NTNU
Norwegian University of
Science and Technology

WORKING TITLE: NOX ECS, A multi-threaded ECS

Author(s)
Per-Morten Straume
Trond Lohne

Bachelor in Game Programmering
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

16.05.2017

Supervisor(s) Mariusz Nowostawski

Sammendrag av Bacheloroppgaven

Tittel:	**TODO: Norwegian title.**
Dato:	16.05.2017
Deltakere:	Per-Morten Straume Trond Lohne
Veiledere:	Mariusz Nowostawski
Oppdragsgiver:	Suttung Digital
Kontaktperson:	TODO: Name, Mail, Phone
Nøkkelord:	TODO: Norwegian Keywords.
Antall sider:	26
Antall vedlegg:	
Tilgjengelighet:	Åpen

Sammendrag: TODO: Write short description of project in norwegian

Summary of Graduate Project

Title:	**WORKING TITLE: NOX ECS, A multi-threaded ECS**
Date:	16.05.2017
Authors:	Per-Morten Straume Trond Lohne
Supervisor:	Mariusz Nowostawski
Employer:	Suttung Digital
Contact Person:	TODO: Name, Mail, Phone
Keywords:	ECS, Entity Component System, Performance, NOX, IMT
Pages:	26
Attachments:	
Availability:	Open

Abstract: TODO: This is the short description of a bachelor thesis. It should contain a short introduction to the area of the thesis and what the thesis contributes to that area.

Preface

Contents

List of Figures

List of Tables

High level overview of the document. Everything written here will obviously be explained further in the proper thesis, this is just to try and give a general skeleton. Section and chapter names might not be exactly the same either, however the flow will be like this outline.

We have chosen to split the document a bit more to have a larger difference on what that is related to the benchmark results etc, and how we worked. Up until the process chapter, we are not really talking anything about how we worked and planned, but rather what we are researching and the results of our implementation. The process chapter is where we start talking about how we were doing planning, scrum models, changes during development etc etc.

1 Introduction

1.1 Game Industry

Short high level description of the issues faced within the games industry: fast iteration cycles, performance.

1.1.1 Fast Iteration Cycles

Why does the game industry need fast iteration cycles? Flexible design, need to test ideas quick etc.

1.1.2 Performance

Why does the game industry value performance? Always pushing the envelope for graphics etc.

1.1.3 Flexibility

Why does the game industry value flexible systems? Reuse existing stuff.

1.2 Entity Component Systems

High level overview over what entity component systems are, and why they are used in the games industry.

1.3 Introduction To Suttung

Introduction to Suttung, who is our stakeholder.

1.4 Introduction To NOX Engine

Short introduction to the NOX engine, which is what we will interface against.

1.5 High-Level Objective

Create an Entity Component System within Suttungs engine, based on the requirements laid down by Suttung.

1.6 Organization

How is this document organized. How we organize ourselves, process etc will be put later in the document.

1.7 Terminology

A small section explaining what the different terms used within the report means.

2 Requirements

2.1 Templates

Explain why we don't want heavy template usage within our system. To avoid heavy compile times within our ecs.

2.2 Performance

Discuss quickly why performance is important for NOX.

2.2.1 Virtual Functions

Discuss how virtual often are implemented, and why that layer of indirection is not necessarily good for modern platforms.

2.2.2 Contiguous Memory

Discuss why contiguous memory is important on modern platforms.

2.2.3 Multi-threading

Discuss why nox would like multi-threading within the new ecs.

2.3 Memory Usage

Discuss how we wanted to keep the memory usage about the same.

2.4 Mobile Platform

Discuss how mobile platforms often aren't x86 architectures, and what that affects. Mention both memory models, as well as the use of memory.

2.5 Familiarity

Discuss the requirement from NOX of familiarity, that the new system should not in a great deal different than the one they already have.

2.6 Focus On Safety

Discuss how nox would prefer us to do stuff safely but performance pessimistic by default.

2.7 Final Problem Statement And Objective

Here we detail the actual problem statement and objective based on the requirements listed above.

3 Measurements

Discuss how we will measure the requirements. (Of those who are measurable).

3.1 Identifying Metrics

Talk about which metrics we use, and why we choose those metrics. What are the problems with those metrics. - Examples: We can't control the entire environment, so counting CPU cycles might not be entirely correct because we might have context switched.

3.2 Test Cases

Describe the different testcases we have created. Each test case will have its own subsection. In each subsection we will argue for why the test is relevant, and also mention weaknesses. Also need to refer to our test case plan.

4 Technical

This chapter will probably be quite a substantial part of the thesis, as we need to motivate everything. Also refer to the technical specification and design document.

4.1 High Level Architecture

Discusses the architecture at a higher level, only mention the different components and how they are tied together, however we are not motivating the different low level choices.

4.1.1 Lifecycle

Why are we doing the lifecycle we chose, and how does it work?

4.1.2 Parallelism

Why did we chose the fork-and-join model, and why do we do the layering approach. Also how does the layering approach work. Mention the problems with parallelizing Entity Models. Look at the Game Engine Architecture book.

4.1.3 Event System

Why did we choose this event system, rather than the one that is used within NOX.

4.1.4 Choices

Find better title Talk about what high level choices and restrictions we put down. Like how we only allow movable types, what motivated these choices? The only movable types was motivated by seeing their usage patterns, and how much they used unique_ptr etc, which is only movable. However what was the consequence of this choice? We could not implement state caching.

4.2 Detailed Architecture

All different modules from the High Level Architecture will here be discussed in a specific format. These are the modules we are thinking of discussing, at they are where we take more "interesting decisions". Each detailed description can probably also refer to the implementation which we can include in the assignment.

- ComponentCollection
- EntityManager
- MetaInformation
- TypeIdentifier
- OperationTypes
- Component
- SmartHandle
- Execution Layers
- Allocators
- Thread Pool

- Event

Format:

4.2.1 Module

Explanation

Explain at a more fine level how the module functions.

Motivation

Explain why we made the choices we made.

Alternatives

Quickly list up potential alternatives to this choice.

Pros

What are the benefits of this approach.

Cons

What do we loose with this approach.

Consequences

What; if any, were the consequences of this choice?

4.2.2 Short Example (Obviously more detailed actual thesis): OperationTypes

Explanation

Function pointers to different parts of a components lifecycle.

Motivation

Allows the user to both work with a member function mindset, and the availability to work with a data oriented mindset, i.e. make use of patterns in the data that the user knows about. Also means that user can decide himself if he wants to work with member functions, or freestanding functions.

Alternatives

Could also have used regular std::function rather than the operation types, however std function does add some overhead that regular function pointers does not necessarily do. However that also means that we cant do lambda captures etc.

Pros

Can make the operation point at nullptr, i.e. remove the call to the function if it is not overloaded. Can write general freestanding functions that can be used on more component types, without the use of templates. Allows us to give the user a familiar interface, because everything look like normal inheritance unless they decide to opt-in to more performance by overloading functions with freestanding functions etc.

Cons

We loose de-virtualization. (However, we are kind of doing that ourselves, seeing as we always point to the "lowest" implementation of a function). We might loose inlining. (Having a container with a subclass type, allows the container to inline code etc.) Loose

a lot of as-if optimizations.

Consequences

The OperationTypes are part of the reason why the MetaInformation and createMetaInformation needed to be implemented, as this was a clean way "hide" the functionality from the user unless they want to explicitly take part in it.

5 Implementation

Discuss various interesting problems we faced, and how we solved them.

Topics to address here:

- Lock-free data structures with relaxed atomics. Trying to expose as little functionality as possible to keep the implementation possible.
- Dealing with low level memory and language features were a challenge. A lot had to be limited.
- Layered execution algorithm.
- Trying to find an abstraction level that was as high as needed, but as low as possible.

6 Benchmarking

Write about the results from the different benchmarks, discuss how and why we think we end up with the results we end up with. Show graphs, talk about the tests and why we think we ended up with those results. Refer to the benchmark appendix.

7 Testing

7.1 User Testing

Try and test with Suttung, if we manage to test with suttung we should include their feedback here.

8 Discussion

Discuss the findings we have both from the user testing and the benchmarks. Discuss in what cases this ecs should be used, where is it beneficial to use our approaches? Which features were worth implementing in terms of performance, and usage? Examples:

- Doing layered execution is a good idea when you do a lot of work in your algorithms, and they can be parallelized, however if they can't be parallelized, then the synchronization within the thread pool will reduce performance over a single threaded program.

8.1 Criticism

General criticism of our findings and testing techniques, i.e. what are the factors we can criticize. Examples:

- We never actually tested on mobile, however we did optimizations based on stuff we know about certain architectures.
- We don't necessarily have a proper "counter" test case, for example we haven't implemented our systems with templates, that might be faster.

9 Future Work

Discuss what sort of work that can be added in even after we are finished. Examples:

- Script that parses code files and finds different dependencies. To avoid the error prone way of listing dependencies yourself.
- Layered execution could be adapted to also factor in read_write, currently that is only treated as unknown.
- Tests should be run on mobile as well.
- Mention all the stuff that needs to be fixed before the product is integration ready.

10 Process

Discuss how we approached this problem.

10.1 Plan

Write up how we planed to approach this task.

10.1.1 Re-scoping

Discuss the different approaches we took to re-scoping, why was it needed and what became the new plan?

10.2 Scrum Process

How were our scrum process supposed to work.

10.2.1 Discussion

Why did we change our scrum process.

10.3 Roles

What roles were given internally within the group.

10.4 Tools

What tools did we use?

10.4.1 Discussion

What could we have done differently with our tools, why did we change tools? Jira -> bitbucket -> github. etc. Jira: To much overhead, Bitbucket: To little functionality, Github: Quite alright.

Windows -> Ubuntu All tools were on Ubuntu.

11 Conclusion

11.1 Self Evaluation / Criticism

What could we have done differently?

- There was a clear distinction of work, Trond mostly worked with the benchmark tests. while Per-Morten mostly dealt with actually developing the system. Not necessarily the best work style.
- Some parts of the project could have been planned better.
- We should have adapted the processes to how we actually worked way sooner.
- Architecture was a bit loose, meaning that we were sort of coming up with some concepts as we went along. Talk about some of the positive and some of the negative features of this. However it was based on prototypes.
- Should probably have used more time, we also started a bit "late".
- Did not work in a more test driven manner, in general tests quickly became a second priority, as we were busy implementing the actual functionality.

11.2 Difficulties

General difficulties we had when working with the project. Large project with a large codebase. Sometimes hard to move around in, and to integrate with their cmake system.

Working with the low level memory was a pain to debug sometimes.

11.3 Learning Outcome

Uncertain if we should include this? However part of goals of the thesis was also to learn about certain topics, and we did get a better understanding of the way memory works, as well as the whole new topic of relaxed atomics.

A Test Case Plan

B Design Document

C Technical Specification

D Project Plan

E Scrum Plans

F Daily Scrum Logs

G Stakeholder Meetings

H Supervisor Meetings

I Sprint Retrospectives

J Group Rules

K Benchmark Data

L Stakeholder Contract

M Code