

Arinesalingam, Thomas  
Bjerke, John Ole  
Heksum, Endre  
Karlsen, Henrik Markengbakken

## DeskSim v2

Prototyping Train Simulation

Bachelor's thesis in Programming  
Supervisor: Tom Røise  
May 2022







Arinesalingam, Thomas  
Bjerke, John Ole  
Heksum, Endre  
Karlsen, Henrik Markengbakken

## **DeskSim v2**

Prototyping Train Simulation



Bachelor's thesis in Programming  
Supervisor: Tom Røise  
May 2022

Norwegian University of Science and Technology  
Faculty of Information Technology and Electrical Engineering  
Department of Computer Science





## Sammendrag av Bacheloroppgaven

|                  |   |
|------------------|---|
| Tittel:          | <b>DeskSim v2</b>   |
| Oppgave nr.:     | 19 - 2  |
| Dato:            | 20.05.2022  |
| Deltakere:       | Thomas Arinesalingam<br>John Ole Bjerke<br>Endre Heksum<br>Henrik Markengbakken Karlsen |
| Veileder:        | Tom Røise   |
| Oppdragsgiver:   | Lokførerskolen  |
| Kontaktperson:   | Isak Kvalvaag Torgersen, isator@jernbanedirektoratet.no                                 |
| Nøkkelord:       | Simulasjon, Spillutvikling, Spillmotor, C++   |
| Antall sider:    | 82  |
| Antall vedlegg:  | 9   |
| Tilgjengelighet: | Åpen  |

---

|             |   |
|-------------|---|
| Sammendrag: | Norsk fagskole for lokomotivførere bruker en togsimulator i utdanningen av nye togførere. Deres togsimulator, DeskSim er skrevet i Java og anvender en utdatert og ikke lenger vedlikeholdt spillmotor ved navn JMonkeyEngine laget i 2003. For å sikre at spillmotoren kan oppdateres og brukes i fremtiden, har Lokførerskolen startet et langsiktig prosjekt for å migrere applikasjonen over til en annen spillmotor. Vår oppgave, for å hjelpe til med denne overgangen, er å analysere ulike spillmotorer og lage en prototype in den spillmotoren som kommer frem som best i analysen. Denne bacheloroppgaven presenterer detaljene rundt analysen og beskrivelse og diskusjon av utviklingen samt utviklingsprosessen av simulatoren. Den ferdigstilte prototypen simulerer et tog som kjører på skinner, blir påvirket av signaler og tilbyr en separat modus for redigering av objekter i ulike scenarioer. |
|-------------|---|

## Summary of Graduate Project

|                 |   |
|-----------------|---|
| Title:          | <b>DeskSim v2</b>   |
| Project no.     | 19 - 2  |
| Date:           | 20.05.2022  |
| Authors:        | Thomas Arinesalingam<br>John Ole Bjerke<br>Endre Heksum<br>Henrik Markengbakken Karlsen |
| Supervisor:     | Tom Røise   |
| Employer:       | The Norwegian Train Driver Academy  |
| Contact Person: | Isak Kvalvaag Torgersen, isator@jernbanedirektoratet.no                                 |
| Keywords:       | Simulation, Game Development, Game Engine, C++  |
| Pages:          | 82  |
| Attachments:    | 9   |
| Availability:   | Open  |

---

**Abstract:** The Norwegian Train Driver Academy (Lokførerskolen) utilizes train simulation to educate locomotive drivers. Their proprietary simulator, DeskSim, was written in Java using the increasingly outdated game engine jMonkeyEngine from 2003. To future-proof and mitigate any risk of obsolescence, Lokførerskolen started their long-term project of migrating the project to a modern game engine. To help with this, we were tasked with analysing suitable game engines for the new simulator, as well as developing a prototype in the game engine of choice. This thesis presents the details of analysing game engines, and train simulator development using Unreal Engine with C++ . The resulting prototype simulates a train driving on rails, manages train signals and systems, and offers a separate mode for editing a scenario.

# Preface

We wish to thank everyone who was involved in our Bachelor's project. Special thanks to our supervisor, Tom Røise, for all the support and feedback you provided throughout the project. We want to thank the Norwegian Train Driver Academy for allowing us to contribute to their future simulator for educating new train drivers. We owe a big thanks to the client representative, Isak Kvalvaag Torgersen, for offering support, guidance, availability, and your interest in the project and our development. Finally, we want to thank all our friends and family who have supported us during the project.

# Contents

|   |           |
|---|-----------|
| Preface . . . . .                           | v         |
| Contents . . . . .                          | vi        |
| Figures . . . . .                           | ix        |
| Tables . . . . .                            | xi        |
| Code Listings . . . . .                     | xii       |
| Acronyms . . . . .                          | xiii      |
| Glossary . . . . .                          | xv        |
| <b>1 Introduction . . . . .</b>             | <b>1</b>  |
| 1.1 Background . . . . .                    | 1         |
| 1.2 Problem Area . . . . .                  | 1         |
| 1.3 Delimitation . . . . .                  | 2         |
| 1.4 Target Audience . . . . .               | 2         |
| 1.5 Group Background . . . . .              | 2         |
| 1.6 Project Goals . . . . .                 | 3         |
| 1.7 Constraints . . . . .                   | 4         |
| 1.8 Roles . . . . .                         | 4         |
| 1.9 The Report . . . . .                    | 5         |
| 1.10 Thesis Clarifications . . . . .        | 5         |
| <b>2 Choice of Engine . . . . .</b>         | <b>6</b>  |
| 2.1 Introduction . . . . .                  | 6         |
| 2.2 Absolute Requirements . . . . .         | 8         |
| 2.3 Desired Requirements . . . . .          | 9         |
| 2.4 Learning Curve . . . . .                | 12        |
| 2.5 Existing Solutions . . . . .            | 15        |
| 2.6 Conclusion . . . . .                    | 16        |
| <b>3 Requirements . . . . .</b>             | <b>18</b> |
| 3.1 Functional Requirements . . . . .       | 18        |
| 3.2 Use Case Diagram: Game Engine . . . . . | 21        |
| <b>4 Development . . . . .</b>              | <b>24</b> |
| 4.1 Plan . . . . .                          | 24        |
| 4.2 Process . . . . .                       | 27        |
| <b>5 Technical Design . . . . .</b>         | <b>36</b> |
| 5.1 System Architecture . . . . .           | 36        |

|           |   |           |
|-----------|---|-----------|
| 5.2       | Application module Architecture . . . . .   | 36        |
| 5.3       | Network Architecture . . . . .              | 38        |
| <b>6</b>  | <b>Product Overview . . . . .</b>           | <b>40</b> |
| 6.1       | Menus . . . . .                             | 40        |
| 6.2       | Simulator mode . . . . .                    | 42        |
| 6.3       | Editor mode . . . . .                       | 43        |
| <b>7</b>  | <b>Implementation . . . . .</b>             | <b>45</b> |
| 7.1       | Game flow . . . . .                         | 45        |
| 7.2       | Spline Tool . . . . .                       | 47        |
| 7.3       | Signal controller . . . . .                 | 50        |
| 7.4       | Signals . . . . .                           | 51        |
| 7.5       | Login and authentication . . . . .          | 54        |
| 7.6       | Editor Gizmo . . . . .                      | 55        |
| 7.7       | Landscape . . . . .                         | 57        |
| 7.8       | Landscape Texturing . . . . .               | 57        |
| 7.9       | Save Functionality . . . . .                | 58        |
| 7.10      | User Interface / Heads-Up Display . . . . . | 61        |
| 7.11      | Plugins . . . . .                           | 62        |
| <b>8</b>  | <b>Deployment . . . . .</b>                 | <b>63</b> |
| 8.1       | Packaging and release . . . . .             | 63        |
| 8.2       | Setting up the project . . . . .            | 63        |
| 8.3       | Deployment of documentation . . . . .       | 64        |
| <b>9</b>  | <b>Testing . . . . .</b>                    | <b>65</b> |
| 9.1       | Student User Testing . . . . .              | 65        |
| 9.2       | Employee User Testing . . . . .             | 67        |
| 9.3       | Hardware testing . . . . .                  | 68        |
| 9.4       | System Testing . . . . .                    | 68        |
| <b>10</b> | <b>Discussion . . . . .</b>                 | <b>71</b> |
| 10.1      | Evaluation of project goals . . . . .       | 71        |
| 10.2      | Choice of Engine . . . . .                  | 74        |
| 10.3      | Development Plan and Process . . . . .      | 76        |
| 10.4      | Version Control System . . . . .            | 78        |
| 10.5      | Critique . . . . .                          | 79        |
| 10.6      | Further Development . . . . .               | 80        |
| <b>11</b> | <b>Conclusion . . . . .</b>                 | <b>82</b> |
| 11.1      | Summary . . . . .                           | 82        |
| 11.2      | Final words . . . . .                       | 82        |
|           | <b>Bibliography . . . . .</b>               | <b>83</b> |
|           | <b>Appendices . . . . .</b>                 | <b>87</b> |
| A         | Source Code . . . . .                       | 88        |
| B         | Project Agreement . . . . .                 | 89        |
| C         | Task Description . . . . .                  | 96        |
| D         | Project Plan . . . . .                      | 99        |
| E         | Clockify Summary . . . . .                  | 119       |

|   |                                      |     |
|---|--------------------------------------|-----|
| F | Code Convention Document . . . . .   | 121 |
| G | Requirements Specification . . . . . | 129 |
| H | Meetings and Notes . . . . .         | 142 |
| I | Work Logs . . . . .                  | 176 |



# Figures

|     |   |    |
|-----|---|----|
| 2.1 | Derail Valley (Altfuture) . . . . .   | 15 |
| 2.2 | Libre TrainSim (GPL 3.0) . . . . .  | 16 |
| 3.1 | Application use case diagram . . . . .  | 19 |
| 3.2 | Game Engine use case diagram . . . . .  | 21 |
| 4.1 | Issue status diagram . . . . .  | 26 |
| 4.2 | Gantt chart . . . . .   | 27 |
| 4.3 | Scrum board in Jira Software . . . . .  | 28 |
| 4.4 | Overview of all sprints . . . . .   | 28 |
| 4.5 | The documentation page for the ATrain class . . . . .   | 33 |
| 4.6 | The documentation for ATrain member function GetSpeed . . . . .   | 34 |
| 4.7 | Comparison of original project plan (purple) and actual project process (orange) . . . . .  | 35 |
| 5.1 | Hierarchy of application start up functionality . . . . .   | 37 |
| 5.2 | The module hierarchy for main menu mode . . . . .   | 37 |
| 5.3 | The module hierarchy for simulating mode . . . . .  | 38 |
| 5.4 | The module hierarchy for editor mode . . . . .  | 38 |
| 5.5 | A diagram for the network architecture . . . . .  | 39 |
| 5.6 | File hierarchy for Source and Content . . . . .   | 39 |
| 6.1 | Log in screen . . . . .   | 41 |
| 6.2 | Main Menu . . . . .   | 41 |
| 6.3 | Settings Menu . . . . .   | 42 |
| 6.4 | Train Driver View . . . . .   | 42 |
| 6.5 | Stop Signal . . . . .   | 43 |
| 6.6 | Drone View . . . . .  | 43 |
| 6.7 | Editor Mode HUD . . . . .   | 44 |
| 7.1 | The module hierarchy for simulating mode . . . . .  | 47 |
| 7.2 | A selected spline deforming a railway mesh along itself. The white squares indicate the spline points, and the white lines indicate the tangent of the selected spline point. . . . . | 48 |

|      |   |    |
|------|---|----|
| 7.3  | A railway conforming to the terrain height . . . . .                | 49 |
| 7.4  | Blueprint showing how a signal switches status . . . . .            | 52 |
| 7.5  | The material used for signal lights . . . . .                       | 53 |
| 7.6  | The translation gizmo on a selected signal . . . . .                | 55 |
| 7.7  | The translation gizmo far away, hovering the X-axis arrow . . . . . | 55 |
| 7.8  | The rotation gizmo in the XY-plane . . . . .                        | 56 |
| 7.9  | Additional meshes around the arrows . . . . .                       | 56 |
| 7.10 | Texture is blended based on angle of the landscape . . . . .        | 57 |
| 7.11 | Added texture variations to create realism . . . . .                | 58 |
| 7.12 | Texture details depends on the distance from the camera . . . . .   | 58 |
| 7.13 | Update Speed function for the train DMI in blueprint . . . . .      | 61 |
| 10.1 | Clockify - Overall time usage for project . . . . .                 | 76 |

# Tables

|     |   |    |
|-----|---|----|
| 2.1 | Table of absolute requirements for game engine features . . . . . | 9  |
| 2.2 | A matrix of file support for each engine . . . . .                | 10 |
| 2.3 | Scoring by G2 . . . . .   | 12 |
| 3.1 | Use Case: Place objects . . . . .                                 | 19 |
| 3.2 | Use Case: Delete objects . . . . .                                | 20 |
| 3.3 | Use Case: Operate drone . . . . .                                 | 20 |
| 3.4 | Use Case: Move or rotate object . . . . .                         | 20 |
| 3.5 | Use Case: Operate Train . . . . .                                 | 21 |
| 3.6 | Use Case: Add Level in Main Menu . . . . .                        | 22 |
| 3.7 | Use Case: Add new object . . . . .                                | 22 |
| 3.8 | Use Case: Add object in Content Browser . . . . .                 | 22 |
| 4.1 | Overview of issue status at the end of sprint 3 . . . . .         | 30 |
| 4.2 | Overview of issue status at the end of sprint 7 . . . . .         | 31 |
| 4.3 | Overview of issue status at the end of sprint 10 . . . . .        | 32 |
| 9.1 | User Test Case 1: Start the scenario named "Testing" . . . . .    | 66 |
| 9.2 | User Test Case 2: Drive the train . . . . .                       | 66 |
| 9.3 | User Test Case 3: Exit the game . . . . .                         | 67 |
| 9.4 | User Test Case 1: Place and edit objects . . . . .                | 67 |
| 9.5 | User Test Case 2: Save, exit and load level . . . . .             | 68 |

# Code Listings

|      |   |    |
|------|---|----|
| 7.1  | Changing menu mode . . . . .  | 46 |
| 7.2  | Changing game mode at runtime. . . . .                              | 46 |
| 7.3  | UPROPERTY macro. . . . .  | 47 |
| 7.4  | Getting the length of the mesh . . . . .                            | 48 |
| 7.5  | Calculating spline segments . . . . .                               | 48 |
| 7.6  | Initialization of spline points . . . . .                           | 48 |
| 7.7  | Applying spline point data . . . . .                                | 49 |
| 7.8  | Finds and stores all signals based on class . . . . .               | 50 |
| 7.9  | Updates signals based on ID and type . . . . .                      | 51 |
| 7.10 | Creates dynamic instanced materials for each signal light . . . . . | 52 |
| 7.11 | Decodes and stores info from userinfo JSON response . . . . .       | 54 |
| 7.12 | FActorData array that holds all data that is saved . . . . .        | 59 |
| 7.13 | Example of a class that inherits ISaveableInterface . . . . .       | 59 |
| 7.14 | SaveGame function from SaveManager . . . . .                        | 59 |
| 7.15 | LoadGame function from SaveManager . . . . .                        | 60 |
| 7.16 | Update speed function in C++ . . . . .                              | 61 |
| 9.1  | Code displaying how to apply resolution changes . . . . .           | 70 |

# Acronyms

**3D** Three Dimensional. 3, 8, 9, 22, 47, 57, 61, 72–74, 78

**API** Application Programming Interface. 8, 12, 14, 23, 77

**AR** Augmented Reality. 7

**CPU** Central Processing Unit. 23

**DMI** Driver-Machine Interface. 3, 23, 37, 62, 69, 71

**ERTMS** European Rail Traffic Management System. 1

**HTML** HyperText Markup Language. 33, 64

**HUD** Heads-Up Display. ix, 36, 37, 44, 61, 62, 75

**IDE** Integrated Development Environment. 64

**JSON** JavaScript Object Notation. 38, 55

**JWT** JSON Web Token. 38, 54

**LFS** Large File Storage. 79

**MIT** Massachusetts Institute of Technology. 8

**MVP** Minimum Viable Product. 29–31

**NTNU** Norges teknisk-naturvitenskapelige universitet. 2, 5

**UE** Unreal Engine. 59, 63, 64, 74, 75, 77

**UI** User Interface. 37, 61, 75

**VR** Virtual Reality. 1, 7–9, 13, 15, 73, 80

**XR** Extended Reality. 7

# Glossary

**actor** A base class for any object that can be placed into a level in Unreal Engine. 15, 36, 50, 51, 59, 60

**blueprint** Unreal Engine's own node-based programming language for visual scripting. 11, 14, 22, 45, 46, 61, 62, 74, 75

**Clockify** A third-party service for tracking time spent on tasks within a project. 26

**CryEngine** A 3D game engine developed by Crytek. 6, 8–14, 16, 17

**DeskSim** The current simulator used by Lokførerskolen today. 1, 3–5, 38, 73, 82

**Doxygen** Standard tool for generating documentation from C++ projects. 26, 33, 64

**emissive light** The model itself can act as a light source, without needing a separate light source. 53, 75

**GitHub** A service for hosting software repositories and version control using Git. 7, 25, 27, 29, 78, 79

**gizmo** An overlay with a functional purpose, providing contextually, visually relevant options to the user. 55, 56, 80

**Godot** An open source, cross-platform game engine. 6, 8–14, 16, 17

**Heads-Up Display** Elements overlaying the screen, displaying information to the user. 61

**instanced dynamic material** A material which is instance-editable and can be edited during runtime. 53

**Jira** A web service for tracking and managing product development, developed by Atlassian. 25–27, 29

**jMonkeyEngine** A 3D game engine written in Java by jME Core Team in 2003. 1, 6, 9, 16

**Kanban** A framework for agile software development which focuses on balancing demands with available capacity. 24, 25

**Lokførerskolen** The Norwegian Train Driver Academy, a vocational school educating train drivers, and the client of this project. 1–6, 8, 9, 16, 23, 24, 32, 38, 41, 54, 63, 65, 67, 68, 71, 72, 80–82

**mesh** A graphic primitive, represented as geometric shapes when rendered. 22, 47–49, 55, 56, 77, 80

**Open 3D Engine** An open source 3D game engine managed by The Linux Foundation. 6, 8, 9, 16

**open source** Computer software released publicly and freely for anyone to modify, use and/or publish. 8, 16

**package** Unreal Engine’s process of collecting files and resources and assembling them into an executable software. 63

**packaging** Short for independent, here referring to developers without financial support from a publisher. 7, 8

**runtime** A word to describe the time while a program currently running. 14, 22, 52, 53, 57, 62, 72, 77, 80, 81

**Scrum** An agile framework for project management within software development. 4, 24, 25, 27, 34, 35, 79

**solution** A collection of one or multiple related projects developed in Microsoft Visual Studio. 22, 64

**spline** A mathematical function for interpolating smoothly between multiple points, creating a customizable curve. 14, 17, 47–50, 71, 72, 75

**sprint** A short period where a team aims to complete a set amount of work within the Scrum methodology. 24, 25, 27–31, 33, 35, 73, 77, 79

**triggerbox** A defined area detecting objects’ entries. 42



**Unity** A cross-platform game engine developed by Unity Technologies. 6, 7, 9–11, 13–17

**Unreal Engine** A 3D game and computer graphics engine developed by Epic Games. 5–7, 9–11, 13–17, 22, 26, 29, 32, 36, 39, 45–47, 49, 52, 55, 57, 59, 61–64, 73–76, 78, 82

**viewport** The area on the screen visible to the user. 23, 38, 61

**Visual Studio** An integrated development environment developed by Microsoft. 22, 64

# Chapter 1

## Introduction

### 1.1 Background

The Norwegian Train Driver Academy, *Lokførerskolen*, is a public vocational school part of The Norwegian Railway Directorate. The school educates locomotive drivers over the course of a year. Their study program consists of learning an academic curriculum and physical fieldwork, where they follow an experienced locomotive driver and get to try driving and operating a train. As a part of their academic learning, they use a train simulator called DeskSim. The simulator was originally implemented as an extra tool for students to learn the new signal system ERTMS when it first arrived. Their simulator is also used to simulate real-life scenarios, especially high-risk situations that do not often happen [1]. Since then, *Lokførerskolen* has also implemented a VR mode where they simulate and practice switching train tracks and connecting different wagons to trains.

### 1.2 Problem Area

As a part of the student's education, *Lokførerskolen* uses DeskSim, a self-developed in-house train simulator. DeskSim builds upon *JMonkeyEngine*, a Java-based game engine developed in 2003 that has recently become outdated in some areas. To avoid irrelevancy, *Lokførerskolen* started their long-term project of changing the game engine from *JMonkeyEngine* into a more modern one. This project is a combination of two parts. The first part is an analysis and comparison of different modern game engines. The second part of the project is to develop a train simulation demo in the chosen game engine.

The project is a combination of two parts. The first part is an analysis and comparison of different modern game engines. The second part of the project is to develop a train simulation demo in the chosen game engine.

## 1.3 Delimitation

We have compared different game engines for developing a simulator in this project. While we were going to compare various game engines, we did not perform a comprehensive technical analysis of each game engine and its features. We also limited ourselves to about 4-5 different game engines.

Due to the technical nature of the simulator, we only implemented core functionality related to the movement of the trains and a basic signaling system. Therefore, the project's goal was not to provide a realistic, detailed, physics-based simulation, but a tool which imitates real train movement. For this reason, the movement of the train is not physics-based. The implementation of trains signals has basic logic for controlling its statuses.

Improving the graphical fidelity was not a goal of the project but was accomplished because the engine was newer and more capable. The task was not to develop new content for the simulator, therefore the group reused assets from the existing simulators when possible.

## 1.4 Target Audience

### 1.4.1 Product

The primary audience of the product is people using the simulator. These are people connected to Lokførerskolen, such as students and teachers. To use the product, you would need a computer that can run the simulator and an account to log into the simulator.

### 1.4.2 Thesis

The target audience for the report is people wanting insight into our development process from a project plan to deployment. The primary target audience for this thesis is the development team and administrators at Lokførerskolen, people interested in game development, simulator applications, project management, and software engineering. The thesis, more specifically, the game engine analysis, can be of interest to developers in the process of choosing a game engine. The thesis is written in the context of computer programming and presumes the reader has basic knowledge about computers and software development.

## 1.5 Group Background

All members of the group are students at NTNU in Gjøvik. We have attended the same educational program for the past three years and therefore have similar knowledge and experience with computer programming. During our time at NTNU, there have been several courses that we deem highly relevant for this

project, teaching us the basics of programming and game logic and advanced concepts within software engineering. These courses include, but are not limited to, *PROG1003 - Object-oriented Programming*, *PROG2002 - Graphics Programming* and *IMT3603 - Game Programming*.

In addition to programming, the group members are particularly interested in computer games and graphics. The group member's interest was a significant factor when deciding the assignment for the bachelor project and has played a substantial role in motivating the development.

## 1.6 Project Goals

The project will provide Lokførerskolen with our recommendation of a game engine and develop a demo in our chosen engine.

### 1.6.1 Analysis Goals

Look into what game engines are available and compare and analyze them against each other based on criteria given to us by Lokførerskolen:

- Must be a modern game engine.
- Must support functionality for virtual reality.
- Must be capable of reproducing all functionality of DeskSim.
- The game engine should be easy to learn.
- The game engine should be able to reuse existing 3D assets from DeskSim.

### 1.6.2 Main Goals

We were given a list by the client of various features they wanted us to implement in the demo, and we managed to develop all of the main features, including most of the optional features. The main goal is to create a demo scenario with the following features:

- Must include at least one train, two signals, one train-DMI, a train track, and a simple landscape.
- The train must be able to move using the controllers Lokførerskolen uses today.
- The train must follow the railway in a realistically.
- Signals must be able to change colors based on specific events happening in the game.

### 1.6.3 Part Goals

- Create a tool for placing, editing, and deleting 3D models in the game world, such as trains, buildings, or signals.

- Make it possible to save the world after editing changes have occurred.
- Create a tool for creating, editing, and deleting train tracks.
- Train tracks must obtain curves and not only go in a straight line.
- Make it possible to place a train on the tracks and drive it.

## 1.7 Constraints

Because we developed the application for Lokførerskolen, they had some imposed constraints and boundaries we had to follow. Any deviations had to be discussed and clarified with the client first.

Since we did not have any previous experience or knowledge regarding train operation, we were not responsible for the educational content of the simulator. Although Lokførerskolen did teach us the basics, academic rightfulness was not a requirement set by them.

We were only supposed to develop a demonstration of how some of their functionalities from DeskSim would function in a different engine. The task was not to develop all of the functionalities in DeskSim into our application but only a few key features set by Lokførerskolen in the task description[appendix link].

We started working on the project on the 17th of January, with the deadline being the 20th of May. The bachelor's project lasted approximately four months. When accounting for time spent on project planning, the game engine analysis, and writing the thesis, the development period was limited to approximately three months.

## 1.8 Roles

**Thomas Arinesalingam** was the *Project Leader*. His role specific responsibilities were to ensure that all group members had equal right to express their thoughts. He was the project's "man of action", motivating the development. He also ensured that all deliverables got the needed attention before the deadlines.

**John Ole Bjerke** was the *Research Manager*, overseeing all research and ensuring the level of obtained knowledge is adequate before the development starts.

**Endre Heksum** was the *Scrum Master* for the project and was responsible for the development of the product and took the role of sprint leader.

**Henrik Markengbakken Karlsen** was the *Writer of Minutes*, writing minutes from all meetings and making sure all team members logged both time and work logs.

**Tom Røise** was our supervisor during the project, and provided guidance and academic support to the group during the development process.

**Isak Kvalvaag Torgersen** represented Lokførerskolen, our client for which we developed the train simulator.

## 1.9 The Report

The thesis is written using *Latex* and is based on a template provided by NTNU[2]. It is divided into eleven chapters:

1. **Introduction** contains an introduction to the thesis.
2. **Choice of Engine** contains the analysis of game engines.
3. **Requirements** contain all of the requirements formed for the development of DeskSim v2.
4. **Development** describes the project's development plan and process.
5. **Technical Design** contains the application's system, application, and network architecture.
6. **Product Overview** contains a description and visualization of the product seen from a user perspective.
7. **Implementation** contains information, code, and design explanations for some of the functionality present in DeskSim v2.
8. **Deployment** contains the required information on setting up the project, packaging and releasing it and deploying the documentation.
9. **Testing** contains the user tests and system tests performed on the system.
10. **Discussion** contains a reflection regarding the game engine analysis, the project process, and the final result.
11. **Conclusion** Summarizes the final result of the product and contains some final words.

## 1.10 Thesis Clarifications

The game engine analysis found in Chapter 2 has been a considerable part of the project. It was also a part of the requirements set by Lokførerskolen and included in the project goals. The time usage of 17% in total (appendix E), and the effort the group has put down to create the best possible analysis for Lokførerskolen suggest that the weighting of the analysis in the context of the bachelor's is about 15% of the total.

This thesis will address the final product as the application or DeskSim v2. The train simulation functionality will be addressed as the application simulator, and the editor functionality will be addressed as the application editor. The application editor differentiates from the Unreal Engine editor, the environment in which we have created the application. The application developed by Lokførerskolen will be addressed as the existing solution, or DeskSim.

## Chapter 2

# Choice of Engine

The first part of the project was to analyse different game engines available and compare their strengths and weaknesses. This game engine analysis was performed in the beginning of the project, where we spent two weeks on testing different engines and writing the analysis. In terms of grading and importance of work, we put the weight of this analysis at around 15% of the total grade of the thesis.

### 2.1 Introduction

We were asked to analyse different game engines by the Norwegian Train Driver Academy, *Lokførerskolen*, since they have plans to migrate their existing simulator<sup>1</sup>, which was created in *jMonkeyEngine*, to another game engine. This change is mainly scheduled to prevent the risk of using an engine that no longer gets maintained or updated. To help them make this transition smoother, we have compared and analyzed different modern engines that is available today, and will give an answer as to which engine we believe fits their requirements best.

It was a requirement by *Lokførerskolen* to include Unity and Unreal Engine in the analysis, while Godot, Open 3D Engine and CryEngine were additionally chosen based on their usage within game development. [3] We compared the game engines based on both the absolute and desired functionalities set by *Lokførerskolen*, code languages, expected learning curve for each engine and feedback from developers that has worked with each of the engines.

#### 2.1.1 Thesis Statement

The increasing obsolescence of *jMonkeyEngine*, a game engine used by The Norwegian Train Driver Academy, motivates the migration of their educational train

---

<sup>1</sup>Lokførerskolen's current simulator: <https://lokforerskolen.no/aktuelt/simulatorsenteret/>

simulator to a modern game engine. To identify and decide upon a future-proof game engine, the functionality and relevancy of several engines must be analysed and compared.

### 2.1.2 Unity

The game engine was released by Unity Technologies in 2005 as a Mac OS-exclusive software, but has later granted support for a variety of platforms including web, mobile, console and Virtual Reality (VR). Unity proves itself as a capable engine and displays its wide range through games such as the mobile hits *Pokémon Go* and *Call of Duty: Mobile*, and large-scale games such as *Cities: Skylines* and *Escape From Tarkov*.

Today, the engine is notorious for its ease-of-use and popularity among packaging developers [4], and is commonly described as an effective and efficient platform. [5] It is no secret that Unity is a popular engine, with their CEO John Riccitiello claiming the engine is behind 60 to 70 percent of all extended reality<sup>1</sup> applications. [6]

The Unity license model<sup>2</sup> is split into Personal, Plus, Pro and Enterprise. Their Personal license makes the engine free to use if the annual revenue is below \$100,000. If revenue exceeds \$100,000, their Plus license applies, starting at \$40 per month. Should annual revenue exceed \$200,000, you are required to use the Pro or Enterprise plans, costing \$150 and \$200 per month, respectively.

### 2.1.3 Unreal Engine

Created by Tim Sweeney in 1998, Unreal Engine was initially used during the development of the first person shooter Unreal. It was designed for fast-paced, multiplayer shooters, but has opened up to a wider audience after its public release. Today, the engine is maintained by *Epic Games*, the developers behind large-scale games such as *Fortnite* and *Gears of War*, contributing to show off the engine's capabilities. It is known in the industry for its photo-realistic graphics, lighting, and advanced functionality such as ray tracing and advanced physics. The engine's full source code is written in C++ and is available on GitHub<sup>3</sup>.

Unreal Engine offers two standard licenses<sup>4</sup> for different uses, the Publishing License and Creators License. The Publishing License allows for games and other off-the-shelf interactive products to be sold, but with a 5% royalty after \$1 million in sales revenue. The Creators License has no royalties, but is for internal, non-commercial, or custom applications. [7]

---

<sup>1</sup>Abbreviated as *XR*. Term referring to alternate reality mediums, including virtual reality (VR) and augmented reality (AR).

<sup>2</sup>Unity license model: <https://store.unity.com/compare-plans>

<sup>3</sup>Unreal Engine source code: <https://www.unrealengine.com/en-US/ue4-on-github>

<sup>4</sup>Unreal Engine license model: <https://www.unrealengine.com/en-US/download>



#### 2.1.4 Godot

Development of Godot started in 2007 by Juan Linietsky and Ariel Manzur. The first release of the engine was in 2014. The engine was created because of the technological progress of hardware devices that the engine present at that time did not account for. [8] There are no large-scale games created with Godot yet. The majority of games created are by small packaging teams with restricted budgets.

Today, the engine focuses on their new release of Godot; version 4. They are working on improving their own API, adding Vulkan's graphics API and other features to improve the engines overall performance. [9]

The engine is published under the MIT license making it free to use for any purpose. [10] It is community driven and allows anyone to contribute to the engine source code. [11]

#### 2.1.5 Open 3D Engine

Open 3D Engine is the successor to Amazon Lumberyard, which itself is based on CryEngine. The initial version of the engine was released on July 6, 2021. Maintained by the Open 3D Foundation, an open source, umbrella organization under the Linux Foundation. [12]

The engine is open source, and there are no fees or royalties for using the engine. It is licensed under Apache 2.0. [13]

Given Open 3D Engine is the successor to Amazon Lumberyard, which featured VR support, we expected Open 3D Engine to also have VR capabilities. However, this seems to not be the case, as VR support is not included in Open 3D Engine. As VR is an absolute requirement for this task, this eliminates the engine from further analysis.

## 2.2 Absolute Requirements

All of the engines analysed has support for 3D and Virtual Reality development, with the exception of Open 3D Engine not supporting VR. All engines has the capability to deploy the application on a Windows operating system. Input detection is supported by all engines as long as Windows detects and recognizes the device. The following table consist of each engine's status for the absolute requirements set by Lokførerskolen.

|                               | Unity   | Unreal Engine  | CryEngine   | Godot  | Open 3D Engine                            |
|-------------------------------|---|--|---|--|---|
| <b>VR support</b>             | Yes, with official plugin [14]                        | Yes  | Yes, requires manual setup, workflow can be tedious <sup>1</sup>                  | Yes, but not as extensive  | No  |
| <b>3D support</b>             | Made for 3D and 2D                                    | Only supports 3D   | Made for 3D   | Yes, but better suited for 2D [15]   | Yes                                       |
| <b>Windows support</b>        | Yes   | Yes  | Yes   | Yes  | Yes                                       |
| <b>External input support</b> | Handled in-engine, and can be mapped in the editor    | Handled in-engine, and can be mapped in the editor   | Yes, but requires manual set-up through code                                      | Uses universal input mapping, advanced input can be challenging                                  | Yes, but requires manual mapping with C++ |
| <b>Modernity</b>              | Continually updated, has a large and active community | Active development, Unreal 5 in early access. Internal tools used by Epic Games are often made public [16] | Has planned further development where developers are involved in the process [17] | Community driven and independent development. Does not rely on sponsor opinion, only developers. | Community driven and future-oriented      |

**Table 2.1:** Table of absolute requirements for game engine features

Technological progress has a major factor when considering if engines can reproduce any functionality. We have to take Lokførerskolen's claim that jMonkeyEngine is obsolete into consideration. The probability of migrating any physics, user interface or other functionality into a modern engine is high due to better, more updated features.

## 2.3 Desired Requirements

Lokførerskolen stated a desire to reuse the assets from their current simulator. The 3D file types currently used are Graphics Language Transmission Format (.gltf), AC3D Files (.ac) and Standard 3D Image Format (.obj). The present audio formats are Waveform Audio File Format (.wav) and MPEG-1 Audio Layer 3 (.mp3). The 3D file types are prioritized higher than the audio files because they

<sup>1</sup>Unlike the other engines, CryEngine requires rebooting in VR-mode for testing. <https://docs.cryengine.com/display/CEMANUAL/0culus+Rift>

would be more time consuming to convert or recreate. Of the 3D file types it is the glTF-files that are the most prioritized because of its quantity in the current simulator.

|      | Unity[18] | Unreal Engine[19] | CryEngine[20] | Godot[21] |
|------|-----------|-------------------|---------------|-----------|
| glTF | No        | Experimental      | No            | Yes       |
| ac   | No        | No                | No            | No        |
| obj  | Yes       | Yes               | No            | Yes       |
| wav  | Yes       | Yes               | Yes           | Yes       |
| mp3  | Yes       | No                | No            | Yes       |

Table 2.2: A matrix of file support for each engine

### 2.3.1 Code Languages

#### C#

This compiled language, developed and maintained by Microsoft, is the only officially supported programming language of Unity. [22] It is also supported as an optional language for Godot and CryEngine. Similarly to Java, the language has features like automatic garbage collection, high-level syntax, and being a object-oriented language.

C# is used either natively or optionally in several game engines, making it a good choice as it gives the opportunity to change game engine without having to learn new code languages, only new engine specific features in combination with the code language.

#### C++

C++ is a low-level language and an extension of C. It is known for its performance, flexibility and efficiency. All of the four compared engines are written in C++, while Unreal Engine and CryEngine also use the language for scripting game logic.

C++ is widely considered the gold standard in game programming [23], and has been used for blockbuster games such as *Counter-Strike* and *World of Warcraft*. Its object-oriented style and manual memory management contributes to emphasize good code practises and game development strategies such as design patterns and optimization. When used correctly, the statically typed language boasts a performance and efficiency unlike any of the other candidates. Working in close proximity to the hardware offers precise control over the environment, yet also introduces room for human error.

Unreal Engine refers to its C++ as *assisted C++* due to offering features such as class wizard which sets up a class from scratch or a garbage collector for all classes.

Unreal Engine also provides ways for Blueprints (2.3.2) and C++ code to work together. [24]

### **GDScript**

GDScript is an interpreted scripting language, meaning it gets interpreted at runtime. It benefits when the program requires the code to be platform independent, dynamically typed, and have a small executable size. GDScript is similar to Python in the way that individual blocks are indented and many of the keywords are similar. The language was created to be integrated and optimized with the Godot Engine. [25] Choosing GDScript over Godot's alternate language C#, results in up to four times slower performance. [26]

## **2.3.2 Visual Scripting**

### **Unreal Engine Blueprints**

The Blueprint system in Unreal Engine is flexible and powerful, as it allows the same concepts, logic and tools only available to programmers to be used by non-programmers. Blueprints and code can be integrated with each other, and blueprints should often extend the functionality of baseline systems created by programmers. There are several types of blueprints for different use cases, such as the “standard” Blueprint Class and the Level Blueprint which acts like a global event graph on a level, among others. [27] Its ability to do almost everything C++ can in terms of scripting, means it is the most robust and powerful visual scripting system of the engines in this comparison.

### **Unity Visual Scripting**

Unity's Visual Scripting system is quite new, and has been included since version 2021.1. Earlier versions of the engine requires Bolt Visual Scripting, an external package, to enable visual scripting. Bolt started as a community project, but was acquired by Unity to be integrated as the official visual scripting system. [28] Visual scripts can be divided in two main categories, Flow Graphs and State Graphs. [29] In order to use a visual script on a GameObject, a Script Machine component is added, which can hold a Visual Script asset. This process decouples the script from the machine. Visual scripts can be integrated with and use code, events and properties from the engine, third party plugins and custom scripts. [30]

### **CryEngine Flow Graph**

Flow Graph is CryEngine's main visual scripting tool. Its main uses include creating mission logic and controlling entities and AI in a level. Flow Graph is also used to prototype sound design, effects, and gameplay, as they allow rapid iteration compared to code.

Schematyc is a beta feature in CryEngine. It is designed to control objects in a level, while Flow Graph is more geared towards scripting for the entire level and mission. Another important design aspect is latency, determination and synchronization, with a more distinct execution flow between nodes of different types. Due to the beta status of this feature, it is not ready for production level projects, and should only be used in test projects. [31]

### Godot VisualScript

VisualScript in Godot is more closely related to GDScript in terms of setup and functionality, compared to other visual scripting tools. Visual scripts are attached to game objects the same way other scripts are, and provides one graph per function. [32] Because the Godot Engine API is the same across different script languages, and the library for VisualScript needs improvements, there seems to be little to no benefit to using VisualScript.

## 2.4 Learning Curve

A learning curve is often used to present the relationship between someone's proficiency and experience for a task. In this section, we will analyze and discuss the game engines and their learning curves to form a holistic understanding of their steepness. To compare the learning curves, we have chosen to review specific aspects we believe will be of great importance for this project.

G2, the world's largest marketplace for reviewing software<sup>1</sup>, have ranked the four engines' ease of use from 1 to 10, based on reviews from users. These ratings from G2's data [33], are as shown in 2.3.

### 2.4.1 Documentation

Software documentation should be structured, concise and contain explanations, examples and tutorials to trivial features everyone will encounter when developing in an engine. The engine documentations are set up as tree structures, branching into pages containing smaller topics, using hyperlinks to navigate to the desired page.

However, navigation isn't the only factor regarding ease of use for documentation. The intuitiveness of the documentation structure also impacts the ease of use. The Information Architecture Institute defines information architecture as "The art and

| Engine        | Score            |
|---------------|------------------|
| Unity         | 8.5              |
| Godot         | 8.2 <sup>2</sup> |
| CryEngine     | 7.9 <sup>2</sup> |
| Unreal Engine | 7.4 <sup>2</sup> |

Table 2.3: Scoring by G2

<sup>1</sup>G2: <http://company.g2.com/about>

<sup>2</sup>Disclaimer: The rating may be inaccurate due to an insufficient amount of reviews for this engine.

science of organizing and labeling web sites, intranets, online communities and software to support usability and findability”. [34] With this definition, both Unreal Engine and Unity scores high because their layout of preview images provides a clear overview of the contents. CryEngine and Godot’s respective documentation does not emphasize findability to the same degree.

As a proof of concept, we tried to find the documentation page for Virtual Reality (VR) in each engine’s documentation, measuring the ease of use based on perception and intuition. For Unity, the reference to VR is found in the main page of the documentation. It contains how to get started, a pre-made VR template and links to other relevant material. Meanwhile in Unreal Engine, the information for VR was found in a subsection from the documentation main page. The information found in the section was start-up guides and best practises for each of the supported headsets. In CryEngine’s documentation, the VR section was located in the main page. There was information on how to use the functionality, but no structured guide or additional information. In Godot, the VR section is exposed on the main page, but further navigation lacked intuitiveness for where to find a start up-guide and general documentation.

## 2.4.2 Community Support

All of the engines have some sort of community support, with either forums, tutorials or community platforms. Unity and Unreal Engine has the biggest community among the four due to their popularity and use in the industry [35], where Unity is usually biggest within indie games while Unreal Engine is better suited for larger scale games. [36] CryEngine has a smaller community compared to Unity and Unreal Engine, making it difficult to find solutions for related issues. There is also less community created resources to learn from. Although its lack in size could make it easier to come in contact with the developers. [37] Godot is very similar to CryEngine in the sense that it has a small community compared to Unity and Unreal Engine. [15]

## 2.4.3 Development Tools

All four engines include tools for editing materials. Unreal Engine, Unity and Godot have the option to use a graph-based approach for editing materials from the included editor features. CryEngine is limited to a drop-down menu for editing materials.

Source code access lets developers change the code on which the engine is built. This comes in hand when the developers wants to make noticeable changes to the physics or other engine attributes. All of the engines offers their source code to be viewed for free except Unity which requires the enterprise licence.

The only engine that offers VR editing mode is Unreal Engine, this is an approach to game development which allows the developer to virtually edit games. [38]

Unreal Engine and Unity offers dynamic occlusion culling<sup>1 2</sup>, a trait of the rendering pipeline deciding which game objects to exclude from the draw call at runtime. This saves the graphics processor from drawing any vertices not seen by the player, offering a great boost in performance.

In CryEngine, the culling is done by the graphics API, ignoring vertices not seen on the final screen, although this gets done by default in all engines. Engine-level culling would need manual setup. Godot's occlusion culling requires manually specifying which objects to be ignored by the draw call.

Both Unreal Engine, CryEngine and Godot have integrated tools for generating splines called Blueprint Splines<sup>3</sup>, Spline Distributor<sup>4</sup> and Path<sup>5</sup>, respectively. These tools generate curved paths based on set points in the engine's world-space, which can be useful for generating train tracks. Unity has similar features through the official package *ProBuilder*.

#### 2.4.4 Scripting

The considered programming languages, disregarding visual scripting, have an established order of difficulty.

GDScript's resemblance to Python, a language well-known for being one of the easiest to learn, makes it a strong candidate for the simplest coding language. It is adequate for beginners due to being dynamically typed and syntactically friendly. To add code logic in Godot, simply add a script to a game object, which generates a script template. Granted Godot becomes our engine of choice, the choice of GDScript as the development language is not certain, due to C#'s advantages (2.3.1).

Similarly, in Unity, C#-scripts are added as components to tell game objects how to behave. The user can define the order of execution by implementing event callbacks such as *Start* and *Update*, while the engine handles the code execution loop. Unity offers several official code libraries for concepts such as math, physics and debugging, to support the development.

Despite having a steeper learning curve, C++ has become an industry standard for high-end game development. As if the language itself wasn't hard enough to grasp, CryEngine requires manually setting up the code framework, separating the code from the engine, and has an insufficient amount of learning resources due to the engine's low popularity. Scripting game logic works similarly to Unity and Godot, where scripts are added to entities.

---

<sup>1</sup><https://docs.unity3d.com/Manual/OcclusionCulling.html>

<sup>2</sup><https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/VisibilityCulling/>

<sup>3</sup><https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/BlueprintSplines/>

<sup>4</sup><https://docs.cryengine.com/display/CEMANUAL/Spline+Distributor>

<sup>5</sup>[https://docs.godotengine.org/en/stable/classes/class\\_path.html](https://docs.godotengine.org/en/stable/classes/class_path.html)

Due to the similarity to Java, and features mentioned in C#(2.3.1), the transition from Java to C# should be straightforward.

C++ is the main scripting language of Unreal Engine. The engine's base building block is called *UObject*, and is used by the *AActor* class, among others. The lifecycle of an *AActor* starts with *BeginPlay*, an event which is called when the actor is first spawned into the game world. It then goes into the Tick life cycle, which is called once per frame until *EndPlay* is called when the *AActor* leaves the world. [24]. Due to the complex functionality stated in C++(2.3.1), the language is commonly defined as one of the more difficult languages to learn as a beginner.

## 2.5 Existing Solutions

To gather insight from the industry, we contacted developers behind relevant games and simulators. The process included finding games made in the applicable engines, determining their relevance to trains, simulation or Virtual Reality, and contacting their studio/author by email. All subjects agreed to having their answers cited as part of our thesis.

When asked which engine they believe is best suited for VR game development, *Vankrupt Games*, the developers of *Pavlov VR*, states it depends on the specific use case:

"Unity is great for beginner devs and getting a project started," they claim, adding that "there is a lot of 3rd party support."

Talking about Unreal Engine, the engine behind *Pavlov VR*, *Vankrupt Games* claim that "support and documentation from 3rd parties can often come second class due the the proliferation of Unity being a more popular engine." But when asked about the reason for choosing Unreal Engine, they justify by explaining that "[it] uses a lower level language and has a great rendering pipeline."

Slobodan Stevic, CEO at *Altfuture* and developer of *Derail Valley*, explains they chose Unity "... because its VR support was better than that of other engines, back in 2016." When asked which engine he believes is best suited for VR simulators, he adds "... it depends on the type of the simulator, its scope and style choice, as well as prior team experience."



Figure 2.1: Derail Valley (Altfuture)





Figure 2.2: Libre TrainSim (GPL 3.0)

Similarly, *HaSa*, a contributor and developer of the open source project *Libre TrainSim*, states that “as long as it supports ‘basic’ requirements it is a matter of personal preference.” Describing the similarity of engine functionality, they also conclude that “the engine doesn’t really matter.”

Godot, the engine used to make Libre TrainSim, gets praised by *HaSa* for its agility. “Iteration speed is key to a good game in general. The more you iterate the better the result is.” They also comment on some downsides of the engine, claiming the renderer is “quite performance heavy out of the box which limits the creative freedom”, and the audio tools are ‘basic’. “We have to develop a lot of features on top to achieve good quality,” *HaSa* writes.

The developers we contacted behind CryEngine’s small list of titles relevant to this analysis were unavailable for comment. When asking all subjects if they, in retrospect, believe their respective engine was the right choice for their game, everyone answered **yes**.

## 2.6 Conclusion

The *jMonkeyEngine* game engine, currently in use by The Norwegian Train Driver Academy, has become primitive compared to modern technology. The importance and frequent usage of their train simulator inspire the need to migrate their software to a modern game engine. To mitigate this concern, several available solutions must be considered and evaluated.

All engines, excluding Open 3D Engine, fulfill the Absolute Requirements (2.2) set by Lokførerskolen. Godot and Unreal Engine supports `glTF`-files, the most prioritized asset type. Meanwhile, CryEngine and Unity are at a disadvantage as they support fewer and less prioritized assets. While all engines require a conversion pipeline for assets, Unreal Engine and Godot can reuse more assets out of the box. This decreases time spent on porting assets to the new engine.

The client being experienced with programming eliminates the need for GDScript as its most significant trait is being beginner-friendly. C++, on the other hand, may be unnecessarily advanced for the project. As for C#, because of its powerful and

modern functionality and versatile usage across game engines, we fail to see any relevant downsides, and we conclude it as a fitting language for the project.

The overall learning curve for the individual game engines sets Unity and Unreal Engine apart from the competitors. The magnitude of the community and related quantity of available material, together with their emphasis on standards in documentation, establishes their strong position in today's game engine market. Godot is still a relatively new engine without any major titles, causing both documentation and community resources to be inadequate in quality and quantity compared to Unreal Engine and Unity. CryEngine initially limited its use to enterprise only, resulting in less publicly available resources and documentation.

The majority of the developers we reached out to advocated the choice of engine to heavily depend on the use case of the project. When developing an open world train simulator, this favors Unreal Engine as their engine provides tools such as dynamic occlusion culling, a dedicated spline tool and frequent release of tools used for in-house games.

In conclusion, we believe all four engines are capable of producing virtually any type of game. The difference lies in time and effort needed to achieve the same result. Throughout the analysis it is evident that Unity and Unreal Engine comes out as superior candidates, and would both be beneficial for this project. Despite Unreal Engine's steeper learning curve, we must consider the functionality provided for the specific use case of the project. Based on the results of this analysis, we conclude by proposing **Unreal Engine** to be the most suited solution for the migration of the current simulator.

## Chapter 3

# Requirements

With the conclusion of the game engine analysis, we could start working on the specifics of development. We first had to assess what the client wanted us to develop to do this. The client stated their initial requirements in the form of one main goal and two sub-goals. The main goal was to create a demo that satisfied the task requirements specified in the task description (appendix C). They also presented two sub-goals, an in-game model placing tool and an in-game railway builder. Our requirement document is based on these requirements but does also extend further. This chapter goes over the specific requirements set for the project by the client, the group itself, and the industry standards.

### 3.1 Functional Requirements

To display and visualize our simulator's core functionalities, we have chosen to utilize use cases. Use cases provide structure and overview of the functionality and work as a tool to force awareness of the requirements in the development phase.

There was an unanticipated issue with defining the use cases for the project. As stated previously, our main goal is to make a demo in the chosen engine and make the code applicable for further development. We discovered in the development of the demo that the classes and structures we created often laid the groundwork for further development. Therefore, we decided to include two separate groups of use cases for our project, one for the application itself and another for the use cases that we facilitated through development. Including this is for the client to understand the functionality more easily. Including both groups of use cases also forces us to focus on the project's code quality and further development perspective.

### 3.1.1 Use Case Diagram: Application

The use case diagram illustrates all available actions for the two groups of users; students and employees. Employees have access to all student functionality even though it is not implied in the diagram to avoid cluttering.

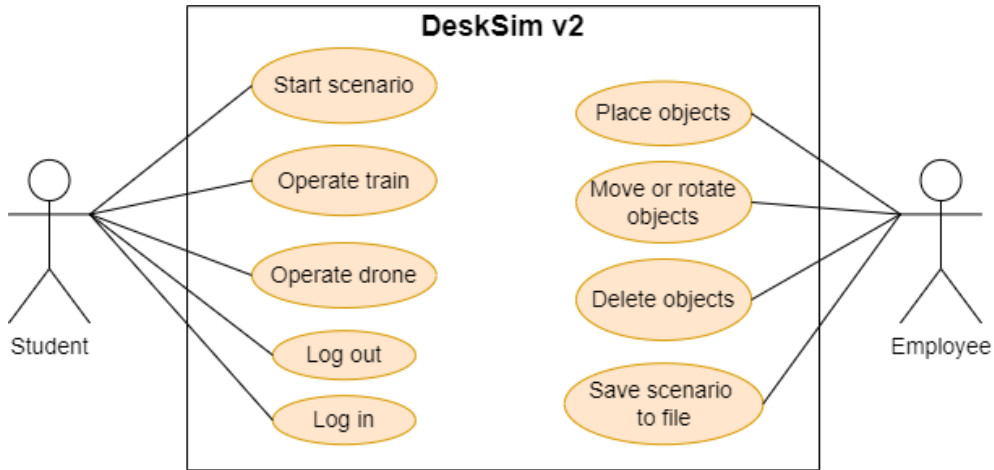


Figure 3.1: Application use case diagram

#### High Level Use Case

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Place objects  |
| <b>Actors:</b>      | Employee   |
| <b>Goal:</b>        | To place the necessary objects such as a train and a railway in a level.   |
| <b>Description:</b> | When a level is opened in editor mode the user is provided a user interface which includes a content browser. The user can click on a item in the content browser and drag it out in the level. The content browser has different categories the user can select in the top bar by clicking on the category buttons. |

Table 3.1: Use Case: Place objects

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Delete objects  |
| <b>Actors:</b>      | Employee  |
| <b>Goal:</b>        | To delete an object in the level.   |
| <b>Description:</b> | When clicking on an object in a level the user will be given the option to remove it. After the click, the user will get a trash can symbol at the top bar, next to the transformation options. After clicking on the trash can, the program will prompt the user for confirmation before permanently removing the object from the scene. |

Table 3.2: Use Case: Delete objects

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Operate drone   |
| <b>Actors:</b>      | Student   |
| <b>Goal:</b>        | Maneuver the drone camera   |
| <b>Description:</b> | The student switches to drone view. This lets them move around freely in all three dimensions, forwards/backwards, horizontally, and vertically. The user can also use the mouse to freely look around in the world, and adjust the movement speed of the camera. |

Table 3.3: Use Case: Operate drone

### Low Level Use Case

|                          |  |
|--------------------------|--|
| <b>Use Case:</b>         | Move or rotate objects   |
| <b>Actors:</b>           | Employee   |
| <b>Goal:</b>             | To move a object or rotate it into the position and position you want  |
| <b>Precondition:</b>     | The user has successfully opened a scenario in editor-mode   |
| <b>Success Scenario:</b> | The employee selects an object by clicking on it. This enables a gizmo, either in the form of three arrows for translation along each of the three-dimensional axes, or a wheel for rotation. The user can then grab the gizmo with the mouse, and drag the arrows to move the object, or drag the wheel to rotate the object around itself. |

Table 3.4: Use Case: Move or rotate object

|                          |  |
|--------------------------|--|
| <b>Use Case:</b>         | Operate Train  |
| <b>Actors:</b>           | Student  |
| <b>Goal:</b>             | To drive the train in a scenario   |
| <b>Precondition:</b>     | The user has successfully opened a level and the levers is connected to the system through a USB port  |
| <b>Success Scenario:</b> | <p>The user accelerates or decelerates the train using the levers or the keyboard, as is provided with information about the current speed in the Driver Machine Interface.. Depending on the scenario, the user has to follow some rules:</p> <p>The user should not exceed the speed limit. Doing so should result in system regulated brakes turned on.</p> <p>Rules regulated by signals:</p> <p><b>Main signal:</b> If this signal is red the user should stop. If user don't stop before the signal this should result in breaks turned on.</p> <p><b>Main signal:</b> One green light means that the user can drive with reduced speed.</p> <p><b>Main signal:</b> Two green lights means that the user can and should continue with the set speed.</p> |

Table 3.5: Use Case: Operate Train

### 3.2 Use Case Diagram: Game Engine

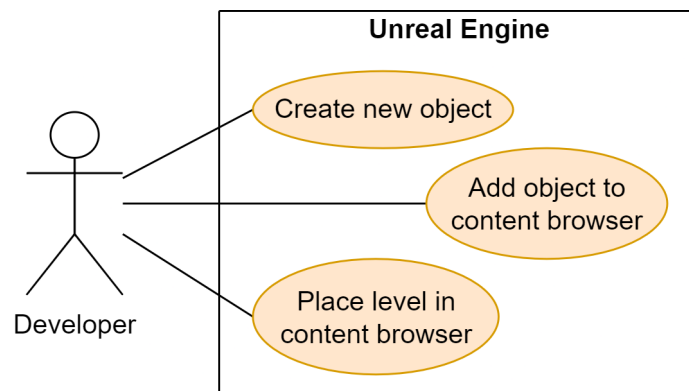


Figure 3.2: Game Engine use case diagram

**High Level Use Case**

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Add Level in Main Menu   |
| <b>Actors:</b>      | Developer  |
| <b>Goal:</b>        | To add a level created in Unreal Engine to the simulator.  |
| <b>Description:</b> | The developer opens up the EditorHUD-blueprint, click on the plus sign and fills in the level name, description, FName level reference, and a preview image. |

**Table 3.6:** Use Case: Add Level in Main Menu**Low Level Use Case**

|                          |   |
|--------------------------|---|
| <b>Use Case:</b>         | Add new object  |
| <b>Actors:</b>           | Developer   |
| <b>Goal:</b>             | To add a new object to the game   |
| <b>Preconditions:</b>    | The developer has Unreal Engine version 4.27.2 or higher, and a 3D model they want to add to the game.  |
| <b>Success Scenario:</b> | The developer adds a new model to the "models" folder inside Unreal Engine. This lets them register the new object by deriving a new blueprint from the C++ class relevant to the objects category. The model then needs to be referenced by the blueprint's static mesh, before the blueprint is compiled and saved. |

**Table 3.7:** Use Case: Add new object

|                          |   |
|--------------------------|---|
| <b>Use Case:</b>         | Add object in Content Browser   |
| <b>Actors:</b>           | Developer   |
| <b>Goal:</b>             | To successfully add a created object in the content browser making it clickable and draggable in runtime.   |
| <b>Preconditions:</b>    | The developer has created a new object as described in the "Add new object" use case.   |
| <b>Success Scenario:</b> | Inside the EditorHUD-blueprint, the developer can add a new item to be registered as an object by the application, with a name, description, and relevant item category. The actor asset needs to be referenced in the blueprint. It is recommended to restart the engine and rebuild the solution with Visual Studio for the object to show up in the content browser. |

**Table 3.8:** Use Case: Add object in Content Browser

### 3.2.1 Operational Requirements

These are the requirements which concerns the application at it's operational stage, this stage begins at the project's deadline which is the 20th of May:

- The application must be able to interact with the existing Rest-API hosted by Lokførerskolen.
- The system must operate on Windows devices.
- The system must manage privileges of users and only allow elevated users to access the editor functionality.
- Must be transferable through a zip-file.
- The system must operate on computers which have 8 gigabytes of memory and an Intel® Core™ i5-4460 CPU or better.
- Should not experience frame rate drops of lower than 60 frames per second.

### 3.2.2 Security and Misuse

To ensure the security of the users and avoid misuse of the application, it:

- Must require authentication of users.
- Should not contain bugs or security flaws that could potentially harm or destroy hardware components. Such flaws include bad memory handling.
- Must not store any passwords in plain text.

### 3.2.3 Interface Requirements

#### Menus

- Menus should be intuitive and easy for a student at Lokførerskolen to navigate the Main Menu.
- The Main Menu should have the same functionality as the previous simulator and only deviate by design.
- All text must be available in Norwegian.
- Buttons should be intuitive to reduce the number of operations required for a task.
- The DMI viewport in a game should be responsive to the gameplay.
- All numbers and measurements must be specified in the metric system.



## Chapter 4

# Development

This chapter describes the work methodology and the process used during this project. We compare two agile software development models and describe our implementation of the chosen model. We also explain three sprints by looking at the sprint goals, results, and retrospective notes from the sprint meetings.

### 4.1 Plan

#### 4.1.1 Development Methodology

The choice of methodology was mainly based on three facts. 1) The group has little experience with game engines. 2) The group has little to no experience working on a project of this scale. 3) One of the requirements of Lokførerskolen was that they would be tightly connected to the development and emphasized this connection in their task description.

Based on the three criteria stated above, the group concluded on using an agile software development model because it allows for rapid change in these requirements. It also allows for changes in the project scope without necessarily having significant impacts on releases or the planned progress.

#### 4.1.2 Development model

When choosing the optimal model for our project, we looked at several different models to see if they would fit our project. Two of the development models that got evaluated were *Scrum* and *Kanban*.

Scrum is an agile development model. It is designed for teams of ten or fewer members who divide their work into increments of work within iterations called sprints, which are usually between two to four weeks long.[39] Development teams utilizing Scrum meet once a day for 15 minutes or less to assess their progress and keep everyone updated. Two other meetings are essential when working

with Scrum; the sprint review, which is often held together with the project stakeholder for feedback, and a retrospective meeting to reflect on the previous sprint results.

Kanban is also an agile development model. The model visualizes the items or tasks from start to finish, usually through a Kanban board.[40]. The main focus for teams working with Kanban is reducing the time a project takes from start to finish by continuously improving their workflow.

The model we chose as our development model was Scrum. Scrum allows and emphasizes reflection and assessment at every step of the process and has a pre-decided structure. Scrum is also the development model most known and used by the group previously. The main reason for this was that although both methodologies could be utilized and work in our project, we wanted the main focus to be on creating the best application possible.

### 4.1.3 Our implementation of Scrum

We aimed to use the iterative nature of Scrum to ensure the quality of the implementation at every stage of the project and quickly adapt to any change in the client's specifications. We implemented Scrum strictly, with daily Scrum meetings throughout the project. At the beginning of the project, we had one-week sprints and continuously discussed if we needed to increase the sprint length based on the upcoming tasks.

These meetings worked as a collaborative tool to identify problems if someone was stuck, keep everyone up to speed on the project process, and create a professional work environment where we kept in touch. We believe the latter was more important than it seemed because most of the project work will be done from home. Although these meetings are important for the group, they could occasionally be skipped if they are found unnecessary.

For managing the project, we utilized *Jira* integrated with *GitHub*. This integration contributed to ensuring the professionalism of the project's development process. We used constrained and secure workflows to ensure that every issue followed the correct workflow. In simpler terms, this is a set of rules for how all issues could be handled throughout the project. Figure 4.1 shows the direction and constraints on how an issue can progress through the different statuses.

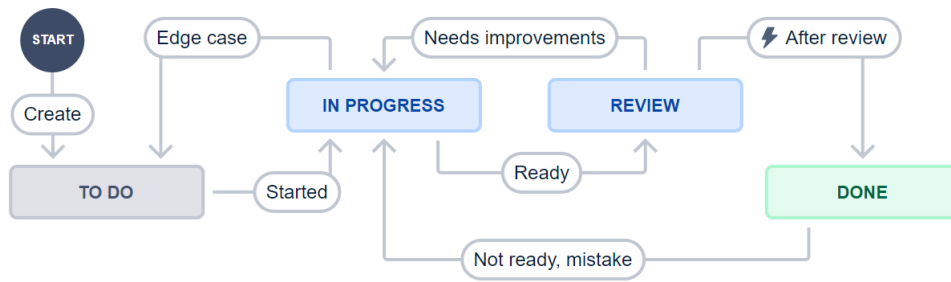


Figure 4.1: Issue status diagram

#### 4.1.4 Process Documentation

To log the time spent on the project, we planned to use the *Clockify* add-on for Jira. The add-on would allow the group to manually start a timer from within the issue in Jira, tracking all work time until it is manually stopped. The elapsed time should get manually logged along with a short comment of what was done that day. For planning the holistic overview of the project, we planned to use the extension *BigGantt*. The extension to Jira allows the group to create a Gantt-chart in the same environment as our issues.

#### 4.1.5 Code Documentation

To ensure the quality and professionalism of the project, we searched for and agreed upon some code conventions for C++ programming in Unreal Engine and standards for development. These conventions can be found in the appendix F. This document introduces *Doxygen*, a documentation generator we utilized for automatically producing code documentation for the client. We saw this as a necessity, especially because we were writing code for a client.

#### 4.1.6 Git Workflow

The group agreed upon some rules for working with git. these rules were made to ensure that the code was well protected against human error. The workflow consists of seven rules and it is mandatory for all group members to follow.

- Always create a new branch when starting work on a feature. No work should be done directly on the main branch.
- This is, and should be the naming convention for branches:  
`<issue_number>-<branch_name>`
- This is, and should be the only commit convention:  
`[#<issue-number>]-<description>`
- When the work is completed on a branch, it must be deleted after the work is merged into the main branch.

- Code should be committed often, either when a task is finished or the newly written code is fully functional.
- Do not commit code that doesn't compile. Code should be tested before it is committed.
- When a feature is complete, its branch should be merged into the current milestone branch. When the work for one milestone is completed, this branch should be merged with main.

### 4.1.7 Gantt Chart

Figure 4.2 presents the milestones (found in appendix D) for the project and their planned schedule. This chart is a visualization of the planned project process, and it will be compared to the actual project process in section 4.2.4.

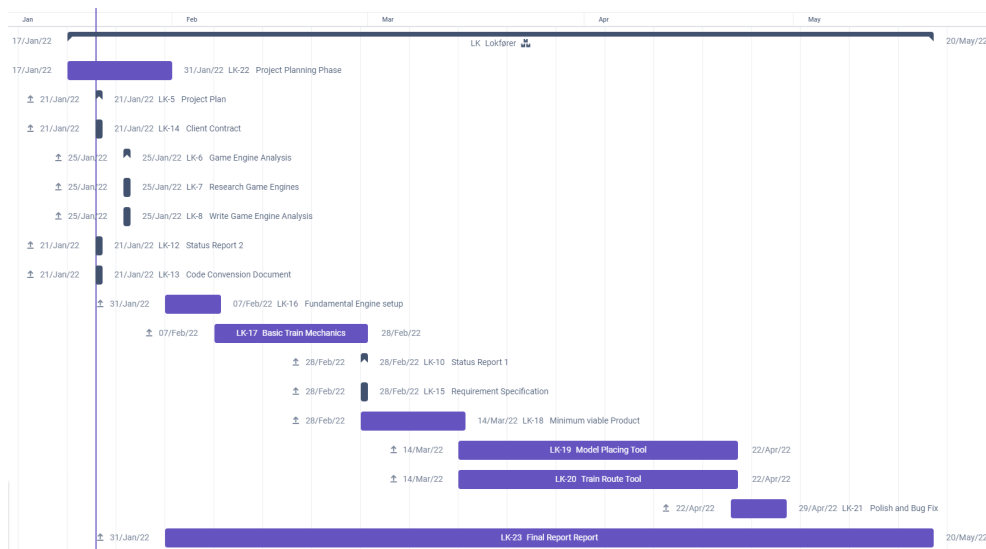


Figure 4.2: Gantt chart

## 4.2 Process

This section will describe our usage of development tools such as Jira. It will also cover the estimation process, examples of weekly sprints, and an overview of the changes and deviations from the original project plan.

### 4.2.1 Jira

The group used Jira to implement Scrum into our project. We created our product backlog and kept track of all issues and sprints in the Jira software. To use Jira to the full extent, we followed the git workflow. When a push was made to GitHub,

using the correct commit conventions provided a commit history available in the software.

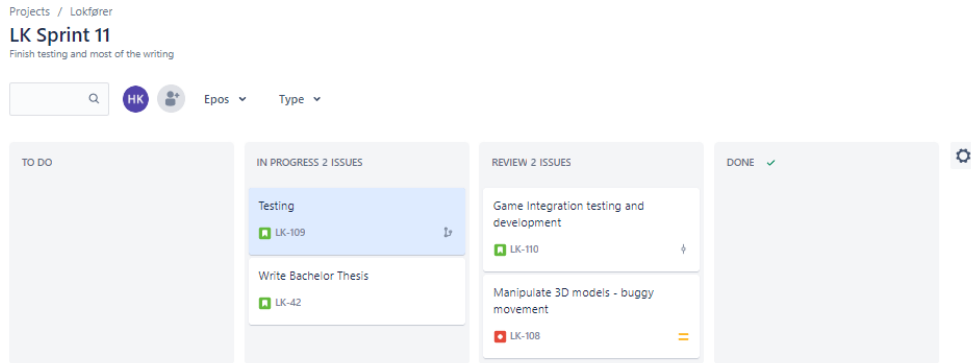


Figure 4.3: Scrum board in Jira Software

### 4.2.2 Sprints

Our project consisted of eleven sprints. Figure 4.4, contains a visualization of the main focus area for each sprint. Sprint three, seven, and eleven are displayed below to illustrate how the group approached and solved issues related to setting up, developing, and testing the system. For each sprint, a table display the result of the sprint with the estimations made.



Figure 4.4: Overview of all sprints

The sprint meeting notes are written with three different sections. A section about the sprint Goals, a section discussing the sprint result, and a retrospective section where the sprint is discussed and reflected with the sprint goals in mind.

### **Estimations**

After sprint three, we hopefully decided to change the estimation strategy to estimate more accurately. We started out estimating issues in T-shirt sizes, ranging from 1 to 4, which imitates small, medium, large, and extra-large issues. We changed the estimations to be an exact number but tried to keep the task sizes low.

### **Statuses**

The issues could have one of five statuses describing its current state in a sprint. The statuses are:

**Wishlist** - The issues taken out of the project scope and in to a list of functionality that we can decide to include if we have time later in the project.

**To Do** - This is the section for issues waiting to be developed.

**In Progress** - The issues that are currently being developed.

**Review** - The issues that is currently under review by peers.

**Done** - The issues that has been completed in the sprint.

## **Sprint 3**

**Date:** 07.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 07.02 - 14.02

### **Sprint Goal:**

- Review and deliver Game Engine Analysis to the client and to our supervisor
- Integrate Jira with GitHub
- Setup the repository in GitHub with an Unreal Engine project

### **Sprint result:**

“This is the first sprint where we managed to get all our goals finished in time. We managed to finish the setup and integration with Jira on time, which resulted in being able finish the MVP within two weeks.”

|           | Other statuses | Done   |
|-----------|----------------|--|
| Issues    | -              | Write Engine Analysis<br>Integrate GitHub and Jira<br>Setup Repository<br>Code Convension Document<br>Learning Engine Basics |
| Estimates | 0              | 4 + 2 + 2 + 2 + 2  |

**Table 4.1:** Overview of issue status at the end of sprint 3

#### Retrospective:

“We finished all the tasks we had in mind and also managed to start learning the engine basics as well. This means that we from next week can start working towards the MVP. We discussed increasing the sprint length from one to two weeks for the next sprint, but decided to keep the one week sprint because we want to have a new retrospective meeting next week and evaluate the accuracy of the story point estimates we made on the issues regarding the MVP to figure out if we can finish the MVP in the allocated time. This decision was made because the time we have set to be finished with the MVP and display it to the client is the 14th of March.”

#### Sprint 7

**Date:** 07.03.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 07.03 - 21.03

#### Sprint Goal:

- Finish first draft of requirement specification and status report 1.
- Finish the MVP scenario

#### Sprint result:

“The MVP got finished and put together. We had some issues with the environment creation and its time consumption, but the estimations we made for three environment creation tasks was more accurate than our previous estimations. All the necessary functionality was there to display the MVP to the client except the basic statuses which we only controlled by a timer instead of an controller. The basic statuses was only missing a few hours of work and is therefore almost finished.”

|           | Wishlist  | To do   | In Progress  | Done   |
|-----------|---|---|--|--|
| Issues    | Editor mode:<br>- Generate flat terrain (14)<br>- Manipulate terrain (20)<br>- Dynamic buttons (10) | Place and edit railway (28)<br>Load scenario from file (14)<br>Save scenario to file (14)<br>Static Buttons on screen (7) | Main Goal:<br>- Signal Controller (21)<br>- Emergency Breaks (7)<br>- Stop Scenario (21)<br>In game content browser (25)<br>Save Object to File (20)<br>Load Object From File (20) | Main Menu (14)<br>Basic Statuses (7)<br>Second iteration Requirement Specification (21)<br>Place 3D Models (12)<br>Manipulate 3D Models (25) |
| Estimates | 44  | 63  | 114  | 79   |

Table 4.2: Overview of issue status at the end of sprint 7

**Retrospective:**

“The sprint goals was achieved, but the basic statuses was only implemented to work with the MVP scenario and not finished to he extent we want in the final product. We want the signals to be controlled by a controller and be able to trigger events. New issues for this will come in the next sprint. We finished 93 work hours in the sprint and we feel that our estimations on the different tasks are beginning to get better and more accurate to how we progress in the sprints.”

**Sprint 10**

**Date:** 21.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 18.04 - 01.05

**Sprint Goal:**

- Finish the demo test level
- Have the simulator ready for testing (all functionality ready)

**Sprint result:**



“We finished the demo test level and got the simulator ready for the student tests. The process was challenging because it was important that the quality was ensured because the simulator is going to be tested on students from Lokførerskolen next Monday.”

|           | Wishlist                     | In Progress   | Done  |
|-----------|------------------------------|---|---|
| Issues    | Place and Edit Railways (28) | Write bachelor Thesis (30)<br><br>Game integration testing (30) | Basic Train cars (10)<br><br>Convert in-game menu to c++ (4)<br><br>Demo Test Level:<br>- Create Environment (4)<br>- Add train and wagon (1)<br>Add signal and triggerboxes (2)<br>- Add railway (2)<br>- Add station (1)<br>- Possesion switch between drone and train (2)<br>Fix camera possesion bug (4)<br><br>Delete editor objects (5) |
| Estimates | 28                           | 60  | 21  |

**Table 4.3:** Overview of issue status at the end of sprint 10

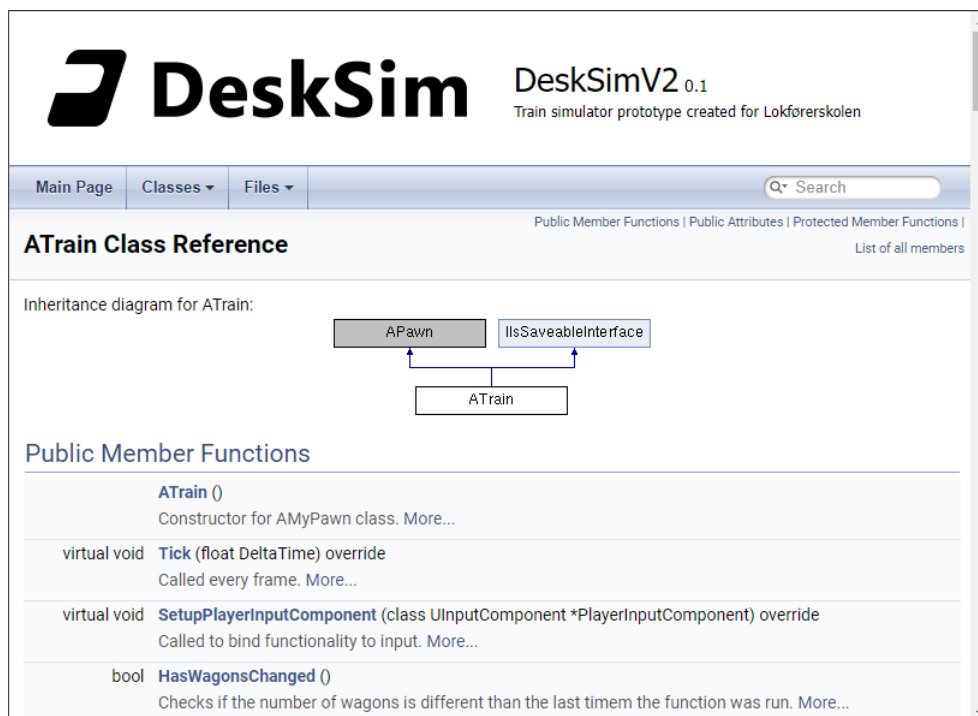
### Retrospective:

“Even though we only finished 21 of the story point estimates, we have spent much time on writing the final thesis. All tasks we set out to do is finished except the *Place and Edit Railway*-issue which got excluded from the project scope. This particular task was taking up a lot of development time due to the unforeseen difficulty of the task. After discussing internally and with the client, we concluded that all the functionality we were developing in the in-game editor mode were reflections of what offers in-engine. This made a hindrance to further development of editor functionality, which was then removed from the project, as the client agreed that it would be easier to utilize Unreal Engine itself for this functionality.

When planning the sprint we added an issue for game integration testing. This issue would contain all the bugs, errors and code mistakes we could find when trying to build and package the game. Since the scope of the issue increases when working on it, we decided to not include as much issues to this sprint to ensure that this issue gets the attention it needs.”

### 4.2.3 Documentation

To document the code, we had to follow Doxygen’s documentation standard. When the development concluded, we generated a folder of HTML-files, which can be hosted as a website to display the documentation. The location of the documentation is inside the /Documentation-folder in the project repository.

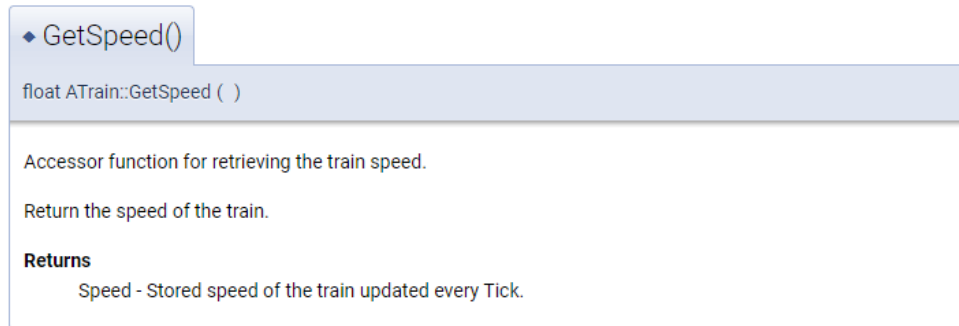


The screenshot shows the documentation page for the **ATrain** class. At the top, there is a logo for **DeskSim** and the text **DeskSimV2 0.1** with a subtitle "Train simulator prototype created for Lokførerskolen". Below this is a navigation bar with "Main Page", "Classes", and "Files" menus, and a search bar. The main content area is titled "ATrain Class Reference" and includes links for "Public Member Functions", "Public Attributes", "Protected Member Functions", and "List of all members". An inheritance diagram shows **ATrain** inheriting from **APawn** and implementing **ISaveableInterface**. Below the diagram, the "Public Member Functions" section lists:

- ATrain ()**: Constructor for AMyPawn class. More...
- virtual void **Tick (float DeltaTime) override**: Called every frame. More...
- virtual void **SetupPlayerInputComponent (class UInputComponent \*PlayerInputComponent) override**: Called to bind functionality to input. More...
- bool **HasWagonsChanged ()**: Checks if the number of wagons is different than the last timem the function was run. More...

**Figure 4.5:** The documentation page for the **ATrain** class

The documentation contains all files, classes, functions, and variables used in the project and presents the user with menus, lists, and a search bar for easy navigation. As shown in Figure 4.5, a class page displays all relevant information for a class. Figure 4.6 is captured further down on the same page, showing an example for a member function description.



**Figure 4.6:** The documentation for ATrain member function GetSpeed

#### 4.2.4 Deviation from the Original Plan

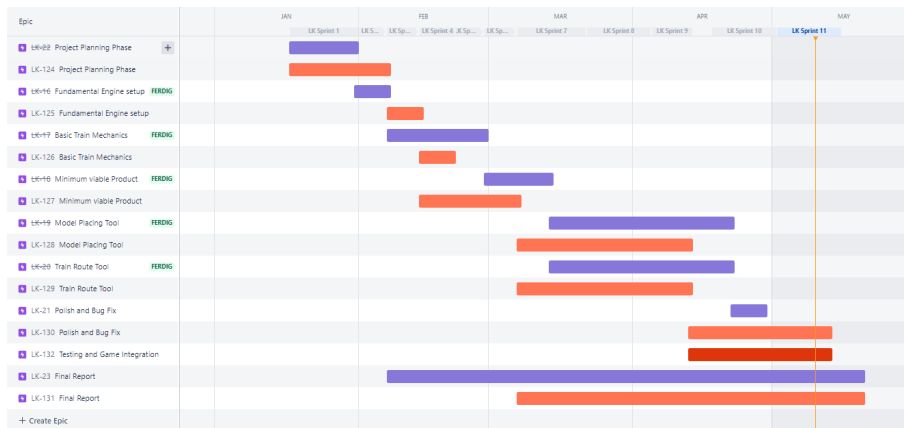
While developing the application, we encountered situations that forced us to deviate from our original plan. This section will explain the main deviations, and they will be addressed further in the discussion (chapter 10).

##### Deviations in Project Process Plan

Figure 4.7 shows the original milestones for the project and the actual project process within its period. As the figure shows, the planned progress was reasonably similar to the planned process. There were some delays at the beginning of the project. Some milestones were in progress simultaneously, such as basic train mechanics and minimum viable project. There was also a new milestone (Testing and Game Integration) we did not expect would take up as much time as it did.

##### Deviation in Estimation Process

As previously stated, the estimations were in T-Shirt sized estimations. We did not find these estimations valuable when working with Scrum. After reviewing the T-shirt-sized issues, the further estimation process was not more manageable because they had offsets such as one to three hours, one day, and two to three days. We wanted to change it to get an exact value we could utilize to set better and more realistic estimations.



**Figure 4.7:** Comparison of original project plan (purple) and actual project process (orange)

### Daily Scrum Meetings

The daily Scrum meetings worked well at the beginning of the project. All group members were sharing their experiences, letting the other group members know what progress had done the previous day, and making sure everyone knew the project's current status. After one month of the development process, these meetings began to feel forced and not very productive. They often went way over the time limit, and noting them down seemed like a waste of time. Therefore, it was decided to stop having these meetings as a forced, structural procedure. We did not stop having the meetings, but the group only noted them down when any relevant information needed to be addressed later in the development that had not already been addressed in the retrospective meetings.

### Sprint Lengths

The group started the project with a sprint length of one week. After sprint six, we decided to increase the length to two weeks because of the magnitude increase in the upcoming milestones with the model placing and train route tool. As previously stated in the retrospective for week six, this decision was to produce more functionality and give the group more time to produce code before evaluating it. The rapid time frame between development and planning also became more of an obstacle than a benefit. It began to halt the process by only allowing us to finish small sections of the functionality at a time.

The change in the sprint length also provided us with the opportunity to take a rest day or a research day. This day was allocated at the beginning of each sprint after sprint 7. This day was voluntary for each member of the group, and the different group members used the day differently. Some used it to continue working on the previous sprint, some used it to write about the implementations they had done in the previous sprint, and some took an extra day of rest.

## Chapter 5

# Technical Design

This chapter describes the design and choices behind the technological systems included in the application. The development environment is based around Unreal Engine and C++ .

### 5.1 System Architecture

The code architecture is established around the object-oriented nature of C++ and patterns within game programming. Limited by the development environment, we had to implement all classes as derivatives from Unreal Engine's base classes. These classes enable standard functionality such as rendering, collisions, and mathematical operations such as translation and rotation. We continued this inheritance style, defining base properties for objects and deriving further into special classes if needed. A prime example of this is in the editor mode of the application. In this mode, each object in a level is treated as an *EditorObject*, including trains and railways. An *EditorObject* is an interactable entity in a level. These are manipulatable by the *EditorController*, which handles all user input and level manipulation.

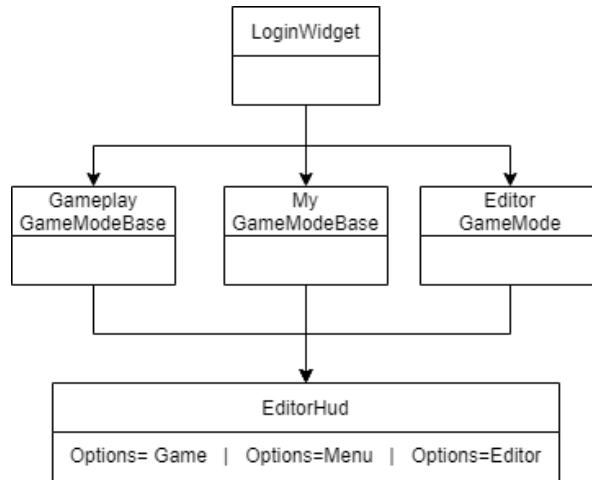
### 5.2 Application module Architecture

The application's hierarchy has been divided into four figures for clarity. The models show all the actors, pawns, widgets, etc, and tries to visualize their interconnectivity in the system.

Figure 5.1 shows the different game modes a level can open as and what HUD is used for the game modes. Both game mode and HUD will be explained in more detail in chapter 7 and section 7.10, respectively.

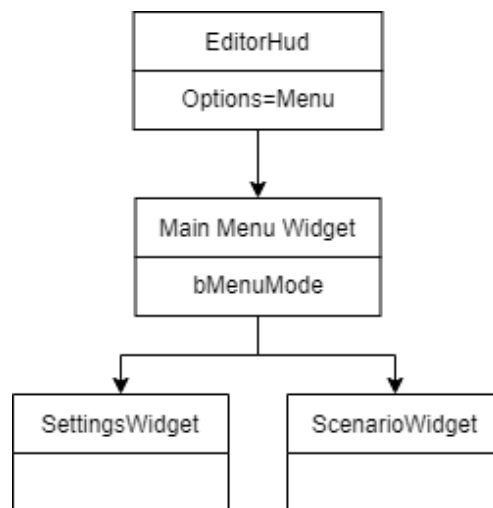
The application's hierarchy has been divided into four figures for clarity. These models show all the actors, pawns, widgets, and so forth, and they try to visu-

alize their interconnectivity in the system. Figure 5.1 shows the different game modes a level can open and what HUD is used for the different game modes. Both game mode and HUD will be explained in more detail in section 7.1 and 7.10, respectively in chapter 7.



**Figure 5.1:** Hierarchy of application start up functionality

An Options string is sent with the new game mode to decide what HUD elements should be displayed. This string could either be blank, which defaults to "Menu", "Game" to draw the train's DMI, or "Editor" to draw the editor UI elements. The three following figures display the hierarchy of these three *gamemodes*, respectively.



**Figure 5.2:** The module hierarchy for main menu mode

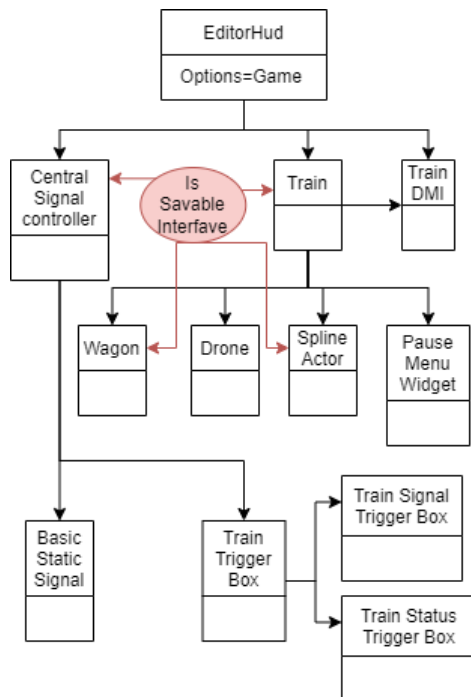


Figure 5.3: The module hierarchy for simulating mode

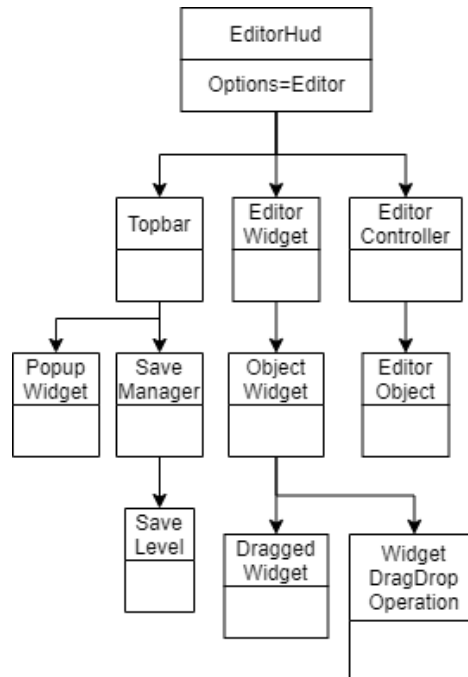


Figure 5.4: The module hierarchy for editor mode

### 5.3 Network Architecture

Authentication is currently the only system that uses the network. While multiplayer is a feature used in the existing DeskSim solution, it is not a part of this demo.

The authentication implementation uses the existing solution in use at Lokfører-skolen. Their system uses two endpoints. The first receives a username and password and returns a JWT (JSON Web Token) on success. The JWT is then sent to the second endpoint, which returns a JSON UserObject containing the user data (ID, Username, Roles, etc).

A login screen is presented to the user when the program launches. The user can then fill in their username and password to enter the application. The user must authenticate in order to use the application. A successful authentication causes the application to store the received information for the duration of the session. The occurrence of errors connected to the authentication process will trigger an error message in the viewport. Once the user authenticates, their info is stored for the duration of the session. The user-info is never stored in any file, which means the user needs to log in each time the application is restarted. Because neither the username, password, or token is ever stored locally, they cannot be extracted from local files to obtain private credentials.

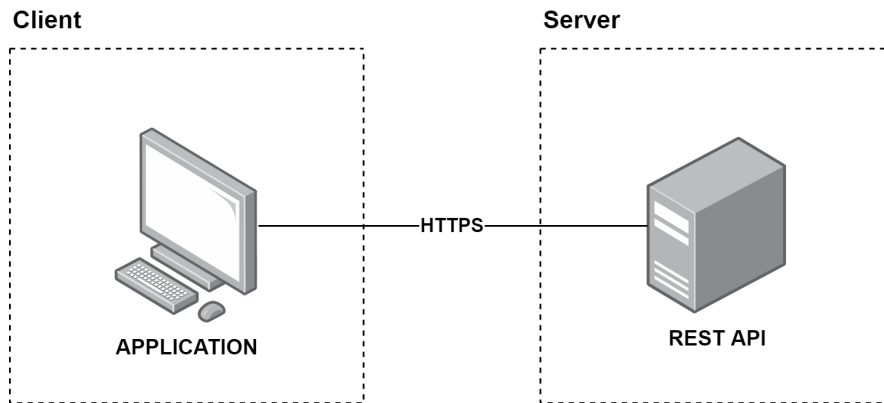


Figure 5.5: A diagram for the network architecture

### 5.3.1 File Structure

Unreal Engine separates files into two main folders, Source for code files and Content for asset files. We decided to structure the code files into folders based on their area of use in the context of the software. All other asset files were separated based on their type inside the Content folder.

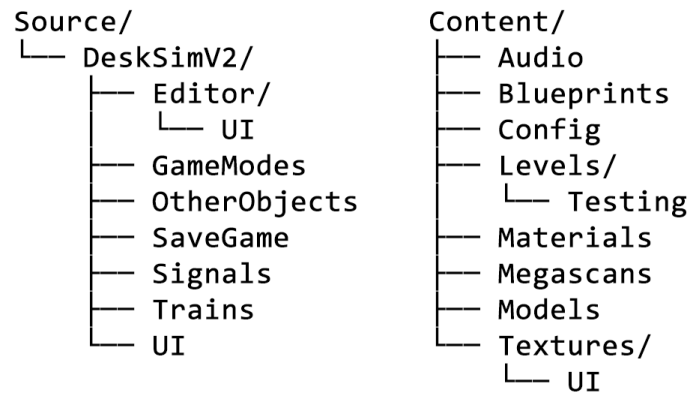


Figure 5.6: File hierarchy for Source and Content



## Chapter 6

# Product Overview

This chapter describes the product as seen from a user's perspective. It focuses on giving a holistic understanding of the simulator to provide context for further chapters.

DeskSim v2 is a functional train simulator and a building tool. The following section will provide a visual representation of both the simulator and the building tool to show how they are presented to the user. All texts and buttons are represented in Norwegian.

### 6.1 Menus

When the application has started, the log-in screen displays a primary user interface where users can enter their credentials to log in and gain access to the simulator. Users must authenticate to use the application since no guest mode exists. If there is an error when logging in, such as an empty field or wrong username and password, a small error text will be displayed to the user.



Figure 6.1: Log in screen

The main menu allows the user to select the scenario they want to run. If the user has admin or teacher privileges, they can start a level/scenario in editing mode as well, as is shown with the "Editor" button in figure 6.2. The scenarios separate into main categories displayed as tabs. The categories are the same as the ones Lokførerskolen uses in their existing simulator.

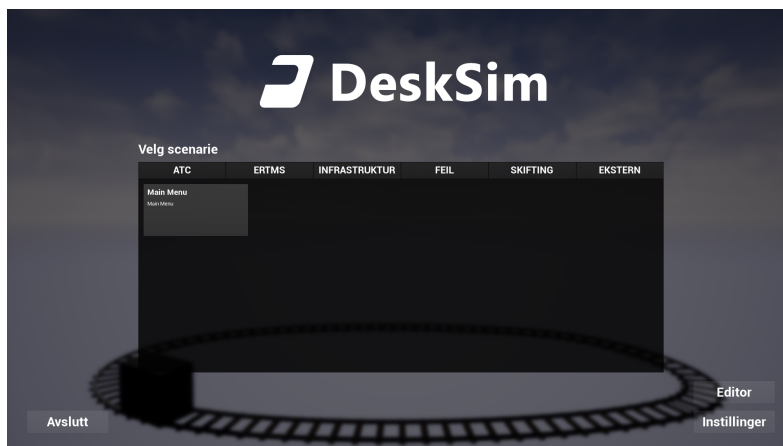


Figure 6.2: Main Menu

Pressing the "Innstillinger" button opens up a settings menu which allows the user to change the screen resolution and the window mode for the application. The settings menu is also available in the game's pause menu, allowing users to change these settings during scenarios.

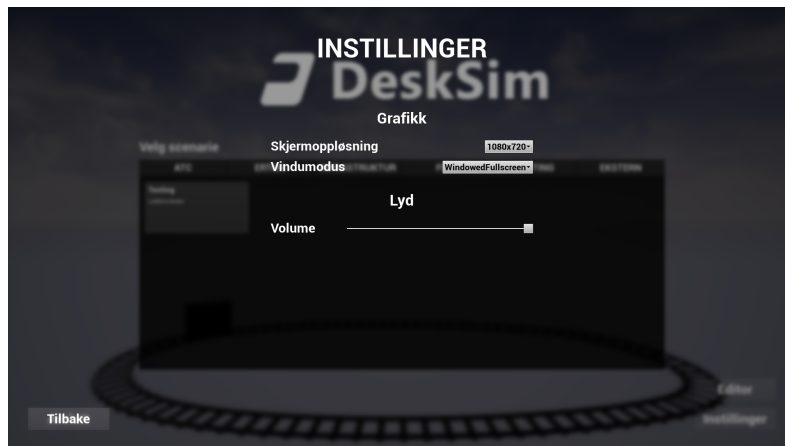


Figure 6.3: Settings Menu

## 6.2 Simulator mode

The in-game experience consists of a view that corresponds to a train driver's view from inside the front wagon of a train. The camera placement represents the train driver looking forwards. The speedometer at the top left of the screen presents the user with the train's current speed.

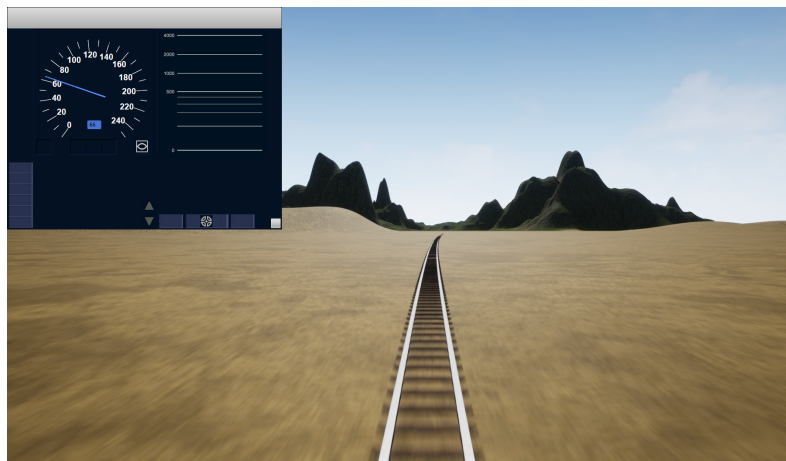


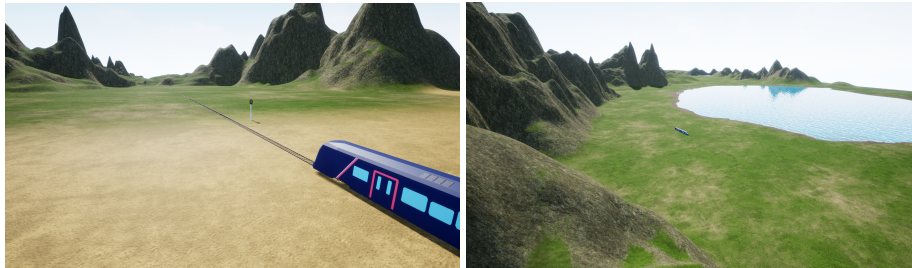
Figure 6.4: Train Driver View

The user will encounter different signals when simulating a scenario. The signal status and logic are pre-decided and controlled through invisible triggerboxes. These triggerboxes react to the train's position, check for certain conditions, and update signal statuses accordingly.



**Figure 6.5:** Stop Signal

The simulator allows the user to see the world in a drone mode, with free movement and camera rotation. The movement of the drone is independent of the train.



**Figure 6.6:** Drone View

### 6.3 Editor mode

The editor mode has buttons presented on the top of the screen, where the user can save their changes, change the current tool mode and delete objects. The editor mode interface also offers a window for adding new objects to the level. Adding objects is done by dragging the object's icon out of the content browser and dropping it into the level.

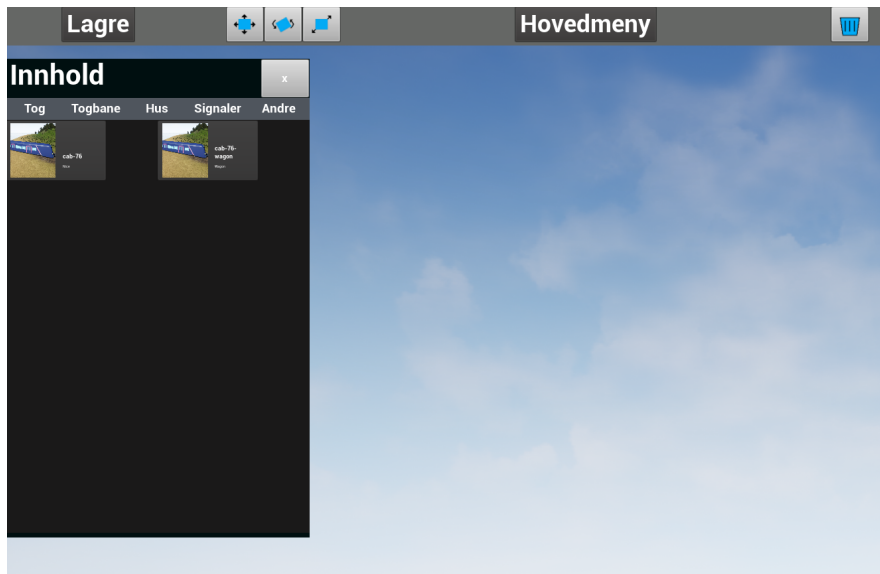


Figure 6.7: Editor Mode HUD

## Chapter 7

# Implementation

This chapter will look closer into the ideas, code, and structure behind various parts of the application. The project group members have written shown code except for the section about plugins. One crucial aspect of Unreal Engine is that the order of axes might be different than other industry standards, as Unreal Engine uses a *Left Handed, Z Up*-coordinate system.<sup>1</sup> In this three-dimensional space, the **X**-axis points forwards, the **Y**-axis points to the right, and the **Z**-axis points upwards.

When prototyping new features in the project development phase, it was common to do this using blueprints, allowing for rapid iteration of features without focusing on code. Once the functionality or feature works as intended, we convert the functionality of the blueprint into a native C++ class. The functionality is then re-created and can be further improved, and a new blueprint is created by deriving from the native C++ class. This implementation allows for the separation of functionality. The computational heavy section runs as native C++ code, while other info and variables such as models and materials are stored in the blueprint. This separation combines the best of both methods, namely the performance of native C++ with the flexibility of blueprints.

### 7.1 Game flow

The flow of the game is mainly handled within `EditorHUD.cpp`. The way Unreal Engine handles level switching is by opening levels based on case sensitive *FName* variables. Level changes happens when a user wants to start a scenario from the main menu, and change back to main menu. To decide if the application should open in editor or simulator mode there is a button in the main menu that allows you to change the mode of the main menu. The implementation of this is displayed below. All scenarios has a menu mode variable that gets updated when the button

---

<sup>1</sup><https://www.techartHub.com/a-practical-guide-to-unreal-engine-4s-coordinate-system/>

gets pressed. This, together with a change in the application layout, is the logic behind the applications ability to both play and edit a level.

The flow of the game is mainly handled within `EditorHUD.cpp`. The way Unreal Engine handles level switching is by opening levels based on case sensitive `FName` variables. Level changes in the application happen when a user wants to start a scenario from the main menu and change back to the main menu. To decide if the application should open in editor or simulator mode, a button in the main menu allows the user to change the game mode view the main menu is displaying. The implementation of this is displayed in code listing 7.1. All scenarios have a boolean menu mode variable that gets updated when the button gets pressed. The boolean variable change and a change in the application layout are the logic behind the application's ability to both play and edit a scenario.

```

1 void AEditorHUD::ChangeMenuMode()
2 {
3     bMenuMode = !bMenuMode;
4
5     // Changing the options for the Menu scenario
6     for (int i = 0; i < ScenarioWidgets.Num(); i++)
7     {
8         ScenarioWidgets[i]->SetMenuMode(bMenuMode);
9     }
10    // Changes layout for the menu
11    MainMenuWidget->ChangeLayout(bMenuMode);
12 }

```

**Code listing 7.1:** Changing menu mode

The game mode is set to its default on level openings if not specified in the option string. Unreal Engine does not yet have a visually appealing approach for handling game mode changes during runtime, and this statement is supported and discussed by the community [41]. The negative aspect of this is that creating the functionality for this may, in some cases, look a bit messy, as shown in the code listing 7.2.

```

1     FString GameMode;
2     GameMode = "?Game=/Game/Blueprints/UI/CDerivedEditor/BP_EditorGameMode.
3         BP_EditorGameMode_C";
4
5     UGameplayStatics::OpenLevel(GetWorld(), MapNameReference, true, GameMode);

```

**Code listing 7.2:** Changing game mode at runtime.

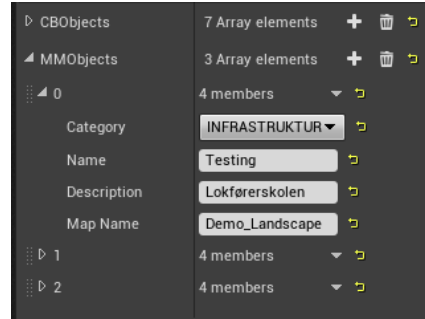
Another critical aspect of the game flow is how the level structure fits further development. When developers have created a new level, they can easily add the level to the main menu through a structure located in the `EditorHUD` blueprint. This implementation works because the blueprint is derived from the `EditorHUD.cpp` file and using the macros `USTRUCT` and `UPROPERTY` (shown in code listing 7.3) exposes the variable to the editor and adds it to Unreal Engines memory system. [42]

```

1  USTRUCT(BlueprintType)
2  struct FMMObjectStruct
3  {
4      GENERATED_BODY();
5
6      UPROPERTY(...)
7      ECategoryMM Category = ECategoryMM::
          ATC;
8      UPROPERTY(...)
9      FString Name;
10     UPROPERTY(...)
11     FString Description;
12     UPROPERTY(...)
13     FName MapName;
14 };
15 UPROPERTY(...)
16 TArray<FMMObjectStruct> MMObjects; ///<
    All objects
17 }

```

**Code listing 7.3:** UPROPERTY macro.



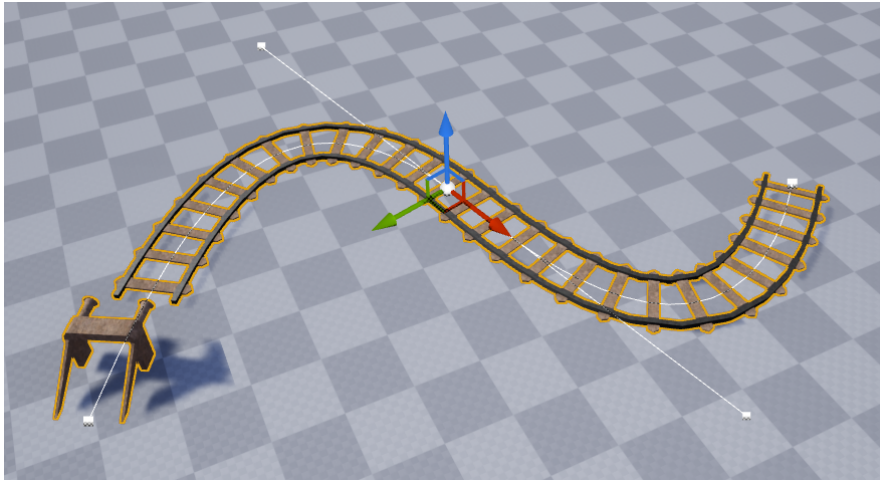
**Figure 7.1:** The module hierarchy for simulating mode

## 7.2 Spline Tool

One of the first goals specified by the client was to move the train along a straight path. Seeing as a later goal was also to implement curvature, we decided that it would be beneficial to begin work on curved train movement right away, as this would likely save us some time. The railway is procedurally generated from a list of points forming a *spline* in the existing simulator. A *spline* can be defined as a mathematical function to interpolate and form a smooth curve between multiple points. Each point is made up of a location vector and a tangent vector for specifying the curvature of the spline at the given position. We also looked into *Bézier curves*, a similar alternative to splines, but Unreal Engine’s built-in spline component made it an obvious choice.

The created `SplineActor.cpp`, is a class for handling all functionality related to the railway. This class contains a *USplineComponent*, a powerful component with functions for placing and deforming a mesh along a spline. In this context, a mesh can be described as a 3D-model. The component initially works by looping over all points in the spline, placing a sub-mesh that is stretched between points  $P_n$  and  $P_{n+1}$ , and bending the meshes by the curvature of the tangents between  $T_n$  and  $T_{n+1}$ . The spline points were defined by the user in the editor, and could be placed freely. The results looked promising, but a major flaw of this method occurs when any distances between each set of spline points aren’t uniform. The meshes would stretch differently along the spline, making the railway look rather odd. A good analogy to this anomaly is imagining a skyscraper with a single window, stretched horizontally along the building instead of separate windows on each





**Figure 7.2:** A selected spline deforming a railway mesh along itself. The white squares indicate the spline points, and the white lines indicate the tangent of the selected spline point.

floor. The solution to this was to split the spline into segments of equal length to that of the mesh/model itself, and add a new sub-mesh at each segment. Essentially cutting up the railway into pieces and replacing the existing spline points. The mesh used in figure 7.2 is a small slice of a railway created with primitive shapes in *Asset Forge*, a third party software.<sup>2</sup>

```
1 const FVector MeshSize = Mesh->GetBoundingBox().GetSize();
2 const float MeshLength = MeshSize.X;
```

**Code listing 7.4:** Getting the length of the mesh

After retrieving the length of the mesh in it's forward axis, we split the spline into segments based of this length. This segment splitting lets us iterate through all the segments in the spline.

```
1 const int SplinePoints = FMath::CeilToInt(SplineLength / MeshLength);
2
3 for (int PointCount = 0; PointCount < SplinePoints; PointCount++)
4 {
5     ...
```

**Code listing 7.5:** Calculating spline segments

For each spline point along the spline, we create a new sub-mesh. Code listing shows the position and tangent calculation for each sub-mesh in the spline.

```
1 USplineMeshComponent* SplineMesh = NewObject<USplineMeshComponent>(this,
2     USplineMeshComponent::StaticClass());
```

<sup>2</sup>Asset Forge, <https://assetforge.io/>

```

3   const float StartDistance = MeshLength * PointCount;
4
5   FVector StartPoint = GetLocationAtDistanceAlongSpline(
6       StartDistance, ESplineCoordinateSpace::Local
7   );
8
9   FVector StartTangent = GetDirectionAtDistanceAlongSpline(
10      StartDistance, ESplineCoordinateSpace::World
11  );

```

Code listing 7.6: Initialization of spline points

These values are also calculated for the position and tangent of the end for each sub-mesh, before applying the vectors to the generated sub-mesh in code listing 7.7.

```

1   SplineMesh->SetStartAndEnd(
2       StartPoint,
3       StartTangent,
4       EndPoint,
5       EndTangent
6   );

```

Code listing 7.7: Applying spline point data

Opting for a custom spline class instead of Unreal Engine's own solution let us create a railway from a spline. We added specific elements such as *buffer stops*, and the metal bars at the end of a railway to stop trains, as seen in figure 7.2. We added an optional parameter to the `SplineActor.cpp`, where the user can input a separate mesh to be displayed as the first and the last sub-mesh of the spline. However, if this field is left empty, the sub-mesh will default to the given railway mesh. We also implemented a boolean flag which is set off if the spline extends a desired angle or curvature to ensure the train is looking natural when traversing the railway.

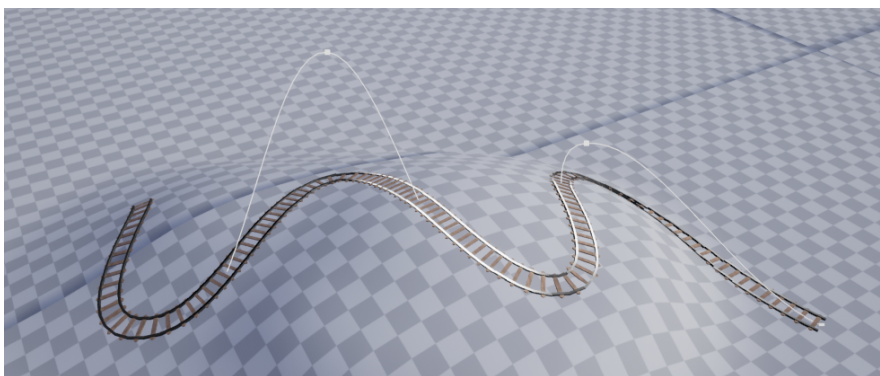


Figure 7.3: A railway conforming to the terrain height

One of the challenges with the tool occurred when manually placing the spline points. The railway curved naturally in the horizontal plane, but if the terrain had

any height difference, the railway would disappear under the ground or hover above the ground. To solve this, we perform a line cast in each spline point's X- and Y-coordinates, from the top to the bottom of the level. This is a way to see if we hit any terrain along with the line cast. If there is a hit, the Z-coordinate of the relevant spline point is set to the height value of the terrain hit. The line casting gets more computationally expensive the longer a spline is but is only run once when a spline changes, so any noticeable effects are minuscule.

### 7.3 Signal controller

The *CentralSignalController* class is responsible for signal and status communication between the signals and trains. The controller receives signal and status updates from either *TrainSignalTriggerBox* or *TrainStatusTriggerBox*, which are placed along the railway and are used to send signal or status updates depending on configured conditions.

When the level is loaded, and the game begins, the *CentralSignalController* searches for all actors inheriting from the *BasicSignal* class. Valid actors are sorted into lists based on which their types, such as *MainSignal* or *ForwardSignal*. These different lists are then used later to send signal updates.

```

1 void ACentralSignalController::FindAllSignals()
2 {
3     // Finds all actors of signal classes
4     UGameplayStatics::GetAllActorsOfClass(GetWorld(), DwarfSignalBP,
5         DwarfSignalActors);
6     UGameplayStatics::GetAllActorsOfClass(GetWorld(), MainSignalBP,
7         MainSignalActors);
8     UGameplayStatics::GetAllActorsOfClass(GetWorld(), ForwardSignalBP,
9         ForwardSignalActors);
10
11     int32 i = 1;
12
13     // Loops through all actors
14     for (AActor* DwarfSignalActor : DwarfSignalActors)
15     {
16         ABasicSignal* DwarfSignal = Cast<ABasicSignal>(DwarfSignalActor);
17         if (DwarfSignal)
18         {
19             FString TagName = FString::Printf(TEXT("Dwarf_%i"), i++);
20             DwarfSignal->Tags.Add(FName(TagName));
21             AllSignalActors.Add(DwarfSignalActor);
22         }
23     }
24 }

```

Code listing 7.8: Finds and stores all signals based on class

When the controller receives a signal update, an new ID, signal state, and signal type must be specified. The controller then searches through the specified signal

list to match the IDs, and when this happens, the new signal state is sent. If more than one signal shares the same ID, the controller continues searching the list. . Given that the list of signals is relatively short, the time spent searching the list is insignificant even when the first match is found.

```

1 void ACentralSignalController::SendUpdatedSignal(FName SignalID, ESignalType
  SignalType, ESignalStatus Status)
2 {
3     TArray<AActor*> Actors;
4
5     // Selects relevant actors to search based on signal type
6     switch (SignalType)
7     {
8     case ESignalType::Main:
9         Actors = MainSignalActors;
10        break;
11    case ESignalType::Forward:
12        Actors = ForwardSignalActors;
13        break;
14    case ESignalType::Dwarf:
15        Actors = DwarfSignalActors;
16        break;
17    default:
18        checkNoEntry();
19        break;
20    }
21
22    // Loops through all signals
23    for (AActor* SignalActor : Actors)
24    {
25        ABasicSignal* Signal = Cast<ABasicSignal>(SignalActor);
26        if (Signal && (Signal->ID == SignalID))
27        {
28            // If the signal matches the incoming id, update the status
29            Signal->UpdateSignalStatus(Status);
30        }
31    }
32 }

```

Code listing 7.9: Updates signals based on ID and type

As the Central Signal Controller is an actor, it needs to be placed in every level that uses signals. One alternative would be to create a subsystem with the same functionality. This alternative would allow the controller to both, work in any level without placing an actor, and having a more predictable and stable lifecycle.

## 7.4 Signals

The class *ABasicSignal* is used for the three different types of signals. The three signal types share the same functionality with the exceptions of the models used, the number of lights used in each model, which colors are used, and how the signal reacts to signal status updates.

Each signal type is stored as a blueprint that stores the model used and signal-type

specific information such as what colors are used and how the signal reacts to signal status updates. This way, only one C++ code file is used for all signals, making it easier to implement new features, fix bugs, and improve the maintainability

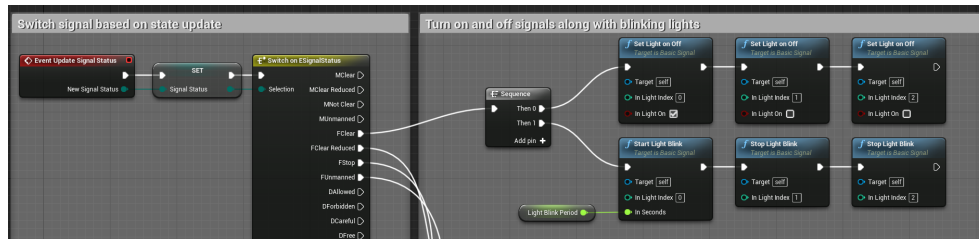


Figure 7.4: Blueprint showing how a signal switches status

Because the different signals use different models, hard-coding the positions of the signal lights is not an option. Instead, sockets are placed on the models in the Unreal Engine Editor, which dynamically places signal lights at runtime. After a level has loaded, spheres are created and placed according to the sockets on the model. A standard naming convention is used for sockets and components. The socket names need to match the socket names on the static mesh model in order for the positioning to be correct.

When components are created at runtime it is essential to register them manually. Otherwise, the components will immediately be garbage collected and disappear. When components are created in the object's constructor, they are automatically registered.

```

1 void ABasicSignal::SetupLight()
2 {
3     RemoveLights();
4
5     // Create all lights in a loop
6     for (int32 i = 0; i < NumLights; i++)
7     {
8         FString SocketName = FString::Printf(TEXT("Socket_%i"), i + 1);
9         FString LightMeshName = FString::Printf(TEXT("LightMesh_%i"), i + 1);
10
11         UStaticMeshComponent* LightMesh = NewObject<UStaticMeshComponent>(this,
12             FName(LightMeshName));
13
14         // Creates a new light struct
15         FSignalLight NewLight(LightMesh);
16
17         NewLight.LightMesh->SetStaticMesh(BaseLightMesh);
18
19         NewLight.LightMesh->SetupAttachment(VisualComponent, FName(SocketName));
20
21         // Creates the instanced dynamic light material
22         NewLight.DynMaterial = UMaterialInstanceDynamic::Create(BaseLightMaterial,
23             this);
24
25         NewLight.DynMaterial->SetScalarParameterValue("Emissive_Strength",
26             MaxLightLevel);

```

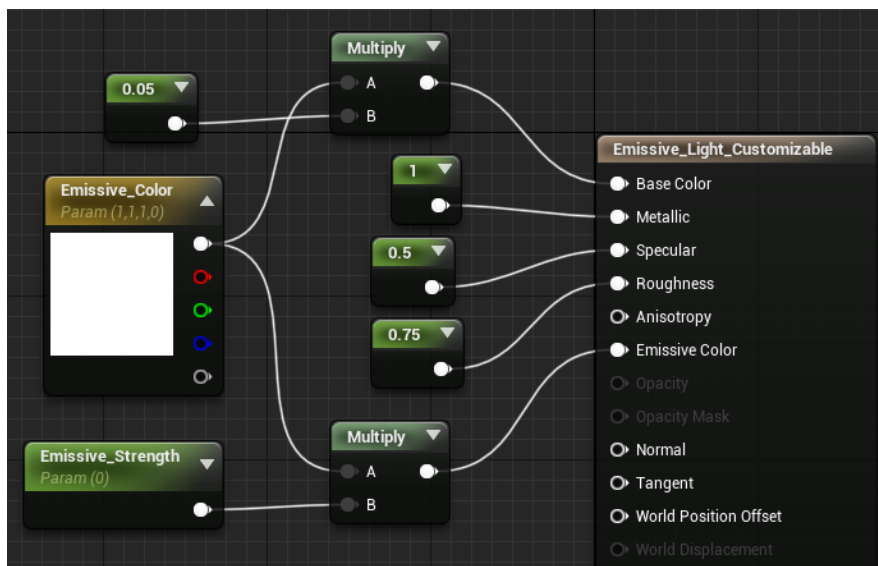
```

24     NewLight.DynMaterial->SetVectorParameterValue("Emissive_Color",
25         FLinearColor(1.0f, 1.0f, 1.0f));
26
27     NewLight.LightMesh->SetMaterial(0, NewLight.DynMaterial);
28
29     NewLight.LightMesh->RegisterComponent();
30
31     Lights.Add(NewLight);
32 }
33 }

```

**Code listing 7.10:** Creates dynamic instanced materials for each signal light

A new Instanced dynamic material is created for each sphere, which uses emissive lighting for the signal lights. The new dynamic instance of the material allows material parameters to be changed during runtime for each different instance, ensuring the lights can change independently of other light signals of the same type.



**Figure 7.5:** The material used for signal lights

The material used for the signal lights is created from scratch in Unreal, using its Material Editor tool. The material has two public parameters, which are used to change the color of the material and the brightness of the Emissive light. The material uses the same color for both base and emissive colors. The base color has a constant low brightness, which is used for the off state. A parameter determines the emissive color strength. All values used range from 0 to 1, except emissive light strength, which can be any value above 0. Higher values result in brighter emissive lights. Currently, the entire model using the material lights up as an emissive light source, but a texture could be used to only use parts of the model as an emissive light source.

## 7.5 Login and authentication

The *ULoginWidget* class contains the functionality to log in, which uses the *VaRest* plugin to encode, decode, send, and receive web requests. The user data is stored in a struct in *UDesksimGameInstance*. This class that is instantiated at the beginning of the application and lives until it is closed, ensuring the user data is available for the entire duration of the application.

Our application connects with the existing login solution in use at Lokførerskolen. Two endpoints are used, where the first takes a username and password and returns a JSON Web Token (JWT) on success. The second endpoint takes this JSON Web Token (JWT) and returns a user object containing various data. This data is then decoded and stored in the gameinstance class, *UDesksimGameInstance*. The application does not verify the JSON Web Token (JWT). It just passes the token on to the following endpoint. While the JSON Web Token (JWT) is valid for a long duration, it is not stored in any files, and a new token is requested each time the user logs in.

```

1 void ULoginWidget::ReadUserData(UVaRestJsonObject* JsonObject)
2 {
3     UDesksimGameInstance* GameInstance = Cast<UDesksimGameInstance>(GetGameInstance
4         ());
5     GameInstance->bIsLoggedIn = true;
6
7     // Set Userinfo in gameinstance
8     GameInstance->UserInfo.bIsAuthenticated = true;
9
10    GameInstance->UserInfo.bIsEnabled = JsonObject->GetBoolField(TEXT("isEnabled"));
11
12    GameInstance->UserInfo.id = JsonObject->GetIntegerField(TEXT("id"));
13
14    GameInstance->UserInfo.Username = FName(*JsonObject->GetStringField(TEXT("username
15        ")));
16
17    GameInstance->UserInfo.SubscriptionID = JsonObject->GetIntegerField(TEXT("
18        subscriptionId"));
19
20    // Find info about subscription and user groups
21    UVaRestJsonObject* Subscription = JsonObject->GetObjectField(TEXT("subscription"))
22        ;
23
24    TArray<FString> GroupNames = Subscription->GetStringArrayField(TEXT("
25        userGroupNames"));
26
27    // Checks which type the user is
28    if (GroupNames.Contains("Administrator"))
29    {
30        GameInstance->UserInfo.UserType = EUserType::Admin;
31    }
32    else if (GroupNames.Contains("Ansatte"))
33    {
34        GameInstance->UserInfo.UserType = EUserType::Teacher;
35    }
36    }

```



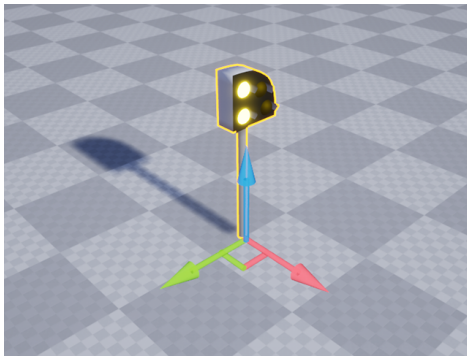
```
33     else //if (GroupNames.Contains("Student"))
34     {
35         GameInstance->UserInfo.UserType = EUserType::Student;
36     }
37 }
```

**Code listing 7.11:** Decodes and stores info from userinfo JSON response

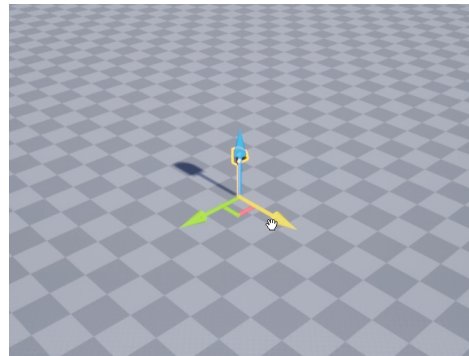
The login functionality uses the VaRest plugin to handle common web-related tasks like encoding and decoding JSON objects. The plugin was chosen to simplify the use of REST communication.

## 7.6 Editor Gizmo

The goal of the editor mode was to give the client a simple tool for editing and train scenarios. Our task was to implement the ability to place, move and rotate objects such as trains and buildings and a way to lay a railway across the terrain. We decided to design this tool based on the tools available in Unreal Engine's editor, as these would accomplish similar tasks. We created a gizmo for moving objects around, consisting of one arrow for each axis, as seen in figure 7.6. A square was added at the bottom for moving the object in the plane along the X- and Y-axes simultaneously. Once an object is selected, this gizmo appears at its position, utilizing custom depth rendering, which enables the gizmo always to be visible even when another mesh covers it.



**Figure 7.6:** The translation gizmo on a selected signal



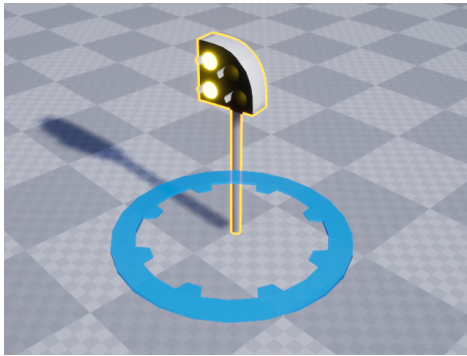
**Figure 7.7:** The translation gizmo far away, hovering the X-axis arrow

The user can move the object in the relevant axis by hovering the mouse over one of the gizmoes arrows and dragging. This is done by casting a line from the camera in the direction of the cursor's position on the screen and processing the results on hit. This is done in a separate collision layer for gizmoes, such that the line is not obscured by any object. The selected arrow will change colors to indicate interaction, as seen in figure 7.7. The tool includes two modes, translation, and rotation. When switching mode to rotation, the arrow meshes are replaced by a

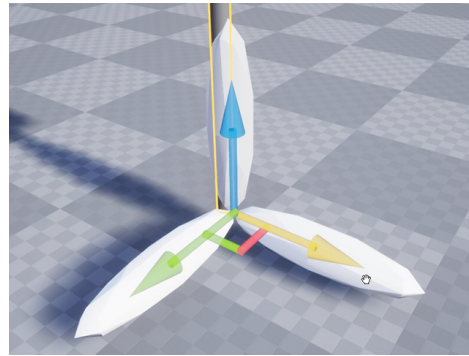


cogwheel in the XY-plane, as seen in figure 7.8, which rotates the object around the Z axis when dragged. These gizmos are made to be a part of the user interface, and get scaled based on their distance from the camera to keep their size uniform.

When grabbing a gizmo arrow to move an object, the cursor's position is saved as a variable in the first frame of holding down the mouse button. This position is used to get the offset between the gizmo's center position (also the object anchor position), and the cursor. While dragging the gizmo arrow, the object's position is set to the location of the cursor minus the offset, resulting in the object moving in parallel with the cursor. This behavior works fine in theory, but we found a lack of user friendliness during testing due to the cursor being required to overlap the gizmo during dragging. To combat this, we added secondary meshes to each arrow that would inflate the area of interaction. These meshes were given an invisible material, so the user could not see them but was able to interact with them. For demonstration purposes, these secondary meshes were given a visible material in figure 7.9.



**Figure 7.8:** The rotation gizmo in the XY-plane



**Figure 7.9:** Additional meshes around the arrows

During user testing with the client, it was made clear that the move object interaction was still hard, as the cursor would, more often than not, exit the area of these meshes. This was towards the very end of the development, and there was not enough time to iterate on this implementation. Nevertheless, due to the curious nature of programmers, we were attracted towards finding a solution to this problem, even if we did not have the time to finish the implementation of these ideas. We came up with several possible solutions that could fix or recreate this system entirely.

One such idea was to spawn large collision planes when the mouse is dragging an arrow at the object's position. If the user moves the object, the plane should stretch like a wall along the relevant axis. It should not be visible to the player but act as a giant safety net, such as the ones seen at golf courts, to capture the depth of the mouse cursor on the relevant axis. This implementation would provide a more intuitive movement system, where the user could drag the object to its desired position without keeping the cursor over the gizmo at all times. The plane would

then be removed when the user lets go of the mouse button. Another idea for a solution would be to take the cursor's two-dimensional position on the screen and rotate the vector based on the angle between the camera and the object. This implementation may also result in clunky movement, as 2D does not always translate well to 3D when forced.

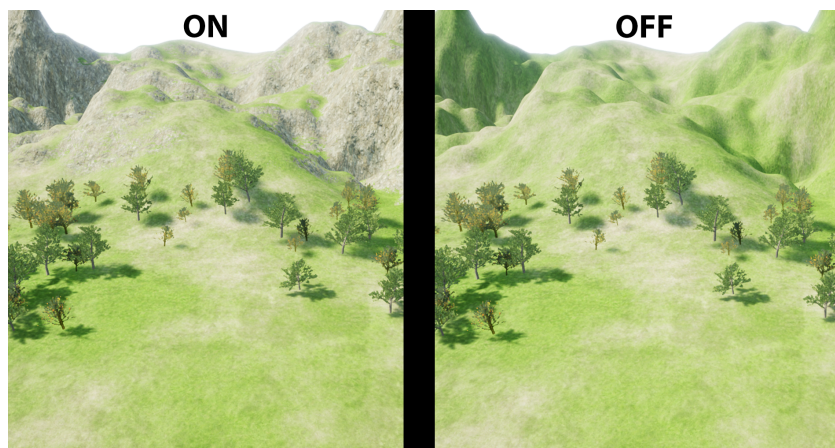
## 7.7 Landscape

All of the landscape is created using the Unreal Engine landscape editor. We tried importing height-maps at first to generate the landscape automatically, which is straightforward if the height map is of good quality. However, most of the height-maps we found had too significant height variations for a train track to be placed, so we decided to make the landscape using the landscape editor.

We originally planned to create a runtime terrain editing tool for the simulator, similar to the landscape editor in Unreal Engine. Although it became evident that such functionality would take a considerable amount of time and was therefore taken out of the project scope. This will be further discussed in chapter 10.

## 7.8 Landscape Texturing

We created our own landscape texture by combining multiple textures from the Megascans Library[43] inside the material editor in Unreal Engine. The landscape texture we created has multiple features such as a blend between the grass and rock texture, which is based on the angle of the texture. This texture is done to create realistic transitions from grass to the cliffs when the landscape gets steeper.



**Figure 7.10:** Texture is blended based on angle of the landscape

There was also implemented some texture variations to create a more realistic texture to mimic more depth in the environment.

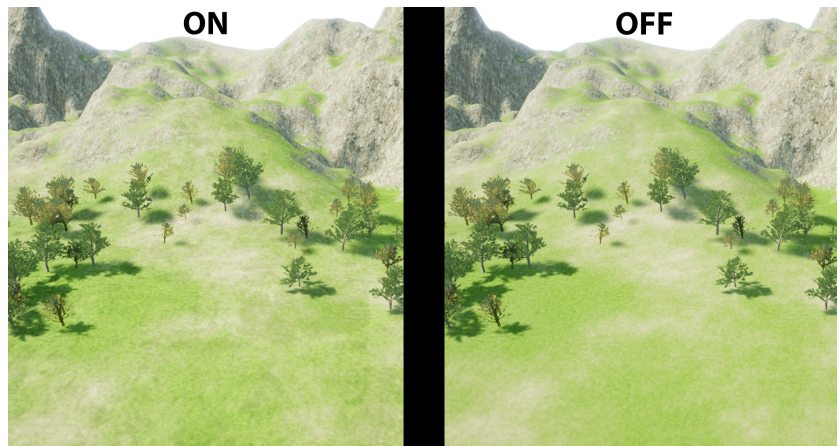


Figure 7.11: Added texture variations to create realism

To increase performance distance blending was added to the textures. Depending on the distance from the camera to the texture a different texture will be used. The closer the camera gets to the texture the more detailed the texture becomes.

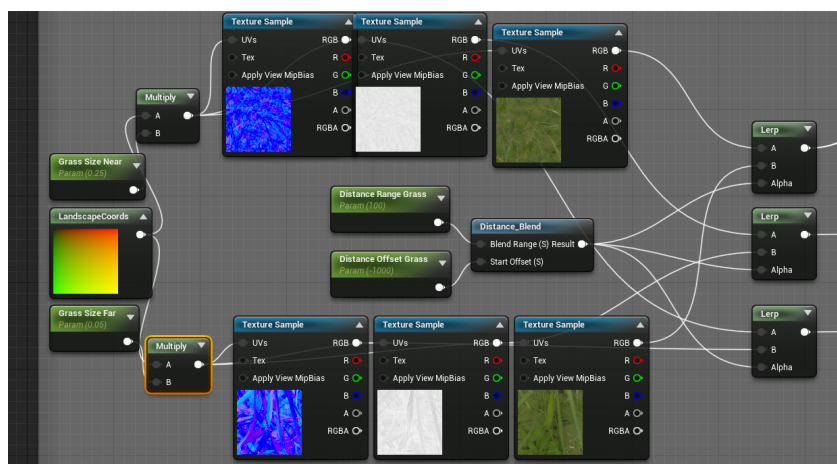


Figure 7.12: Texture details depends on the distance from the camera

## 7.9 Save Functionality

The concept of saving a game can vary from game to game, but the idea is to store contextual information persistently. An example of this is allowing users to quit and re-open a game, resuming progress where they last left off. In this project, we have implemented an editor mode where the player can edit and change the scenario they are playing. But for the editor mode to serve its purpose we need to be able to save the changes we made to the level so we can open it in simulator mode and simulate a train scenario with the saved changes.

The *USaveLevel* class inherits from UE's *USaveGame* class which is used to preserve information across multiple play sessions. We mainly save the transformation of placed or edited objects in the scene, as well as identifiable data such as actor name, class or level name into a struct *FActorData*. All the information from the saved actors are stored inside an array in a *.sav* file which is the save file format used by Unreal Engine.

```

1     ...
2     TArray<FActorData> SavedActors;
3     ...

```

**Code listing 7.12:** FActorData array that holds all data that is saved

An issue we encountered handling which actors to save, since some actors in a scenario may remain unedited, and would be unnecessary to save persistently. We explored different options, one of which was to create a base actor class which every savable actor should inherit. This was not possible, as some actors already inherited from different classes. Instead we created an interface the actors can inherit from if they are savable, giving every class the ability to be saved regardless of their base class.

An example of how a class can be made savable by inheriting the interface in the class declaration is shown in code listing 7.13.

```

1 UCLASS()
2 class DESKSIMV2_API ATrain : public APawn, public IISaveableInterface
3 {
4     ...

```

**Code listing 7.13:** Example of a class that inherits *ISaveableInterface*

All the save functionality is handled by the *SaveManager* which has two main functions, *SaveGame* and *LoadGame*. *SaveGame* handles all the functionality related to saving the current game state into a *.sav* file by checking for any actors that inherits the *IISaveableInterface* interface.

```

1 void ASaveManager::SaveGame()
2 {
3     ...
4     // Loop through ALL Actors in Scene
5     for (TActorIterator<AActor> It(GetWorld()); It; ++It)
6     {
7         // Checks if the Actor inherits from IISaveableInterface
8         if (It->GetClass()->ImplementsInterface(UIISaveableInterface::StaticClass()
9             ))
10        {
11            AActor* Actor = *It;
12            FActorData ActorData;
13            ...
14
15            // Creates a memorywriter and archive for Actor Data

```

```

16     FMemoryWriter MemWriter(ActorData.ByteData);
17     FObjectAndNameAsStringProxyArchive Ar(MemWriter, true);
18
19     // Only Saves Variables with UPROPERTY(SaveGame)
20     Ar.ArIsSaveGame = true;
21
22     // Serializes Actors UPROPERTIES into binary
23     Actor->Serialize(Ar);
24
25     SaveGameInstance->SavedActors.Add(ActorData);
26 }
27 }
28 ...
29 }

```

Code listing 7.14: SaveGame function from SaveManager

The *LoadGame* function in *SaveManager* reads and loads data from a .sav-file with a matching file name to the name of the loaded level. Then it updates the data of the existing actors with the matching data from the save file based on the actors name, which is found through the custom function *FindActorByName*. If it doesn't find the actor from the savefile in the level it will create a new actor with the data from the save file.

```

1 void ASaveManager::LoadGame()
2 {
3 ...
4     // Iterate through all actors in save and point to their corresponding
5     // actor in scene
6     for (FActorData ActorData : SaveGameInstance->SavedActors)
7     {
8         AActor* Actor = FindActorByName(ActorData.Name);
9         if (Actor)
10        {
11            // Updates Actor found in scene
12            Actor->SetActorTransform(ActorData.Transform);
13        }
14        else
15        {
16            FActorSpawnParameters SpawnParams;
17            ...
18            // Creates a new Actor with data from save
19            Actor = GetWorld()->SpawnActor(ActorData.Class, &ActorData.
20                Transform, SpawnParams);
21
22            // Creates a memoryreader and FArchive
23            FMemoryReader MemReader(ActorData.ByteData);
24            FObjectAndNameAsStringProxyArchive Ar(MemReader, true);
25
26            // Only Filters the Variables with UPROPERTY(SaveGame)
27            Ar.ArIsSaveGame = true;
28
29            // Converts serialized binary into Actors UPROPERTIES
30            Actor->Serialize(Ar);
31            ...
32        }
33    }
34 }

```

```

33     ...
34     }
35     ...
36 }

```

Code listing 7.15: LoadGame function from SaveManager

## 7.10 User Interface / Heads-Up Display

In this application the Heads-Up Display or HUDs is used to describe all user interface elements. It provides feedback to the user in the viewport.

The HUD consist of seven different user widgets. User widgets are Unreal Engine's components that allows for placement of UI elements in the viewport for 3D games. This section will explain the development of the logic which displays the user widgets.

In the first iteration of the user widgets they were created as blueprints. This meant that all of the widgets and its logic was created as a collection of nodes. According to the Unreal Engine forum, blueprints run between 10-40 percent slower than C++ [44]. This was one of the reasons we felt that it was necessary to convert the code from blueprint to C++ files. Another reason is that blueprint coding as a practise, does not allow us as students to display our skills in both programming and documentation to the same degree.

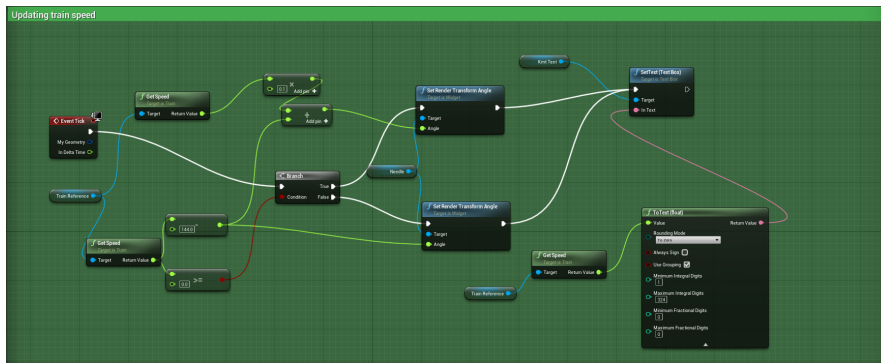


Figure 7.13: Update Speed function for the train DMI in blueprint

Figure 7.13 and code listing 7.16 contain the same functionality for updating the needle and text of the speedometer displayed to the user. All of the widgets were initially created as blueprints, but got converted to C++ at some point in the project.

```

1 void UTrainDMI::Update(float Speed)
2 {
3     if (Speed >= 0.0)
4     {

```



```

5     float RenderAngleDriving = (Speed * 0.1) + (Speed - 144);
6
7     int Text = FMath::FloorToInt(Speed);
8     FString SpeedText = FString::SanitizeFloat(Text, 0);
9
10    UpdateDMI(RenderAngleDriving, FText::FromString(SpeedText));
11  }
12  else
13  {
14
15    int Text = FMath::FloorToInt(Speed);
16    FString SpeedText = FString::SanitizeFloat(Text);
17
18    UpdateDMI(Speed - 144, FText::FromString(SpeedText));
19  }
20 }

```

Code listing 7.16: Update speed function in C++

The file `EditorHUD.cpp` handles which user widget is displayed to the user. What the HUD displays is based on the game mode the game is currently using, which is decided by which part of the application the user is currently in. If the user is in the simulator game mode, the train DMI will be displayed.

## 7.11 Plugins

The following functionality described in this section is not developed by us but needs to be included as it is used in the simulator.

Unreal Engine did not support the usage of two similar input devices using raw input. The solutions we found were to purchase and utilize a plug-in created by *Lemontmoon* called *VM Input Manager*<sup>1</sup> or to implement an input mapping system to our own application. We chose to utilize the plug-in in our application. We discuss this choice in 10.

The plug-in consists of 18 C++ classes and 31 blueprints. The benefit this had to our project was that we could manually map the USB input peripherals during runtime. The plug-in is used in the `BP_GameplayGamemodeBase` blueprint to spawn the necessary widget as a red dot in the top left of the screen.

The *VaRest* plug-in handles common tasks related to performing REST calls. It simplifies the process of working with requests and responses, by providing functions to create and send requests. While the plug-in is designed to be used in blueprints without any C++ code, all the functionality is available to be used in C++ . The same results could be achieved without the use of the *VaRest* plug-in, however it would require more work unrelated to the main goal.

---

<sup>1</sup>,

<sup>1</sup><https://www.unrealengine.com/marketplace/en-US/product/wm-input-manager>

## Chapter 8

# Deployment

This chapter will detail how the software can be deployed, both for development and release as a packaged product. It will also present how you compile the code to create and deploy documentation.

### 8.1 Packaging and release

Unreal Engine comes with a built-in package system. The system compiles source code, cooks content and builds the game. The game is then packaged as an executable file and game data is stored in .pak-files. The entire folder can be compressed and uploaded to a server where users can download, extract and play the game without any further installation. This is the simplest solution for deploying the finished product, but requires re-downloading the entire software when updates are packaged and released. A similar system is already in use at Lokførerskolen, where the software is downloaded in its entirety and updates requires a complete re-download. Due to its simplicity we will use this solution, however there are more advanced options in Unreal Engine if a better updating system is required.

One option to improve the update process is to create an auto-updater and automatically patch the game with new updates. Doing this would most likely require more work than it is worth if the game is not updated often. In UE it is possible to use clustering in the .pak files, which allows for easier streaming and patching of game data.

### 8.2 Setting up the project

The project uses Unreal Engine 4.27. The engine can be installed on the Epic Games Launcher, which is also used as a hub for various features related to UE. After the engine is installed, it can be used to open any Unreal Engine 4 projects.



Each UE project consists of some standard files and folders, most notably the `.uproject`-file which contains some general info about the project, like engine version, code modules and plug-ins used. User-made C++ code needs to be compiled before the editor can start, which is done by generating a Visual Studio solution for the project. Visual Studio is the default IDE to use with Unreal Engine, but other options are available. After the project is compiled for the first time, additional changes made during development can be compiled in the UE editor, which hot-reloads the changed files.

Plug-ins can be installed in two ways. The first and easiest method is to get the plug-in via Unreal Engine's marketplace for assets and install it to the engine. The second method requires some more work, but installs the plug-in in the project itself, allowing it to be shared between team members. This is a good way to share paid plug-ins within a project. Some manual work is required to set this up, mainly involving copying either compiled or source files to the proper directory in the project and adding files to source control to allow sharing.

There are three main directories shared in an Unreal Engine project, namely `Source` which contains all the C++ code written for the project, `Content` which contains all assets like materials, models and blueprints, and `Config` which contains extra configuration files used in various places. A `Plug-in` folder is also needed if any plug-ins are shared in the project. These are the essential folders to track using source control. During startup and use other intermediate folders are created, such as `Build`, `Binaries` and `Intermediate`. These intermediate folders are automatically generated and normally should not be included in source control.

### 8.3 Deployment of documentation

During the development we have used Doxygen for documenting code. It lets us compile the source folder into generated HTML-files with a formatted documentation of all comments. The resulting webpage is self-contained and features links between related classes and functions. We generated this using *Doxywizard*, which offers a graphical user interface for Doxygen. To regenerate this documentation, run Doxygen with `/Source/DeskSimV2` as the source code folder input. The program assumes all code to be commented using the Doxygen standard.<sup>1</sup>

---

<sup>1</sup>Doxygen Manual, <https://www.doxygen.nl/manual/docblocks.html>

## Chapter 9

# Testing

This chapter explains and elaborates the testing performed on the developed system.

### 9.1 Student User Testing

In collaboration with the client, we set up user tests towards the end of development. We visited Lokførerskolen in Oslo and borrowed three students who have previously used their current simulation software. The goal of user testing was to further iterate on the software based on given feedback, but we didn't have a lot of time as the testing was done two weeks prior to the deadline. The tests were done by prompting the user with a set of tasks. These were simple tasks, designed to test the user interface, software structure and overall user experience. The observer measured the time spent, while writing down notes from the user's performance. The user was also prompted with questions about the experience.

|                            |   |
|----------------------------|---|
| <b>Case 1:</b>             | Start the scenario named "Testing"  |
| <b>Preconditions:</b>      | The software is launched  |
| <b>Expected execution:</b> | The subject navigates to the correct tab, and clicks "Testing"  |
| <b>Actual execution:</b>   | The first two subjects started by navigating through the main menu tabs to search for "Testing", and clicked it once they saw it. The third subject clicked the correct tab as their first action, but the button did not respond, making the subject believe the scenario was not placed under this tab. This resulted in the subject launching the wrong scenario, making them exit the application to try again. |
| <b>Results:</b>            | The subjects complimented the look and feel of the interface. In a later discussion, they all agreed when one of the subjects said the interface was more user friendly than the existing simulator.  |

**Table 9.1:** User Test Case 1: Start the scenario named "Testing"

|                            |   |
|----------------------------|---|
| <b>Case 2:</b>             | Drive the train   |
| <b>Preconditions:</b>      | The subject is familiar with the controls from the existing simulator   |
| <b>Expected execution:</b> | The subject accelerates the train, riding through the whole map   |
| <b>Actual execution:</b>   | The subjects, being used to the old simulator, quickly resorted to the controls they were used to, and continued to operate the train.  |
| <b>Results:</b>            | This case was less structured and more focused on the general feedback from the subjects. We gave them freedom to explore the simulator while asking for their impressions. All three subjects commented on the train not properly following the tracks, but experiencing a slight delay when turning in curves, claiming that the camera falls a bit behind. This was most likely due to a last minute fix we implemented before testing, and will be discussed in chapter 10. |

**Table 9.2:** User Test Case 2: Drive the train

|                            |   |
|----------------------------|---|
| <b>Case 3:</b>             | Exit the game   |
| <b>Preconditions:</b>      | The subject has started a scenario  |
| <b>Expected execution:</b> | The subject presses the "escape"-button, then navigates to the main menu before exiting the game  |
| <b>Actual execution:</b>   | All subjects started by pressing the escape key, then clicking "main menu", and "exit".   |
| <b>Results:</b>            | This went smoothly, as if all subjects had done it before. We received compliments for the simplicity of the menu navigation, and praise for designing it in such a way that there is no need to exit the application when starting another scenario. |

**Table 9.3:** User Test Case 3: Exit the game

## 9.2 Employee User Testing

We also had a few tests with the client where we tested the editor mode, which is only available to employees at Lokførerskolen.

|                            |   |
|----------------------------|---|
| <b>Case 1:</b>             | Place and edit objects  |
| <b>Preconditions:</b>      | The subject has admin privileges and can open editor mode   |
| <b>Expected execution:</b> | The subject places an item from the editor menu and moves it  |
| <b>Actual execution:</b>   | The subject tried to drag and drop an item from the editor menu but nothing happened. After a while they clicked the item first then dragged it and it got placed in the level. They then moved the item in the level with some issues were the program stopped moving the item and they had to re select the object again. |
| <b>Results:</b>            | The subject managed to navigate the editor menu without issue, but struggled with placing and moving the object.  |

**Table 9.4:** User Test Case 1: Place and edit objects

|                            |   |
|----------------------------|---|
| <b>Case 2:</b>             | Save, exit and load level   |
| <b>Preconditions:</b>      | The subject have opened editor mode and placed an item in the level   |
| <b>Expected execution:</b> | The subject clicks save, then main menu twice and opens the "Testing" level   |
| <b>Actual execution:</b>   | The subject finds and clicks save, then clicks main menu and clicks on the warning. They then wait a bit before trying to click main menu again, which returns them to the main menu. Then they open the "Testing" level and sees the changes made. |
| <b>Results:</b>            | The subject had some issues were it took some time before they understood they had to click the main menu button again after being returned the warning prompt.   |

**Table 9.5:** User Test Case 2: Save, exit and load level

### 9.3 Hardware testing

During the visit to Lokførerskolen, we were able to deploy a build of the simulator on the client's hardware. These tests were conducted by confirming or disproving whether the operational requirements (3.2.2) could be fulfilled. We acquired a Windows computer, and successfully ran the simulator with over 60 frames per second.

### 9.4 System Testing

These are the system tests performed by the developers on the packaged product. It is important to note that the behaviour of a packaged project could be different to the same project executed in the editor. This caused some confusion initially for us, but also allowed the group to get a better understanding of the packaging process and how unreal handles memory in the editor as well as in a packaged project.

#### 9.4.1 System test 02.05.2022

To ensure the quality and resilience of our system some basic tests were performed on the system functionality. Thees tests were performed in accordance to the use cases conformed for the simulator.

When the correct password was typed in the main menu of the simulator got launched.

**Student use cases****Use case: Log in**

The log in functionality worked correctly and in accordance to the use case description. The first test was performed by writing wrong password and the system did not grant access to the program and allowed us to try authenticating again.

**Use case: Log out**

Logging out of the system worked correctly, when clicking the Log out button the, the log in screen got launched.

**Use case: Start scenario**

Clicking on a level in the blueprint to start the level worked and the program now gave the user the opportunity to control the train.

**Use case: Operate train**

Operating the train was handled exactly as expecting, but there was an issue when switching to the drone view which caused the application to crash. The issue did not give any relevant logs so we connected the application process to visual studios to try to see where the crash occurred. As it turns out, the code that updates the Driver-Machine Interface (DMI) is ran every Tick(). This is unnecessary use of computational power and a system handled delay was added to fix the issue. After updating this code the program ran as expected.

**Use case: Operate drone**

The drone camera did not spawn at the accurate position on the switch, and it turned out that this was because of how the drone referenced the train. In earlier iterations we linked the train to the drone through blueprint macros and UPROPERTY(). The problem with this was that it only works if the train don't get deleted and added to the level again. To ensure that this wont be an issue in the future, the drone got implemented to programmatic find the train actor when spawned in a level. This solution doesn't depend on manually assigning the train in the editor and would therefore work even if the two actors were to be removed and added again.

**Employee Use Cases****Use case: Place objects**

There was one issue found with this functionality and it was the ability to place two of the same actors in a level. After a review of the code, and some debugging of the system the issue was found. The drag and drop functionality actually drags the widget (which is a representation for the actors) out of its position and garbage collects it. Therefore, on the next occasion where drag and drop is performed on

the same widget it is no widget there to drag out. This was one of the examples where the editor version and the packaged version behaved differently.

The solution became to change to spawn a new widget when drag was detected and move this widget instead of the original one.

**Use case: Move or rotate objects**

The moving and rotating functionality in the editor mode worked as expected. There are one issue we are aware about and that is the buggy movement of objects.

**Use case: Delete objects** The delete object functionality worked as expected.

**Use case: Save scenario to file** The functionality for saving objects that gets added or deleted is working. This means that you could remove a signal in the editor and it would not be visible in the same level when driving the simulator.

## 9.4.2 Other issues found by testing the system

**Opening Pause menu** Opening the pause menu could sometimes cause the program to crash. The reason was an unhandled nullptr exception in the logic for spawning the pause menu. It would try to spawn the menu without it being created and therefore cause the program to crash. The only thing necessary to fix was a null check.

**Change resolution and window mode** when updating the desired resolution and window mode the actual settings would not open. The issue was that the code for actually applying the resolution and window mode shown in code listing 9.1, was not included in the functionality.

```
1 GEngine->GameUserSettings->ApplyResolutionSettings(false);
```

**Code listing 9.1:** Code displaying how to apply resolution changes

# Chapter 10

## Discussion

This chapter will discuss and reflect on the planning, development and administration of the project, as well as the product.

### 10.1 Evaluation of project goals

To evaluate the result of the final product, the project's initial goals should be taken into consideration.

#### 10.1.1 Main goals

The main goal of the project was to create a demo scenario with the following features:

**Must include at least one train, two signals one train-DMI a train track and a simple landscape.**

As shown visually in the Project overview (6) and more technical in the Implementation (7) chapter of the thesis, all of the stated elements is present in the demo.

**The train must be able to move using the controllers Lokførerskolen uses today.**

The train can be controlled using the external throttles used at Lokførerskolen (*Logitech Quadrant*).

**The train must follow the railway in a realistic way.**

This was achieved early in the project, with the implementation of the spline. However during user testing, we found that the train behaved abnormally and



would look like it derailed during turns. This was due to an optimization where the spline's level of detail was decreased in fear of having a big impact on performance during testing. The original railway rendered a spline point every 5 meters, while the railway in the user test used 50 meter increments. This caused dissonance between the visible railway and the actual spline which the train followed, causing the anomaly, but has since been reverted to the original version.

### **Signals must be able to change colors based on specific events happening in the game.**

The signals are implemented according to the goals set and with additional feedback from the client. The system works by having a central signal controller in the level, which is responsible for receiving and updating signals. When discussing with the client early in the project the possibility of multiple signal controllers was mentioned, however the current design and implementation of the application uses only one signal controller per level. (7)

#### **10.1.2 Part goals**

The assignment also included part goals that could be necessary extensions if there was any time to allocate for it. The group quickly figured out and planned for the execution of this from the beginning of the project. These part goals were as follow:

#### **Create a tool for placing, editing and deleting 3D models in the game world, such as trains, buildings or signals.**

The functionality for editing 3D models in the game world is present in the current simulator, these added objects are the ones stated above and some others.

#### **Make it possible to save the world when it is edited.**

Saving the edited objects was made possible through the save functionality described in Implementation.

#### **Create a tool for creating, editing and deleting train tracks.**

We tried to make the functionality, but along the way we started to doubt the necessity of the functionality we were trying to create. One of the reasons we chose to develop the simulator using unreal engine was the fact that it provided a state of the art spline tool. Trying to emulate this at runtime, and create our own spline tool seems like a waste of time when you could do this more easily in the Unreal editor. We would therefore argue that even though the runtime spline tool did not get created, we have tested the development and provided documentation and a dedicated branch with testing code for Lokførerskolen if they would like to pursue this issue in the future.

**Train tracks must contain curves and not only go in a straight line.**

As shown in both Product Overview and Implementation, the train track can contain curves, the goal is therefore achieved.

**Make it possible to place a train on the tracks and drive it.**

The program allows you to place a train next to, or on a track and the train will then start driving from the nearest railway point.

### 10.1.3 Analysis goals

**Must be a modern game engine.**

Our analysis concludes on Unreal Engine being a modern game engine. The newest iteration of the engine, Unreal Engine 5, was launched during this project, proving its own modernity.<sup>1</sup>

**Must support functionality for Virtual Reality**

As stated in the analysis, Unreal Engine has native support for VR, with multiple supported VR-headsets. Developing and testing Virtual Reality can easily be done within the Unreal Engine editor, with minimal to no disruption of the workflow.

**Must be capable of reproducing all functionality of DeskSim**

This goal was the only one that still had some doubt after the analysis. we based facts on the likelihood that this new, and maintained engine would have the same opportunities as an outdated engine. After working with the project this was not an issue. We have only implemented some of the functionality, but have not found a single issue regarding this goal.

**The game engine should be easy to learn.**

The game engine was found by us to be moderately easy to learn. It took about a month before the development got more coding than it was researching. This change could also be visualised in the sprint planning meetings if you look at it in the perspective before and after sprint 6.

**The game engine should be able to reuse existing 3D assets from DeskSim.**

The 3D asset formats used for the existing solution are a mix of file formats, some of which are less used in modern game engines. The two primary file formats used by Unreal Engine is .fbx and .obj, with a preference for .fbx due to its ability

---

<sup>1</sup>Unreal Engine 5 launch, <https://www.unrealengine.com/en-US/blog/unreal-engine-5-is-now-available>

to store more types of data such as models, textures, animations, rigs, lights etc. On the other hand .obj files only store model data with some basic material and texture data. Of the file formats used for 3D models in the existing solution, only .obj has native support in UE.

There are two options to import files of .gltf and .ac, the first is to find a plugin for UE which adds support for these files. The second option is to convert these files into either .fbx or .obj using external software such as *Blender*. The second option is the preferred method, as all files would be of the same format, in addition to native support by the engine to import them, rather than relying on external plug-ins.

Adding and using sound in the simulator has not been a priority, and as such we have not used any of the sound files from the existing solution. The existing solution uses the formats .wav and .mp3. Unreal Engine has native support for .wav, so converting sound files from .mp3 to .wav will be required to reuse all sound assets.

## 10.2 Choice of Engine

In the game engine analysis we concluded that Unreal Engine was the most suited option for this project. We can compare how the conclusion fits with our experience after working with Unreal Engine in this project.

### 10.2.1 Experience with Unreal

When we compared the different code languages we found C++ to be the language with the steepest learning curve, due to its manual memory management and low-level nature. When using C++ during development we found the language to be easier than expected, due to the extensive use of Unreal Engine specific C++ libraries. Almost all types and structures, apart from some basic types like integers, floats and booleans, use a type made by and optimized for Unreal Engine. Some examples are strings, which have three different types depending on use and needed functionality, as well as custom arrays and map containers. We found using these types to be easy, and scripting uses almost exclusively libraries made for Unreal Engine. Almost all classes derive from the *UObject* class which features garbage collection, so the developer usually does not need to worry about memory management. Overall using C++ for this project has been an positive experience.

During this project the official documentation for Unreal Engine has been our primary source of tutorials and knowledge when working in Unreal Engine. The documentation is a mix between documents explaining and giving examples to features, as well as tutorials for both C++ and blueprints for various features. This combination means we could rely mostly on official documentation, but supplementing it with community guides or forums when needed.

Since Unreal Engine supports both C++ and blueprints for creating functionalities within the game engine, it can feel like the community is split in two. Sometimes we would find a solution to an issue we had only to find out that it was solved with blueprints. Although since the community is huge we did not run into any issues where we did not find a solution to a problem.

Working with and customising the spline tool using Unreal Engine's built-in spline components was more intuitive in the context of blueprints, as all available functionality was presented by the engine. When developing the spline in C++ , a lot of time was spent on research and reading through the documentation for UE's spline component. But once the barrier of game engine context was passed, UE showed just how powerful its developer tools are, making for an engaging development experience.

When working with game related systems unrelated to programming, such as materials or landscapes, we were able to do all the work inside Unreal Engine. By using both textures and colors, then modifying them in the material editor we were able to create several materials and customise them to our needs, which are used in the simulator. While these materials are made using a node based system very similar to blueprints, and therefore does not showcase any programming, they can interact with scripts to influence the material. The best example of this is the material used for signal lights, where both the color and emissive light strength is controlled through code. Unreal Engine also comes with its own landscape editor where you are able to sculpt and paint the landscape with textures. It also comes with a foliage editor which automatically creates realistic terrain foliage as long as you supplement it with the foliage models and the parameters for it. Both of these editors are simple and powerful to use.

### 10.2.2 Learning curve

The learning curve can feel pretty steep in the beginning of the project, as we did not have any previous experience with Unreal Engine and little experience with other game engines. Once the basic layout and gameplay flow is understood, and where functionality should be made and extended, working with the Engine and creating new content and functionality became much easier. Learning different aspects of the Engine, such as normal actor logic compared to Heads-Up Display (HUD) and User Interface (UI) functionality, can provide developers with useful perspective and knowledge.

In some areas of a game, like its UI, it is possible to create all the functionality in C++ , but it may not be the optimal choice. The UI is usually made mostly in blueprints, but we decided to do as much as possible in C++ for this project. However we did combine scripts and blueprints by deriving a blueprint from the finished script class, which provides the functionality while the blueprint contains the models and materials.

## 10.3 Development Plan and Process

This section will discuss some key concepts of our development process and taking the original project plan into consideration.

### 10.3.1 Time Usage

The original agreement stated in the project plan under the rules section states that all group members should work approximately 30 hours per week and provide documentation for the work that has been done. By looking at the overall number in figure 10.1 and dividing this on the number of group members and weeks during the project, we get approximately 28 weekly hours. This is after taking into consideration that approximately one week in total was dedicated to another course and a week-long Easter break. This was a bit lower than expected, but we figured out during the writing process of this thesis that some hours had not been registered correct according to the log everyone filed each day of the project. We therefore assume that the weekly time usage per person is a bit higher and closer to what we agreed upon at the start of the project.

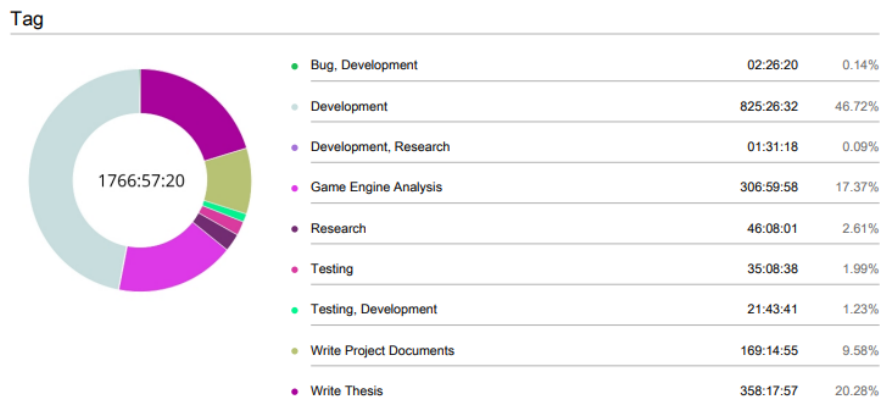


Figure 10.1: Clockify - Overall time usage for project

### 10.3.2 Issues Removed from the Scope

There were some trouble with issues we chose to extend the scope of the project. The examples of this was the plan to develop an environment tool for editing the environment in real-time. We had some discussions with the client on the subject, and he stated that this functionality is not required and is viewed as a wishlist item in their plan for the simulator. Although we did not finalize the editing of terrain, we will provide the source code containing the progress made on the issue, as a branch in the Git repository. The section below is an in-depth explanation and overview of alternatives for creating the functionality in the future.

Unreal Engine already provides a tool for editing terrain in their editor. Modifying

or creating custom terrain needs to be done in the Unreal Editor, which needs to be done during development. There are some alternatives to this. One option is to utilize and understand a runtime complex mesh component, called *Procedural Mesh* which uses the graphical API's to create a component which is modifiable through user input. After researching this opportunity and trying to implement a prototype on how this functionality would turn out, we made a decision to exclude this as a part of the development plan and excluded terrain manipulation as a part of the project. Because the issue was not covered in the task description and client the understanding the decision, the main story and related tasks was removed from the project. The decision to exclude the task is taken because the group wants the final product to highlight its magnitude as well as complexity, the general impression is that this task would disrupt this balance.

In addition to the Procedural Mesh component there are other ways to handle the issue. There are some available plug-ins which could be appropriate to use in the simulator for handling the issue as well. Two plug-ins called *Voxel Plugin*<sup>2</sup> and *Voxel Plugin PRO*<sup>3</sup> makes runtime landscape editing less overwhelming. There is also a chance that UE will implement runtime editing as a plug-in or feature in the editor. There are many in the community that wants a feature like this<sup>4</sup> and Epic Games are generous when it comes to creating and sharing features that their users wants.

### 10.3.3 Estimation Process

The change in estimation strategy came from our desire to actually use the estimations to something useful. We wanted to limit the overhead we had per sprint by having the necessary knowledge and reflections on estimations done earlier. After changing the estimation strategy it got easier to both estimate the issues because we could provide an actual number of hours instead of a fixed size. The change in estimation strategy made both the planning of the sprint easier and the retrospective outcome of more useful for further sprint planning

### 10.3.4 The Group Cooperation

The collaboration and general group mood has been good throughout the project. The only things that could be noted is the few occasions where group members arrived late for the working session. We planned to meet every morning at 9 p.m. from Monday through Friday and this has, with rare exceptions, been the the group's regular practise throughout the entire project.

The different tasks got assigned to the group members based on each group members desire. The group had to ensure that all members got to display their compet-

---

<sup>2</sup><https://voxelplugin.com/>

<sup>3</sup><https://www.unrealengine.com/marketplace/en-US/product/voxel-plugin-pro>

<sup>4</sup><https://forums.unrealengine.com/t/terrain-editing-in-runtime/18271/125?page=9>

ence, and there was a shared responsibility to make sure that every group member got the tasks and time needed to display their skills.

Disagreements has not at all been an obstacle in the development process. There has been some discussions that has taken some time to agree upon, but the process has not been hindered because of it. The group routine that states that the group leader has the final word if the group can not agree were never practiced in the project. Most of the disagreement were resolved by a factual discussion and in some rare occasions they were resolved in a bit louder manner.

### 10.3.5 Other Interactions

We had supervisor meetings almost every week, and meetings with the project client representative every other week. The feedback from both meetings was used to guide for further writing and development. The client interaction was especially helpful and allowed us to rapidly address concerns or issues we faced during development and testing.

## 10.4 Version Control System

Version control or source control, is the practice of tracking and managing changes to software code often within a development team. [45]

Before we started development we had to choose which version control system we would use. On one side we have *Perforce*, often a standard choice for larger studios and is favored within game development due to its handling of large binary files. On the other side we have Git; a standard choice for developers all around due to its many features for teams and how it supports multiple branches which helps with cooperation within a team. The biggest structural difference is that Perforce uses a centralized model unlike Git which uses a distributed decentralized model. Deploying a Perforce server is also necessary before you can use it, unlike Git which is ready to be used from a cloud service such as GitHub or Azure DevOps. Throughout our bachelor program we have used Git as a version control system and collaborative tool to help us keep the code base consistent and maintained, and is one of the reasons for using it in this project as well. Git can also be hosted on the cloud through for example GitHub, which we utilized in this project. Although Perforce would probably be a better choice for bigger projects and teams since it can handle bigger files more efficiently. [46]

The biggest issue we faced with Git was its single file size limit of 100 megabytes, since Unreal Engine projects contain bigger files such as 3D models, textures and maps. We had a few instances where a map file containing the level scene was bigger than Git's allowed file size limit. As a fix we used the level layer tool in Unreal Engine to split the level file into multiple smaller map files. This temporarily fixed the issue and we were no longer hitting the max file size limit of git, additionally

this also made it possible for us to edit and work on the same level without having to overwrite each others changes, since we could just work on different level layers. This is due to the maps being binary files which Git cannot merge. You either have to pick the current one or the incoming file when merging, unlike cpp-files which can be edited by multiple people and can be properly merged together after.

As a more permanent solution to the file size limit we tried using Git's *LFS* at the beginning. Although GitHub only provides a limited LFS storage capacity and bandwidth of 1 gigabyte, which we quickly used up in a few commits. If we wanted a more permanent solution with unlimited LFS storage capacity and bandwidth, using Azure DevOps instead of GitHub might be a solution. Azure DevOps provides unlimited LFS storage capacity and bandwidth and it is fully supported. [47]

## 10.5 Critique

### 10.5.1 Project Plan and Process

Although we changed the estimation strategy we did not use the estimations to the full extent as planned. We should have created a visualization of the estimations made per sprint to fully explain the importance it had for further development.

The group had a strong desire to finish the development of the final product, this desire eventually led to down prioritizing both testing and thesis writing. This made the last days of the project a bit stressful to complete in time. If we would have done the process again we would have changed this process and taking the advantages Scrum provides. As written in the Plan: "Scrum, in it's nature, allows and emphasizes reflection and assessment every step of the way". To downsize the development portion of the project would have been fine as long as the decision is made based on the importance of the bachelor thesis.

### 10.5.2 Thesis

During the project we were greatly inspired by earlier academic work within similar fields. Especially when researching and analysing game engines, we appreciated well-written and structured thesis's enough to motivate our own writing. We hope, and feel, that our own thesis may contribute to future graduate students in search of guidance for writing. When searching for relevant subject matter for our analysis, we struggled to find reliable sources that compare game engines, but hope our contribution adds more foundation to this subject area. We believe and agree that the contents of the Choice of Engine is thorough, and that it remains objective up to the conclusion, with properly cited sources and citations.

As mentioned in The Group Cooperation, it may have been beneficial to allocate more time for writing the thesis, as we are aware of its shortcomings. The time before the deadline was spent adding the final content to the thesis, but the look and layout of the content itself did not get enough work. We fear some of the



figures tables may be badly structured or unintuitive for the reader. We believe this could have been avoided if we either allocated more time for final polish, or had a better understanding of *Latex*.

### 10.5.3 Application

Although nobody is at fault, we do not feel that the development has created something uniquely new or groundbreaking. The reason for this is that the assignment goal itself does not inherently introduce any new concepts or ideas by migrating an existing application.

As with most software, the application was developed for a purpose and a target audience, making testing an important part of the development. We feel as our application could use not only more rounds of testing, but more time for iterations after testing. This would increase the client and user satisfaction, and help our application better fulfill its purpose. One thing that was made clear to us during the Employee User Testing is that the gizmo tools needs to be reworked. This could have been avoided if we did the tests more thorough and performed them earlier.

By the nature of time-limited software development, there are bugs and performance issues in our application that needed more attention. Possible optimizations that would improve the application during both development and runtime include reducing the number of polygons in meshes used repeatedly, utilizing multiple threads for parallel operations, improving the visual look of the simulator, adding more train-related functionality, and adding a VR mode. While some of these are not part of the task, it would improve the application nonetheless.

## 10.6 Further Development

When developing the application we always had in mind that Lokførerskolen should be able to continue the development after we were finished. Since they plan to migrate their current simulator and should be able to migrate their functionalities to this application. In that regard we have documented the process from start to finish, both in and out of code. We made sure to make this process as easy and simple as possible for Lokførerskolen.

Below is a list of features we would recommend Lokførerskolen to continue on:

- Improve the railway spline to be able to connect multiple railways into one or split a railway into different ones.
- Improve the performance of editing the spline in the Unreal Editor, by offloading work on a different thread
- Improve the performance of the simulator, notably the railway mesh and the texture distance blend.
- Add a multiplayer mode where players can play together and interact with one another.

- Add a Virtual Reality mode where you can interact with world by for example connecting wagons or switching railway tracks. Should also be able to play with another player.
- Create an own input manager that is specialized to use the throttles used by Lokførerskolen.
- Improve the physics of the train to be able to simulate going uphill and downhill, by decreasing and increasing the train speed.

There are some functionality which we would recommend against spending more time on, mainly the editor mode we created for runtime editing of maps. We highly recommend using the Unreal Editor instead of spending time and resources recreating an editor.

# Chapter 11

## Conclusion

### 11.1 Summary

Lokførerskolen is looking to migrate their existing DeskSim to a more modern game engine. We hope that this project will help make this transition easier by first providing them with a game engine analysis comparing 5 different engines up against each other. As a result of this we came to the conclusion that Unreal Engine was the best choice for them. Which proved to be correct based on our experience with the engine throughout this project. Secondly we also created a train simulator demo in Unreal Engine with all of the main requirements set by Lokførerskolen and also most of the desired requirements. We ended up pretty happy with the results of the application with some minor roadblocks and issues along the way.

### 11.2 Final words

We believe we have achieved the goal of enabling an easier migration of DeskSim to a new game engine, even if Lokførerskolen does not choose Unreal Engine, as the underlying concepts we have used and documented translates well between engines. We are thankful for our thorough planning and administrative work during the early phases, which have helped us maintain a certain level of structure and quality of work throughout the project. Looking back, the scope of technologies for the development could've been bigger. Even though there was more than enough work to do, a lot of it was rudimentary, and taking on more challenging or ambitious tasks could've yielded more impressive results.

We have learned a lot about software development, working in a game engine, working in a team, and working for a client, which is highly relevant in preparing the group for further work. When all is said and done, we are happy with how the project was planned, developed and documented overall.

# Bibliography

- [1] M. Dahl. ‘Simulert togframføring.’ (2019), [Online]. Available: <https://lokforerskolen.no/aktuelt/simulatorsenteret/> (visited on 07/02/2019).
- [2] I. Farup, *Copcse-ntnu/thesis-ntnu: An ntnu thesis latex document class for bachelor, master, and phd theses*. [Online]. Available: <https://github.com/COPCSE-NTNU/thesis-NTNU>.
- [3] G2 INC, *Unity software vs. unreal engine vs. cryengine vs. godot | g2*. [Online]. Available: <https://www.g2.com/compare/unity-vs-unreal-engine-vs-cryengine-vs-godot> (visited on 08/02/2022).
- [4] M. Dealessandri, *What is the best game engine: Is unity right for you?* Jan. 2020. [Online]. Available: <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you> (visited on 08/02/2022).
- [5] S. Games, *Unity pros and cons: Choose unity 3d: Why unity game studios use it?* Apr. 2021. [Online]. Available: <https://stepico.com/why-choose-unity-3d-engine/> (visited on 08/02/2022).
- [6] A. Chaudry, *A unity review: Pros and cons*, Jul. 2020. [Online]. Available: <https://citrusbits.com/a-unity-review-pros-and-cons/> (visited on 08/02/2022).
- [7] ‘Unreal licences.’ (2022), [Online]. Available: <https://www.unrealengine.com/en-US/download> (visited on 21/01/2022).
- [8] Apr. 2016. [Online]. Available: <https://steamlug.org/cast/s04e05> (visited on 08/02/2022).
- [9] R. Vershelde, *Major milestone ready for testing: Godot 4.0 alpha 1 is out!* Jan. 2022. [Online]. Available: <https://godotengine.org/article/dev-snapshot-godot-4-0-alpha-1> (visited on 08/02/2022).
- [10] ‘Godot licence.’ (2022), [Online]. Available: <https://godotengine.org/license> (visited on 21/01/2022).
- [11] Godot Engine, *Introduction*. [Online]. Available: <https://docs.godotengine.org/en/stable/about/introduction.html> (visited on 08/02/2022).

- [12] 'Open 3d foundation.' (2022), [Online]. Available: <https://o3d.foundation/> (visited on 31/01/2022).
- [13] 'Open 3d engine.' (2022), [Online]. Available: <https://www.o3de.org/> (visited on 31/01/2022).
- [14] Unity Technologies, *Getting started with vr development in unity*, Jan. 2022. [Online]. Available: <https://docs.unity3d.com/2022.1/Documentation/Manual/VR0verview.html> (visited on 08/02/2022).
- [15] M. Dealessandri, *What is the best game engine: Is godot right for you?* Apr. 2020. [Online]. Available: <https://www.gamesindustry.biz/articles/2020-04-14-what-is-the-best-game-engine-is-godot-right-for-you> (visited on 08/02/2022).
- [16] 'Unreal engine 5 early access.' (2022), [Online]. Available: <https://www.unrealengine.com/en-US/unreal-engine-5> (visited on 31/01/2022).
- [17] Crytek, *Cryengine 5.7 roadmap update and future plans*, Dec. 2021. [Online]. Available: <https://www.cryengine.com/news/view/cryengine-5-7-roadmap-update-and-future-plans> (visited on 08/02/2022).
- [18] Unity Technologies, *Supported model file formats*, Feb. 2021. [Online]. Available: <https://docs.unity3d.com/2020.1/Documentation/Manual/3D-formats.html> (visited on 08/02/2022).
- [19] Unreal Engine, *Datasmith supported software and file types*. [Online]. Available: <https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/Datasmith/SupportedSoftwareAndFileTypes/> (visited on 08/02/2022).
- [20] W. A. Haan, *Asset types*, May 2019. [Online]. Available: <https://docs.cryengine.com/display/CEMANUAL/Asset+Types> (visited on 08/02/2022).
- [21] N. Lovato, *Importing 3d scenes*, Oct. 2020. [Online]. Available: [https://docs.godotengine.org/en/stable/tutorials/assets\\_pipeline/importing\\_scenes.html](https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/importing_scenes.html) (visited on 08/02/2022).
- [22] Unity Technologies, *Creating and using scripts*, Feb. 2021. [Online]. Available: <https://docs.unity3d.com/2020.1/Documentation/Manual/CreatingAndUsingScripts.html> (visited on 08/02/2022).
- [23] S. Terziyan, *What is the best language for game development?* Jan. 2020. [Online]. Available: <https://game-ace.com/blog/best-language-for-game-development/> (visited on 08/02/2022).
- [24] *Introduction to c++ programming in ue4*, Aug. 2021. [Online]. Available: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/> (visited on 08/02/2022).
- [25] Godot Engine, *Gdscript basics*. [Online]. Available: [https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript\\_basics.html](https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html) (visited on 08/02/2022).

- [26] Godot Engine, *C# basics*. [Online]. Available: [https://docs.godotengine.org/en/stable/tutorials/scripting/c\\_sharp/c\\_sharp\\_basics.html](https://docs.godotengine.org/en/stable/tutorials/scripting/c_sharp/c_sharp_basics.html) (visited on 08/02/2022).
- [27] 'Blueprint overview.' (), [Online]. Available: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/> (visited on 01/02/2022).
- [28] 'Unity technologies acquires bolt.' (), [Online]. Available: <https://ludiq.io/blog/unity-acquires-bolt> (visited on 04/02/2022).
- [29] 'Visual scripting.' (), [Online]. Available: <https://unity.com/products/unity-visual-scripting> (visited on 04/02/2022).
- [30] 'Avout visual scripting.' (), [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.visualscripting@1.7/manual/index.html> (visited on 04/02/2022).
- [31] 'Schematyc.' (), [Online]. Available: <https://docs.cryengine.com/display/CEMANUAL/Schematyc> (visited on 04/02/2022).
- [32] 'What is visual scripting.' (), [Online]. Available: [https://docs.godotengine.org/en/stable/tutorials/scripting/visual\\_script/what\\_is\\_visual\\_scripting.html](https://docs.godotengine.org/en/stable/tutorials/scripting/visual_script/what_is_visual_scripting.html) (visited on 04/02/2022).
- [33] *Best game engine software in 2022: Compare reviews on ... - g2*. [Online]. Available: <https://www.g2.com/categories/game-engine> (visited on 08/02/2022).
- [34] [Online]. Available: [https://www.iainstitute.org/sites/default/files/what\\_is\\_ia.pdf](https://www.iainstitute.org/sites/default/files/what_is_ia.pdf) (visited on 08/02/2022).
- [35] D. Buckley, *Unity vs. unreal – choosing a game engine*, Jan. 2022. [Online]. Available: <https://gamedevacademy.org/unity-vs-unreal/> (visited on 08/02/2022).
- [36] T-F Evelyn. 'Unity vs unreal engine: Game engine comparison guide for 2021.' (2021), [Online]. Available: <https://www.evercast.us/blog/unity-vs-unreal-engine> (visited on 12/07/2021).
- [37] M. Dealessandri, *What is the best game engine: Is cryengine right for you?* Jan. 2020. [Online]. Available: <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-cryengine-the-right-game-engine-for-you> (visited on 08/02/2022).
- [38] Unreal Engine, *Unreal engine vr mode*. [Online]. Available: <https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/VRMode/> (visited on 08/02/2022).
- [39] M. Rehkopf, *Kanban vs scrum*. [Online]. Available: <https://www.atlassian.com/agile/kanban/kanban-vs-scrum>.
- [40] *Kanban (development)*, Mar. 2022. [Online]. Available: [https://en.wikipedia.org/wiki/Kanban\\_\(development\)](https://en.wikipedia.org/wiki/Kanban_(development)).

- [41] *Changing gamemode at runtime*. Dec. 2016. [Online]. Available: <https://forums.unrealengine.com/t/changing-gamemode-at-runtime/81504>.
- [42] M. Romero, *The uproxy() macro*, Jun. 2020. [Online]. Available: <https://romeroblueprints.blogspot.com/2020/10/the-uproxy-macro.html>.
- [43] U. Engine, *Megascans marketplace*. [Online]. Available: <https://www.unrealengine.com/marketplace/en-US/content-cat/assets/megascans?count=20&sortBy=effectiveDate&sortDir=DESC&start=0> (visited on 20/05/2022).
- [44] *Blueprint vs c++ performance*. Aug. 2016. [Online]. Available: <https://forums.unrealengine.com/t/blueprint-vs-c-performance/3398/17>.
- [45] Atlassian, *What is version control: Atlassian git tutorial*. [Online]. Available: <https://www.atlassian.com/git/tutorials/what-is-version-control> (visited on 10/05/2022).
- [46] Unreal Engine, *Every project needs a version control system*, Jan. 2021. [Online]. Available: <https://unrealcommunity.wiki/every-project-needs-a-version-control-system-qgpkrf2> (visited on 10/05/2022).
- [47] V. Machiraju, B. Miller, K. Sharkey, andrey shuvalov, M. Jacobs, J. Parente, S. Danielson, P. Berger and E. Batkoski, *Manage and store large files in git*, Feb. 2022. [Online]. Available: <https://docs.microsoft.com/en-us/azure/devops/repos/git/manage-large-files?view=azure-devops>.

# Appendices



## **A Source Code**

The project repository contains assets owned by Lokførerskolen, and is kept private as per an agreement between the group members and the client.

## **B Project Agreement**

*Fastsatt av prorektor for utdanning 10.12.2020*

## **STANDARDAVTALE**

### **om utføring av studentoppgave i samarbeid med ekstern virksomhet**

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

#### **Forklaring av begrep**

##### **Opphavsrett**

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

##### **Eiendomsrett til resultater**

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

##### **Bruksrett til resultater**

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

##### **Prosjektbakgrunn**

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

##### **Utsatt offentliggjøring**

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

## 1. Avtaleparter

|  |
|--|
| Norges teknisk-naturvitenskapelige universitet (NTNU)<br>Institutt: Datateknologi og informatikk |
| Veileder ved NTNU:<br><a href="mailto:tom.roise@ntnu.no">tom.roise@ntnu.no</a> , 97139769        |
| Ekstern virksomhet:<br>Ekstern virksomhet sin kontaktperson, e-post og tlf.:                     |
| Studenter: <sup>1</sup><br>Student: Thomas Arinesalingam<br>Fødselsdato: 21.11.1998              |
| Student: John Ole Bjerke<br>Fødselsdato: 21.12.2000  |
| Student: Endre Heksum<br>Fødselsdato: 29.05.2000   |
| Student: Henrik Markengbakken Karlsen<br>Fødselsdato: 25.01.1999                                 |

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

## 2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

|                 |   |
|-----------------|---|
| Masteroppgave   |   |
| Bacheloroppgave | X |
| Prosjektoppgave |   |
| Annen oppgave   |   |

|                       |
|-----------------------|
| Startdato: 10.01.2022 |
| Sluttdato: 20.05.2022 |

Opgavens arbeidstittel er: DeskSim simulatoroppgave for Lokførerskolen.

---

<sup>1</sup> Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

### **3. Ekstern virksomhet sine plikter**

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven: Dekker transport til og fra skolen med buss, tog eller bil. Og dekker utgifter dersom det er avtalt for utlån av utstyr til oppgaven.

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

### **4. Studentens rettigheter**

Studenten har opphavsrett til oppgaven<sup>2</sup>. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

### **5. Den eksterne virksomheten sine rettigheter**

---

<sup>2</sup> Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

#### Alternativ a) (sett kryss) Hovedregel

|                          |  |
|--------------------------|--|
| <input type="checkbox"/> | Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven |
|--------------------------|--|

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

#### Alternativ b) (sett kryss) Unntak

|                          |   |
|--------------------------|---|
| <input type="checkbox"/> | Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt |
|--------------------------|---|

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

#### 6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

#### 7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

#### 8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

|                                     |                              |
|-------------------------------------|------------------------------|
| <input checked="" type="checkbox"/> | Oppgaven skal være offentlig |
|-------------------------------------|------------------------------|

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

| Sett kryss               | Sett dato |
|--------------------------|-----------|
| <input type="checkbox"/> | ett år    |
| <input type="checkbox"/> | to år     |
| <input type="checkbox"/> | tre år    |

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

## 9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det

likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

### Signaturer:

|                     |                             |
|---------------------|-----------------------------|
| Instituttleder:     |                             |
| Dato:               |                             |
| Veileder ved NTNU:  |                             |
| Dato:               |                             |
| Ekstern virksomhet: |                             |
| Dato:               |                             |
| Student:            | <i>John Ole Bjerke</i>      |
| Dato:               | <i>24.01.2022</i>           |
| Student:            | <i>Henrik M. Karlsen</i>    |
| Dato:               | <i>25.01.2022</i>           |
| Student:            | <i>Thomas Arnesdalingam</i> |
| Dato:               | <i>25/01/2022</i>           |
| Student:            | <i>Endre Hekrum</i>         |
| Dato:               | <i>29.01.2022</i>           |



## **C Task Description**

# Forslag til bacheloroppgave innen IT fra Lokførerskolen

## Simulatorutvikling

Norsk fagskole for lokomotivførere, eller Lokførerskolen, er en offentlig statlig godkjent fagskole som utdanner lokomotivførere til hele landet. Som en del av utdanningen bruker vi en selvutviklet programvare kalt DeskSim for simulering av togframføring og andre relaterte oppgaver. Denne programvaren er bygget med Java i spillmotoren jMonkeyEngine og 3D-modelleringsverktøyet Blender brukes for å lage modeller.

### Oppgaven

Spillmotoren jMonkeyEngine har eksistert siden 2003 og selv om det fortsatt er i bruk og vedlikeholdes så er ikke engasjementet rundt dette prosjektet det samme som det har vært. Deler av jMonkeyEngine begynner å bli utdatert og Lokførerskolen vil unngå å havne i den situasjonen at verktøyene vi bruker ikke lenger er vedlikeholdt eller støttet. Dagens situasjon med tanke på moderne spillmotorer som er åpent tilgjengelig for bruk er også en helt annen enn når utviklingen av DeskSim ble startet. Vi har derfor startet et langsiktig prosjekt for å kunne overføre DeskSim til en moderne spillmotor.

Oppgaven består av to deler:

- 1. Innledende del;** undersøke hvilke alternativer som finnes med tanke på moderne spillmotorer som er offentlig tilgjengelig og gjøre en faglig vurdering av fordeler og ulemper med de forskjellige alternativene.
- 2. Hoveddel;** gjenskape elementer av DeskSim ved hjelp av den spillmotoren som gruppen vurderer som det beste alternativet. Denne delen vil være delt inn i et hovedmål og flere delmål. Delmålene kan studentene jobbe med dersom de ser at hovedmålet er oppnådd.

### Oppgavens mål

Målet med oppgaven er å gi Lokførerskolen et godt grunnlag for å vite hvilken spillmotor vi skal satse på i framtiden.

### Oppgavens krav

#### Innledende del

- Kartlegge og sammenfatte informasjon om moderne spillmotorer som kan være aktuelle for Lokførerskolen å bruke.
  - Spillmotorene må tilfredsstillende følgende krav:
    - Spillmotoren **må** være 'moderne'.
    - Spillmotoren **må** gjøre det mulig å gjenskape de eksisterende elementene i DeskSim.
    - Spillmotoren **må** være egnet for VR-utvikling.
    - Spillmotoren **bør** gjøre det mulig å gjenbruke eksisterende ressurser slik som 3D-modeller fra DeskSim.
    - Spillmotoren **bør** være så lett som mulig å lære seg
  - Funnene skal presenteres i en oversiktlig og lettfattelig rapport som inneholder følgende:

Kontaktperson: Isak Kvalvaag Torgersen

Epost: isator@jernbanedirektoratet.no

- Informasjon om viktige faktorer om spillmotorene slik som programmeringsspråk, lisensmodell, osv.
- En begrunnet vurdering av hvordan de forskjellige spillmotorene etterlever Lokførerskolens krav og hvilken spillmotor gruppa anser som det beste alternativet.
- Vi ønsker at gruppa skal se på **minst** tre alternativer inkludert Unreal Engine og Unity.

## Hoveddel

### Hovedmål

- Lage en demo i den spillmotoren som gruppa anser for å være det beste alternativet. Demoen må tilfredsstillende følgende krav:
  - Demoen må inneholde minst et tog, minst to signaler, en tog-DMI som ligner på den i DeskSim, togspor og et enkelt landskap. Det er en bonus hvis man kan gjenbruke ressurser fra DeskSim.
  - Toget må kunne startes og stoppes av bruker ved hjelp av spakene som Lokførerskolen bruker i dag.
  - Toget må kunne kjøre en kort strekning hvor det følger togs�innene på en realistisk måte.
  - Signalene må kunne endre seg underveis (for eksempel skifte fra rødt til grønt eller vis versa) basert på forhåndsbestemte faktorer (slik som togets posisjon).
  - All koden som brukes for å lage demoen må være veldokumentert og det bør være mulig å bygge videre på den koden.
  - Gruppen bør følge etablerte prinsipper for programvareutvikling/-design og gjøre rede for valgene de tar.
  - Det kan være nødvendig å justere eller tydeliggjøre kravene underveis og det forventes at utviklingen skjer i tett samarbeid med Lokførerskolen
  - Det bør være fokus på at verktøyet bør være så enkelt og intuitivt som mulig å bruke.

### Delmål 1

- Lage et verktøy for å plassere ut 3D-modeller (tog, signaler, bygninger, osv.) langs forhåndsdefinerte jernbanestrekninger.
  - Det må også være mulig å flytte eller fjerne eksisterende modeller.
  - Når nye modeller har blitt plassert langs en strekning må de være mulig lagre endringene slik at de blir reflektert når man skal kjøre strekningen neste gang.

### Delmål 2

- Lage en strekningsbygger. Med strekning menes en virtuell strekning med jernbanespor hvor det er mulig å kjøre simulerte tog.
  - Det må være mulig å opprette, redigere og fjerne strekninger.
  - Strekningene må kunne inneholde kurver (ikke bare rett fram).
  - Når en strekning er opprettet må det være mulig å plassere ut en togmodell og kjøre strekningen.
  - Det er ikke et krav at det er mulig å endre på landskapet i strekningen eller justere på høydekurver i sporet o.l., men det må tas høyde for at dette skal bli mulig senere.

Video som illustrerer DeskSim finner du her; <https://lokforerskolen.net/tmp/bachelor/>

## **D Project Plan**

PROG2900 - BACHELOR THESIS

---

# Project Plan

Lokførerskolen Sim v2

---

*Authors:*

Thomas Arinesalingam

John Ole Bjerke

Endre Heksum

Henrik Markengbakken Karlsen

January, 2022

---

# Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Goals and Constraints</b>           | <b>1</b> |
| 1.1      | Background . . . . .                   | 1        |
| 1.2      | Project Goals . . . . .                | 1        |
| 1.2.1    | Effect Goals . . . . .                 | 1        |
| 1.2.2    | Learning Goals . . . . .               | 2        |
| 1.3      | Constraints and Boundaries . . . . .   | 2        |
| <b>2</b> | <b>Scope</b>                           | <b>2</b> |
| 2.1      | Subject Area . . . . .                 | 2        |
| 2.2      | Delimitation . . . . .                 | 3        |
| 2.3      | Existing Solution . . . . .            | 3        |
| 2.4      | Task Description . . . . .             | 4        |
| <b>3</b> | <b>Project Organization</b>            | <b>4</b> |
| 3.1      | Responsibilities and Roles . . . . .   | 4        |
| 3.2      | Routines and Group Rules . . . . .     | 5        |
| 3.2.1    | Group Rules . . . . .                  | 5        |
| 3.2.2    | Violation of Rules . . . . .           | 5        |
| 3.2.3    | Group Routines . . . . .               | 6        |
| <b>4</b> | <b>Planning and Process</b>            | <b>6</b> |
| 4.1      | Work Methodology . . . . .             | 6        |
| 4.1.1    | Scrum . . . . .                        | 6        |
| 4.1.2    | Process Documentation . . . . .        | 7        |
| 4.2      | Weekly Schedule . . . . .              | 7        |
| 4.3      | Technologies and Environment . . . . . | 8        |
| 4.3.1    | Version Control and Process . . . . .  | 8        |
| 4.3.2    | Reports, Documents and Logs . . . . .  | 8        |
| 4.3.3    | Meetings and Communication . . . . .   | 8        |

---

|          |  |           |
|----------|--|-----------|
| 4.3.4    | Coding and Development . . . . .                   | 8         |
| <b>5</b> | <b>Quality Assurance</b>                           | <b>9</b>  |
| 5.1      | Standards . . . . .                                | 9         |
| 5.2      | Documentation . . . . .                            | 9         |
| 5.3      | Configuration Management . . . . .                 | 9         |
| 5.3.1    | Git Workflow . . . . .                             | 10        |
| 5.4      | Risk Analysis . . . . .                            | 10        |
| 5.4.1    | Identification, Probability and Severity . . . . . | 11        |
| 5.4.2    | Risk Matrix . . . . .                              | 12        |
| 5.4.3    | Mitigation and Measures . . . . .                  | 12        |
| 5.5      | Testing Plan . . . . .                             | 12        |
| 5.5.1    | Unit Testing . . . . .                             | 12        |
| 5.5.2    | Acceptance Testing . . . . .                       | 12        |
| <b>6</b> | <b>Implementation Plan</b>                         | <b>12</b> |
| 6.1      | Gantt Chart . . . . .                              | 12        |
| 6.2      | Product Backlog . . . . .                          | 13        |
| 6.3      | Development Milestones . . . . .                   | 14        |
| 6.4      | Administrative Milestones . . . . .                | 15        |
| 6.5      | Decision Points . . . . .                          | 15        |
|          | <b>Appendices</b>                                  | <b>16</b> |
| <b>A</b> | <b>Gantt Chart</b>                                 | <b>16</b> |

---

# 1 Goals and Constraints

## 1.1 Background

The Norwegian Train Driver Academy, *Lokførerskolen*, is a vocational school part of The Norwegian Railway Directorate, educating locomotive drivers. As a part of the education they use *DeskSim*, a software developed in-house to simulate operating a train. DeskSim is built upon *jMonkeyEngine*, a Java-based game engine developed in 2003 that has recently become outdated in some areas. To avoid irrelevancy, Lokførerskolen have started their long-term project of changing game engines.

We are therefore tasked to initialize the long-term goal of migrating the simulator between game engines. To begin with, we will look into different modern game engines and find the one best matching the criteria. After that, we will create a demo of the simulator in the game engine of choice.

## 1.2 Project Goals

The goal of the project is to look into different game engines and find the best suited to develop the train simulator for Lokførerskolen. The specific goals required by Lokførerskolen:

- Help *Lokførerskolen* decide which engine best fit their use
- In-depth analysis of at least three game engines fulfilling the requirements of:
  - Supporting functionality for virtual reality
  - Being a modern engine
  - Being capable of reproducing all functionality of DeskSim
- Developing a working demo of a train simulator in the chosen game engine
- Developing a tool for placing additional 3D-models along predefined railway lines
- Developing a tool for generating new train routes with railway lines

### 1.2.1 Effect Goals

Today *Lokførerskolen* uses DeskSim as a part of their educational program, where our role is to lay the foundation of their long-term project of changing game engine. Some effect goals are:

- Create a useful learning tool that *Lokførerskolen* can use in their educational program



- 
- Lay the foundation for further development in a new game engine, ensuring a smooth transition from the old engine
  - Prevent that their tools become irrelevant and outdated, such as their game engine

### 1.2.2 Learning Goals

Through this project, we wish to focus on the learning outcome, and have defined a list of goals we seek to achieve.

We aim to:

- Get a deeper understanding of game engines, their features, limitations and overall role in a game development process
- Gain experience from a working methodology, roles and teamwork within professional game development
- Apply relevant terminology to compare, discuss and present our decisions
- Gain experience in self-elected technologies, code languages and software
- Apply knowledge and skills gained through our studies

## 1.3 Constraints and Boundaries

Since we are creating an application for Lokførerskolen and not for ourselves, they have imposed constraints and boundaries we must follow. Any suggestions from us that goes beyond the specified project scope must be clarified with the client.

We do not have any previous experience related to train operation, and are not responsible for the train-related knowledge and specifications needed in this project. These specifications are provided by Lokførerskolen, as our only responsibility will be developing the application itself.

The project has a deadline, which means we have to carefully schedule the work according to the restricted time allocated for this project. We have to take this into consideration when prioritizing which features to implement.

## 2 Scope

### 2.1 Subject Area

The use of simulators in an educational setting is very useful for training in specific scenarios. Simulators allow students to train in a safe environment, where expensive

---

or dangerous environments or situations prohibit training with real equipment in the real world.

Even though developing a simulator for a specific use case can be costly both in time and resources, the value of such a simulator increases in the long run. Better, more focused training combined with savings on teachers, rooms, equipment and other resources can make simulators a worthwhile investment.

As the simulator runs on a computer with some extra peripherals used, a normal game engine can be used for handling common tasks such as game loops, rendering, loading assets, audio, input, etc. While there are many modern game engines available, choosing the right one for the project is important, both for implementing the required functionality but also with future expansion in mind.

## 2.2 Delimitation

In this project we will compare a few different game engines for developing a simulator. While we are going to compare different engines, we will not be performing a comprehensive technical analysis of each engine and its features. We will also limit ourselves to about 4-5 different engines.

Due to the technical nature of the simulator, we are only going to implement core functionality related to movement of the trains, as well as a basic signaling system. The train tracks need some complexity, such as curves and merging/splitting, to allow for realistic train movement.

As the task is not to develop new content for the simulator, we will reuse assets from the existing simulators when possible. Improving the graphical fidelity is not a goal of the project, but this may be done as a part of using a newer, more capable engine. We are planning to reuse the same style of user interface as the existing simulator.

## 2.3 Existing Solution

An existing simulator for trains is in use at Lokførerskolen called DeskSim. The simulator is used during classes and helps with teaching, allowing the students to try out different scenarios during class. The simulators are also free to use outside of class where students can train for the scenarios they want.

The existing simulator is written in Java using the jMonkeyEngine. Blender is used to develop models for the game, and will also be in use for this demo. An external flight throttle is used to control the speed of the train, which the demo is also going to use.

---

## 2.4 Task Description

The goal of the project is to select a game engine and create a small demo of the simulator. This demo is intended for future use as the foundation for migrating the existing simulator to a new engine. The functionality of the simulator is to be made from scratch, with guidance from the existing simulator and the client. The simulator is going to use the same peripherals as the existing simulator. The demo must run on Windows OS.

The project can be divided into two parts. The first part is to compare different game engines, look at some of their strengths and weaknesses, and decide which engine to use in the second part of the project. The different engines have to comply with a few requirements by the client, as specified in the [Project Goals](#). There are also some goals which are important, but not required. These are reusing existing assets and models from the simulator, and how simple it is to learn and use the engine.

The second part of the project is to make a demo of the simulator in the engine of choice. As the train-specific knowledge required to make a simulator is outside of the scope of our education and task, we are going to create a demo with some basic key features. The demo should contain a scenario, with train tracks that can be curved, at least two train signals which can change state based on in-game factors, a train which can be moved along the tracks in a realistic fashion using the external peripherals.

The intention for this demo is to become the foundation for the full simulator on a new engine. Therefore it is important to document the code and choices we make. It should be possible to extend the functionality of the code for more features as needed.

## 3 Project Organization

### 3.1 Responsibilities and Roles

**All group members** have the roles of researchers, developers and designers of the product. Everyone must ensure the quality and production of own work, as well as the well-being of other team members. They must read and comply to the set of rules and routines prepared by the group.

**Thomas Arinesalingam** is the *Project Leader*. His role specific responsibilities are to ensure that all group members have equal right to express their thoughts. Be the project's "man of action", motivating the development. He will ensure that all submissions are delivered on time.

**John Ole Bjerke** is the *Research Manager*, overseeing all research and ensuring the level of obtained knowledge is adequate before the development starts.

**Endre Heksum** is the *Scrum Master* for the project and is responsible for the

---

development of the product and takes the role of sprint leader.

**Henrik Markengbakken Karlsen** is the *Writer of Minutes*, writing minutes from all meetings and making sure all team members logs both time and work log.

**Lokføerskolen** is the product owner, overseeing the final product.

**Tom Røise** is our supervisor during the project, providing guidance and academic support to the group in the development process.

**All group members** have the responsibility to read and comply to the set of rules and routines prepared by the group.

## 3.2 Routines and Group Rules

The process of a large project requires structured and organized routines and rules. These routines and rules should not be constraints, but provide guidance. They should be used to dissolve conflicts in the group. The consequence of a rule breach depends on both quantity and severity factor of the action.

### 3.2.1 Group Rules

- Each member of the group is expected to work approx. 30 hours per week, and should provide documentation as proof if necessary.
- All group members must meet at the agreed upon time.
- If a group member is unable to attend a meeting, the group should be notified the day in advance.
- The group members are expected to show up to each of the scrum meetings listed in the [Weekly Schedule](#).

### 3.2.2 Violation of Rules

1. By a breach of small rules the student will receive a verbal warning from the group leader.
2. By several minor breaches the student will receive a warning by mail, stating that a meeting will be scheduled with the supervisor, if the group member fail to change their behaviour.
3. By a serious offense, the student will receive a notice about their behaviour and a warning of their last chance before a formal request to dismiss the student from the group is sent to the supervisor.

---

### 3.2.3 Group Routines

- Any work started, developed or finished must follow a workflow as described in the [Git Workflow](#). It is important to adhere to these protocols for an efficient workflow.
- When a work day is finished it is important to log the total work time and write a note of what was done.
- Should any disagreement disrupt the group's work, a factual discussion will be held, taking all parties into consideration. If the dispute is not resolved by discussion, the opinion of the majority will decide. If a majority vote is not possible, the group leader should have the deciding vote regarding the matter. Should the problem still persist, a discussion between the group leader and the project supervisor will conclude the final decision.

## 4 Planning and Process

### 4.1 Work Methodology

The choice of work methodology is crucial in the beginning of software development to ensure reaching the desired product in the project period. The methodology we think works best for this project is Scrum. The reasons for this are several, but the most important are that the client stated in the project proposal that they expect that we have a development process where they are heavily involved in case they see the need to clarify or adjust the requirements.

We are choosing an agile development methodology we want to make sure the client is satisfied with the demo.

#### 4.1.1 Scrum

We aim to use the iterative nature of Scrum to ensure the quality of the implementation at every stage of the project, and quickly adapt to any change in the client's specifications. In the beginning of the project we will have one week sprints and continuously discuss if we need to have bigger sprints as we go based on the upcoming tasks. We are going to implement Scrum in a strict way where we have daily scrum meetings throughout the project.

These meetings should be used as a collaborative tool to identify problems if someone is stuck, keep everyone up to speed on the project process and create a nice work environment where we keep in touch. We believe the latter is more important than it may seem, because most of the work is done from home, so it is nice to keep in touch with the group at least once every day. Although these meetings are important for the group, we can skip them occasionally if they are not needed.

For managing the project, we are using Jira integrated with GitHub. This integration contributes to ensuring the methodology professionalism we aim for in the project. We will be using constrained and secure workflows to ensure that every issue gets handled with care. In simpler terms, this sets the rules for how all work is to be handled throughout the project. Figure 1 shows the direction and constraints on how an issue can progress through the different statuses.

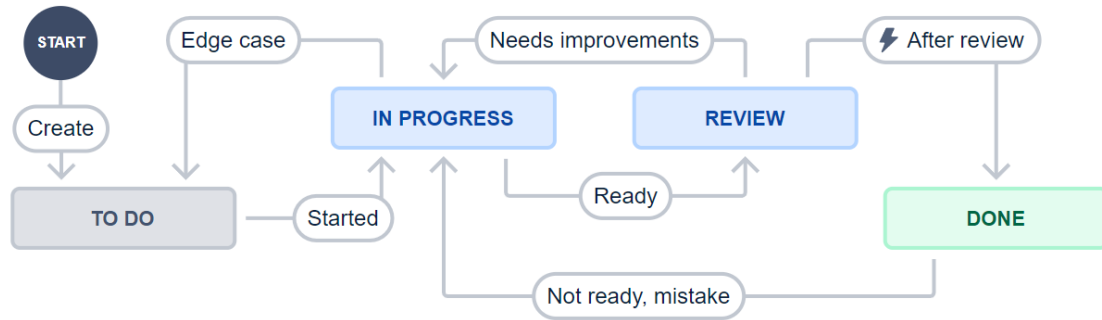


Figure 1: Issue status diagram

#### 4.1.2 Process Documentation

To log the time spent on the project, we use the Clockify add-on for Jira. When starting work on an issue, a timer is manually started from within the issue, tracking all work time until it is stopped. This time is manually logged along with a short comment of what was done that day. For planning the holistic overview of the project, we use BigGantt. This is an extension to Jira which allows us to create a Gantt chart in the same environment as our issues.

## 4.2 Weekly Schedule

We plan on working Monday through Friday, from 9:00 a.m. to 4:00 p.m. every week during the project period. The only exception is during a planned Easter break from the 13th of April to the 17th of April.

| Meeting Type            | Time                         |
|-------------------------|------------------------------|
| Daily Scrum Meeting     | 9.00 a.m.                    |
| Supervisor Meeting      | 1:30 p.m. every Wednesday    |
| Sprint Planning Meeting | 2:00 p.m. every Monday       |
| Client Meeting          | 9:00 a.m. every other Monday |

Table 1: The current meeting schedule

---

## 4.3 Technologies and Environment

### 4.3.1 Version Control and Process

The project files are hosted as a repository on GitHub, explained further in [Configuration Management](#). We utilize Jira for keeping track of the development process, with the extensions BigGantt and Clockify, as specified earlier in [Process Documentation](#).



Figure 2: The logos of GitHub, Jira, BigGantt and Clockify, respectively

### 4.3.2 Reports, Documents and Logs

All reports and submissions are written in LaTeX, using Overleaf as the collaborative writing environment. Overleaf is also used to document and log work time and minutes of meetings.

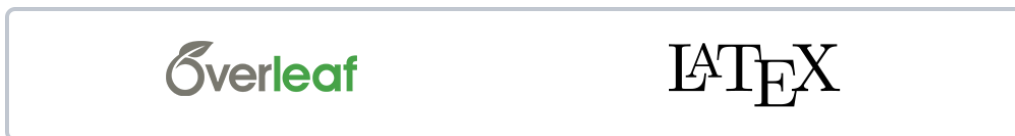


Figure 3: The logos of Overleaf and LaTeX, respectively

### 4.3.3 Meetings and Communication

The group communicates via Discord for internal meetings, and Microsoft's Teams for external meetings.

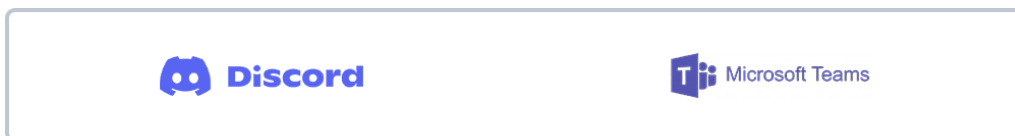


Figure 4: The logos of Discord and Microsoft Teams, respectively

### 4.3.4 Coding and Development

The game engine and the tools used for writing code are not yet decided. As further explained in [Documentation](#), we will utilize a tool for generating code documentation depending on which programming language we will choose. Seeing as the file type of the assets from the existing simulator may be outdated, the group might have to utilize a 3D-modeling software such as Blender for file conversion.

---

## 5 Quality Assurance

### 5.1 Standards

As the assignment requires a professional assessment of game engines prior to the development phase, it is unclear which technologies, software and coding languages we will be utilizing as of the time writing. We have decided to set general standards which can be modified at a later date to fit the direction of the project.

For standardizing the format of our code, we aim to respect coding conventions defined as industry standards by most programming languages. These standards and coding conventions, when specified, will be available as a separate document.

In general, all variables, functions and methods must be named accordingly to their function or context. All code and comments must be written in English.

### 5.2 Documentation

As the software is to be developed for a client, it is crucial to produce intuitive and understandable source code by documenting all code functionality. This will also increase the groups common understanding of the code. In-line commenting should only be applied where needed in order to reduce the amount of unnecessary text.

We plan to introduce a tool for documentation generation after the group has decided on a programming language. All classes, functions or methods must be commented sufficiently and formatted to the standards of the selected tool. This will let us generate a separate document explaining the functionality of the source code, making it easier for the client to continue the development. Example of such tools are *Doxygen* for *C++* and *DocFX* for *C#*.

### 5.3 Configuration Management

The project will be hosted in a GitHub repository, where all features should be developed on separate branches, such that the main branch always consists of a working version with fully implemented features. Before any feature is merged to the main branch, it should be reviewed by another group member. Small or critical fixes such as spelling errors may be edited directly on the main branch.

Each developer is responsible for the quality of the code they add to the project, but the group has a shared responsibility of the project as a whole. To ensure a certain degree of quality, completed issues from the product backlog are to be reviewed by the rest of the group before they are marked as *done*, as part of the weekly sprint planning meetings.



---

### 5.3.1 Git Workflow

To ensure good code quality and project management it is important to have a Git workflow that allows you to find the commits and branches you are looking for. The workflow stated below is mandatory for all group members.

- Always create a new branch when starting work on a feature. No work should be done directly on the main branch.
- This is, and should be the naming convention for branches:  
`<issue_number>-<branch_name>`
- This is, and should be the only commit convention:  
`[#<issue-number>]-<description>`
- When the work is completed on a branch, it must be deleted after the work is merged into the main branch.
- Code should be committed often, either when a task is finished or the newly written code is fully functional.
- Do not commit code that doesn't compile. Code should be tested before it is committed.
- When a feature is complete, its branch should be merged into the current milestone branch. When the work for one milestone is completed, this branch should be merged with main.

## 5.4 Risk Analysis

Identifying and eventually mitigating risk is important to avoid problems in software development. During the project process, the agility of scrum helps to identify risk in the development process. The weekly sprint planning meetings makes it possible to address the problems quickly.

---

### 5.4.1 Identification, Probability and Severity

After determining the probability and severity of each risk, it is categorized into a range Low, Medium or High in their structural level..

#### Technological Level

| Title                   | P | S | Description   |
|-------------------------|---|---|---|
| GitHub Incidents        | H | L | In 2021, GitHub reported <sup>1</sup> 4 severe incidents that had an average respond time of 3 hours and 25 minutes |
| Jira unavailability     | L | L | Jira only reported <sup>2</sup> two incidents in 2021   |
| Functionality migration | L | H | If not all functionality can be migrated it would break with the clients requirements                               |
| Steep learning curve    | L | M | The group has experience with different programming languages and game engines, but mostly in 2D                    |

Table 2: The probability for technology level risks where P is Probability and S is Severity

#### Business Level

| Title                         | P | S | Description  |
|-------------------------------|---|---|--|
| Misunderstanding and setbacks | L | M | Every other week there will be meetings with the client to review the progress         |
| Productivity issues           | L | M | Sickness, stress or other issues that can disrupt a group member's workflow            |
| Insufficient Risk management  | M | M | Incorrect estimation of risks leading to issues, or failure to identify important risk |

Table 3: The probability for business level risks where P is Probability and S is Severity

#### Project Level

| Title                | P | S | Description  |
|----------------------|---|---|--|
| Scope creep          | M | M | Unforeseen expansion of scope due to lack of knowledge and experience in train simulation                  |
| Poor quality of code | L | H | The groups shared academic competence in documentation and quality testing should benefit the code quality |

Table 4: The probability for project level risks where P is Probability and S is Severity

---

<sup>1</sup>Source for GitHub status: <https://www.githubstatus.com/history?page=1>

<sup>2</sup>Source for Jira status: <https://status.atlassian.com/history>

---

### 5.4.2 Risk Matrix

We have decided to mitigate the identified risk that has both severity and probability higher or equal than medium. As for the rest, we accept the risk.

|             |        | Severity |        |      |
|-------------|--------|----------|--------|------|
|             |        | Low      | Medium | High |
| Probability | Low    | 1        | 3      | 2    |
|             | Medium | 0        | 2      | 0    |
|             | High   | 1        | 0      | 0    |

Table 5: Risk assessment matrix

### 5.4.3 Mitigation and Measures

**Poor risk management** can be mitigated by analyzing the possible risks, making sure we plan for mitigation.

**Scope creep** can be mitigated by having a tight connection to the client, sharing any concerns about possible scope expansion. It will also help to inform the client if the new work environment is taking more time than expected, and to evaluate whether we should deprioritize the projects minor tasks.

## 5.5 Testing Plan

### 5.5.1 Unit Testing

After a feature is implemented on a branch, the feature must be treated as a unit and pass a Unit Test before it is merged into the next branch. These tests are done by each developer, and are only documented as the relevant testing code itself.

### 5.5.2 Acceptance Testing

We plan to conduct acceptance tests done by the client during client meetings after each milestone is reached. This will ensure the project going in the right direction.

## 6 Implementation Plan

### 6.1 Gantt Chart

The Gantt chart is available as an appendix.

---

## 6.2 Product Backlog

Issues are divided into epics, stories, tasks and bugs. An epic is a milestone or collection of issues, which can contain stories, tasks and bugs. A story tracks features or functionality, while a task tracks a small distinct piece of work. Stories and tasks can have child issues if needed. Bugs track errors and problems which needs to be fixed. We are estimating the time required to finish each product by the scale of 1-4, where 1 is up to a half day's work, 2 is up to a days of work, 3 is up to three days of work to complete. The estimate 4 is used for the major issues like report writing and big deliverable where the epic or story are part of many subissues.

| <b>Issue ID</b> | <b>Issue</b>               | <b>Type</b>      | <b>Estimate</b> |
|-----------------|----------------------------|------------------|-----------------|
| LK-4            | Project Plan               | Story            | 4               |
| LK-6            | Game Engine Analysis       | Story            | 4               |
| LK-7            | Research Game Engines      | Subtask of LK-6  | 4               |
| LK-8            | Write Game Engine Analysis | Subtask of LK-6  | 4               |
| LK-10           | Status Report 1            | Story            | 3               |
| LK-12           | Status Report 2            | Story            | 3               |
| Lk-13           | Code Convension Document   | Story            | 3               |
| LK-14           | Requirement Specification  | Subtask of LK-10 | 3               |
| LK-16           | Fundamental Engine Setup   | Epic             | 3               |
| LK-17           | Basic Train Mechanics      | Epic             | 4               |
| LK-18           | Minimum Viable Product     | Epic             | 4               |
| LK-19           | Model Placing Tool         | Epic             | 4               |
| LK-20           | Train Route Tool           | Epic             | 4               |
| LK-21           | Polish and Bug Fix         | Epic             | 3               |
| LK-21           | Project Planning Phase     | Epic             | 4               |
| LK-23           | Final Report               | Epic             | 4               |

Table 6: All issues of the product backlog as of the time of writing

---

## 6.3 Development Milestones

### 1. **Feb. 07, 2022 - Fundamental Engine Set-Up**

Set up the project in our game engine of choice, learn how to use the game engine. This includes importing assets, basic user interface and setting up the collaborative work space.

### 2. **Feb. 28, 2022 - Basic Train Mechanics**

Have a train moving from input. Have two of the required train signals implemented<sup>1</sup>. Added the necessary libraries, code and bindings to get any external peripherals working.

Implement train mechanics, signals and game controls

### 3. **Mar. 14, 2022 - Minimum Viable Product**

It should be possible to operate a train, bug-free, on a predefined train route. The train should be able to start and stop using the external peripherals provided by Lokførerskolen. Train based logic should translate to in-engine components and user interface, such as signals and speed. We consider this to be the minimum viable product.

### 4. **Apr. 22, 2022 - Model Placing Tool**

The simulator includes a map editor where 3D-models such as signs or buildings can be added, placed, moved and deleted from the map. These changes to existing train routes should be saved persistently.

### 5. **Apr. 22, 2022 - Train Route Tool**

It should be possible to edit, delete and create new train routes. Creating a route should generate tracks which the user can ride as a part of the simulator. These tracks must be able to curve, and should be developed to support dynamic height levels.

### 6. **Apr. 29, 2022 - Polish and Bug Fix**

As the final milestone, the simulator should fulfill all requirements set by the client, and should be optimized without any bugs.

---

<sup>1</sup>Subject to Change. At the time of writing we have not received a proper demonstration of train signal functionality from the client, and cannot assume the workload for this task.

---

## 6.4 Administrative Milestones

1. **Jan. 31, 2022 - Analysis, Project Plan and Client Contract**  
Analyse and write a descriptive comparison between the game engines and conclude on what engine is best suited for our project. Finish the project plan and set up a collaborative agreement contract with the client.
2. **Feb. 07, 2022 - Simulator Demonstration and Code Convention**  
Visit The Norwegian Train Driver Academy on Oslo to experience the current simulator, gaining an understanding of any train-specific functionality, and the expectations for the final product. And writing a document defining the group's rules and guidelines for the layout and style of coding, after the coding language is decided.
3. **Feb. 28, 2022 - Status Report 1 and Requirements Specification**  
A report of the current progress and group status, together with a document of the software requirements specification.
4. **Mar. 28, 2022 - Status Report 2**  
A second report of the current project's progress and status.
5. **May. 20, 2022 - Final Graduate Report**  
The final bachelor thesis report is finalized and delivered.

## 6.5 Decision Points

1. **Jan. 31, 2022 - Game Engine Decision**  
An in-depth analysis is performed, and which game engine to use is decided.
2. **Feb. 14, 2022 - 3D-Asset Recycling**  
If the file types of the existing assets might be incompatible with the new engine, and we may have to create new 3D assets.
3. **Mar. 14, 2022 - Additional Tools**  
At this stage we should have an overview of our progression, and must decide if it is be possible to develop the additional tools specified by Lokførerskolen.

# Appendices

## A Gantt Chart



## **E Clockify Summary**

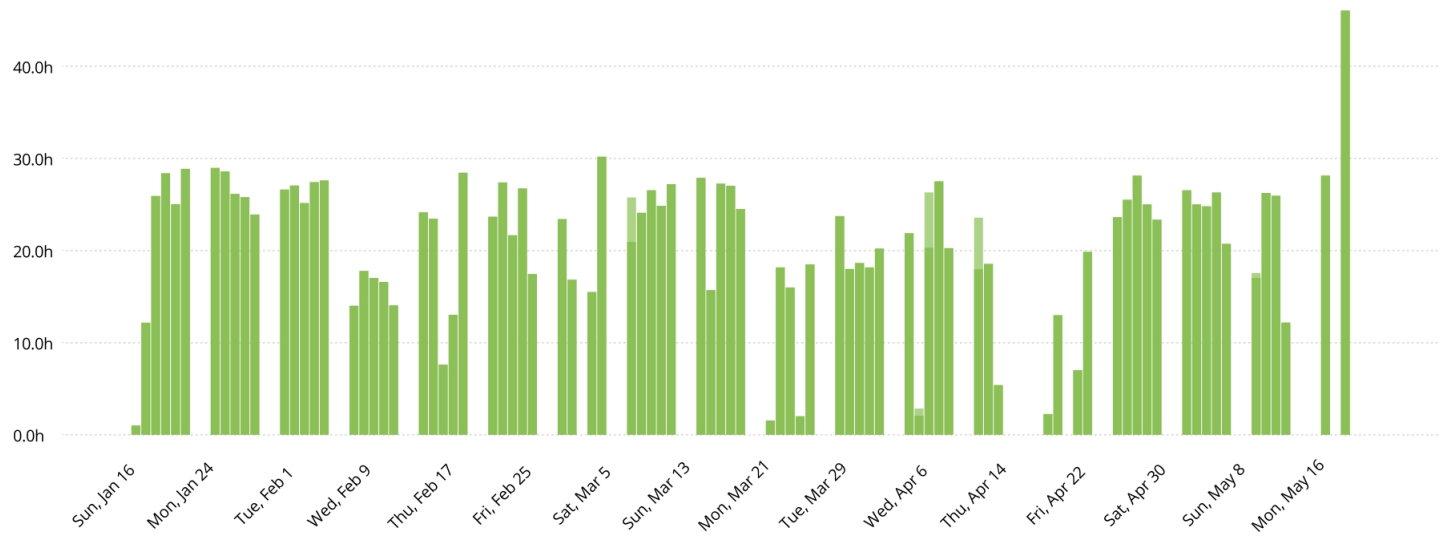


# Summary report

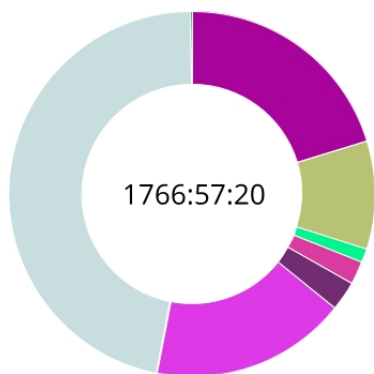
16/01/2022 - 20/05/2022



Total: 1766:57:20 Billable: 1749:11:17 Amount: 23,175.06 NOK



## Tag



|                         |           |        |
|-------------------------|-----------|--------|
| Bug, Development        | 02:26:20  | 0.14%  |
| Development             | 825:26:32 | 46.72% |
| Development, Research   | 01:31:18  | 0.09%  |
| Game Engine Analysis    | 306:59:58 | 17.37% |
| Research                | 46:08:01  | 2.61%  |
| Testing                 | 35:08:38  | 1.99%  |
| Testing, Development    | 21:43:41  | 1.23%  |
| Write Project Documents | 169:14:55 | 9.58%  |
| Write Thesis            | 358:17:57 | 20.28% |

## **F Code Convention Document**

PROG2900 - BACHELOR THESIS

---

# Code Convention Document

Lokførerskolen Sim v2

---

*Authors:*

Thomas Arinesalingam

John Ole Bjerke

Endre Heksum

Henrik Markengbakken Karlsen

February, 2022

---

# Contents

|          |                           |          |
|----------|---------------------------|----------|
| <b>1</b> | <b>Introduction</b>       | <b>1</b> |
| <b>2</b> | <b>Naming Conventions</b> | <b>1</b> |
| 2.1      | Classes . . . . .         | 1        |
| 2.2      | Methods . . . . .         | 1        |
| 2.3      | Variables . . . . .       | 2        |
| <b>3</b> | <b>Documentation</b>      | <b>2</b> |
| 3.1      | Commenting . . . . .      | 2        |
| 3.2      | Doxygen . . . . .         | 3        |
| <b>4</b> | <b>Code Formatting</b>    | <b>4</b> |

---

# 1 Introduction

With the goal of producing a highly understandable and intuitive code base, we declare this document to define the standard coding practices used in this project. Consistent naming conventions, documentation standards and layout formatting are essential for ensuring and maintaining professionalism with emphasis on readability. The standards defined in this document are inspired by the established standard documentation for Unreal Engine, as written by Epic Games<sup>1</sup>.

## 2 Naming Conventions

### 2.1 Classes

All classes should follow the PascalCase naming convention where each word in the variable name should start with a capital letter. Example:

```
class PlayerAbilities {  
  
}
```

The only exceptions to this is the special Unreal Engine Class inheritance naming listed below:

- Classes which inherits from UObject are prefixed by U.
- Classes which inherits from AActor are prefixed by A.
- Classes which inherits from SWidget are prefixed by S.
- Classes that are abstract interfaces are prefixed by I.

### 2.2 Methods

The naming of methods should be used to describe the effect of the method. Or describe the return value if the method has no effect. Methods should follow the PascalCase naming convention.

One exception is for functions that returns boolean variables. these methods should always ask a true/false question such as:

```
HasScored();  
ShouldEndGame();
```

---

<sup>1</sup><https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/DevelopmentSetup/CodingStandard/>

---

## 2.3 Variables

Variable names should follow the camelCase naming convention, meaning the first word of the variable name should be all lower case letters, all following words in the name should start with a capital letter. The words used should be nouns and describe the variable. This practise supports the variable names:

```
// Good naming
int    scoredGoals = 3;
string playerName = "Maradonna";
```

but not the variable names:

```
// Bad naming
int    scored_goals = 3;
string PlayerName = "Maradonna";
```

The letters in constant names should be all upper case:

```
const int MAXGOALS = 32;
```

## 3 Documentation

### 3.1 Commenting

- Comments should be written in U.S. English.
- All code should be self-documenting. If code is considered unclear or bad, it should be rewritten.
- If there is a need for inline comments, they should be simple and descriptive.

**Example:**

```
// Good:
velocity = acceleration * time;

// Bad:
v = a * t;    // calculate velocity
```

---

## 3.2 Doxygen

*Doxygen*<sup>1</sup> is a tool for generating formatted code documentation. The tool reads a folder of source code as input, and generates a document formatted as in readable markdown format including HTML, PDF and L<sup>A</sup>T<sub>E</sub>X. The generator interprets comments that have been formatted in a standard fashion, such that the compiler can convey the comments as explanation or definition of the relevant class or method.

We plan to integrate this commenting standard in all C++-code we produce, to generate documentation of our source code.

**Classes and methods** should be introduced by a comment block starting with two \*'s:

```
/**
 * ... text ...
 */
void myMethod();
```

Classes must include the author(s) and the date of writing. Both classes and methods must include a brief description of its own functionality, and that of any parameters or return value.

**Variables** should be explained by a single line comment prefixed with “///  
”:

```
float gravity; ///  
A constant downwards force
```

The syntax for commenting is similar to Javadoc<sup>2</sup>, containing various tags prefixed with “@”:

**@author** is the author of a class or interface, repeatable if authors exceed one.

**@date** is the author(s) of a class or interface.

**@brief** is a short one-line description of a method.

**@param** is the definition of any parameter for a method or constructor.

**@return** is the return value from a method.

**@see** is a reference to a cross-referenced class or method.

---

<sup>1</sup><https://www.doxygen.nl/index.html>

<sup>2</sup><https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html>

---

## Example of commenting:

```
/**
 * @brief Represents a train.
 * @author John Doe
 * @date January 2022
 */
class Train {
public:
    /**
     * @brief Check if the train has enough fuel.
     *
     * Checks if the train has enough fuel,
     * by comparing its amount of fuel to
     * the fuel needed.
     *
     * @param fuelNeeded The amount of fuel needed.
     *
     * @return True if the train has enough fuel, false if otherwise
     * @see FuelManager::fuelBurnRate
     */
    bool HasEnoughFuel(float fuelNeeded);
private:
    float fuel;    ///< The current amount of fuel for this train
}

```

## 4 Code Formatting

- `nullptr` should be used instead of `NULL`.
- `auto` shouldn't be used, except for:
  - binding a lambda to a variable.
  - iterator variables, where iterator's type is verbose.
  - template code.
- Curly braces should be placed before the line break.

```
// Use This
void MyMethod() {
    ...
}

// Not This
void MyMethod()
{
    ...
}
```



- 
- enum classes should be used instead of namespaced enums.

```
// Old enum
UENUM()
namespace Thing
{
    enum Type
    {
        Thing1,
        Thing2
    };
}

// New enum
UENUM()
enum class Thing : uint8
{
    Thing1,
    Thing2
}
```

- Floating point literals should always have a radix point:

```
// Good
float scale = 0.5f;

// Bad
float scale = .5f;
```

- Control flow using nested if-statements should be implemented as guard clauses:

```
// Good
if(player == nullptr) return;
if(!player->isAlive) return;

player->Destroy();

// Bad
if(player != nullptr) {
    if(player->isAlive){
        player->Destroy();
    }
}
```

## **G Requirements Specification**

PROG2900 - BACHELOR THESIS

---

# Requirements Specification

Lokførerskolen Sim v2

---

*Authors:*

Thomas Arinesalingam

John Ole Bjerke

Endre Heksum

Henrik Markengbakken Karlsen

March, 2022

---

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                         | <b>1</b>  |
| <b>2</b> | <b>Functional Requirements</b>              | <b>1</b>  |
| 2.1      | Use Case Diagram - Application . . . . .    | 2         |
| 2.2      | High Level Use Case - Application . . . . . | 2         |
| 2.3      | Low Level Use Case - Application . . . . .  | 5         |
| 2.4      | Use Case Diagram - Game Engine . . . . .    | 6         |
| 2.5      | High Level Use Case - Game Engine . . . . . | 7         |
| 2.6      | Low Level Use Case - Game Engine . . . . .  | 8         |
| <b>3</b> | <b>Operational Requirements</b>             | <b>9</b>  |
| <b>4</b> | <b>Security and Misuse</b>                  | <b>10</b> |
| 4.1      | Interface Requirements . . . . .            | 10        |

---

# 1 Introduction

This is a requirement specification document for DeskSim v2. It is developed as part of a bachelor thesis by students at the Norwegian School for Science and Technology in cooperation with The Norwegian Train Driver Academy. The general purpose of the project is, as described in the game engine analysis, a migration of an educational train simulator to a modern game engine. The software itself is implemented as a prototype of the aforementioned simulator, with the main purpose of operating a train through a *scenario*. The simulator provides the user with information such as speed limits and train signals along the railway, to simulate a real-world exercise for locomotive driver students. It also includes an editor mode for teachers and administrators, which includes functionality for creating and customizing these scenarios.

## 2 Functional Requirements

The client has stated a list of requirements for not only the main section of the software, but also additional functionality if the group deem this possible. We have decided to pursue these extra features, and have included them as a part of our project roadmap. The simulator should be able to:

- Read input from external throttles for acceleration and braking.
- Move trains along a railway in a realistic fashion.
- Curve and conform railways to the terrain height.
- Display the current speed of the train through a driver machine interface.
- Display warnings and messages to the user.
- Simulate the behaviour of train signals automatically by predefined conditions and scenario configurations.

## 2.1 Use Case Diagram - Application

The following diagram presents the possible use cases of the software. The system contains two actors; a student and an employee. Even though it is not implied in the diagram, the employee acts as an upgraded version of the student role, and has access to everything included in the student role.

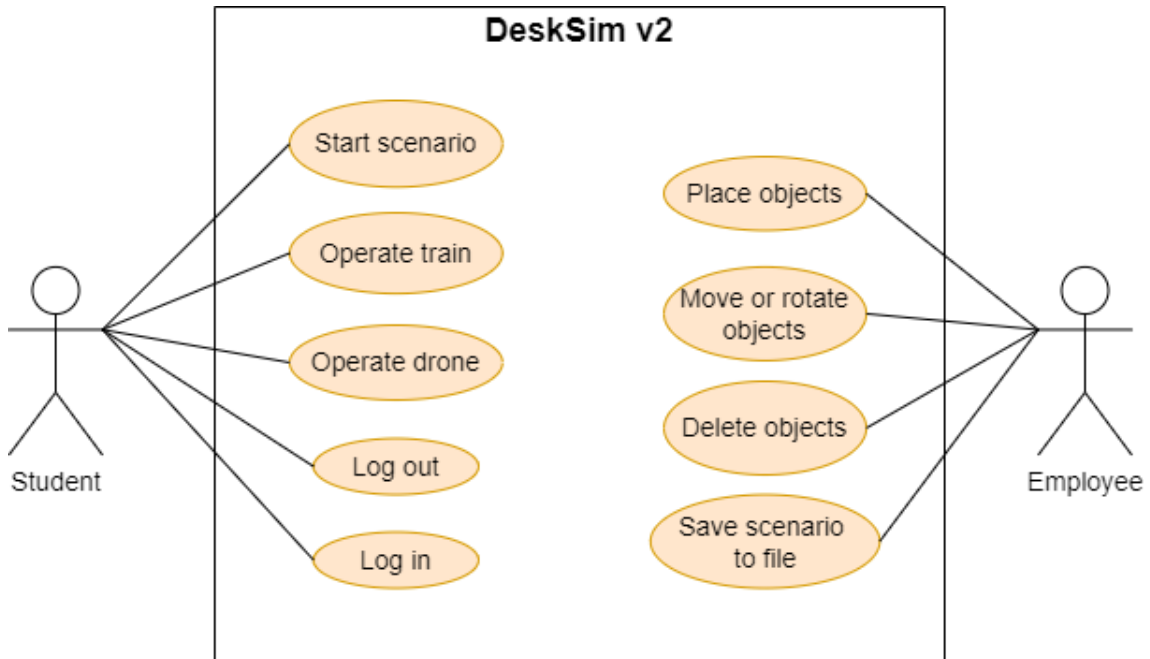


Figure 1: Use case diagram - Application

## 2.2 High Level Use Case - Application

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Place objects  |
| <b>Actors:</b>      | Employee   |
| <b>Goal:</b>        | To place the necessary objects such as a train and a railway in a level.   |
| <b>Description:</b> | When a level is opened in editor mode the user is provided a user interface which includes a content browser. The user can click on a item in the content browser and drag it out in the level. The content browser has different categories the user can select in the top bar by clicking on the category buttons. |

Table 1: Use Case: Place objects

---

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Delete objects  |
| <b>Actors:</b>      | Employee  |
| <b>Goal:</b>        | To delete an object in the level.   |
| <b>Description:</b> | When clicking on an object in a level the user will be given the option to remove it. After the click, the user will get a trash can symbol at the top bar, next to the transformation options. After clicking on the trash can, the program will prompt the user for confirmation before permanently removing the object from the scene. |

Table 2: Use Case: Delete objects

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Start Scenario  |
| <b>Actors:</b>      | Student and Employee  |
| <b>Goal:</b>        | To initialize a simulator scenario.   |
| <b>Description:</b> | In the Main Menu widget view, the user should select a scenario and then left click the mouse button on that object. This will open the correct level and start the scenario. |

Table 3: Use Case: Start Scenario

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Save scenario to file   |
| <b>Actors:</b>      | Employee  |
| <b>Goal:</b>        | Save scenario details to file   |
| <b>Description:</b> | When a teacher has created or edited a scenario. The details of the scenario file should get updated. When the teacher presses the save button in the editor, the scenario details will be saved to a file. |

Table 4: Use Case: Save scenario to file

|                     |   |
|---------------------|---|
| <b>Use Case:</b>    | Log in  |
| <b>Actors:</b>      | Student and Employee  |
| <b>Goal:</b>        | To authenticate user and receive correct access level   |
| <b>Description:</b> | When the program starts a login screen is shown to the user. The user then inputs a username and password, which is sent to a server to authenticate the user. The response contains whether the user successfully authenticated and what level of privilege the user has. The user is then sent to the main menu, and the authenticated level is stored in the program for use. If the authentication fails, the user can try again. |

Table 5: Use Case: Log in

---

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Log out  |
| <b>Actors:</b>      | Student and Employee   |
| <b>Goal:</b>        | To log user out of program   |
| <b>Description:</b> | When the user wants to exit or log out of the game, the info of the logged in user is not saved to any file. If the player chooses to log out of the game without closing it, the previous user data is removed and the user can log in again. |

Table 6: Use Case: Log out

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Operate drone  |
| <b>Actors:</b>      | Student  |
| <b>Goal:</b>        | Maneuver the drone camera  |
| <b>Description:</b> | When the student clicks the drone view button "2", the student can freely move around in the scenario using W, A, S and D for forwards, backwards and horizontal movement, Q for downward and E for upward movement. The user can also use the mouse to freely look around in the scenario, and change the movement speed with the scroll wheel. |

Table 7: Use Case: Move or rotate objects



---

## 2.3 Low Level Use Case - Application

|                          |  |
|--------------------------|--|
| <b>Use Case:</b>         | Move or rotate objects   |
| <b>Actors:</b>           | Employee   |
| <b>Goal:</b>             | To move a object or rotate it into the position and position you want  |
| <b>Precondition:</b>     | The user has successfully opened a scenario in editor-mode   |
| <b>Success Scenario:</b> | <ol style="list-style-type: none"><li>1. The employee clicks on the object he wants to edit.</li><li>2. The object will display a gizmo, either in the form of arrows the move it along either it's x, y or z axis, or a wheel to rotate.</li><li>3. The user changes the mode to the one he want from the top bar icons.</li><li>4. For translation:<ol style="list-style-type: none"><li>(a) The user hovers the mouse over the gizmo arrow to select one axis, or in the middle of two gizmos to select a plane and presses the gizmo.</li><li>(b) The user drags the mouse to the position he wants the object to be located</li></ol></li><li>6. For rotation<ol style="list-style-type: none"><li>(a) The user presses the wheel and drags the mouse around the wheel to get the desired rotation.</li></ol></li></ol> |

Table 8: Use Case: Move or rotate object

|                          |   |
|--------------------------|---|
| <b>Use Case:</b>         | Operate Train   |
| <b>Actors:</b>           | Student   |
| <b>Goal:</b>             | To drive the train in a scenario  |
| <b>Precondition:</b>     | The user has successfully opened a level and the levers is connected to the system through a USB port   |
| <b>Success Scenario:</b> | <ol style="list-style-type: none"> <li>1. The user pushes the left lever or "w" key to accelerate the train.</li> <li>2. The user pushes the right lever to apply break force on the train.</li> <li>3. Depending on the scenario, the user has to follow some rules: <ul style="list-style-type: none"> <li>• The user should not exceed the speed limit. Doing so should result in system regulated brakes turned on.</li> <li>• The user is provided information about the current speed in the Driver Machine Interface.</li> <li>• The user should follow the rules regulated by signals: <ul style="list-style-type: none"> <li>– Main signal - If this signal is red the user should stop. If user don't stop before the signal this should result in breaks turned on.</li> <li>– Main signal - One green light means that the user can drive with reduced speed</li> <li>– Main signal - Two green lights means that the user can and should continue with the set speed.</li> </ul> </li> </ul> </li> </ol> |

Table 9: Use Case: Operate Train

## 2.4 Use Case Diagram - Game Engine

The following diagram presents the use cases of the game engine which got facilitated by the development of the application. The system has one actor; a developer.

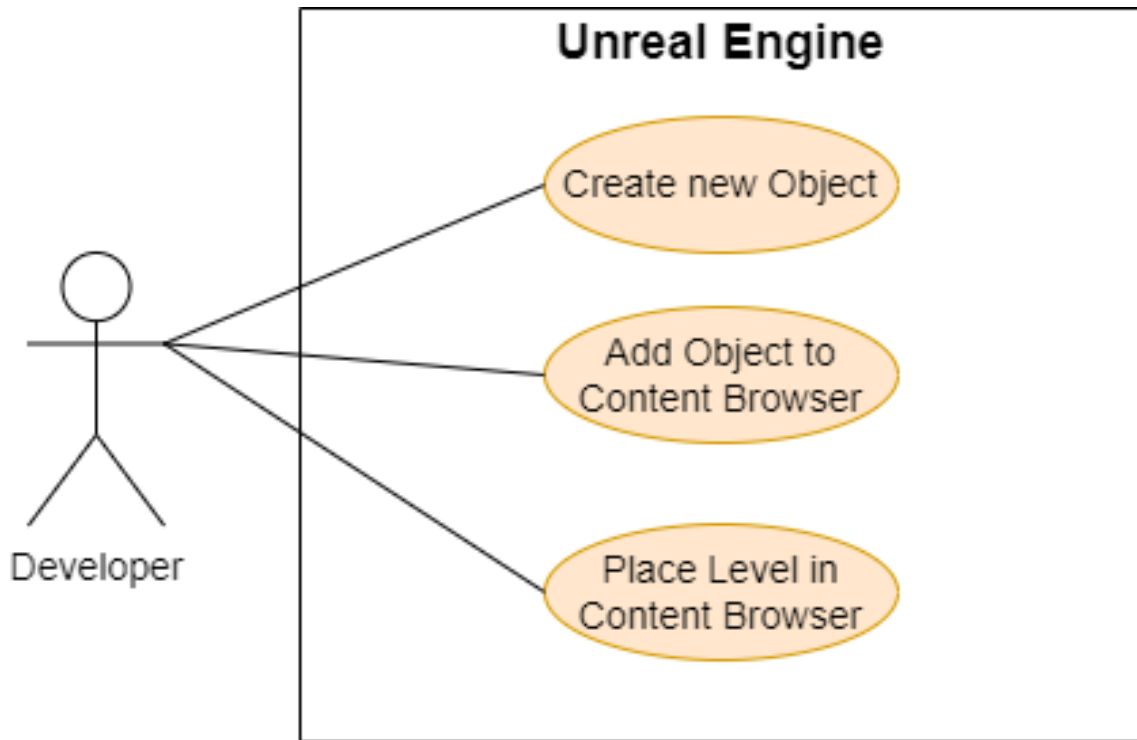


Figure 2: Use case diagram - Game Engine

## 2.5 High Level Use Case - Game Engine

|                     |  |
|---------------------|--|
| <b>Use Case:</b>    | Add Level in Main Menu   |
| <b>Actors:</b>      | Developer  |
| <b>Goal:</b>        | To add a level created in unreal engine to the simulator..   |
| <b>Description:</b> | When the developer has created a scene he wants to be a part of the simulator he must know the name of the level. The level name is stored as a FName, and the content browser only need its value. The FName's are immutable and case sensitive so it's important to have the right name. Open the MMObjects inside <i>BP_EditorHUD</i> blueprint located in " <i>DeskSim V2/Source/DeskSim V2/Editor/UI</i> ". The developer now clicks the + button to add the new level and fills in the name, description and the FName reference to the map. |

Table 10: Use Case: Add Level in Main Menu

---

## 2.6 Low Level Use Case - Game Engine

|                          |  |
|--------------------------|--|
| <b>Use Case:</b>         | Add new object   |
| <b>Actors:</b>           | Developer  |
| <b>Goal:</b>             | To add a new object to the game  |
| <b>Preconditions:</b>    | The developer has a working version of Unreal Engine version 4.27.2 or higher. The developer has a 3D model he wants to be added in the game.  |
| <b>Success Scenario:</b> | <ol style="list-style-type: none"><li>1. The developer uploads the model to the "models" folder inside unreal engine</li><li>2. The developer navigates to "C++ classes" in the content browser.</li><li>3. The developer right clicks on either "BasicStaticObject", "Train" or "BasicSignal" or "wagon", based on what item type the object is.</li><li>4. The developer clicks on "Derive blueprint from c++ class..." in the drop-down menu and selects the appropriate place to store the blueprint.</li><li>5. The developer opens the blueprint and drags the imported model from step 1 into the "Static mesh" variable in the details panel for the object.</li><li>6. The developer compiles and saves the blueprint</li></ol> |

Table 11: Use Case: Add new object

|                              |  |
|------------------------------|--|
| <b>Use Case:</b>             | Add object in Content Browser  |
| <b>Actors:</b>               | Developer  |
| <b>Goal:</b>                 | To successfully add a created object in the content browser making it clickable and draggable in runtime.  |
| <b>Preconditions:</b>        | The developer has created a new object as described in the "Add new object" use case.  |
| <b>Success Scenario:</b>     | <ol style="list-style-type: none"> <li>1. The developer open the <i>BP_EditorHUD</i> blueprint located in "<i>DeskSimV2/Source/DeskSimV2/Editor/UI</i>".</li> <li>2. The developer clicks on the + icon for the CBFObjets.</li> <li>3. The developer adds the category the actor should be a part of.</li> <li>4. The developer writes a suitable name and description for the object</li> <li>5. The developer adds the reference to the actor he wants to include.</li> <li>6. Close unreal engine desktop and visual studios and navigate to the file system for the project. Right-click on the <i>DeskSimV2.uproject</i> files and select "generate visual studios project files" from the dropdown menu.</li> <li>7. The item should be visible and draggable in-game in the content browser.</li> </ol> |
| <b>Alternative Scenario:</b> | <ol style="list-style-type: none"> <li>7. Open visual studio and right click on DeskSimV2, select "rebuild" and let the solution rebuild.</li> <li>8. When its built, open DeskSimV2.uproject and check if the object is added.</li> </ol>   |

Table 12: Use Case: Add object in Content Browser

### 3 Operational Requirements

These are the requirements which concerns the application at it's operational stage, this stage begins at the project's deadline which is the 20.th of May:

- 
- The application must be able to interact with the existing Rest-API hosted by Lokførerskolen.
  - The system must operate on Windows devices.
  - The system must manage privileges of users and only allow elevated users to access the editor functionality.
  - Must be transferable through a zip file
  - The system must operate on computers which has 8GB of ram and a Intel® Core™ i5-4460 CPU or better.
  - Should not experience frame rate drops of lower than 60 frames per second.

## 4 Security and Misuse

To ensure the security of the users and avoid misuse of the application the application:

- must require user authentication for usage. The authentication process should be a token based system where you receive a token. The token has an expiration and should be used to authenticate the user up to its expiration. When the token expires the user will be asked to authenticate again and receive a new token.
- should not contain bugs and/or security flaws that potentially could lead to harm or destruction of hardware components. Such flaws include bad memory handling.
- must not store any passwords in plain text.

### 4.1 Interface Requirements

#### Menus

- It should be intuitive and easy for a student at Lokførerskolen to navigate the Main Menu.
- The Main Menu should have the same functionality as the previous simulator and only deviate by design.
- All text must be available in Norwegian.
- Buttons should be intuitive to reduce the number of operations required for a task.
- The DMI viewport in a game should be responsive to the gameplay.
- All numbers and measurements must be specified in the metric system.

## **H Meetings and Notes**

PROG2900 - BACHELOR THESIS

---

# Meetings and Notes

DeskSim v2

---

*Authors:*

Thomas Arinesalingam

John Ole Bjerke

Endre Heksum

Henrik Markengbakken Karlsen

May, 2022



---

# Contents

|          |                                 |           |
|----------|---------------------------------|-----------|
| <b>1</b> | <b>Client Meetings</b>          | <b>1</b>  |
| <b>2</b> | <b>Supervisor Meetings</b>      | <b>5</b>  |
| <b>3</b> | <b>Sprint Planning Meetings</b> | <b>13</b> |
| 3.1      | Statuses . . . . .              | 13        |
| <b>4</b> | <b>Daily Scrum Meetings</b>     | <b>26</b> |

---

# 1 Client Meetings

---

**Date:** 17.01.2022

**Present:** Endre, Henrik, John Ole, Thomas the other group and Client.

**Agenda:** Going through the project description with the client and ask questions.

**Meeting Log:**

- Everyone shortly presented themselves before we went through the project description.
- There will be two weeks between every client meeting, we must have our sprint goals ready for this.

## Questions

**What is meant by reproducing the elements in desksim?**

*You should be able to recreate all the functionality in desksim.*

**What format is the 3D object?**

*glTF files, some are in ac3D, when new models are created it is glTF.*

**What operative system is used?**

*Windows.*

**How should we ensure the external throttle support?**

*It is best to develop the controllers as the one they will be using. The client will check the possibility for us to borrow some of their used damaged ones.*

**Could we visit the School and get a tour of the simulator?**

*They are also scheduling for us to come and get a tour of the school and get a proper demonstration of the current simulator with covered expenses.*

**How will their code be shared with us?**

*The client has uploaded to a git repository to share with us after the contract is finished.*

**How should the game engine research be performed?**

*Comparison of 2-3 pages, but it could go beyond that.*

---

**Date:** 24.01.2022

**Present:** Endre, Henrik, John Ole, Thomas and Isak (Client representative)

**Agenda:** No Idea

**Questions:** Ask what they think about the user testing perspective. Scheduled

---

---

meeting times and update project plan. Do you see the need now or in the future to use networking (multiplayer cross devices) in the application? Do you want internalization? How to implement VR? Would you like to get a overview on how we plan to work for the ongoing period.

**Description:**

- We must fill in the confidentiality agreement.
- The title of the project is not yet decided but could be something like: DeskSim; Simulator for Lokførerskolen.
- Lokførerskolen will cover travel expenses for the students to and from their facility in Oslo.

Arbeidstittel: Desksim simulatoroppgave for lokførerskolen. Eksterne virksomheter sine plikter: Dekker transport til og fra skolen med buss eller tog (Mulighet for bil?). Og dekker utgifter dersom det er avtalt, utlån av spaker. Opphavsrett: Diskusjoner om det nå internt, tror de lander på at de har bruksrett og at vi har eiendomsrett, alternativ A. Trenger ikke å være unndratt offentligheten og signere avtalen. Fysisk møte hos lokførerskolen den syvende februar 2022. Relevans med testing: Brukertesting muligens ikke relevant fordi det ikke kan læres intuitivt. Meetings with client every other week from week starting this week from 9.am-10.am. The existing software has some scenarios using shiftin, drive trains back and fourth and changes equipment. One in VR (the changer), and one on the screen. A local server starts on the pc, two instances. In the future they want to use networking to get the functionality. One vr one on pc. Per idag så tas ingen hensyn til språk, det er et krav for å komme inn på lokførerskolen at du har gode norsk kunnskaper.

---

**Date:** 07.02.2022

**Present:** Henrik, John Ole, Thomas and Isak (Client representative)

**Agenda:** Tour of the school and demonstration of the current simulator.

**Questions:**

**Description:**

We went to Lokførerskolen and got to try the simulator in use today at the school. We got some answers to trivial questions about the simulator and our implementation.

---

**Date:** 21.02.2022

**Present:** Endre, Henrik, Thomas and Isak (Client representative)

---

**Agenda:** Feedback on the game engine analysis, train signal explanation and showing off our progress

**Description:**

The client said that the analysis was way above what he expected from us. We got a fifteen minute explanation on how three different train signals work and tips on how to implement them. We also showed our progress so far and the client seemed pleased with the amount of progress we have at this point. We informed the client that our MPV will be ready for the next meeting and we agreed to send the executable before the meeting.

---

**Date:** 07.03.2022

**Present:** Endre, Henrik, John Ole, Thomas, Tom (Supervisor) and Isak (Client representative)

**Agenda:** We sent the MVP to the client on last week. He has now tested it and will give feedback.

**Questions:** What in the MVP would you like us to change or further develop? ask what they want us to develop now, regarding the MVP. how will the sub goals will be implemented, as in-game functionality, or in-editor features.? If we get sort of free reigns, ask their thoughts of implementing height maps for generating environment?

**Description:** *Feedback on MVP*

He thought that the MPV was good, and was impressed with the amount of work we had done in the time we had.

The aspect missing from the main goal from the MVP is a running scenario where at least one signal is working. The client said that ideally, the tool should be able to define curvatures for the landscape and stretch a spline along that curvature in a natural way. The client said that in the long term they want their own runtime editor. The client desired manually changing the landscape rather than a heightmap. The client also emphasized that they want a dynamic scenario builder where they can take sections from different landscapes and combine different to create a scenario.

The client said that they must have way of running different scenarios. That could be for example reading a pre-configured .xml file.

---

**Date:** 21.03.2022

**Present:** Endre, Henrik, John Ole, Thomas and Isak (Client representative)

**Agenda:** Discuss alternatives to the terrain editing task. Those alternatives could be: A log in system, communicating with the existing API. Teacher/Student multiplayer. Logging of relevant user data from a scenario or to add more functionality to the existing work.

**Description:**

---

---

We discussed what task would be most relevant to choose from from the client's perspective and found out that sign in functionality would be the most relevant. The client explained to us how the system they are using are working and described it like this:

- User types username and password in the software
- The system send an api call to Lokførerskolens api.
- The user recieves an json web token which is valid a certain amount of time.

The client was open to suggestion and for us to research different merhods and industry standards on the functionality. Our plan is to research and find out the best way to solve it and then discuss this with the cient and later implement teh agreed upon solution to our application. After the meeting we got an Email saying that we are allowed to use the client's api to test the solution through their endpoint's.

Other functionalities that could be relevant if we have time are: More complete demo, a savestate, amking pausing and rewinding scenarios possible.

We discussed user testing options and found out that lokførerskolen was open for usto come and test the application of their computer and to test it on some of their students.

---

**Date:** 19.04.2022

**Present:** Endre, Henrik, John Ole, Thomas and Isak (Client representative)

**Agenda:** Show the progress and ask some questions to the client

**Questions:** User testing of scenario and the editor. Dates? between the 27Th and the 6Th of May.

**Description:**

We displayed the current progress so far, where the new functionality was the object placing tool and saving a scenario. We also displayed the progress of the spline editing tool. The client seemed happy with the amount of progress.

id, username, status, operator (eget brukerobj), subscription id, usergroupnames (brukergrupper ), voippassword,

Showed the editor mode and the

Log in - Can log in but not saving the user.

Can send a password and username and recieve a json webtoken.

User testing: første uka i mai, skal høre når det passer. den uka skal vi kjøre det. Editor modus

Deployment: Per nå er det enkleste at den kan innosetup. Drømmescenario, kjøre det i nettleseren!

---

Oppdateringer er å innstalere på nytt per nå. Burde skrive i rapporten noe om dette.

Planen vår videre er å fullføre editor, spline og log inn.

Brukertesting, kjører gjennom et scenario: Tips: gjør det enkelt, kjøre et stykke, stopp signal før det blir grønt også kjøre videre. Kjøre et stykke, du er i mål. Startog mål burde være et. 2.May 10:00

---

**Date:** 09.05.2022

**Present:** Endre, Henrik, John Ole, Thomas and Isak (Client representative)

**Agenda:**

**Questions:**

**Description:**

brukertesting med klienten

slet bittelitt med translation, mista "grepet" på gizmoen venta på at den skulle tilbake til menu etter advarsel om du har saved kom opp, skjønte ikke at man måtte trykke på menu igjen ca 40-50fps(lavt?)

legge til isak til github, slik at han kan forke prosjektet når vi er helt ferdig med dev

## 2 Supervisor Meetings

---

**Date:** 12.01.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Some questions we had regarding the project and just a casual discussion with the supervisor

**Description:**

The fact that the project is split into an analytical part and a developing part does not necessary change how we should look at the work methodology because of this, although different methodologies can be used if we find it sensible. There is no problem writing the report in English if we want to from the NTNU's side, but we should take into consideration what the client thinks about this. We will get a walk through on the integration project in the next supervisor meeting to get to know what we could have done differently and better. We got advise on some other bachelors that we could reed which relates to our assignment or just good assignments. With regards to Latex, we got told that this is a good tool to use and something that is great to have experience with. Them that wants, should use

---

---

Latex. The source control versioning program we choose is kind of irrelevant. The only note is that if we don't use the school provided GitLab, we are responsible if the service is down or out of service, but this rarely or never happens. Should also take the client preferences on this matter. Both of the technologies have been used earlier. We got told that we should, early on, write down and have discussions on some of the choices we make, not all, but a selected amount. We will have weekly meetings with the supervisor at 13:30 - 14:00, otherwise we have to plan exceptions. The referential style we choose is not important if we use it consistently throughout the project. We should ask the client if they are willing to pay for the visit. the contract between us and the client must be printed out and signed on.

---

**Date:** 19.01.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (supervisor)

**Agenda:** Going through our delivery in the integration class. Going briefly through the project plan, and ask about all the questions we had after the meeting with the client.

**Description:**

**The Integration Project**

The demo was good, the report was a bit weaker. It was good that we had a lot of tests and covered many cases. Negative that we heavily relied on known resources, should have challenged us self more and made some parts more complicated. The implementation was a bit easy, and not maybe the most relevant. We were a bit defensive on how to do stuff, asking the supervisor and not having opinions ourselves. Should focus on making more of our own choices. Should have discussions with focus on the specific objective details. We had overall too few references, so we should focus on references in the bachelor. Missing discussion around code, and time. Make sure that all requirements are right.

**Project plan:** What does lokførerskolen want to get out of this? This is an example of an effect goal. Be concrete on roles, who does what? More clarity on group rules, specify what happens. Git workflow should be on Quality assurance. It is regular to plan vacations, but mention it in the plan. We should decide when the status reports should have due date. Decision points is the time we need to make some key decisions. The confidentiality agreement can be signed between only us and the client. But also needs the standard agreement between client, us and NTNU.

**The game engine analysis:** Make it objective and reference based! It should be a deep analysis, but short report.

**The question round and answers:** We decide on when and how to deliver the status reports. The decision points are when we will make the major decisions that will impact the project further The project agreement should be in the NTNU standard. If the client has the need to require a confidently agreement they should make that separate and be just between the developing team and them.

---

---

**Date:** 26.01.2022

**Present:** Endre, Henrik, Thomas, John Ole and Tom (Supervisor)

**Agenda:** Get feedback on the project plan. And a question round.

**Questions:** What should be the text size and line spacing? Any notes on testing? Should the length of the meetings be documented in the meeting log? Should we reference the task description from the client in the analysis? How should we write meeting notices? Estimating issues. 1day, 1 week, 1 month, is that good? We want physical supervisor meetings from next week.

**Description:**

**Project Plan:** Fewer administrative milestones if possible. Decided to remove the first effect goal into result goal and

Line spacing: 12, 1.5 - You could deviate a little but not ;8 or ;16, just to fit with the requirements.

Testing aspect: Ask lokførerskolen if there is an option to get the software tested on some of their students for user testing

Length of meetings: Not required, document the decisions made instead.

Reference Description: Just state what requirements were provided from Lokførerskolen and not reference the actual document.

Meeting Notices: Made some meeting notices for important meetings, thees should be in the project's appendix..

Estimation of issues: In our project, the sprint is the object for the estimation. hour and days. S M L XL, take these on the issue, in a normal sprint we should. Do planning poker.

Physical meeting from next week: We have planned i physical meeting with the supervisor for Thursday 03.02.2022 if he is able to come to campus. The supervisor will inform us about this in Tuesday. If there's no physical meeting we will have the meeting on teams at the usual time on Wednesday.

---

**Date:** 03.02.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Feedback on the project plan. Asking questions we have. Going through the analysis

**Questions:** Should we have a section about developer feedback? How to write objective about the Documentation for each engine?

**Description:**

On the physical meeting on Monday 07.02.2022 we should clear out the relationship our report should have to the other group's report. Should we do things differently? Have different focus points. Important what the client opinions are. We should also find a name for the product we are deelooping.

**The Project Plan** It is not necessary to have three point numeration in documents that is only 15 pages long. The first goal section should be named result goals because we name the other categories. We should state in the plan and in the bachelor report what the distribution between the different tasks stated in the task



---

description. Analyse vs Implementation and Implementation features vs each other. The last effect goal is not necessary as it is a bit far fetched to include. Include the date in which the deadline is in the constraints. The goals and the task description should be consistent with each other, if you have stated the thing as a goal, the task description should not diverge from that. The role Research Manager should be considered renamed to Analysis Manager. We should describe the role our client has in the project with more detail. How are they going to be present in the project? Should they take part in the scrum meetings and how will they review in the development process. In the section about methodology we should talk about more details like kanban vs scrum. More candidates and the criteria of why we choose scrum instead. Describe what we will do to migrate poor risk management - allocate time for it each sprint and other aspects. We should plan and describe how we are going to perform the acceptance testing.

**Questions** We were encouraged to include the material we have gathered from developers in the analysis of the engines. Writing objective about documentation could be performed by looking at aspects like overall structure and quantity/quality.

---

**Date:** 10.02.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** The Game Engine analysis

**Description:**

We got feedback on game engine analysis is that we should decide how much it should count on the grade and how we should reference the analysis. If we should have it in the assignment. We should also have a section for describing what approach we had to writing the analysis. The most important aspect is to figure out how to incorporate the analysis in the bachelor thesis.

---

**Date:** 23.02.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Show jira and general questions

**Description:**

We showed jira and the supervisor thought the issues and layout of the project looked good. We talked about the estimations should be included mostly to be used in planning the next sprints and knowing how many tasks you can do in one sprint.

We should always state plug-ins and solutions we borrow from others and discuss whether these solutions are good. We should also state in the final report if we used forums. The supervisor informed us about the upcoming course hosted by NTNU on the report writing.

---

---

**Date:** 17.03.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** General progress discussion and feedback on the status report and requirement specification document

**Description:**

The first order of business was the group bringing up some concerns about one of the tasks we have started on in the project. The task it refers to is the editing tool of the landscape we started working on in sprint 7. The feedback we got is that it could be a risk trying to use much effort in a task like this in the project. Because of its complexity and magnitude its possible that it could cause the group to get stuck and/or use most of the time left in this single task. Because of this uncertainty the group find it reasonable to approach the client with the issue and discuss an alternative solution. After all, the task is not a part of the initial goals from the client.

We should document, and host at least three retrospective meeting where the agenda is the most recent sprints and discuss the progress so far, the issues and their estimations and always try to elevate the quality of our process by learning from the good and bad things we did.

**Status Report** It seems like we have a good connection to the client and the supervisor stated the importance of this. He told us that if we use different plug-ins in the project that we should always try to give some quality assurance for this. Is the plug-in sustainable, will it be maintained and all the other factors that may affect the client when they take over the development and maintenance of the application. The supervisor re-stated his advise that we should code as much as possible, even though some design must be performed to create the application it is important to get as much quality code as possible.

**Requirement Specification** The introduction was nice, the only thing missing is an introduction to how the different requirements were made. Was them there in the beginning of the process or has the project gradually expanded and occurred over time? We should also introduce the use case diagram. The extended use of the "extend" keyword in our second iteration was addressed. The supervisor informed us that the usage of this should be when an already created system gets extended with some additional functionality for some users.

The most crucial aspect of the feedback we got was that we should broaden the requirement specification. There are some missing parts about various system details and user types that is not included in this iteration. The missing users are the admins and teacher. We got told that many of our use cases missed details. For example the "Place Object" use case did not specify what the objects are nor the way we access the objects. Does they get created by the user, are they pre-made and stored in a file system? Those aspects of the use case should be described. In the "Fail Scenario" we should also list the system response to the scenario failing.

The non-functional requirements should be more testable. We should also list the deployment requirements for the application. Should the application be ran on all computers, or only the ones at Lokførerskolen. Should also discuss the security

---

aspects of the application.

The requirement specification document should in general be filled in with much requirements over the entire system and we should prioritize the tasks we do. We should bring this subject up with the client to discuss the progress and what to include in the scope.

In the next meeting we will discuss what was considered as options with Lokførerskolen and how we have decided to expand the project.

---

**Date:** 21.03.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Talk about the plan to tackle multiple users. Admin, User. Evt the thesis.

**Description:**

---

**Date:** 24.03.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Free meeting where we asked some general questions

**Description:**

We asked the supervisor about testing and what he thinks of our plan to perform users tests on some students at Lokførerskolen. The supervisor said that it is important to make sure that the tests are objective and focuses on the parts of the simulator we want feedback on. We want the feedback for further development so we should make sure the test's can give us some feedback on this.

We should discuss with Lokførerskolen how they want us to deploy the simulator at the end of the project. If they want it as a zip-file or other solutions and then update the requirement specification.

The abstract and "Sammendrag" section in the report should be the same text, not directly translated, but include the same information.

The supervisor empathised on using figures to illustrate what we talk about in the final report.

---

**Date:** 31.01.22

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

**Agenda:** Feedback on status report 2 and the requirement specification document

**Description:**

**Status report 2** The section about the terrain generation was a bit narrow, at

---

---

least to use in the final report. We said that the original discussion and alternative solution was included in the status report but excluded because of its length. Its important to not only say that we excluded a planned task, but explain why, and provide alternative solutions in the final report.

In the plug-in's section we have to provide information about the quality, maintenance and future plans of the plug in. We should reason on why we use it and ensure that it will be maintained.

The status report says that we stopped with daily scrum meetings, but the supervisor missed the part about why we stopped with them, when we stopped and how it is better without them.

We should have more retrospective meetings, we have had some and although we have not called them retrospective, we have always after a sprint discussed the progress and where we are. It could be an option to go back in the scrum planning meeting and change the layout to ensure that readers know that the retrospective aspect of the project are present and to highlight their place in the report.

The supervisor will have a full Easter brake so the next meeting will be on Wednesday the 20Th at 15:00.

### **The Requirement specification**

The operational requirements are missing and could include such requirements as log in requirements, cpu requirements and software crash requirements. The functional requirements was good, the only note was that some of them could be merged together, maybe not that important for us since the diagram and information does not include extensively amounts.

The use case should be clearer on the object description. "Place an object" - what does it mean, in editor, runtime, give better descriptions. The use cases should display all the work we have done, it may be necessary to include the facilitating work we have done in editor to make further development easier for the client. If Lokførrerskolen will further progress the project we must describe the engine specific use cases. The supervisor thought that the "Operate drone" use case did not need a low level description and that maybe create scenario would be better to include as a low level. It was also emphasized that the use cases should be a guide for developers in the development phase of the project.

We should include a update plan. If the software will be updated how will this be done. We discussed on this for a while and figured that since we are creating editor functionality without networking functionality we cant include it in the project scope, but we should show in the report that the distribution issue is discussed and what solutions we finds to be the best.

---

**Date:** 20.04.2022

**Present:** Endre, Henrik, John Ole, Thomas and Tom (Supervisor)

---

**Agenda:**  
**Description:**

We showed off what we have been working on since the last meeting, including loading levels, the editor mode with the content browser and gizmos, saving and loading of scenarios. We spoke about the relevancy on creating elements that emulate already existing functionality from Unreal Engine. We have prioritized creating this functionality as it is a goal set by the client, but we are considering concluding that it is a waste of effort, and are prepared to reallocate our time to work on something else.

Notes:

Remove the third level of chapters in the table of contents. Should the theory part contain a discussion about using games/simulators in education. Lokførerskolen should be introduced before the engine analysis. Explain the sizing/importance of the analysis. End the game engine analysis less abruptly. "Bråslutt, bør ha 3.1.7 anbefaling og konsekvenser anbefalingen har for både lokførerskolen på lang sikt, Selve utviklingen dere gjør og Innholdet i prototypen dere utformer." Use cases: write more about the content browser "has different categories" Move or rotate objects: try and fail scenarios, error messages

Add new object: nærmere brukermanual Use cases, write more about what the user can do, not how they do it

More specific requirements, measurable numbers instead of vague wishes

Add more screenshots and figures, show the game to people reading. Display trains and context before delving into code and technicalities. Read from the point of view of an outsider that hasn't seen the software running.

Not ER, but class diagram diagram to show all classes made by and used by us ingame

Keep system architecture on a holistic level, explain technicalities such as protocols and code later in the report

Design choices; we chose to use c++ instead of blueprint? how actors communicate, why actors and stuff inherit from the stuff they do,

placement of development process can be changed

Talk about what Unreal does for us, and what we have made what can they use to build further on, what is just for demo

weighting in the report

smart to look at other thesis papers

its ok to fix stuff for demo after thesis paper deadline, focus on paper then fix

13:15 4. Mai

---

11.05.2022

Tips: - Les emnebeskrivelse - Leser kjenner ikke caset - Tittel? - Lage abstract - nummerering holder med 2 eks: 3.3 ikke: 3.3.3 - Legge ved timeliste - Diskutere rundt avik fra plan - fordele figurer jevnt rundt i rapporten - Sjekk for konsistente referanser - Sjekk språk! - unngå å skrive "at vi gjorde det..." "Vi jobbet fulle dager tirsdag, onsdag, torsdag, dette fungerte fint for oss" heller "GRuppen valgte å ha tre dager i uken dedikert til arbeid med bacheloroppgaven" - Få frem det som vi er stolte av, ikke så mye om hva som gikk feil

Theory - eller bakgrunn rundt spill - hva er spillets rolle for lovførerskolen? - hva skal de bruke det til?

Trimme rapporten, ikke ha med tomme sider Vite mer om arbeidsgiver

Game engine analysis - er informasjonen riktig?

Mer diksjon gjennom rapporten Product backlog - knytte det med use casene, er vært case ett issue?

Hvorfor valgte vi scrum, fortelle casen bak valget Methodology - hvordan vi brukte scrum

sette implementation før deployment gi et oversiktig bildet over modulene i produktet

Editor mode - kilder? oppsummere hvor mye kode er skrevet, hva er gjort fra bunn og hvor lett kan lovførerskolen bygge videre op produktet

sjekk kravspekk når vi tester

Disussion - hente resultat mål

Kilder, sjekk at det er konsistent

## 3 Sprint Planning Meetings

The scrum meetings are written with three different section. A section about the Goals we have for the sprint. A section we write when the sprint is completed, which covers the result of the sprint. And a retrospective section where we will discuss and reflect on the result of the sprint with the goals in mind.

### 3.1 Statuses

The issues in a sprint can be set to one of five different statuses. The statuses are respectively: **Wishlist** - The issues that is taken out of the project scope and in to a list of functionality that we can decide to include if we have time later in the project. **To Do** - This are the section for issues that are waiting to be developed. **In Progress** - The issues that are currently being developed. **Done** - The issues

---

that has been completed in the sprint.

---

## Sprint 1

**Date:** 18.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 17.01 - 31.01

### Sprint Goal:

- Finish the project plan and the initial product backlog.
- Finish the game engine analysis
- Deliver client contract to NTNU and sign Lokførerskolen's confidential agreement.

**Sprint result** We finished the product backlog and got feedback on the delivered project plan. The project plan is good for now and only small changes needs to be done with it. Not necessary to deliver it again. We started to write the game engine analysis, but did not finish it. We have to move the issue to the next sprint.

|           | In progress           | Done  |
|-----------|-----------------------|---|
| Issues    | Write Engine Analysis | Project Plan<br>Client Contract<br>Research Engines |
| Estimates | 4                     | 4 + 1 + 2   |

Table 1: Overview of issue status at the end of sprint 1

**Retrospective** The Game Engine Analysis did not get completed in the planned amount of time. The reason for this was that we did not have the necessary knowledge on the complexity of the task. We thought that it would be many references and other papers we could base our analysis on, but found out that we have to do most of the work our self if we want the quality and objectiveness we want to give Lokførerskolen and our group the best starting point for selecting the game engine.

---

## Sprint 2

**Date:** 30.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

---

---

**Period:** 31.01 - 07.02

**Sprint Goal:**

- Finish writing the game engine analysis
- Start on the code conversion document
- Set up a Unreal project with git

**Sprint result:** The game engine analysis is only missing the conclusion and a round of reviewing, so we expect finishing it in on next Tuesday. The code convention document was started, but not finished because of the focus put to finish the analysis. We have to move the setup of unreal to the next sprint.

|           | To do                                      | In progress              | Review                |
|-----------|--|--------------------------|-----------------------|
| Issues    | Setup Repository<br>Integrate Git and Jira | Code Conversion Document | Write Engine Analysis |
| Estimates | 2 + 2                                      | 2                        | 4                     |

Table 2: Overview of issue status at the end of sprint 2

**Retrospective** The sprint went worse than expected. We actually thought we would finish all the three tasks, but it was the game engine analysis that is the reason for the delay. We have now written 14 pages and every page is consistently written to ensure the integrity and objectivity of the paper. The good news is that very little are remaining and that we probably will be able to start learning the engine and begin the basic implementation next week. One of the milestones we had set was the engine setup, we will most likely save time on this milestone because the analysis has given us the knowledge we need to do this.

---

### Sprint 3

**Date:** 07.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 07.02 - 14.02

**Sprint Goal:**

- Review and deliver Game Engine Analysis to the client and to ur supervisor



- Integrate Jira with GitHub
- Setup the repository in GitHub with an Unreal project

**Sprint result:**

This is the first sprint that we actually managed to get all our goals finished in time. We managed to finish the setup and integration with Jira and this results in a decrease of time delay from our original plan to be finished with the MVP within two weeks.

|           | Other statuses | Done   |
|-----------|----------------|--|
| Issues    | -              | Write Engine Analysis<br>Integrate GitHub and Jira<br>Setup Repository<br>Code Convension Document<br>Learning Engine Basics |
| Estimates | 0              | 4 + 2 + 2 + 2 + 2  |

Table 3: Overview of issue status at the end of sprint 3

**Retrospective:**

The sprint seen in perspective of the sprint alone went well. We finished all the tasks we had in mind and also managed to start learning the engine basics as well. This means that we from next week can start working towards the MVP. We discussed increasing the sprint length from one to two weeks for the next sprint, but decided to keep the one week sprint because we want to have a new retrospective meeting next week and evaluate the accuracy of the story point estimates we made on the issues regarding the MVP to figure out if we can and will finish the MVP in the allocated time. This decision was made because the time we have set to be finished with the MVP and display it to the client the 14.Th of March has been shortened down with a week because of the setbacks with the game engine analysis

**Sprint 4**

**Date:** 14.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 14.02 - 21.02

**Sprint Goal:**

- Finish the basic train mechanics
- Finish the first MVP level environment
- Finish the Basic spline railway

**Sprint result:**

We set out the sprint with very high expectations because of the delays we have had in previous sprints. All the basic train mechanics and the railway got finished as well as some of the environment and signals.

|           | To do   | In progress  | Done  |
|-----------|---|--|---|
| Issues    | Railway integration (7)<br><br>Curvature warnings (7) | Environment:<br>- Modeling (7)<br>- Texturing (3)<br>- Foliage (3)<br><br>Basic statuses (3) | Input Handling (7)<br><br>Calculate acceleration (3)<br><br>Move along Path (7)<br><br>Signals:<br><br>Add Models (3)<br><br>Railway:<br><br>- Procedural mesh (7)<br><br>Spline Formation (14) |
| Estimates | 14  | 16   | 41  |

Table 4: Overview of issue status at the end of sprint 4

**Retrospective:**

One of the group members lost some work days because of sickness. We underestimated the time it takes to model, texture and set up foliage for the environment. These tasks should each have been estimated as a 3 and is together with the sickness the reason we did not finish all tasks this sprint. We decided to re-estimate the tasks left from the previous sprint and therefore all of the environment tasks will be set as 3 and the basic statuses will also be set on 3. Basic statuses involves more complex functionality than first expected.

We had a plan to use the estimations of 1 = 3 hours, 2 = 7 hours and 3 = 14 hours, but we decided to change this. The reason we want to change the estimation is because we feel that it is not accurate enough and we should not restrict our estimation process to only three categories. We therefore decided that further estimations should be done by hours alone and not by a predetermined scale.

---

## Sprint 5

**Date:** 21.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 21.02 - 28.02

### Sprint Goal:

- Finish Main Menu in game menu and pop-up messages
- Finish basic statuses
- Finish The environment creation

### Sprint result:

All the initial UI got finished in blueprints. The environment creation is nearly finished. The basic statuses task was increasing in size because...

|           | Wishlist                | To do   | In progress  | Done  |
|-----------|-------------------------|---|--|---|
| Issues    | Curvature Warnings (10) | Speed Limit Indicator (10)<br><br>Signals:<br>- Railway Integration (4) | Basic Statuses (20)<br><br>Environment:<br>- Modeling (14)<br>- Texturing(12)<br>- Foliage(12)<br><br>Add Speedometer(8) | Drone Mode (7)<br><br>Spline Terrain Adaptation (10)<br><br>Main Menu (7)<br><br>Popup Message (3)<br><br>In Game Menu (4)<br><br>Options Menu (10) |
| Estimates | 10                      | 14  | 66   | 41  |

Table 5: Overview of issue status at the end of sprint 5

### Retrospective:

The sprint was set to contain 121 work hours, this is all the time we as a group has for a full week since every group member should have 30 hours of work each week. We only finished work hours of about 41 estimated hours. And we estimate that half of the in progress tasks are finished. This puts our completed work to 74 hours. We have underestimated both the time and complexity of several tasks.

---

We have to use this knowledge on further estimations meaning we have to estimate issues with a larger margin of error

---

## **Sprint 6**

**Date:** 28.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 28.02 - 07.03

### **Sprint Goal:**

- Finish first draft of requirement specification and status report 1.
- Finish the MVP scenario

### **Sprint result:**

The MVP got finished and put together. We had some issues with the environment creation and it's time consumption, but the estimations we made for three environment creation tasks was more accurate than our previous estimations. All the necessary functionality was there to display the MVP to the client except the basic statuses which we only controlled by a timer instead of an controller. The basic statuses only misses a few hours of work and is therefore almost finished.

|           | Wishlist                   | In progress        | Done   |
|-----------|----------------------------|--------------------|--|
| Issues    | Speed Limit Indicator (10) | Basic Statuses (5) | Environment Creation: <ul style="list-style-type: none"> <li>- Moddeling (10)</li> <li>- Texturing (10)</li> <li>- Folaige (10)</li> </ul> Add speedometer (4)<br>Requirement specification (20)<br>Status report 1 (10)<br>Signals: <ul style="list-style-type: none"> <li>- Railway integration (4)</li> </ul> Gravel Mesh Under Spline (10)<br>Bug: <ul style="list-style-type: none"> <li>- Spline hovering above ground (15)</li> </ul> |
| Estimates | 10                         | 5                  | 93   |

Table 6: Overview of issue status at the end of sprint 6

### Retrospective:

The sprint goals was achieved, but the basic statuses was only implemented to work with the MVP scenario and not finished to he extent we want in the final product. We want the signals to be controlled by a controller and be able to trigger events. New issues for this will come in the next sprint. We finished 93 work hours in the sprint and we feel that our estimations on the different tasks are beginning to get better and more accurate to how we progress in the sprints. We decided after this retrospective meeting to increase the size of the sprints from one to two weeks for the upcoming sprints. We are now going to work on two milestones simultaneously and feel that a two week sprint will give us time to produce more and that the development will be more efficient.

---

## Sprint 7

**Date:** 07.03.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 07.03 - 21.03

### Sprint Goal:

- Finish first draft of requirement specification and status report 1.
- Finish the MVP scenario

### Sprint result:

The MVP got finished and put together. We had some issues with the environment creation and it's time consumption, but the estimations we made for three environment creation tasks was more accurate than our previous estimations. All the necessary functionality was there to display the MVP to the client except the basic statuses which we only controlled by a timer instead of an controller. The basic statuses only misses a few hours of work and is therefore almost finished.

|           | Wishlist  | To do   | In Progress  | Done   |
|-----------|---|---|--|--|
| Issues    | Editor mode:<br>- Generate flat terrain (14)<br>- Manipulate terrain (20)<br>- Dynamic buttons (10) | Place and edit railway (28)<br><br>Load scenario from file (14)<br><br>Save scenario to file (14)<br>Static Buttons on screen (7) | Main Goal:<br>- Signal Controller (21)<br>- Emergency Breaks (7)<br>- Stop Scenario (21)<br>In game content browser (25)<br><br>Save Object to File (20)<br><br>Load Object From File (20) | Main Menu (14)<br><br>Basic Statuses (7)<br><br>Second iteration Requirement Specification (21)<br><br>Place 3D Models (12)<br><br>Manipulate 3D Models (25) |
| Estimates | 44  | 63  | 114  | 79   |

Table 7: Overview of issue status at the end of sprint 7

### Retrospective:

The sprint goals was achieved, but the basic statuses was only implemented to work with the MVP scenario and not finished to he extent we want in the final product. We want the signals to be controlled by a controller and be able to trigger events. New issues for this will come in the next sprint. We finished 93 work hours in the sprint and we feel that our estimations on the different tasks are beginning to get better and more accurate to how we progress in the sprints.

## Sprint 8

Date: 22.03.2022

---

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 22.03 - 04.04

**Sprint Goal:**

- Finish the editor mode, meaning the content browser and all the tools for saving, placing and the UI.
- Finish the third iteration of the requirement specification.
- Finish the Main Goal which is a working scenario with a controller for the signals.

**Sprint result:**

We finished the third iteration of the requirement document and the main goal. The editor mode is about halfway done.

|           | Wishlist   | To do  | In Progress   | Done   |
|-----------|--|--|---|--|
| Issues    | Editor mode:<br>- Save Scenario to File (14)<br>- Load Scenario from File (20) | Static Buttons on Screen (7)<br><br>Place and Edit railways (28) | Save Object to File (20)<br><br>Load Object From File (20)<br><br>Log In:<br>- Add User Types (10)<br>- Restrict functionalities (10) | Main Goal:<br>- Signal Controller (21)<br>- Emergency Breaks (7)<br>- Stop Scenario (21)<br><br>In game content browser (15)<br><br>Third Iteration Requirement Specification (10) |
| Estimates | 34   | 35   | 60  | 74   |

Table 8: Overview of issue status at the end of sprint 8

**Retrospective:**

The estimations we made was not very accurate to the workload. the content browser for example has been challenging to get working. This is because it introduced many problems that we did not account for when estimating. This issue which we thought initially would take approximately 4 days to complete has now been a work in progress for about two weeks. Because of the troubles we have had we will use the next sprint (which includes the easter) to continue working towards finishing up the editor mode.

---

---

## Sprint 9

**Date:** 04.03.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 04.04 - 13.04

### Sprint Goal:

- Finish the editor mode, meaning the content browser and all the tools for saving, placing and the UI.
- Finish the fourth iteration of the requirement specification and deliver a draft of the thesis.

### Sprint result:

We finished both the requirement specification and the requirement specification. The place and edit railway issue was started but there was some issues with the relevancy and meaning of including it in the project.

|           | In Progress                  | Done  |
|-----------|------------------------------|---|
| Issues    | Place and Edit railways (28) | Save Object to File (5)<br>Load Object From File (5)<br>Log In:<br>- Add User Types (7)<br>- Restrict functionalities (7)<br>Static buttons on screen (25)<br>Fourth iteration Requirement Specification (10)<br>Bachelor Thesis draft (20) |
| Estimates | 28                           | 79  |

Table 9: Overview of issue status at the end of sprint 9

### Retrospective:

On our meeting with the supervisor he brought up that it may be necessary to cut out the railway editing tool. Our discussion on the subject. We figured that because of the problems we have had with it and the fact that the functionality we can produce would not be better than using the original spline tool in the UE4 editor



---

we decided that Thomas will work on the issue for one more day to see find out if the issue is worth resolving or if it's smarter to cut out the functionality and focus on other parts of the project. Update, the place and edit railways issue will be excluded from the backlog because we want to focus on other parts of the assignment.

---

## **Sprint 10**

**Date:** 21.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Period:** 18.04 - 01.05

### **Sprint Goal:**

- Finish the demo test level
- Have the simulator ready for testing (all functionality ready)

### **Sprint result:**

We finished the demo test level and got the simulator ready for the student tests. The process was challenging because it was important that the quality was ensured because the simulator is going to be tested on students from Lokførerskolen next Monday.

|           | Wishlist                     | In Progress   | Done  |
|-----------|------------------------------|---|---|
| Issues    | Place and Edit Railways (28) | Write bachelor Thesis (30)<br><br>Game integration testing (30) | Basic Train cars (10)<br><br>Convert in-game menu to c++ (4)<br><br>Demo Test Level:<br>- Create Environment (4)<br>- Add train and wagon (1)<br>Add signal and triggerboxes (2)<br>- Add railway (2)<br>- Add station (1)<br>- Possesion switch between drone and train (2)<br>Fix camera possesion bug (4)<br><br>Delete editor objects (5) |
| Estimates | 28                           | 60  | 21  |

Table 10: Overview of issue status at the end of sprint 10

### Retrospective:

Even though we only finished 21 of the story point estimates, we have spent much time on writing the bachelor. All tasks we set out to do is finished except the Place and edit railway issue which got excluded from the project scope. This particular task was taking up a lot of development time due to the unforeseen difficulty of the task. After discussing internally and with the client, we concluded that all the functionality we were developing in the in-game editor mode were reflections of what offers in-engine. This made a hindrance to further development of editor functionality, which was then removed from the project, as the client agreed that it would be easier to utilize Unreal Engine itself for this functionality.

When planning the sprint we added an issue for game integration testing. This issue would contain all the bugs, errors and code mistakes we could find when trying to build and package the game. Since the scope of the issue increases when working on it, we decided to not include as much issues to this sprint to ensure that this issue gets the attention it needs.

---

**Date:** 22.04.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:**

**Description:**

**Sprint goal:**

Create a demo level that students can test the simulator functionalities on. Demo test level

Place and edit railways: On our meeting with the supervisor he brought up that it may be necessary to cut out the railway editing tool. Our discussion on the subject. We figured that because of the problems we have had with it and the fact that the functionality we can produce would not be better than using the original spline tool in the UE4 editor we decided that Thomas will work on the issue for one more day to see find out if the issue is worth resolving or if it's smarter to cut out the functionality and focus on other parts of the project.

## 4 Daily Scrum Meetings

---

**Date:** 19.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Review last days writing

**Description:**

Went through and quality assured everyone's writing and planned the work for today. Took 3 hours to go through everything, was a bit more time consuming than we had imagined.

---

**Date:** 20.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Plan further work, and how to proceed with the engine analysis .

**Description:**

We need to find the different alternatives and maybe find some working demonstration with trains in the different engines. When finding alternatives it is important that they are modern, are VR supported and can reproduce all functionality in DiskSim. Some other features it would be great if the engine has are the ability to re-use gltf and ac3d files and that it is easy to use. John will start on the analysis today, while the three other group members will continue on the project plan.

---

**Date:** 21.01.2022

---

---

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Status for progress

**Description:**

We discussed the missing and/or unfinished parts in the project report and found it to be the responsibility and roles (small), the gantt chart (medium), product backlog (small, takes all group members), testing (small, needs both supervisor and client input) decision point (small) and overall fixup (Big). We are planning on doing the overall fixup after the client meeting to have a first draft to show the supervisor the 26.th of January.

---

**Date:** 24.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Work progress for the day

**Description:**

Continue working on the game engine analysis.

---

**Date:** 25.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Discussion on the key aspects of the game analysis.

**Description:** We discussed what to write in the analysis. We decided that we would have an introduction for each game engine. We would then have 5 different subsections where we compare the different engines abilities. The 5 subsections are: Absolute requirements, Code language, DeskSim and Learning curve.

---

**Date:** 26.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Continue analyzing engines

**Description:**

We have changed what engine each of us are analyzing and will work on today.

---

**Date:** 27.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Continue on the engine analysis

**Description:**

We changed the analysis rotation and started working.

---

---

---

**Date:** 28.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Go quickly through the project plan and check if it's ready. Continue on the engine research.

**Description:**

Fixed up the last TODO's and correction's of the project plan, and started on the last engine each had to analyse.

---

**Date:** 30.01.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Plan today's progress.

**Description:**

Started to write the analysis of game engine. Each got their own parts.

---

**Date:** 03.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** What are the remaining parts of the game engine analysis?

**Description:**

Write the introduction for the analysis. [x] Double check and add all graphics api's used in the different game engines. [] Review open3D engine. [x] Review C++ in code language. [x] Finish CryEngine and Godot in visual scripting and review the two others. [] See what that can be removed from Code languages and Scripting: similarities. [x] Rewrite learning curve introduction. [x] Write about documentation, use guidelines. Review community Support. [] Final Walkthrough []

---

**Date:** 04.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Agree upon a name for the product

**Description:**

We came up with some name suggestions for the solution: DeskSim v2, Simulator-utvikling for lokførerskolen, DeskSim v2 Demo, Trainsimulator Demo, DeskSim, Simulator Development for Lokførerskolen. Sent a mail to the supervisor to get some feedback on our suggestions.

---

---

**Date:** 08.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** What work each person should focus on? And a status update on the group.

**Description:**

Should finish the game engine analysis

---

**Date:** 08.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Find out what standards to use in our code conversion document and finish the document.

**Description:**

Everyone was pleased with the progress yesterday and is ready to start working on the code conversion document and start the implementation process.

---

**Date:** 11.02.2022

**Present:** Henrik, John Ole and Thomas

**Agenda:**

**Description:**

We decided to continue working on learning the engine.

---

**Date:** 17.02.2022

**Present:** Henrik, John Ole and Thomas

**Agenda:** Demonstration of each persons progress so far

**Description:**

Everyone showed their progression throughout the week.

---

**Date:** 18.02.2022

**Present:** Henrik, John Ole and Thomas

**Agenda:** Demonstration of each persons progress so far

**Description:**

Everyone showed their progression throughout the week.

---

**Date:** 22.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

---

---

**Agenda:** Each person are showing off their progress from the week.

**Description:**

It turned out one of the group members went outside the project scope for their task. This was due to misunderstanding and it got sorted out what should be the objective for the person.

---

**Date:** 23.02.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Each person are showing off their progress from the week.

**Description:**

Thomas showed the progress in the menu development. Had implemented some basic screen scaling. John Ole had progressed in creating the environment. Henrik and Endre has been working on importing their assets correctly and therefore has little to show.

---

**Date:** 07.03.2022

**Present:** Endre, Henrik and Thomas

**Agenda:** Plan the meeting with lokførerskolen

**Description:**

If not commented about: What in the MVP would you like us to change or further develop? We should ask what they want us to develop now, regarding the MVP. Ask how the subgoals will be implemented, as in-game functionality, or in-editor features. evt: If we get sort of free reigns, aks their thoughts of implementing hight maps for generating environment.

---

**Date:** 15.03.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Everyone will show the sprint progress and plan the work for the remaining time.

**Description:**

Some of us showed the result of work with a content browser that 3D objects could be dragged out from and a tool for moving and rotating them. One showed the plan and current process regarding the signaling controller.

---

**Date:** - 18.03.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Discuss the client meeting on Monday and what we should address.

---

---

**Description:**

What we want to address: The terrain editing tool task will be a large task and doesn't seem feasible to include in the bachelor. we feel that we want to show more knowledge in other subjects. Our suggestions for the other functionality we can make: - Log in system (Communication with your API's); - Teacher-student and student-student multiplayer. (Samkjøring) - Logging of relevant data for simulator. (For example quizzes) - More train functionality, multiple tracks and wagons?

---

**Date:** 04.04.2022

**Present:** Endre, Henrik, John Ole and Thomas

**Agenda:** Plan the writing process for the upcoming sprints. And plan how to do scenarios.

**Description:**

Divide the writing tasks for the first iteration of the graduate project. - Start writing 3a Engine choice (John Ole), 3b Requirements (Henrik) , 3c Design (Thomas), Development Process (Endre), Implementation (Everyone), Deployment (Endre or later), Testing (Henrik og Thomas).

Scenarios: What to include in the files: Depends on what will be in the scenario. Want to have an editor which don't require us to use separate configuration files for scenarios, but have them stored in the map instead. We had a discussion on this, and our understanding is that it would be less complex, easier and more efficient to utilize the functionality provided by unreal to solve this.



## I Work Logs

PROG2900 - BACHELOR THESIS

---

# Work Logs

DeskSim v2

---

*Authors:*

Thomas Arinesalingam

John Ole Bjerke

Endre Heksum

Henrik Markengbakken Karlsen

May, 2022

---

# 1 Work Log

---

**Date:** Monday 17.01.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Met with the client and started setting up the project plan, GitHub and Jira | 04:27           |
| Henrik      | Met with the client and started setting up the project plan, GitHub and Jira | 04:27           |
| John Ole    | Met with the client and started setting up the project plan                  | 01:00           |
| Thomas      | Met with the client and started setting up the project plan, GitHub and Jira | 04:27           |

---

**Date:** Tuesday 18.01.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Wrote about the scope of the project in the project plan   | 05:35           |
| Henrik      | Wrote about the work methodology, and finished risk identification and started writing risk analysis in the project plan | 06:45           |
| John Ole    | Wrote about Goals and Constraints in the project plan  | 06:15           |
| Thomas      | Wrote about quality assurance  | 06:56           |

---

**Date:** Wednesday 19.01.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Worked some more on project scope, went over the project plan with the group. Meeting with our supervisor  | 06:55           |
| Henrik      | Watched videos, and learned about Jira and wrote the section about it in the project plan (not done). Removed unnecessary details in risk assessment and overall fixes | 06:50           |
| John Ole    | Went over the project plan and had meeting with Tom. Also finished Effect Goals under Goals and Constraints  | 06:30           |
| Thomas      | Finalized the milestones and overall fixes   | 7:12            |

---

**Date:** Thursday 20.01.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Finished writing scope section of project plan, reviewed other parts of the plan  | 05:32           |
| Henrik      | Finished the writing about risk assessment and work methodology, and reviewed other peoples work  | 06:15           |
| John Ole    | Started working on research for the game engine analysis, Started writing on it, as well as going over changes made to the project plan | 06:17           |
| Thomas      | Worked on formatting, tables and figures for the project plan   | 6:49            |

---

**Date:** Friday 21.01.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Reviewed project plan with group, started researching different game engines   | 06:47           |
| Henrik      | Finished the project plan and started on the game engine analyse. And migrated all minutes of meeting documents and work logs to Latex | 07:25           |
| John Ole    | Finalized the project report with the group, and continued working on the game analysis  | 07:10           |
| Thomas      | Finalized the project report, its layout, content and appendices   | 07:29           |

---

**Date:** Monday 24.01.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Downloaded Unreal and started some tutorials. Had a meeting with the client. Set up a plan for analyzing the different engines        | 07:22           |
| Henrik      | Read about the godot engine and had a meeting with the client. After the meeting we planned the structure for the game engine analyze | 07:22           |
| John Ole    | Worked and did research on the game engine analysis and had meeting with Lokførerskolen   | 07:20           |
| Thomas      | Met with the client, finished the project plan, sent mails to developers about VR engine choices, research                            | 06:52           |

---

**Date:** Tuesday 25.01.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Reading about and looking at some tutorials for CryEngine, downloaded and tested basic use of CryEngine                 | 07:01           |
| Henrik      | Read about the Unity engine and noted down the things I found relevant to include in the analysis with it's references. | 07:17           |
| John Ole    | Read and did research on the Godot engine and took notes regarding inportant info                                       | 07:06           |
| Thomas      | Researched Unreal Engine 4, its pros and cons for simulation and VR   | 7:10            |

---

**Date:** Wednesday 26.01.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Researched Unreal and looked at some tutorials. Downloaded and tried the engine a bit | 06:04           |
| Henrik      | Researched Godot and wrote down the things relevant to the analysis                   | 06:50           |
| John Ole    | Researched Cry Engine and noted down important info                                   | 06:51           |
| Thomas      | Researched Unity, its cons and pros   | 6:22            |

---

**Date:** Thursday 27.01.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Researched the strengths and weaknesses of Unity, downloaded and tried a VR demo to look at vr integration | 06:27           |
| Henrik      | Analysed CryEngine, pros and cons and wrote down the things I found relevant for the analysis              | 06:14           |
| John Ole    | Researched Unreal Engine and noted down important info   | 06:50           |
| Thomas      | Analyzed Godot as a game engine for simulation and VR  | 6:16            |

---

---

---

**Date:** Friday 28.01.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Researched Godot, both the current version (3.4) and a bit about 4.0. Downloaded the engine and looked at a VR demo. | 06:06           |
| Henrik      | Analyzed Unreal Engine with the requirements in mind   | 06:05           |
| John Ole    | Researched Unity and noted down important info   | 05:36           |
| Thomas      | Analyzed CryEngine, and took notes   | 06:07           |

---

**Date:** Monday 31.01.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Started writing the engine analysis, and comparing game engines with the group.        | 06:22           |
| Henrik      | Wrote introduction about Godot and contributed many places in the game engine analysis | 06:36           |
| John Ole    | Started writing the analysis and wrote about Cryengine                                 | 06:31           |
| Thomas      | Compared notes and summarized our findings into paragraphs for the analysis document   | 07:06           |

---

**Date:** Tuesday 01.02.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Did some research, testing and writing about both Unreal Blueprints and Unity Visual Scripting | 06:24           |
| Henrik      | Wrote about GDScript and other parts in the Game Engine analyzes                               | 06:36           |
| John Ole    | Wrote on the analysis about C++ as well as started with community support                      | 06:58           |
| Thomas      | Wrote about C# and it's use in game engines, and the learning curve of the engines             | 07:03           |

---

**Date:** Wednesday 02.02.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Finished writing about visual scripting in Unity, almost done with research and writing about Cryengine                 | 06:22           |
| Henrik      | Wrote about additional development tools and contributed to quality checking others work. Small cleanups in some parts. | 05:58           |
| John Ole    | Continued with writing about community support in the analysis as well as some small changes to other parts             | 05:52           |
| Thomas      | Wrote about scripting, development tools and learning curve   | 6:56            |

---

**Date:** Friday 04.02.2022



---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Finished visual scripting, reviewed parts of engine analysis with group.   | 06:54           |
| Henrik      |  |                 |
| John Ole    | Started on conclusion, went over and rewrote some section in the analysis with group and finished the introduction | 06:39           |
| Thomas      | Wrote about existing solutions and the conversations with developers, and started on the conclusion                | 06:59           |

---

**Date:** Monday 07.02.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Sick  | 00:00           |
| Henrik      | Had a meeting at Lokførerskolens location where we got a walk through of the simulator and got the joysticks. | 06:00           |
| John Ole    | Went to the client for a demonstration of their current simulator   | 06:00           |
| Thomas      | Went to the client for a demonstration of their current simulator   | 06:00           |

---

**Date:** Tuesday 08.02.2022

| <b>Name</b> | <b>Description</b>                                      | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Sick  | 00:00           |
| Henrik      | Wrote the finalized version of the game engine analysis | 06:47           |
| John Ole    | Went over the analysis and finished it                  | 04:26           |
| Thomas      | Finalized the game engine analysis                      | 06:33           |

---

---

**Date:** Wednesday 09.02.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Sick  | 00:00           |
| Henrik      | Wrote the naming convention on the code convention document. Finished integrating Jira with GitHub and started on creating the template for the Unreal Engine project in GitLab | 05:43           |
| John Ole    | Started and finished the code convention document as well as wrote some issues for MVP  | 05:35           |
| Thomas      | Finished the code convention document   | 05:44           |

---

**Date:** Thursday 10.02.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Sick  | 00:00           |
| Henrik      | Had a meeting with the Supervisor, set up the Unreal Engine project in GitLab and explored with the engine by trying to implement some input detection and movement from the USB controller | 05:31           |
| John Ole    | Started with Supervisor meeting, then started with setting up the game engine and trying it out   | 05:32           |
| Thomas      | Supervisor meeting + explored Unreal Engine, trying to implement simple train movement  | 05:32           |

---

**Date:** Friday 11.02.2022

| <b>Name</b> | <b>Description</b>                                   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Sick   | 00:00           |
| Henrik      | Researched unreal engine and learned the basics      | 05:15           |
| John Ole    | Researched and tested Unreal Engine and how it works | 06:34           |
| Thomas      | Researched Unreal Engine basics                      | 2:13            |

---

---

---

**Date:** Monday 14.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Started catching up on group work, learning about unreal                       | 03:33           |
| Henrik      | Implemented the basic input mappings and movement                              |                 |
| John Ole    | Setup Git with Unreal and started working on environment creation              | 06:52           |
| Thomas      | Finished setting up Unreal with Git and started coding a spline mesh component | 06:50           |

---

**Date:** Tuesday 15.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Fixed issues related to unreal project and git, started some work on signals                           | 04:13           |
| Henrik      | Implemented acceleration and breaks for the train.   | 06:34           |
| John Ole    | Fixed some issues with Git and unreal, as well as implemented some terrain based on heightmaps         | 06:05           |
| Thomas      | Fixed technical issues with Git and Unreal, and finalized the procedural mesh part of the railway tool | 06:36           |

---

**Date:** Wednesday 16.02.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       |   |                 |
| Henrik      | Started on implementing train movement along spline                           | 04:14           |
| John Ole    |   |                 |
| Thomas      | Worked on the spline generator, so it snaps the mesh to the terrain/heightmap | 03:21           |

---

**Date:** Thursday 17.02.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Imported basic train signal model, learned about materials and emissive light   | 03:03           |
| Henrik      | Cleaned up train movement and started importing a new plugin for input handling | 03:00           |
| John Ole    | Researched how to create your own texture                                       | 03:00           |
| Thomas      | Worked on the tangent to improve the curving of the railway spline              | 03:11           |

---

**Date:** Friday 18.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Added working emissive light in code for train signal  | 08:19           |
| Henrik      | Finished the input handling for now and closed the train movement story. Merged train movement and railway creation to form the first milestone branch | 7:09            |
| John Ole    | Created a our own texture and added a blend between grass and dead-grass texture   | 07:09           |
| Thomas      | Added rotation to the railway spline around its forward axis   | 05:48           |

---

---

**Date:** Monday 21.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Finished signal timer to switch signal states, did some research for central signal controller, about events and delegates   | 06:27           |
| Henrik      | Had a meeting with the client. Merged and set up the first milestone branch and started on the speedometer. I did only set up the class and research the approach for making it. | 05:46           |
| John Ole    | Had some trouble with git lfs with reaching the max bandwidth, so I had to revert the changes and reduce the sizes of the files to max 100MB for now.                            | 04:30           |
| Thomas      | Had a meeting with the cliend and worked on detecting angle and height change in the railway   | 06:56           |

---

**Date:** Tuesday 22.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Did some more research about events and actors. Started work on signal system with different signal types. Tried helping with 3d model problems  | 06:44           |
| Henrik      | Tried to fix problems with the imported train models. Created a new issue regarding this. Worked on the speedometer after that.  | 06:48           |
| John Ole    | Added macro variations of the dry grass and cliff textures, and landscape materials now depend on the slope and sharpness of an angle as to create rocky hills with grass on top(greentop) | 06:57           |
| Thomas      | Implemented a main menu level, with buttons to start the game and a settings menu, which is able to change the resolution and window mode  | 06:53           |

---

**Date:** Wednesday 23.02.2022

---

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Worked on base signal and light functionality   | 04:46           |
| Henrik      | Worked a bit on trying to fix the train model files.<br>Started on the drone class because the model problem got very frustrating | 05:35           |
| John Ole    | Created a custom specular channel for the landscape material  | 05:21           |
| Thomas      | Worked on the main menu and settings menu   | 05:55           |

---

**Date:** Thursday 24.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Reworked basic signal to allow for better customization  | 06:37           |
| Henrik      | Finished movement, camera, and possession changes between the train and the drone. Small bug in drone movement needs to be fixed | 07:15           |
| John Ole    | Created a new level with mountains and ocean   | 06:24           |
| Thomas      | Added a popup blueprint for warnings and messages, added a scenario select to the main menu, and added a pause menu              | 06:26           |

---

**Date:** Friday 25.02.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Sick   | 00:00           |
| Henrik      | Fixed a movement issue with the drone and started on the DMI HUD for the train. Got a adequate design prototype ready and next step is to add functionality to the speedometer | 05:48           |
| John Ole    | Continued modelling the level and fixed the landscape texture to work properly   | 05:13           |
| Thomas      | Worked on the main menu, adding scenarios and tabs for different types of scenarios  | 06:25           |

---

---

---

**Date:** Tuesday 01.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Started work on central controller for signals, worked on status rapport                         | 06:22           |
| Henrik      | Sick   | 00:00           |
| John Ole    | Worked on status report 1 as well as did some minor changes to the textures                      | 04:08           |
| Thomas      | Implemented signals snapping to the nearest railway, and wrote towards finishing status report 1 | 06:18           |

---

**Date:** Thursday 03.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Finished setting up the different statuses for different lights, researched delegates/events, interfaces, actor communication | 05:02           |
| Henrik      | Finished the train DMi and wrote a bit in the requirements specification.   | 3:54            |
| John Ole    | Worked on requirement specification   | 04:10           |
| Thomas      | Formatted status report 1 and created the requirements specification document   | 06:32           |

---

**Date:** Friday 04.03.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Finished writing status rapport and requirements specification, helped with technical issues, made lights cycle statuses for MVP           | 07:47           |
| Henrik      | Finished status report 1 and requirement specification   | 07:03           |
| John Ole    | worked on Status rapport and required specification, as well as finished MVP where i extended the level and added more foliage and terrain | 07:22           |
| Thomas      | Finished status report 1, the requirements specification, and collectively finalized the MVP to a buildable executable                     | 07:58           |

---

**Date:** Monday 07.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b>        |        |
|-------------|--|------------------------|--------|
| Endre       | Client meeting about MVP, scrum planning meeting and planning work ahead   | 05:50                  |        |
| Henrik      | Had a client meeting and had scrum planning meeting where we created new issues and discussed the further development. Wrote about the implementation of train, drone and dmi. | 07:00                  |        |
| John Ole    | Client meeting   | scrum planning meeting | 05:50: |
| Thomas      | Client meeting, scrum planning for the next milestone, and research for future functionality   | 06:05                  |        |

---

**Date:** Tuesday 08.03.2022



---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Started work on central controller, added box trigger to change signal status                          | 06:41           |
| Henrik      | Finished the Main Menu widget and started working on the Static on-screen buttons for the editor mode. | 05:00           |
| John Ole    | Started working and researching how to implement procedural generated terrain                          | 05:52           |
| Thomas      | Created the camera movement and gizmos for translation of objects in the in-game editor mode           | 06:32           |

---

**Date:** Friday 11.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Learned more about ticking and fixed ticking bug with TrainTriggerBox, reworked and improved some logic in signal controller, added triggers to MVP level | 06:49           |
| Henrik      | Worked on the in game content browser, currently implementing asset selection and dragging  | 06:53           |
| John Ole    | Started researching how to save and load levels   | 06:31           |
| Thomas      | Started converting the editor blueprints into C++, achieving movement and gizmo component structuring for the translation and rotation tools              | 06:57           |

---

**Date:** Monday 14.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Added emergency stop to train and signal controller, needs some bugfixing                        | 06:48           |
| Henrik      | Finished the first iteration of the content browser and started on terrain generation            | 07:16           |
| John Ole    | Research and taking care of personal things  | 06:51           |
| Thomas      | Worked on converting the editor controller into C++, adding gizmo meshes and tool mode switching | 06:57           |

---

---

---

**Date:** Tuesday 15.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Testing train emergency stop, worked on requirements specification with group                       | 03:05           |
| Henrik      | Wrote the second iteration of the requirement specifications and started on the terrain generation. | 05:00           |
| John Ole    | Worked on Savelevel function  | 04:19           |
| Thomas      | Iterated on the requirements specification and worked on the editor controller code                 | 03:12           |

---

**Date:** Wednesday 16.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Train improvements and bugfixes, reimplemented train model properly in fbx format, turned the train into a blueprint so others can easily be added later  | 06:27           |
| Henrik      | Worked on creating a flat square from code when generating a new scenario in editor mode. You can now create a 100x100 to 10000x10000 flat area. Next task is to connect it to the other editor functionality | 06:03           |
| John Ole    | Worked on save and load level, and now saves train position to file for testing purposes  | 07:14           |
| Thomas      | Worked on the gizmos for the editor controller  | 07:14           |

---

**Date:** Thursday 17.03.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Restructured some c++ code to improve organization, split functionality of traintriggerbox into two new classes for better usage | 06:33           |
| Henrik      | Was in a meeting with the supervisor and is currently moving the content browser functionality from blueprints to c++            | 06:40           |
| John Ole    | Supervisor meeting as well as worked on Moving save and load functionalities from the train class                                | 06:50           |
| Thomas      | Implemented line casting for interaction with editor objects and gizmos in c++, supervisor meeting                               | 06:58           |

---

**Date:** Friday 18.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Almost finished working on split trigger box functionality and central controller emergency stop                      | 06:18           |
| Henrik      | Have read about game control flow for the game mode and started to implement the content browser in c++.              | 06:11           |
| John Ole    | Started working on a new SaveManager which handles all save and load functionalities                                  | 06:10           |
| Thomas      | Finished the editor controller with movement and gizmos for transforming objects, started working on the spline gizmo | 05:51           |

---

**Date:** Monday 21.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Meeting with client about progression of project, talked about which tasks to focus on | 01:32           |
| Henrik      | Client meeting and small meeting afterwards  | 1:25            |
| John Ole    | Client meeting   | 01:30           |
| Thomas      | Client meeting and project planning  | 1:30            |

---

---

---

**Date:** Tuesday 22.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Added new issues and started new sprint with group, finished emergency trigger box, started research on networking | 05:21           |
| Henrik      | Worked on converting the content browser and object placement tool to c++  | 06:30           |
| John Ole    | Continued working on SaveManager   | 06:14           |
| Thomas      | Porting spline behaviour into in-game editor, working on collision for mosue interaction in C++                    | 06:34           |

---

**Date:** Wednesday 23.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Added doxygen comments to signal related code files, did more research on multiplayer and online identity system to handle logins | 06:44           |
| Henrik      | All of the content browser functionality is set up except the dragging of object to the scene                                     | 06:50           |
| John Ole    | Mostly research into how to best store objects and its values within SaveManager  | 06:49           |
| Thomas      | Researched and read documentation for collisions  | 2:26            |

---

**Date:** Thursday 24.03.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Spent most of the day on lecture and meeting with supervisor, spent some time on requirements specification | 00:46           |
| Henrik      | Iterated over the requirement specification   | 01:13           |
| John Ole    | Supervisor meeting and worked on requirement specification  | 01:00           |
| Thomas      | Had a lecture and supervisor meeting before quickly iterating on the requirements specification             | 01:14           |

---

**Date:** Friday 25.03.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Fixed weird bug with doxygen file comments, continued researching various online and multiplayer systems in unreal                   | 06:46           |
| Henrik      | Continued on the browser content, but encountered issues with the drag and drop functionality, had to make additional files for this | 06:40           |
| John Ole    | Implemented ways of storing Actor data into save-manager   | 05:58           |
| Thomas      | Fixed the bug where the railway did not register collisions, and started implementing in-game manipulation of spline points          | 05:45           |

---

**Date:** Monday 28.03.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Merged MVP branch into main, helped team with some programming bugs and issues, found plugin for REST calls from unreal, started learning about UIs and menus to create login screen | 06:50           |
| Henrik      | Worked on the raycast from the viewport to the ground, encountered several problems with this  | 07:00           |
| John Ole    | Worked on loading data from the save files and updating their respective object/Actor  | 03:16           |
| Thomas      | Made the railway interactable in the editor mode, started working on manipulating spline points  | 06:37           |

---

**Date:** Tuesday 29.03.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Did some research about online authentication, worked on requirements specification and status rapport 2                      | 06:19           |
| Henrik      | Finished raycasting and placing objects, plus cleaned up the code   |                 |
| John Ole    | Added a SaveableActor class which is inherited by Actors that is being saved, and is done to filter out which actors is saved | 05:37           |
| Thomas      | Worked on implementing exposed spline points for railway manipulation   | 06:02           |

---

**Date:** Tuesday 31.03.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Mostly finished with login ui, started testing VaREST plugin and how to use it properly, had meeting with supervisor                 | 04:29           |
| Henrik      | Merged functionality from editor mode, savestate and content browser together. Bug fixed, and made it work together                  | 07:30           |
| John Ole    | Supervisor meeting and Worked on merging save and load level with model placing tools and continued working on Spawning saved Actors | 06:57           |
| Thomas      | Supervisor meeting, had to research about collisions in Unreal and worked on gizmo handling  | 06:43           |

---

**Date:** Friday 01.04.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Continued work on auth api in unreal, working on gameinstance to store data, encountered and tried to fix unreal engine crash               | 06:54           |
| Henrik      | Continued working on integrating the functionalities. Got done with the editor mode, and savestate needed more work from own branch         | 07:47           |
| John Ole    | Loadgame now spawns actors from savegame into the scene and fixed an issue where the game would crash when saving after running for a while | 06:55           |
| Thomas      | Reworked the system for handling multiple gizmos along the railway  | 06:22           |

---

**Date:** Monday 04.04.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Spent a lot of time trying to fix error with unreal engine and corrupted blueprint which stopped progress, tried downloading and compiling the engine from source, eventually found a backup to corrupted file. Finished prototyping auth, can log in to Lokfører servers | 06:18           |
| Henrik      | Integrated both editor mode and savegame into the milestone branch and bug fixed  | 06:40           |
| John Ole    | Can now save and load variables from Actors using SaveGame tag in UPROPERTY(SaveGame)   | 02:28           |
| Thomas      | Continued working on interactable points along the railway, spawning a selectable mesh at each point to place the gizmo tools   | 06:29           |

---

**Date:** Tuesday 05.04.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Had a short scrum meeting before working on other subjects | 00:31           |
| Henrik      | Cleaned up and commentated the HUD                         | 01:32           |
| John Ole    | Short Meeting before working on other subjects             | 00:47           |
| Thomas      | Worked on another course                                   | 0:00            |

---

**Date:** Wednesday 06.04.2022



---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Started writing about the development process on the thesis paper, downloaded and looked around Unreal Engine 5 for a bit | 06:13           |
| Henrik      | Started writing the fourth iteration of the requirement specification and continued on the HUD                            | 07:03           |
| John Ole    | Started working on the Choice of engine section in the bachelor thesis  | 06:00           |
| Thomas      | Wrote about technical design and architecture, creating ER-diagrams, and helped debugging some code for the HUD           | 07:04           |

---

**Date:** Thursday 07.04.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Converted most auth functionality to c++, should work but not yet tested  | 06:37           |
| Henrik      | Converted main menu to c++ and started to integrate all functionality and create a basic game flow. Had some problems with changing the pawn/controller that controls the game. Got it sort of working, but contains small bugs | 07:18           |
| John Ole    | Worked on saving the MapName and to load the correct save relative to the map opened. Had a couple of issues where it sometimes would crash on load but it would only happen in rare occasions.                                 | 06:10           |
| Thomas      | Worked on the editor tools, converting functionality from blueprint to c++, and worked on system architecture and entity relationship in the thesis   | 07:23           |

---

**Date:** Monday 11.04.2022

---

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Gameinstance now stores data received from login, can be used across the game to check info about the logged in user, added comments and some polish       | 05:56           |
| Henrik      | Converted the train DMI to c++, added a gamemode base for the editor and in game playing mode for the simulator and is currently working on the game flow. | 06:05           |
| John Ole    | Worked on the bachelor thesis and fixed merge issues   | 05:36           |
| Thomas      | Worked on the gizmo tool for working with both spline points and actors, spawning gizmo meshes for each spline point to manipulate the spline itself       | 05:56           |

---

**Date:** Tuesday 12.04.2022

| <b>Name</b> | <b>Description</b>                                       | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Continued writing on the thesis paper                    | 05:49           |
| Henrik      | Sick   |                 |
| John Ole    | Worked on bachelor thesis                                | 06:16           |
| Thomas      | Worked on the editor mode and gizmo tools for the spline | 06:26           |

---

**Date:** Wednesday 13.04.2022

| <b>Name</b> | <b>Description</b>                      | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Continued writing on the thesis paper   | 03:12           |
| Henrik      |   |                 |
| John Ole    |   |                 |
| Thomas      | Worked on the first draft of the thesis | 02:10           |

---

**Date:** Tuesday 19.04.2022

---

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Meeting with client   | 01:37           |
| Henrik      | Client meeting and wrote logs   | 03:00           |
| John Ole    | Had Client Meeting and fixed a bug with loading a saved train in editor mode. Rest of the day spent working on other subjects | 02:20           |
| Thomas      | Client meeting and small fixes  | 02:14           |

---

**Date:** Friday 22.04.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Started merging functionality between login and editor branches, usertype from login can now restrict access to editor functionality | 06:00           |
| Henrik      | Started writing about the methodology in the bachelor report   | 07:00           |
| John Ole    | Started creating a new demo landscape which is longer than MVP <i>landscape</i>  | 07:50           |
| Thomas      | Wrote on the thesis  | 05:52           |

---

**Date:** Monday 25.04.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Writing on the thesis paper and reading other thesis papers  | 05:51           |
| Henrik      | Converted the settings UI to c++ and wrote about the sprints in the project report. Will create the ingame menu and change the creation of levels as well as continueing on the report tomorrow. 05:47 |                 |
| John Ole    | Finished landscaping the new demo level  | 05:55           |
| Thomas      | Worked on writing testing documentation and adding functionality for deleting objects  | 06:02           |

---

---

**Date:** Tuesday 26.04.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Writing on the thesis paper, assisted with creating the demo level for testing   | 06:18           |
| Henrik      | Converted the in game pause menu to c++ and set up the train and railway for the user testing scenario. We made a collective decision to drop the create new scenario use case. Reasoning for this will be stated in the report. | 06:15           |
| John Ole    | Started adding foliage to the demo level   | 06:21           |
| Thomas      | Implemented deletion of objects in the editor mode, debugged spline bugs   | 06:35           |

---

**Date:** Wednesday 27.04.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Added comments and formatting to login code files, reading other papers  | 06:10           |
| Henrik      | Packaged the game, and fixed errors, missing camera placement for the train on packaging and several other bugs. | 08:03           |
| John Ole    | Worked on Foliage, packaging the game, fixing errors and landscaping   | 07:12           |
| Thomas      | Worked on debugging the spline a bit, then wrote about implementation in the thesis                              | 06:42           |

---

**Date:** Thursday 28.04.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Worked on debugging camera problems with packaged builds                        | 05:11           |
| Henrik      |   |                 |
| John Ole    | Wrote about the analysis and helped debugging                                   | 06:19           |
| Thomas      | Wrote about splines, the railway tool and implementation, helped with debugging | 06:30           |

---

**Date:** Friday 29.04.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Fixed problem with train camera in packaged builds, system testing and several bugfixes, working on spline mesh performance | 06:02           |
| Henrik      | Bug fixing editor mode and wrote about user testing   | 06:01           |
| John Ole    | Worked on optimizing the game and finished the game engine analysis part  | 04:43           |
| Thomas      | Wrote on the bachelor thesis, read through some previous work and other theses, added more images and code                  | 06:33           |

---

**Date:** Monday 02.05.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Reintegrated and fixed joystick input to work with keyboard, merged login branch and started bugtesting | 06:30           |
| Henrik      | working on fixing the editor mode bug and wrote about testing and development process in the bachelor.. | 06:04           |
| John Ole    | Worked on the introduction in the thesis  | 07:01           |
| Thomas      | Visited the client and performed user tests, wrote about testing and formatted test results             | 06:55           |

---

---

---

**Date:** Tuesday 03.05.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Did some research into splines, instanced splines and optimisation, started working on bugfixing | 06:09           |
| Henrik      | Bug fixing and testing the system  | 06:02           |
| John Ole    | Worked on the introduction   | 06:20           |
| Thomas      | Debugged the latex document errors and wrote more about user tests                               | 06:26           |

---

**Date:** Wednesday 04.05.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Countinued debugging, started writing about signal controller and authentication | 06:05           |
| Henrik      | Bug fixing and system testing  | 06:00           |
| John Ole    | Worked on the introduction   | 06:09           |
| Thomas      | Debugged editorcontroller and wrote more on testing                              | 06:33           |

---

**Date:** Thursday 05.05.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Wrote about authentication design, started writing about signal controller. Reviewed progress on debugging the upcoming usertests | 06:17           |
| Henrik      | Bug fixing, system testing and writing about development process  | 06:30           |
| John Ole    | Started working on the discussion part, with why we chose to use git  | 06:33           |
| Thomas      | Finished writing about user testing and wrote about the editor and gizmos in the thesis   | 06:56           |

---

---

---

**Date:** Friday 06.05.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Wrote about central signal controller and packaging.<br>Debugged latest changes on test build                   | 06:12           |
| Henrik      | Wrote about system testing  | 03:30           |
| John Ole    | Wrote about other substitutes for git we considered<br>in the discussion  | 06:51           |
| Thomas      | Read through implementation and system<br>architecture in other theses, and wrote about gizmo<br>implementation | 04:09           |

---

**Date:** Monday 09.05.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Had meeting with client where they tested the<br>software, wrote about implementation on the thesis<br>paper | 06:41           |
| Henrik      | Client meeting and wrote about the system testing  | 03:20           |
| John Ole    | Client meeting and worked on the thesis, discussion<br>part  | 06:33           |
| Thomas      | Client meeting, and wrote on the thesis  | 04:19           |

---

**Date:** Tuesday 10.05.2022

---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Writing about implementation on thesis paper                            | 06:23           |
| Henrik      | Wrote about the development process and system testing                  | 06:20           |
| John Ole    | Merged foliage changes into LK-19 and continued on discussion on thesis | 06:54           |
| Thomas      | Worked on figures and models for the thesis paper, wrote some           | 06:36           |

---

**Date:** Wednesday 11.05.2022

| <b>Name</b> | <b>Description</b>   | <b>Duration</b> |
|-------------|--|-----------------|
| Endre       | Wrote about implementation of signals, had meeting with supervisor   | 06:57           |
| Henrik      | Changed latex document and corrected errors in the document. Rewrote some requirement specification and had a client meeting | 07:05           |
| John Ole    | Supervisor meeting, and worked on the thesis   | 05:56           |
| Thomas      | Supervisor meeting, wrote on the thesis and worked on diagrams   | 05:58           |

---

**Date:** Thursday 12.05.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Added missing pictures to implementation, started writing about login | 06:05           |
| Henrik      |   |                 |
| John Ole    |   |                 |
| Thomas      | Worked on another course  |                 |

---

**Date:** Monday 16.05.2022

---



---

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Finished writing about implementation of login, polished some images and code format styling, got an overview of what work remains with group | 07:01           |
| Henrik      | Wrote about process, HUD implementation and time usage  | 07:30           |
| John Ole    |   |                 |
| Thomas      | Wrote about code documentation, abstract and worked on formatting for images, tables, code snippets and overall structure                     | 07:09           |

---

**Date:** Wednesday 18.05.2022

| <b>Name</b> | <b>Description</b>  | <b>Duration</b> |
|-------------|---|-----------------|
| Endre       | Finishing writing the thesis paper                              | 11:04           |
| Henrik      | Writing bachelor thesis   | 11:12           |
| John Ole    |   |                 |
| Thomas      | Went through the whole thesis and wrote on the missing elements | 11:46           |

