

# Project Plan

## 1 Goals and Setting

### 1.1 Background

The video game industry is one of the largest entertainment industries in the market today. With a focus on creating unique experiences as well as pushing the performance and graphics fidelity forward the video game industry have encountered a plethora of technological challenges. The technology powering these games must be flexible enough to allow for creative gameplay situations, while still able to meet the stringent performance requirements of modern games. This balance issue of flexibility vs performance have lead to some interesting patterns within the industry.

#### 1.1.1 Entity Component Systems

Entity Component Systems originally appeared towards the end of the 1990's, however the popularity of this approach has only grown. [1] Entity Component Systems exists in various forms with different flavors, however they all seem to share the same backbone, the focus on domain specific, decoupled and isolated behavior building blocks. These building blocks can then be combined in almost endless fashions to create complex behavior. The goal of an entity component system is to reduce complexity and decoupling within games. This in turn leads to greater flexibility, higher code reuse, and shorter iteration cycles. An entity is often defined as something that exists in the game world. How the entity functions is described through the component it contains, i.e. there is a larger focus on aggregation rather than inheritance. [2, components] As mentioned earlier there are several forms of entity component systems. Some implementations keep a more traditional OOP implementation, [2, components] while others are more akin to relational databases, where entities are simply identified through a unique ID. [3]

#### 1.1.2 Data Oriented Design

Data Oriented Design(DOD) represents a shift in perspective in relation to software and modeling. Traditional Object Oriented Design favors layers and layers of abstraction and focus on single elements, DOD takes a completely different route. DOD focuses on the data that will flow through the program, what sort of data is it, its frequency, how its transformed, what patterns does it follow, etc. The architectural decisions when designing the program are made to ensure that the flow of data and transformations happens as efficiently as possible. [4, 5] The DOD focus on data is also influenced by the growing gap between memory and CPU speed. The fact that memory access is a huge bottleneck within games, lead DOD down the road of optimizing for memory,

with a focus on homogeneous transformations on large sets of data at the time. [6]

### 1.1.3 NOX Engine

The NOX Engine is an open source game engine originally developed by former students at Gjøvik University College. The engine is still in use today by the company Suttung Digital. After several discussions with Suttung Digital regarding the aim of this project, it was decided that we would develop a new entity component system for their engine. This entity component system would have a focus on performance, support multi-threading and have a data-oriented mindset.

## 1.2 Goals

This project will have a deep focus on entity component systems, both in a practical sense and theoretical sense. The goals of the project is therefore:

- Give insight into different Entity Component Systems. This is done through published articles, as well as our own development and testing.
- Implement a functioning Entity Component System that can work within the NOX Engine. The new ECS should have the same functionality as the old one, as well as improved performance through a more efficient architecture, memory use, multi-threading and a DOD mindset.
- Give insight into performance differences in a relevant test case. This case will be done with both the original NOX Entity system, our proposed system, as well as a more standard Object Oriented solution.

## 1.3 Setting

The following section looks at some of the factors that influence our project.

### 1.3.1 Legal Setting

The NOX Engine is open source under an MIT license. This license is something we have to take into account when developing the new ECS.

### 1.3.2 Scientific Setting

There does not seem to be a great deal of academic articles published in relation to this topic, However what we actually have found seem to be of high quality. This means that we will also have to look into non-academic literature. luckily there seems to exists more sources within the non-academic literature, written by industry professionals.

### **1.3.3 Time**

The bachelor thesis has a hard delivery date in the middle of May. This means we have to structure our project around this deadline.

## **2 Scope**

### **2.1 Subject Area**

The bachelor will revolve around the idea of the entity component system, which is a design pattern commonly used in game engines, Unity and Unreal to name a few. The pattern tries to make it possible to create entities, which is a composition of components, where each component has its each unique separate feature. Different types of combinations of components will result in entities with different functionality. It is this feature of entity component system which gives the larger game engines the possibility to cover such a huge range of games.

For NOX-ECS 2 we will mainly focus on the performance side, and trying to improve it with data-oriented design and multi-threading. This means we will try to structure our implementation in a way that utilizes memory and cache mechanisms as optimally as possible, as well as a focus on effective multi-threading support.

### **2.2 Delimitation**

#### **2.2.1 NOX-ECS 2**

We will only provide the same functionality the old system offered, as doing anything in addition to this would put us on an even tighter time schedule than we already are.

#### **2.2.2 Inheritance Based Implementation**

The traditional object oriented implementation with inheritance is developed with the intention to be able to run the samples like our ECS system and NOX-Engine does, and nothing more.

### **2.3 Project Description**

We are creating a new entity component system module called NOX-ECS 2 for the NOX-Engine. It is going to have a focus on being threaded, having good performance, being maintainable, and achieving the same flexibility as the old entity component system.

#### **2.3.1 NOX-ECS 2**

We will develop an entity component system, which will contain all the old functionality of the NOX-Engines system, but with an added benefit of perfor-

mance and maintainability. If the development of this module goes smoothly, it will hopefully be adopted into the NOX-Engine, and subsequently improve the overall quality of the engine.

### **2.3.2 Inheritance Based Module**

In addition to creating a new entity component system for the NOX-Engine, we will also develop a more inheritance based object oriented version in order to having something akin to a worst case scenario to compare against. This implementation will focus more on the performance part, rather than the other aspects, meaning it will be more streamlined towards our samples; not being able to do much more than required to replicate said samples.

## **3 Plan for implementation**

### **3.1 Important Project Characteristics**

Our bachelor project has several characteristics which affect how we select our development model:

- Only two people doing the development, meaning that the development model needs to have an appropriate scale.
- Need to be agile, still many aspects that are unknown about this project.
- A need for an iterative process, as we have multiple modules that needs to be developed.
- We would like to incorporate multiple practices to ensure a sufficient amount of artifacts are produced to help in writing the thesis.
- Want to have the whole process well documented, in order to still have a good grip of how the entire process went at the end of the bachelor.

### **3.2 Arguments for development model**

For the development of our bachelor project, we decided to go for scrum, together with certain practices borrowed from XP and RUP that we feel are appropriate.

We want our development model to produce a sufficient amount of artifacts and documentation for writing the thesis after the development process is finished. In addition, our previous knowledge of scrum allows us to start utilizing the model much faster, and everyone is up to speed on what to do from the get go. Theres also the fact that we have access to JIRA, which provides us with all the scrum utility tools in one place, making the whole process much easier.

### **3.2.1 Why not eXtreme programming?**

The first major issue we had considering XP was the lack of artifacts and documentation produced by the process. We could have made it work by changing the model around to include more planning and documenting, but that would move the model away from what it is supposed to achieve, which is fluid development throughout the process with new requirements added, changed and removed the entire time. In addition we need a plan to show Suttung that they can approve, in order for us to know if we are on the right track.

### **3.2.2 Why not RUP?**

Even though the whole inception, elaboration, construction and transition phases fits quite well with our project, we feel that this is on the opposite side of the spectrum compared to XP when it comes to amount of documentation and artifacts. The amount of overhead caused by RUP would be too much for a development team with only two members. Use-cases and business models are important in the RUP model, but both are not worth investing our time into. Use-cases is a bit over the top for us, and creating business model is not useful. If we wanted to make RUP fit our requirements, we would remove a lot of its features, making it into something resembling scrum, hence why not just choose scrum.

### **3.2.3 Why not Waterfall model?**

A bit on the fence about going agile or not, since our project does require a substantial amount of research and planing before we can start the development, and there is a hard deadline for the thesis making a rigid plan plausible to utilize. Uncertainties with the specific requirements both from not knowing what kind of entity component system we are gonna make, and how it should be structured to fit into the NOX-Engine means that we cannot use the waterfall model. It would've required way too much time in the planning phase to completely understand everything needed to implement our system.

## **3.3 Applying the Model**

We are going to use the standard scrum model with all of its various features, and also including some other features from some other development models as well.

These are the practices from XP and RUP that we use in our development model.

- **Coding standard**

The coding standard is there to give a consistency throughout our code, making it easier for the team to read others code, and also come to an agreement on what is right and wrong ways to write certain bits of code.

- **Collective code ownership**  
Everyone should have as much insight in what happens in the entire project at all times. This is going to be solved with weekly code reviews, explaining newly implemented code to the rest of the team.
- **Simple design**  
In every aspect of the project, we will have the mindset of trying to create a new functionality as simple as possible.
- **Project glossary**  
In order to make the thesis easier to read through we want to include a project glossary to give an explanation/description on several abbreviations, terminologies, and other words which aren't necessarily that obvious.
- **Software Architecture Description**  
We want a proper laid out plan for the architecture of our entity component system. This is tied up with one of our requirements to give Suttung a design document for the system and then have them approve it before we move into developing the actual system.

## 4 Quality Assurance

### 4.1 Intention

The idea of this project is to end up delivering a functional product to Suttung digital, meaning the product will be used in a professional setting. As a result of this goal, special care has to be put into quality assurance.

#### **Coding Convention**

Coding conventions will as far as feasible follow the internal coding conventions of Suttung digital. Conventions that does not follow the original conventions of Suttung will be discussed with Suttung. No matter how the final coding conventions ends up looking, the focus will always be on writing readable, maintainable and well performing code. This will be enforced through routine code reviews and testing.

#### **Documentation**

Documentation will be a huge factor, as the documentation as well as examples is what Suttung will be reading to gain an understanding of the new ECS. All interfaces will be documented, even self-explanatory ones, as what is and what is not self-explanatory can change from person to person. The exact format of the documentation will be cleared with Suttung.

### **Git Usage**

All developed code will be stored on git. To avoid breaking builds for the other group member, all implementation of new features will happen on separate branches. Branches will be merged into the master branch when the new code has been tested and reviewed by a fellow group member.

## **5 Risk Analysis**

### **5.1 Identified Risks**

Bellow follows the identified risks related to this project, as well as the consequences related to the individual risks.

#### **Poor Performance**

Performance is a pretty hard topic, especially when combined with systems as complicated as ECS's.

**Probability:** Medium

**Impact:** Medium

**Consequence:**

A poor performing system would mean that we would have to spend more time optimizing the ECS, which could get in the way of the other tasks. A suboptimal solution would probably not be something Suttung would be interested in using either.

#### **Incompatible Implementation**

The main goal of this project is to allow Suttung to easily integrate the new ECS into their system. However it might be that our system ends up fundamentally different than theirs, and that the integration is not an easy process.

**Probability:** Medium

**Impact:** High

**Consequence:**

An incompatible implementation would mean that the Suttung group would not be able to easily integrate the ECS into their systems without major changes. In this case Suttung might decide that the new ECS is not worth it, and stay with their current solution.

#### **Multi-threaded Related Bugs**

Writing multi-threaded code is a difficult challenge, and finding bugs in multi-threaded code is even more difficult and time consuming.

**Probability:** High

**Impact:** Depends

**Consequence:**

The consequence of a multi-threaded related bug depends on what sort of bug, and how fatal it is. Non fatal bugs with low recurrence is obviously not wanted,

but it might be acceptable. This cannot be said for fatal program terminating bugs that recur with a high frequency.

#### **Unrepresentative Test Case**

The plan is to write a representative test case with the NOX Engine that we can use to see the flexibility and usage of the engine. However Suttung might decide that this case is not representative enough for their normal usage patterns.

**Probability:** Low

**Impact:** Medium

**Consequence:**

In the case where the test case is not representative enough, we would probably have to rewrite the test case, which will then take up time that we don't really have available.

#### **Lost Work**

Lost work could take many different forms, it could be loss of source code or documents.

**Probability:** Medium

**Impact:** Depends

**Consequences** The consequence of losing work would be to either redo it, or drop the features if possible. How bad this is for the project depends on how much work that needs to be redone, and if we have enough time to redo the work.

#### **Internal Conflict**

Internal conflict would in this case mean all discussions and disagreements that severely hamper the progress of the project.

**Probability:** Low

**Impact:** Depends

**Consequences** The consequences of an internal conflict could be quite severe, in that it would hurt our overall progress. Again the impact here depends on how severe the conflict is and at what time the conflict is over.

#### **Problem Understanding the NOX Engine**

The NOX Engine is a big and complex system, getting a proper overview of the engine might prove difficult.

**Probability:** Medium

**Impact:** High

**Consequences** The largest consequence of this is that if we don't have a proper understanding of the engine, then we won't be able to fully understand how we should optimally implement our new structures either. This will also result in a loss of time that we could have used on development.

#### **Missing Group Members**

Chances are quite large that some days one of the group member won't show



up for work. If this is a result of illness or any other factors is not really that relevant.

**Probability:** High

**Impact:** Depends

**Consequences** The consequence of this situation will be a loss of time that could be spent on work, both for the person who is gone, as well as the member who shows up. How much this impacts the project depends on how long the member is gone, and how much the other member relies on their work.

### **Lack of Proper Measurement Tools**

Measuring performance will require specific tools. Hopefully the tools we want to use will fit the problem at hand, as well as giving us the wanted precision.

**Probability:** Medium

**Impact:** High

**Consequences** If we can't find proper tools for the measurements we want, we will have to find less precise measurement methods, which might hurt our case when we present the performance results.

### **Wrongly Estimated Time**

Wrongly estimating the time it will take to work on each "module" within the project is a real risk.

**Probability:** Medium

**Impact:** Medium

**Consequences** The consequence in time estimation would be that we would have to drop a planned module, like implementing the OOP test case.

## **5.2 Mitigation Strategies**

Bellow follows our strategies for mitigating the chances of the identified risks happening. And how we might deal with them if they would occur.

### **Poor Performance**

Performance problems will be mitigated through recurring profiling. Benchmarks should ideally be run quite often, and these benchmarks should guide further optimizations if needed.

### **Incompatible Implementation**

The implementation will first go through a design phase, before it is presented to Suttung. This proposed design will be built on the research that have been conducted into the Suttung engine, as well as a relevant test case. Hopefully this will be enough to mitigate the changes of our implementation being incompatible with the NOX Engine.

### **Multi-threaded Related Bugs**

Reducing the risk of multi-threaded bugs is a difficult task, however some general

guidelines helps when developing multi-threaded code. We will have a focus on clear access patterns, and local scope when writing multi-threaded code. There might also be some libraries out there that allows us to build tests specifically aimed at multi-threaded code.

#### **Unrepresentative Test Case**

The test case will be developed based on usage patterns observed in real life use of the NOX Engine. The plan is also to verify the case with Suttung.

#### **Lost Work**

All of the source code will be uploaded to on-line repositories, this will also apply to the thesis itself, as well as all other documents related to the project. The group members are encouraged to commit work to these repositories often, preferably after each discrete "unit of work".

#### **Internal Conflict**

Both members of the group have worked together previously on project, which has gone quite well. Because of this we have not added extra mitigation factors, outside of the group rules.

#### **Problem Understanding the NOX Engine**

The best way to mitigate this risk is to use more time with the engine. We will therefore write the test cases that we are going to use for comparisons in the NOX Engine first. Suttung will also give us access to one of their games which is built with the NOX Engine. This will give us some clearer examples to look at, and hopefully reduce the time used to understand the engine.

#### **Missing Group Members**

Missing group members is hard to mitigate, however we will through daily meetings keep the other member up to date on what we are working on. Hopefully this will lessen the impact if a group member does not show up one day. In addition group members are encouraged to work remotely when possible, if they for some reason cannot show up for work.

#### **Lack of Proper Measurement Tools**

There are not that many performance measurement tools directed at C++ out there. We have already found a few profilers with a high precision level. However if we are not able to get the data we need out of these profilers, we might have to go to less precise measurement methods, like measuring frame rate.

#### **Wrongly Estimated Time**

We have already mitigated the consequences of this risk a bit, by leaving the OOP implementation to have a low priority and not being critical. Planned regular meetings will also help reducing this risk, as it allows us to keep track of how we use our time, and potentially scale back some features or tasks.

## 6 Project Organization

### 6.1 Roles and Responsibilities

The group consists of two members, Per-Morten Straume and Trond Lohne. Both members will be responsible for the implementation of the bachelor project, as well as the writing of the bachelor thesis.

Additionally the official project lead will be Per-Morten Straume. The project lead will have the additional task of keeping track of progress, that the development follows the plan, and that the plan is kept up to date.

A signed version of the group rules is included in the appendix.

## 7 Development Plan

### 7.1 Milestones

#### 7.1.1 Milestone 1

##### **Approved Proposed Design**

The first milestone will be to get a proposed ECS design approved by Suttung. This design will give an overview of how the proposed system will function, and will be based on our findings in different literal sources.

##### **Approved Test Case**

In addition to the approved design, we will at this point also have created a relevant test case for our performance tests with the NOX Engine. This will also be approved by Suttung.

#### 7.1.2 Milestone 2

##### **Implemented Proposed ECS**

At this point in time the plan is to have our proposed entity component system functionally implemented. Changes might come up throughout the rest of the project however the API should be relatively set in stone at this point. Optimization also falls under the category of implementation, as we will do performance measurements within the test case as well to guide our optimizations.

##### **Implemented Proposed ECS Test Case**

The other part of this milestone is to have our ECS functioning properly within the test case approved in the last milestone.

#### 7.1.3 Milestone 3

##### **Implemented Inheritance Based Test Case**

At this point in time we also want to have implemented the last of our "modules", for the performance comparisons. It will be the same test cases as the other two milestones, however this will be implemented with a more traditional inheritance focused object oriented mindset.

### **Performance Measurements**

At this stage in the project we will be doing some performance comparisons between the different implementations.

#### **7.1.4 Milestone 4**

##### **Integrated ECS into NOX Engine**

At this point we will reach our final goal, which is to get the ECS integrated into the NOX Engine. We will have done potentially needed refactoring and documentation.

##### **Gamejam with Suttung**

The plan is to have a gamejam with Suttung as a way of testing what they think of the new ECS. It might also be interesting to do gamejam with other people without prior experience with the NOX Engine to compare feedback on the two different systems.

### **7.2 Documentation**

Writing the thesis report will be a continuous effort throughout the entire semester, however it takes over the main focus after the milestones has been reached. We have set off 18 days to write the thesis, including 2 days to review it. We also have some days as a buffer as we have not taken weekends into account in the diagram.

### **7.3 Development Schedule**

As seen in the schedule we will be running 11 sprints, tied up to our 4 milestones. How many sprints that is required per milestone is based on our estimations. We think that both planning and implement the ECS system will take the most time, which is why they get more sprint periods than the other modules.

ID		Task Name	Start	Finish	Duration	Q1 17				Q2 17	
						Jan	Feb	Mar	Apr	May	
1	Development		30-Jan-17	17-Apr-17	55d						
2	Sprint 1		30-Jan-17	03-Feb-17	5d						
3	Sprint 2		06-Feb-17	10-Feb-17	5d						
4	Sprint 3		13-Feb-17	17-Feb-17	5d						
5	Sprint 4		20-Feb-17	24-Feb-17	5d						
6	Sprint 5		27-Feb-17	03-Mar-17	5d						
7	Sprint 6		06-Mar-17	10-Mar-17	5d						
8	Sprint 7		13-Mar-17	17-Mar-17	5d						
9	Sprint 8		20-Mar-17	24-Mar-17	5d						
10	Sprint 9		27-Mar-17	31-Mar-17	5d						
11	Sprint 10		03-Apr-17	07-Apr-17	5d						
12	Sprint 11		10-Apr-17	14-Apr-17	5d						
13	Milestone 1 (Proposed solution, NOX Test Case)		20-Feb-17	20-Feb-17	0d						
14	Milestone 2 (Implement ECS, ECS Test Case)		20-Mar-17	20-Mar-17	0d						
15	Milestone 3 (Profiling, Inheritance Test Case)		03-Apr-17	03-Apr-17	0d						
16	Milestone 4 (ECS Integration, GameJam)		17-Apr-17	17-Apr-17	0d						
17	Documentation		17-Apr-17	16-May-17	21d						
18	Report		17-Apr-17	10-May-17	18d						
19	Report Review		11-May-17	12-May-17	2d						
20	Report Buffer		15-May-17	15-May-17	1d						
21	Report Deadline		16-May-17	16-May-17	0d						

## References

- [1] <https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system#History>. retrieved: 24.01.2017.
- [2] Robert Nystrom. *Game Programming Patterns*. Genever Benning, 2014.
- [3] Entity systems are the future of mmog development part 2. <http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/>. retrieved: 24.01.2017.
- [4] Noel. Data-oriented design (or why you might be shooting yourself in the foot with oop). <http://gamesfromwithin.com/data-oriented-design>, 12 2009. retrieved: 24.01.2017.
- [5] Richard Fabian. Data-oriented design. <http://www.dataorienteddesign.com/dodmain/dodmain.html>, 06 2013. retrieved: 24.01.2017.
- [6] Tony Albrecht. Pitfalls of object oriented programming. [http://harmful.cat-v.org/software/OO\\_programming/\\_pdf/Pitfalls\\_of\\_Object\\_Oriented\\_Programming\\_GCAP\\_09.pdf](http://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf), 2009.