholders wishes.

This section describes the scope of the project, and specifies all requirements and objectives in this project.

1.1 Summary

This project has produced a network bandwidth measuring and visualization tool, easy to implement into the physics engine Havok with a graphical interface for our client. Our client is an AAA game developer who use the Havok Physics engine in their development.

In this project we have developed HavNet, a software built to easily interact with Havok to bridge the gap between Havok and RakNet. HavNet measures the network bandwidth impact of different physics models and objects used in Havok Physics. As specified by the client, the project produced a product with a graphical interface to easily be used in conjunction with Havok, solely on the platform Windows 10. All the source code and tools needed to use our API is included in the final delivery to the client, including instructions and documentation on how to use the API. More details can be found in section 1.7 Solution Design and Modelling under the Development Methodology section of the document.

Initially the project had four milestones and one stretch goal, however there were changes during the project as one of them was deemed obsolete, which will be explain more in detail in part 1.4 Project Objectives.

The HavNet library is delivered together with a version of the Havok demo framework with HavNet integrated. The Havok demo framework is an application used for running and rendering Havok physics simulations. HavNet is integrated with the demo framework, which provides the client with a ready to use graphical application to measure network bandwidth impact of various physical objects.

1.2 Definitions and Acronyms

In the list below, terms used in this document are defined:

- *API (Application Programming Interface)*, a set of functions and procedures which forms the public interface between a computer library and its users.
- *clumsy*[1], a network tool that simulate different network conditions in a controlled manner.
- *Git*[2], a version control system for tracking changes in computer files.
- *Havok*[3], the physics engine used for the project.
- *Library*, a collection of resources used by computer programs.
- *Network protocol*, a set of rules for communication over networks.
- *Open source*, a category of software for which the original source code is made freely available, and for which the creation of derived works based upon its source under the same license as the original is allowed.
- *Physics engine*, a computer software used for simulating physics in video games, computer graphics and film.
- *RakNet*[4], a cross platform, open source, C++ networking engine for game programmers.
- *Scrum*, an agile method or framework for managing a software development project. Emphasis is put on people over processes and on fast, iterative work.
- *Snapshot interpolation*, an algorithm for synchronizing geometry data over a network.
- *UDP (User Datagram Protocol)*, a low-level internet protocol suitable for synchronization of world state.

1.3 Project Description

This section describes the clients need and motivates the project.

The Apex engine that the client use to produce games incorporates Havok for physics simulations. Several interesting challenges arises when running physics simulations over a limited network that warrant explanation in more detail. For example, how does one ensure that synchronization problems do not cause different actors' views of the world state to drift apart? How can one minimize the bandwidth usage when replicating world state over a network? To aid this important research, the client required a software product that facilitates their physics experiments.

The product was to be able to measure and visualize bandwidth usage of and potential lag of objects in Havok Physics, which would allow the client to find a good trade-off between quality and performance in networked open-world games. It should also allow the recording

[1] https://jagt.github.io/clumsy/ 2018-04-30
[2] https://git-scm.com/ 2018-04-30
[3] https://www.havok.com/physics/ 2018-04-30
[4] https://github.com/facebookarchive/RakNet 2018-04-30

and playback of physics simulations so that one synchronization algorithm can easily be compared against another.

1.3.1 Software Architectural Diagram

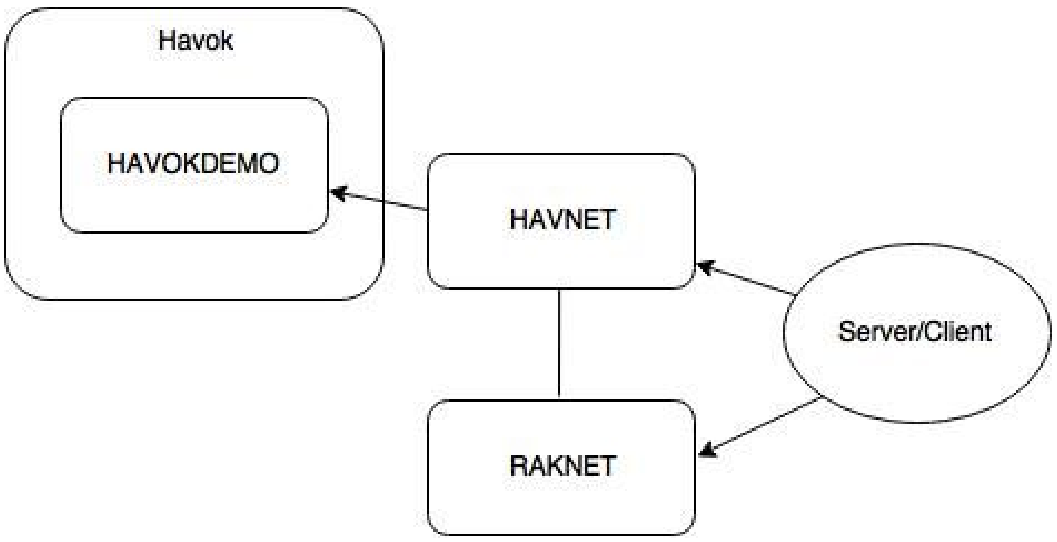


Figure 1: Software architecture *diagram.*

The four entities of the software architecture diagram are:

1. ***Havok*** is the physics simulation library used in the project. The use of Havok is required by our client. Havok product keys was supplied to the group from the client since Havok is a licensed software.
2. ***Havok Demo Framework*** is a graphical standalone application which renders Havok physics simulations and allows the user to interact with the simulations with mouse and keyboard. The application also provide options to see certain statistics of the physics demo, such as framerates and memory/CPU consumption. Examples of demos can be seen in *Figure 2-5* below. As a deliverable to Avalanche, HavNet was integrated with the Havok demo framework.
3. ***HavNet*** was the main deliverable for the project. HavNet was a code library for networking Havok simulated physics. With HavNet the user was able to send and receive information about the objects of the Havok simulated world. HavNet also allowed the user to alter network conditions of the simulation and view network statistics.
4. ***RakNet*** was the open source networking library used in the project. The use of RakNet was a requirement from our client. HavNet used RakNet for the network communication over the UDP network protocol.

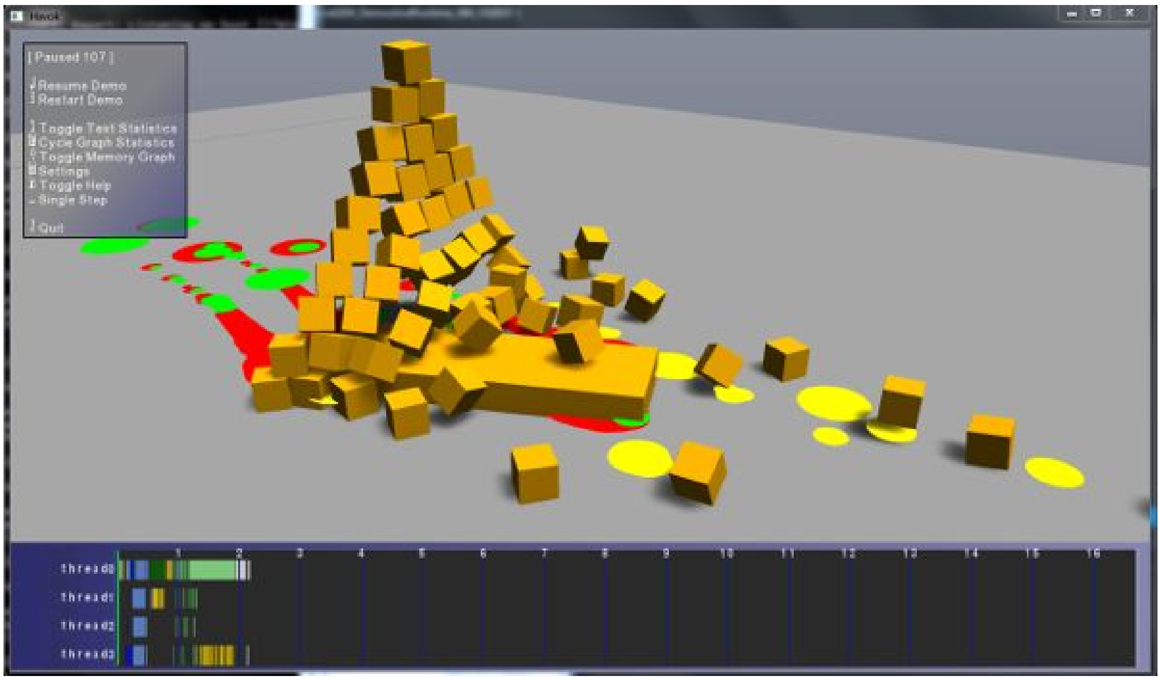


Figure 2. Havok physics demo simulating destruction, while showing CPU core load.

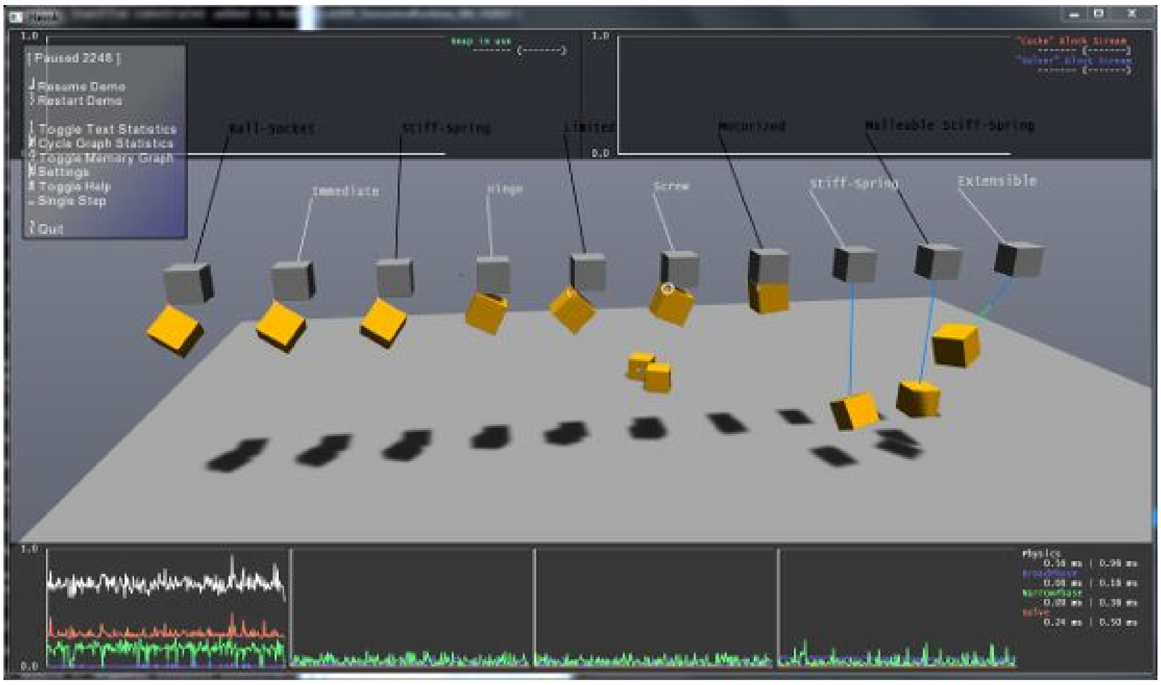


Figure 3. Havok physics demo demonstrating different suspensions, along with some real time statistics.

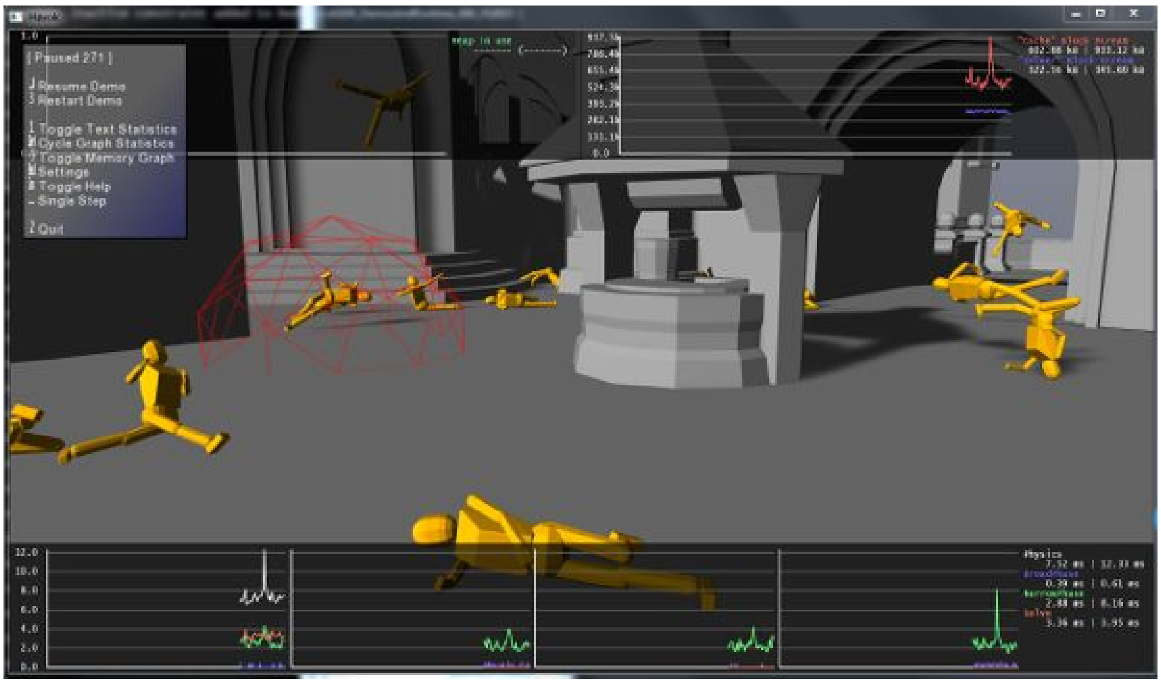


Figure 4. Havok physics demo showing a more game-like scenario, where the user can choose different weapons and see their effect on the rigid dolls, as they bounce around the scene.

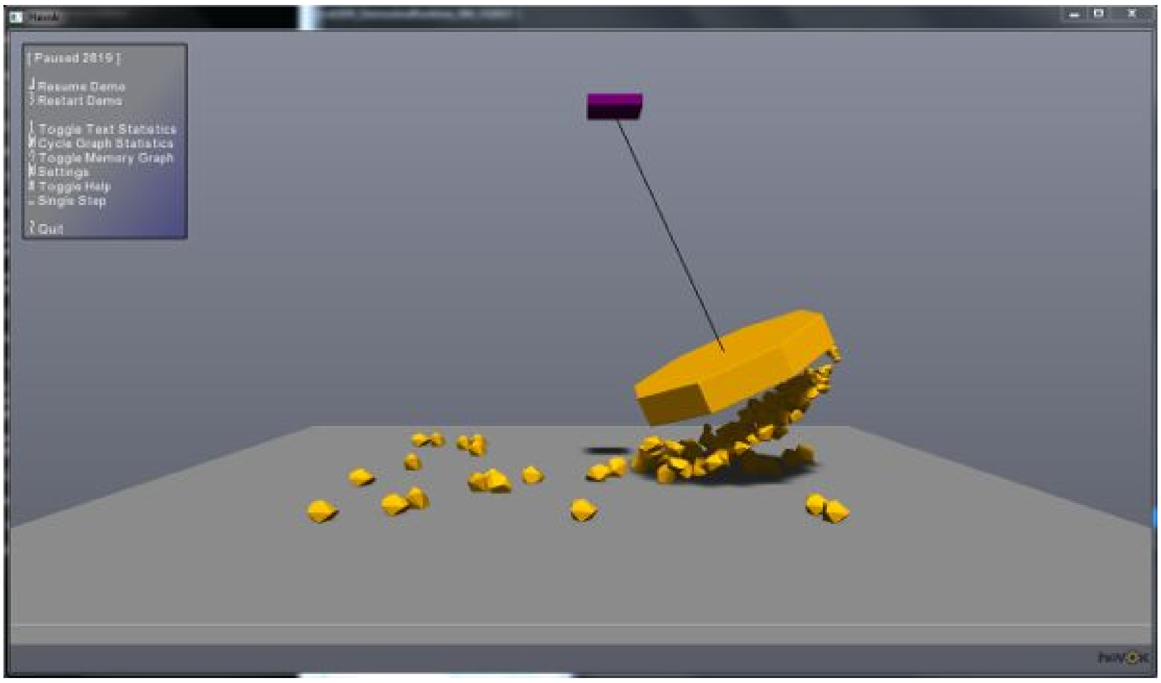


Figure 5. Havok physics demo simulating a magnet.

1.4 Project Objectives

At the start of the project our client provided us with a list of goals that they wanted our project to achieve. These goals were well specified, but did not require any specific method to achieve them, only that certain libraries were to be used. The focus was instead on the result to be delivered. Our client were open to discuss, and potentially change, some of the goals to better fit our groups competence level. There were several different hurdles which we as a group needed to overcome in order to complete our work, such as learning C++ programming and the open-source library RakNet, as well as Havok itself.

Initial Project Objectives

Objective Deliverables	**Due Date**
Milestone 1	
● Research Phase	18/01/15
Milestone 2	
● Replication of Physics	18/02/21
Milestone 3	
● Visualization of Bandwidth	18/03/21
Milestone 4	
● Recording and Playback	18/04/25

Table 1: Initial Project Milestones

1.4.1 Initial Major Objectives

Our client provided us with three primary objectives to fulfill and one extra objective that would be useful should we find ourselves with extra time and interest on our hands. These four objectives revolved around creating software for Havok using the RakNet library in order to monitor bandwidth usage of physics simulations. As such, the objectives were structured in a way to provide us with a clear route to follow throughout the project.

The first primary objective was the replication of physics simulation in the Havok demo framework. This goal centered around creating a modified version of the client's Havok demo framework where it would be possible to run two simultaneous instances, one acting as a server and one acting as a client. The server would be responsible for simulating the physics of the program and also replicating the simulation to the client. The client would be required to be able to receive the data which the server sends and be able to display the correct results based on the simulation run on the server. These two instances should be capable of running on two seperate machines, which in turn would enable simulation of a network connection using tools such as clumsy. This goal was the first and most important goal in our project and a prerequisite for the other major goals.

The second primary objective was visualization of bandwidth usage. This goal centered around making a graph or some other form of visual indicator showing the amount of bandwidth the simulation used. It was requested that the graph should be able to show bandwidth usage in real-time and it needed to accurately represent the highs and lows of the simulation's network load. The meaning of this goal was to be able to derive what simulations, or specific action within a simulation, that uses the most and least amount of bandwidth in order to optimize the implementation of the physics simulation.

The third and fourth objectives were recording and playback of server and recording and playback of client state respectively. The point of these two goals was to be able to compare the results of different implementations in order to evaluate the benefits of different means of physics replication. It's important to note that implementations of physics must consider both bandwidth usage as well as visual quality, as an excess or lack of either typically means either high bandwidth usage or poor rendering. Another point of these objectives were able to compare the expected outcome to the actual outcome, in other words making sure that the client results were the same, or sufficiently similar to the servers results.

1.4.2 Additional Objectives

During the initial meetings of the group it became clear that there would be a large need for us to study several different topics that would be required to complete our project. As such we decided to add one additional objective to our project plan, a research objective of sorts. This research objective would be completed individually by studying key topics during the first four weeks of the project in order for everyone to understand Havok's code base and to able to meaningfully contribute to the development.

The first goal of the research was to bring every group member's understanding and knowledge of C++ up to a serviceable standard. Several members of the group had no experience with C++ and it was deemed absolutely required that every member of the group both could read and write C++ code. Previous experiences in adapting to new programming languages, as well as varying degrees of experience working with C++, made this objective hard to achieve collectively and it was deemed optimal if everyone was allowed to learn at their own pace.

The second goal of the research was to familiarize ourselves with the libraries that our client had requested we should use in the development of the project, namely RakNet and Havok. Initially we thought it would be a good idea to split the team into two parts; one focusing on Havok and another focusing on RakNet. Then team members would be less strained by not having to need to learn all aspects of both libraries. Eventually however, we came to the conclusion that to understand the project a person would need to understand both libraries.

1.4.3 Final Milestones

As the project progressed, the structure of the objectives, and also some of the objectives themselves, changed. During one meeting with the client we discussed the issue and came to the conclusion that it would be best to scrap the third primary goal; recording and playback capabilities on the server. This decision was not made lightly, but due to several factors that include, but are not limited, to time constraints, usefulness, as well as the fact that a tool with the same functionality was already available and working.

Project Objectives over the course of the project.

Project Objectives	**Due Date**		
Milestone 1			
● Research Phase	18/01/15		
Milestone 2			
● Replication of Physics	18/02/21		
Milestone 3			
● Visualization of Bandwidth	18/03/21		

Table 2: Final Milestones

Moreover, we also decided to split up our main objectives into smaller parts in order to make them achievable for individuals during a seven-day sprint. Using Asana to document all of our objectives we managed to complete almost 50 tasks in total. While naming every task we have documented in Asana would be excessive, we would like to showcase a few examples of how we split up the larger objectives of the project in order to tackle them as individuals.

During the development of the bandwidth usage graph tool we decided to split up the development process into several smaller tasks. These tasks were then in most cases assigned a difficulty level as well as a priority. Listed below are the most important tasks from Asana which are directly related to the second primary objective:

- The three objects with the highest bandwidth effect should be marked with colors
- Don’t report network statistics before client is connected
- Make two graphs, one sent and one received
- Feed data to the new graph
- Change color on marked object and change back when a new one is selected. Save the id of the marked object
- Add a new menu exclusively for HavNet graphs and parameters

Several other smaller tasks were also created, typically for bugs and optimization. These tasks were all considered achievable within seven days and were assigned to individuals to work on during the course of our sprints. These tasks were intentionally kept rather small as unforeseen complications or unexpected difficulty was a recurring theme within the project.

1.5 Development Methodology

The project team consists of 10 members who all are students at either SU or KTH. The project leader has ensured that the project as a whole goes the way it’s planned and, together with the scrum master and lead programmer, ensured that the group stays focused on the right tasks and completed given tasks to reach our goals in time.

After assessing our risks and potential pitfalls, it was decided early on to use Scrum for software development (Scrum is further described in chapter 2.5).This ensured that focus was on the right tasks, to improve on what’s most important at a given time, and it enabled us to quickly adjust when needed.

1.5.1 Group Roles

Members	**Roles**	**Focus**
Niklas Kamateros	Project Leader/Product Owner	Management/Client relations
Edvin Von Platen	Lead Developer	Technical Lead
Björn Lindqvist	Scrum Master	Scrum management/development
Ahmet Sezer	Developer	Development
Edin Suta	Developer	Development
Mohammad Javad Ahmadi Amin	Developer	Development
Danilo Catalan Canales	Developer	Development
Marko Kocic	Developer	Testing/development
Nils Hammar	Developer	Development
Joakim Ericsson	Developer	Development

During the start of the project we decided to split the programming into two groups. One would primarily focus on RakNet, an open source networking library, whereas the other would primarily focus on the Havok Physics Engine. Both teams would however work together, but with more focus on each teams respective area. After a couple of weeks into the project, it was clear that the need for two separate team leaders was redundant, and we switched the team constellation to what it is above.

1.5.2 Communication

To ensure that the project moved forward, we have held weekly meetings where we discussed the status of the project and how the sprint had gone, as well as assign tasks for the next sprint. Initially meetings were frequent to become acquainted with each other and build good team-spirit, but later on moved to proper scrum-meetings every week. To mitigate communication issues, we used Slack as our only text-based communication, and Asana to assign tasks and set individual deadlines as well as specifying our backlog. Risks and issues in the project are discussed and explained in the section 3.2 Risk and Issues Management, where we further explain our methodology in more detail.

1.6 The Scrum Framework

In this section, an overview of Scrum is provided. Section 2.5 contains a more detailed analysis of the Scrum process along with descriptions of the adaptations to the methodology for our situation that the group made.

Scrum is a framework for managing work with an emphasis on software development. It is designed for teams of three to nine developers who break their work into actions that can be completed within timeboxed iterations, called sprints (typically two-weeks long) and track progress and re-plan in 15-minute stand-up meetings, called daily scrums. The framework is designed to make the development process resilient to unforeseen circumstances. Timeboxed iterations, in which project stakeholder feedback is incorporated, ensures that the work of the group never strays to far of from customer expectations.

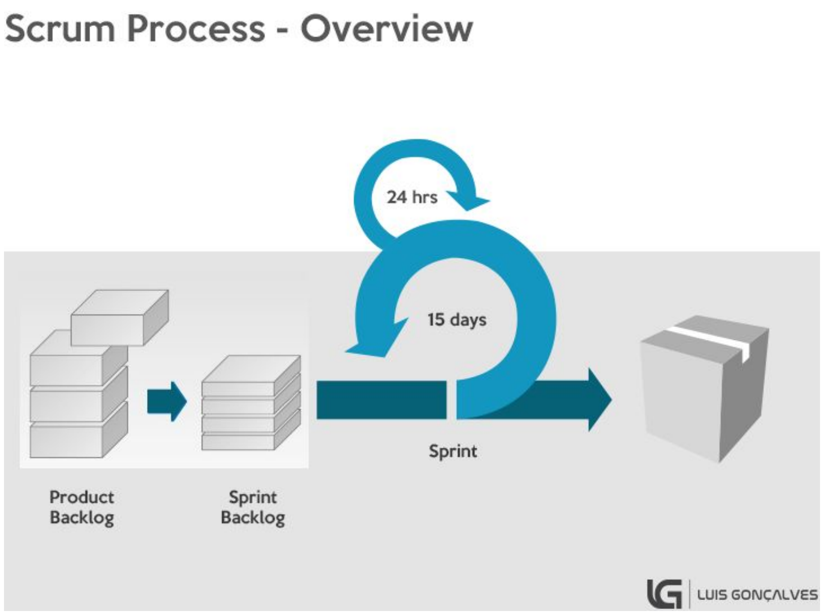


Figure 6: Schematic view of the Scrum process.

The three central artifacts of the Scrum process is the product backlog, the sprint backlog and the working product. Inputs to the team comes as items on the product backlog - a list of graded requirements negotiated between the customer and the product owner. During each sprint, the most important items are lifted from the product backlog and placed onto the sprint backlog. Responsibility for creating the sprint backlog falls upon the project leader and scrum master, but usually involves the team as a whole. The team then has the full sprint to work on items in the sprint backlog.

After the sprint is completed, the result of the sprint is summarized. A new sprint is organized and items from the previous sprint are carried over. Often, a retrospective is held in which the team can reflect on what went well and not so well during the sprint.

An important feature of the Scrum process is regularity. Sprints end at the given dates and there is no way to extend the duration of running sprints.

Unfortunately, scrum requires several key features that our development group simply did not possess. The group consisted of students from two separate institutions, with widely diverging and busy schedules, so implementing scrum to the letter was not possible. In the section 2.5 below, our adaptations to scrum is documented.

1.7 Solution Design and Modeling

In this section, a high-level overview of the design of the solution is given.

The solution was delivered (note: the alpha release was delivered to the client on April 26, 2018 but at the time of writing, the final release isn't complete) to the client as an easy-to-use C++ library called HavNet. The client was able to plug the library into their own applications and run experiments involving various synchronization scenarios. Emphasis was put on making this library as easy to use as possible.

HavNet implements both client and server functionality over the UDP protocol. The user is able to specify during startup if the process should assume the server or client role. If the server role is assumed, changes in Havok world state is propagated to the client process. Otherwise the client role is taken and changes are received from a server process.

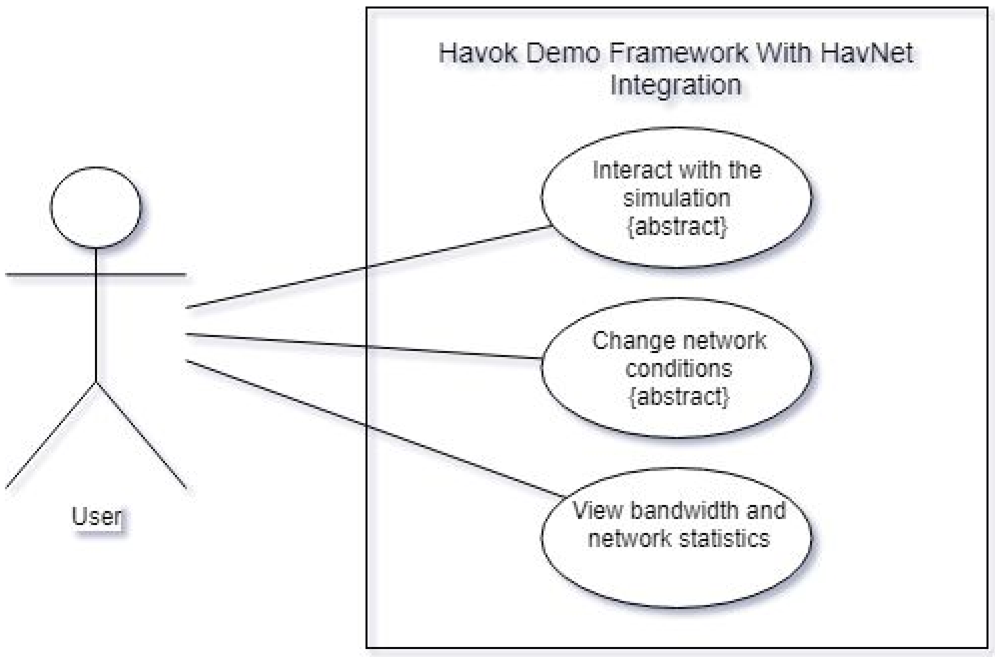
HavNet is also able to write metrics about the simulation to disk. A separate Python application has been developed for the customer to review and analyze this data. This application is described in detail in section 2.4.9.

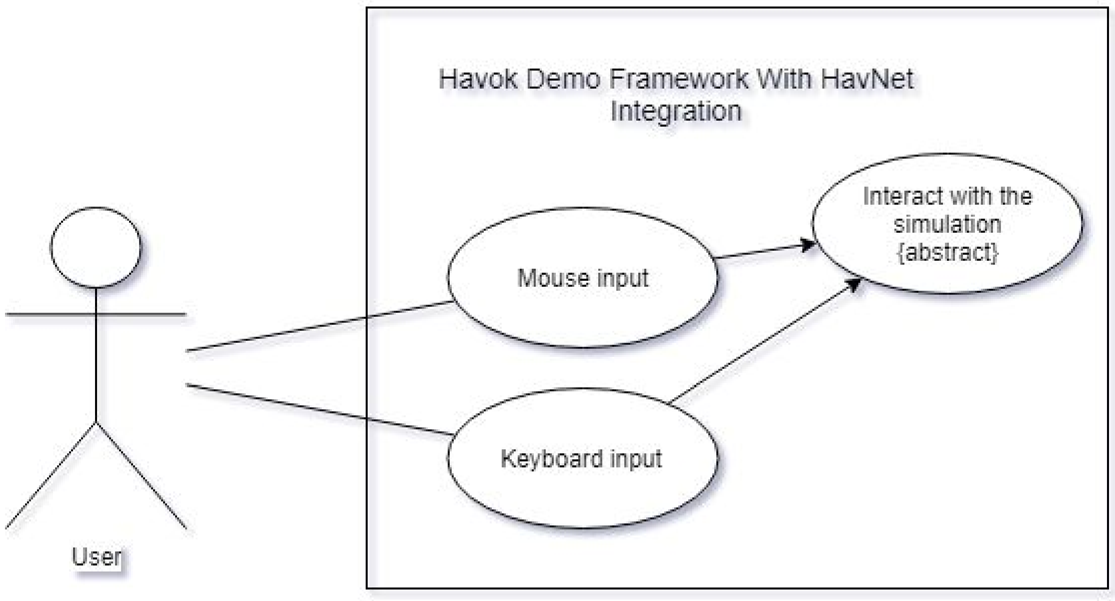
1.7.1 HavNet Use Case

The HavNet library is supposed to be integrated with an application running a Havok physics simulation. In this application, the user is able to interact with and change the state of the simulation. The integration of HavNet into the Havok Demo Framework serves as a testbed and demonstrates how it can be integrated in any Havok-based application.

It took several mail conversations and a meeting with the client to find the correct integration approach for HavNet into existing software. Because from the outset it wasn't clear that the approach we finally settled on was the right one.

Figure 7 depicts an UML-diagram of the use cases for HavNet integrated in the Havok demo framework.


Havok Demo Framework With HavNet Integration
Interact with the simulation {abstract}
Change network conditions {abstract}
View bandwidth and network statistics
User


Havok Demo Framework With HavNet Integration
Interact with the simulation {abstract}
Mouse input
Keyboard input
User

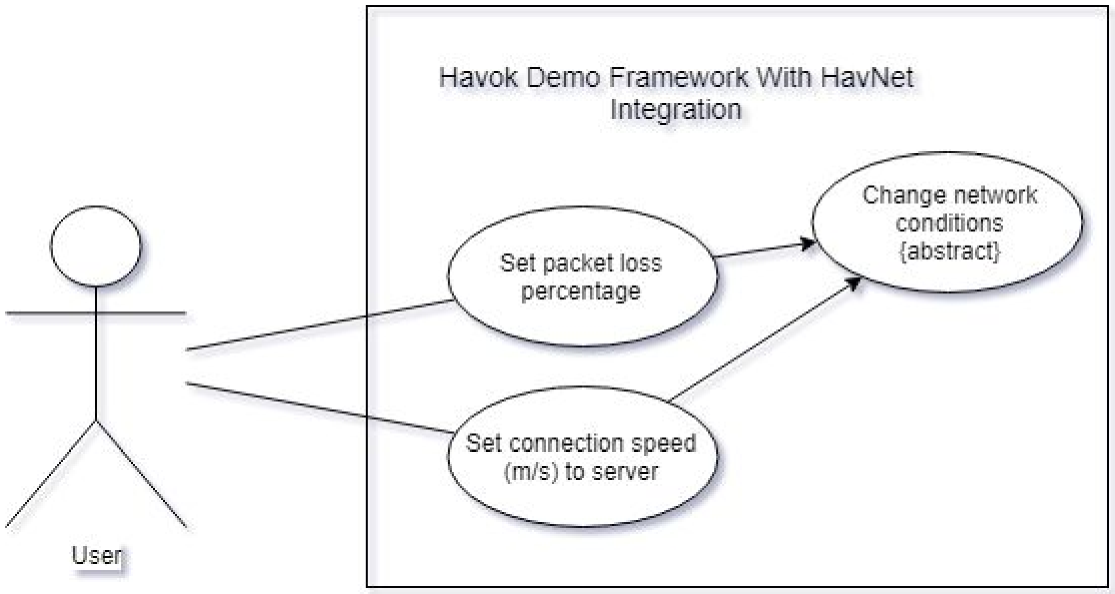

Havok Demo Framework With HavNet Integration
Change network conditions {abstract}
Set packet loss percentage
Set connection speed (m/s) to server
User

Figure 7: HavNet use case diagram

The three main use cases for the user as seen from the figures were:

1. Interacting with the simulation.
2. Changing network conditions.
3. Viewing bandwidth and network statistics.

User interaction was performed using the keyboard and the mouse. Network conditions were altered using the user interface. Viewing of bandwidth and network statistics visualization were done using functions found in the Havok Demo Framework. From these use cases, we elicited requirements such as **F#3 client user simulation input** and **F#5 network configuration.**

1.7.2. Use Scenarios

The main use scenario of HavNet for the client is using the HavNet library integrated in the Havok Demo Framework. As described by the client, the desired workflow for them is:

1. Create a physics demo in the Havok Demo Framework. Simulate a specific use case which needs research.
2. Setup the network conditions for the simulation.
3. View the simulation with live statistics and interact with the simulation.
4. Review the data gathered from the simulation.

1.7.3. HavNet Class Diagram

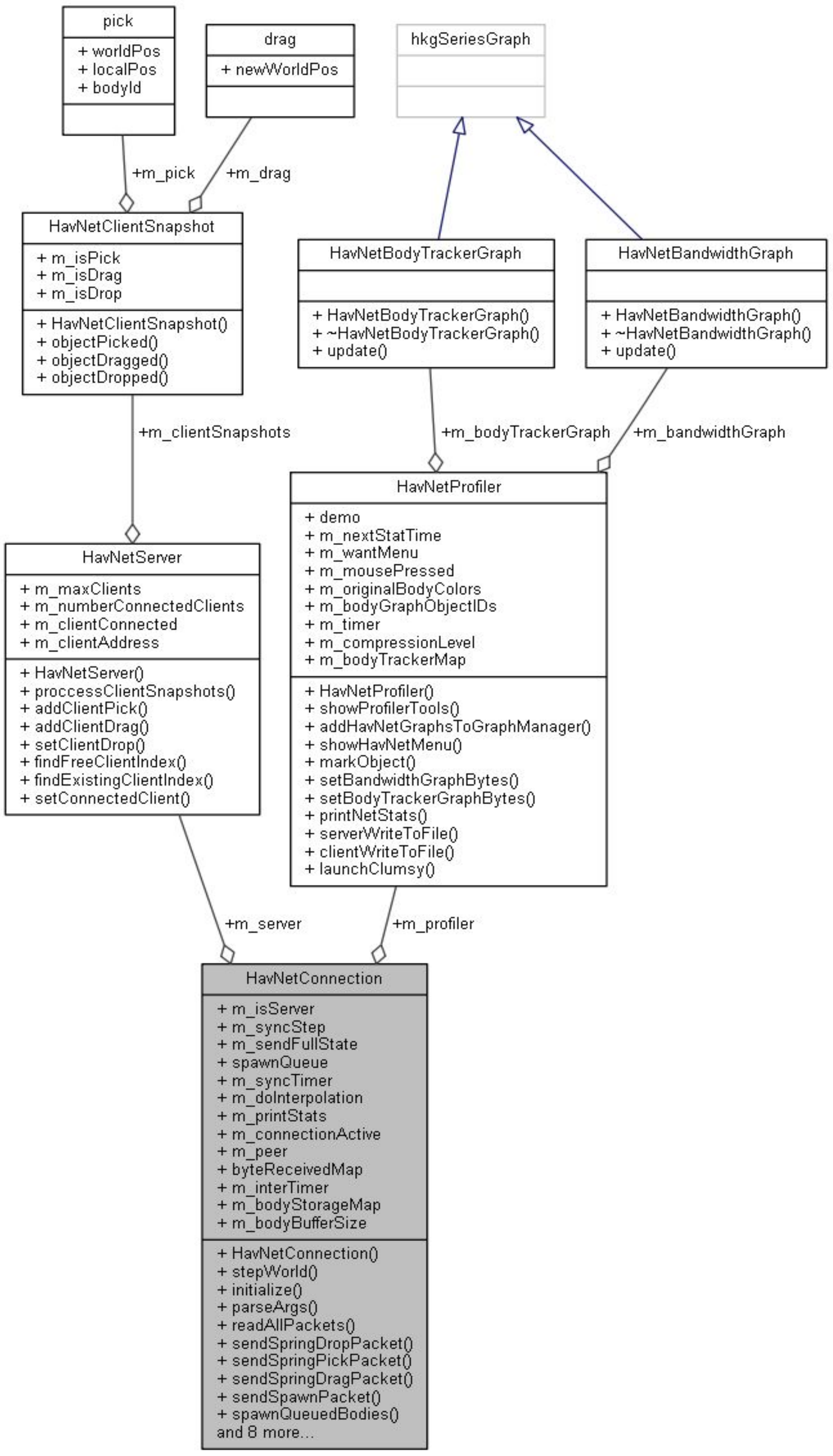


Figure 8: A UML-diagram of the HavNetConnection class in HavNet. It is the main class of HavNet.

This UML-diagram depicts how the classes in HavNet are connected to each other and to other parts of Havok and RakNet.

2. Project Analysis

This section provides a description of the project in detail, and analyzes methodology and requirements of the project as well as a detailed overview of the final product.

2.1 Background Study

Our client needed a standalone client-server application to perform isolated networked physics experiments. There were many challenges with networked physics that the client wanted to explore and find solutions for. These solutions would be implemented in future Avalanche games.

The Avalanche game engine Apex use the Havok physics engine for physics simulation. HavNet also use the Havok engine to ensure that simulations identical to the Apex engine can be performed.

Networking physics requires implementation of a client and server and establishing a connection between them. Communication between a client and server is done through either TCP or UDP sockets, for fast paced games the UDP type is preferred[5]. The Havok engine is written in C++, to allow simple integration with the Havok engine, the networking library should also be written in C++. A networking library matching these criterias is the open source Raknet library. Raknet was also recommended by our client and is used by them. There are however multiple alternatives such as ACE[6] and Boost.Asio[7]. The RakNet library was chosen due to the recommendation. An alternative to using a library would be programming our own, but due to a lack of experience in the area, and time constraints it was not deemed feasible.

The graphical application is required to be able to render physics simulations scenarios and process user input. At first there were two options, either build a new application with the QT library and integrate both Havok and Raknet with it or use and modify the Havok Demo Framework. The QT library would require writing a seperate renderer but would provide easier customization of the graphical interface. However the Demo Framework is capable of rendering physics demos and is already in use by the client when testing physics simulation. Together with the client the idea of using QT was quickly discarded and the solution of modifying the Demo Framework was chosen. The Demo Framework provides the required capabilities of different visualization techniques for the simulations.

[5] https://gafferongames.com/post/udp_vs_tcp/ 2018-04-30
[6] http://www.cs.wustl.edu/~schmidt/ACE.html 2018-04-30
[7] https://www.boost.org/doc/libs/1_67_0/doc/html/boost_asio.html 2018-04-30

2.2 Market Analysis

A bandwidth measuring tool is nothing new to the market. To improve and see the change of new networking methods would make a bandwidth measuring tool very essential to get actual data when change actually occur. PeakHour for Mac OS, NetWorkx Iftop for linux are some examples that measure bandwidth in a span of time. Our objective, however, is to integrate an open source networking library to the physics engine chosen by our client.

Networking tools have been seen in several physics engines, some more complex than others, but mostly support premade networking for multiplayer connectivity (Unity, cryEngine, Frostbite, etc.).There still are physics engines (Unreal Engine 4, cryEngine 3, cryEngine 5, source engine) that do provide an integrated bandwidth measurement tool. However, they create a tool which has features that is expected to be used by developers or developed to work for specific tasks. This has created a variation on features and differentiated strategies on displaying and handling the bandwidth data. Some even made standalone programs that would handle the bandwidth data in a different window that wouldn't be depending on the physics engine at all. So our design would lean more towards our clients already comfortable use of Havok Engine and make an integrated user interface with it.

2.3 Functional and Non-Functional Requirements

The functional requirements were elicited from meetings held with our client during the project timeline. Gathering the requirements was an iterative process where the client wanted features to expand on the product demo that was given at most meetings. Some features and requirements present in the final product were suggested by the HavNet group, such as **#F9** and **#F10**.

Below is the finalized requirements listing:

Label	**Functional Requirements**	**Priority 1-5**
F#1	Client user simulation input	4
F#2	Bandwidth profiling	4
F#3	Real Time demo framework graph visualization of bandwidth	4
F#4	Real time demo framework graph visualization of havok bodies bandwidth impact	3
F#5	Real time colorization of Havok bodies depending on bandwidth impact	3
F#6	Capability of changing bandwidth compression level	4
F#7	Real time configuration of bandwidth compression level	2

F#8	Graphical HavNet interface integrated into the Havok demo framework	5
F#9	Save simulation session bandwidth data on disk	2
F#10	Generate bandwidth usage graph for a simulation session	2
F#11	Specification of network conditions with the Clumsy tool	5
F#12	Snapshot interpolation method implementation	5
F#13	Display bandwidth statistics of selected body	3
	Non-Functional Requirements	
NF#1	The networked simulation should be able to handle a high number of simulated objects	3
NF#2	The program should be stable and not crash due to user input	2
NF#3	The HavNet library should be user-friendly	3
NF#4	The bandwidth visualisation should be done in compliance with researched visualisation techniques	2

Table 3: Functional and nonfunctional requirements.

2.3.1 Requirements Description

Below is a more detailed description of each requirement:

F#3, Client user simulation input: The client needs to be able to interact with the simulated world. This is done by sending user inputs from the client to the server. The inputs consists of dragging and dropping of bodies with the mouse.

F#4, Bandwidth profiling: The HavNet library needs to provide methods for profiling the bandwidth usage of the simulation.

F#5, Real Time demo framework graph visualization of bandwidth: To improve the visual user experience a real time graph of bandwidth usage was implemented in the Havok demo framework. The graph visualizes bandwidth in bytes per second received and sent for the active instance.

F#6, Real time demo framework graph visualization of havok bodies bandwidth impact: Specific bodies can have more or less impact on the bandwidth of the simulation. A graph displaying the bytes received for the three bodies receiving the biggest packets.

F#7, Real time colorization of Havok bodies depending on bandwidth impact: To enable easy profiling and visualization of a large amount of bodies simultaneously, bodies are colored according to bandwidth usage. Four bandwidth usage thresholds was implemented in the levels: 0-24%, 25-49%, 50-74%, 75-100%. Dynamically according to a modifiable user specified threshold.

F#8, Capability of changing bandwidth compression level: To visualize different bandwidth load during simulation. Implementation and configuration of different compression levels is required.

F#9, Real time configuration of bandwidth compression level: Implement keybindings for changing the compression level of packets during the simulation. Two modes will be supported either full or no compression.

F#10, Graphical HavNet interface integrated into the Havok demo framework: A menu with important profiling information and HavNet keybindings is to be added to the Havok demo framework.

F#11, Save simulation session bandwidth data on disk: Save bandwidth data for each simulation session to enable post session profiling and visualization.

F#12, Generate bandwidth usage graph for a simulation session: With saved bandwidth data from a simulation session construct bandwidth usage graphs for the whole session.

F#13, Specification of network conditions with the Clumsy tool: Enable launching of the Clumsy network tool with the Havok demo framework.

F#14, Snapshot interpolation method implementation: To enable profiling of simulation of physics over network the Snapshot Interpolation method was chosen for its easy of implementation compared to other methods.

F#15, Display bandwidth statistics of selected body: The user can select a body with the mouse and view incoming packets for the selected body.

NF#1, The networked simulation should be able to handle a high number of simulated objects: The delivered application should be able to handle a demanding simulation to ensure that the client will have a real use case for the application.

NF#2, The program should be stable and not crash due to user input: To provide a pleasant user experience the application should not crash due to normal usage.

NF#3, The HavNet library should be user friendly: The HavNet library should be easy to use and easy to modify. Focus on abstraction and documentation will enable users to have a pleasant experience working with the library.

NF#4, The bandwidth visualisation should be done in compliance with researched visualisation techniques: For the visualisation to be useful it needs to be clear and well presented. To enable this the HavNet library will implement researched visualisation techniques.

Initially the second milestone for the second period of the project was *Recording and Playback* as seen in Table 4 below.

Milestone Deliverables scheduled for completion over the second period

Milestone Deliverables	Due Date	% Completed	Deliverable Status
Milestone 1			
● Visualization of Bandwidth	18/03/21	100%	Completed
Milestone 2			
● Recording and Playback	18/04/25	0%	Removed

Table 4: Milestones deliverables.

This milestone was removed, together with the requirements associated with it, during the second period after discussions with the client that deemed the feature redundant. The reason is that a debug application is included with the Havok demo framework and it already supports recording of physics simulation.

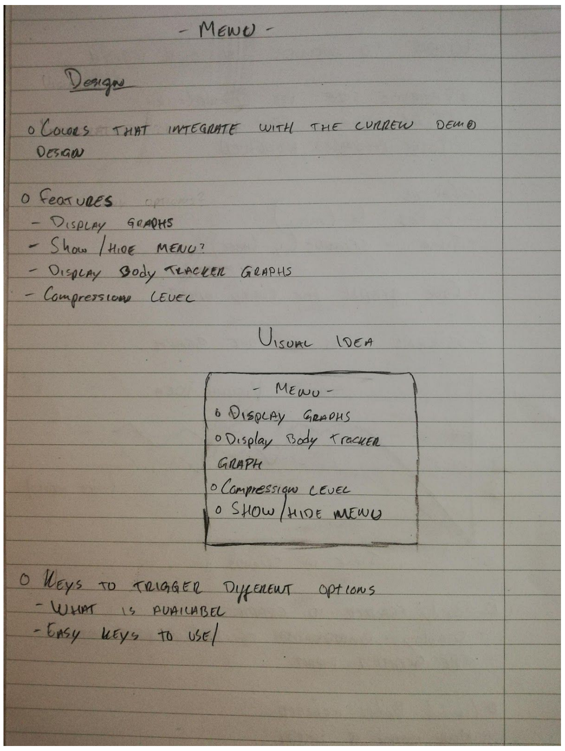
With the removal of the milestone, development was focused on expanding HavNets profiling capabilities and user experience.

2.4 Interface Description

This section describes and displays HavNet's user interface and covers all the aspects of the application. The images will show the different HavNet features.

2.4.1. Initial Prototype

Our design was modeled and then sketched to be as similar to the Havok design as possible. This would make it easier for our client to familiarize with it. We started off with small discussions and questioning what would be on the design. From there we drew out possible layouts and made it more realistic. Our sketches began with some drawings showing where the graphs would be shown on the screen while a demo would be running.



- MENU -

Design

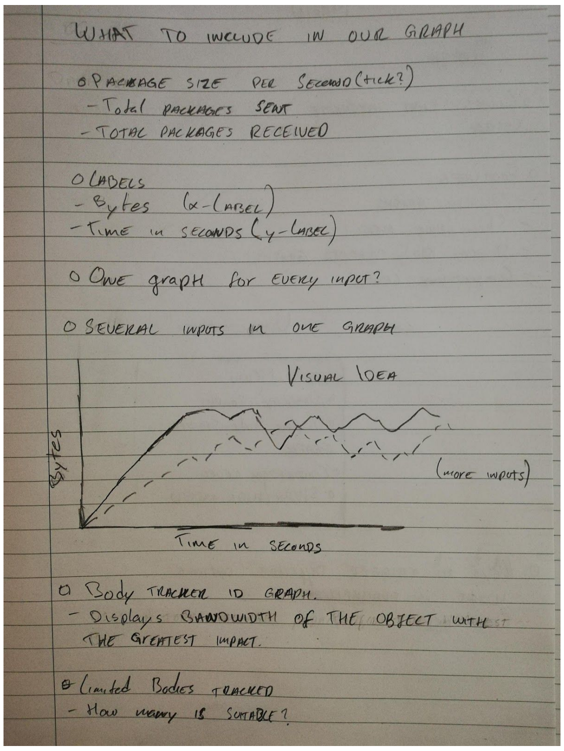
o Colors that integrate with the currew demo design

o Features
- Display graphs
- Show / Hide menu?
- Display Body tracker graphs
- Compression level

Visual idea

- Menu -
o Display graphs
o Display Body tracker graph
o Compression level
o Show / Hide menu

o Keys to trigger different options
- What is avaialbel
- Easy keys to use/



What to include in our graph

o Package size per second (tick?)
- Total packages sent
- Total packages received

o Labels
- Bytes (x-label)
- Time in seconds (y-label)

o One graph for every input?

o Several inputs in one graph

Visual idea

Bytes
Time in seconds
(more inputs)

o Body tracker ID graph.
- Displays bandwidth of the object with the greatest impact.

o Limited Bodies tracked
- How many is suitable?

Figure 9: Early prototype.

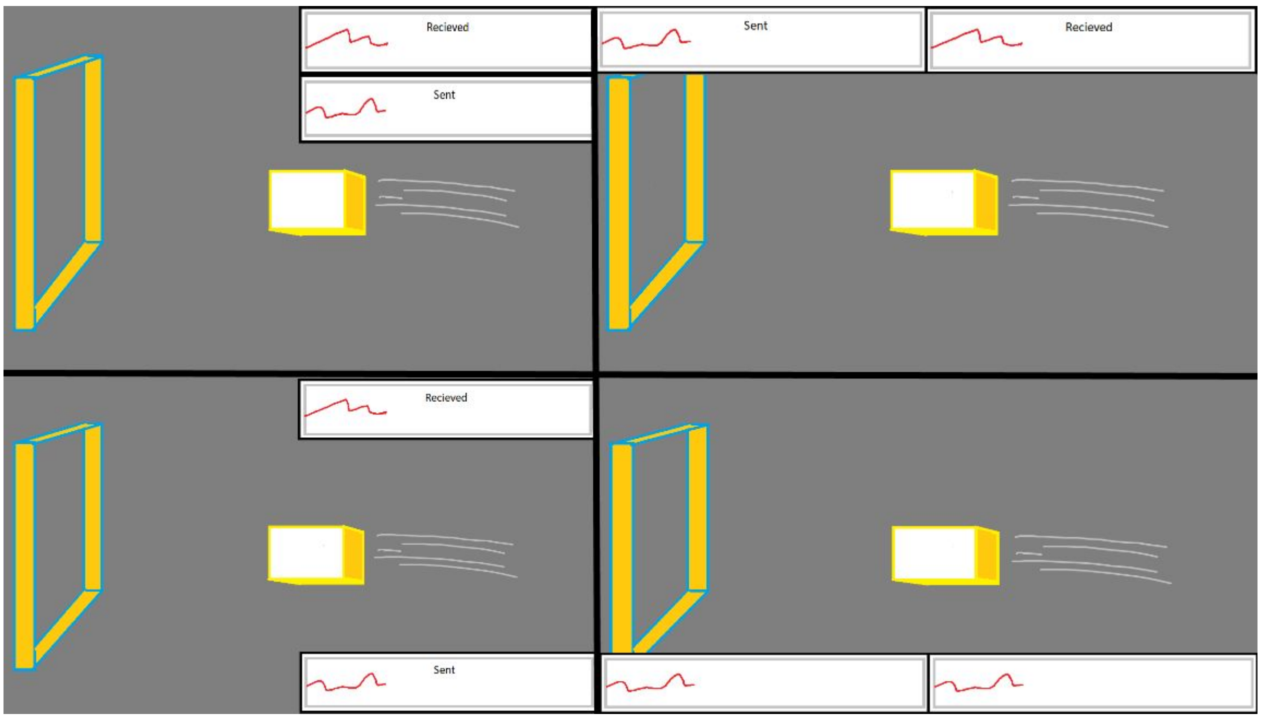


Figure 10: Sketches of bandwidth packages.

The graphs would show the bandwidth packages sent and received by the client and server. Further development would need more adding to the sketch, which is shown below.

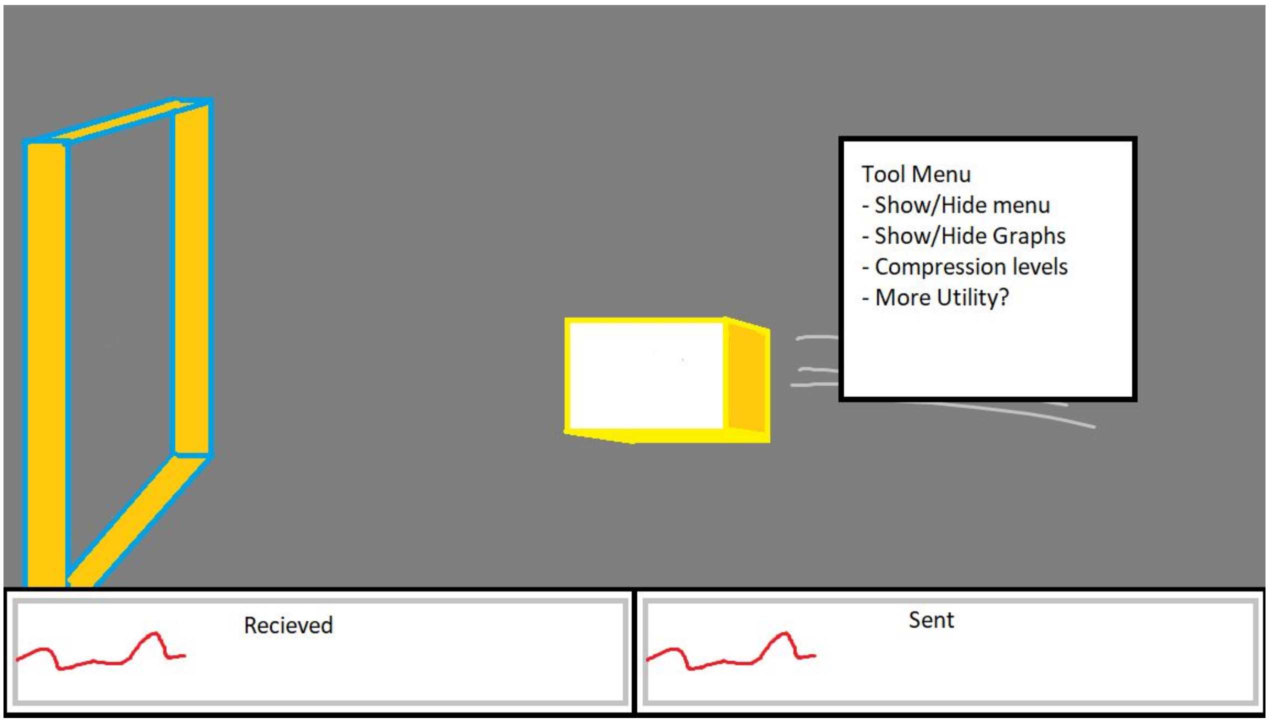
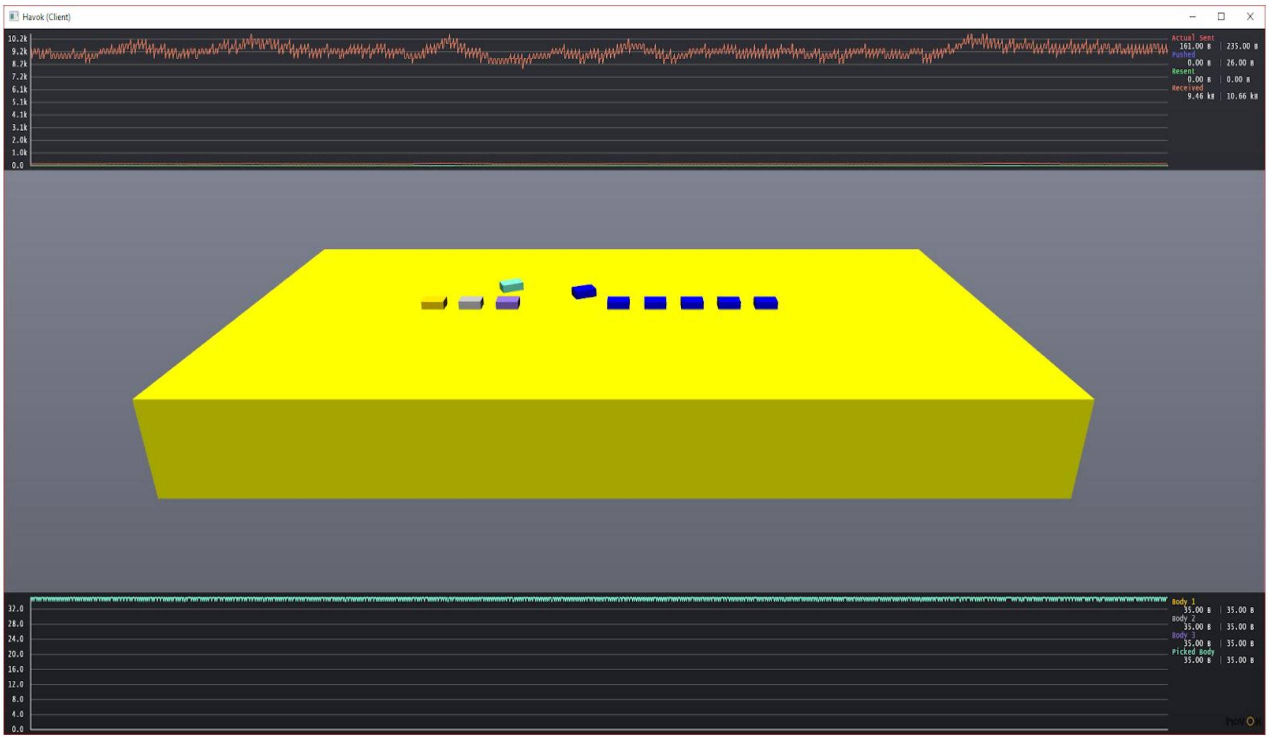


Figure 11: Showing an early prototype



```
Actual Sent
  161.00 B | 235.00 B
Pushed
    0.00 B | 26.00 B
Resent
    0.00 B | 0.00 B
Received
    9.46 kB | 10.66 kB
```

```
Body 1
   35.00 B | 35.00 B
Body 2
   35.00 B | 35.00 B
Body 3
   35.00 B | 35.00 B
Picked Body
   35.00 B | 35.00 B
```

Figure 12: Both HavNet graphs active along with body bandwidth usage based coloring.

2.4.2 Final Product Interface

Below is a detailed interface description of HavNet.

In the finished product, we have merged the received and sent graphs to save screen space. We have also arranged the graphs so that total bandwidth received and sent is on the top edge and bandwidth received and sent per object is on the bottom edge of the display. The HavNet Menu has seven options. Our product complies with the final specifications and can serve as a functional bandwidth measuring tool for Havok.

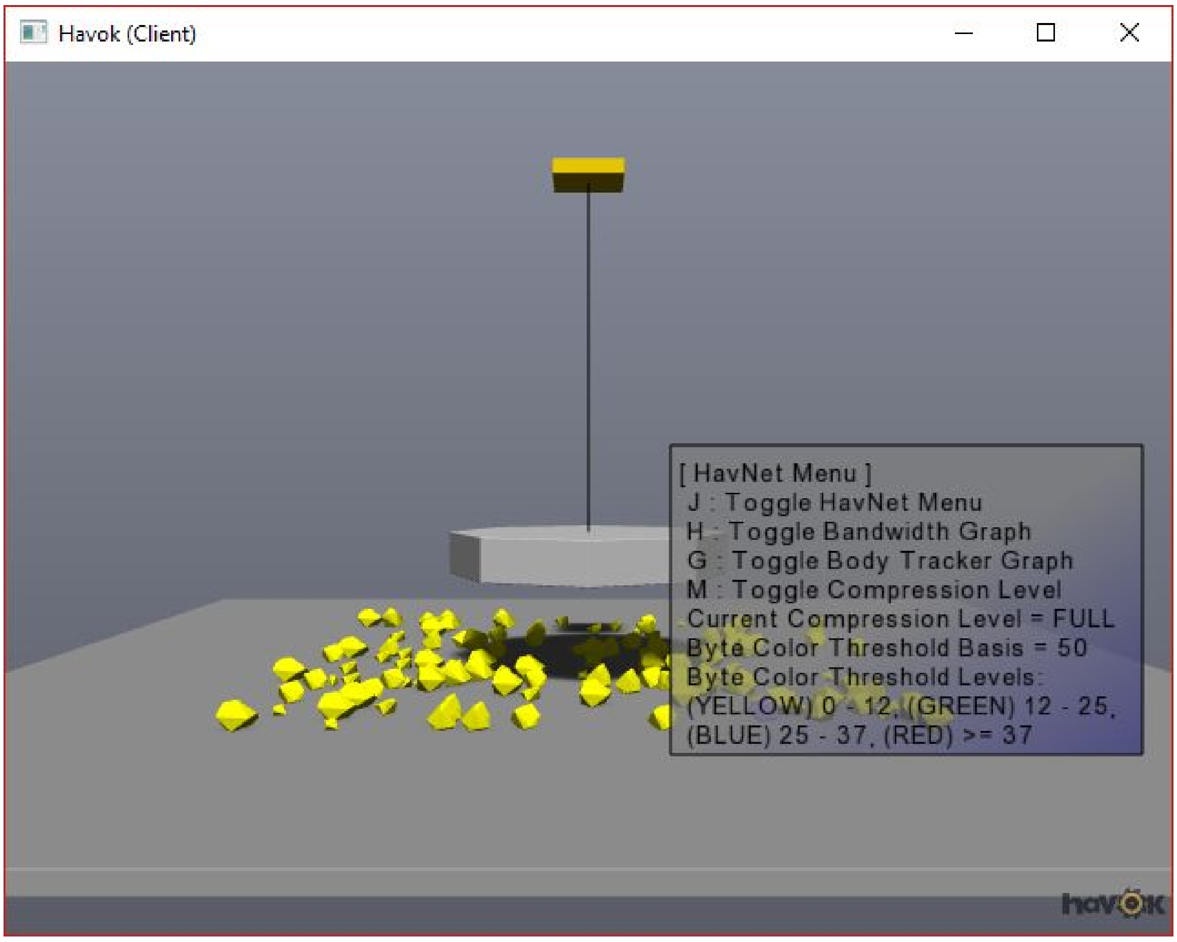
2.4.3 Starting a Server and Client

Running the executable Demos_x64-vs2015_debug.exe from the Demo\Demos directory would prompt a console message inviting the user to choose between a Server or Client.

Two instances of the mentioned executable, one for server and another for client, must be running in order for the networking connection to be established and it is required to first run the server and then the client. Otherwise there will be no connection between the two instances.

2.4.5. HavNet Demo Menu

When the user launches a Havok demo (client or server) they will be presented with a menu on the right side of the application window (in the form of a Havok demo menu). In the HavNet menu the user is able to toggle the menu (\`J\` key), the bandwidth graph (\`H\` key), the body tracker graph (\`G\` key) and change the level of compression (\`M\` key), where the level of compression also is shown (See Figure 13).



```
[ HavNet Menu ]
 J : Toggle HavNet Menu
 H : Toggle Bandwidth Graph
 G : Toggle Body Tracker Graph
 M : Toggle Compression Level
 Current Compression Level = FULL
 Byte Color Threshold Basis = 50
 Byte Color Threshold Levels:
 (YELLOW) 0 - 12, (GREEN) 12 - 25,
 (BLUE) 25 - 37, (RED) >= 37
```

Figure 13: HavNet Menu.

2.4.6 Bandwidth Demo

For network testing purposes a demo was created. In this demo the user is able to spawn a number of dynamic bodies of their choosing. This is to observe the effect of having many active bodies and how they impact the bandwidth usage, how much package loss there is, etc. This demo is located in `Bandwidth\Spawn Objects` among the different Havok Demo alternatives (See Figure 14).

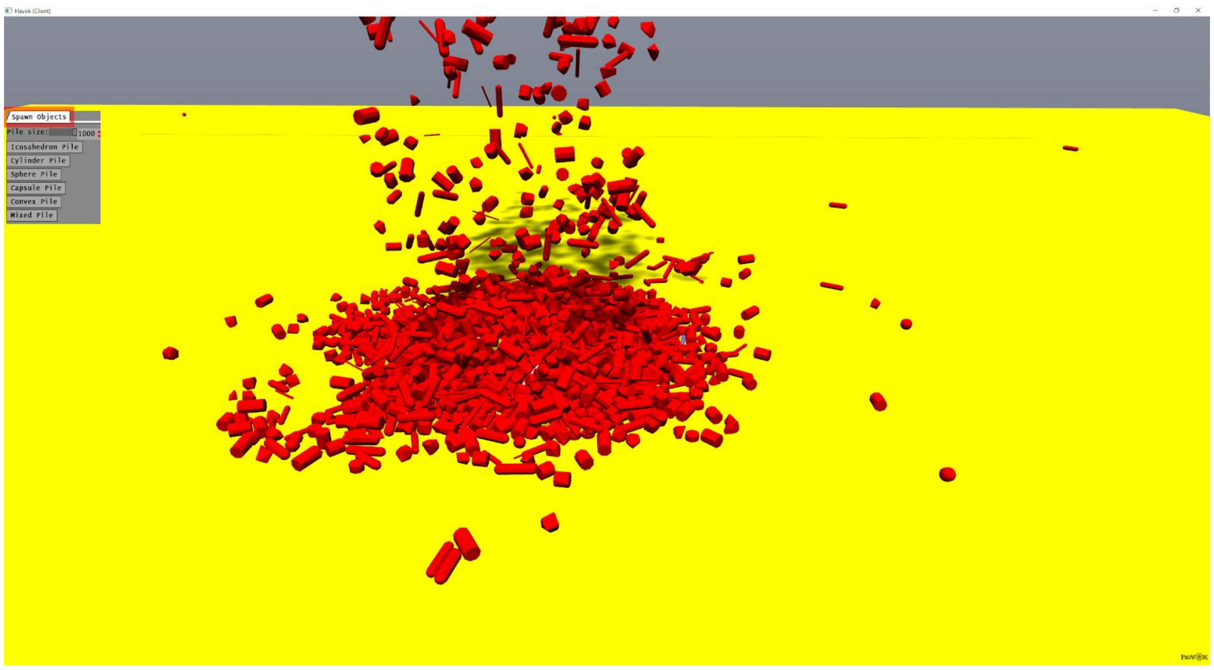


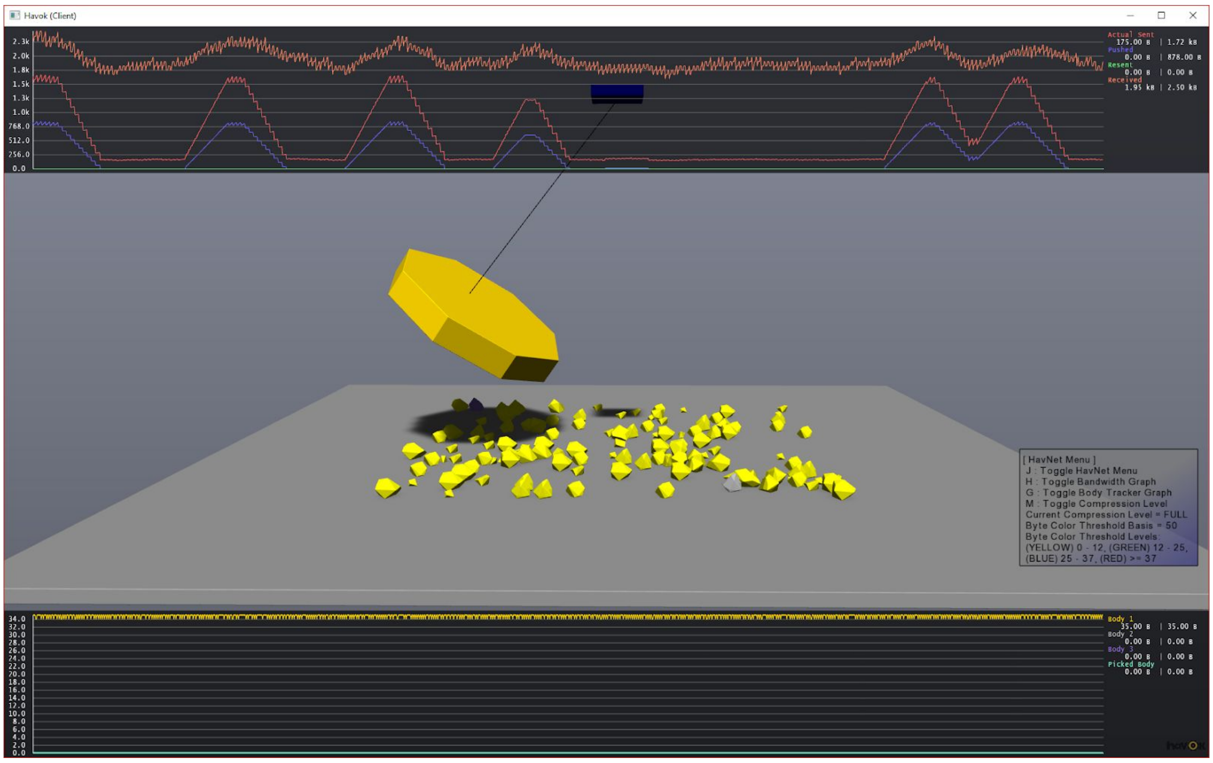
Figure 14: HavNet bandwidth demo.

2.4.7 Bandwidth Graph

Real-time network statistics are displayed using integrated Havok graphs. If the server and client are connected properly the bandwidth graph would present the data size sent, pushed, resent and received by the server and client on their respective windows (See Figure 15 upper graph).

2.4.8 Body Tracker Graph

The body tracker graph provides bandwidth statistics for the body which is selected by the user. Along with tracking the picked body it also tracks the top 3 bodies that have the most bandwidth impact in the demo, where body No. 1 is the body with the highest impact out of the three. All of the tracked bodies are colored differently from the rest of the bodies for easy identification. This graph tracks the data received for each of the mentioned bodies (See Figure 15 lower graph).



```
Actual Sent
  175.00 B  | 1.72 kB
Pushed
    0.00 B  | 878.00 B
Resent
    0.00 B  | 0.00 B
Received
    1.95 kB | 2.50 kB
```

```
[ HavNet Menu ]
 J : Toggle HavNet Menu
 H : Toggle Bandwidth Graph
 G : Toggle Body Tracker Graph
 M : Toggle Compression Level
 Current Compression Level = FULL
 Byte Color Threshold Basis = 50
 Byte Color Threshold Levels:
 (YELLOW) 0 - 12, (GREEN) 12 - 25,
 (BLUE) 25 - 37, (RED) >= 37
```

```
Body 1
   35.00 B  | 35.00 B
Body 2
    0.00 B  | 0.00 B
Body 3
    0.00 B  | 0.00 B
Picked Body
    0.00 B  | 0.00 B
```

Figure 15: HavNet graphs

2.4.9 Recorded Bandwidth Usage on Disk and Matplotlib

While running a demo, HavNet records the bandwidth statistics and log them in a file that is saved as a `.csv` file under the directory `HavNetGraphs`. These files are named by the date, time, the demo name, server/client status and contains information of bytes received and sent per second (in that order).

The logs saved in the mentioned directory is used afterwards in order to show and compare data in graphs created by Matplotlib in python. `HavNet\HavNetBandwidthPlotter.py` is the file which takes care of plotting the bandwidth statistic graphs. In order for this to work Matplotlib must be downloaded within the python IDE. Running the file would prompt a window which asks the user to either plot graphs from network statistics files or plot graphs with the difference between two network statistics files. The difference between the corresponding data would be calculated and displayed for both received and sent bytes during a time span, which is shown in Figure 16 in the lower part. The graphs in the upper part of the figure show the data from the files chosen in separate colors.

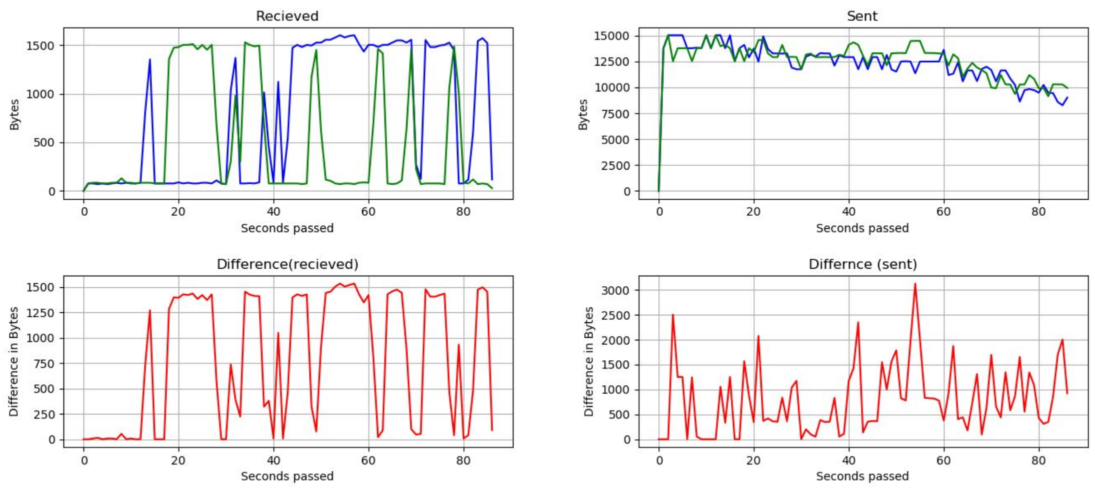


Figure 16: MatPlotlib

2.5 Analysis of the Scrum Implementation

This section describes how scrum was implemented in our project and analyzes the outcome of the process in detail.

2.5.1 Omitted Workflow

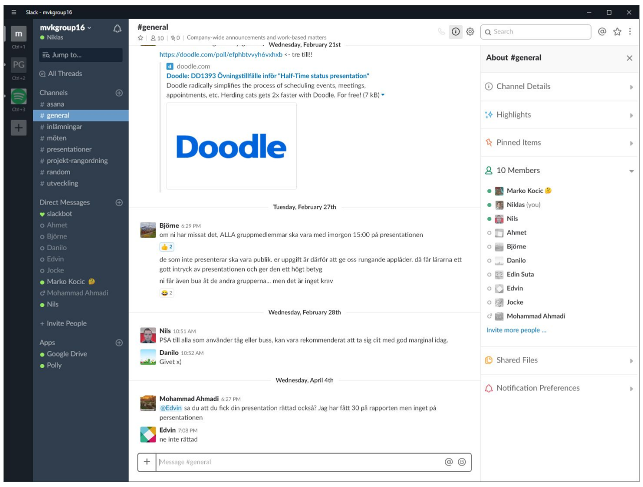
Because of scheduling issues, daily meetings were not possible. For the same reason, we did not involve all members in constructing the product backlog, nor did we hold proper retrospectives. However, we held weekly face-to-face meetings in which all members of the group were obliged to attend. In those meetings, the week's sprint backlog was decided upon and each member signed up for one or more tasks to complete during the coming week.

2.5.2 Definition of Done

A conservative Definition of Done (DoD) was used for the project which covered both the customer and the developer perspective. In particular, our DoD implied that a task did not cause any regressions.

2.5.3 Planning Tools

As the team was geographically distributed, most of the communication and management were done using various web tools to mitigate fallout from the low number of face-to-face meetings. Slack was our main communication channel, which was divided into several different channels as seen in the picture below. This made it easy to focus on the correct task and easy to find the right people to discuss with. We also used Asana to plan and assign our sprint plans, which is a web-based project management tool. With the help of Asana it was always easy to know what needed to be done, and it helped manage the project in an efficient way.



#general
10 | 0 | Company-wide announcements and work-based matters

Wednesday, February 21st

https://doodle.com/poll/efphbtvvyh6vxhxb <- tre till!!

doodle.com
Doodle: DD1393 Övningstillfälle inför "Half-Time status presentation"
Doodle radically simplifies the process of scheduling events, meetings, appointments, etc. Herding cats gets 2x faster with Doodle. For free! (7 kB)

Tuesday, February 27th

Björne 6:29 PM
om ni har missat det, ALLA gruppmedlemmar ska vara med imorgon 15:00 på presentationen
👍 2
de som inte presenterar ska vara publik. er uppgift är därför att ge oss rungande appläder. då får lärarna ett gott intryck av presentationen och ger den ett högt betyg
ni får även bua åt de andra grupperna... men det är inget krav
😂 2

Wednesday, February 28th

Nils 10:51 AM
PSA till alla som använder tåg eller buss, kan vara rekommenderat att ta sig dit med god marginal idag.

Danilo 10:52 AM
Givet x)

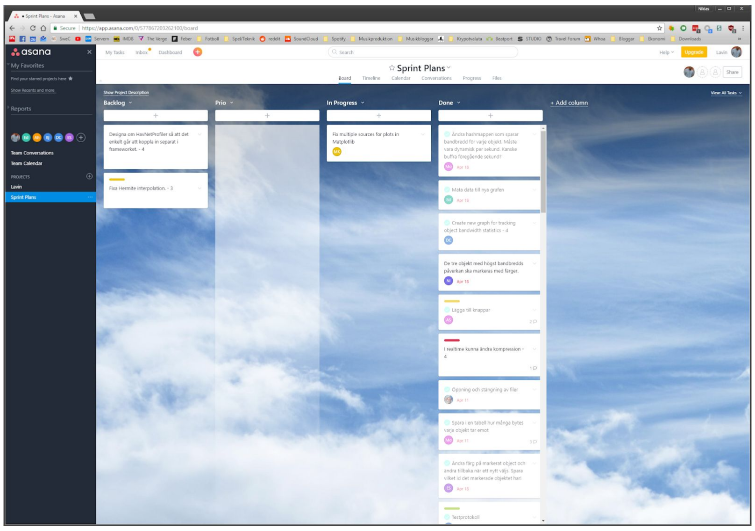
Wednesday, April 4th

Mohammad Ahmadi 6:27 PM
@Edvin sa du att du fick din presentation rättad också? Jag har fått 30 på rapporten men inget på persentationen

Edvin 7:08 PM
ne inte rättad

Message #general

Figure 17. A Slack window showing an ongoing discussion between several developers.



Backlog
- Designa om HavNetProfiler så att det enkelt går att koppla in separat i frameworket. - 4
- Fixa Hermite interpolation. - 3

Prio

In Progress
- Fix multiple sources for plots in Matplotlib

Done
- Ändra hashmappen som sparar bandbredd för varje objekt. Måste vara dynamisk per sekund. Kanske buffra föregående sekund? Apr 18
- Mata data till nya grafen Apr 18
- Create new graph for tracking object bandwidth statistics - 4
- De tre objekt med högst bandbredds påverkan ska markeras med färger. Apr 18
- Lägga till knappar
- I realtime kunna ändra kompression - 4
- Öppning och stängning av filer Apr 11
- Spara i en tabell hur många bytes varje objekt tar emot Apr 11
- Ändra färg på markerat object och ändra tillbaka när ett nytt väljs. Spara vilket id det markerade objektet har! Apr 18
- Testprotokoll

Figure 18. Sprint plans in Asana.

2.5.4 Sprint Meetings

Although we began working with the project in December 2017, we held our first formal sprint meeting on March 7, 2018 and then on every wednesday since. These meetings became an instrumental part of the process from that date and onwards.

The meetings were led by the groups Scrum Master and the project leader handled the notes of the meetings. The notes turned out to be an incredibly useful resource which we often had reason to refer back to. Most of the analysis in this section is based on information extracted from those notes.

2.5.4.1 Meeting Format

The same agenda, reproduced below, was used for all sprint meetings.

1. Secretary appointment
2. Retrospective
3. Reporting of results

4. Technical discussions
5. Assignment of new tasks
6. Miscellaneous

The meetings began with one person volunteering to take notes. Then a quick retrospective was held in which each group member reflected on the groups work during the previous sprint. The intention of this exercise was to find ways in which the workflow of the group could be improved. Perhaps the step wasn't as fruitful as it could have been because most members of the group were unfamiliar with Scrum.

The next point on the agenda was time and result reporting. This entailed each member stating the amount of hours spent on the project and the tasks assigned to them were finished or not. Time and result reporting is analyzed in more detail in section 2.5.7.2.

The fourth point was designated to open technical discussion. For example, if members had ran into technical difficulties or had discovered bugs that they needed to talk over they could do it at this phase.

The fifth point was for assigning new tasks. Members picked tasks from the backlog to complete during the upcoming week and also estimated how many hours would be required to finish the tasks. Section 2.5.7.1 analyzes the groups time estimation in more detail.

Thanks to the formal meeting structure, very few meetings dragged on. We didn't keep exact track of time, but almost all meetings were finished in about two hours.

2.5.6.2 Attendance and Absence

As previously mentioned, attendance was mandatory but unforeseen circumstances such as sickness and vacations made it impossible for everyone to be present at every meeting. The average number of meeting attendances per person was 6.8 or 85 %. Only three members were able to attend to every meeting held. Figure 17 shows the number of sprint meetings per member.

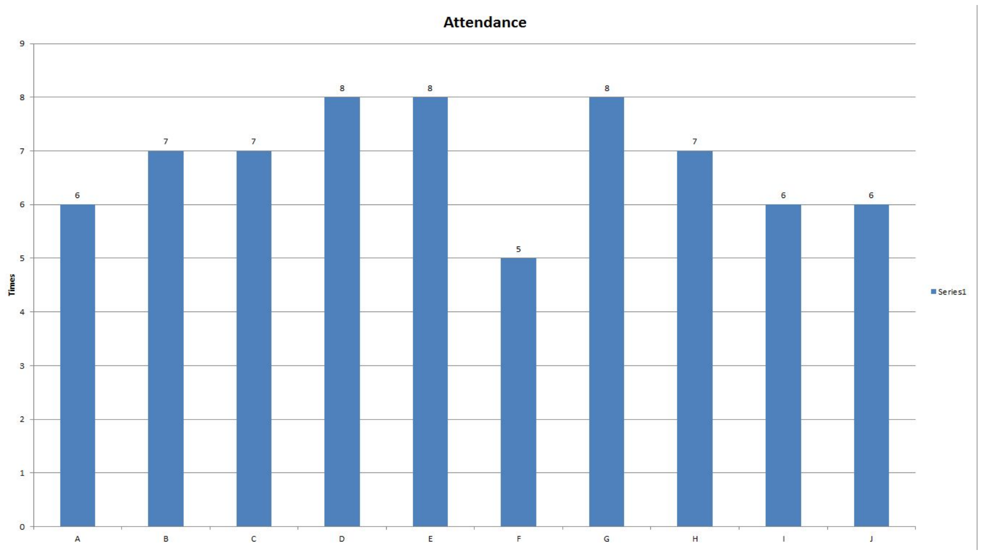


Figure 17. Meetings attended per group member.

The amount of absence is likely fairly typical for the Swedish spring season which is usually plagued by colds.

2.5.6 Task Assignment

Tasks were primarily assigned by group members own volition. Many times tasks that weren't finished were "carried over" and onto the same group member for the next sprint. We felt it was reasonable to allow group members who were almost done to finish what they were doing. Especially as they were likely to be extra knowledgeable of the area that they were working on. But in some cases, members felt that they were stuck on their tasks and they were then reassigned to other group members.

There was also an apparent skill differential between those group members who had prior experience with C++ and Git and those who hadn't. That made it so that more complicated tasks naturally gravitated towards those. Here we could perhaps have employed pair-programming to try and spread the knowledge. But, as mentioned, the distributed nature of the group made pair programming sessions hard to schedule.

A problem we hadn't expected was that we ran out of tasks. For some sprints, team members had more free time than we had tasks to assign. That can partially be explained by us not managing the backlog properly and partially by the nature of the project; some tasks involved 90 % research and 10 % implementation and those were impossible to split up.

2.5.7 Time Management

This section describes issues related to time reporting and estimation in detail. Early on we decided that it would be most fair if everyone in the project spent an equal amount of time on it. For obvious reasons, it wouldn't be fair if one or two persons did most of the work while the other team members did very little. Therefore, and to make planning easier, we used a simple time management system consisting of two steps; time estimation and follow-up reporting.

2.5.7.1 Time Estimation

In step five of the sprint meetings, members were assigned tasks and also estimated how many hours those would take to complete. It is well known that it is incredibly difficult for developers to produce accurate time estimates, and our group was no exception. Many times, the estimates completely missed the mark. Underestimates were much more common than overestimates. In meeting situations, peer pressure has a tendency to decrease time estimates.

However, coming up with time estimates is something virtually all developers have to do, so the experience was probably beneficial.

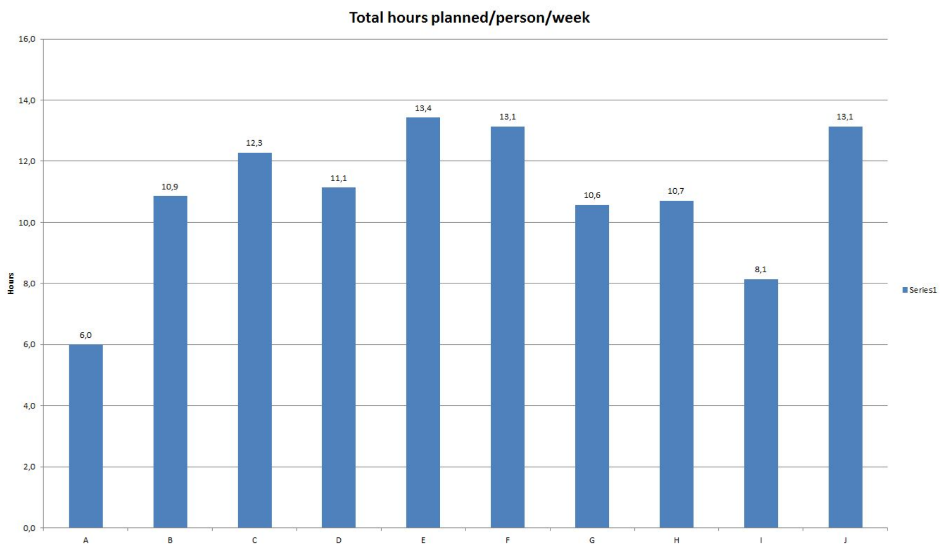


Figure 19: Average number of hours estimated per team member (A-J).

Figure 19 shows the average amount of hours estimated per person per week (sprint). The group average is 10.93 and the median 11.0. The figure shows that, while there were differences, by and large the number of hours allocated to each person were equal.

2.5.7.2 Time Reporting

Time reporting was done in step three of the sprint meetings. The members reported how many hours they had spent on the project and how "finished" they were with their tasks on a scale from 0 to 100 %. This allowed us to calibrate time estimates based on how much time tasks actually took. Unsurprisingly, most tasks took more time than expected.

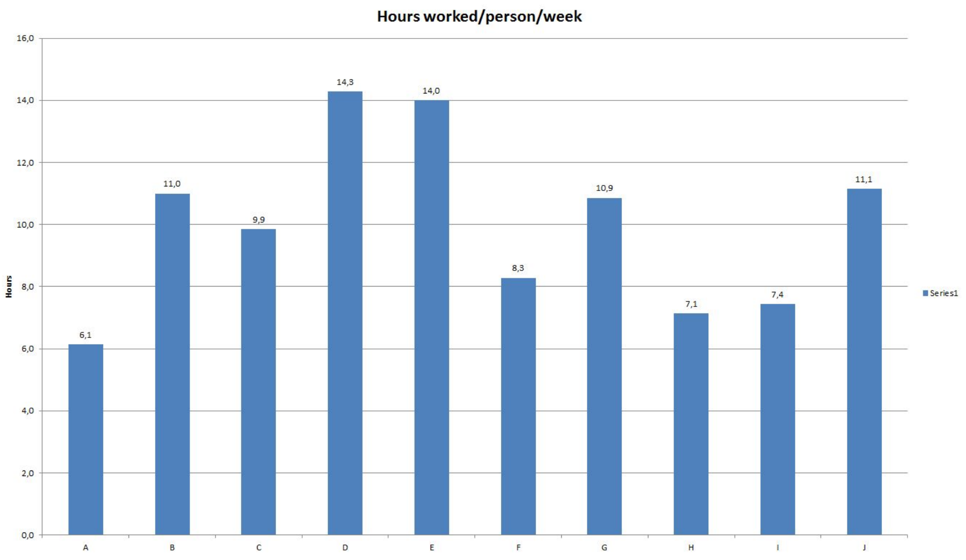


Figure 20: Average number of hours reported per team member per week (A-J).

Figure 20 shows the average number of hours reported per person per sprint. In all likelihood the numbers are severe underestimates of actual time spent working on the project. Members often forgot to not only include time spent coding, but also time spent in meetings, researching and helping other group members in their reports.

Figure 21 shows the total number of hours reported for the team per week. The falling trend during the meetings in April is likely not caused by decreasing activity, but by a decreasing number of sprint backlog items available.

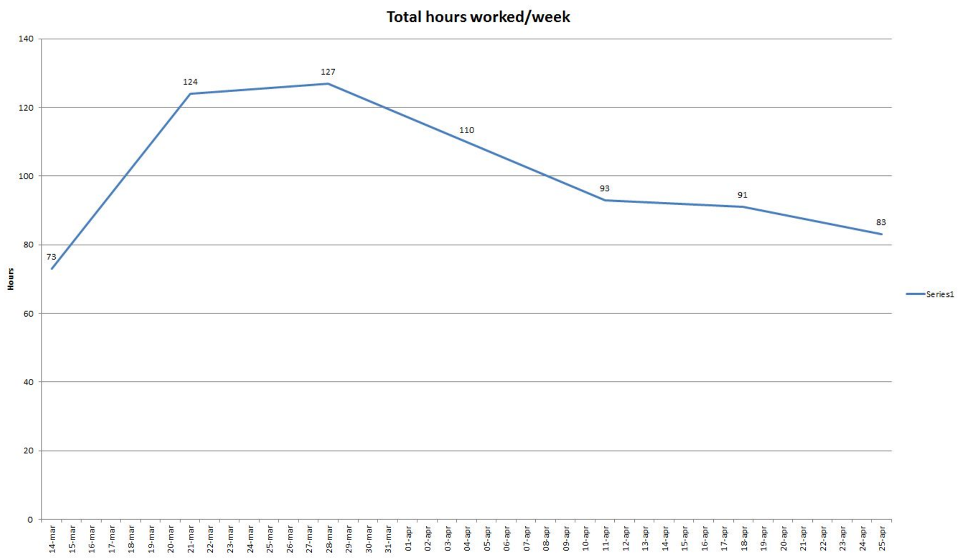


Figure 21: Total hours reported for the group per week.

2.5.8 Perception of the Process

All group members felt that they would like to work in Scrum-based projects in the future, and that the task assignment system clarified who was supposed to do what and when. The group also thought that more time working together, such as in scheduled "hack-a-thons" would have been preferable over each person working alone.

3. Quality

This part of the document explains our verification and validation plan of the project, and describes why and how decisions were made.

3.1 Finalized Project Requirements and Specification

The final project objectives following discussion within the team as well as extensive discussion with our client were the first and second initial major objectives, specifically the replication of physics simulation in the Havok demo framework as well as the visualization of bandwidth usage. We also made our own research objective, which concerned the education of the team members in C++, Havok and RakNet.

The replication of physics simulation in the Havok demo framework works well, and has been tested by several different team members and confirmed to be working. This is the most key aspect of the project goals and was imperative that we achieved, else we've had no ground to stand on in order to achieve the other objectives of the project.

The visualization of bandwidth usage was the second major objective, and all the features are implemented and working. The major visualization features are:

- Bandwidth statistics graph, displaying bandwidth data.
- Body bandwidth graph, displaying bandwidth usage of the three bodies with the biggest bandwidth impact.
- Dynamic coloring of bodies based on their bandwidth impact. Bodies are given a color based on the size of their received network packets. The color is chosen based on a packet size threshold basis which is configurable in real time when running the simulation.
- Received network packet tracking for a selected body. The user can select a body and view the packets received for the selected body.

There were some small bugs concerning the second graph which caused some values in the graph to freeze in place, resulting in small clutter. This issue was later resolved and this major objective is deemed fully completed.

Finally we have the research objective, which concerns mostly the code language C++, as well as the libraries Havok and RakNet. While it is to varying degrees, it can be confirmed that we've all seen a major increase in our understanding of C++. The members which had not used C++ before in particular have seen a sharp rise in their understanding of the language, and while this is expected for a project based on C++, the achieved degree of understanding is exceptional. Our understanding of Havok and RakNet has followed suit, and has also increased massively, seeing as none of us had any previous experiences working with the libraries.

Project Objectives over the course of the project.

Project Objectives	Due Date	% Completed	Deliverable Status
Milestone 1			
● Research Phase	18/01/15	100%	Completed
Milestone 2			
● Replication of Physics	18/02/21	100%	Completed
Milestone 3			
● Visualization of Bandwidth	18/03/21	100%	Completed

Table 5: Project objectives.

3.2 Risks and Issues

During the start of the project, several risks and potential issues were identified and many of them are common in regards to software-development in particular. They mainly concerned technical, programming, architectural and functional requirements. As in all kinds of projects, there were also organizational, scheduling and other personnel related risks in this project as well. Identified risks are shown below, and throughout the process the project leader continuously made risk assessments and appropriate adjustments were made accordingly to counter potential pitfalls. During the project the team came with insights, and in particular the lead programmer, who has a great overview of the programming progress together with the scrummaster, as well as our contact client side, Per Hugoson.

Other specified risks of the project were important to acknowledge and handle, and during the assessment of risks in the project, they were graded as below, and counter-measures issued.

3.2.1 Project Risk Management

Project Risk Management

Risk ID	Risk and Description	Risk Chance	Risk Impact	Risk Priority	Risk mitigation
Risk #1	Communication failure	High	High	High	Slack for communication and Asana used for task-planning.
Risk #2	Estimation and scheduling	High	Medium	Medium	Revamped team structure, Scrum experience

Risk #3	Team members not contributing	Medium	High	Medium	Individual tasks, regular team-meetings.
Risk #4	Incapability of reaching milestones	Medium	Medium	Low	Using help from Avalanche, meetings and getting feedback.
Risk #5	Technical skill too low in team	High	High	High	Research and educating each other
Risk #6	Low team-spirit	Medium	High	High	Ensuring that everyone gets heard during meetings, project leader has individual contact with team members.
Risk #7	Misunderstanding of requirements	Low	High	Low	Continuous contact with the client to ensure correct requirements.

Table 6: Risk management

Risk #1: Communication failure To ensure a good communication-protocol, a number of practices was put in place to reduce the risk of malfunctioning communication. Slack was used since the start as our only text-based communication, which ensured fast and secure communication. A potential pitfall when using Slack is that there will be a lot of messages, and it's easy to lose important information along the way, which is why regular meetings face-to-face was of great importance. During meetings everyone was asked to "check-in" at the beginning of the meeting to give their status, about where they are in the project, how they feel and what their focus is at the moment.

Risk #2:Estimation and scheduling Correct estimations and scheduling is hard to achieve and maintain throughout a project, but the use of Scrum and a revamped team structure made a huge improvement.With only one Lead Programmer, it was much easier to get a good overview of the progress, and it was easier to for the project leader to get a quick estimate on how things were going. We also talked with the clients programmers to get a grip on time frames and initial planning, which was of great help early on in the project.

Risk #3:Team members not contributing To make sure that all members contribute, it's important that every member has a value, and that they feel seen. The implementation of Scrum and the use of the task-planner Asana, made sure that every member had personal tasks to attend to. During our weekly Scrum-meetings, the retrospect also served as a platform for when every member could talk about their personal contribution to the project, which was a great way to realise if someone was slacking behind. The project leader also talked to everyone individually before big deadlines, either in person or through Slack to make sure everyone knew what to do, and was contributing and doing their part.

Risk #4: Incapability of reaching milestones To reach our milestones, it was important to use all the help and expertise we could get from our client, as the challenges this project showed mainly concerned new areas of technology where almost all team members lacked experience. The project leader, scrum master and lead programmer constantly had an overview to make sure that the team reached our deadlines in time. It was vital to look for signs of loss of focus from team members, and try to mitigate potential risks as early as possible to hinder a major breakdown and stop in the project. Continuous status-reports during the scrum-meetings made it easier to acknowledge issues early and made it possible for the team to respond accordingly.

Risk #5: Technical skill too low in team There is always a risk that the technical skill is lacking when developing new software, and in this case it was apparent that the team most likely lacked experience. To remedy, we started of the project with a heavy focus on research to ensure everyone had a good understanding about the problem at hand. We had some programming-sessions where our tech lead served as a "on-call" to ensure everyone understands the code, and so that more senior programmers could help the less skilled programmers.

Risk #6: Low team-spirit To avoid low team-spirit, it is important that every member of the team get heard, and that everyone get to know each other. Tuckman describes four stages in group development, *forming, storming, norming and performing* which are different stages a group goes through. In the beginning of the project we found the group in the forming stage where some people are excited and some might be anxious. The roles and responsibility was not fully clear and it was important that the project leader play a dominant role in this stage. Next, the team moved into the *storming* phase. This is where people might push in different directions and conflicts occur. Some might question authority or feel overwhelmed by workload. Gradually the group moved in to the *norming* phase, where people resolved potential differences, and started work more like a team and appreciate other members strengths as well as respecting the project leaders authority as a leader. At the final stage of the project we reached a state of *performing* where work was completed without friction and towards the team's goals, and everyone contributed to the teams progression.

Risk #7: Misunderstanding of requirements Requirements changed somewhat during the project, and it is of utmost importance that the client and project team have the same picture about what is supposed to be delivered. To ensure that we understood the requirements, we asked to get requirements written down early in the project, and the team was in contact with our client throughout the project. We had recurring meetings with the client to ensure a correct understanding about requirements, and potential changes that was needed.

3.2.2 Issues

Project Issues Management

Issue ID	Issue and Description	Project Impact	Priority	Issue Status	Issue resolution
Issue #1	Division of responsibilities unclear	High	High	Finished	Scrum planning and incorporating Asana for task-planning.

Issue #2	Team formation not optional	Medium	High	Finished	Changed team roles to fit the project better
Issue #3	Lack of programming skills	High	High	Finished	Small programming-session with tech lead.
Issue #4	Changes in requirements	High	High	Finished	Backlog prioritization changed

Table 7: Issue management

During the project there has been issues of varying importance and most of them were dealt with early in the project. Asana was a great complement to make the scrum-process easier and gave a great overview of the status of the project.

Issue #1: Division of responsibilities unclear: The project leader introduced the task-planner tool Asana, which solved the issue concerning responsibilities, and made them more clear and easier to grasp for every individual in the project. Once the proper scrum-meetings started, responsibilities were handed out during meetings, and was easy to follow up using Asana.

Issue #2: Team formation not optimal: After the first couple of weeks in to the project, we saw that the need for a separate product owner was in excess and made the responsibilities a bit unclear. We also switched to a single Lead Developer, which helped tremendously in planning and overall performance as a group, and made the team work more efficiently.

Issue #3: Lack of programming skills: Most of the work regarding programming was made by our lead developer, with the help of a handful of the other programmers, and very much so in the beginning of the project. To mitigate and make sure that everyone could contribute with the programming our lead developer had a whole sprint where he was “on-call” just to help the other programmers. He would also explain and give tips during our scrum-meetings.

3.3 Verification and Validation plan

3.3.1. Responsibilities

In this section all the responsible parties for the Verification and Validation planning and process, along with the scope of their responsibility are listed.

1. Project leader: The project leader was mainly responsible for ensuring the correct implementation of all aspects in HavNets verification and validation planning.
2. Scrum management: The scrum management team had the responsibility of making the appropriate assignments.
3. Testing team: The testing team was accountable for testing and debugging the product both during and at the end of the project.

4. Developers: The development team was responsible for providing accurate documentation and reporting bugs or software failures which they might stumble upon during the development phases.

Fault avoidance is always crucial for the project to make sure that it runs smoothly and gives an expected outcome. To make sure we were building the correct product with the correct parameters, communication with the client was key. The management made sure that the project kept focus on fault avoidance and made sure we covered all steps mentioned below:

1. An accurate system specification (preferably formal).
2. Software design based on information and encapsulation.
3. Extensive validation reviews during the development process.
4. An organizational quality assurance to drive the software process.
5. Planned system testing to expose faults and assess reliability

3.3.2. Verification

During the different phases of development, plans were put in place to ensure that the product met the specific requirements which were set at the start of each phase. Plans such as regular test execution, reviews of the newly implemented code and providing documentation for each development phase. Software reliability was taken into account through continuous testing which is explained in detail in section *3.4 Testing.* The flow of the verification process in every phase is illustrated in Figure 22. The process starts by requirements analysis which is explained in detail in chapter *2.3 Functional and Non-functional requirements*. Next step is scrum planning which is the process of creating and assigning tasks to the development team according to the conclusions of the requirements analysis. During the development of the product the necessary documentation and reports were written by the developers and provided to the testing team which is assigned for testing and debugging of the code, and in other words the testing team makes sure there are no errors and the product works as it was intended in the requirement analysis in start of each phase. If no issues were found in this process then the verification was basically done and we started with the next phase. Otherwise the issues were reported back to the leading team which made the appropriate decision on how to resolve them. This cycle kept on going until all the requirements imposed for each phase were fulfilled accordingly.

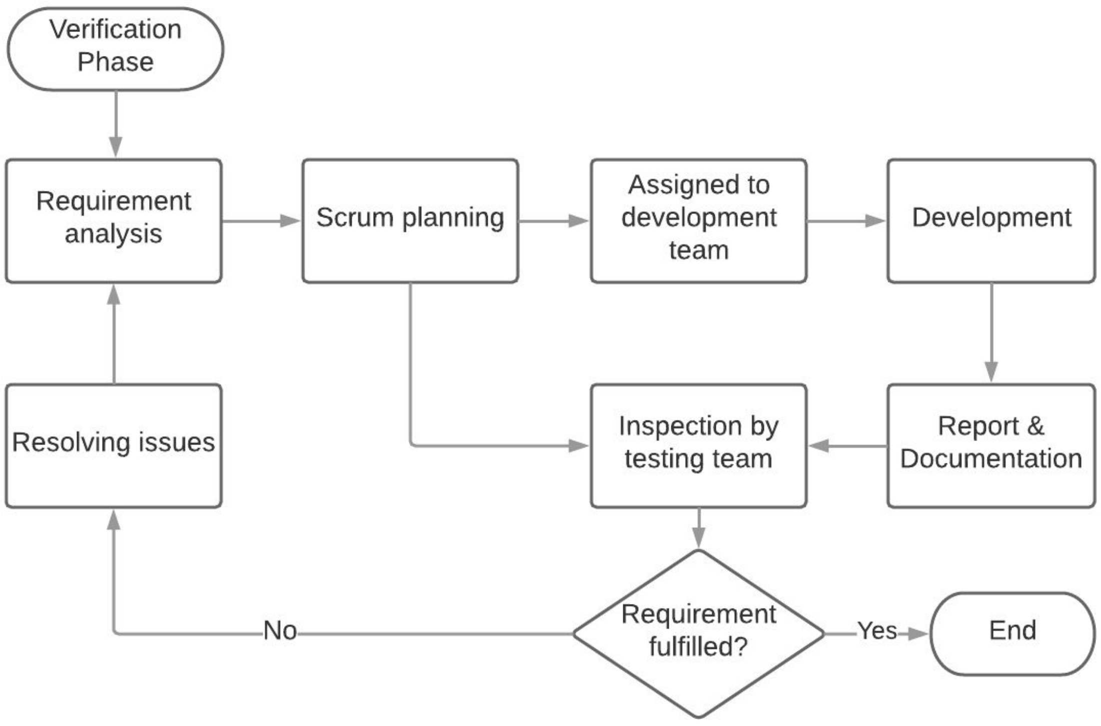


Figure 22: Verification Procedure Flow Diagram

3.3.3 Validation

As the users of the end product are seasoned programmers, with great knowledge of Havok, the main focus during the beginning of the project was to make sure that we had understood the functional business requirement of our client as precisely as possible.
To make sure that we satisfied the client, and not only the requirements, we continuously stayed in contact with our client. Specifically, we scheduled face-to-face meetings with them to get input on our progress and adhere to certain changes that appeared during the project.

During these meetings we ran our program on a laptop and projected it onto a big screen so that the present representatives from our client could have a look and give us feedback. Since HavNet is integrated into the Havok Demo Framework, which our client is well versed in, this setup is to be considered a realistic environment, since this is how they will be using the tool most of the time (on a single computer, testing different physics scenarios). So these meetings functioned both as validation for us as well as for our client. To prepare for the meetings we followed the test plan described in section 3.4.1 and onwards.

We tried to schedule meetings with our client only after certain major goals were met, so that an overall input could be gathered from them rather than input on specific requirements. If there were smaller issues, our project leader contacted our client without the need for a face-to-face. This also helped to keep the distinction between verification and validation clear to us in the group (see previous chapter 3.3.2 Verification).

Table 8 below shows when the meetings were held and which requirements that was demonstrated and approved by our client at that specific meeting (see chapter 2.3 for the requirement codes). At the time of writing of this report, our client haven't had time to actually field test our solution in a "real" environment, e.g. with multiple computers hooked up and actual in-game physics scenarios. This is something that we and the client are looking forward to.

Date	Approved functional requirements	Approved non-functional requirements
21/2	F#1, F#2, F#3	
4/4	F#4, F#5, F#8, F#12, F#13, F#14	
26/4	F#6, F#7, F#9, F#10, F#11, F#15	NF#1, NF#2, NF#3, NF#4

Table 8: Approved requirements.

3.4 Testing

Written below is the testing documentation in ten parts following IEEE 829[8] loosely, as testing on certain levels could not be performed without enormous amounts of research into Havok's demo framework given the timeframe of the project. As such, the master test report section of IEEE 829 will be combined with the level test report.

3.4.1 Master Test Plan

Testing was done mostly on a high-level using a testing protocol as a checklist of the programs main functions. Testing on the lower levels using unit testing and integration testing were much harder to perform as the demo framework by Havok consists of many moving parts, while the HavNet code is directly integrated into the demo framework. All tests done with the testing protocol were done using acceptance testing.

3.4.2 Test Plan

No documented tests were performed using unit testing or integration testing.

The documented testing is done using acceptance testing with a custom-made testing protocol specifically tailored to the goals of the project. The testing protocol consists of twelve different test cases each testing different functionalities of the program that are necessities to the goals of the project. The testing protocol is available for all project members on Git, and the tests are recorded manually in a Google Sheets document also available for all project members. The scheduling between each tests is quite lax and can vary heavily, however our aim was that no more than seven days should pass between each test, as then a risk of not identifying problems in time would exist.

[8] https://standards.ieee.org/findstds/standard/829-2008.html, 2018-04-30

The items being tested with the testing protocol is only the finished executable. The features tested here are the connection between server and client, the profiler tool of HavNet, starting parameters and file writing system. The only tasks necessary to perform are the twelve cases listed in the testing protocol. While every person has access to the testing protocol and the testlog, only one person is assigned to actively make sure tests are being performed. The other members of the project can help with testing if they notify this assignee, otherwise the testing is left alone to this one person.

The testing is mostly done to assure that the product is functional. If any of the test cases fail, then the product does not meet the requirements set by our client. Additionally, if the intervals between each test pass is larger than seven days then there exists a larger possibility of a test case failing, which creates more issues with identifying when and where the issue occured.

3.4.3 Test Design

The most important aspect of designing the tests are that they meet the requirements presented by Avalanche Studios. The tests are designed to verify that the connection between the server and client are stable and that the visualization of bandwidth works under various different circumstances. The testing protocol was the choice to handle testing since the project members were mostly familiar with following a testing protocol with instructions on what should be done and what the expected result should be.

3.4.4 Test Case

The testing protocol consists of twelve test cases:

1. Run the server and the client in the *Magnet* demo. Press the arrow keys on the server. Verify that the magnet moves as expected on both the client and server.

   The pass criteria for this test is simultaneous movement of the magnet on both server and client.

2. Run the server and the client in the *Magnet* demo. Press H in the HavNet menu, in both server and client. Verify that the bandwidth graphs appears. Press H to close. Verify the bandwidth graphs disappears.

   The pass criteria for this test is simply verifying that graphs show up and disappear upon pressing the H key on the keyboard.

3. Run Demos_x64-vs2015_debug.exe --role=server. Verify that the window title is "Havok (Server)" and that no client or server prompt is shown.

   The pass criteria for this test is identifying that the prompt “(C) or (S)erver” does not show in the command line and that the server is running.

4. Run Demos_x64-vs2015_debug.exe --role=client. Verify that the window title is "Havok (Client)" and that no client or server prompt is shown.

The pass criteria for this test is identifying that the prompt “(C) or (S)erver” does not show in the command line and that the client is running.

5. Run the server and client in the *Change Shape* demo. Enable the bandwidth graphs on both instances. Verify that the "Received" graph on the client roughly corresponds to the "Actual sent" graph on the server.

   The pass criteria for this test is checking on both the server and client the bandwidth graphs shows when pressing the H key, and that the server’s package sent roughly matches the client’s package received (roughly because the graph updates every 1/60th of a second).

6. Run the server and client in the *Change Shape* demo with parameter --syncfreq=1. Enable the bandwidth graphs on the server. Verify that "Actual sent" is about 400 bytes/second.

   The pass criteria for this test is to check that the client updates every second of what is being displayed on the server (instead of every 1/30th of a second by default), and that the server’s sent statistic is roughly 400 bytes per second.

7. Run the server and client in the *Change Shape* demo with parameter --syncfreq=1. Enable the bandwidth graphs on the server. Verify that "Actual sent" is about 7 kilobytes/second.

   The pass criteria for this test is to check that the client updates every second of what is being displayed on the server (instead of every 1/30th of a second by default), and that the client’s sent statistic is roughly 7 bytes per second.

8. Run the server and the client in any demo. Verify that the files sClient.csv and sServer.csv are created in the current directory.

   The pass criteria for this test is to check that upon running any demo that files are being written to the disc. These files contain data taken from RakNet’s library. Note, since writing this protocol the file name and saving directory has changed. However, the content is still the same and the testing case still stands.

9. Run the python script HavNetBandwidthPlotter.py. Verify that four plots are shown in one figure.

   The pass criteria for this test is run the Python script, choose two .csv files generated by HavNet and verify that plots are being drawn accurately to the files. Note, this also changed since the writing of the protocol and now does not automatically draw four plots. Instead files have to be chosen manually for the plots to show up.

10. Run the server and client in the *Gravity field* demo. On the server, press enter, 3 to restart the demo. Verify that it looks restarted on the client too.

The pass criteria for this test is to simply restart the demo on the server and verify that the client's view is also restarted.

11. Run the server and client in the *Attach detach* demo. Start clumsy.exe. Check *Drop* and set *Chance(%)* to 50.0. Verify that movement on the client is very jerky.

    The pass criteria for this test verify both that Clumsy works as intended and that the program correctly handles packet loss with the jerky movement.

12. Run the server and client in the *Attach detach* demo. Enable bandwidth graphs on both instances. Start clumsy.exe. Check *Drop* and set *Chance(%)* to 50.0. Verify that "Received" on the client is about half of "Actual sent" on the server.

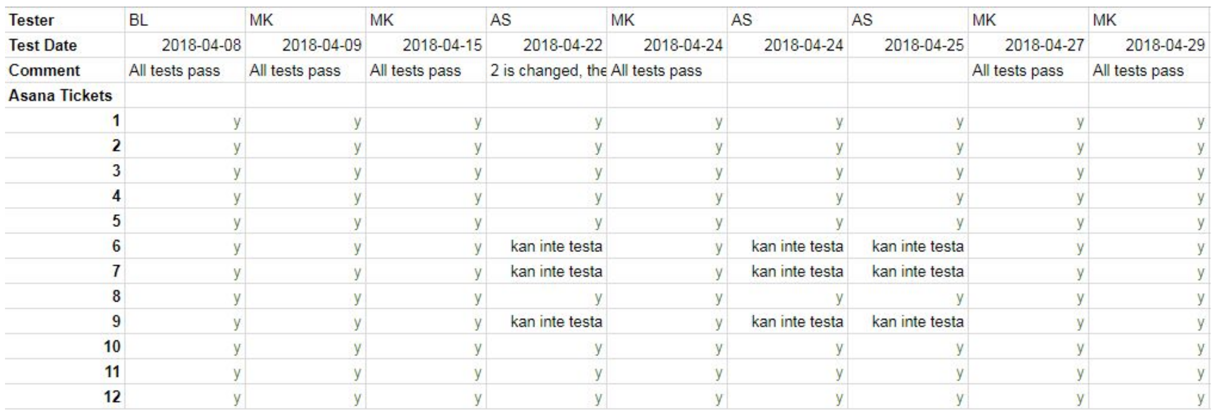
    The pass criteria for this test is to check that Clumsy works as intended and that the packet loss simulated is accurately represented on the client.

3.4.5 Test Procedure

Before testing, the program needs to be built using Visual Studio. Additionally, the matplotlib library needs to be installed with Python as otherwise one test case is undoable. The actual testing begins when the executable is ready to be run, and every test is run in the order they are listed in the protocol. Each test starts with no instances of the executable being run, then following the instructions a result should be obtained for each test. If the result is the expected yield then it gets marked as pass, otherwise fail.

3.4.6 Test Log

The test log contains all relevant information on who did the tests, when they were done, what tests passed alternatively failed and comments on the general testing. They do not contain additional comment on why the tests failed however, but this was controlled offscreen, see further in anomaly report on the issues. The tests listed are the same listed in test case and in the same order, see Figure 23 for more details on the look of test log.



Tester	BL	MK	MK	AS	MK	AS	AS	MK	MK
Test Date	2018-04-08	2018-04-09	2018-04-15	2018-04-22	2018-04-24	2018-04-24	2018-04-25	2018-04-27	2018-04-29
Comment	All tests pass	All tests pass	All tests pass	2 is changed, the	All tests pass			All tests pass	All tests pass
Asana Tickets									
1	y	y	y	y	y	y	y	y	y
2	y	y	y	y	y	y	y	y	y
3	y	y	y	y	y	y	y	y	y
4	y	y	y	y	y	y	y	y	y
5	y	y	y	y	y	y	y	y	y
6	y	y	y	kan inte testa	y	kan inte testa	kan inte testa	y	y
7	y	y	y	kan inte testa	y	kan inte testa	kan inte testa	y	y
8	y	y	y	y	y	y	y	y	y
9	y	y	y	kan inte testa	y	kan inte testa	kan inte testa	y	y
10	y	y	y	y	y	y	y	y	y
11	y	y	y	y	y	y	y	y	y
12	y	y	y	y	y	y	y	y	y

Figure 23: Test log

3.4.7 Anomaly Report

Only three tests did not pass throughout the testing process, these being the synchronisation frequency tests and the bandwidth plotter test. What is consistent with the issues however is the tester being the same every time, and it is always the same three tests with the reasoning being “could not test”. In more detail, the synchronisation frequency tests require two instances of executables with parameters being used and Visual Studio only allows one instance of the executable being run with parameters. For instance, the role parameter tests (4 and 5) also require pre-execution parameters, however only one instance of program needs to be executed which is easily handled by Visual Studio’s debug feature. The workaround is to use command-line to test these, but the tester had too many issues with his Windows installation (Windows on a MacBook) so he could not get the test to work. The final test was an issue with Python and matplotlib not being installed correctly. If the matplotlib library is not installed for Python then the test is undoable, which is the case here.

3.4.8 Interim Test Status Report

Some tests have been slightly modified since the writing of the protocol. For example, in test 8 the program initially wrote two files on the same directory as the executable. This changed later on with the project requirements being changed, so that two files being written have different names and are placed in a different directory as well. However, since the files being written contain the same content as before, and the test was to verify that files are being written on to the disc in the first place. The passing criteria for the test changed slightly, but the functions being tested were the same as before.

Another test that has seen changes is test 2, which tested if the bandwidth graph was being displayed. Initially the hotkey to show the graph was “enter then up”, but later on it got changed to simply pressing the H key. This is reflected upon the testing protocol, as confusion could be caused as the initial hotkey commands “enter then up” does nothing now in the program.

Finally test 9 has also seen changes since the writing of the protocol. Initially, simply the execution of the script would show the four plots in one figure. Now, the graphs have to be chosen manually and it is also possible to choose between server and client statistic of two files, or the difference between two graphs. The main difference now is that some additional steps have to be done between executing the Python script and getting the four plots, otherwise the test is the same.

3.4.9 Test Report

Surprisingly, 75 percent of the tests have never failed during testing, which equates to nine out of twelve tests. Never once since the testing has started have these nine tests failed. This could be due to extensive testing before pushing code into Git, or that in certain cases an issue that would affect the result of tests has been resolved between the testing intervals. Regardless, this is a very positive result as then most of the requested features by our client have always worked during acceptance testing.

The only tests that did “fail” were the two synchronisation frequency tests and the bandwidth plotting with Python test, and it only “failed” from one tester. What could be improved in this regard is more communication and instructions on what is needed to perform all of the tests, as matplotlib is not a default library in Python and how the synchronisation tests should have proper instructions on how to run the parameters. This could cause concern with how Avalanche Studios would handle our program, but the most important part was that the tests were not actual fails, rather that they could not be performed.

Since there was no specialised hardware to test on, it was quite difficult to verify that testing results were exactly the same on all hardware (three different hardware setups were used in testing). This could cause difference in how each tester viewed whether or not each test was actually a pass, as some testers could be stricter than others in what classifies as a pass. The testing effort could also be considered lacklustre in some parts, as there was a long pause between certain test attempts and with one tester not all tests could be performed.

Overall the testing could be considered far more successful than it would a failure, as most functionalities that were requested by Avalanche were passing the tests. Those tests that did not pass had more issues with a tester not being able to perform more than them actually having faulty code, and even then, those tests that failed still passed with other testers.