BACHELOROPPGAVE:

Accommodating a Game Engine for Community Adoption

FORFATTERE:
Joakim N. Stien
Sverre B. Stensby

DATO:
15.05.2015

Sammendrag av Bacheloroppgaven

Tittel:	Tilpassning av spillmotor for adopsjon hos open-source samfunnet		Nr: 25
			Dato: 15.05.2015
Deltakere:	Joakim N. Stien		
	Sverre B. Stensby		
Veiledere:	Mariusz Nowostawski		
Oppdragsgiver:	Suttung Digital		
Kontaktperson:	Asbjørn Sporaland, sporaland@suttungdigital.com, 95080723		
Stikkord	Åpen kildekode, spillmotor, Nox, Suttung Digital		
Antall sider: 70	Antall vedlegg: 8	Tilgjengelighet: Åpen	

Kort beskrivelse av bacheloroppgaven:
Oppgave om tilpasning og utvidelse av spillmotor for å gjøre den til et bedre alternativ for utvikling av spill. Vi har lagt til partikler, nettverk, normal og specular maps. Vi har forbedret implementasjonen av lyd, skyggegenerering, og API.

fontsize1416Summary of Graduate Project

Title:	Accommodating a Game Engine for Community Adoption		Nr: 25
			Date: 15.05.2015
Participants:	Joakim N. Stien		
	Sverre B. Stensby		
Supervisor:	Mariusz Nowostawski		
Employer:	Suttung Digital		
Contact person:	Asbjørn Sporaland, sporaland@suttungdigital.com, 95080723		
Keywords	Open source, game engine, Nox, Suttung Digital		
Pages: 70	Appendixes: 8	Availability: Open	

Short description of the main project:
Thesis on improving and expanding a game engine, to make it a better alternative for game development. We have added particles, network capabilities, normal and specular maps. We have improved the implementation of the audio, the shadow generaton, and API.

Preface

We would like to thank Mariusz Nowostawski for his guidance, and for supervising our project. We would also like to thank Magnus Bjerke Vik and Asbjørn Sporaland and the rest of Suttung Digital for providing the task, engine and helpful input. We would also like to thank Simon McCallum for supporting us trough our time at Gjøvik University College.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Task Presentation

Suttung Digital is an independent game developer residing in Gjøvik, Norway. Suttung Digital is developing a game engine called the Nox engine for internal projects. The engine has been released as an open source project. See the discussion in Section 1.8 for a more detailed description.

The original task description initially gave us the impression that we were to *optimize* a game engine. In meetings with Suttung Digital, it became clear that what the task was really about was to *improve* the game engine by adding new features **and** optimizing existing ones. We changed the subject of the thesis to *"Improving the Audiovisual Aesthetic Presentation of Nox"* before deciding on our final subject in the middle of March; *"Accommodating an Open Source Game Engine for Commmunity Adoption"*.

Suttung Digital wishes to attain a hobbyist and professional community around the Nox engine. One of the visions for the engine is to have a *module* system, through which user created modules for Nox can be freely distributed. The module system would be a crossover between a Linux package manager and other community distribution channels like the Unity Store for the Unity 3D Engine. In order for such a system to exist, a relatively large user base is required.

Our task is to *accommodate the Nox Engine for community adoption*. We consider this task to be similar to *selling* the engine. No money will change hands, nor will any advertisements be displayed. When "selling" an open source game engine, the currency received is a feeling of accomplishment and community improvements on the engine.

We believe, through experience and reasoning, that there are three general steps in committing to a third-party library solution of any kind. The time spent in step 1 and 2 are proportional to the significance a commitment to the library will result in. A library to parse XML will generally have a low significance compared to a fully fledged game engine, as it can be easily replaced should it fail to satisfy. A game built on top of a game engine is almost always *locked* to that particular engine. The three steps are:

1. Interest
   The software peaks the interest of the user through impressive features, demonstrations or portfolio.

2. Investigation
   The user gets his first deep impression with the software by either playing with demonstration applications, implementing simple applications or reading the documentation.

3. Long term commitment
   The user was satisfied with the investigation, and decides to commit to the solu-

tion in a project.

In the above model, step 1 and 2 are the steps where potential users are most likely to decide against using a software solution. If they reach step 3, where they commit to the solution, they are less likely to abandon the solution in favour of something else. Our primary objective is therefore to reinforce the experience of steps 1 and 2. In direct terms of a game engine, we believe to have identified the key aspects of getting a user hooked *(interest)*, and being satisfied with the investigation.

- Interest
  - Wide variety of advanced features
    A wide feature set increases the likelihood that the user will continue investigating.
  - Portfolio of released games using the engine
    A portfolio of released games shows that someone else have already committed to the engine and succeeded.
  - Branding and previous knowledge
    Having some previous knowledge of the engine, to literally any extent, increases the likelyhood that a user will decide to further investigate it.
- Investigation
  - Ease of use
    The user needs to be able to get something quick and dirty running in 20 minutes or less. This could be something as simple as compiling the library or a demonstrational project.
  - Documentation
    The documentation needs to be as complete as possible, showing not only that the developers of the engine are serious, but also that if the user gets stuck, he may figure it out easily.
  - Size of user base
    The size of the user base shows that other people have already committed to the engine. This implies that the engine is in good shape and that help is easy to get.

1.2 Intended Audience

This report is aimed at those who wish to learn about open source development, game engine development and combining these two to attract a larger audience. Readers are expected to be in the very least familiar with object oriented programming and software design. The report does not go in depth of *what* can be achieved with the developed features (or the Nox engine in general), but rather how the features were implemented, and how they benefit Nox in attracting potential users.

The work we have done is aimed towards the users of the Nox engine, both present *(Suttung Digital)* and future. We believe that our contributions to the engine can help Nox attract hobbyist developers, which hopefully will further contribute to the engine.

1.3 About the Authors

Both of us have attended the course Game Programming at Gjøvik University College since 2012. Neither of us have worked on any open source projects of significance in the past, but both of us have a fascination for all the amazing projects that has been developed as open source projects with collaborators spread across the globe. Neither of us have prior to this project contributed significantly on an open source project with *other* contributors. Actually doing this was something we both wanted to do. We had not used the Nox engine previously, but we had a rough overview from the details described in Pyroeis [1].

In order to perform the tasks that were required, we would need to have in-depth knowledge of network programming, network protocol design, graphics programming *(OpenGL)*, and getting acquainted with a large, existing code base.

From previous projects, Joakim had knowledge of network programming in games and other utility applications, and some experience in network protocol design for games. Sverre had knowledge and experience of programming game logic, physics and graphics, but no experience with expanding a pre-existing project of this scale.

The both of us have been exposed to OpenGL via courses at Gjøvik University College, and we have used a handful of game engines and frameworks for hobby and school projects - most notably Cocos2D, Game Maker, Unity 3D, SFML and SDL.

The most challenging, and quite possible the most valuable element we needed to learn was to simply *read and understand code*. Code someone else has written. Code with missing or outdated documentation. Obviously, this was also the first thing we needed to do in order to ascertain *what* needed to be done *where*. This element is also tightly knit with another aspect neither of us had been previously exposed to, *complying with **all** standards defined for the code base (documented or not)*, and *sticking to the existing architecture and patterns*.

1.4 Thesis Scope

The scope of this thesis is limited to features we deem attractive in a game engine. This concerns primarily audiovisual effects and general features, but usability and performance improvements within reasonable frames are also included.

1.5 Project Roles

Our group structure has been quite flat, as a team of two people leave little room for leadership. Disputes and differences in opinion have been settled via discussion rather than via a supreme authority.

All code and work written by both members of the group was equally shared. However, when working on multiple, disjoint features, it's natural that one person takes additional responsibility in designing, implementing or testing a feature. Joakim led the development of the stencil generated shadows (Chapter 3), audio improvements (Chapter 6), the simplified API improvements (Chapter 7), and normal & specular maps (Chapter 4). Sverre led the development of the particle system (Chapter 5). The Nox Networking Protocol was designed by Joakim, while the implementation of the network system was a shared effort.

1.6 Terminology

- Game Session
  Game session refers to a period of time in which two or more players are connected from different game clients to the same server.
  I am going to join this game session and play a match.
- Lobby
  The *lobby* refers to **all** connected clients in a networked *game session*.
  I am going to send this packet to the lobby.
- Synchronized Actor
  A *synchronized actor* is an actor which is **owned** by one game client, and **mirrored** on the rest of the lobby.
  The avatar of Alice is controlled by Alice's game client, and mirrored on Bob's game client
- NNP
  The Nox Networking Protocol. The term refers to the logical data flow and rules that govern the communication between connected game clients in a game session.
  The NNP is divided into two main phases.
- AABB
  AABB refers to an *Axis Aligned Bounding Box* - a rectangle whose horizontal and vertical lines are perfectly perpendicular to the X and Y axes respectively. The size of the box perfectly encapsulates the entirety of a shape.
  The AABB of a circle with $r=1$ *is a square of size* $2x2$.

1.7 General Requirements

When working with Nox, we set some guidelines and rules for ourselves. Some of them, like conforming to Nox's already defined standards, were required to make Suttung Digital even consider the developed features.

When working on an existing code base, it is important that new additions adhere to the existing rules and guidelines for submissions. For Nox, this includes conforming to the existing code standards (see Appendix C). Suttung Digital already has games built on top of Nox, meaning that changes and additions should not break existing APIs or features, unless the benefit of doing so greatly outweighs the work required to leave the existing APIs untouched. All features should be developed with the design philosophy of Nox in mind.

All *new* libraries integrated with Nox must be open source. This will increase ease of adding support for mobile platforms, or platforms running on other architectures than x86 in the future.

The Nox engine originally supported Mac OS X, Windows and Linux. All our features, included libraries and code must function identically on all of these three platforms.

1.8 Introduction to Nox

1.8.1 History of Nox

The Nox Engine was initially developed by Asbjørn Sporaland, Magnus Bjerke Vik, Tomas Klungerbo Olsen, Håkon Nordling and Edvin Avdagić all of whom are previous students at Gjøvik University College. The initial engine was developed for the course IMT3601 *(Game Programming)*. Magnus Bjerke Vik and Asbjørn Sporaland continued the development of the engine for their bachelor's thesis, Pyroeis[1]. For the Pyroeis project, Nox was integrated directly into the game as a monolithic structure. It was later extracted from the game, and published as a standalone project. April 1st, 2014, the five original developers and Jakob Solheim founded Suttung Digital, an independent game studio, where the development of Nox continued. Nox was made public as an open source project licensed under the MIT License [1] on *bitbucket.org* in January 2015. Nox is currently being used for Suttung Digital's upcoming game release, Astral [2] [3] and other internal projects.

1.8.2 Nox Design Philosophy

To allow game designers to change parameters in a game without constantly recompiling, Nox focuses primarily on defining aspects of the world via JSON files. Actors, texture atlases, control schemes and game scenes are defined in JSON files.

Nox separates code explicitly via namespaces. Namespaces are used in Nox much like *packages* are used in Java. There are four "primary" namespaces; `app`, `logic`, `util`, and `window`. Most classes exists within one of these four namespaces, and some exists at deeper levels of namespacing. The directory structure of the source code reflects the namespaces.

Nox is designed to be a modularized system, where key components of the engine are interchangeable. Core subsystem classes are therefore always abstracted behind interfaces.

1.8.3 The Relevance of the In-House Game Engine

Game engines are expensive to develop and maintain. As of early 2015, the options for commercially developed and open source game engines available for free [4] are seemingly endless. How can any small developer gaze at the sea of game engines and economically justify developing a game engine from scratch? We will look at some of the reasons a game developer, primarily a small one, may have.

Bugs in the engine

Anyone even slightly familiar with software development knows that bugs are impossible to avoid. Bugs occurring in the in-house engine are relatively easy to identify and eliminate, as the entire team has full access to, and is likely already familiar with, the source code. Open source third party engines are *in theory* equal in this aspect. However, the team is unlikely to have in-depth knowledge and familiarity with the inner aspects of the game engine, making it harder to find and fix bugs without first getting properly acquainted with the code base. Closed source propri-

[1] http://opensource.org/licenses/MIT

[2] https://web.archive.org/web/20150420121031/http://www.indiedb.com/games/astral

[3] https://web.archive.org/web/20140913133344/http://suttungdigital.com/project/astral/

[4] Unreal Engine 4 is free up to a certain gross income level. Unity Engine 5 offers more advanced tools and features as a subscription service.

etary engines are *black boxes* whose internals cannot be studied or changed by the users of the engine.

The best approach for bugs in third party engines, regardless of source code availability, will often be to file a bug report to the development team and wait for a fix. With open source game engines, the users of the engine may collaborate with the engine developers to fix the issue. With closed source engines, waiting for a fix is **only** approach which can be taken[5]. If a developer is stuck on an issue with a very popular engine, it is likely that someone else have experienced the same issue, and there is a solution to be found after a few minutes with a search engine.

The large commercial game engines has been tested on a large variety of hardware. This makes it less likely that there are specific bug that makes it unusable on particular hardware configurations that a studio does not have in it's test environments.

Specific requirements

Occasionally, the developed game has specific enough requirements that existing game engines simply doesn't satisfy the needs. Games such as Just Cause 2 *(Avalanche 2)*, Grand Theft Auto 5 *(Rockstar Advanced Game Engine)*, and the Battlefield franchise *(Frostbite engine)* are examples of where a game's specific requirements and the budget of the game justifies the additional cost of developing a custom made game engine, over fitting the game into existing technologies. However, these are multi-million dollar budget games with high financial ambition, neither of which most game studios have. While all of the aforementioned games are in 3D, which is the strongest pusher of technology boundaries, games in 2D may also have such specific requirements that developing a custom engine is just.

Not Invented Here

It's common to be more comfortable when working with code you have written yourself than working with someone else's. The *Not Invented Here Syndrome* is likely a common basis for the development of in-house engines among smaller independent game studios. As described by Wikipedia [6]:

> "Not invented here (NIH) is the philosophical principle of not using third party solutions to a problem because of their external origins. ... In programming, it is also common to refer to the "NIH syndrome" as the tendency towards reinventing the wheel (reimplementing something that is already available) based on the belief that in-house developments are inherently better suited, more secure, more controlled, shorter development time, or lower overall cost (including maintenance cost) than using existing implementations."

Licensing Fees

There are commonly some licensing costs with a successful game. If the studio expects to sell enough games that the cost of royalties to a third-party game engine developer exceeds the cost of developing their own engine, it would make sense to produce their own engine. With a small developer, this is a less achievable goal.

[5]One might be able to work *around* the problem, but that is not always the case.
[6]http://en.wikipedia.org/wiki/Not_invented_here

1.8.4 Open Source, Game Engines and Nox

With Unity, Unreal Engine and Source recently adopting, to varying degrees, business models centralized around giving away the engine for free, are open source game engines still relevant? Erlend Sogge Heggen, the Community Manager of the jMonkeyEngine certainly things so [2]. He believes that open source game engines "keeps the big guys honest", and that when open source is done right, everybody wins.

> "...as long as we can show that these things can be accomplished by a part-time hobbyist just for the heck of it, the end user gets a fair price (i.e. free!), and our fellow hardcore misfits will continue trying to solve the most difficult problems the industry has to offer"

Erlend Sogge Heggen states that Facebook and Google compete in having the best free and open source framework for web development to ultimately create the best user experience in their own products [2]. Great open source projects attract great developers, which further help develop the frameworks resulting in a better user experience for Facebook, Google, and the users of their open source frameworks. As Sogge Heggen states, *everybody wins*.

For a company to attract high quality developers to their open source project, the first stone to be laid is the *initial investment*. To make someone use and further develop your product, it must first be built. The initial product must show high enough quality to differentiate itself from not only all other related open source projects, but also other commercially available products. Nox is not only competing for users with other open source engines such as LibGDX, jMonkeyEngine and Cocos2D - now, more than ever, is it also competing with commercial game engines like Unity 3D and Unreal Engine.

We think that a small developer like Suttung Digital would not generally benefit from using an in-house engine, as they are expensive to create and maintain. However, when it is an open source engine, it can be used as a marketing tool. The cost of developing and maintaining the engine will still be significant, but releasing the engine under an open source license may prove beneficial for both games released by the studio, and the engine itself. Frictional Games, a studio most famous for developing *Amnesia: The Dark Descent*, and the *Penumbra* franchise, released their in-house engines used for *Penumbra: Overture*, the *HPL Engine 1*[7], under the GNU General Public License (GPL)[8].

1.9 Nox Architecture

The architecture of Nox is inspired by the architecture defined throughout the book Game Coding Complete, 4th Edition, written by Mike McShaffry and David Graham [3]. [9]

As stated in Pyroeis by Magnus Bjerke Vik and Asbjørn Sporaland [1]:

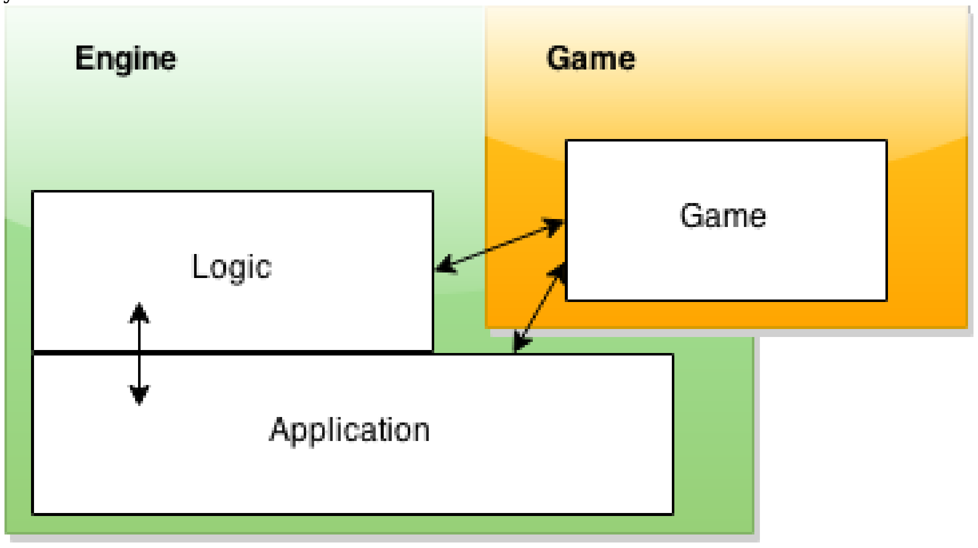
> The architecture of the system is two-layered: Application and logic. Application communicates with the platform (SDL2, OpenGL, OpenAL, and the OS), while

[7] http://en.wikipedia.org/wiki/HPL_Engine
[8] http://www.gnu.org/copyleft/gpl.html
[9] This has been stated my Magnus Bjerke Vik in discussions about Nox.

logic handles the game's states and is platform-independent. In the logic there are three main components: Actor, game view, and event. An actor is an entity in the world. A game view is an interface to the program for a human, ai, or remote player, that often control an actor.

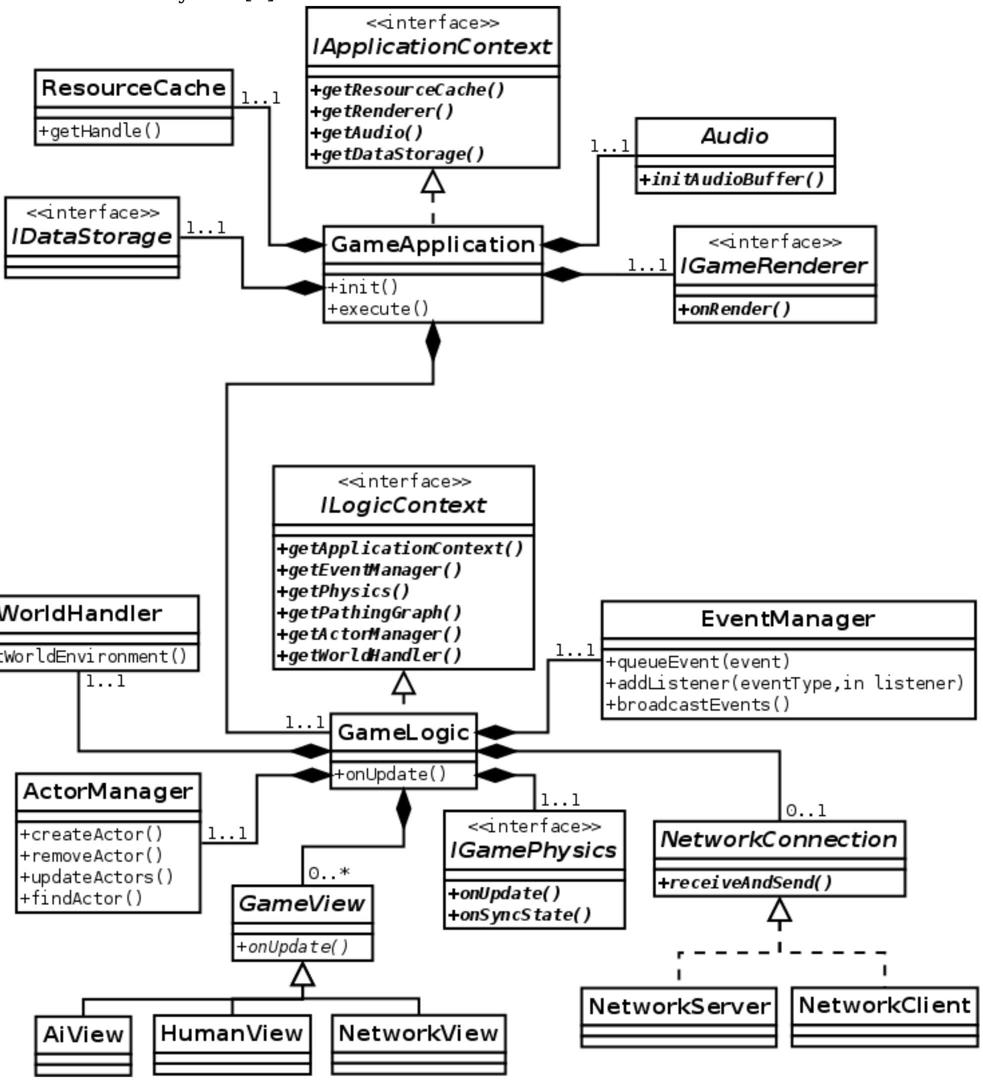
In the original Nox architecture (Figure 1), the game code was considered hierarchically equal to the logic layer. The game code was responsible for a lot of the work required to make the logic and application layers communicate with each other. The changes made to the layer structure and the responsibilities of the game code is discussed in greater detail in Chapter 7.2.

Figure 1: The original Nox layer structure. The game code is hierarchically equal to the logic layer.



Nox is designed to be a modular, loosely coupled system. Implementations of specific frameworks and libraries such as OpenGL and Box2D are abstracted behind generic interfaces. This allows these libraries to be replaced by other libraries providing the same functionality, should this be desired.

Figure 2: The architecture of Pyroeis (Nox). Only the most important classes and methods are included. Many of the names have been changed, but the core design remains the same. Figure taken from Pyroeis[1].



1.10 Considered Features

Multiple features were proposed (either by us or by Suttung Digital) and taken into consideration for implementation. Not only did we not have the time to implement them all, but we needed to focus on the features which would aid the engine in attracting users. One of the criteria we used for deciding whether or not we considered a feature to be *attractive* enough, was to ask ourselves if the feature in itself would be *immediately noticeable*. One of the items which did not pass this test was to refactor the graphics system. We also took into consideration whether or not the feature could be considered to be *expected* in the engine. Particle systems, further discussed in Chapter 5, were one of those features. The time required to develop the feature at hand was also taken into account.

Network and Multiplayer

One of the first features that was requested was networking. Networking is generally expected to *some degree*, depending on the scope of the engine. Game frameworks like SDL and SFML need not provide fully realized networking protocols. The work of implementing network is described in Chapter 2.

Improved Audio Support

Nox originally had a primitive support for playing sound, which they requested to be improved upon. The new audio system is described in Chapter 6.

Particle System

Suttung Digital wanted support for particle systems in Nox. The particle system is described in Chapter 5.

Normal and Specular Mapping

We considered normal and specular maps to be a feature with the potential to raise the grahpical aesthetics of games drastically. The implementation of this is described in Chapter 4.

Use Stencil for Shadows

Based on previous experience from hobby game engines, we believed we could improve the system used to generate lights and shadows by utilizing the stencil buffer found in OpenGL. The implementation is described in Chapter 3.

Simplified API

While using and developing Nox, we noticed that the engine has a relatively high amount of boilerplate and overhead for new projects. We decided to eliminiate as much of this as possible. The simplified API is described in Chapter 7.

Mobile Support

Mobile support was requested by Suttung Digital as a feature for Nox. The two largest platforms, iOS and Android, would be the primary focus of support. Targeting the explosively popular mobile platforms which has become the hobby developer's favorite platform would likely increase Nox's popularity drastically. We decided against implementing mobile support to focus on other features which benefited the engine in other ways.

Physics Based Skeleton Animation

Suttung Digital requested skeletal animations in Nox. Skeletal animations are conjoined physical bodies which move in a pre-defined pattern.

World Editor

The Nox engine relies almost entirely on JSON files to define worlds and the entities populating them. They had noticed that the artists were not as comfortable with working in JSON as they were. The world editor would allow artits and programmers alike to work in a graphical *"what you see is what you get"* environment when designing levels and worlds.

In-game Console for Live Editing

Many games, dating back to the late 1990's, included an in-game console through which multiple debugging properties were available for configuration. Suttung Digital requested this feature as a *live-editing* and scripting-tool.

Rewrite of the Graphics System

Many of our features included modifying the graphics system of Nox. In doing so, we learned that the system was due for a proper overhaul. To paraphrase Suttung Digital’s Asbjørn Sporaland; “It is designed to be fast, nothing else.“

Actor Creation Pool

Suttung Digital proposed implementing an actor creation pool. The "pool" would preallocate (or reuse) Actor instances to avoid having to avoid the overhead of allocating multiple actors during brief bursts of actor creation. The pool would allocate actors slowly in the background, and potentially also reuse old actor instances which are no longer required.

Multithreaded Box2D

Suttung Digital proposed either parallelizing Box2D ourselves, or finding a fork which already did it, and integrating it with Nox.

2 Network

2.1 Network Requirements

We were tasked with implementing online multiplayer features into Nox. We believe that networking is an important feature to include in the engine in order to build *interest* around Nox. As discussed in Section 1.1, we believe more interesting features can help to build a community around Nox. Suttung Digital had few, but well defined requirements for the networking system.

- The solution must support seamless drop-in play
  Alice should be able to invite Bob into her game session without leaving the game or losing game progress.
- The solution must handle **at least** 8 concurrent players
  8 simultaneous players is the absolute minimum number of connected clients the solution must support. The latency and performance should be reasonable for every connected player.
- The solution must support dedicated servers
  Large-scale projects may require dedicated server hardware, which the network solution must support.

Nox is a general purpose game engine, and is not designed with any one game genre or type in mind. This meant that the networking protocol must meet the same criteria; it must fit as well into a real-time strategy game as a side-scrolling shooter or a match three puzzle game.

2.2 Network Architectural Design

The Nox Networking Protocol and API were designed by identifying the *"lowest common denominator"* of multiplayer games and implementing up to and including this point. Advanced and game specific features are therefore not within the scope of the Nox networking system.

In high-level terms, the points we identified as common denominators are:

- Discovering servers running on the local network
  Servers running on the local network must broadcast their existence and connection information. Clients on the local network must be able to connect to the server without automatically entering the IP address of the server.
- Connecting to and disconnecting from servers
  Game clients must be able to connect to a server running the same game with the same version by only supplying the IP address and port number of the server.
- Server side white- and black-lists

The server must support automatic rejection and acceptance based on a configuration file containing *whitelisted* and *blacklisted* players or IP addresses.

- Synchronizing game actors
  The process of synchronizing game actors must be performed automatically by the network system. Details of synchronization are subject to the specific game at hand.
- Synchronizing events
  Through a simple API, events can be sent to the server and broadcast to the whole lobby. The server must be able to validate received game events.

The network system is **not** required to handle advanced lag compensation features which are unlikely to be required or useful for most games. It is therefore very important that the networking solution is extendable, overrideable and scalable to allow for games to implement the missing features.

Requiring players to invite friends into their session seamlessly pointed us initially towards a *Peer-to-peer Architecture (P2P)* in which no client is superior. The additional requirement that dedicated server should be an option pointed us again towards the *Client-Server Architecture (CS)*. To meet both the requirement of dedicated servers *and* drop-in play, we considered two options. The first option would be to modularize the implementation and implement both P2P and CS and require the game implementation to load the correct module at the correct time. The second option was to go fully for the CS architecture and run the server module *within* the game client.

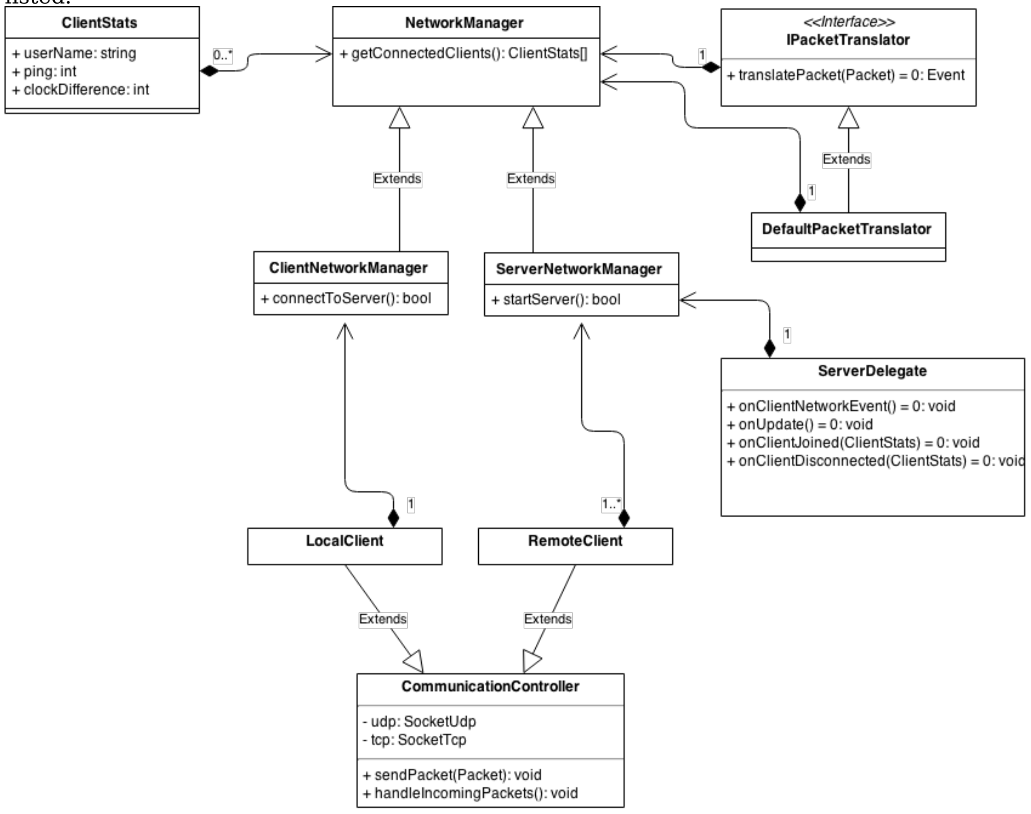
We chose architectural alternative number 2, bundle the game server and client functionality within the game client. A game application process can be any combination of server and client. The four unique *network configurations* are:

1. Neither; playing offline
2. Playing client
3. Dedicated server
4. Server and playing client

2.2.1 Network System Design

To allow for both a server and a client module to be loaded inside the same running process, the two modules share the same game state; the server module reads and acts upon the exact same objects as the client module does. The abstract class diagram depicting the most significant classes of the network system can be seen in Figure 3.

Figure 3: Network system class diagram. Only the most significant classes and methods are listed.



2.2.2 Indistinguishable Network Configurations

From an API perspective, we set out to make the difference between the four *network configurations* (see list in Section 2.1) as indistinguishable as possible. We believed this would provide the best interface to the networking functionality, as the game implementations need not act differently based on which configuration is currently running. We also believe that by not requiring special treatment for each network configuration, it will be easier to integrate network features into an originally intended *pure offline* game.

The difference between the different network configurations is made small largely by the way the Nox engine is already laid out. Almost all of the game logic occurs within the game components, and game-wide communication is done via events - little logic occurs without initiation from either a component or an event.

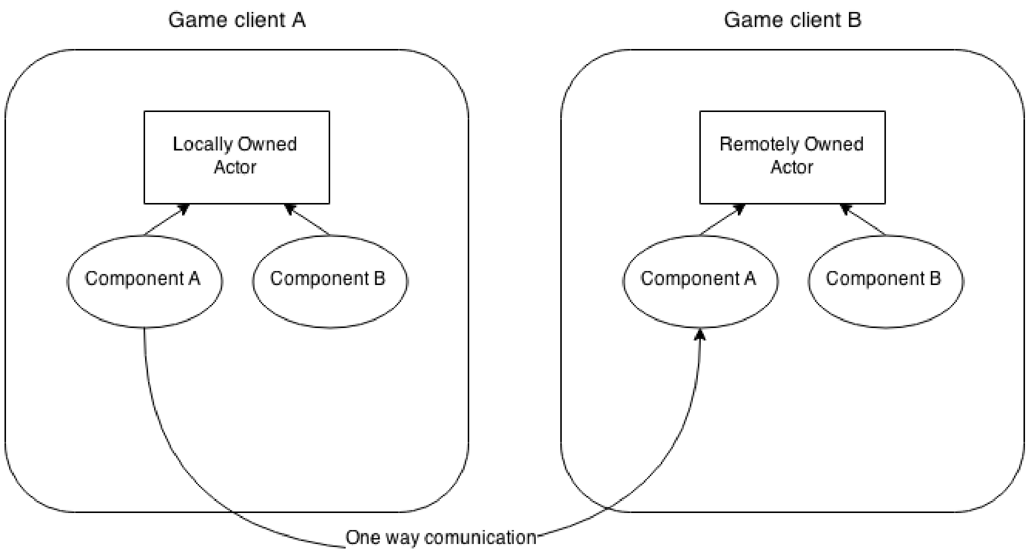
The client module is completely unaware if the server it is connected to is running within the same process or a server half a world away. The server is *aware* if a client is connected from within the same process, but only uses this knowledge to avoid sending redundant data to "itself".

2.2.3 Synchronized Actors

An Actor in Nox is any object in the game world, be it a rock, an umbrella, the player character, or a bullet casing falling to the ground. Actors are what *define* the game. In order to play online, the state of the actors in the game must therefore be synchronized. An actor's state is the aggregated state of all the attached components - in itself, an actor has no attributes of significance in terms of synchronization.

A synchronized actor is an actor that is *owned* by one game client, and *mirrored* on the other connected game clients. A synchronized actor is immutable for everyone but the owning game client[1]. Actors only regard synchronizing a subset of their components - not all components may be deemed necessary to synchronize. The game itself must specify explicitly which components should be synchronized. Components are by default not synchronized.

Figure 4: Logical flow diagram of component synchronization



In Figure 4, the synchronized actor is *owned* by game client A, and mirrored on game client B. The actor only synchronizes the state of it's attached component A. Game client B is unable to directly affect the state of the actor. This does not fully exclude any alterations to the state of the actor from client B or the server; the event channel can be used to *force* client A change the state of the actor. The Component and Event Channel are described in detail in Section 2.2.4.

2.2.4 Channels

The Nox Networking Protocol has two primary channels over which game states can be synchronized. The *Component Channel* and the *Event Channel*.

In an event-driven game engine where most of the game state and logic is encapsulated within components, no actions of significance should occur that affects the game state without ***(A)*** *being initiated by an event or a component*, or ***(B)*** *raising an event or notifying the affected components*. Clicking the "jump" button may result in an event notifying the system of the pressed key, which results in the component controlling the player actor jumping. The Nox Networking Protocol therefore only supports "game initiated communication" over these two channels. The engine is al-

[1]Allowing synchronized actors to be mutable on remote copies is a point of further improvement.

lowed to communicate outside of this constriction. Most notably, this happens during the *initiation phase* of the Nox Networking Protocol, which is described in Section 2.3.3.

Component Channel

The component channel is responsible for synchronizing *synchronized actors* between connected hosts in a session. The component channel is only sending data over the UDP transport layer protocol.

A subset of the synchronized actor's components are synchronized across the connected clients. The owner of the actor serializes the components and sends them to the server at a frequent interval. The server in turn broadcasts all the received component states to all connected clients, bar the originator of the component state.

Components have an interface for writing *to* and reading *from* a network packet. Components are free to perform interpolations based on the estimated age of the packet, and general lag compensation. Components which are continuous in nature *(such as physics)* benefit heavily from compensating for the lag. To accommodate these cases, an interpolation class, aptly named `Interpolator`, was added. The `ActorPhysics` and `Transform` components contains default interpolation methods for synchronization. We expect developers to replace the synchronization in these components to better suit the specific requirements of the game being developed.

Event Channel

The event channel is responsible for transmitting events raised from a game client or the server to one or more counterparts. Clients are only allowed to send events to the server, while the server may send to any subset of the connected clients. The decision whether or not to forward received events from the server is implemented in the game space of the server.

The event channel can use either the TCP or UDP transport layer protocol. If the underlying protocol is unspecified, UDP is used by default.

2.3 Protocol Implementation

2.3.1 Relationships and Identities

It was our goal to completely eliminate the knowledge of *who exactly* is the server. When developing the game logic of a networked game, we intended for there to be absolutely no distinction or awareness in whether a process is *just a client, a client and the server* or *just the server*. Removing this awareness eliminates the requirement to adapt based on the current identity, eliminating extra code, thus potentially eliminating extra bugs.

2.3.2 Protocol Overview

The Nox Network Protocol is divided into two primary phases. The initial phase is the *Initiation Phase*. In the initiation phase, the client and server exchanges information to ensure that the applications are compatible. The second phase is the *Game Phase*. In the game phase, game states are synchronized between the lobby.

2.3.3 Initiation Phase

The NNP initiation phase is the first phase of the NNP protocol. In order to initiate the protocol from a client, the only knowledge required about the server is the IP address and TCP listening port. The goal of the initiation phase is to ensure that the client and server are compatible (i.e. running the same game at the same version), and to create UDP connections. The overview of the NNP Initiation Phase can be seen in Figure 5.

The initial step is the sending of the `ClientJoinRequest` packet from the client. This packet includes information about the client, such as the user name, and the game and version of the application. The IP address of the client, his username, the game, the version of the game, and the number of already connected clients are taken into consideration when the server decides whether or not to accept the client.

As can be seen from the NNP initiation phase diagram, the client creates two UDP sockets. This is required to allow both hosts to have a two-way UDP socket.

1. The client binds a UDP socket listening on any free port, bound outwards to an arbitrary port. The client sends his listening port to the server.
2. The server binds an UDP socket listening on any free port, bound outwards to the listening port of the client. The server sends his listening port to the client.
3. The client destroys the socket created in step 1, and creates a new UDP socket listening on the same port as the old one, but bound outwards to the listening port of the server.

After both hosts have (re)created their UDP sockets, a simple echo over UDP ensures that the sockets are connected to each other. The echo is performed by sending two "empty" packets of type `ClientJoinRequest` and `ClientJoinResponse`.

After the client has been successfully accepted into the server lobby, the previously connected clients are informed of new arrival, and the client is informed of the existing clients already in the lobby.

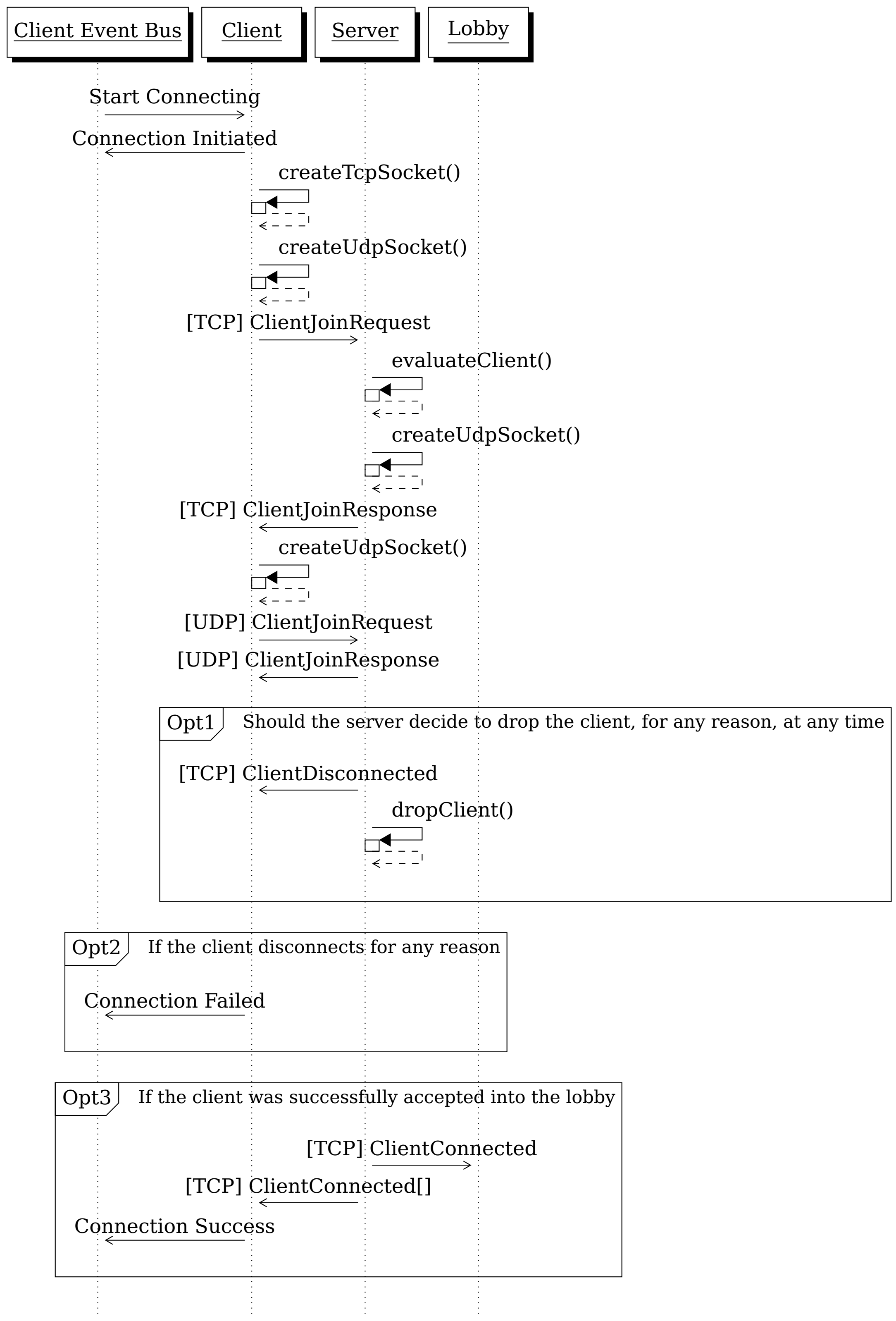


Figure 5: NNP Initiation Phase

2.3.4 Game Phase

The game phase of the NNP is where most of the time is spent. The goal of this phase is to synchronize the game states of the connected hosts, inform the lobby when a client (dis)connects, and continuously estimate the clock difference between the server and the clients.

Continuously during the game phase, a *heartbeat* [2] is executed. The heartbeat is initiated by the server once every second, and consists of three steps. The heartbeat calculates the round-trip-time between the server and the client, and estimates the clock difference between the clocks of the client and server. The clock difference is critical when compensating for lag, as it can be used to accurately calculate the age of any packet sent from any client. In addition to this, the heartbeat ensures that the hosts are able to communicate with eachother with a reasonable amount of latency. If the server deems a client as unresponsive, it may drop the client without warning.

The game can define a *component update frequency*. This value is shared in the common ancestor of the primary client class (ClientNetworkManager) and the server class (ServerNetworkManager), the class NetworkManager. During the game phase, both the client and server send out all available component states at this frequency.

The optional clauses in Figure 5 labeled "opt1" and "opt2" apply to the game phase as well.

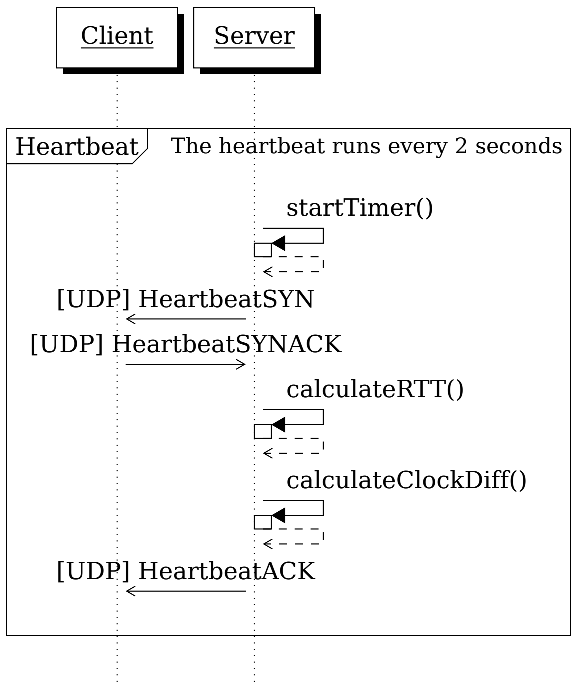


Figure 6: NNP Game Phase Heartbeat

[2] http://en.wikipedia.org/wiki/Heartbeat_%28computing%29

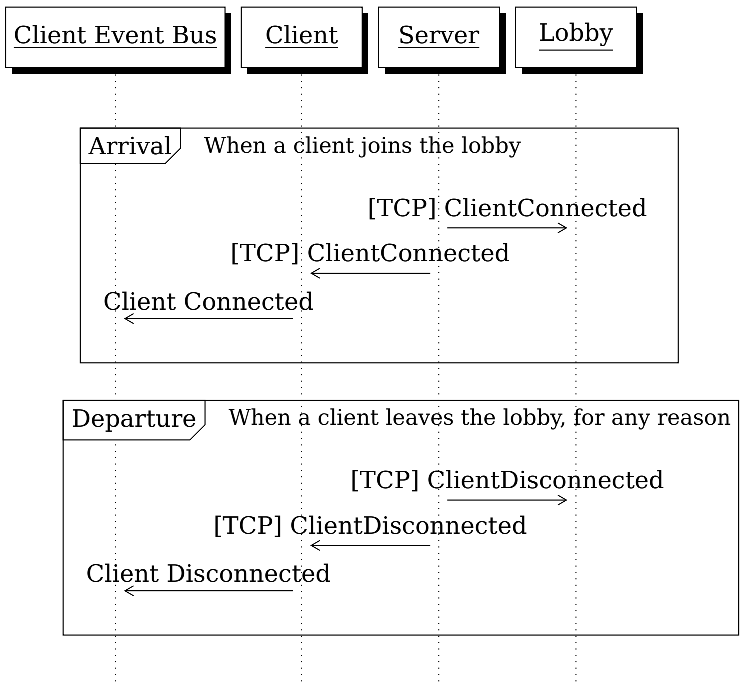


Figure 7: NNP Game Phase Arrival and Departure

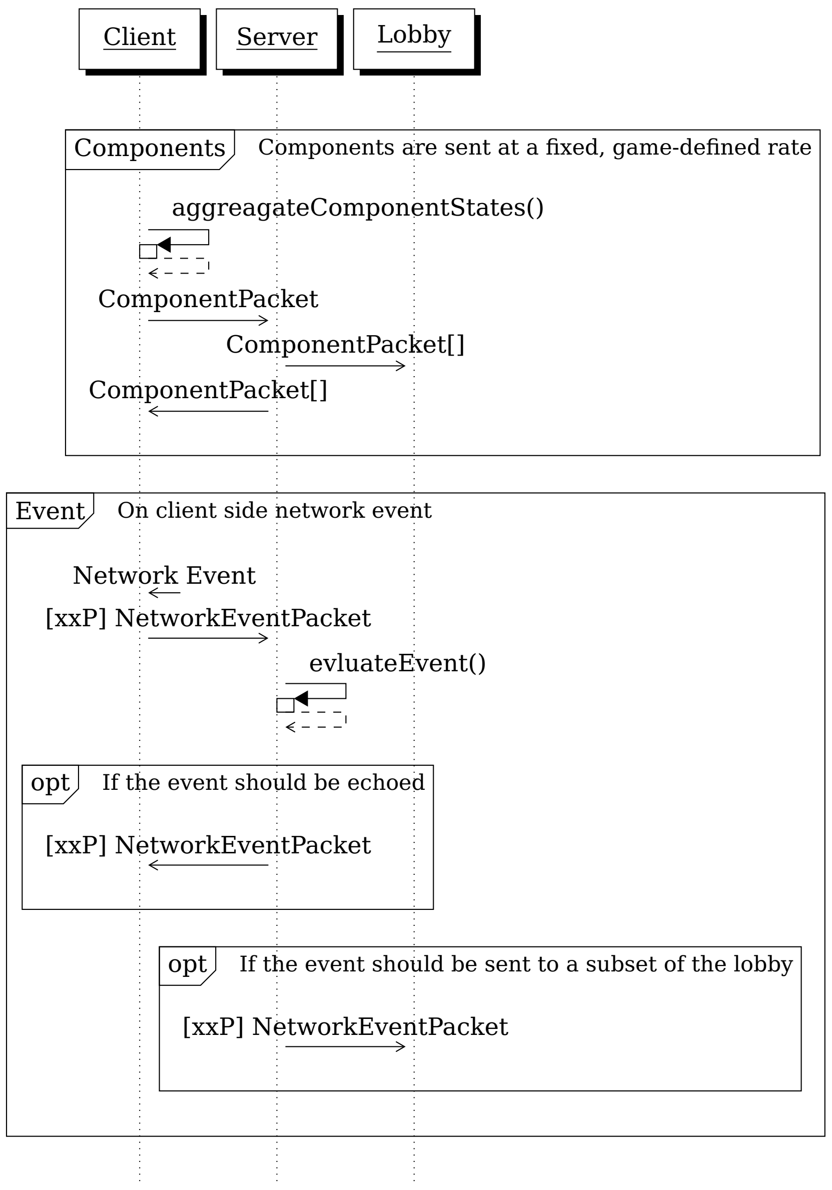


Figure 8: NNP Game Phase Component and Event Channels

2.3.5 Security and General Flaws

The NNP implementation is not encrypted or encoded in any way, all data is sent in clear, uncompressed text. This is obviously a security concern, which should be improved upon prior to using the NNP for commercial or widely distributed products.

There are no measures in place to improve the packet-loss rate of UDP communications in the NNP implementation. A lost packet will be lost forever, as the sender keeps no track of it once sent.

2.3.6 Packet Implementation

The class `Packet` is responsible for storing data, providing access to the stored data, and serializing to and from encoded binary formats.

The packet class supports the most common primitive data types, in addition to some more advanced structures.

Table 1: Supported data types in packets

Type	Size (bytes)	Description
UINT32	4	Unsigned 32 bit integer
INT32	4	Signed 32 bit integer
UINT64	8	Unsigned 64 bit integer
INT64	8	Signed 64 bit integer
FLOAT32	4	32 bit float
BYTE	1	Unsigned 8 bit integer
VEC2	8	Two 32 bit floating point numbers
STRING	1*	Array of unsigned 8 bit integers (null terminated)
ARRAY	4*	Array of unsigned 8 bit integers
PACKET	12*	Binary encoded packet

* The size depends on the content. The listed size is the minimum size, given an empty container.

Packets are able to encapsulate other packets, as seen in the final row of Table 1. There is no limit to the size of a packet, as long as the total size of the final, encapsulating packet is less that $2^{32})$ bytes. This limit is however not enforced.

The packets are *loosely typed*. There is no governing mechanism in place to ensure that a packet with a specified identification conforms to the expectations set to that specific packet type. We consider the development overhead required to predefine the structure of all required packets to be too significant to outweigh the flexibility and speed a loosely typed structure provides. *Dynamically* altering the structure of a packet, as seen in Figure 14, is hard to combine with a strictly typed packet system.

The storage and retrieval API for the Packet class is similar to the APIs for `std::iostream`, as can be seen from Figure 9.

Figure 9: Packet API example code. The contents of the packet are purely demonstrational - Nox has no built in support for chat.

```

void sendChatPacket(std::string message, ClientId recipient)
{
    Packet packet;
    packet << PacketId::CHAT_MESSAGE;
    packet << recipient;
    packet << message;

    sendPacket(packet);
}
```

Binary Format

The packets aims to use as little overhead as possible. An empty packet require only 12 bytes of common data. The first four bytes of a packet contains the total length of the packet. The next 8 bytes contains the timestamp from when the packet was serialized. When serializing fields, a single byte is used as the *field header* to declare the type of the next field. A 32 bit integer will therefore require 40 bits (5 bytes) in serialized form.

2.3.7 Packet Translator

It can be seen in Figure 3 that the Network Manager class owns an instance of an `IPacketTranslator`, and a `DefaultPacketTranslator`. This class is required to allow packets, which as discussed in Section 2.3.6 are typeless, to be translated into strictly typed Event instances. When events are serialized to Packets, they simply stream all their data, including a *type header*, into it. The Packet is then flagged by the network system as an *event packet*. Upon arriving at the remote host, an instance of the original type must be instantiated and initialized with the contents of the packet. This could not be wholly done within the engine itself, as Nox is unaware of the Event subclasses existing in the game's code. The *core* Event subclasses in the engine are handled by the `DefaultPacketTranslator` class.

2.3.8 Dedicated Servers

Dedicated servers are not natively supported by Nox, although the requirements specified it. The problem is that the Server is responsible for issuing commands to create synchronized actors, while the Client is responsible for creating them. When no Client is running within the Server's process, there is nobody to heed the command. When there *is* a Client running within the same process as the Server, the Client creates the synchronized actor within the game state, which is shared between the Client and the Server. As the dedicated server has no synchronized actors in it's game state, it is unable to synchronize the actors it has commanded the creation of.

A potential solution to this problem is to *fake* a Client on dedicated servers. The fake Client would connect to the Server and handle commands like a regular Client, but not be treated as a *playing Client*.

2.4 Network API Design and Usage

The APIs for the network system were developed in accordance with our own general requirements (defined in Section 1.7). In addition to this, the system was designed to require as little input and be as automatic as possible.

2.4.1 Declaring an Actor as Synchronized

An actor can be declared synchronized. This is done in the JSON definition of an actor.

Figure 10:

```

"Player":{
    "components":
    {
        "Transform": { ... },
        "Physics": { ... },
        "Sprite": { ... }
    },
    "synchronized":
    {
        "components":[ "Transform", "Physics" ],
        "destroy": "MyCustomHandler"
    }
}
```

In Figure 10, the actor named *Player* will have it's *Transform* and *Physics* components synchronized across the lobby. When the server decides that an actor of type *Player* should be destroyed, the custom handler named *MyCustomHandler* will be executed. The actor is not necessarily *destroyed*, but it will always be de-synchronized. When an actor is de-synchronized, the network system ceases to syncrhonize the components. The actor can still exist in the game world. There are two default destroy handlers:

- destroy
  The actor will be de-synchronized and destroyed immediately.
- nosync
  The actor will be de-synchronized, but will remain active.

Custom destroy handlers were added to allow for things such as rag-doll physics to be enabled when a character dies, or other custom logic required by the game at hand. Custom destroy handlers are added as lambda expressions:

2.4.2 Network Events

Whenever an event of significance occurs, for instance when a player fires a gun, the originating host should notify the server of the event. The server is free to forward the event to any of the connected clients. The server may for instance also check if the player even *has* a gun before forwarding the event. Events that should be sent to the server are raised like any other event. The special event type `NetworkOutEvent` contains the event to be transmitted to the server, along with the transport layer protocol to send the event over. The network layer listens for events of type `NetworkOutEvent`, and sends it's content to the server upon reception.

If the server decides that a received event was valid, it may broadcast the event to any subset of the connected hosts. When the receiving hosts receives the event

Figure 11:

```

nox::logic::world::Manager *manager = this->getWorldManager();
manager->addSyncDestroyHandler("ragdoll",
    [](Actor *a, Manager *m)
    {
        activateRagdollPhysics(a);
    });

manager->addSyncDestroyHandler("explode",
    [](Actor *a, Manager *m)
    {
        createExplosion(getPositionOf(a));
        destroyActor(a);
    });
```

from the server, they are broadcast like any other event. The receiving hosts' game spaces have no way of telling if the event originated locally or from the server. In the pseudo code showing how to send events over the network (Figure 12), the Gun Component is completely unaware of *who* raised the FireGunEvent.

Figure 12: Sending an event to the server

```
// PlayerComponent.cpp
void PlayerComponent::fireGun()
{
    FireGunEvent fireEvent(getGun());
    NetworkOutEvent netEvent(fireEvent);

    // Send the event to the server
    raiseEvent(netEvent);

    // If we expect the server to be OK with the FireGunEvent,
    // we may raise it locally.
    raiseEvent(fireEvent);
}

// GunComponent.cpp
void onFireEvent(FireGunEvent fireEvent)
{
    displayMuzzleFlare();
}
```

2.4.3 Making Game Critical Decisions

The client/server networking model in games is based around making the server the final decider in all game related questions. In order to incorporate this into the design where the difference between *client* and *server* is nigh indistinguishable (as discussed in Section 2.2.2), the *decider* was inserted into the server space. The decider class, `ServerDelegate`, is overridden by the specific game and given to the `ServerNetworkManager`. The server delegate is responsible for responding to events received from the lobby, defining the size of the lobby, and is notified whenever a client leaves or joins.

As the server runs *within* the game's process, the server delegate has full access to the game state. From it's update call, it is able to make game-flow decisions, such as notifying the lobby that the match is over, or ensuring that clients are behaving properly.

Figure 13: Making server-side game decisions

```
class MyServerDelegate: public nox::ServerDelegate
{
    void onClientNetworkEvent(ClientId sender, Event event)
    {
        if (event.isTypeOf(FireGunEvent::ID))
        {
            // Check if the sender can fire his gun
            if (canClientFire(sender))
            {
                // Broadcast the event to everyone else
                broadcastToEveryoneElse(sender, event);

                ClientId hitClient = getPlayerHitByGun(sender, event);
                if (hitClient != NOTHING_HIT)
                {
                    HitPlayerEvent hitEvent(hitClient);
                    broadcastToEveryone(hitEvent);
                }
            }
        }
    }

    void onUpdate(Duration deltaTime)
    {
        // Ensure that all clients' actors move legally (i.e., not
        // moving at above-allowed speeds).
        for (Client client : clients)
        {
            if (!validMove(client->currentPosition, client->previousPosition))
            {
                // Force the client to it's last valid position
                ForceMoveEvent event(client->previousPosition);
                broadcastExclusivelyTo(client->id, event);
            }
        }
    }

    // ...
};
```

2.4.4 Creating a Synchronized Component

Actors composiste components, and synchronized actors allow a subset of these components to be synchronized across the network. The components must handle the synchronization input and output themselves through specifically designed interfaces.

The pseudo code for the class `CarDrivingAiComponent` in Figure 14 shows how to synchronize a component. The class is an AI agent driving a car. The agent is allowed to drive forwards and to brake. Depending on whether the agent is accelerating or braking, different data is sent to the remote copy. Components are free to send as much data as they like.

Advanced interpolation and lag compensation should be done in the component methods `serialize()` and `deSerialize()`. The network system's estimation of the packet's age can be retrieved from the packet object, and used for estimation.

Figure 14: Synchronizing a component's state

```cpp
enum CarState { DRIVING, BRAKING };

class CarDrivingAiComponent: public nox::Component
{
    CarState state;
    float gasPedalPressure;
    float brakePedalPressure;
    int gear;

public:
    // serialize() gets called on synchronized components attached
    // to locally owned actors.
    void serialize(Packet& packet)
    {
        packet << state;
        if (state == DRIVING)
        {
            packet << gasPedalPressure;
            packet << gear;
        }
        else
        {
            packet << brakePedalPressure;
        }
    }

    // deSerialize gets called on synchronized components attached
    // to remotely owned actors.
    void deSerialize(const Packet packet)
    {
        packet >> state;
        if (state == DRIVING)
        {
            packet >> gasPedalPressure;
            packet >> gear;
            brakePedalPressure = 0.0;
        }
        else
        {
            packet >> brakePedalPressure;
            gasPedalPressure = 0.0;
        }
    }
};
```

2.4.5 Sending Large Files

A relatively common use-case in multiplayer games it to send large binary files. The files may be the profile image of players or user-made levels. The Nox networking system does support sending large binary files over the event channel. The component channel may also be used, but the content only needs to be sent once, rendering the event channel more suitable.

Figure 15: Sending large binary files

```cpp
void sendLevel(std::string levelFile)
{
    std::vector<unsigned char> binaryContent = readBinaryFormat(levelFile);

    BlobTransmitEvent blobEvent(binaryContent);
    blobEvent.setDescriptor("custom level");
    blobEvent.setFileName(levelFile);

    NetworkOutEvent netEvent(blobEvent);
    raiseEvent(netEvent);
}
```

3 Stencil Shadows

3.1 Stencil Shadow Requirements

The existing algorithm for generating shadow shapes was based on ray-casts in the physics engine. The shadows were generated using as many threads as the running computer have logical CPU cores. This was the only way the implementation would provide acceptable performance, as ray-casts can be expensive in terms of processing. We suggested improving the system to Suttung Digital, and they welcomed the suggestion.

3.2 Light and Shadow in 2D

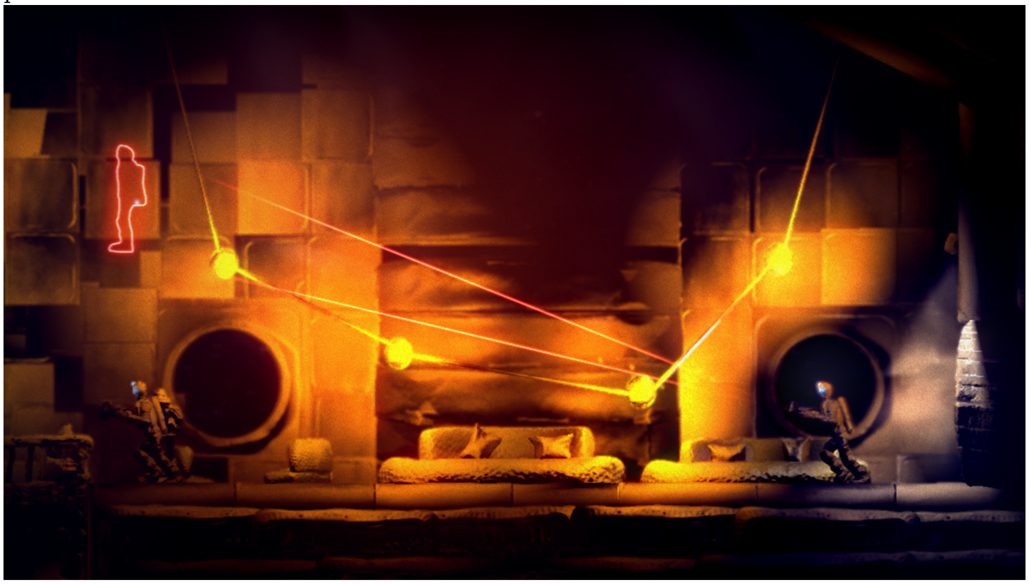
Light and shadow are features with the potential to raise the aestethical presentation of a game significantly. Light and shadows are a definite *juicy* feature. Juicy features, as described by Martin Jonasson at his GDC 2013 talk, *"Juice it or Lose it"* [4]:

> “Juicy things are things that wobble, squirt, bounce around, and make little cute noises; it’s sort of a catch-all phrase for things that make a game more satisfying to interact with.“

Lights and shadow can increase the interactivity of a game by making the world feel more immersive, dynamic and *real*. Jordan Thomas, who worked as a game developer on *"Bioshock"* and as Creative Director on *"Bioshock 2"* states that lighting in 3D scenes “allows you to shout in subtext“. Thomas also states that in addition to providing an illusion of depth on a two dimensional screen, lights *(and sound)* are “hidden paths into our hearts“ [5].

There is no reason for lighting to have any less potential in 2D games. Games like "The Swapper"16 uses lighting to create a highly dynamic and lifelike environments, entirely in two dimensions.

Figure 16: Screenshot from *"The Swapper"*. The game utilizes lights, shadows, and normal maps to achieve its unique look. Image taken from Facepalm Games's Press Kit for The Swapper [2]



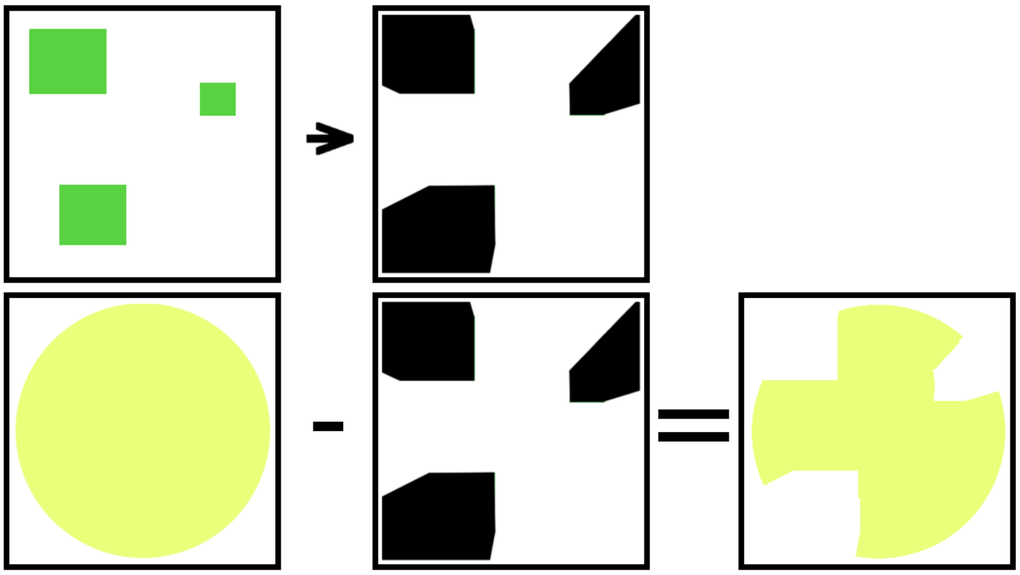
The hardest and most important task of the shadow generating algorithm is to find a way to *exclude* the beshadowed areas of the scene. While the details differ wildly, the general high-level steps for achieving this in 2D is essentially the same across all implementations.

1. Create polygons containing the shadow filled regions.
2. Create a circular or cone shaped polygon, the *light polygon*.
3. Subtract the shadow regions from the light polygon.

These steps are clearly visualized in Figure 17. There are multiple ways of achieving this goal. Two of the most common methods, stencil based shadow rendering and ray-cast based shadow generation are discussed in Sections 3.4 and 3.3.

[2] `http://facepalmgames.com/press/index.php`

Figure 17: The final light geometry is created by subtracting the shadow polygon from the circular light shape.



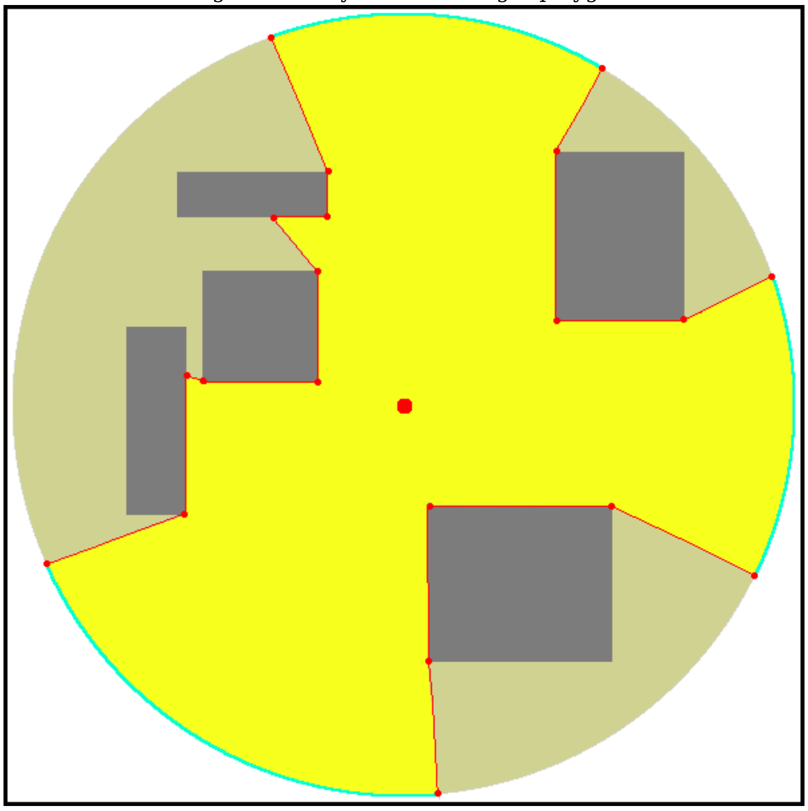
3.3 Existing Ray-cast Based Implementation

The ray-cast based shadow generation algorithm originally used in Nox was heavily CPU-bound, and relied largely on expensive ray-casts in the world.

The algorithm creates a fully defined, *ready-to-render* polygon for each light, where the shadowed areas are exluded from the light geometry. As can be seen in Figure 18, *redundant* vertices and edges with no direct line of sight to the light's center are ignored from the final shadow area.

The created light polygon is absolutely positioned, which is a necessity in order to also "*include*" the shadow regions in the polygon. This means that even in a scene with no shadow casters, the entire light geometry must be recreated whenever the light's transform changes.

Figure 18: Ray-cast based light polygon



The red points in Figure 18 indicate significant vertices in shadow-casting shapes. The red and blue lines indicate the absolute shape of the light polygon. The grey areas show where the shadows are cast.

Figure 19: Ray-cast shadow generation pseudo code

```
// This function is run for every light across as many threads
// as are available to the system.
void generateShadowData(Light* light)
{
    List shapes = getShapesInAABB(light->getAABB());
    List shadowVertices;
    Phyisics *physics = getPhysics();

    for (Shape *shape : shapes)
    {
        for (Vertex *vert : shape->vertices)
        {
            if (physics->lightHitVertex(light, vertex))
            {
                shadowVertices.add(vertex);
            }
        }
    }

    // This recreates the entire geometry of the light
    generateLightShape(light, shadowVertices);
}
```

The algorithm works by querying the world for all shapes within the light's AABB, and casting *rays* to the vertices of the shapes to determine which vertices will cast shadow. The pseudo-code in Figure 19 is a major oversimplification of the actual algorithm, but it shows the processing required to calculate the shadow geometry. The generated data is ready to be rendered without any advanced techniques, as can be seen from the rendering pseudo code in Figure 20.

Figure 20: Pseudo-code illustrating the rendering process of ray-cast generated shadow data.

```
void renderLights(List lights)
{
    clearScreen(getClearColor());

    VertexData vertices;
    for (Light *light : lights)
    {
        vertices.addLight(light);
    }

    drawVertices(vertices);
}
```

3.4 Stencil Shadows in 2D

The stencil based implementation works by *always* assuming that a shadow-casting shape with at least one vertex within the AABB of the light will cast shadow from all of it's edges. The algorithm is more intensive on the GPU than the existing ray-cast based algorithm, but is in turn less CPU intensive.

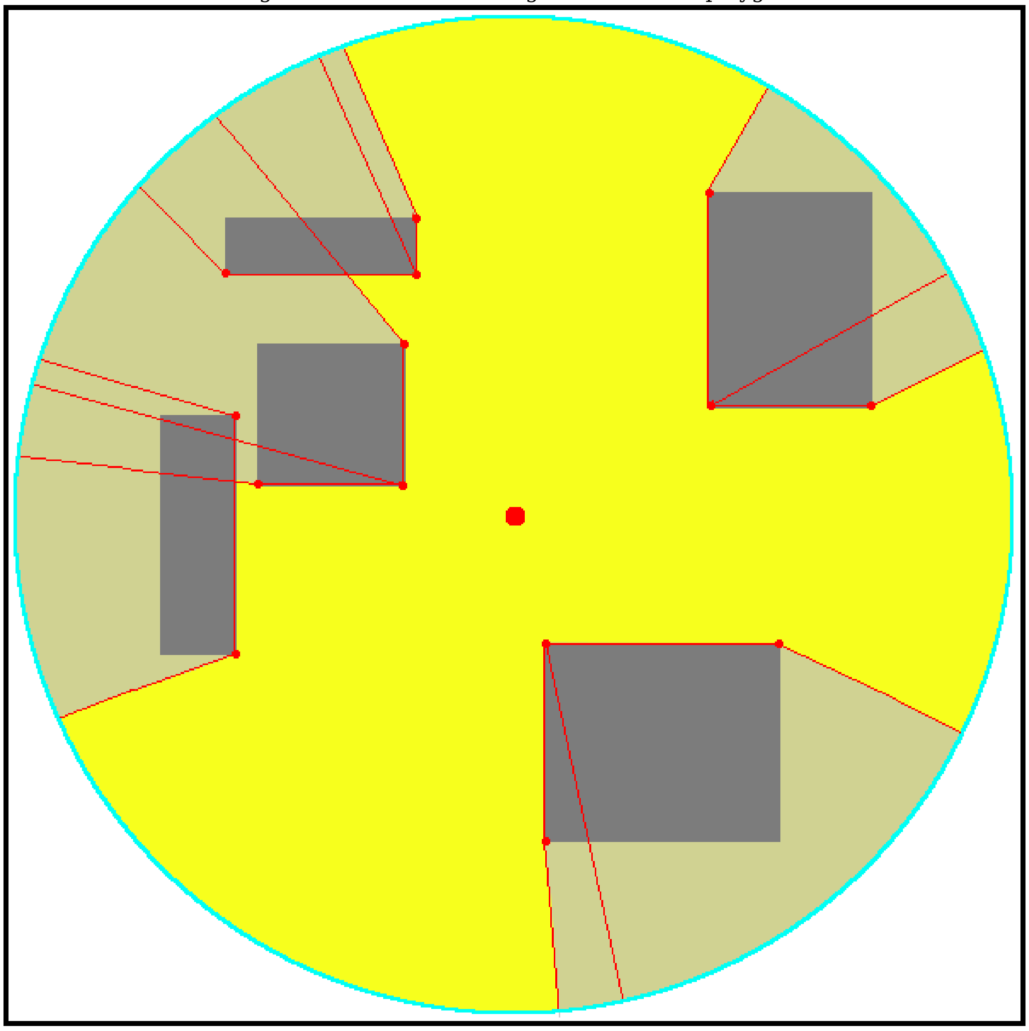
Figure 21: Stencil based shadow generation pseudo code

```
// This function is run in succession for every light, on a single thread
void generateShadowData(Light* light)
{
    List shapes = getShapesInAABB(light->getAABB());
    List shadowVertices;

    for (Shape *shape : shapes)
    {
        light->addVerticesFromShape(shape);
    }
}
```

With this approach, the shadow geometry and light geometry are disjunct and treated as two separate entities. The light geometry is created as a unit circle and is scaled and moved to the desired location. The shadow geometry is excluded by utilizing the stencil buffer during the rendering process. Only dramatic changes which changes the attributes of the light regardless of transform requires the light geometry to be recalculated.

Figure 22: Stencil based light and shadow polygon



The red points of Figure 3.4 indicate the vertices of shadow casting shapes which cast shadow, and the red lines show the connection between them. Each shadow casting edge is projected far beyond the radius of the light and drawn as a rectangle. The projected edges are not shown in the figure. The darkened areas is where the shadows will be cast, and will act as the *stencil* for the light geometry.

3.4.1 Rendering

The rendering of stencil based lights requires two passes per light. As each light *pollutes* the stencil buffer with it's shadow data, the buffer must be cleared after drawing each light to prevent lights from interfering with each other in an unintended fashion. Lights which does **not** cast shadows, or lights whose shadows does not overlap any other light's shadow or light geometry can in theory be drawn in a single pass while still guaranteeing no interference, but the current implementation does not do this.

Figure 23: Stencil based light & shadow rendering pseudo code

```
void renderLights(List lights)
{
    for (Light *light : lights)
    {
        drawToStencilBuffer(light->getShadowGeometry());
        drawToColorBuffer(light->getLightGeometry());
        clearStencilBuffer();
    }
}
```

In comparison with the ray-cast generated shadows seen in Figure 20, the rendering is far more complex. The stencil shadow rendering requires 2N passes, where the ray-cast based shadow rendering requires only 1.

3.5 Performance Analysis

The motivation for changing Nox's light and shadow system was to increase performance, especially on lower-end systems. In order to assess the whether or not this was the case, we needed to test the performance of both implementations on a set of computers with a broad spectrum in processing power. We had three machines available between us, and Magnus Bjerke Vik from Suttung Digital graciously offered to run the tests on his machines as well.

Table 2: The test environments used throughout the entire document.

Name	OS	CPU	Graphics Card	RAM	Resolution
Environment 1	Arch Linux	Intel i7 4790K, 4 cores, 8 threads, 4.0GHz	Nvidia GeForce 9800 GTX+ (Nvidia 340.76 driver)	8GB	1920x1080
Environment 2	Ubuntu 14.04	Intel i5, 2 cores, 4 threads, 2.3GHz	Intel Graphics HD 3000 (Mesa 10.6.0 driver)	6GB	1280x800
Environment 3	Ubuntu 14.10	AMD FX-6360, 6 cores, 4.2GHz	Nvidia GeForce GTX 780 Ti @1202MHz base	16GB	1920x1080
Environment 4	Antergos	AMD E2-6110, 4 cores, 2x1.5GHz + 2x1.0GHz	AMD Radeon R2	8GB	1600x900
Environment 5	Arch Linux	Intel i5 2500k 3.3GHz, 4 cores	Nvdia GeForce GTX 680 (Nvidia 349.16 driver)	8GB	1920x1080
Environment 6	Ubuntu 15.04	Intel i5 M 450, 2.4GHz 2 cores, 4 threads	ATI Mobility Radeon HD 5650 (Mesa 10.5.2 driver)	8GB	1366x768

The performance test ramps up from 1 moving light to a final total of 100 moving lights. There are always 400 shadow casters in the scene. The tests have been run on all the environments listed in Table 2. For each *configuration* of the test [3], performance data was gathered over the course of 1000 frames. The configurations were run sequentially as unique processes with no other heavy task executed simultaneously. Figure 24 illustrates the light performance test application. For the test cases, all 100 lights are updated and rendered every frame[4].

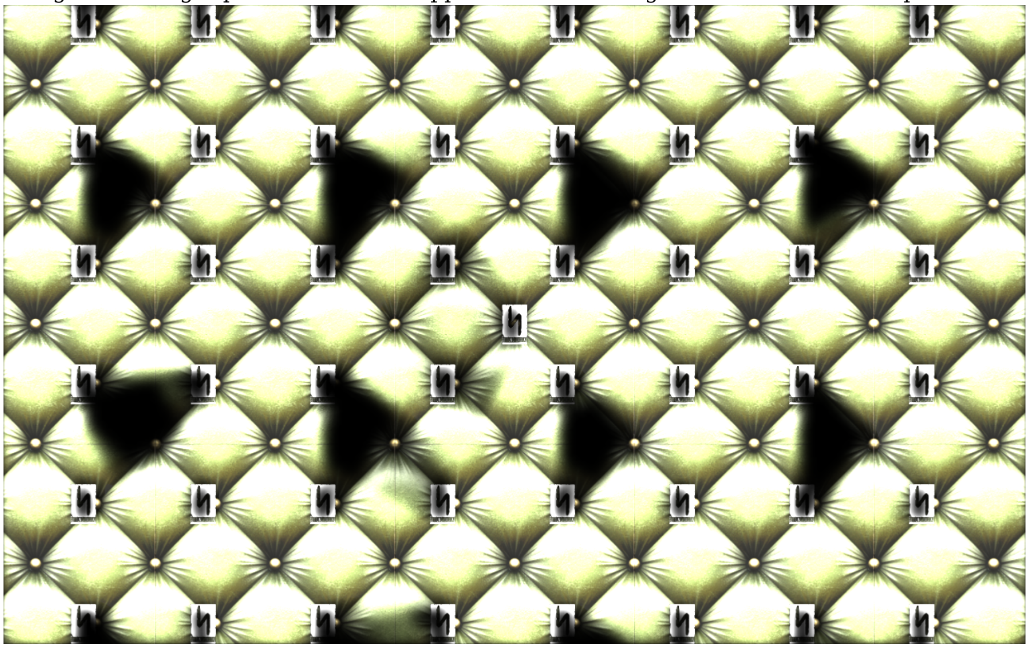
The ray-cast based shadow method recreates the light geometry when *modifications to the light transform* or *modifications to the transform of any shadow casting*

[3]The number of lights visible in the scene is considered the *configuration* of the test.
[4]Lights which are off-screen are still culled by OpenGL.

fixture within the light's AABB occurs. The stencil based shadow method recreates the shadow geometry every frame.

The tests run at the system's native resolution, as opposed to running in a resolution all the systems supports. While this weakens the integrity of the data as the tests were run under different conditions, the data still offers a good overview of what performance can be expected from machines of different specifications.

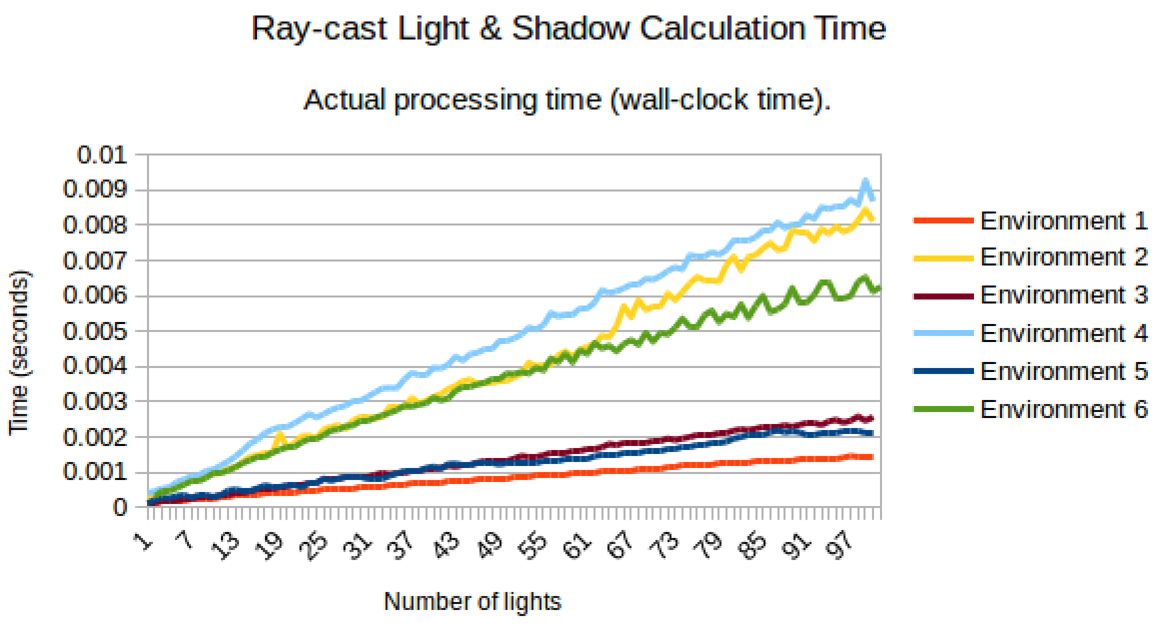
Figure 24: Light performance test application. All the lights move in a circular pattern.

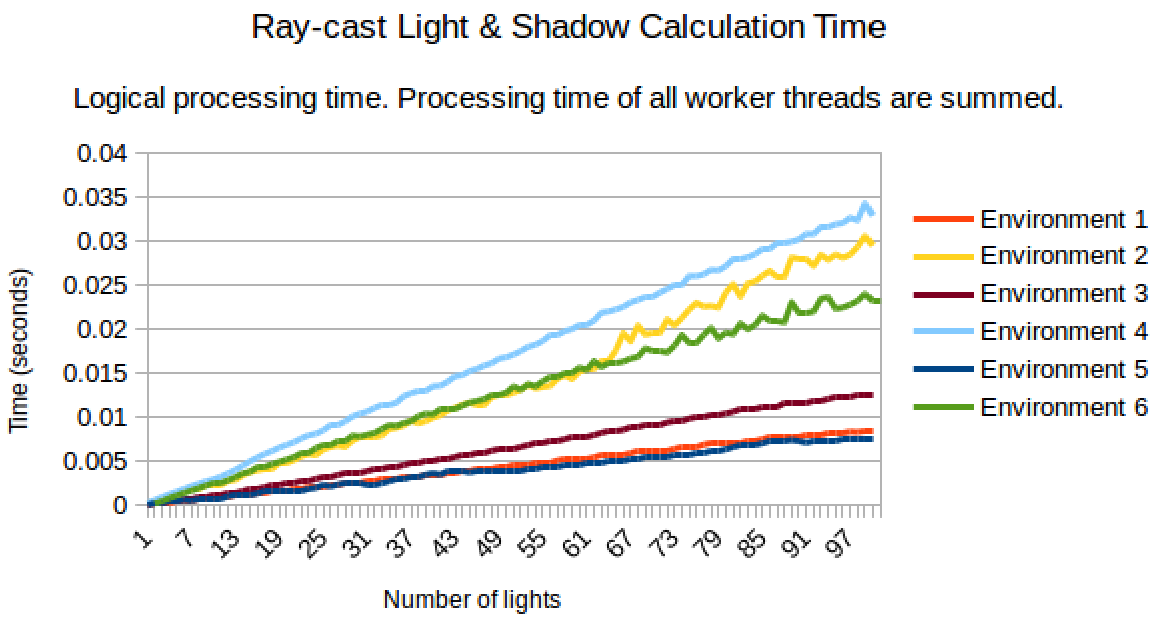


3.5.1 Shadow Generation Performance

The *"shadow calculation time"* refers to the time it takes for the system to generate the shadow (and light, for the ray-cast implementation) for all the lights in the scene.

Figure 25: Ray-cast light & shadow generation time

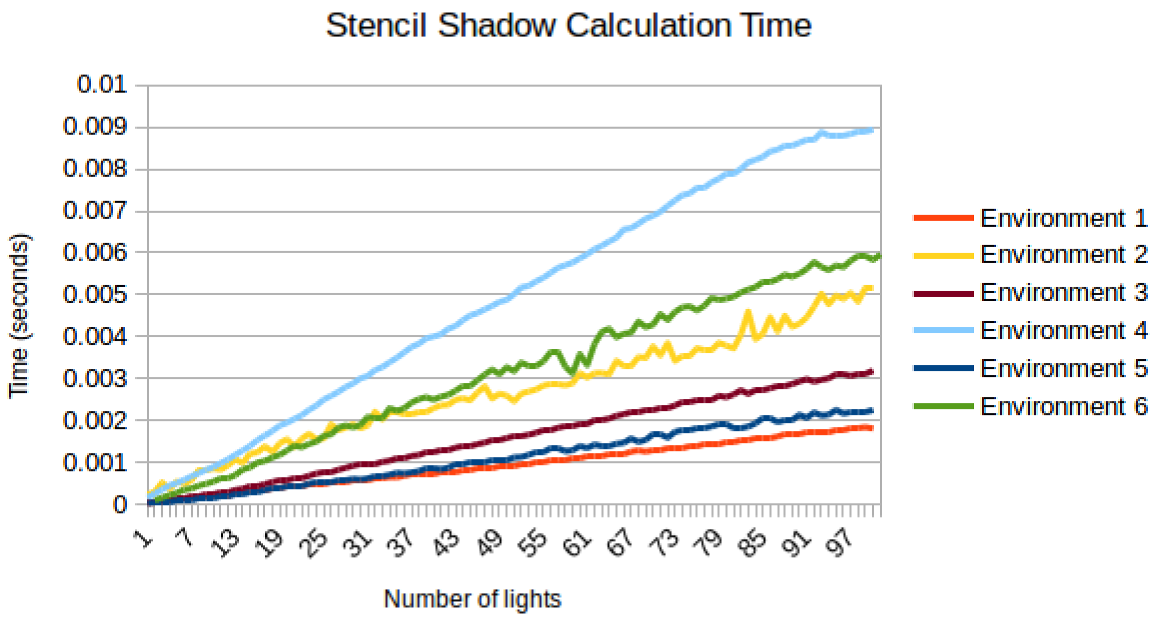




The ray-cast based shadow generation implementation is as previously discussed in Section 3.3 heavy on the CPU, and easy on the GPU. In order to achieve acceptable performance, the algorithm is executed on as many threads as are available to the system. This can be seen in Figure 25, where the *wall-clock* processing times are significantly shorter than the *logical* processing times.

As can be seen in Figure 25, all tested environments start at *close to zero* processing times for 1 light. The performance increases linearly and proportional to the processing power available to the specific system. The slower environments ends up spending close to 9 milliseconds (wall-clock) for 100 lights, whereas the faster environments lie closer to 2.

Figure 26: Stencil based shadow generation times



When compared to the logical processing time of the ray-cast generation, the stencil calculation is significantly faster, as can be seen in Figure 26. However, when comparing against the wall-clock time of the ray-cast generation, the difference in performance is trivial. The stencil shadow generation can in theory be performed over multiple threads as well, as each light are entirely independent of one another. This is a potential point of future improvement.

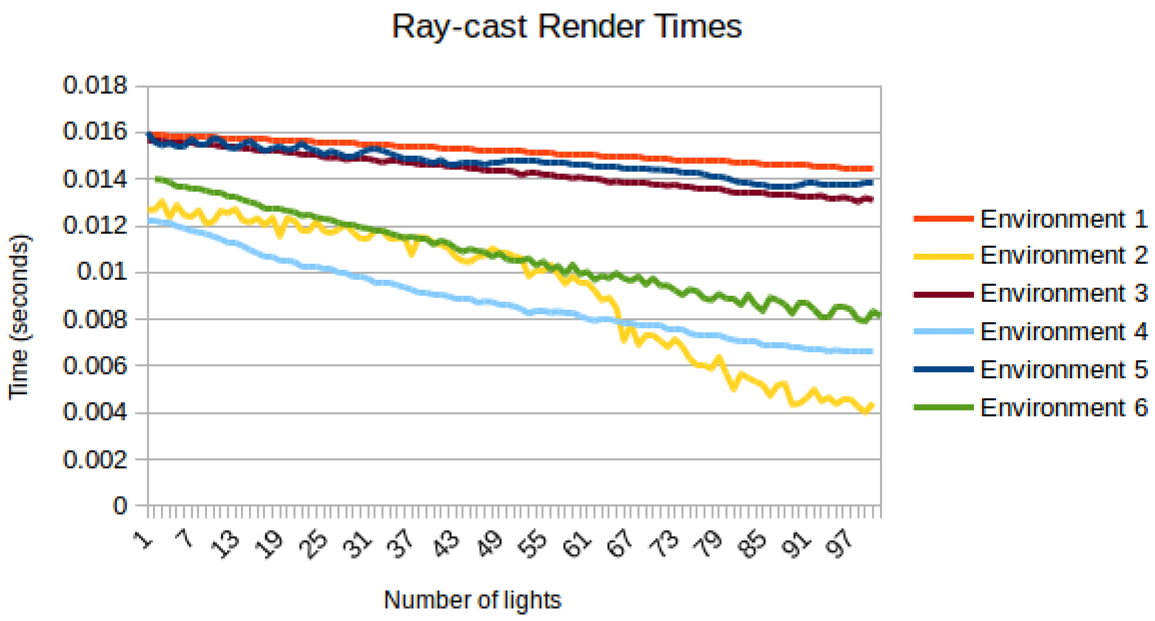
3.5.2 Rendering Times

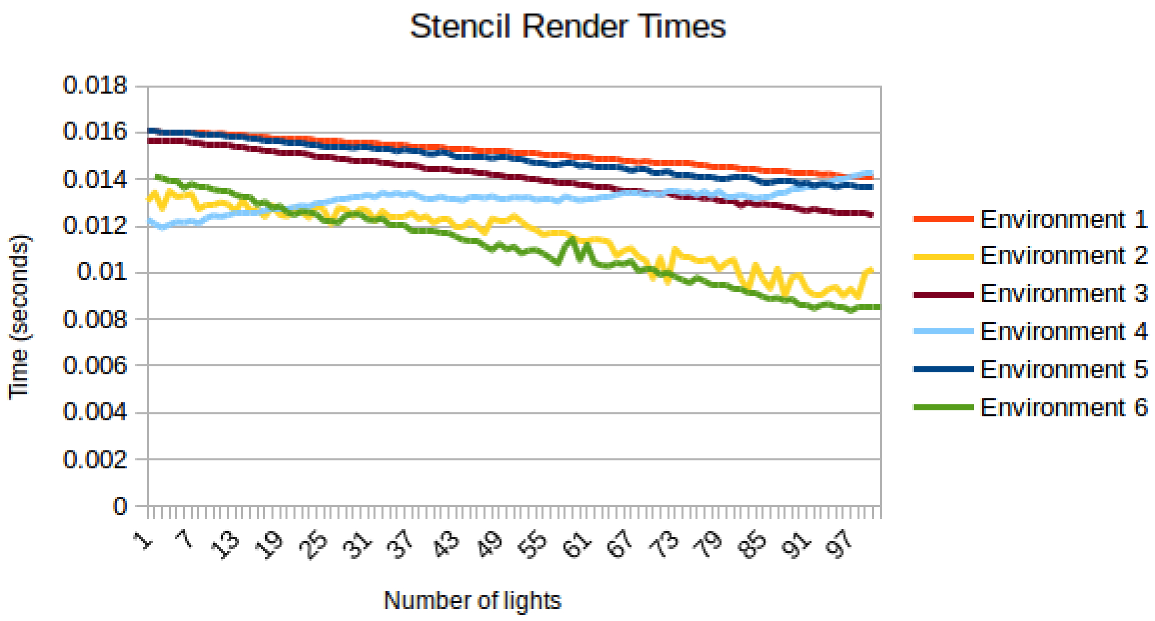
The *"rendering time"* of a test is the time the method `OpenGLRenderer::onRender()` takes to complete. This method handles all things related to rendering, except the copying of the data that will be drawn. The copying of data happens asynchronously in background threads.

The ray-cast generated shadows can as previously discussed in Section 3.3 be drawn in a single pass due to the nature of the data. The largest determinator of performance in this regard is therefore the size of the data that is drawn.

The stencil based shadow rendering requires as discussed in Section 3.4.1 2 passes per light. There is less data to be copied to the graphics card when drawing it, as only the shadow data needs to be copied. The light geometry remained unchanged throughout the entirety of the test. The largest determinator of performance with regard to the stencil generated shadows is therefore likely to be a combination of the number of lights and the number of shadows cast in total.

Figure 27: Ray-cast and stencil based shadow rendering times





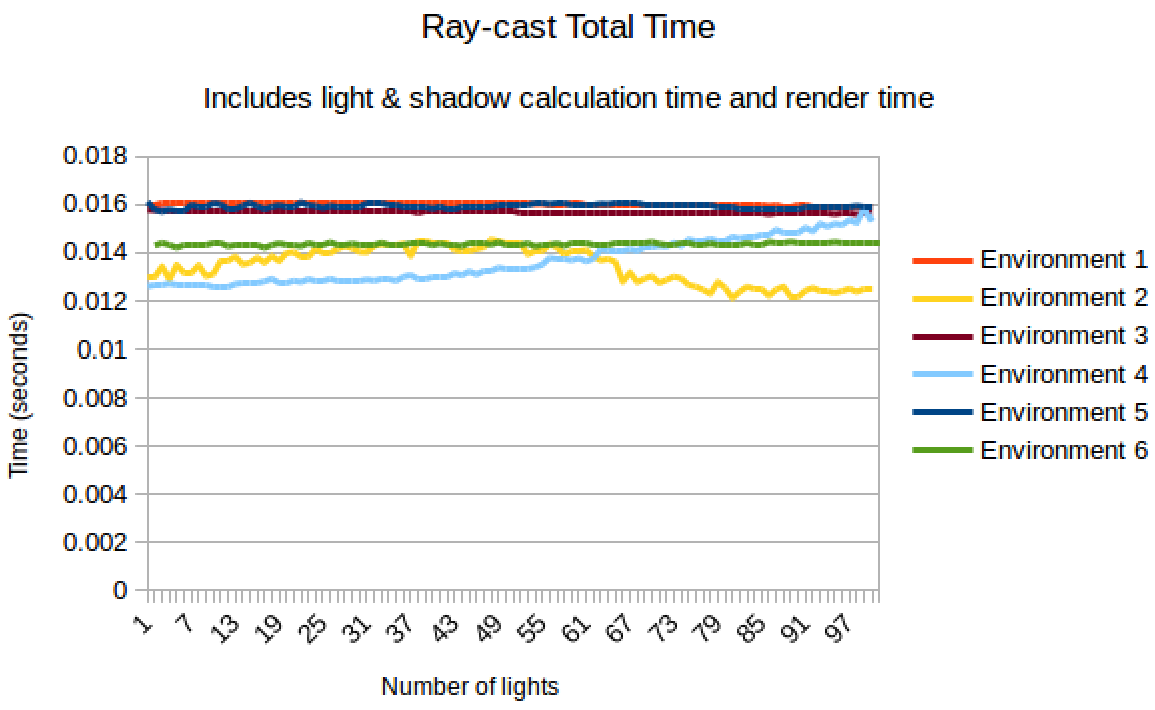
As Figure 27 shows, the rendering time for both implementations starts relatively high, and **decreases** as more lights are added to the scene. The only exception to this is Environment 4, which slowly increases with the stencil implementation. This data surprised us, and we were unable to determine the cause of the behavior. We saw similar trends in a previous round of testing, where each test confguration were only run for 200 frames. We suspected that the graphics cards (and CPUs) might "ramp up" as the system realizes it is under heavy load, causing the performance to increase over time. This was the reason we decided to run the tests for 1000 frames, in the hope that any anomalies caused by the system speeding up over time would be removed. We believe this to be an unlikely source of the (unintended) performance gain, as the tests take between 30 and 90 minutes to complete, and that the automated test script uses between 1 and 3 seconds between configurations to prepare the next test.

Performance anomalies aside, the rendering times of the high-end machines *(environments 1, 3, and 5)* lie between 14 and 16 milliseconds for 100 lights. We consider none of these rendering times to be within the reasonable time frame, as little time would be allowed for other tasks in the engine to run, while still achieving 60 frames per second.

3.5.3 Total System Impact

The final determinator of the performance of the two implementations is to combine the time taken to generate the shadow (and light) data, and how long it takes to draw it.

Figure 28: Total system impact of light implementations



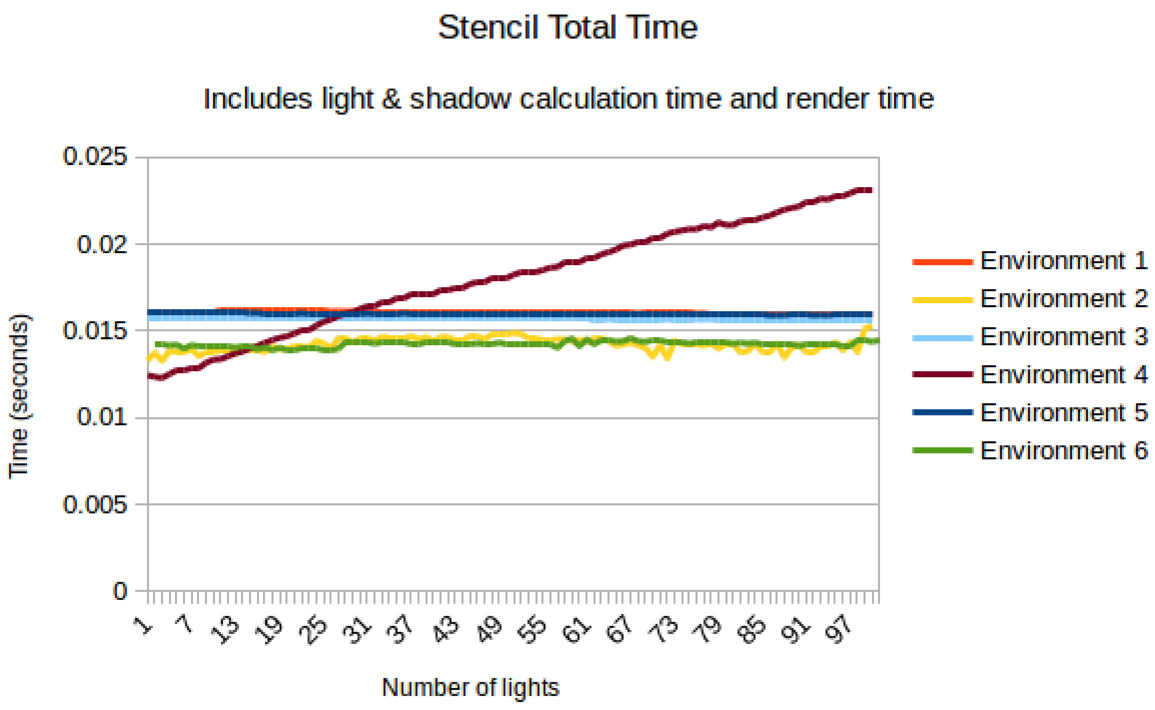


Figure 28 shows the total system impact of the two light & shadow implementations. It’s worth to notice that because the rendering times decreases as lights are added, the shadow calculation time increases near equally, resulting in *very* stable performance on 5 of the 6 environments, regardless of the number of lights. The performance difference between the two systems also shows that the performance difference is insignificant.

3.5.4 Performance Conlcusion

The stencil light implementation set out to increase the performance of the light and shadow generation in Nox, but as can be seen from Figure 28, the difference is at best marginal. However, that does not mean the endeavour was fruitless. With the ray-cast implementation, the engine creates a "burst" of work on all the system’s cores once every frame. If an existing or new feature of Nox were to be given it’s own thread, it is likely that it would behave more smoothly with the stencil lights as opposed to the ray-cast based lights. The complexity of all the shadow related methods was also significantly reduced, as shown in Table 3.

Table 3: Light implementation complexity and SLOC comparison. The complexity is measured in cyclomatic complexity. [6] Only significant methods primarily used in generating or rendering lights and shadows are included. *RC* is shorthand for Ray Cast. *S* is shorthand for Stencil. *CC* is shorthand for Cyclomatic Complexity. *SLOC* is shorthand for Source Lines Of Code.

Method Name	RC SLOC	RC CC	S SLOC	S CC
Box2DSimulation::calculateVisibility	426	41	-	-
OpenGlRenderer::lightRenderingDataPrepare	85	15	111	15
OpenGlRenderer::onLightMapRender	44	9	159	14
Box2DVisibilityMapper::getInvisibilityRegions	-	-	92	7
Box2DVisibilityMapper::getShapeVertices*	-	-	25	2
Box2DSimulation::calculateLightVisibility	-	-	27	4
Box2DSimulation::calculateLightGeometry	-	-	56	5

4 Normal and Specular Maps

4.1 Requirements

We proposed to implement normal and specular mapping in Nox to Suttung Digital, an improvement they gladly welcomed. The implementation is required to utilize normal and specular maps in combination with the default "diffuse" textures.

4.2 Normal and Specular Maps

Normal mapping is a way of achieving depth and *texture* on a plane, where none previously existed. The most common use-case for normal maps exists in the realm of three-dimensional graphics, where proper use or normal maps can **greatly** reduce the number of required polygons to achieve *(largely)* the same visual effect.

A normal map works by showing the normals of the 3D model against the plane it has been flattened to. The r, g, and b values to show the as x, y, and z values as the normal vector. This allows us to calculate the reflection, using the angle of the light. A specular map works very similarly. It only gives one value, as it is a single channel image. This value is used to represent the reflectiveness of the image. Given a 3-dimensional color vector (RGB format) $\vec{c}$, the color can be transformed to a normal vector $\vec{n}$ via the following formula:

$$\vec{n} = 2\vec{c} - [1, 1, 1]$$

When rendering a texture with normal and specular mapping enabled, the intensity is added to the texture's red, blue and green chanels, while the alpha channel remains untouched by the calculation.

$$\text{Color} = [\text{Tex}_r + \text{Intensity}, \text{Tex}_g + \text{Intensity}, \text{Tex}_b + \text{Intensity}, \text{Tex}_a]$$

The intensity is calculated by taking three factors into account; the angle at which the light hits the object, the attenuation *(falloff)* of the light, and the specularity of the object. $\vec{D}$ is the distance vector from the surface of the object to the light. $\vec{U}$ is the normal of the surface. Tex is the diffuse texture. Spec is the specular texture.

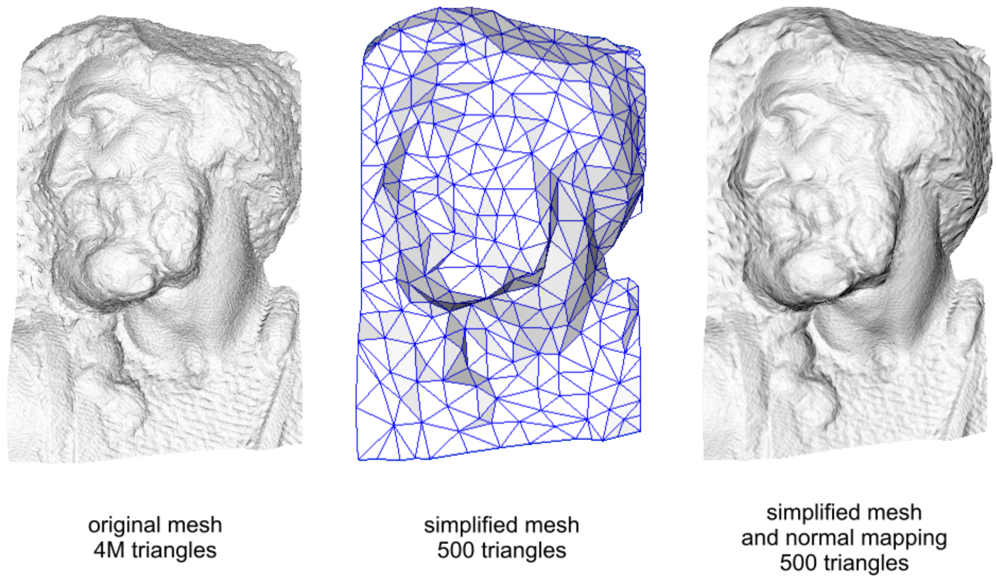
$$\text{Intensity} = \text{Tex}_A + \sum_{i=1}^{n} \max(0, \vec{D} \cdot \vec{U}) \times \text{attenuation}(|\vec{D}|) \times \frac{\text{Spec}_r + \text{Spec}_g + \text{Spec}_b}{3}$$

The attenuation function is the *constant-linear-quadratic* formula for attenuation[1].

$$\text{attenuation}(d) = \frac{1}{c + ld + qdd}$$

[1]https://goo.gl/jkoygh

Figure 29: Normal map example in 3D by Paolo Cigoni[2]



In Figure 29, the original mesh uses 4,000,000 triangles to define the bumps and dents in the face. The rightmost mesh consists of a mere 500 triangles *(0.0125% of the original)* and achieves the same general look through the use of a normal map.

Figure 30: Normal map in 2D[3]

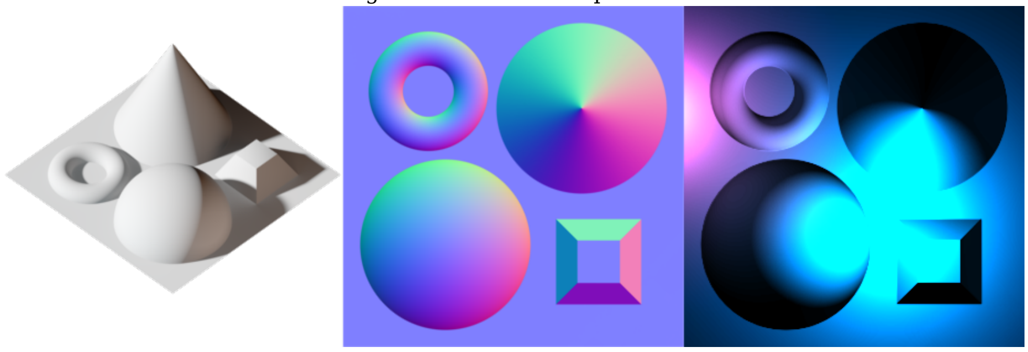
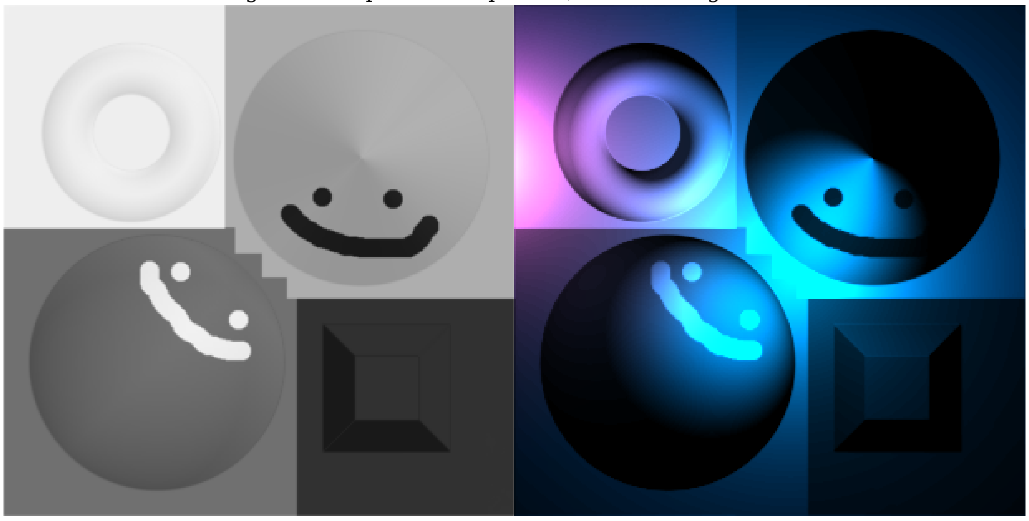


Figure 30 shows a 3D model converted to a normal map, and then rendered to a 2D surface. The first cell shows a 3D model, the second cell shows a normal map generated from said 3D model. The third cell shows the effect of rendering with said normal map. The rendering was done in the Nox engine, using a white background, one purple and one blue light.

[2] "Normal map example" by Paolo Cignoni. Licensed under CC BY-SA 1.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Normal_map_example.png#/media/File:Normal_map_example.png

[3] Modified version of "Normal map example with scene and result" by Julian Herzog. Licensed under CC BY 4.0 via Wikimedia Commons http://upload.wikimedia.org/wikipedia/commons/2/2e/Normal_map_example_with_scene_and_result.png

Figure 31: Specular map in 2D, based on Figure 30.



The second cell in Figure 31 shows the usage of both specular and normal maps. It uses the same setup as Figure 30, with the addition of the black and white image in cell one as a specular map.

4.3 Implementation

Implementing normal and specular maps in Nox required some changes to the existing light system. The lights themselves become naturally more complex when normal mapping is taken into account. Lights now need attenuation paramters and a height. The height is required to simulate the light in a 3D environment, which enhances the effect of the normal map procedure. Lights previously had a falloff parameter, but this was a constant factor. The lights were changed to instead use the *constant-linear-quadratic* formula for attenuation.

After the lights were given additional attributes, the remaining task was to extend the rendering pipeline of Nox to load the maps, and use them in the relevant shaders. The usage of normal and specular maps was implemented in the shader `spriteLightLuminance`, which are responsible for drawing sprites affected by light. The final result can be seen in Figures 30 and 31.

4.4 API Usage

The API for adding the normal and specular maps is very simple. By adding the objects "normalMap" and "specularMap" with values for what files contains them to the object in "textureAtlases", they are automatically used. This is done in line 10 and 11 of Figure 32.

Figure 32: Normal and specular map definition

```
{
  "texelsPerMeter": 100,
  "textureAtlases":
  [
    {
    "name": "graphics/textureAtlas",
    "imageExtension": "png",
    "dataExtension": "json",
    "mipMap": true,
    "normalMap":"graphics/normalAtlas.png",
    "specularMap":"graphics/specularAtlas.png"
    }
  ]
}
```

There are no safety measures in place to ensure that the normal and specular maps are of the same resolution or aspect ratio as the primary texture. Using normal and specular maps of lower resolution than the primary texture may reduce the overall visual quality of the scene, but may increase performance on slower computers.

5 Particle System

5.1 Requirements

Suttung Digital asked us to implement a particle system in the engine. The particle system was wanted for visual effects. It needed to look good, but it did not need to be so consistent that it could be used as a consistent gameplay element. As the visual aspect is the main reason for adding the particle system to Nox, the rendering functions have to be easy to replace. Different games may want to use different particle effects, and having an expandable system is necessary to allow users to create the exact effects needed for the aesthetics of their game.

With an engine already using Box2D, a particle system based around Box2D would be the best solution. Otherwise we would have to replace the entire physics system. Performance was an important factor, as having your engine run on as many machines as possible is crucial. We did not want our particle system to exclude machines with weak processors, as this may exclude a significant portion of the potential player base.

It has to support Windows, Mac OS X, and Linux. Even tough we are not supporting it in this project, support for Android, iOS or any other platforms is a big plus.

5.2 Selecting a particle system

When selecting a solution for the particle system in Nox, Asbjørn Sporaland quickly suggested using LiquidFun[1], as he had seen some demos of it, and knew it was a fork of Box2D. We looked for any alternative particle solutions in case there were any better options, but LiquidFun seemed to be the best alternative. We also conscidered writing our own implementation of a particle system, without collision detection, but using the gravity forces in the engine.

5.2.1 Writing our own particle system

Writing our own particle system based on Box2D would take time. However it would be made to perfectly fit our engine, and we would know everything about it. It would do what we wanted it to do, and *only* what we wanted it to do. Having it only do what we want it do do, might give us slightly better performance, as it does not do anything unnecesary. But having it only do what we wanted it to, would mean more work if we were to add features, that already exist in LiquidFun, later on. We had not planned on implementing collisions in our own particle system.

5.2.2 Using LiquidFun

The particle data you get out of LiquidFun is easy to use with OpenGL. This is not an advantage, as we could make sure this would be true for our self made particle system. But at least it it works the way it should. LiquidFun is a fork of Box2D, and

[1] http://google.github.io/liquidfun/

is therefore easy to put into the engine, by simply replacing the submodule in the git repository. This would allow us to focus on implementing the surrounding particle system.

LiquidFun being a fork, there will be some delay in getting upstream updates from Box2D. However, Box2D has not been updated since November 2013[2] (as of 13.05.15), so updates are few an far between. Box2D also currently works exactly as we need it to do, so no updates are required.

The major disadvantage is that Nox will now be further locked to LiquidFun than it was to Box2D previously. Nox has generally had as many *replaceable* modules as possible, and adding more interaction with the physics engine makes it more difficult to replace it. LiquidFun supports the three desktop platforms already supported by Nox, in addition to Android and iOS.

5.2.3 Conclusion

We went with LiquidFun, as it is more feature rich than our own solution would have been, it is proven as a solution for adding particles to Box2D, it is easy to work with, and it should be less work than writing our own system.

5.3 Particle systems in 2D

Particles in a 2D game engine work similarly to the rest of the physics system. You could in fact just use tiny physics objects as particles, but this can be slow.

As a particle system can often have hundreds or thousands of particles in it at the same time, it is important that it is highly optimized.

[2]https://code.google.com/p/box2d/downloads/list

Figure 33: Particle accuracy

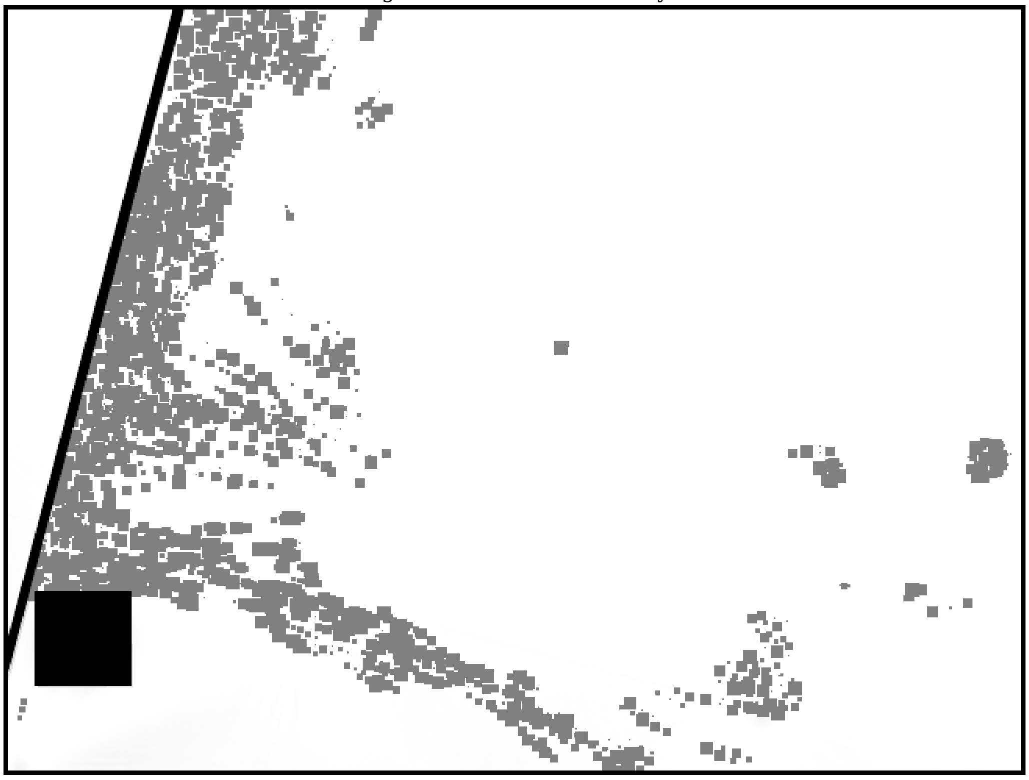


Figure 33 was made in the Nox engine. There might be a couple (or a lot) of particles in there that are not where they "should" be in a perfect simulation, but it does not matter, because the general mass of particles are where they should be, and everything looks nice.

A good solution for optimizing particles is to only make them points, instead of using physical bodies. Calculating a collision within a physical body is simpler with a point than another body. Box2D checks if particles are within a certain distance of eachother in both dimensions, and if so it will (depending on particle flags) push them i opposite directions, or assert a pressure on them.

5.4 The Design

The particle system is outlined in Figure 34 and the this section of the document.

The component side of the particle system is responsible for creating new partcles, and sending them to Box2DSimulation. Box2DSimulation contains LiquidFun, and LiquidFun updates the particles every cycle. When the particles are updated, Box2DSimulation sends them to the particle renderer, wich updates its references to them. When a new frame is rendered, the particle renderer renders the particles using its latest references to the particles.

The component side of the particle system is based on actors having a particle component, containing one or more particle emitters, each containing one or more particle definitions (class Particle from Figure 34).

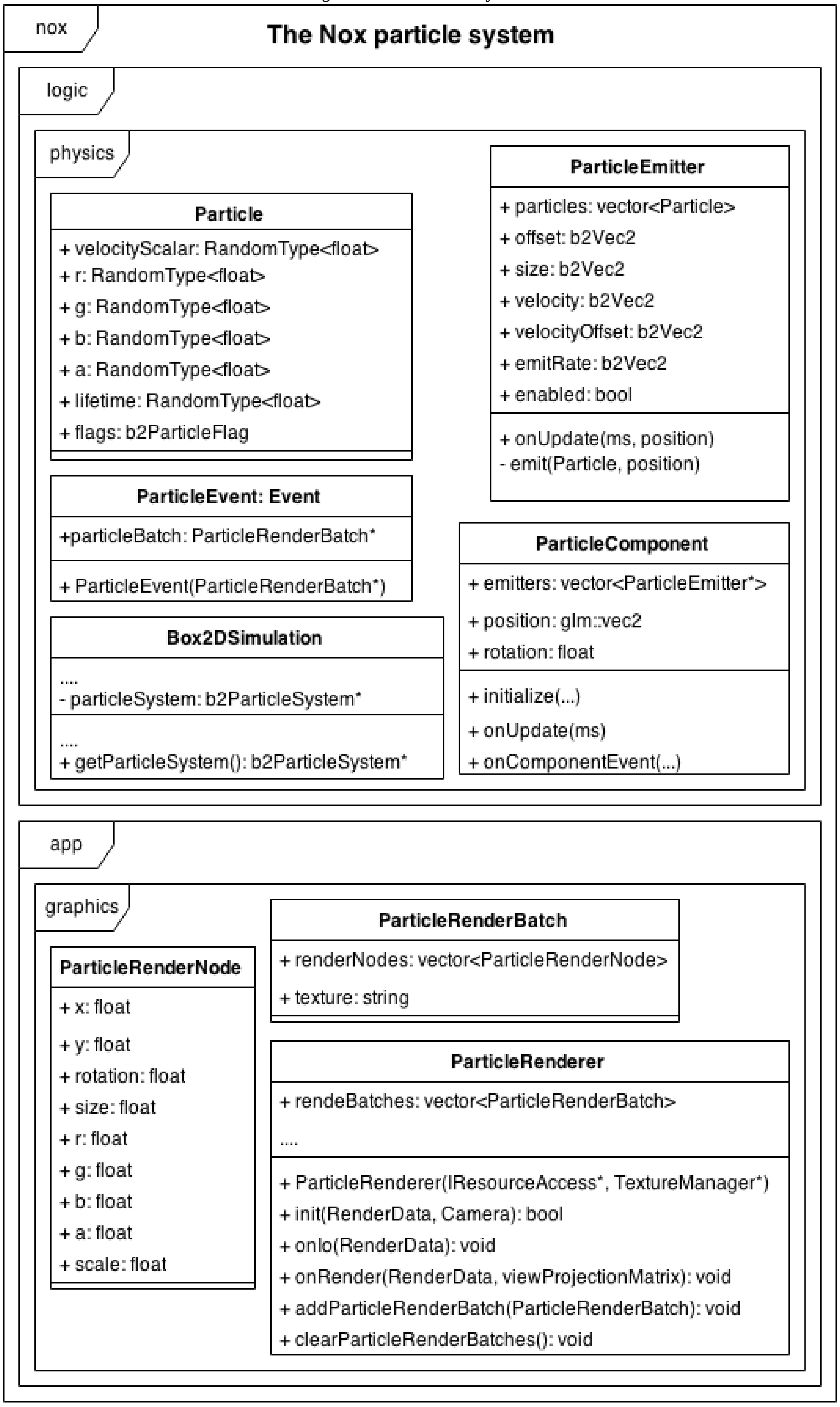
The particle component holds the particle emitters, and is responsible for positioning and updating them. Whenever the actor it is attached to changes its position, it sends out an event. This is picked up in the particle component, and it updates its own copy of the position. When the system is updated, it calls for an update in each of its particle emitters.

The particle emitters get the time since last update, and the global position of the actor from the particle component when it updates them. It checks the time against the emit rate of particles, and decides if and how many particles to emit. For each particle to be emitted, it creates a b2ParticleDef, and sends it to the b2ParticleSystem in the Box2DSimulation.

The emitters do not update the emitted particles. Those are dealt with in LiquidFun. The emitters emit particles on update when needed.

When Box2DSimulation updates the particle system, it sends out a ParticleEvent, containing a ParticleRenderBatch and the texture to use for the particles. This is picked up in the view, and the ParticleRenderBatch is sent to the ParticleRenderer. The ParticleRenderer extracts the ParticleRenderNodes from the ParticleRenderBatch, and stores them in a map based on the texture used by that particular batch. When rendering the particles, all particles using the same texture are rendered in the same pass. The particle renderer therefore uses N passes, where N is the number of unique textures used by all the visible particles. The particle renderer always discards the particle data after drawing it. Particles are supposed to move, bounce, and scatter about, so the probability that a significant portion of the data will remain valid until the next frame is low.

Figure 34: Particle system



5.5 API Usage

The API for the particle system in Nox was designed in the same spirit as Nox. It therefore uses JSON, like all other components. Given time, the particle system would be given an API upgrade in the simplified API improvements described in 7, so developers would be free to construct particle systems directly in the code.

One thing to note in this section is that there are many values with corresponding offset values. These refer to RandomType values. They allow for an offset to be set. If set, the value, will be a random value within "offset" from its original value. These values are named "*Offset". These are not necesary, as they will initialize to 0 if undefined. They are however reccomended, as some randomness adds "life" to the system.

As described in Section 5.4, the particle systems primary components are particle components, particle emitters, and particles.

The definition of the particle component is a "Particle" object, containing an "Emitters" array. The "Emitters" array must contain *at least* one valid object, as a particle component without a valid emitter is completely useless. In the same way, the "particle" array in an emitter must contain at least one valid Particle definition for the emitter object to be valid. You can see an example of a validly defined component in Figure 35. 50 times a second, JohnTheEmitter will emit PollyTheParrot. Polly will go at a speed between 4.5 and 5.5, be a blueish color, and live between 0.2 and 1.0 seconds.

Figure 35: Particle component definition

```
"components":
{
    "Particle":
    {
        "Emitters": [ {
            "name": "JohnTheEmitter",
            "position": {"x":-2.0, "y":-2.0},
            "size": {"x":1.5, "y":0.5},
            "emitRate": 50.0,
            "velocityDirection": {"x":10.0, "xOffset":5.0, "y":-5.0, "yOffset": 2.5},
            "particle":[ {
                "name": "PollyTheParrotParticle",
                "velocityScalar": 5.0,
                "velocityScalarOffset": 0.5,
                "color": {"r": 0.1, "g": 0.1, "b": 0.9, "a": 0.8, "rOffset": 0.1,
                             "gOffset": 0.1, "bOffset": 0.1, "aOffset": 0.2},
                "lifetime": 0.6,
                "lifetimeOffset": 0.4,
                "flags": ["tensile", "staticPressure", "repulsive"]
            } ]
        } ]
    }
}
```

The value named "position" in the emitter sets the position relative to the position of the Actor the component is attached to. The size is the size of the area within particles can spawn. The "emitRate" is a floating point value, giving the rate of emission in particles per second. The "velocityDirection" describes the vector particles will spawn along. The offsets allow for randomness in the direction. The vector is normalized before emission, and multiplied by the speed of the emitted particle.

In the particle array in the particle emitter definition contains particle definitions. The offsets in here are optional, but as in the emitter definition, they are reccomended for the purpose of livening up the system. Even in just a small degree, they are useful.

The "velocityScalar" is the speed at which particles spawn along the direction defined in the emitter. The "color" is defined by red, green, blue and alpha. The value is defined in floats from 0 to 1. The "lifetime" is the number of seconds a the particle live for. The "flags" array contains flags that give properties to each particle. These are defined in Table 4

Table 4: Particle flags[3]

Flag	Effect
waterParticle	Water particle.
zombieParticle	Removed after next simulation step.
wallParticle	Zero velocity.
springParticle	With restitution from stretching.
elasticParticle	With restitution from deformation.
viscousParticle	With viscosity.
powderParticle	Without isotropic pressure.
tensileParticle	With surface tension.
colorMixingParticle	Mix color between contacting particles.
barrierParticle	Prevents other particles from leaking.
staticPressureParticle	Less compressibility.
repulsiveParticle	With high repulsive force.

[3]Based on https://google.github.io/liquidfun/API-Ref/html/b2_particle_8h.html#a73d3b011893cb87452253b1b67a5cf50

6 Audio Improvements

6.1 Requirements

The Nox engine originally only supported the absolute minimum in terms of playing sounds. Sounds could only be played without position, meaning that sounds originating from afar are heard as being right in front of you. This was something Suttung Digital requested to be improved upon, and we believe will help make the engine more attractive and interesting, as discussed in Section 1.1.

The previous implementation used a bare-bones wrapper around OpenAL. To avoid having to reinvent the wheel, we decided to investigate what third-party solutions exists. We were looking for third-party sound libraries with the following criteria:

- Cross platform support
  The third-party solution must support Linux, Mac OSX and Windows natively. Other platforms are a bonus.
- Basic audio properties
  The third-party solution must support positional sounds, preferrably also supporting velocity & doppler effects.
- Effect and filter support
  The third-party solution must offer a relatively comprehensive set of effects and filters out of the box. If the solution also supports custom effects and filters, that is a bonus.
- Open source
  The third-party solution must be open source, and released under a license that allows for binary distributions of the application using the library (The MIT license, for example). Suttung Digital are strong believers in OSS, and wishes to bring Nox to mobile platforms. Open source libraries are easier to deal with in terms of cross compiling, as you don't have to rely on the developer to provide pre-compiled libraries.

In addition to the list above, we were also taking into consideration the size of the user base, the age of the project and the frequency of updates. Through our research, we found a handful of potential libraries.

The libraries evaluated for integration with Nox, and the major downsides with each of them were:

- PortAudio *(No sound effects)*
- IrrKlang *(Proprietary, only free for non-commercial products)*
- SoLoud *(No sound effects)*
- FMOD *(Proprietary, only free for non-commercial products)*
- SFML::Audio *(No sound effects)*

- cAudio *(Difficult to add custom sound effects)*

With all requirements taken into account, `cAudio` was the chosen library. It supports sound effects, filters, position, velocity and more. It is licensed under the `zlib/libpng` license [1], which means that it is freely distributable in binary form, as long as the copyright notice is included in the distribution. cAudio is based on OpenAL, rendering its cross platform support as wide as that of OpenAL.

6.2 Design

Positional sound requires that a movable object in some way or another becomes *associated* with the sound. This was solved by adding an abstract class called `AudioBufferPositionProvider`. The class contains a method `updateAudioBuffer(AudioBuffer* buffer)` in which implementations may assign *available* and *desired* attributes to the sound.

As the *listener* can have both a position and a velocity, a similar class named `AudioListenerPositionProvider` was added. The class is designed in the same fashion - the `AudioListenerPositionProvider` is registered with a `AudioSystem`, and the `AudioSystem` calls the virtual method `updateListenerPosition(AudioSystem*)` every frame. The `AudioListenerPositionProvider` also here assigns all *available* and *desired* attributes.

In order to seamlessly tie the system together, a component named `ActorAudio` was added to the engine. The component extends the abstract class `AudioBufferPositionProvider`. This component has an interface for playing sounds, and sounds played via this class are automatically associated with the component. The component checks every tick for *sibling components* [2] of type `Transform` and `ActorPhysics`. The `Transform` component provides the position of the actor, and the `ActorPhysics` provides the velocity. Having a `Transform` and / or `ActorPhysics` sibling is strictly optional, but useful. A new camera class was added to ease the work of assigning the listener's position. The camera class simply assigns it's up-vector and position to the listener.

6.3 Implementation

Being an engine with modularization as a key aspect, extending Nox to support a second sound framework is by design easy. The sound sub-system, among others, uses *interfaces* to abstract out the *implementation*. When using OpenAL as the back end for playing sound, the game developer never (or rarely) interfaces with `OpenALSystem` and `OpenALBuffer` - the interfaces `AudioSystem` and `AudioBuffer` are used instead. Bar the creation of the `AudioSystem` instance, no significant reference to `OpenALSystem` is ever made. Adding cAudio is therefore trivial. The classes `CAudioSystem` and `CAudioBuffer` will handle the logic required to play sounds via cAudio, and `Application` will decide, based on compiler flags, which audio system (OpenAL or cAudio) to instantiate.

The previous sole implementation using OpenAL did not support 3D sound or velocity. These attributes had to be added to the abstract interfaces `AudioSystem` and `AudioBuffer` in order to access the deeper functionality of cAudio. The OpenAL-

[1] http://opensource.org/licenses/Zlib
[2] Components attached to the same parent actor.

specialized classes implements these methods, with empty bodies. [3]

[3] OpenAL supports positional sounds, effects and velocity, but the OpenAL-specialized classes does not

7 Simplified API

7.1 Requirements

The basis for improving the API was primarily a large amount of boilerplate required to get a simple project running, as discussed in more detail in Section 7.2. Improving the API was our own suggestion, and Suttung Digital welcomed the idea as it could make the engine more attractive.

The requirements we set for the simplified API were:

- Insert a new *"binding"* layer between the game code and the Logic layer *(see Section 1.9)*
  The new layer should abstract away the difference between the Logic and Application layers and perform most of the tasks related to making these layers communicate with the game code.
- Expose more of the APIs through code to offer functionality previously exclusive through JSON files.

We aim to make the engine easier to use, in order to please potential users *investigating* the engine. [1]

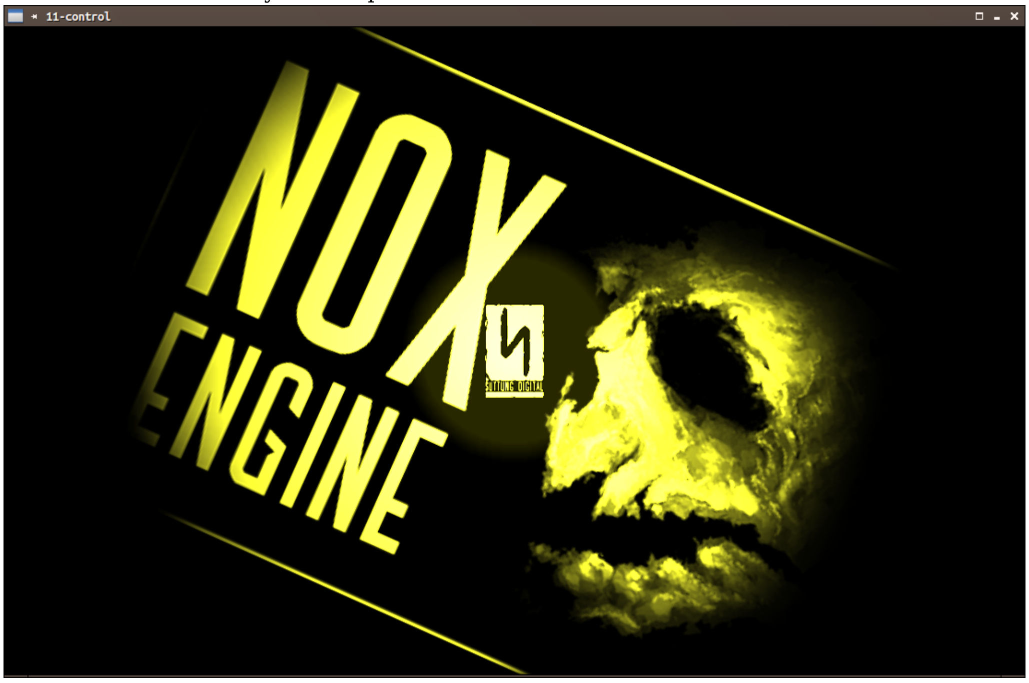
7.2 Nox Usage

As discussed in the architectural description of Nox, the engine is structured into two primary layers; *the application layer* and *the logic layer*. The game code can be considered a third layer, hierarchically equal to the logic layer, as shown in Figure 1. The game code is responsible for doing much of the *binding* between the logic and application layers.

Suttung Digital has created a wide array of samples demonstrating how to use and access the various aspects of the Nox Engine [2]. We will go into detail about one of the samples called `11-control`, which demonstrates how to create a simple scene in which an actor can move around in the game world.

[1] See discussion in Section 1.1.
[2] Nox samples are hosted at https://bitbucket.org/suttungdigital/nox-engine-sample.

Figure 36: Screenshot from the sample `11-control`. The Suttung Digital logo can freely move around the world via keyboard input.



The game code must provide an `Application` and a `View`. The `Application` is responsible for initializing the game and providing *application layer services* [3]. The `View` is primarily concerned with controlling the actor, and providing methods for the player to *view* the game logic by displaying it on the screen.

In concrete terms, the responsibilities required for the sample `11-control` to initialize and **use** the engine, are listed below. Responsibilities which are solely concerned with controlling the actor, i.e., the *non-boilerplate aspects* are omitted from this list. There is made no distinction between the `Application` and `View` in this list. These are steps required for nearly every application created using Nox, but the list is not comprehensive; more advanced applications have additional responsibilities. *Some class names have been changed to be more verbose to more efficiently communicate their purpose. The class referred to as `FileHandle` is for instance really called `Handle`.*

- Create a `ResourceCache` to gain access to *file system providers*
- Create file system providers directories for all the directories containing files required by the sample
- Create and initialize the physics subsystem
- Make Components included in the Nox engine available to the World Manager
  This is required to allow the World Manager to recognize Components by string

[3] As described in section 1.9, the application layer is responsible for providing low-level features such as file access and graphical functionality.

references (for instance, "ActorPhysics", and create an instance of the corresponding class.

- Load actor definitions
  This is required to let the World Manager know how to create actors by string references. For instance, the actor "Player" is defined in `player.json`.
- Load the world
  1. Get a file `FileHandle` to the world file from the `ResourceCache`
  2. Ensure that the file is of JSON-format
  3. Create a `WorldLoader` and register the `View` with the `WorldLoader`
  4. Load the world by giving the `WorldLoader` the `FileHandle`.
- Define which texture atlas to use
  The texture atlas is defined in a JSON file containing the specific image files to use and the regions of the sub-sprites in the sprite sheet.
- Create a `Camera` instance, and assign it to the `OpenGlRenderer` class
- Dispatch `WindowWasResized` events to the `Camera` instance
- Handle `TransformChange` events
  This is required to update the graphical renderer when an Actor moves, is rotated or in another way has had it's transform altered.
- Handle `SceneNodeCreated` and `SceneNodeRemoved` events
  Whenever an actor is created or removed, it is the game code's responsibility to update the scene graph.

All of the above responsibilities are explicitly handled by `11-control`. Nearly every game built with Nox must also handle the items above. More advanced games have additional responsibilities. Playing sounds require the game code to handle the playing sounds, for instance.

The sample consists of five C++ files; `main.cpp`, `ControlApplication.cpp`, `ControlApplication.h`, `ControllingWindowView.cpp`, and `ControllingWindowView.h`. The total application sums up to 352 lines of source code *excluding comments and blank lines*. In addition to this, 202 lines of JSON divided between six files is also required to define the world, the actors, the control scheme, and the texture atlas. The build system must also be defined, either through CMake, or by setting up IDE projects.

Requiring a total of 554 lines of code & JSON to merely move a sprite around on the screen is in our opinion excessive. In order to create such a simple application, the knowledge required about the engine creates a high barrier of entry. We believe that these metrics will prevent the engine from attaining a large user base, as they are in direct conflict with our belief that a game engine should have a low barrier of entry & productivity [4].

It can be seen from the list of application responsibilities earlier in this section that applications are required to handle a **lot** of binding between aspects of Nox -

[4]See discussion in Section 1.1.

most of which can, and should be hidden away internally in the engine.

7.3 New API Design

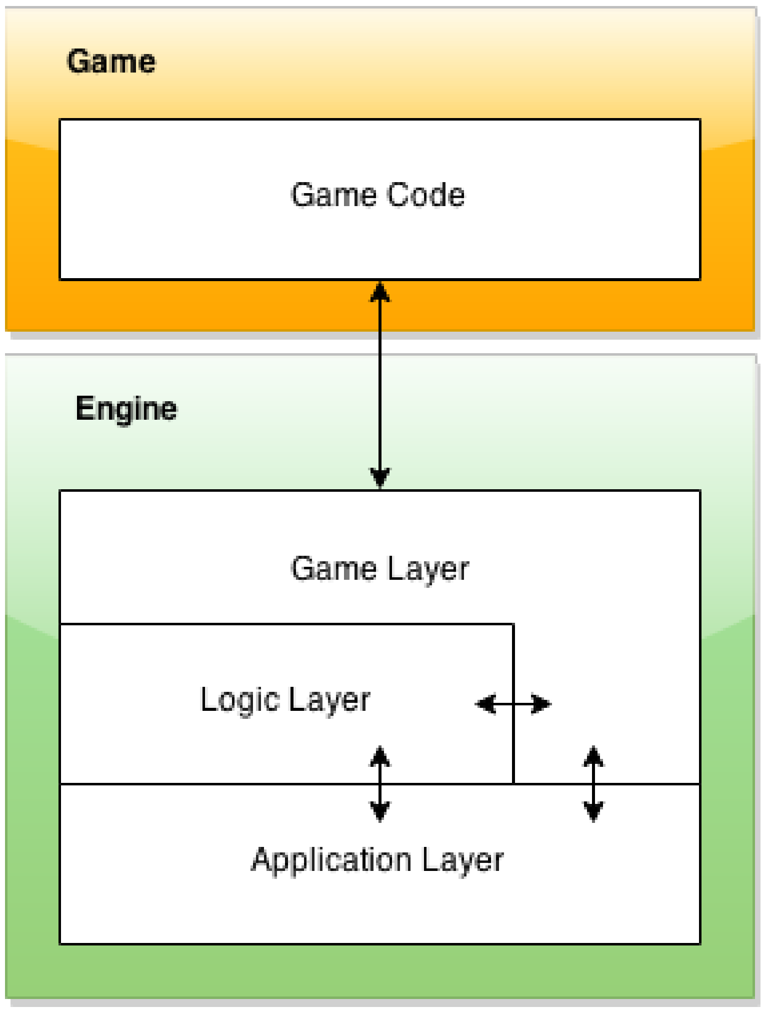
Abstraction

It can be seen from the list of application responsibilities in Section 7.2 that applications are required to handle a **lot** of binding between aspects of Nox - most of which can, and should be hidden away internally in the engine. The large number of responsibilities are rooted in the fact that the engine was designed as a two-layered system *(Application- and Logic layer) as discussed in Section 1.9*, where the game code is directly incorporated into the logic layer. As a result, there are many loose threads that must be tied together, as the game code is largely responsible for making the logic and application layer communicate.

The improved API is required to abstract away the interconnection between the logic and application layer. Users of the engine should not be required to know about the application and logic layer, even less the specific responsibilities each layer has.

The game code was originally hierarchically equal to the logic layer, as desribed in Section 1.9 and shown in Figure 1. We inserted a new layer called the *Game Layer* between the two. The game layer is responsible for eliminating as much boilerplate from the game code as possible, and abstracting out the difference between the Logic and Application layers. The new game architecture is shown in Figure 37.

Figure 37: Nox's new 3 layer architecture



Expose APIs

We consider learning a new set of APIs easy compared to learning the expected object structure of a JSON file. With modern IDEs you have powerful visualization tools to graphically show class relations and methods. We can expect that potential users are already familiar with C++ - however, we cannot assume that they are familiar with working extensively in JSON. We therefore believe that in order to improve the *ease of use* discussed in Section 1.1, the Nox Engine should provide much of the same functionality through C++ APIs as it does via JSON files. The C++ APIs must provide at least the *core* functionality. We expect that the majority of the users desiring to use the advanced features will be more willing to *go the extra mile* and learn the JSON formats - the C++ APIs will mostly target beginners. Advanced users are more likely to use the JSON interfaces even if the functionality offered is identical, as the project does not require to be recompiled when changing the contents of a JSON file, thus greatly increasing the development cycles.

We believe that all the JSON functionality should be exposed in C++ to some degree, but we have chosen to focus only on the APIs to create actors and populate the world.

7.4 Implementation

7.4.1 Game Layer Abstraction

The game layer, inserted between the logic layer and the game code, eliminates the need for a large portion of the boilerplate code. The layer also removes to some extent the requirement of a deep knowledge of Nox's internal structure.

With the original design, a subclass of `Application` usually handled most of the *loose threads* of the API. This included initializing core objects such as the graphics renderer, the resource access object, and registering these objects with the proper objects. This responsibility has been moved to the new class `GameController`, which is now the new entry point of Nox *(the previous entry point was `Application`)*. In order to maintain the flexibility of the *do-it-yourself* approach to initialization that previously existed, the Game Controller has multiple methods whose sole purpose is to *create* objects. The default implementations of these methods returns the most commonly used specializations of the requested interfaces, and can be overridden to customize the program flow further. Internally during initialization, the Game Controller queries itself through these methods to create objects of certain types, and does the binding of these objects "behind the curtains". All the *creation-interfaces* are listed in Figure 38.

Figure 38: Game Controller Creation-interfaces

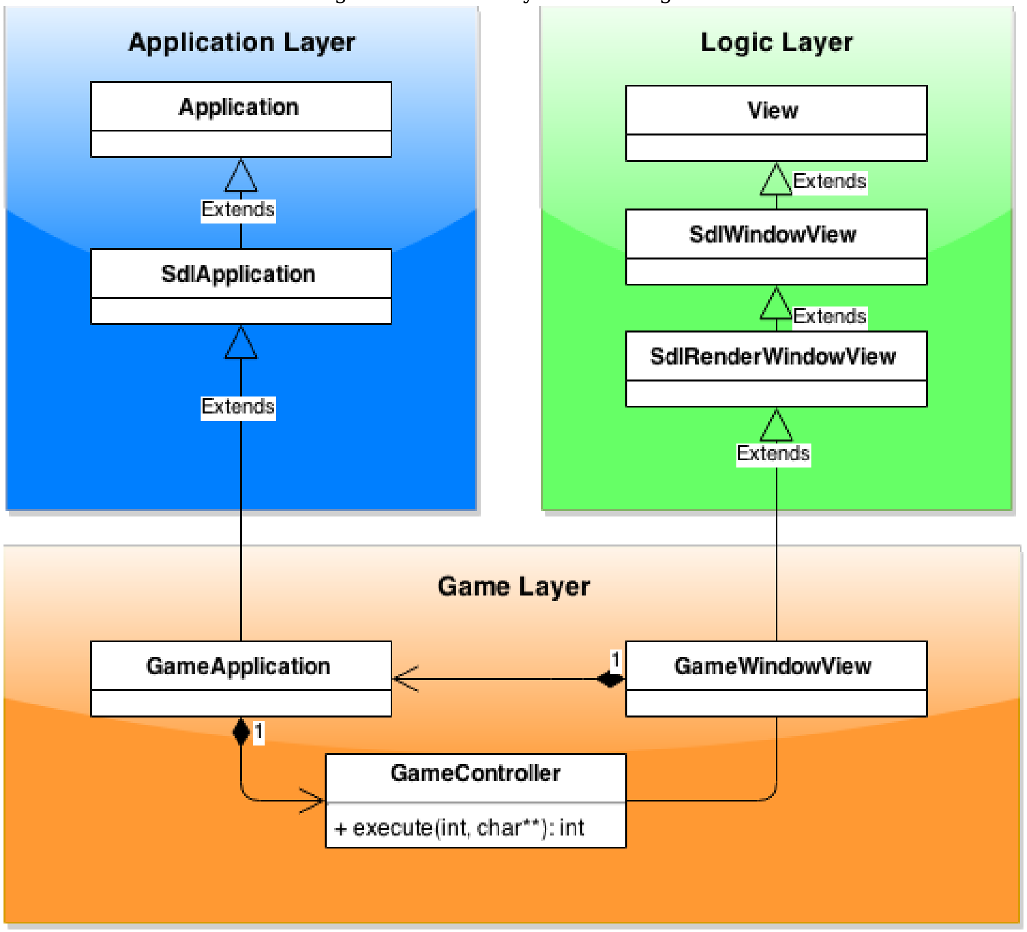
```
class GameController
{
public:
    virtual std::string getGameName() const;
    virtual std::string getOrganizationName() const;
    virtual std::unique_ptr<GameApplication> createApplication();
    virtual std::unique_ptr<logic::Logic> createLogic() const;
    virtual std::unique_ptr<app::resource::Cache> createResourceCache() const;
    virtual std::unique_ptr<app::resource::Provider>
            createResourceProviderForAssetDirectory(const std::string& dir) const;
    virtual std::unique_ptr<app::storage::IDataStorage> createDataStorage() const;
    virtual std::unique_ptr<GameWindowView> createGameView() const;
    virtual std::vector<std::string> getResourceCacheDirectories() const;
    virtual std::vector<std::string> getActorDefinitionSubdirectories() const;
    virtual std::vector<std::string> getWorldAtlasDefinitionFiles() const;
    virtual std::string getWorldAtlasName() const;

    // --omitted--
};
```

The simplified API aims to make the system as flexible as possible, but some flexibility removal was ineviteble. Games using the game layer are forced to commit to SDL and OpenGL. The commitment to SDL and OpenGL hails from the new required types for `View` and `Application`, which are `GameWindowView` and `GameApplication`. As can be seen from the diagram in Figure 39, they are both derived from SDL-specialized implementations of View and Application. In order to replace either SDL or OpenGL, Game Window View and Game Application must be rewritten to some degree.

We consider these "abstract dependencies" to be superior to "explicit dependencies". If SDL or OpenGL were to be replaced entirely in Nox, only Game Window View and Game Application would require a rewrite, and the game would probably not be able to tell the difference (and therefore not require any alterations). With "explicit dependencies", where the game code itself requests specific technologies like OpenGL or SDL, all games built using the *old* library must be changed to explicitly use the *new*.

Figure 39: Game layer class diagram



The sample `11-control` which was discussed in Section 7.2 was reimplemented with the new game layer API. The original implementation required 352 lines of effective C++ code. The implementation using the game layer API requires 54 lines of code to achieve the exact same functionality. Both samples depend on the same JSON files, which sums to 202 effective lines. We consider this to be a step in the right direction for users *investigating* the engine (see Section 1.1).

7.4.2 Exposing JSON Functionality

We chose to focus on exposing the APIs to create actors, attach components to actors, and configure components. Previously, this was only possible to do through JSON definition files.

Figure 40: Actor creation in JSON and C++

```json
{
"Player": {
    "name": "PlayerActor",
    "components": {
        "Transform": {},
        "Sprite": {
            "spriteName": "player.png",
            "scale": 3.0
        },
        "GunFireComponent": {
            "fireRate": 15.0
        }
    }
}
```

```cpp
void createActor()
{
    GameController *game = getGameController();

    // GameController::createActor creates an actor, adds it to the world,
    // and returns a raw pointer to it.
    Actor *actor = game->createActor("PlayerActor");

    actor->addComponent<Transform>();
    actor->addComponent<Sprite>();
    actor->addComponent<GunFireComponent>();

    actor->getComponent<Sprite>()->setSpriteName("player.png");
    actor->getComponent<Sprite>()->setScale(3.0);

    actor->getComponent<GunFireComponent>()->setFireRate(15.0);
}
```

The difference between creating an actor in JSON versus creating one via the C++ APIs can be seen in Figure 40. While there are strong benefits in favor of the JSON approach, we believe the C++ APIs should be an option to the user of Nox. We especially believe this to be beneficial for new users, as discussed in Section 7.3.

8 Group Work

8.1 Group Structure

Our work process has been freely structured within a set of rules. We agreed that we would work full 40 hour weeks, unless special conditions applied (and the other person was notified). We worked primarily from own our homes, except during the few occasions where "pair programming" benefited us greatly.

We used a modified version of Scrum as our development model. Being only two people, we didn’t have need the need nor the resources to designate either to being the Scrum Master. Also, as we had no designated working area, we had our daily stand-up meetings over voice chat. Our sprints were two weeks in length, and generally [1] ran from Friday to Thursday to align our Sprint Review meeting with our meetings with Suttung Digital, which were every other Thursday. Over the course of the thesis, we relied more and more on brief and frequent communication and task delegation over voice chat and instant messaging services than tracking issues and progress in Jira. We think this was a reasonable choice for us, given our group size.

In our biweekly meetings with Suttung Digital, we would present the work we had done for the past sprint, and receive suggestions on what to do next or how it could be improved.

8.2 Evaluation of Group

Our group effort has largely worked well. Our communication has been frequent in both person and over voice chat services. However, the communication of smaller, day to day issues like asking for help when we were stuck could be definitely be improved. We never had any code review sessions internally, thus fragmenting the knowledge between us, and subsequent difficulties when treading into the other person’s code. In retrospect, we consider the lack of code review sessions to be our greatest hindrance as a team.

[1] Some of our sprints were extended and shifted by a couple of days.

9 Task Critique

Our task was initially unclear, as discussed in Section 1.1. The Task Description given by Suttung Digital, found in Appendix B, gave us the impression that we would spend our time primarily on *optimizing* existing features. In the initial meetings with Suttung Digital it became clear that our task would be focused more towards improving the engine by adding functionality and features. We changed our focus towards *improving the audiovisual aesthetics* of Nox, and later again shifted our focus to *accommodating Nox for community adoption*.

Even though the focus of the thesis changed twice, we consider all our developed features to fit into our final thesis. Suttung Digital were supportive of our thesis's best interest throughout the entire process, and never *demanded* us to implement any one feature; we were able to decide ourselves which proposed features we felt would make the best thesis. The topic chosen gave us the freedom to implement the features *we* consider to be important in a game engine.

10 Conclusion

10.1 Goal Discussion

As stated in section 1.1 we were tasked to *accommodate the Nox Engine for community adoption*. To do this, we were going to focus on the first two steps in choosing a game engine, *interest* and *investigation*. By adding features and making the engine easier to use, we hoped to make the engine a more viable solution when selecting a game engine.

By adding better audio, a particle system, networking features, normal and specular maps we have hopefully made Nox a more interesting game engine.

By changing the API, as described in chapter 7, we have shortened the boilerplate code needed to set up a project from 352 lines of C++ code, to 54 lines of C++ code. This combined with the extended feature set will hopefully make the users that *investigate* the engine more likely to go trough with using the engine.

In retrospect, we consider *not* adding mobile support (listed in Section 1.10) to Nox as the biggest mistake of our feature selection. Mobile games are popular to develop among hobby developers, and the distribution channels [1] have lower barriers of entry than the PC distribution channels [2]. If we were to do this project again, we would likely drop the stencil-shadow implementation in favor of mobile support. If the time available had sufficed is unknown, but this would be the first feature to be replaced.

10.2 Continued Development

Working with the Nox engine there were certain things we were asked to consider, or noticed a need for, but were outside the frames of our task. In case we or anyone else were to continue development on the Nox engine, we have put a few suggestions for future improvements that could benefit the engine in this section. We consider the suggestions listed below to be in line with the subject of this thesis.

Mobile support

We chose not to focus on maintaining a mobile build for Nox. All the libraries used in Nox are available on mobile. Making the Nox engine run on more platforms, specifically Android and iOS. There are certain things that must be changed for mobile, but allowing for games to be developed for other platforms than the main desktop operating systems would be a vital tool for "selling" the engine to potential users.

World Editor

Creating tools for easily creating worlds, using a graphical editor might make game design easier, especially when prototyping a game. The current approach of writing your own JSON files is time consuming, and improvements could be made here.

[1] Most notably Apple AppStore, Google Play, and Amazon Appstore.
[2] Most notably Steam, Good Old Games, and GamersGate.

GUI functionality

The engine does not currently have any GUI components in it. Creating a framework for adding graphical components directly mapped to the screen layout could prove a rather big task, but would add a lot of value to Nox. Creating the base system for GUI elements, adding a few layout options. Basic functionality like buttons, text inputs, text rendering, checkboxes and similar components would be important for actually making games with the engine.

10.3 Conclusion

Our task has been to accommodate the Nox engine for open source community adoption. We considered our primary objective in this regard to enforce the experience of users *investigating* the engine, and to make it more *attractive*, as discussed in Section 1.1.

We consider our improvements to the engine's attractiveness to be our biggest achievement. The engine now has more *stand out* features, which may tip investigating users in Nox's direction. The APIs of the added features are easy to use, and fit into the Nox philosophy.

While we don't expect the Nox engine to explode in popularity after our additions, we consider our efforts a good step along the road to community adoption.

Bibliography

[1] Vik, M. B. & Sporaland, A. 2014. Pyroeis.

[2] Heggen, E. S. 2015. In the age of free aaa game engines, are we still relevant?

[3] McShaffry, M. & Graham, D. 2012. *Game Coding Complete, Fourth Edition*. Cengang Learning.

[4] Gamasutra, Jonasson, M., & Purho, P. 2013. Is your game ’juicy’ enough? "juice it or lose it".

[5] Gillen, K. 2010. Illuminated ones: Shadow & light in games.

A External Repositories

Our Jira and Confluence spaces can be found at the following address.
`https://dev.imt.hig.no/confluence/display/OPTNOX/OptNOX`

Our fork of the Nox engine. All developed features decribed in the thesis are as of May 14th, 2015 in the process of being merged into Suttung Digital's repositories.
`https://bitbucket.org/optnox/nox-engine`

Our fork of the Nox engine samples. The features developed for this thesis are included in their own sample projects, in directories prefixed with `"on-"`.
`https://bitbucket.org/optnox/nox-engine-sample`

Suttung Digital's Nox repository.
`https://bitbucket.org/suttungdigital/nox-engine`

Suttung Digital's sample repository for the Nox engine.
`https://bitbucket.org/suttungdigital/nox-engine-sample`

B Task Description

Oppdragsgiver: Suttung Digital
Kontaktperson: Asbjørn Sporaland
Adresse: Raufossvegen 40, 2821 Gjøvik
Telefon: +47 95080723
Epost: sporaland@suttungdigital.com

Summary

Customising a game engine to improve the performance in a specific development area

Background

Suttung Digital is a company located at Mustad in Gjøvik and is currently developing a game called Astral. They use an in-house game engine called NOX Engine written in C++ for this game. The NOX Engine is a game engine designed to be flexible and extendible to make it applicable for different types of games. It is highly modular so that changing the subsystem implementations can easily be done without breaking the system. This engine has advantages and disadvantages in comparison to industry game engines such as Unity3D, Torque2D/3D, and Unreal Engine

Description

The first part of this task is to evaluate the various game engines that are available, and assess their features in relation to managing timelines, controlling the use of 3D and 2D, lighting effects, AI authoring, and 3D sound manipulation.

The main programming work for this project is to extend one of the open source engines, NOX Engine, or Torque2D/3D, to improve the handling in one of the three areas reviewed. This will involve creating wrappers around game functionality and editing the core functionality of the game engines. The technical improvements will be tested using a simple game developed to demonstrate the improved performance of the game engine.

The task will involve system design, high level C++ programming and some low level programming (potentially GPU shaders, or low level sound management). The technology used is C++ (ISO C++11), OpenGL 3 and several libraries which could include SDL2, Box2D, Bullet, ODE, OpenAL, Boost and more.

C Nox Coding Conventions

C++ Code Style

Source

- Use **tabs for indentation**, not spaces.
- End all files with one newline.

Naming

- Generally names should be **short, sweet and to the point**. They should describe the purpose of the function, but not elaborate.
- Never use **prefix or postfix** on names (variables, functions, everything).

Classes

- CamelCase is used for **classes**. E.g. "class SomeAwesomeClass".
- Don't use **several uppercase letters** for a word. E.g. an AABB box class will be "Aabb".
- **Interface classes** must begin with a capital “I”. e.g. class IOnEvent.

Variables and Functions

- camelBack is used for **variables and functions**, including class members.
- **Global variables** start with the "g_" prefix. E.g. g_systemLogger.
- **Static constant** variables (global const and static class member) are all uppercase with underscores as spaces. E.g. STATIC_CONSTANT.

Files

- **Header files** have the extension ".h".
- **Source files** have the extension ".cpp".
- If a **file contains a single class**, the file is named exactly the same as the class (including cases) plus the extension.
- If a **file contains several classes/functions or other things**, they are named with all lower case and underscores.

Coding

Files

- Headers must have **include guards**, rather than “pragma once”. Include guards should have the format <NAMESPACE><<_NAMESPACE>...>_FILENAME_H_. As an example, a file named LruCache.h with the namespace nox::app::resource would have the include guard "NOX_APP_RESOURCE_LRUCACHE_H_"
- Avoid including header files in header files as much as possible. Rather **forward declare** classes and include header files in the source file.
- **Include header files** in the following order:
    1. Header file corresponding to this source file (if applicable).
    2. Headers from the same component.
    3. Headers from other components.
    4. Third party library headers.
    5. C++ Standard Library headers.
- Have a newline as the final character of the file.

Classes

- Always use **this->** when you reference a member of a class (variables and functions).
- Avoid using **static member** variables and functions. Constants are an exception.
- Always use the **override** keyword when overriding a base class method.
- Avoid **public member variables**.
- Avoid **protected member variables**, rather create protected member functions to operate on the variables.
- Avoid **deep inheritance hierarchies.**
- Constructors taking one parameter must be marked as **explicit**.
- When inheriting, the colon must be placed like this: "class SomeClass: public InheritedClass".
- Constructors should initialize the member variables through an **initializer list** rather than assignments.
- When using an initializer list, there should be no space between the constructor and the colon. All variables initialized should be placed under the constructor on one line each with one tab to the left.

Functions

- No space between the name and parentheses.
- Space after comma between each parameter.

Formatting

- **Curly braces** always start at a new line, if not a list (e.g. std::initializer_list).
- Space between test (if/while) and parantheses of statement.

Patterns

- Avoid using **global variables**.
- Avoid using **singletons** (which in practice is a global variable).
- Follow these guidelines in regard to **pointers** (ownership is important): http://stackoverflow.com/a/8706254
- Use **const** wherever sensible, as on methods (e.g. accessors), references, pointers, that shouldn’t change the data.
    - **Variables passed** to methods should be **const** if they don't change.
    - **Objects passed** to methods should be passed as **const references** if they don't change.
    - **Methods** not modifying class members should be marked as **const**.
    - **Variables declared** that won't change should be marked as **const**.
- Methods can be **inlined** if they are short and called many times per frame.
    - Inlined methods must be defined below the class declaration and not directly in the method declaration.
- Defined types (struct, enums, etc.) used privately within a class should be defined in the class' private scope.
- Defined types (struct, enums, etc.) that are tightly connected to a class, but are used publicly, should be defined in the class' public scope.

Casting

- Never use **C-style cast**, always use one of C++'s explicit cast expressions (most commonly static_cast). This also goes for numbers. Se http://en.cppreference.com/w/cpp/language/explicit_cast

Other

- **Pointers and references** must be declared with the asterisk or ampersand to the left (e.g. int* a;)
- Never use “**using namespace**”, always use “namespace::thing”.
- The **auto keyword** usage should be minimal. It should only be used for very long-typed iterators and where the value is obvious.

Example

Class

```
// WorldBody.h

#ifndef NOX_WORLD_WORLDBODY_H_
#define NOX_WORLD_WORLDBODY_H_

namespace nox { namespace world
{

class WorldBody: public Body, public IEventListener
{
public:
 const glm::vec2& getPosition() const;
 float getRotation() const

 void setPosition(const glm::vec2& position);
 void setRotation(const float rotation);
 void setTarget(const Target* target);

 void onEvent(const Event& event) override;

private:
 glm::vec2 position;
 float rotation;
 const Target* target;
}

}

#endif
```

auto keyword

```
void someFunction()
{
 int a = 0;
 auto b = a + 1; // BAD.

 std::vector<SomeClass> elements;

 // BAD. Need to look at the elements declaration to see what the type is. It is
even worse if elements is not declared locally.
 for (const auto& element : elements)
 {
 }

 // GOOD. Quicker to see what the type is.
 for (const SomeClass& element : elements)
 {
 }

 // OK. begin() always returns an iterator, and typing a vector iterator typename
is a hassle and clutters the code.
 auto elementIt = elements.begin();

 // OK.
 auto somePointer = std::make_shared<SomeClass>();
}
```

Formatting

```
void someFunction()
{
 int a = 0;
 int b = 0;

 std::cin >> b;

 if (a == b)
 {
  for (int i = 0; i < 10; i++)
  {
   printf("%s", "So awesome.\n");
  }
 }
}
```

D Nox Documentation Conventions

C++ Documentation

What to Document

- Public type definitions (classes, structs, enums, typedefs, etc.)
- Public variable and function declarations (member functions/variables, global functions/variables).

How to Document

Format

The Javadoc style of the Doxygen format is used. The Doxygen generator will run over the code and generate HTML documentation, so it's important to strictly follow the format. All lines written should start with a capital letter.

Multiple lines of comments are enclosed with a /** */ style comment. Each line starts with an asterisk:

```
/**
 * This thing is so awesome
 * that it has to be written
 * over several lines.
 */
```

The Javadoc brief description is used, meaning that the first sentence of the comment will be used as the brief description. Write this description short, but to the point. The rest of the comment will appear in the detailed description.

```
/**
 * This is the brief description (until the dot). The rest
 * is the detailed description.
 */
```

One-line documentation should be written with the //! format:

```
 //! This is a short one-line description.
```

When documenting items in a list, such as an enum, the documentation can be written after the item using the //!< format:

```
enum Things
{
 THING_1, //!< This is the first thing.
 THING_2  //!< This is the second thing.
}
```

Classes

Describe the purpose of the class, and how to use it. Some classes might only need a brief description, while others will need a more elaborate detailed description. E.g. "Tree placed in the world", "Object simulated by physics".

Functions

Describe the purpose of the function, its parameters, and its return value. Short functions such as getters and setters might not need documentation.

The purpose should not be written as a thing that it does, but what you do when you call it.

- Good: "Calculate the addition of two numbers."

- Bad: "Calculates the addition of two numbers."

Each parameter should have one line of documentation describing what they are used for. If a parameter is an output parameter, use "@param[out]". Some parameters won't need documentation as their point is obvious.

The return value should have one line of documentation describing its purpose. Some return values won't need documentation as their purpose is obvious.

For templates, the template types must, if they are not obvious, be documented using @tparam.

Example C++ Function

```
/**
 * Convert a vec2 type to another vec2 type.
 * @tparam InputVec2 Type of input vector. Must have public InputVec2::x and
InputVec2::y.
 * @tparam OutputVec2 Type of returned vector. Must have constructor taking
InputVec2::x and InputVec2::y.
 * @param inputVector Vector to convert.
 * @return Converted vector.
 */
template<class OutputVec2, class InputVec2>
OutputVec2 convertVec2(const InputVec2& inputVector);
```

E Project Plan

Kristoffer was originally a part of the group. After a mutual agreement, he left the group.

PROJECT PLAN

Sverre Blekastad Stensby
Kristoffer Alvin Dahlgren
Joakim Nordstrand Stien

TABLE OF CONTENTS

1. GOALS AND CONSTRAINTS

1.1. Background

Suttung Digital is a software company localized in Gjøvik. Suttung Digital wishes to have their open source game engine, *NOX Engine*, improved. Improvements may include the addition of new features, and improvement and optimization of existing features.

The engine is used by Suttung Digital in development of their game *Astral*, and has been in development for a few years by the employees at SD. The engine is part of *Pyroesis*, a bachelor thesis by Magnus Bjerke Vik and Asbjørn Sporaland[1], both of which are now working at Suttung Digital.

1.2. Project goals

Goals

The project should result in an improved game engine which will offer more functionality than what currently exist. Additions to functionality will be easily accessible when developing a game using the engine. To the greatest extent possible, our work on the engine will **not** break existing APIs. Existing projects built upon the engine will be, if not fully compatible with our changes, very easy to migrate to the new APIs. The project will not break compatibility with NOX's currently supported environments; Windows, GNU/Linux and Mac OS X.

The performance of various aspects of NOX Engine will be measured and analyzed. Based on our findings, areas with poor performance may be optimized. The performance of demo projects will be compared with similar projects written for other game engines to provide a meaningful performance benchmark.

To demonstrate the additions and performance, a simple game will be developed to showcase the new features of NOX Engine.

Purposes

The [resultatmål] of this project is to attain the skill to understand and work on an existing codebase initially unknown to us, and the skill of collaborating on an open source project. We will learn how to analyze performance and profile the performance of complex systems.

1.3. Task Description

The task will include:

- Literature review of game development and practices
- Feature addition

[1]http://hdl.handle.net/11250/216763

- Feature improvement
- Optimization

1.4. Constraints and boundaries

The work will be primarily limited to the NOX Engine. To some extent, other game engines will be used to compare performance against the NOX Engine. When there are libraries we can integrate into NOX instead of developing our own, existing libraries will be used.

2. PROJECT ORGANIZATION

2.1. Roles and responsibilities

- Sverre Stensby
  Programmer
- Kristoffer Alvin Dahlgren
  Programmer
- Joakim Stien
  Programmer, team leader.

Under circumstances of high disagreement in the group, the team leader has authority to make the final decision, if the team is unable to come to an agreement.

2.2. Team rules and routines

We are meeting with the project supervisor, Mariusz Nowostawski, every Tuesday. Every other Thursday, we meet with Suttung Digital. Following both of these meetings - and every Thursday, we meet to discuss the progress internally. Each morning we have a daily scrum stand-up meeting over Skype *(or a similar service)*.

All meetings are mandatory by all team members.

3. PLANNING

3.1. Choice of development model

The project will be developed using the agile Scrum method. Scrum has been chosen due to the uncertainties of *what* we will be developing, and the duration of these tasks.

Our sprints will be two weeks in duration, and run from Friday to Thursday. Sprint review meeting will be done with Suttung on Thursday evening. Sprint retrospective meeting and sprint planning meeting will be held afterwards internally.

3.2. List of activities Work Breakdown Structure)

- Physics based animations
- Network functionality
- Sound with mixing and effects
- Profile and possibly optimize in the following areas
    - Graphics
    - Physics
    - Resource managment
    - Network
    - Sound

In addition to this, we may add other features, as they occur to us or Suttung Digital.

4. ORGANIZATION OF QUALITY ASSURANCE

4.1. Documentation

To ensure that no data losses occur, all files used in the project are hosted online via different hosting services. All source code related files are hosted on BitBucket, and all documentation is hosted on HiGs Confluence server.

4.2. Risk Analysis

Description	Probability (0-10)	Impact (0-10)
One of the developer loses access to hardware capable of running the game (fire/water damage ...).	1	6
One of the developers drops/is dropped from the project (illness/disagreement within the group).	2	3-6
Complete data loss.	0	10
Partial data loss.	9	1-2
We are unable to finish a satisfactory product on time.	3	1-10
Major cross platform compatibility issues	2-3	6-9
Fruitless optimization.	4-5	2

4.3. Project Consequences and Preemptive Measures

- We are unable to finish a satisfactory product on time
  To ensure that we are able to deliver on what we are planning to do, our bi-weekly review meetings *(Sprint Retrospective)* will aim to identify activities with scopes beyond our time limits and capabilities.
- Major cross platform compatibility issues
  NOX Engine is currently supported on Mac OS X, GNU/Linux and Windows. It is important for Suttung Digital that no additions or changes break this. Our team is working on GNU/Linux and Windows, but Mac OS X is available for asserting the platform compatibility.
- Fruitless optimization
  Knowing the improvements of an optimization prior to implementation is nigh impossible. Many factors, often unknown, come into play, potentially rendering the attempted optimization to have negative effects. This is very likely to occur, and if it does, we will analyze *why*.

4.4. Team rules

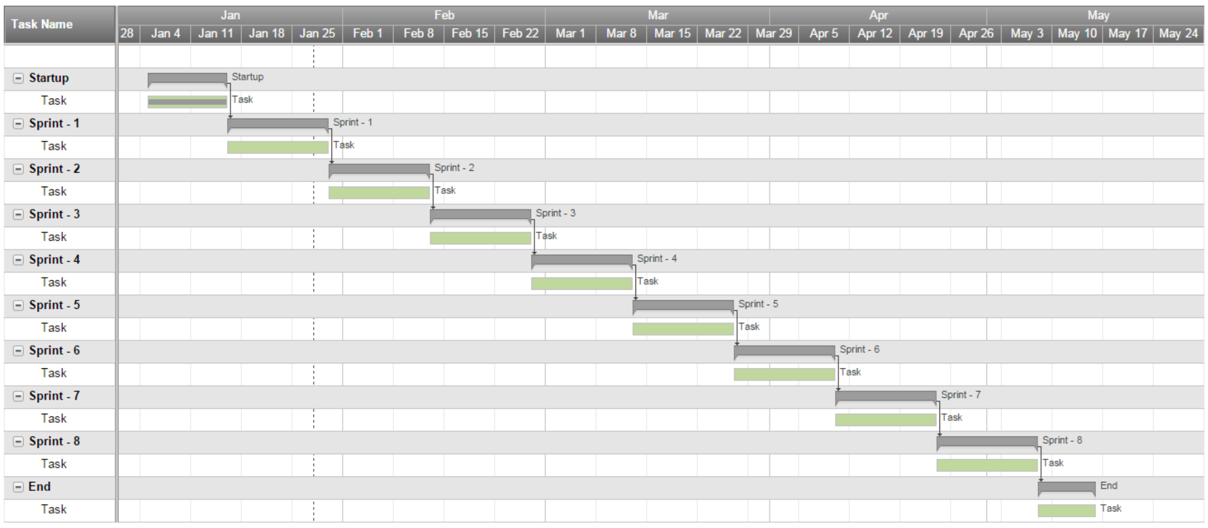
The rules are written in Norwegian. As point 10 states possible, the meeting schedule has been adjusted.

1. Forventede arbeidstimer er 40 timer i uka. Ved ekstraordinære omstendigheter kan dette fravikes. Research, møter og studie av litteratur er inkludert i denne tidsrammen.
2. Meld fra om man ikke kan nå forventede arbeidstimer.
3. Alt arbeid dokumenteres. Både i Confluence, Jira og i timeliste.
   a. Wikien skal ha god struktur.
   b. Alt skal kunne finnes kun ved å trykke på linker. Forsiden skal linke til overordnede elementer, og disse skal igjen inneholde eller linke til underelementer.
4. Kommenter koden.
5. Følg coding convention.
6. Aldri push til master før du er sikker på at det funker (så godt det lar seg gjøre).
7. Bruk branches. Commits skal tydelig forklare hva som er gjort. Mindre commits til samme oppgave bør slås sammen.
8. Vær tilgjengelig på Skype i normal arbeidstid.
9. Vær tilgjengelig til morgenmøte ("sprint stand-up", alle hverdager) på Skype. Tid avtales dagen før.
10. Møt opp på fysisk møte mandag og onsdag 13:00 hver uke og fredag 18:00 på slutten av hver sprint. Dette kan endres ettersom vi endrer arbeidsmetoder.
11. Om gruppen har et problem med noens arbeidsmetoder eller adferd, så tas dette opp i gruppen, og om det ikke rettes annses det som regelbrudd.

12. Om noen konsekvent bryter regler, skal dette tas opp i gruppen og evt. følges opp hos veileder. Det kan til slutt føre til utkasting av gruppen.

5. EXECUTION PLAN

The project will consist of 9 sprints, the first 8 of which are two weeks in length. The final sprint will only be one week in length. Sprints run from Thursdays to Fridays to best integrate with Suttung Digital's schedule.



F Project Agreement

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Suttung Digital / Asbjørn Sporaland (oppdragsgiver), og

JOAKIM NORDSTRAND STIEN
SVERRE BLEKASTAD STENSBY
KRISTOFFER DAHLGREN (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 5/1-15 til 15/5-15.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Mariosz Nowostawski

Oppdragsgivers kontaktperson (navn): Asbjørn Sporaland

Student(er) (signatur): [illegible] dato 27/01-15

Kristoffer Alvin Dahlgren dato 27/01-15

[illegible] dato 27/01-15

_______________ dato _______

Oppdragsgiver (signatur): [illegible] dato 27/01-15

IMT Dekan/prodekan (signatur): _______________ dato _______

Versjon Januar 2011hb

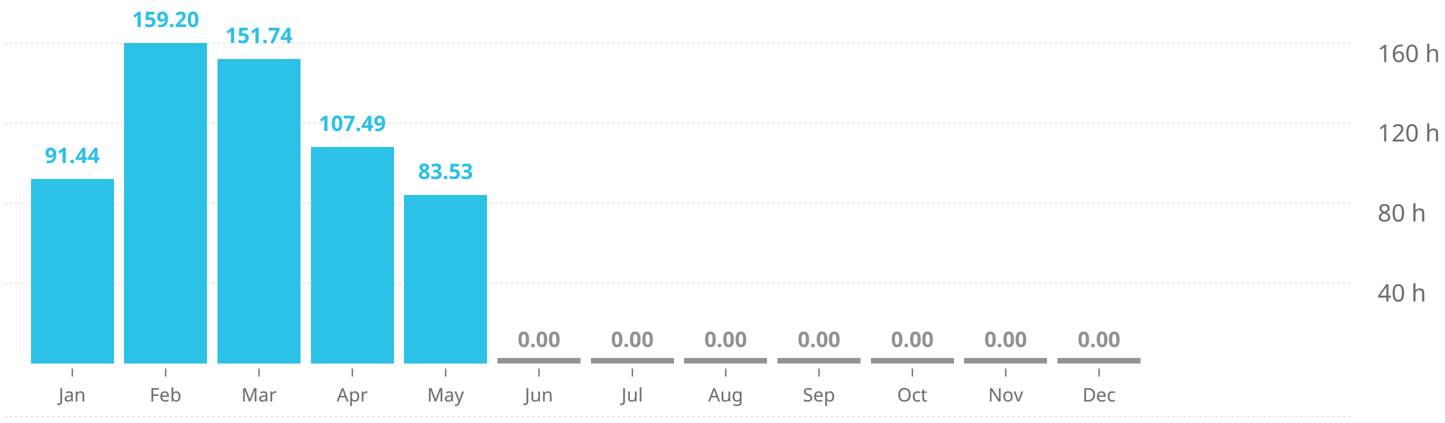
G Time Log

Sverre frequently forgot to track his working hours, so the recorded data is not comprehensive. Joakim’s time logs are comprehensive.

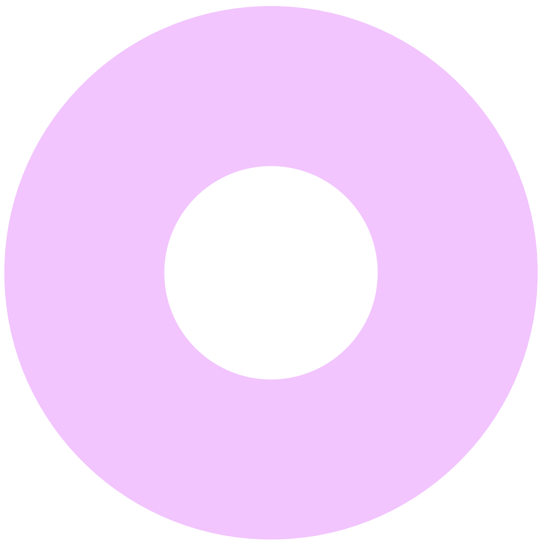
Summary report

2015-01-01 - 2015-12-31
Total 593 h 24 min

Joakim Stien selected as users
Suttung Digital selected as clients

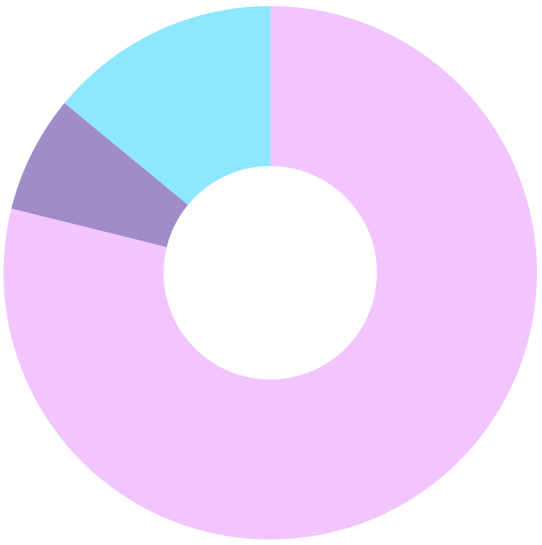


Projects



● NOX - Suttung Digital 593:24:17

Time entries

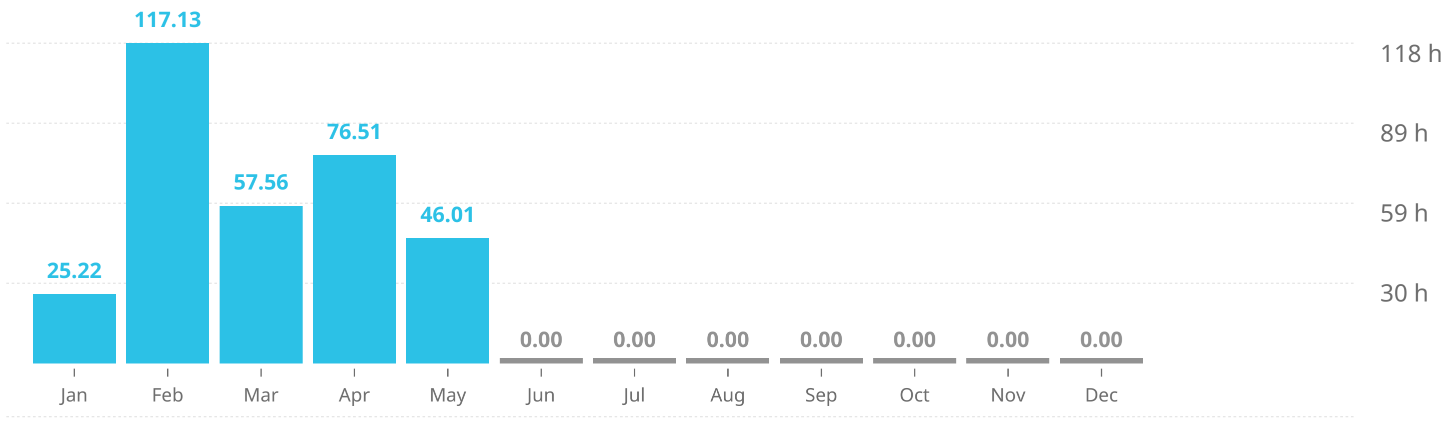


● 468:00:47
● Networking 42:11:14
● Other 83:12:16

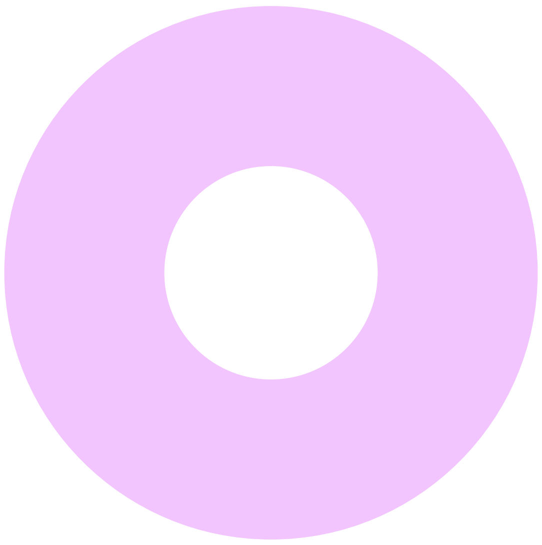
Summary report

2015-01-01 - 2015-12-31
Total 322 h 25 min

Sverrebs selected as users
Suttung Digital selected as clients

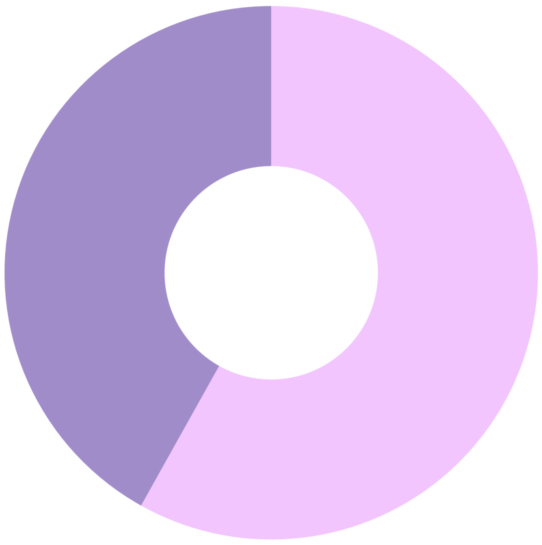


Projects



● NOX - Suttung Digital 322:25:45

Time entries



● 187:22:40
● Other 135:03:05

H Commit Log

```
db9559a Joakim Stien    Wed Jan 21 15:28:43 2015 +0100
Started integration of existing networking code. Moved default output directories of NOX
      library and test executables.

bdf2b30 Joakim Stien    Wed Jan 21 18:58:59 2015 +0100
Improved the Packet class quite a bit. Added test coverage for Packet stream methods

7b8e096 Joakim Stien    Wed Jan 21 19:04:08 2015 +0100
Transitioned added code to the NOX standard

6b6fc78 Joakim Stien    Wed Jan 21 19:20:38 2015 +0100
Integrated ConnectionListener. This includes server socket functionality.

4e23795 Joakim Stien    Thu Jan 22 12:56:59 2015 +0100
[NET] Added IPacketTranslator interface.

570a7c7 Joakim Stien    Thu Jan 22 17:58:07 2015 +0100
Added the skeleton of NetworkController

3a1fc8b Joakim Stien    Thu Jan 22 17:58:44 2015 +0100
Ignoring KDevelop files

84468e4 Joakim Stien    Thu Jan 22 18:01:28 2015 +0100
What savage replaced my spaces with tabs??

7faa946 Joakim Stien    Fri Jan 23 11:45:09 2015 +0100
Renamed NetworkController to CommunicationController

26c5371 Joakim Stien    Fri Jan 23 11:45:24 2015 +0100
Temporary commmit (switching computer)

627808e Joakim Stien    Fri Jan 23 13:09:54 2015 +0100
Another remote-repo-stash. Switching computers again.

cde77e9 Joakim Stien    Fri Jan 23 14:25:30 2015 +0100
Temporary commit

6190534 Joakim Stien    Fri Jan 23 14:27:26 2015 +0100
Merge resolve

505f5b6 Joakim Stien    Fri Jan 23 14:33:18 2015 +0100
Merge resolve

96a785f Joakim Stien    Fri Jan 23 14:39:12 2015 +0100
Moved application/net to app/net

de052ae Joakim Stien    Fri Jan 23 14:44:30 2015 +0100
Successfully merged with SuttungDigital/NOX

ec4c394 Joakim Stien    Fri Jan 23 14:48:55 2015 +0100
Merge resolve

9e9bfb2 Joakim Stien    Fri Jan 23 15:11:29 2015 +0100
REALLY moved application/net to app/net.

d4667f5 Joakim Stien    Fri Jan 23 15:19:07 2015 +0100
```

```
Changed all references to ILogicContext to IContext.

6464263 Joakim Stien    Fri Jan 23 15:19:48 2015 +0100
The test suite now compiles fully.

c57238e Joakim Stien    Fri Jan 23 15:19:56 2015 +0100
Merge branch 'master' into pimms

b718083 Joakim Stien    Fri Jan 23 15:33:33 2015 +0100
Fixed silly errors

6aa00a1 Joakim Stien    Fri Jan 23 15:42:41 2015 +0100
Fixed obscure compiler crash under MSVC.

a4e5a5d Joakim Stien    Fri Jan 23 15:45:08 2015 +0100
Merge branch 'master' into pimms

fd941a2 Joakim Stien    Fri Jan 23 15:45:30 2015 +0100
Merge branch 'pimms' of bitbucket.org:optnox/nox-engine into pimms

0ce00f7 Joakim Stien    Fri Jan 23 18:04:33 2015 +0100
test-net now compiles. Woo-fkn-doo

73dea94 Joakim Stien    Fri Jan 23 18:06:15 2015 +0100
Merge branch 'master' into pimms

49f5af5 Joakim Stien    Sat Jan 24 00:40:11 2015 +0100
Pre-weekend commit. Packet and Socket may need more functionality to better accommodate
    the IConversation implementations.

3ad003b Joakim Stien    Mon Jan 26 16:36:59 2015 +0100
Applied stash from 'pimms' onto 'master'. Trying to clean this up nicely...

9888e01 Joakim Stien    Mon Jan 26 16:37:28 2015 +0100
Applied the correct stash onto the correct branch.

864a1a9 Joakim Stien    Mon Jan 26 16:37:49 2015 +0100
Revert "Applied stash from 'pimms' onto 'master'. Trying to clean this up nicely..."

b8c6bd9 Joakim Stien    Mon Jan 26 18:49:53 2015 +0100
Potentially implemented the setup TCP protocol... poorly. Holy shit this is a huge task.

61e44be Joakim Stien    Mon Jan 26 18:50:08 2015 +0100
Merge branch 'pimms'

c02899c Joakim Stien    Mon Jan 26 18:55:15 2015 +0100
Salvaged the common_convo files from 3ad003b

69387d4 Joakim Stien    Mon Jan 26 19:05:21 2015 +0100
Fixed some compiler errors and warnings

4d364a7 Joakim Stien    Mon Jan 26 19:05:31 2015 +0100
Merge branch 'pimms'

04b1b4a Joakim Stien    Mon Jan 26 19:06:29 2015 +0100
Hot dang, double Ds

5b07676 Joakim Stien    Mon Jan 26 19:06:42 2015 +0100
Merge branch 'pimms'

9a4bcd8 Joakim Stien    Mon Jan 26 21:15:32 2015 +0100
ClientNetworkView is now theoretically, potentially able to join a running server.

0998b93 Joakim Stien    Mon Jan 26 21:26:42 2015 +0100
```

```
Changed the functionality of common_convo; the SocketBase parameter is no longer being
     used. Remove it from IConversation perhaps?

ac67854 Joakim Stien     Mon Jan 26 23:21:38 2015 +0100
The ServerNetworkView may now potentially accept new connections. Testing this jazz is
     the next item on the agenda.

8c7e36e Joakim Stien     Mon Jan 26 23:34:09 2015 +0100
Like the lovely lad that I am, all my commits compile......

02a4f67 Joakim Stien     Mon Jan 26 23:35:57 2015 +0100
Merge branch 'pimms'

f99f968 Joakim Stien     Wed Jan 28 13:01:06 2015 +0100
Added find module for SDL2_net

775e5e2 Joakim Stien     Wed Jan 28 13:10:24 2015 +0100
SDL2_net is now it's own dependency in src/CMakeLists

5a3860e Joakim Stien     Wed Jan 28 13:12:45 2015 +0100
Messaging about the use of SDL2_net

0ff7635 Joakim Stien     Wed Jan 28 13:22:16 2015 +0100
fucks sake

a645eeb Joakim Stien     Wed Jan 28 13:26:45 2015 +0100
Added correct class to CommunicationController::handleIncomingPackets

cb2c176 Joakim Stien     Wed Jan 28 14:27:11 2015 +0100
Added bunch of print's to common_convo. TODO: Use the Logger API

fffc1bf Joakim Stien     Wed Jan 28 15:11:15 2015 +0100
Verbosified common_convos

ba5f461 Joakim Stien     Wed Jan 28 15:24:30 2015 +0100
Added getter for command line arguments in Application.

175ba9a Joakim Stien     Wed Jan 28 16:00:27 2015 +0100
Moved the argv-copying to the top of Application::init(int,char**)

ad1a5dd Joakim Stien     Wed Jan 28 16:01:06 2015 +0100
Common convo packets are now filled and retrieved from in a consequent manner.

258417f Joakim Stien     Fri Jan 30 10:38:01 2015 +0100
Merge branch 'pimms'

54dec7d Joakim Stien     Fri Jan 30 11:03:03 2015 +0100
Now assigning actual client IDs to new clients.

0444ef6 Joakim Stien     Fri Jan 30 11:15:06 2015 +0100
Moved protocol::PacketId to protocol.h

5124e39 Joakim Stien     Fri Jan 30 11:26:24 2015 +0100
Refactored IConversation - gave it a default socket instance and renamed it to
     Conversation.

d5989a3 Joakim Stien     Fri Jan 30 11:43:31 2015 +0100
Packet now inherits from iostream

a5e2325 Joakim Stien     Fri Jan 30 13:33:25 2015 +0100
Revert "Packet now inherits from iostream"

c6cd934 Joakim Stien     Fri Jan 30 13:33:58 2015 +0100
```

```
Added (de)serialization to Packet instances while still maintaining backwards
     compatibility

ff5e8bc Joakim Stien     Fri Jan 30 15:36:14 2015 +0100
Renamed Event::serialize() and Event::deSerialize() to doSerialize and doDeSerialize
     respectively.

d5e5c92 Joakim Stien     Fri Jan 30 16:38:46 2015 +0100
Added ClientStats and some management methods in NetworkView

14f87fa Joakim Stien     Sun Feb 1 19:25:58 2015 +0100
Merged in upstream

508b384 Joakim Stien     Mon Feb 2 13:45:23 2015 +0100
Refactored IDecisionManager to only return the maximum lobby size. Added getters to
     NetworkView for this value.

80e80d7 Joakim Stien     Mon Feb 2 14:00:17 2015 +0100
Added ClientStats to src/CMakeLists.txt.

b1faf0a Sverre B. Stensby    Mon Feb 2 14:47:54 2015 +0100
Added NetworkOutEvent and tests

9d36613 Sverre B. Stensby    Mon Feb 2 15:05:25 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

79d300a Joakim Stien     Mon Feb 2 15:36:02 2015 +0100
initial addition of BroadcastSender

e1bf02d Joakim Stien     Mon Feb 2 15:48:52 2015 +0100
Using constant instead of hard-coded port no for broadcast

d67bb65 Joakim Stien     Mon Feb 2 16:48:54 2015 +0100
Initial addition of BroadcastReceiver

652b601 Joakim Stien     Mon Feb 2 17:44:24 2015 +0100
Fixed some critical errors in BroadcastReceiver

f09b3ea Joakim Stien     Mon Feb 2 18:20:09 2015 +0100
Removed all printf calls, replaced them with proper Logger calls.

c0cf0d6 Joakim Stien     Mon Feb 2 18:37:47 2015 +0100
Fixed the initialization procedure

290f4b7 Joakim Stien     Mon Feb 2 18:51:51 2015 +0100
Added Jimmy-proofing to the ClientNetworkView

fb56426 Sverre B. Stensby    Mon Feb 2 19:31:40 2015 +0100
OPTNOX-19: Added onEventPacketRecieved

33e01e9 Joakim Stien     Mon Feb 2 19:32:23 2015 +0100
Better log handling

80be12c Joakim Stien     Mon Feb 2 19:35:02 2015 +0100
Removed verbosity

eec3a31 Joakim Stien     Mon Feb 2 19:39:12 2015 +0100
Merged master into pimms

dc22421 Joakim Stien     Tue Feb 3 05:29:33 2015 +0100
Added ServerBroadcastEvent, raising it appropriately in BroadcastReceiver

a979ced Joakim Stien     Tue Feb 3 05:39:30 2015 +0100
Merge bitbucket.com:suttungdigital/nox-engine into merge
```

```
57ff357 Joakim Stien    Tue Feb 3 05:40:23 2015 +0100
merge resolution

eca2743 Joakim Stien    Tue Feb 3 05:43:23 2015 +0100
Temporary fix to CMake issues (test/application/net/event does not exist or is empty)

a5c67a5 Joakim Stien    Tue Feb 3 06:25:52 2015 +0100
Now using the Uint32 host IP as the unqiue ID in the map in BroadcastReceiver. The code
     is a mess, and will be carefully refactored when I'm awake.

424eb06 Joakim Stien    Tue Feb 3 11:43:38 2015 +0100
Added 'sourceIp' member to Packet. This will help identifying broadcasting servers.

da60904 Joakim Stien    Tue Feb 3 11:59:44 2015 +0100
Now using the source IP of the broadcaster in ServerConnectionInfo instances hailing
     from BroadcastReceiver

bcf97e0 Joakim Stien    Tue Feb 3 12:23:12 2015 +0100
Now properly setting the source IP of UDP broadcast packets

e9b9807 Joakim Stien    Wed Feb 4 09:14:42 2015 +0100
Added (de)serialization of std::vector<byte> when Events are (de)serialized to std::(i|o
     )streams

5a03125 Joakim Stien    Wed Feb 4 09:17:45 2015 +0100
Now heeding UserData::deSerialize return value in Event::operator>>(UserData&)

edc943b Joakim Stien    Wed Feb 4 09:24:14 2015 +0100
Added (de)serialization interface to/from bool in logic::event::Event

9768aab Joakim Stien    Wed Feb 4 09:25:31 2015 +0100
Added ClientConnectionEvent

f90dc57 Joakim Stien    Wed Feb 4 09:32:52 2015 +0100
Removed all but one warning when compiling under GCC 4.9

f6b66cb Joakim Stien    Wed Feb 4 10:08:35 2015 +0100
 Potentially distributing connection of new clients to the lobby

f53f2d0 Joakim Stien    Wed Feb 4 10:14:01 2015 +0100
Renamed ClientConnectionEvent to ClientConnectedEvent and made it single purpose only

752ba58 Joakim Stien    Wed Feb 4 10:55:07 2015 +0100
Added ClientDisconnectedEvent

c927b51 Joakim Stien    Wed Feb 4 10:55:51 2015 +0100
Added call to superclass in ClientConnectedEvent::do(De)Serialize

1c91884 Joakim Stien    Wed Feb 4 11:08:15 2015 +0100
Now raising ClientDisconnectedEvent from ServerNetworkView when a client drops out.

6455992 Joakim Stien    Wed Feb 4 11:17:24 2015 +0100
Removed IdType parameter to NetworkOutEvent, using static const NetworkOutEvent::ID

77369aa Joakim Stien    Wed Feb 4 11:17:40 2015 +0100
src/CMakeLists.txt now reflects the source tree properly.

6d55480 Joakim Stien    Wed Feb 4 12:18:32 2015 +0100
Added octalIpAddress() function in protocol.(h|cpp). Using this instead of
     SDLNet_ResolveIP to generate the hostname of a discovery broadcast.

dab7a5b Joakim Stien    Wed Feb 4 12:26:28 2015 +0100
Fixed stupid bug in octalIpAddress()
```

```
4118caa Joakim Stien    Wed Feb 4 13:06:46 2015 +0100
Added test for (de)serializing UserData

d3163ad Joakim Stien    Wed Feb 4 13:09:48 2015 +0100
Carefully removed head from bowels

c5f5bdf Joakim Stien    Wed Feb 4 13:57:09 2015 +0100
Setting the ClientId properly in common_convo's UserData.

899b784 Joakim Stien    Wed Feb 4 13:57:21 2015 +0100
Merge branch 'pimms' of https://bitbucket.org/optnox/nox-engine into pimms

ca241fe Joakim Stien    Wed Feb 4 14:19:04 2015 +0100
Now not binding to any host/port in BroadcastReceiver

66655c2 Joakim Stien    Wed Feb 4 15:45:02 2015 +0100
Added method for kicking a client

867a589 Sverre B. Stensby   Thu Feb 5 08:52:15 2015 +0100
OPTNOX-18: Start of work. Needs more work to connect things.

2243880 Sverre B. Stensby   Thu Feb 5 08:57:10 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

7aa6b7c Joakim Stien    Thu Feb 5 14:32:52 2015 +0100
Added nox::util::IniFile for INI format R/W.

a49103b Joakim Stien    Thu Feb 5 16:12:05 2015 +0100
Added octalToIp(std::string) and related tests

5663dcc Joakim Stien    Thu Feb 5 16:12:39 2015 +0100
Added getter for all values in IniFile

d5ef3f0 Joakim Stien    Thu Feb 5 16:36:00 2015 +0100
Initial commit of AccessList. IniFile must be changed to support 64 bit unsigned as well
      as int.

216c19b Joakim Stien    Fri Feb 6 10:52:18 2015 +0100
Replaced 'int' and 'float' in IniFile with 64 bit counterparts, long and double.

ec9784b Joakim Stien    Fri Feb 6 11:23:13 2015 +0100
AccessList implementation appears complete.

b0f1def Joakim Stien    Fri Feb 6 11:58:01 2015 +0100
The AccessList is now the prime decider (next to free space, obv.) when accepting new
     clients

5e6b08a Joakim Stien    Fri Feb 6 12:10:02 2015 +0100
IniFile.cpp now upholds the oath sworn in IniFile.h

2e0aba1 Joakim Stien    Fri Feb 6 13:27:03 2015 +0100
Added banClient() method in ServerNetworkView

8e17995 Joakim Stien    Fri Feb 6 13:28:41 2015 +0100
Merge branch 'pimms' of bitbucket.org:optnox/nox-engine into pimms

99c1bfb Joakim Stien    Fri Feb 6 14:20:44 2015 +0100
Resolved suttungdigital/NOX-Engine issue #21 - Failing shutdown

dcc101c Joakim Stien    Fri Feb 6 15:07:27 2015 +0100
Merged src/CMakeLists.txt

a3cf375 Joakim Stien    Fri Feb 6 15:11:26 2015 +0100
```

```
Merge branch 'merge'

a455e47 Joakim Stien    Fri Feb 6 15:19:06 2015 +0100
Some content got lost in the merge. Re-added it.

80a3367 Joakim Stien    Fri Feb 6 16:49:02 2015 +0100
Added default settings file. Application now always looks for 'settings.ini'. Don't know
      if this should be overrideable.

7826a69 Joakim Stien    Fri Feb 6 17:19:13 2015 +0100
Doing initialization in the correct order.

bbaf44d Joakim Stien    Mon Feb 9 08:33:17 2015 +0100
Moved settings file to logic::IContext

7802471 Joakim Stien    Mon Feb 9 09:07:36 2015 +0100
Merge branch 'master' into merge

514df79 Joakim Stien    Mon Feb 9 09:09:00 2015 +0100
Merge branch 'merge' into pimms

e7e370f Joakim Stien    Mon Feb 9 09:30:36 2015 +0100
Improved the IniFile-usage in logic::Logic

5356228 Joakim Stien    Mon Feb 9 09:32:52 2015 +0100
Keeping jsoncpp up to date

2bfc3dd Joakim Stien    Mon Feb 9 09:39:24 2015 +0100
JoinServerConversation and AcceptClientConversation now detects the ID of the app and it
     's version. Mismatched clients are denied access.

90b28f8 Joakim Stien    Mon Feb 9 09:47:44 2015 +0100
BroadcastSender now uses the logic::IContext-defined appId and versionId.
     BroadcastReceiver ignores incompatible servers.

79a3141 Joakim Stien    Mon Feb 9 09:58:13 2015 +0100
Renamed ServerConnectionInfo::(gameName|version) to (appId|versionId)

03455b0 Joakim Stien    Mon Feb 9 11:14:59 2015 +0100
Added Component* Actor::findActor(std::string) const

51c4868 Joakim Stien    Mon Feb 9 11:26:52 2015 +0100
Added serialize(Packet&) and deSerialize(const Packet&) to Component

192630e Joakim Stien    Mon Feb 9 12:09:06 2015 +0100
Added FieldType::VEC2 and corresponding handlers

b31793d Joakim Stien    Mon Feb 9 12:11:30 2015 +0100
Fixed somme typos :/

8984419 Joakim Stien    Mon Feb 9 12:22:57 2015 +0100
Added Packet::operator<<(Packet) and Packet::operator>>(Packet)

db8eae3 Joakim Stien    Mon Feb 9 12:25:30 2015 +0100
Added setter for outFieldIndex

9d38928 Joakim Stien    Mon Feb 9 12:48:50 2015 +0100
Constified setOutStreamPosition()

15a9632 Joakim Stien    Mon Feb 9 12:50:11 2015 +0100
Added SyncOut-component

53337ff Joakim Stien    Mon Feb 9 12:53:31 2015 +0100
Added serialize(Packet&) and deSerialize(const Packet&) to Transform
```

```
454bdca Joakim Stien     Mon Feb 9 13:12:07 2015 +0100
Added setter for angular velocity in the physics subsystem

8bd0d37 Joakim Stien     Mon Feb 9 13:13:09 2015 +0100
Removed method from SyncOut which clearly belongs in SyncIn

5b79fd8 Joakim Stien     Mon Feb 9 14:55:52 2015 +0100
Fixed stuff under Windows

fb25633 Joakim Stien     Mon Feb 9 14:56:38 2015 +0100
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

f9fbaf3 Joakim Stien     Mon Feb 9 18:31:12 2015 +0100
Resolved OPTNOX-39, retrieving incoming packets and distributing them (indirectly) to
    active Conversations

aa2939b Joakim Stien     Mon Feb 9 18:31:42 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

fc0c19e Sverre B. Stensby    Tue Feb 10 03:33:43 2015 +0100
OPTNOX-18: A lot of connecting done.

80e22a3 Sverre B. Stensby    Tue Feb 10 03:36:24 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

ad44fad Sverre B. Stensby    Tue Feb 10 04:23:54 2015 +0100
bugfix

2450df1 Joakim Stien     Tue Feb 10 12:05:52 2015 +0100
Added SyncIn component

c8d8ba0 Joakim Stien     Tue Feb 10 12:15:29 2015 +0100
Renamed IDecisionManager to ServerDelegate

6227841 Joakim Stien     Tue Feb 10 12:19:45 2015 +0100
Merge branch 'pimms'

357b726 Joakim Stien     Tue Feb 10 12:22:38 2015 +0100
Properly setting JoinServerConversation::done and AcceptClientConversation::done

80ddc74 Joakim Stien     Tue Feb 10 12:22:48 2015 +0100
Merge branch 'pimms'

0c799c6 Joakim Stien     Tue Feb 10 12:27:34 2015 +0100
Now logging informationally when a Join/Accept convo ends

67984da Joakim Stien     Tue Feb 10 12:31:26 2015 +0100
ServerDelegate.cpp is no longer suspicioiusly empty

fa1adda Joakim Stien     Tue Feb 10 12:32:04 2015 +0100
Resolved OPTNOX-43. #underscope

5f97fe8 Joakim Stien     Tue Feb 10 12:33:33 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

6699ed2 Joakim Stien     Tue Feb 10 13:30:39 2015 +0100
Added ServerDelegate::createSynchronizedActor

6925860 Joakim Stien     Tue Feb 10 13:46:36 2015 +0100
Fixed circular dependency between ServerNetworkView and ServerDelegate

6a79cb9 Joakim Stien     Tue Feb 10 13:46:50 2015 +0100
Fixed typo in CreateSynchronized.cpp
```

```
33e092b Joakim Stien     Tue Feb 10 13:59:03 2015 +0100
Setting ServerDelegate::networkView from a position when the context actually exists.

8a0e11f Sverre B. Stensby    Wed Feb 11 04:39:10 2015 +0100
Hipefully finished OPTNOX-18. Needs comments and testing.

d43db64 Joakim Stien     Wed Feb 11 11:11:25 2015 +0100
Parsing synchronized object data in actor::Factory

3e7da04 Joakim Stien     Wed Feb 11 11:26:05 2015 +0100
Parsing of SynchronizedAttr properly overwrites values in the inheritance line.

f93ce7f Joakim Stien     Wed Feb 11 11:40:24 2015 +0100
Small bugfix in Synchronized parsing

d3a477c Joakim Stien     Wed Feb 11 11:40:38 2015 +0100
Merge branch 'pimms'

ee5cebb Joakim Stien     Wed Feb 11 11:59:46 2015 +0100
Aded destroySynchronized(SyncId) and destroyAllSynchronized(ClientId) in ServerDelegate

4d8df85 Joakim Stien     Wed Feb 11 12:04:17 2015 +0100
Added Destroy Synchronized event to Cmake

5ed39b8 Joakim Stien     Wed Feb 11 12:29:39 2015 +0100
Added lambda handlers for Packets (PacketId based) in CommunicationController

cf4b28f Joakim Stien     Wed Feb 11 13:05:12 2015 +0100
Added 5 new events needed for OPTNOX-52

de084fb Joakim Stien     Wed Feb 11 13:05:41 2015 +0100
Raising connectivity events about self from ClientNetworkView

63be025 Joakim Stien     Wed Feb 11 13:27:25 2015 +0100
Resolved OPTNOX-52. The Client now raises events on key connectivity events.

2e07787 Joakim Stien     Wed Feb 11 13:44:23 2015 +0100
Removed ServerController (empty files)

e7d3e3b Sverre B. Stensby    Wed Feb 11 16:56:03 2015 +0100
Merge branch 'master' into sverre

9720cc8 Joakim Stien     Wed Feb 11 19:00:44 2015 +0100
Resolved OPTNOX-47. Synchronized actors are now created when the CreateSynchronized
    event is raised.

ca6ac02 Joakim Stien     Wed Feb 11 19:08:10 2015 +0100
Forgot to call setup(..,..) on ListenerManagers

5e6abba Joakim Stien     Wed Feb 11 19:08:24 2015 +0100
Merge branch 'pimms'

1685140 Sverre B. Stensby    Thu Feb 12 03:50:37 2015 +0100
Bugfix, should now mostly be done

45a2349 Sverre B. Stensby    Thu Feb 12 03:51:23 2015 +0100
Merge branch 'sverre'

136d99d Joakim Stien     Thu Feb 12 12:48:14 2015 +0100
Added NetworkOutEventListeners to ServerNetworkView and ClientNetworkView. Made
    ClientNetworkView broadcast listening optional.

38b333d Joakim Stien     Thu Feb 12 12:55:39 2015 +0100
```

```
Added delayed initialization of broadcast listening in ClientNetworkView

268bcd8 Joakim Stien     Thu Feb 12 13:52:21 2015 +0100
Removed NetworkOutEventListener from XxxxxNetworkView

e32f529 Joakim Stien     Thu Feb 12 14:33:35 2015 +0100
Added UDP handshake to JoinServer/AcceptClient convos

30dc178 Joakim Stien     Thu Feb 12 14:34:16 2015 +0100
Merge branch 'master' into pimms

1ce4e50 Joakim Stien     Thu Feb 12 14:46:26 2015 +0100
Resolved OPTNOX-62. Clients now properly disconnect when the connection is lost.

f73b684 Sverre B. Stensby    Thu Feb 12 17:10:49 2015 +0100
Shortened PacketId. It should now work just great.

a4f19ee Sverre B. Stensby    Thu Feb 12 17:10:54 2015 +0100
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

cd977d4 Joakim Stien     Thu Feb 12 18:12:28 2015 +0100
Added functionality for removing actors safely without breaking the existing API

1914569 Joakim Stien     Thu Feb 12 18:33:56 2015 +0100
Added parsing of and the storing of the Destruction Handler function name in Actor,
     SyncAttributes

eca7656 Joakim Stien     Thu Feb 12 19:41:51 2015 +0100
Removal of actors should now be safe under normal circumstances.

db0f202 Joakim Stien     Thu Feb 12 19:48:48 2015 +0100
Fixed compiler errors

0825968 Sverre B. Stensby    Fri Feb 13 10:42:23 2015 +0100
Forgot to incrementally commit. The client now recieves packets. They are just a bit
     wrong.

cd62ca7 Sverre B. Stensby    Fri Feb 13 10:42:28 2015 +0100
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

9f2bf9a Joakim Stien     Mon Feb 16 13:06:33 2015 +0100
Temporary commit to clear my stash

a1b73c1 Joakim Stien     Mon Feb 16 13:30:16 2015 +0100
Slight refactor

b2e6e44 Joakim Stien     Mon Feb 16 14:06:28 2015 +0100
Almost resolved issue with packetsending

c381532 Joakim Stien     Mon Feb 16 15:40:14 2015 +0100
Added Box2DVisibilityMapper

9af890a Joakim Stien     Tue Feb 17 12:51:26 2015 +0100
Added own creation of light vertices

faacfd3 Joakim Stien     Tue Feb 17 13:01:06 2015 +0100
Using Light::alphaFallOff correctly

fc6b056 Sverre B. Stensby    Wed Feb 18 03:14:51 2015 +0100
Made the code compile on Linux again.

3e75dbb Sverre B. Stensby    Wed Feb 18 04:58:46 2015 +0100
Sending and recieving packets now works just the way it should.
```

```
4ccb0d5 Joakim Stien    Wed Feb 18 10:22:31 2015 +0100
Merge branch 'master' into pimms

3a87d0e Joakim Stien    Wed Feb 18 12:12:31 2015 +0100
[OPTNOX-51] Handling SyncOut and SyncIn in the ComController class hierarchy. Now also
    adding SyncOut component to remotely owned Actors on the server (in RemoteClient).

8c2654f Joakim Stien    Wed Feb 18 12:50:44 2015 +0100
[OPTNOX-58][OPTNOX-64] Resolved 58, removed all NetworkOutEvent usages from
    ServerNetworkView for 64.

5e7a073 Joakim Stien    Wed Feb 18 12:51:40 2015 +0100
Merge branch 'pimms' of bitbucket.org:optnox/nox-engine into pimms

68355d5 Joakim Stien    Wed Feb 18 13:11:40 2015 +0100
Merge resolution

8279e63 Sverre B. Stensby    Wed Feb 18 19:28:39 2015 +0100
OPTNOX-55 done.

3469b3d Sverre B. Stensby    Wed Feb 18 19:28:57 2015 +0100
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

2a2e7cc Sverre B. Stensby    Thu Feb 19 02:50:24 2015 +0100
Replaced Box2d with liquidfun.

861e2fd Joakim Stien    Thu Feb 19 12:01:11 2015 +0100
[OPTNOX-66] Added Event::Event(const nox::app::net::Packet*) constructor which
    deSerializes the event. Improved IPacketTranslator interface.

bd16cbf Joakim Stien    Thu Feb 19 12:01:56 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

3379259 Joakim Stien    Thu Feb 19 12:14:25 2015 +0100
[OPTNOX-63] Renamed all references to NetworkView to NetworkManger.

a2baab3 Joakim Stien    Thu Feb 19 12:28:48 2015 +0100
[OPTNOX-63] Added setters and getters for (Client|Server)NetworkManager

e6eca11 Joakim Stien    Thu Feb 19 12:34:35 2015 +0100
[OPTNOX-63] Updating NewtorkManagers in logic::Logic

d85803b Joakim Stien    Thu Feb 19 12:37:58 2015 +0100
Merge branch 'master' into pimms

47b919f Joakim Stien    Thu Feb 19 16:12:56 2015 +0100
[OPTNOX-65] Added DefaultPacketTranslator. Resolved issue with Event(const Packet*)
    constructor. Moved EVENT_PACKET handling to (Local|Remote)Client.

db34841 Joakim Stien    Thu Feb 19 16:33:39 2015 +0100
Now REALLY not using NetworkOutEvent on the server.

51082af Joakim Stien    Fri Feb 20 10:39:56 2015 +0100
Fixed bug related to SYNC_UPDATE recovery. Made Box2DSimulation::worldModifyMutex std::
    recursive_mutex

b96e1e9 Joakim Stien    Fri Feb 20 11:29:12 2015 +0100
[OPTNOX-84] Now creating existing synchronized actors on new clients

1c1287e Joakim Stien    Fri Feb 20 14:13:53 2015 +0100
resolved issue where flooding of the network made the server quite unresponsive

040ef5d Joakim Stien    Fri Feb 20 14:58:21 2015 +0100
Slight fixes all over the face
```

```
db0cbd1 Joakim Stien     Sat Feb 21 15:25:41 2015 +0100
[OPTNOX-85] REALLY fixed the synchronization handling

8139ace Joakim Stien     Sat Feb 21 15:43:29 2015 +0100
[OPTNOX-90] The DestroySynchronized event is now serialized into a packet (as it should
     be)

dd8658e Joakim Stien     Sat Feb 21 16:12:31 2015 +0100
Fixed bug in world::Manager where removeActor(const actor::Identifier&) would always
     raise a SIGSEGV. This also resolved OPTNOX-90.

a57e14c Joakim Stien     Sat Feb 21 16:28:52 2015 +0100
Added shadow vertex GeometrySet to graphics::Light

9cc7c86 Joakim Stien     Sat Feb 21 16:39:18 2015 +0100
Consequent usage of this->totalLightCoords in OpenGlRenderer

798198a Joakim Stien     Sat Feb 21 16:57:30 2015 +0100
Salvaged performance.(h|cpp)

0c564f9 Joakim Stien     Sat Feb 21 17:32:41 2015 +0100
Added Clock class for measuring elapsed time

fddc9e5 Joakim Stien     Sat Feb 21 17:34:30 2015 +0100
Well I'm  a fkn moron. Publicized the Clock methods.

259ed42 Joakim Stien     Sat Feb 21 19:01:53 2015 +0100
[OPTNOX-9] Added PerformanceMonitor and related classes with API compliance to Evernote
     doc '15.01'

73da648 Joakim Stien     Mon Feb 23 11:47:13 2015 +0100
Replaced all occurrences of four spaces with tab characters

feac767 Joakim Stien     Mon Feb 23 13:02:30 2015 +0100
Now loading normal maps. Like a boss.

5cc7345 Joakim Stien     Mon Feb 23 15:02:35 2015 +0100
Loading specular maps into OpenGlRenderer

aacb28e Joakim Stien     Mon Feb 23 15:08:31 2015 +0100
RenderData now is the definer of glActiveTexture

4cc5f0a Joakim Stien     Mon Feb 23 15:45:26 2015 +0100
The OpenGlRenderer now uses the new RenderData API.

f6bfaaa Joakim Stien     Mon Feb 23 16:15:29 2015 +0100
Added normal and specular uniforms to spriteLightLuminance shader

e4b72b6 Joakim Stien     Mon Feb 23 20:37:50 2015 +0100
Normal mapping ALMOST works!

138b86a Joakim Stien     Mon Feb 23 21:36:36 2015 +0100
90% done with normal maps

fb75390 Joakim Stien     Tue Feb 24 12:53:20 2015 +0100
Poking around, it looks a million times better already.

a3882dc Joakim Stien     Tue Feb 24 13:47:51 2015 +0100
[OPTNOX-91] Added attenuation and Z-height to graphics::Light and ActorLight component.

447d4d9 Joakim Stien     Tue Feb 24 14:19:36 2015 +0100
[OPTNOX-91] The normal maps looks fucking gorgeous now.
```

```
1f62397 Joakim Stien    Wed Feb 25 12:18:43 2015 +0100
Sprite lighting is now calculated based on the distance between the center of light and
    the fragment, normalized to the range [0,100], where 100=(light.range)

e294113 Joakim Stien    Wed Feb 25 12:51:19 2015 +0100
Oops, fixed silly syntax error in SLL.frag

31cd6d9 Joakim Stien    Wed Feb 25 13:24:14 2015 +0100
Added getter methods for app::graphics::Light from logic::graphics::ActorLight. Added
    const setter methods for mutable fields in Light.

dec096a Joakim Stien    Wed Feb 25 14:18:38 2015 +0100
[OPTNOX-93] Resolved bug.

e4b5210 Joakim Stien    Wed Feb 25 14:41:03 2015 +0100
[OPTNOX-92] Fixed stupid issue where off-screen lights would be counted

508f30e Joakim Stien    Wed Feb 25 15:19:04 2015 +0100
Specular fkn maps! Looks gorgeous, etc

2f1a31c Joakim Stien    Wed Feb 25 17:12:11 2015 +0100
Tweaked specular shader

e860fe1 Joakim Stien    Wed Feb 25 17:29:18 2015 +0100
No longer ignoring YCM configs

8f0051b Joakim Stien    Wed Feb 25 17:31:03 2015 +0100
No ignoring YCM compiled config settings

1da0110 Joakim Stien    Thu Feb 26 14:13:43 2015 +0100
Merged in Gachapen/optnox-nox-engine/fix_shader_issues (pull request #1)

9f450f9 Joakim Stien    Thu Feb 26 14:21:34 2015 +0100
Merged in upstream NOX

f791995 Joakim Stien    Thu Feb 26 14:26:17 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox-engine

39a2adf Joakim Stien    Thu Feb 26 14:58:35 2015 +0100
Purging warnings

bd0e8ec Joakim Stien    Fri Feb 27 15:42:36 2015 +0100
Change in YCM config

9f20088 Joakim Stien    Fri Feb 27 15:43:11 2015 +0100
Trying to multithread the graphics

cfcc588 Joakim Stien    Fri Feb 27 16:49:04 2015 +0100
Merge branch 'master' into pimms

9dd6ab5 Joakim Stien    Fri Feb 27 18:39:38 2015 +0100
Now requesting OpenGL 3.0 explicitly

26f2f02 Joakim Stien    Fri Feb 27 19:52:14 2015 +0100
Revert "Trying to multithread the graphics". Will retry later.

dacb079 Joakim Stien    Mon Mar 2 11:12:07 2015 +0100
Merged in master

05cb029 Joakim Stien    Mon Mar 2 11:13:38 2015 +0100
Explicit isystems for third party libraries in YCM config

61acb1a Joakim Stien    Mon Mar 2 15:38:57 2015 +0100
Temporary commit, going to school.
```

```
ac1aa79 Joakim Stien     Mon Mar 2 19:06:27 2015 +0100
Light are now rendered via use of the stencil buffer. phyislcs::Simulation::
     isPointShadowed is likely broken now.

5e56df4 Joakim Stien     Mon Mar 2 19:07:19 2015 +0100
Merged branch pimmslight

872d880 Joakim Stien     Mon Mar 2 19:07:34 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox—engine

0b52e92 Joakim Stien     Mon Mar 2 19:07:36 2015 +0100
Merge branch 'pimms'

9342590 Joakim Stien     Tue Mar 3 08:50:48 2015 +0100
Running light generation on the main thread

c1d710b Joakim Stien     Tue Mar 3 08:50:57 2015 +0100
Merge branch 'pimmslight' of bitbucket.org:optnox/nox—engine into pimmslight

82a97a8 Joakim Stien     Tue Mar 3 10:37:47 2015 +0100
Made light and shadow generation single threaded, and as non—redundant as possible.
     Light geometries are only recreated when one of their geometrical properties (range,
      alphaFallOff, isRadial, coneAngleRadian) has changed.

662ff49 Joakim Stien     Tue Mar 3 10:38:20 2015 +0100
Merge branch 'pimmslight' into pimms

dd1b9de Joakim Stien     Tue Mar 3 10:38:32 2015 +0100
Merge branch 'master' of bitbucket.org:optnox/nox—engine

4d1b67c Joakim Stien     Tue Mar 3 11:12:35 2015 +0100
Fixed graphics::Camera::getBoundingAABB. Now not rendering shadows nor lights that are
     off—screen.

57a802b Joakim Stien     Tue Mar 3 11:24:45 2015 +0100
Changing the range of a light no longer forces the geometry to be recreated

2e4f4ab Joakim Stien     Tue Mar 3 13:28:24 2015 +0100
Shadows are no longer being generated for off—screen lights. However, the shadows are
     generated one frame *after* the light actually appears, due to the ordering of
     update—dispatches i logic::Logic.

0c43595 Joakim Stien     Tue Mar 3 13:45:04 2015 +0100
potentially worthless change in Camera::getBoundingAABB()

9d8c0c3 Joakim Stien     Thu Mar 5 14:56:14 2015 +0100
Initial improvement of lag compensation

728015b Joakim Stien     Thu Mar 5 15:05:39 2015 +0100
Merge branch 'master' into pimms

19d13a4 Joakim Stien     Fri Mar 6 10:55:34 2015 +0100
Merge resolution

65c93e1 Joakim Stien     Fri Mar 6 11:57:18 2015 +0100
No longer applying interpolated states until the remote state has arrived in
     ActorPhysics. Increased UDP update frequency to 20hz

3d9eb4b Joakim Stien     Fri Mar 6 13:34:02 2015 +0100
Added User—Hash to UserData, to identify when the hosting player has connected over
     loopback

288b630 Joakim Stien     Fri Mar 6 14:04:28 2015 +0100
```

```
no longer synchronizing the host over the network, as the game state is shared.

ebbd85f Joakim Stien     Sun Mar 8 14:07:04 2015 +0100
Made all ActorPhysics synchronizable attributes optional via JSON—definitions.

87f6831 Joakim Stien     Sun Mar 8 14:40:16 2015 +0100
Added synchronization to 2dActorDirectionControl

401e9aa Joakim Stien     Sun Mar 8 14:55:52 2015 +0100
Estimating the position with a hard—coded 100ms round—trip—time. on—02 looks — overall —
      pretty damn fine.

2f79a6e Joakim Stien     Fri Feb 27 18:39:38 2015 +0100
Now requesting OpenGL 3.0 explicitly

3d13384 Joakim Stien     Sun Mar 8 18:30:26 2015 +0100
Merge branch 'master' of bitbucket.org:suttungdigital/nox—engine into suttungmaster

5cd04d0 Joakim Stien     Mon Mar 9 08:27:13 2015 +0100
Only explicitly requesting an OpenGL 3.0 context under Linux.

76b4d51 Joakim Stien     Mon Mar 9 08:33:42 2015 +0100
Merge resolution

a92385c Joakim Stien     Mon Mar 9 12:32:10 2015 +0100
Implemented heartbeat and all related functionality

e014141 Joakim Stien     Mon Mar 9 13:32:22 2015 +0100
The ClientNetworkManager is now aware of all connected clients.

63d0b4e Joakim Stien     Mon Mar 9 13:49:39 2015 +0100
Now estimating the age of incoming packets

ab6ef92 Joakim Stien     Mon Mar 9 13:59:53 2015 +0100
Now using the estimated age of the packet in ActorPhysics

61f0e5d Joakim Stien     Tue Mar 10 11:16:12 2015 +0100
Now taking the component channel frequency into account in ActorPhysics interpolation

0883054 Joakim Stien     Thu Mar 12 10:16:02 2015 +0100
Now calculating the difference between the host's clocks

1d67015 Joakim Stien     Thu Mar 12 17:54:08 2015 +0100
Now calculating the time difference properly. Focks sake.

de944aa Joakim Stien     Thu Mar 12 18:29:47 2015 +0100
ClientStats now eases into the new clock difference to lessen the impact of sudden
     spikes

675ea5b Joakim Stien     Fri Mar 13 08:28:21 2015 +0100
Packets now always include the creation time (INT64)

6ca66c6 Joakim Stien     Fri Mar 13 09:44:41 2015 +0100
Packets can now contain serialized Packets recursively

a4edb8d Joakim Stien     Fri Mar 13 10:30:06 2015 +0100
Optimized component—channel packets for packet—level creation time transfer.

b1c585c Joakim Stien     Fri Mar 13 12:08:21 2015 +0100
Now estimating the age of incoming packets more accurately.

5341c2a Joakim Stien     Fri Mar 13 13:13:21 2015 +0100
Improved lag compensation
```

```
9ad9ef5 Joakim Stien     Tue Mar 17 11:08:20 2015 +0100
Fixed some critical issues in the Performance utils

37fbbab Joakim Stien     Tue Mar 17 11:15:07 2015 +0100
Added '--perfdump' cmd flag handling in Application. PerformanceManager now creates
     intermediate directories in the perflog output path

3df25d6 Joakim Stien     Tue Mar 17 14:55:32 2015 +0100
Merged in SuttungDigital/nox-engine

6b184ee Joakim Stien     Wed Mar 18 11:29:58 2015 +0100
Removed all extremities from bodily cavities.

94dd6b5 Joakim Stien     Thu Mar 19 11:00:23 2015 +0100
Lights now only update their shadows if ((a||b)&&c) where (a) the light has moved, (b)
     shadow casters within range has moved, (c) the light casts shadow and is on screen

5cb4e38 Joakim Stien     Thu Mar 19 13:53:45 2015 +0100
Small improvement to light pipeline

c44cbb0 Joakim Stien     Thu Mar 19 14:43:46 2015 +0100
Improved the light pipeline further. Data is now only updated if any changes have *
     actually* happened, and are only uploaded to the GPU if the data was updated.

4c84888 Joakim Stien     Thu Mar 19 15:02:58 2015 +0100
Using a more reliable (and more correct) PTM-ratio in OpenGlRenderer

2910b1f Joakim Stien     Sun Mar 22 11:33:37 2015 +0100
The camera AABB is now calculated properly.

6534ca9 Joakim Stien     Mon Mar 23 09:48:12 2015 +0100
Copied gitignore from master

28fea49 Joakim Stien     Mon Mar 23 12:40:31 2015 +0100
Added CMake Find-module for cAudio (Linux only, barely working ;-) )

51cee32 Joakim Stien     Mon Mar 23 12:41:15 2015 +0100
Added intial, basic cAudio implementation. Currently offers no more features than the
     existing OpenAL implementation.

b6d0265 Joakim Stien     Mon Mar 23 12:54:42 2015 +0100
NOX_USING_CAUDIO or NOX_USING_OPENAL are now #defined from CMakeLists.txt when either is
     enabled.

a3a18c6 Joakim Stien     Mon Mar 23 13:04:56 2015 +0100
Added creation of default audio::System to app::Application

86a2968 Joakim Stien     Mon Mar 23 13:06:32 2015 +0100
Merged in audiorama so the sample suite compiles

48ad435 Joakim Stien     Mon Mar 23 13:14:22 2015 +0100
resource::Descriptor no longer includes the '.' when returning the file extension (prev:
     '.ogg', new: 'ogg')

32e545b Joakim Stien     Tue Mar 24 11:55:15 2015 +0100
Full overhaul of sound-playing API

b6034d3 Joakim Stien     Tue Mar 24 12:10:12 2015 +0100
Added PlaySound event

c1a769c Joakim Stien     Tue Mar 24 12:25:55 2015 +0100
logic::Logic is now listening for logic::audio::PlaySound, and plays a sound if it can
     upon catching one
```

```
5c46a05 Joakim Stien     Tue Mar 24 13:35:35 2015 +0100
Added methods in app::resource to translate Descriptors into usable relative paths

3c80912 Joakim Stien     Tue Mar 24 13:37:12 2015 +0100
Proper use of audio::System::shutdown() in CAudioSystem

d29206f Joakim Stien     Tue Mar 24 13:48:38 2015 +0100
Increased cAudio log level to ERROR

91164b1 Joakim Stien     Tue Mar 24 13:51:11 2015 +0100
Merged in Audiorama

0e23733 Joakim Stien     Tue Mar 24 14:17:31 2015 +0100
added position, velocity, direciton, up-vector, etc to audio interfaces and cAudio
    implementations

6440c1f Joakim Stien     Tue Mar 24 17:10:06 2015 +0100
Added BufferPositionProvider

2b88143 Joakim Stien     Wed Mar 25 14:30:08 2015 +0100
3D audio now actually works. Wow!

a271536 Joakim Stien     Wed Mar 25 14:30:55 2015 +0100
Merge branch 'audiorama'

820fad5 Joakim Stien     Sat Mar 28 12:07:50 2015 +0100
Added ListenerPositionProvider for automatically assigning the listener position

fc4262f Joakim Stien     Sat Mar 28 12:24:41 2015 +0100
graphics::Camera is now an audio::ListenerPositionProvider, and provides sound positions
    . Sounds should be integrated into 11-control to implement and assert up-vector
    assignment.

2e2139b Joakim Stien     Sat Mar 28 18:22:43 2015 +0100
Moved logic::audio::PlaySound handling to RenderSdlWindow

525a6ab Joakim Stien     Sat Mar 28 19:12:14 2015 +0100
Removed old, outdated and unintended code from Transform.h

a2d8aaa Joakim Stien     Sat Mar 28 19:12:52 2015 +0100
Added 'loop' attribute to PlaySound event

3e25a15 Joakim Stien     Sat Mar 28 19:13:18 2015 +0100
The Camera is now taking the rotation into account when assigning it's transform to the
    audio-listener

c9b1fe4 Joakim Stien     Sat Mar 28 19:13:44 2015 +0100
window::RenderSdlWindowView now takes PlaySound::loop into account when creating an
    event-initiated sound.

5891be8 Joakim Stien     Sat Mar 28 19:16:41 2015 +0100
Undid some stupid doings

900120a Joakim Stien     Mon Mar 30 12:30:25 2015 +0200
Moved renderer, camera and rootSceneNode responsibilities into RenderSdlWindowView

79658a0 Joakim Stien     Mon Mar 30 13:19:23 2015 +0200
Moved net/ subpackage to logic/ from app/

fca0a2a Joakim Stien     Mon Mar 30 13:23:08 2015 +0200
[Net] Changed all header guards and includes referring to <net> as an <app> subpackage
    to refer to it as a <logic> subpackage

f3450b9 Joakim Stien     Mon Mar 30 13:32:25 2015 +0200
```

```
[Net] Changed all references to app::net to logic::net

e4121b1 Joakim Stien    Mon Mar 30 15:31:55 2015 +0200
Merged Audiorama into Master

f0862e1 Joakim Stien    Mon Mar 30 15:52:02 2015 +0200
[Audiorama] Manual merge fixes

b26f9d8 Joakim Stien    Mon Mar 30 16:02:03 2015 +0200
Merged in suttung/master

c398c89 Joakim Stien    Tue Mar 31 13:30:07 2015 +0200
Moved net source-group to the logic/ supergroup

2cb6fbb Joakim Stien    Tue Mar 31 15:30:13 2015 +0200
Resolved all resolvable warnings (cAudio is doing some funky jazz though)

4d73517 Joakim Stien    Tue Mar 31 16:25:11 2015 +0200
Renamed LagInterpolator to Interpolator

0fe02b8 Joakim Stien    Mon Apr 6 08:09:09 2015 +0200
The camera is now listening 'in the right direction'.

0f319ec Joakim Stien    Mon Apr 6 09:36:09 2015 +0200
cAudio is now included as a submodule of Nox

699196a Joakim Stien    Mon Apr 6 09:44:19 2015 +0200
[cAudio] Staying up to date with pimms/cAudio

a1edf34 Joakim Stien    Mon Apr 6 11:48:51 2015 +0200
[MacOSX] Mac OSX compatibility changes

2b7429e Joakim Stien    Mon Apr 6 12:21:06 2015 +0200
Consistently using uintXX_t in critical networking applications. Removed typedef 'byte'
    (was unsigned char).

9a919f1 Joakim Stien    Mon Apr 6 14:50:04 2015 +0200
Compatibility with OSX and uintXX_t

bda0498 Joakim Stien    Mon Apr 6 15:40:31 2015 +0200
Staying up to date with cAudio

b51699b Joakim Stien    Mon Apr 6 14:50:11 2015 +0200
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

5a893df Joakim Stien    Mon Apr 6 15:57:07 2015 +0200
Staying up to date with cAudio

cb3863b Joakim Stien    Mon Apr 6 16:07:24 2015 +0200
Added license header to all un-licensed file

5541ada Joakim Stien    Mon Apr 6 17:38:56 2015 +0200
[Merge] Network, audio documentation & network cleanup

fca45fb Joakim Stien    Mon Apr 6 17:44:32 2015 +0200
[Merge] Moved network unit-tests to the proper logic/ subdir

acb5974 Joakim Stien    Mon Apr 6 17:59:51 2015 +0200
Revert "Replaced Box2d with liquidfun."

e2a4e78 Joakim Stien    Mon Apr 6 18:04:41 2015 +0200
Reverted silly change in usage of b2World::Step(...)

ac491f1 Joakim Stien    Tue Apr 7 09:36:55 2015 +0200
```

```
Merge branch 'audiorama' into easyapi

e3f8dc3 Joakim Stien    Tue Apr 7 09:47:36 2015 +0200
Merged in audiorama

4201ad6 Joakim Stien    Mon Apr 6 09:36:09 2015 +0200
cAudio is now included as a submodule of Nox

56c6dad Joakim Stien    Tue Apr 7 09:57:13 2015 +0200
Merge branch 'audiorama' into easyapi

37e167e Joakim Stien    Wed Jan 28 15:24:30 2015 +0100
Added getter for command line arguments in Application.

27dc119 Joakim Stien    Tue Apr 7 13:21:57 2015 +0200
Added GameController, GameWindowView and omniscient 'nox.h'.

e31695a Joakim Stien    Wed Apr 8 10:39:47 2015 +0200
added the missing GameApplication class. Things appear to be fully functional.

cc3bfe8 Joakim Stien    Wed Apr 8 12:35:23 2015 +0200
Made actor::Component::initialize(..) void instead of bool. Added interface for creating
      empty actors and creating components directlly from an Actor instance

809ddee Sverre Blekastad Stensby    Fri Apr 10 18:50:34 2015 +0200
Revert "[Merge] Moved network unit-tests to the proper logic/ subdir"

7b2d1dc Sverre Blekastad Stensby    Fri Apr 10 18:51:15 2015 +0200
Revert "Revert "[Merge] Moved network unit-tests to the proper logic/ subdir""

df5d3f9 Sverre Blekastad Stensby    Fri Apr 10 18:51:40 2015 +0200
Revert "Revert "Replaced Box2d with liquidfun.""

aae1462 Sverre Blekastad Stensby    Fri Apr 10 19:32:45 2015 +0200
Windows comaptibility changes

9a2d5e3 Sverre Blekastad Stensby    Fri Apr 10 20:38:20 2015 +0200
Merge commit 'aae1462'

e98abc4 Joakim Stien    Mon Apr 13 11:49:42 2015 +0200
Appliction arguments are now placed into nox::app::Application::appArgs

89d531c Joakim Stien    Tue Apr 14 12:42:39 2015 +0200
Up to date with cAudio

9fa89ff Joakim Stien    Tue Apr 14 13:16:42 2015 +0200
Inlined all namespaces bar 'nox'

b068278 Joakim Stien    Tue Apr 14 13:19:55 2015 +0200
Inlined all namespaces bar 'nox' in cpp files

11cec56 Joakim Stien    Tue Apr 14 13:40:51 2015 +0200
Renamed logic::IContext to logic::ILogicContext

7111815 Joakim Stien    Tue Apr 14 13:54:54 2015 +0200
Cleaned up app::graphics::Light. Un-constified the reference to app::graphics::Light
     from LightRenderNode and ActorLight.

02618c6 Joakim Stien    Tue Apr 14 14:03:18 2015 +0200
Renamed app::IContext to app::IApplicationContext

531a548 Joakim Stien    Tue Apr 14 14:12:29 2015 +0200
Updated nox.h and unit tests to use ILogicContext and IApplicationContext
```

```
df7c29d Joakim Stien     Thu Apr 16 12:10:21 2015 +0200
Removed IResrouceAcces::getPathToResource and Provider::getResourcePath(Descriptor&)

b250591 Joakim Stien     Wed Apr 15 21:11:04 2015 +0200
StencilTiledTextureRenderer now works with stencil lights

0c67130 Joakim Stien     Wed Apr 15 19:55:30 2015 +0200
Readded ycm_extra_conf

71d3083 Joakim Stien     Tue Apr 14 09:38:26 2015 +0200
CAudioBuffer now plays sounds via IAudioManager::createFromRaw

d9d78b5 Joakim Stien     Tue Apr 14 08:36:03 2015 +0200
Verbose floating points

556601c Joakim Stien     Tue Apr 14 08:27:53 2015 +0200
Removed Box2D include from ActorPhysics.cpp!

74d2f9b Joakim Stien     Tue Apr 14 08:27:13 2015 +0200
Removed the possibility to loop tracks from PlaySound event

ca20109 Joakim Stien     Tue Apr 14 08:22:25 2015 +0200
Removed Transform::connectedSoundBuffers

9504538 Joakim Stien     Mon Apr 13 12:02:00 2015 +0200
Derp. Missing word in documentation.

27b351c Joakim Stien     Mon Apr 13 11:47:23 2015 +0200
Updated documentation of audio::Buffer::initialize(...)

001b9e7 Joakim Stien     Mon Apr 13 12:34:01 2015 +0200
Updated TestLog to reflect the changes made to the LogLevel enumerations

436a6a2 Joakim Stien     Thu Apr 16 12:30:49 2015 +0200
resource::Descriptor is no longer dependent on boost::filesystem to find the file
     extension

7a10755 Joakim Stien     Thu Apr 16 12:35:23 2015 +0200
Removed unused forward declarations from app::Application

949c94a Joakim Stien     Thu Apr 16 12:52:04 2015 +0200
CMakeLists.txt now disabled OpenAL and enables cAudio if both are disabled or enabled

7069ed8 Joakim Stien     Thu Apr 16 17:56:37 2015 +0200
Added AccumulationType::AVG_FRAME to util::perf

def26c6 Sverre B. Stensby    Mon Apr 27 12:49:53 2015 +0200
Started managing NetworkOutEvents again

509ed5e Joakim Stien     Mon Apr 27 12:50:59 2015 +0200
Added interfaces to ServerDelegate to broadcast events to the lobby. Properly ignoring
     system events in RemoteClient.

df52ed4 Sverre B. Stensby    Mon Apr 27 12:53:16 2015 +0200
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

f115308 Sverre B. Stensby    Mon Apr 27 12:59:49 2015 +0200
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine

63392a2 Joakim Stien     Mon Apr 27 13:07:25 2015 +0200
Added nullptr check in CNM::onEvent

c4b3df5 Joakim Stien     Mon Apr 27 13:09:14 2015 +0200
ServerDelegate::broadcast*** method suite now takes a TLProtocol
```

```
055e7f8 Joakim Stien     Tue Apr 28 12:50:09 2015 +0200
Fixed issue when sending large packets over TCP

d4b279c Joakim Stien     Tue Apr 28 12:56:22 2015 +0200
UDP is now completely forbidden from ever sending more than MAX_UDP_PKT_LEN bytes in a
     single packet.

79bb477 Joakim Stien     Tue Apr 28 13:02:49 2015 +0200
updated documentation

5323be5 Joakim Stien     Wed Apr 29 14:58:29 2015 +0200
Removed box2d library

6001f50 Joakim Stien     Wed Apr 29 16:00:55 2015 +0200
SoundExtraData::frequency was duplication of SED::samplesPerSec. Removed ::frequency.

a291e5f Joakim Stien     Wed Apr 29 16:04:51 2015 +0200
Removed debug printf from ClientStats

c1bd340 Joakim Stien     Sun May 3 12:02:16 2015 +0200
Added getter for logic::IContext in Actor

6cacba6 Joakim Stien     Sun May 3 13:28:25 2015 +0200
[Audio] Added AudioListenerCamera and ActorAudio

b4b2bc7 Joakim Stien     Sun May 3 13:28:25 2015 +0200
[Audio] Added AudioListenerCamera and ActorAudio

79a4dcc Joakim Stien     Sun May 3 15:38:18 2015 +0200
Merge branch 'master' into easyapi

22c6545 Sverre B. Stensby    Fri Feb 20 05:34:24 2015 +0000
started prelim work on OPTNOX-77

9e7509a Sverre B. Stensby    Tue Feb 24 09:46:15 2015 +0100
Added structure and loading of particle system.

721d2ad Sverre B. Stensby    Tue Feb 24 10:04:33 2015 +0100
Adding particle attributes.

67c6161 Sverre B. Stensby    Wed Feb 25 05:41:22 2015 +0100
Added Test. Does not pass. Parsing needs work.

659198b Sverre B. Stensby    Wed Feb 25 05:58:33 2015 +0100
Test now passes.

04b4a26 Sverre B. Stensby    Thu Feb 26 09:45:56 2015 +0100
More work on OPTNOX-80

587c46f Sverre B. Stensby    Thu Feb 26 10:17:05 2015 +0100
OPTNOX-80 Terminated.

abb2312 Sverre B. Stensby    Fri Mar 6 04:34:36 2015 +0100
Rendering particles almost works.

d7dacce Joakim Stien     Mon Mar 9 15:54:47 2015 +0100
Renamed FireParticleRenderer to TextureParticleRenderer. Created new render step for
     particles.

08f6cc1 Joakim Stien     Mon Mar 9 17:33:48 2015 +0100
Changed the particle data flow into the IRenderer-class.

6cff04c Joakim Stien     Tue Mar 10 11:20:34 2015 +0100
```

```
ParticleRenderNode wrapped in the warm embrace of nox::app::graphics.

66528db Joakim Stien    Tue Mar 10 11:40:05 2015 +0100
ParticleEvent now provides the data expected in IRenderer

8321ce6 Joakim Stien    Tue Mar 10 11:42:43 2015 +0100
Moved unit tests to the proper location

bdbfbc6 Joakim Stien    Wed Mar 11 09:34:59 2015 +0100
Renamed Fire Particle shaders to TextureParticle

97227e6 Joakim Stien    Wed Mar 11 09:36:22 2015 +0100
Renamed textureParticleLight to particleLight

1cd40d7 Joakim Stien    Wed Mar 11 10:37:12 2015 +0100
*some* particle data is being drawn through the use of nasty paths and weird stuff

d12c65e Joakim Stien    Wed Mar 11 10:38:08 2015 +0100
Moved the ParticleEvent dispatch in ParticleEmitter to update()

f243071 Joakim Stien    Wed Mar 11 14:40:48 2015 +0100
Temporary commit so Sverre can see my swizzy

b0a52f1 Joakim Stien    Wed Mar 11 15:46:47 2015 +0100
Rendering of particles almost works

ed374d5 Sverre B. Stensby   Thu Mar 12 02:37:32 2015 +0100
ParticleEmitters now spawn according to position of object.

ccefe62 Sverre B. Stensby   Thu Mar 12 03:57:55 2015 +0100
ParticleEmitters now take rotation into account for position.

da911ff Sverre B. Stensby   Thu Mar 12 04:03:14 2015 +0100
ParticleEmitters now take rotation into account for velocity.

95fbcba Joakim Stien    Thu Mar 12 09:23:53 2015 +0100
Temporary commit

8990468 Joakim Stien    Mon Mar 16 09:07:29 2015 +0100
Fixed include guards

f5438f6 Sverre B. Stensby   Wed Mar 18 05:10:20 2015 +0100
ParticleEmitters now push things into separate particle systems.

7f256c2 Sverre B. Stensby   Thu Mar 19 04:06:48 2015 +0100
bugfix

cb7da98 Sverre B. Stensby   Thu Mar 19 08:20:49 2015 +0100
Added particle flags

136fe1c Sverre B. Stensby   Tue Mar 24 04:03:36 2015 +0100
Added RandomType.

bc395f7 Sverre B. Stensby   Tue Mar 24 04:11:43 2015 +0100
Cleanup

fa7d4a9 Sverre B. Stensby   Tue Mar 24 04:30:33 2015 +0100
Added randomness to spawn locations.

649fcbb Sverre B. Stensby   Tue Mar 24 14:24:45 2015 +0100
Added randomness to more things in particle system

d2f30f7 Sverre B. Stensby   Thu Mar 26 02:35:04 2015 +0100
Added hex translator with tests, and a test-all build target that runs all tests.
```

```
6a37221 Sverre B. Stensby   Thu Mar 26 05:13:29 2015 +0100
Made size of particles 1/10th original size.

04ac307 Sverre B. Stensby   Tue Apr 7 02:55:20 2015 +0200
This should work now

881e173 Sverre B. Stensby   Tue Apr 7 12:58:10 2015 +0200
It's working! It's working! [insert podracer sounds]

3a14e04 Sverre B. Stensby   Wed Apr 8 12:34:39 2015 +0200
Simulation now sends particle updates to the renderer. Should now only happen once per
    update (look into only once per frame).

19dfbad Joakim Stien    Fri Apr 10 14:08:25 2015 +0200
Removed light rendering from ParticleRenderer

130e5f8 Joakim Stien    Fri Apr 10 14:11:46 2015 +0200
Added texture and shader names to ParticleEvent

59a5376 Joakim Stien    Fri Apr 10 14:17:22 2015 +0200
Removed CreationData and other invalid attributes of ParticleRenderer

089b836 Joakim Stien    Fri Apr 10 14:21:19 2015 +0200
Made OpenGlRenderer::particleRenderer stack allocated (non ptr)

82e7169 Joakim Stien    Fri Apr 10 14:28:05 2015 +0200
Partial revert of e56200b

4447009 Joakim Stien    Fri Apr 10 22:41:24 2015 +0200
Removed customizable shader from ParticleRenderer

2ec04ee Joakim Stien    Sun Apr 12 15:10:27 2015 +0200
[OPTNOX-143] Using IResourceAccess to retrieve shader contents for ParticleRenderer

87c912f Joakim Stien    Sun Apr 12 16:25:17 2015 +0200
Now using ParticleRenderBatch in ParticleRender to aggregate particles using the same
    texture into the same draw call

1b284e2 Joakim Stien    Sun Apr 12 18:55:24 2015 +0200
[OPTNOX-145] Textured particles are now supported.

09b2cc1 Sverre B. Stensby   Tue Apr 14 12:08:51 2015 +0200
Added color back into the particle system.

6d9e60b Sverre B. Stensby   Wed Apr 15 10:04:09 2015 +0200
zoom now works on particles

76c486a Sverre B. Stensby   Wed Apr 15 10:04:24 2015 +0200
zoom now works on particles

dc1c54f Sverre B. Stensby   Mon May 4 04:55:22 2015 +0200
merged to fit refactoring.

dde655a Joakim Stien    Mon May 4 11:50:34 2015 +0200
Updated entire repo to EasyAPI conventions

cfa43aa Joakim Stien    Mon May 4 12:27:45 2015 +0200
Manual merge of accidentally unmerged feature

85f0a4a Sverre B. Stensby   Sun May 3 18:31:55 2015 +0200
ActorAudio only updates buffers when there is a transform change.

39ab834 Joakim Stien    Mon May 4 12:29:37 2015 +0200
```

```
Merge branch 'master' into easyapi

be61a61 Joakim Stien    Mon May 4 12:30:09 2015 +0200
Removed ActorAudio::onUpdate

c0dc9bc Joakim Stien    Mon May 4 12:36:35 2015 +0200
Merge branch 'master' into easyapi

fbe4ec6 Joakim Stien    Mon May 4 12:57:55 2015 +0200
Re-merged the audio subsystem

500dce6 Joakim Stien    Mon May 4 13:32:49 2015 +0200
RenderSdlWindowView now uses AudioListenerCamera by default

019700c Joakim Stien    Mon May 4 14:52:59 2015 +0200
Merge branch 'suttungmaster'

b89a954 Joakim Stien    Mon May 4 14:58:30 2015 +0200
Compatibility changes for merge

6dd27c2 Joakim Stien    Mon May 4 14:59:00 2015 +0200
Merge branch 'master'

9e6a714 Joakim Stien    Mon May 4 15:00:39 2015 +0200
Merge branch 'master' of bitbucket.org:optnox/nox-engine

c217c57 Joakim Stien    Mon May 4 15:39:12 2015 +0200
Merge branch 'master' into easyapi

c445488 Joakim Stien    Mon May 4 16:22:41 2015 +0200
ActorSprite now work when initiated from code *and* JSON

2a8a143 Joakim Stien    Mon May 4 17:13:37 2015 +0200
Added interfaces for handling SDL_Event in the Game Layer

0403b9a Joakim Stien    Mon May 4 17:47:04 2015 +0200
Gave GameWindowView the functionality to handle control of the Controlled Actor

892b329 Sverre B. Stensby   Tue May 5 01:50:07 2015 +0200
Minor change. Get list of emitters from component

85b21b3 Sverre B. Stensby   Tue May 5 01:51:56 2015 +0200
Merge branch 'master' of https://bitbucket.org/optnox/nox-engine into particle_merge

1036b0d Sverre B. Stensby   Thu May 7 09:51:59 2015 +0200
fixed cmake

26f3a26 Sverre B. Stensby   Thu May 7 09:52:48 2015 +0200
moved position and component update to onComponentEvent

d99766e Sverre B. Stensby   Fri May 8 08:33:54 2015 +0200
Removed unused particle renderers, moved renderer from opengl/ to 2d/

84a7527 Sverre B. Stensby   Fri May 8 18:11:06 2015 +0200
Merge branch 'particle_merge'

4c572ae Joakim Stien    Fri May 8 21:08:16 2015 +0200
Added support for constant gravity

b731684 Joakim Stien    Fri May 8 21:09:24 2015 +0200
Merge branch 'master' of bitbucket.org:optnox/nox-engine

25d85e7 Joakim Stien    Fri May 8 21:17:43 2015 +0200
Fixed some compiler errors from gravity patch
```

```
301ab5a Joakim Stien     Sun May 10 18:17:58 2015 +0200
Added ServerDelegate::onUpdate

2a0684a Joakim Stien     Mon May 11 15:31:07 2015 +0200
Reverted inlined namespaces

0ce56ce Joakim Stien     Mon May 11 15:30:32 2015 +0100
Windows compatibility changes

43899d2 Joakim Stien     Mon May 11 18:27:02 2015 +0200
Fixed problem with invalid packet estimation. Heartbeats are always started once every
    second.
```

appendix/commitlog.txt