Bachelor's thesis

Hivemind

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Kongsberg

Course: TS3000 Bacheloroppgave
Date: May 22, 2023
Title: Hivemind

This report forms part of the basis for assessment in the subject.

Project group: 1
Group members: Aurora Moholth,
Hilde Marie Moholth,
Harald Moholth,
Nils Herman Lien Håre,
Ruben Sørensen

Internal Supervisor: Dag Samuelsen
External supervisors: Jan Dyre Bjerknes,
Anete Vagale

Project partners: Kongsberg Defence & Aerospace
Semcon Norge

Acknowledgements

We would like to express our heartfelt appreciation to Jan Dyre Bjerknes and Anete Vagale, our external project supervisors, for their invaluable guidance and unwavering support throughout this endeavor. Their expertise, insights, and feedback have played a crucial role in shaping our project and pushing us towards excellence.

We are also deeply grateful to our internal supervisor from the university, Dag A. H. Samuelsen, for his dedicated mentorship, timely advice, and assistance in ensuring our project remained on track. His commitment to our progress and his insightful feedback have been instrumental in our success.

Hivemind would also like to acknowledge Joakim Bjørk, who rose to the challenge of evaluating our project and has done so in the best possible way.

Furthermore, we extend our sincere thanks to the University of South-Eastern Norway. Their support and the conducive learning environment they fostered have made this entire experience joyful and immensely enriching.

Abstract

This project deals with the problem of developing software for planning and controlling a drone swarm light show. The proposal in the project consists of the proposed software architecture that can address this problem, and the concrete implementation this software architecture. The novelty of this project lies in the completely original software architecture for planning light shows with drone swarms, and the solution proposed wherein all components are modular. This modularity means that components such as the specific route planning algorithm chosen can be easily exchanged, while new components can be added without changing the underlying algorithm. Because the proposed architecture is flexible and scalable, it is easy to adapt the proposed software for other purposes, such as researching and comparing different route planning algorithms. Other applications of the software are numerous, including general purpose route planning for agriculture, cinematography and surveying.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 28

API Application Programming Interface. 51, 55, 58–60, 72

CD Continuous delivery/deployment. 40

CI Continuous integration. 40

DOM Document Object Model. 72

GDAL Geospatial Data Abstraction Library. 51, 52, 59, 60

GIS Geographical Information Science. 55

GML Graphical Markup Language. 60, 94

GUI Graphical User Interface. 7, 25, 38, 42, 54–56, 62, 63, 74, 76, 77, 79, 80, 82–85, 88, 94, 95

HTML HyperText Markup Language. 40, 55, 72

HTTP HyperText Transfer Protocol. 7, 58, 59, 62, 64

IDEF0 Integration Definition for Process Modelling. 49, 50

IPC Inter-Process Communication. 54

JSON JavaScript Object Notation. 12, 69–72, 74

KDA Kongsberg Defence & Aerospace. 22, 23, 26, 32, 40

MAMSL Meters above mean sea level. 7, 68

MATLAB MATrix LABoratory. 28

MVP Minimum Viable Product. 24, 25, 36, 37, 49, 61, 86

NASAMS Norwegian Advanced Sufrace-to-Air Missile System. 22

PCL Point Cloud Libraries. 56

PHP Hypertext Preprocessor. 34

ROS Robot Operating System. 39, 56, 72, 91

Rviz ROS Visualization. 56

SAE Society of Automotive Engineers. 17

SQL Structured Query Language. 34

TIFF Tag Image File Format. 51

UAV Unmanned Aerial Vehicle. 16, 17, 19, 20, 22, 25, 27–29, 44, 56, 96

UI User Interface. 46

UML Unified Modeling Language. 37, 38

URL Uniform Resource Locator. 55, 64

USN University of South-Eastern Norway. 22, 34, 212

UTM Universal Transverse Mercator. 51–54, 56, 57, 63, 79, 81, 95

VCS Version Control System. 40

WCS Web Coverage Service. 60, 94

WMS Web Map Service. 55, 62, 63

XML Extensible Markup Language. 25, 60, 70, 72

Glossary

agent An agent refers to an autonomous unmanned aerial vehicle (UAV), or a drone. Each drone is identified by a unique ID and can perform tasks independently or in coordination with other drones in the system . 44, 56, 64, 75–77, 81–85, 88, 94–96

cartesian coordinate A *Symmetric Coordinate system* is a system where the origin is located at the center, and the axes are symmetrically arranged around the origin. This means that both positive and negative values of the coordinates are equidistant from the origin, and the axes have equal scales in both directions. An *Asymmetric Coordinate system* has its origin located in the upper left corner, rather than at the center, and is often defined by the size of the coordinate system. This means that the positive x and y axes extend to the right and down. 7, 53, 56–58, 64, 68, 78

geographical coordinate Geographic coordinates are a reference system used to locate positions on the Earth's surface. They are expressed in terms of angles of latitude and longitude that are measured from the center of the Earth and are referenced starting at the Equator and the Prime Meridian, respectively. Latitude is measured from the Equator and longitude from the Prime Meridian. Latitude is the measurement of a location's distance from the Equator and is designated as 0ř at the Equator and 90ř at the poles. Longitude is the measurement of a location's distance from the Prime Meridian and is designated as 0ř at the Prime Meridian and 180ř on the opposite side of the Earth. The degrees of latitude and longitude can be further divided into minutes and seconds for greater precision[5]. 25, 53, 54, 56, 57, 76, 78, 79

git A distributed version control system widely used in software development, for handling versioning of files and resources. 40

inter-process communication Features provided by an operating system that enable processes to handle and control data that is shared among them. 54

JavaScript Object Notation A lightweight file format that is easy to read and understand for both humans and computers. JSON is a common format to use when you have to transmit data to for example a drone or a website. JSON consists of name and value pairs and Ordered lists. Basically every programming language can parse and generate JSON files which has made it a very popular data format for transmitting data[6]. 69

keyframe A structure in Hivemind representing an agent's position at a point in time. 24, 25, 64, 68, 74–77, 81–83, 88, 94

route A route is a path that a drone will follow in a specific scenario. The route is defined by a series of cartesian coordinates that the drone will pass through in order. The Hivemind system uses the Routemaker component to generate these routes based on the keyframes provided by the user . 24–26, 44, 51, 55, 58, 61, 64, 65, 67, 68, 74–76, 82, 83, 88, 94

scenario A scenario refers to a predefined set of keyframes that describe the movements and actions of drones in a given geographical area. The scenario includes information such as the geographic origin, size of the area, and the keyframes for each drone. 7, 23–25, 44, 61, 62, 64, 68, 71, 74–77, 82, 83, 85, 94, 95

Universal Transverse Mercator Universal Transverse Mercator (UTM), which stands for Universal Transverse Mercator, is a system of coordinates based on a family of 120 Transverse Mercator map projections. The Earth is divided into 60 zones, each 6 degrees wide in longitude, with a central meridian for each zone[7]. The Easting has a value of 500,000 meters at the central meridian of each zone, while the Northing, or Y value, is 0 meters at the equator for the northern hemisphere and 10,000,000 meters at the equator for the southern hemisphere. The numbering of zones starts at 180ř and goes eastward, with Zone 1 covering 180W to 174W, Zone 2 covering 174W to 168W, and so on. Each zone also has a central meridian, with Zone 1 having a central meridian of 177W, Zone 2 having a central meridian of 171W, and so on. Positions are expressed as Easting/Northing, with the UTM zone and hemisphere optionally specified for positions near zone junctions. 51

unmanned aerial vehicle An aircraft without an onboard pilot. 16

UTM 33N Universal Transverse Mercator 33N is the 33th zone in the UTM coordinate system. It covers an area that spans from 12 degrees to 18 degrees east, and from the equator to 84 degrees north in the Northern Hemisphere. UTM 33N is used in Norway for nationwide data, such as topographic maps, and is the default coordinate system for data provided by Geonorge because it is the UTM zone that is in the latitude center of Norway[8]. 54, 79

WGS84 The World Geodetic System 1984 (WGS84) is a global geodetic reference framework used for global positioning, navigation, and mapping. It is based on a consistent set of constants and model parameters that describe the Earth's size, shape, and gravity fields. WGS84 is the standard reference system for the Global Positioning System (GPS) and is compatible with the International Terrestrial Reference System (ITRS). Its defining parameters include the semi-major axis of the WGS 84 ellipsoid, the flattening factor of the Earth, the nominal mean angular velocity of the Earth, and the geocentric gravitational constant[9] . 53, 57

1 Introduction

1.1 Overview

HMM | *NH*

Drones are quickly establishing themselves as tools of choice for cheap and safe operations within the fields of agriculture, surveying, disaster management and warfare. This paper considers a swarm of drones in the context of entertainment - to enact a light show in formation, and develops a software for the planning of drone flight paths, taking into account local height and buildings, that can easily be expanded to include further functionality. The project has been dubbed "Hivemind", demonstrating the overarching goal of this project to create a software that can act as the "hive mind" controller of a swarm of drone "bees".
This document will first explain the domain of the project and the problem put forward for the group to solve. It will then briefly address related work and demonstrate how this project will expand the current body of research. The group's project management framework and methods will be presented, before detailing the software development process utilized in the project.

The technical section will explain the conceptual software model, including software components, their interfaces, derived use cases and the resulting architecture. The process underpinning the conceptual development will also be explained. Following this, the sections for implementation and testing will explain the technical implementation of the entire software, libraries used and methods used for testing.

Finally, this document will conclude with evaluating the product risk, to which extent the requirements put forward by the client were met, and conclude with challenges faced, future work, and how Hivemind can be used in the future.

1.2 Group members

RS | *HM*

Hivemind consists of five computer engineering students, namely Aurora, Harald, Hilde Marie, Nils Herman and Ruben. Each of them can be seen in tab. 1, which showcases their individual images, full names, initials, engineering disciplines and designated roles within the Hivemind project.

1.2.1 Initials

RS | *HM*

To the right of several headings in this report, the initials (tab. 1) of the authors and proofreaders of the corresponding subsections and subsubsections have been included. Although this is a group project, it is individually graded and reviewed. Accordingly, it is valuable to be transparent with regards to which member has contributed to each section of the report. The authors and proofreaders are separated by a vertical line, with the authors on the left, and proofreaders on the right. The proofreader is not responsible for the content, but rather confirms the general form of the section, and helps fix minor spelling and punctuation mistakes.

To further clarify the technical contributions of each group member, an overview of who has done what is provided in appendix K.

	***Name***	**Aurora Moholth**
	Initials	AM
	Discipline	Computer engineer - Cyber physical systems
	Role	Architecture & Competence flow & Team building
	***Name***	**Harald Moholth**
	Initials	HM
	Discipline	Computer engineer - Cyber physical systems
	Role	Requirements & Testing
	***Name***	**Hilde Marie Moholth**
	Initials	HMM
	Discipline	Computer engineer - Cyber physical systems
	Role	Documentation & Information flow & Social media
	***Name***	**Nils Herman Lien Håre**
	Initials	NH
	Discipline	Computer engineer - Virtual systems
	Role	Risk management & Document templates
	***Name***	**Ruben Sørensen**
	Initials	RS
	Discipline	Computer engineer - Cyber physical systems
	Role	Version control & Implementation

Table 1: Group members

2 Domain: unmanned flying vehicles

Unmanned aerial vehicles (UAVs) are the main domain of Hivemind, and refer to aircrafts that can be flown without an onboard pilot, as the name implies. In this report, UAV will be used interchangeably with the term "drone".

This section will briefly explain a number of concepts relevant to UAVs, such as defining what a drone swarm is, and how groups of UAVs are commonly used for agricultural, military and surveying purposes. A concise overview of how exactly swarms can be controlled through automation, and different network architectures to achieve control of the swarm, will also be provided. Finally, some algorithmic methods that can aid in calculating the route for each UAV will be presented.

2.1 What is a drone swarm? **HMM** | *NH*

There are many proposed definitions "swarm", where the most intuitive of these evoke the image of a large number of things moving together or in an organized fashion [10][11][12]. Others have chosen to define "swarm" in the technical context of robotics [13], emphasising on aspects of swarming behaviour where individuals group up to "achieve a common goal"[12]. In [14], this technical aspect was further detailed, defining a "swarm architecture" wherein the swarm is defined as having "distributed task queues, speculative out-of-order task execution, and ordered task commits". A set of criteria to define a robot swarm that encompass the ideas of a swarm containing a number of individuals working together to achieve a common goal in a self-organizing fashion was put forward in [15], which proposes that a swarm:

- contains 3 or more individual members
- is subject to or requires limited human control
- is cooperative (works together to meet a common goal)

2.2 Usage of drone swarms **HMM** | *NH*

UAV swarms have seen multiple theoretical and practical areas of use, and continue to be a field of continuous development. As UAVs can be directed from a distance, they allow operators to perform search operations in disaster areas remotely, mitigating the risk to operators. They are also a low-cost way to perform surveying or photography that would normally have needed helicopters or airplanes to complete. As computers grow smaller and the computational power on each individual member of the swarm increases, there are also increased opportunities for more sophisticated UAV swarm operations where the swarm can use the benefits of swarming behaviour and intelligence to complete tasks cheaper and faster than any individual UAV or drone operator could do by themselves.

The safety benefits of operating UAV swarms have been taken advantage of in a variety of scenarios. In [16] and [17], teams of UAVs were used in disaster management settings, specifically for tracking and extinguishing wildfires. UAV swarms have also been deployed for searching and tracking operations, for example in [18] and [19]. UAVs and UAV swarms also have obvious offensive areas of use in a military context. [20] and [21] propose frameworks for the use of UAV teams in search and attack missions, and the US army is planning on making use of the benefits of UAV swarms by deploying its own "Super Swarm" army [22].

Work has also been done in the field of UAV swarms for other tasks, such as cinematography [23], agriculture [24][25] and for medical [26] and commercial[27] delivery services.

While the literature on UAV swarms is particularly rich in the context of agriculture and military, there is a dearth of research into applications of UAV swarms for drone light shows.

2.3 Controlling the swarm

HMM | *NH*

It hopeless to expect one human operator to control all members of a UAV swarm individually. A certain level of autonomy from the drone swarm or the software that plans its routes is therefore necessary for any successful multi-UAV ground control station. Autonomy can be defined in relation to the amount of active interaction required from an operator or driver when the vehicle is operating. Although there is as of yet no consensus about the definition of levels of autonomy in UAVs, a number of suggestions for general levels of autonomy in unmanned systems and aviation have emerged based on the driving automation levels defined by the Society of Automotive Engineers (SAE).

The SAE divides automation into levels ranging from 0 to 5, where 0 is the driver having full control of the vehicle with assistance from automation functions, with 5 is full automation [1]. In [2], the authors apply this schema to aviation autonomy. They define levels 0 to 3 as levels wherein the primary pilot is the human, and the automation taking over from level 4 until 5. From least to most automated, the amount of work to fulfill a certain task that the human pilot is able to delegate to the system automation gradually increases, whereas the amount of events that the human needs to respond to through some kind of intervention (such as collision avoidance, decision making, identifying targets and so on) decreases. The idea of a tiered division of the extent of autonomy for system is taken even further in [3]. This paper does not build on the SAE levels, but instead divides any mission into subsystems, systems and the system of systems. These can further be divided into levels of autonomy based on whether an individual or group of unmanned systems can achieve its mission goal in a static or dynamic environment, with the most autonomous systems able to complete its missions as a group in a dynamic environment. In general, though there is no consensus on specific levels, the tacit agreement appears to be that the more autonomous a system is the more advanced tasks it is able to complete without the direct intervention of a pilot.

Another important topic to consider in an autonomous multi-UAV (or UAV swarm) system is how the pilot or ground control station should seamlessly control and interact with multiple UAVs. To break control down further into the steps required for a system to act autonomously, [28] defined a decision chain in autonomous vehicle into the perception and planning phases. The vehicles must receive information from sensors to perceive their environment, then use some algorithm or other method to decide what changes to make, if any, given the current

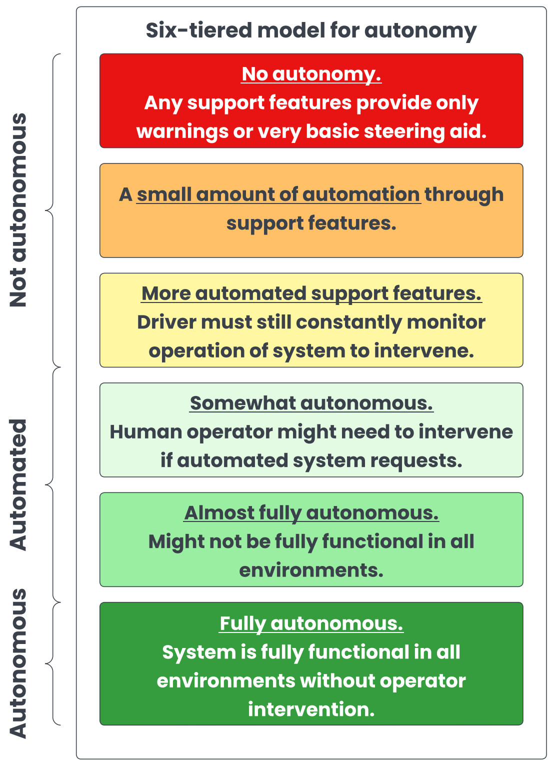


Figure 1: Six-tiered model for autonomy, as seen in [1] and [2]

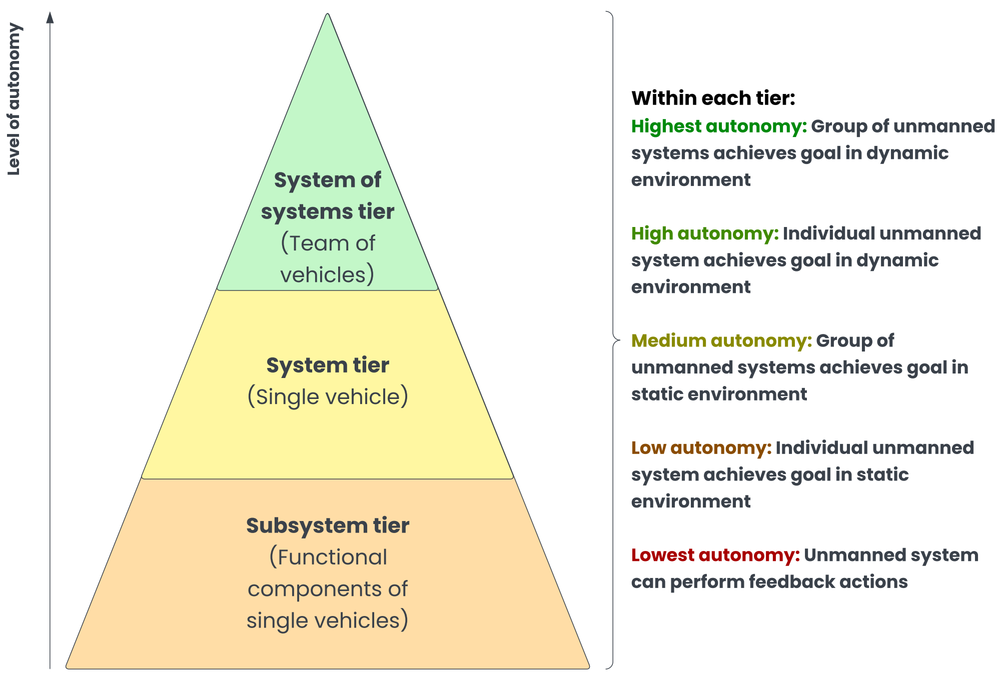


Figure 2: Three-tiered model for autonomy, as seen in [3]

environment. The question of where the perceiving and planning is done can be answered by many extant suggestions for swarm communication architectures.

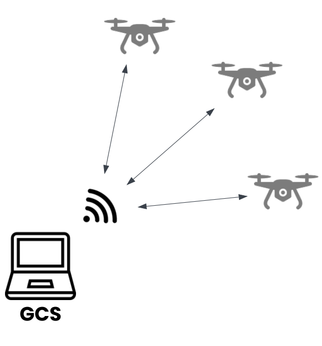
In most cases, at least some, if not all, of the planning takes places on a ground control station, which tends to have more powerful computational abilities than what is possible to implement on any individual drone or quadcopter. The computational capabilities of the members of the swarm further determines the level of interconnectivity and intelligence possible. In [29] and [4], the authors divide UAV swarm networks into four categories corresponding to the following descriptions:

- The centralized network, sometimes referred to as infrastructure-based architecture, wherein each individual UAV is dependent on direct communication with the ground control station to send telemetry. The ground control station then determines what action should be taken and sends updated instructions to each UAV. An example of this is seen in [30] or any other commercial software where one ground control station controls one or multiple UAVs with no other communication between members of the swarm.

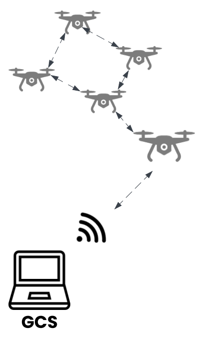
- The ad hoc network, where only one UAV needs to be in contact with the ground control station and acts as a router for the information to all other UAVs in the swarm.

- The multi-group network, which describes a network containing of several UAV teams that has its own router or backbone UAV that communicates with the ground control station

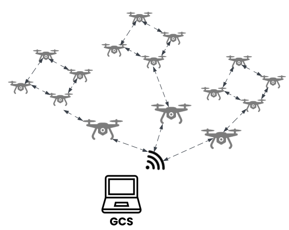
- The multi-group ad hoc network or multi-layer ad-hoc network, where the ground control station only needs to be in contact with any one UAV at a time, and this UAV further relays communication and telemetry between swarm and ground control station from the multiple extant groups



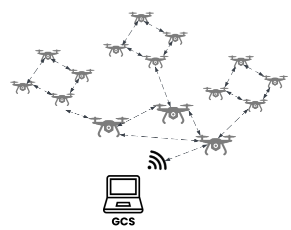
(a) Infrastructure based model



(b) Ad hoc network model



(c) Multi group model



(d) Multi-layer ad-hoc network

Figure 3: Swarm network models, as seen in [4]

In general, previous works indicate that swarms operating within a multi-group ad hoc network will be more robust [31] [32] [33], because infrastructure-based networks have a large weakness in its single point of failure of the ground control station. The more decentralized the network can become, the more likely the swarm is able to continue operating if one UAV shuts down. On the other hand, the more sophisticated the communication between UAVs, the higher the requirements for their onboard computational hardware will be.

While it is possible for the ground control station to individually calculate each member's ideal route, there are other algorithmic methods that simplify the act of controlling the swarm while preventing intra-swarm collisions. One well-known method is the Boid algorithm, suggested in 1987 as a general model to simulate a flock of birds [34]. The Boid algorithm is relatively straightforward and simple, containing three major rules for flocking:

- 1. Collision avoidance
- 2. Velocity matching
- 3. Flock centering

Further work has been done on the Boid algorithm in more recent times, extending it to controlling UAV swarms, for example in [35].

Many other rules have been implemented in research to control a swarm of UAVs, many which include methods for collision avoidance and velocity matching, reminiscent of the Boid algorithm. In [36], the authors implemented formation controllers and collision avoidance controllers on each UAV, which allowed the swarm to maintain a formation while still avoiding obstacles. This approach is an example of the consensus-based approach to formation control, wherein members of the swarm uses the states of its neighbors to adjust its own state until the states of each UAV converges upon a group consensus [37]. Other examples of researchers who have taken this approach to control UAV swarms include [38] [39] [40]. A more straightforward and potentially less computationally demanding approach is the leader-follower method, where a leader UAV is assigned and all other members follow the leader using specified rules, as has been seen in [41] [42] [43]. A final commonly seen approach defining the drone as a virtual structure, proposed in [44]. Here, all the members of the swarm are defined in a structure, with a rigid geometric relationship to all the other swarm members and to a point of reference. This approach has been used to control multiple drones simultaneously, for example in [45] where drones were organized into a virtual structure and controlled simultaneously using a joystick.

Apparent from the existing literature, research into UAVs, UAV swarms, their applications and the most efficient way of controlling these is abundant. UAV swarms, which refers a group of three or more UAVs, that are at least partially autonomous and are able to work cooperatively. Such swarms have a large range of real-life applications, including within military offensive missions, geographical surveying, and agriculture. A number of different methods have been implemented to communicate throughout and with the swarm, and to control the paths of each individual UAV. When choosing what method to implement, it is important for researchers to take into account the onboard hardware of their drones and their computational capability, the range over which they should travel, as well as the ease of which the different solutions can be implemented. Although the research certainly covers a wide range of different application of drone swarms, limited attention has been placed on tailor-made software specifically for planning and executing light shows. The next section will present how Hivemind seeks to start filling this gap, through creating a framework for a flexible and extendable software which can be used for planning light shows.

3 Problem: Route-making in drone swarm management

3.1 Project context

3.1.1 Kongsberg Defence & Aerospace

HMM | *NH*

Kongsberg Defence & Aerospace (KDA) was founded in 1814, and is at present Norway's largest manufacturer of defence systems [46]. Throughout its long history of over two centuries, the company has developed notable products such as the Penguin missile, Norwegian Advanced Suface-to-Air Missile System (NASAMS), Joint Strike Missile and Naval Strike Missile. They supply defence systems to a number of different countries, including the Norwegian Armed Forces [47] and the United States [48].

As a company, their values include innovation and collaboration, which is perhaps one of the reasons why their co-operation with universities and students across Europe is particularly strong. KDA supports the development of students every year through a number of internships, a competition, the industrial master's degree program, in addition to providing projects and guidance in bachelor's thesis projects at the University of South-Eastern Norway (USN). Their oldest running continous summer internship is the Local Hawk project, which has been arranged every summer since 2008. This project focuses on performing Unmanned Aerial Vehicle (UAV)-related research, such as methods to increase drone flight time, and is a project meant to give students an opportunity to work in a team to complete a practical project.

3.1.2 Envisioned drone swarm light show

HMM | *NH*

In 2024, Kongsberg will be celebrating the 400th anniversary of its founding. For this occasion, KDA was planning on arranging a drone light show with drones and software developed by student projects. The show was envisioned to take place somewhere in central Kongsberg, with drones flying in formation and illuminating certain buildings. As a drone-related project, this is an extension of the Local Hawk summer project.

To do this, various engineering bachelor's project groups were tasked with the different elements necessary to arrange such a drone light show: the drones themselves, the systems and hardware necessary to control the spotlights, the flight controller software on the drones, and of course, the route planning software to allow one operator to plan out the entire spectacle.

Unfortunately, as all the student projects progressed and discovered the limitations of hardware and software in relation to the requirements put forward, it became apparent that the safety issues involved in arranging a light show involving a team of drones, and the technical work required to address these issues, was too extensive for the time allotted to each individual project. For example, one goal was to design drones with a total weight of less than 250 grams to avoid the strict requirements related to registration, training and permits to fly. Such drones, however, would necessarily carry only the bare minimum of sensors, which severely limits each individual drone's ability to perform calculations and maneuvers related to dynamic anti-collision. The lack thereof poses a significant safety hazard. The spotlights necessary for the drones to perform their light show function, additionally, turned out to be strong enough that safety equipment was needed to work with them. This also presents as a potential hazard to the drone light show audience. After much deliberation, the actual execution of the drone light show in 2024 has been shelved due to the difficulty of safely arranging it.

This project was assigned the task of developing the route planning software for the light show. Although the software will most likely no longer be used for a light show in 2024, however the project was still completed as planned with the aspirations that the software would be useful to someone planning routes for a drone swarm in the future.

3.2 User requirements

AM | *RS*

Requirements were derived through a preliminary meeting with the client. In this meeting, the client (KDA) put forward a number of requirements and aspiration for the route planning and monitoring software. 10-15 drones would perform the show simultaneously in sequence, and the software would need to be able to control and communicate with the drones. As the operator of these drones, the client imagined being able to open their computer, set a location, and have the computer query the drones for their locations. The computer would then display a map of where the drones are, and the operator could then dictate what should happen. The computer would then generate a plan for each drone and allow the operator to run the program. The operator would also need to know the status of all drones at all times, including their locations, connection status with the computer, battery status, and expected flight time based on battery level. In the case of the drone battery level falling below a certain threshold, the operator would be alerted through the software. If the battery level were to drop even lower without intervention, the drone would then be forced to perform an emergency landing. However, the client wanted to keep the ability to override the emergency landing, if necessary. Time is of the essence; therefore, the drones would need to be synchronized in terms of time, and the software should facilitate this.

The operator should be able to create a scenario (that is, a timed route plan for all drones) and simulate it before running the program. They should also be able to save and load previously created scenarios. For safety reasons, the software should also include the option of designated no-fly zones for the entire scenario which would be taken into account by the route planning algorithm. Similarly, the software would also need to keep track of buildings, trees and other obstacles to avoid collisions. The client recommended obtaining the data from "hoydedata.no" for this purpose. The client also requested the ability to designate emergency landing zones where drones could safely land without the risk of harming spectators if necessary.

The client requested that all drones should be able to locate each other, which should be a simple task as the envisioned drone light show area will be fairly small. It was recommended that equirectangular mapping was investigated for this purpose. It was also suggested to implement a follow-the leader mode of formation control.

These requirements can be roughly distilled into the following list, showing that the software should:

- feature a graphical user interface through which the light show operator could interact with the system
- enable the planner to ensure the drones fly safely in relation to spectators
- ensure the drones do not collide with each other
- include functionality to perform a safe emergency landing if necessary

In addition to these functions, the client requested that the software:

- have a mode for illuminating buildings or other structures visible on a topographical map
- allow the user to store and upload scenarios/routes that have already been made
- have a mode to direct the drones to fly in formation following a leading drone
- allow for the use of more than one drone to fly in a swarm
- allow for the simulation of the drone swarm to verify that the selected route is realistic and will not lead to collisions

3.2.1 Making the requirements verifiable

HM | *HMM*

After this client meeting, a list of requirements were derived from the client's wishes, which were then developed into 11 user stories that covered the main functionality from a user standpoint that the client mentioned, see appendix C. The user stories were then developed into 11 use cases containing step by step lists of how each user story would appear in practice, see appendix D. Finally, each use case was expanded until a list of 73 verifiable requirements had been defined, see appendix E.

3.3 Minimum viable product

HMM | *NH*

After exploratory technical work to determine how to implement the requirements agreed to by the client, it became clear that the scope of the project would far exceed that of a bachelor's thesis if planning, simulation and monitoring of a light show drone swarm were to be completed. In agreement with the client, the project therefore focused on the planning part of the Hivemind software, with particular attention paid to designing software that is scalable and flexible to allow further additions in the future.

3.3.1 Features

HMM | *NH*

The goal for the Minimum Viable Product (MVP) was to be able to plan a simple route for one drone using two keyframes. A keyframe is the position of a drone at a certain point in time, with the goal being that the Hivemind software is able to calculate the path of a drone between two keyframes using a specific algorithm. After defining two geographical points, the software will calculate the simplest possible route between these.

The software will be able to save and load scenarios. Scenarios are files that gather routes for several drones, though the MVP will only support one drone to start with. A route is the flight plan for a single drone, and the MVP will accordingly be able to produce scenarios containing only a single route. It was initially decided that these scenarios should be saved into and loaded from XML formats.

It was also decided that the system should be able to convert from geographical coordinates to cartesian coordinates, as it is far more intuitive for a drone operator to direct the UAVs in a relative XYZ system than using a geographic coordinate system.

3.3.2 Modules and functionality in the MVP **HMM** | *HM*

- Graphical User Interface (GUI) with tabs and options
- Save/load scenarios
- Map data query
- Height data query
- Coordinate system converter (geographical to cartesian)
- Basic route algorithm (Routemaker)
- Basic graphical representation of key frames in GUI
- Functionality for a single drone

3.3.3 Requirements and tests for the MVP features **HMM** | *NH*

The requirements and tests have been adapted from the full requirements table, which has also been altered to expand the number of optional requirements. Appendix F shows the revised requirements table, separating between requirements categorized *A*, *B* and *C*, where the requirements categorized as *A* define the minimum viable product.

A preliminary flowchart was developed while simultaneously working on the MVP, which can be seen in appendix B. This figure, while only being an initial draft, nevertheless provides a visual representation of the proposed system's key functionalities and how they are connected. This early planning stage allowed the team to identify potential bottlenecks and areas for improvement in the MVP development process. As the project processed, the group returned to the flow chart and other representations of the MVP to make alterations as necessary.

3.4 Project problem **HMM** | *NH*

The specific problem solved by Hivemind is the creation of a flexible and versatile software architecture and software implementation of a drone route planning software. This software was envisioned as part of a KDA led initiative to perform a drone light show in central Kongsberg in 2024. Though the light show was cancelled due to the difficulty of properly implementing measures to ensure the safety of spectators, the Hivemind project was nevertheless completed. The assignment given by the client was to create a route planning system that could create a route for at least one drone, and should include both a planning, simulation and launch mode. The client also requested some specific functionality for these modes, such as no-fly zones, emergency landing, monitoring, and functionality for spotlight control. Tests were defined for all the client's requirements.

If the project was unable to complete all these functions, the software should nevertheless be designed in such a way that it could be completed by someone else. This added requirements that the software design should be scalable and flexible. A minimum viable product was developed to meet the most important requirements for the clients, while creating a versatile starting-point for further development of Hivemind.

As this report continues to detail the project work methods and technical development, it will also become apparent that the scope of the project gradually shrunk. This is because both client and group gradually realized how much work it would be to implement planning, simulation and monitoring of a drone light show. Consequently, the software that is presented toward the end of this report is markedly different in functionality from what was initially put forward by the client. That being said, all adjustments to the list of included functions and the requirements were continuously altered in close cooperation with the client. The actual functionality built into the final product ordered by the client will be detailed further in the minimum viable product section.

4 Related work

In this project, the customer requested drone swarm route planning software that includes functionality for directing a light show, anti collision, simulation and emergency landing. Given these customer specifications, there are two main questions that need to be answered to guide the direction of the project.

- First, what algorithms and strategies are there for efficient route planning? Which ones are the most relevant for this project's given restriction?
- Second, what sort of software for route planning, simulation and execution already exists out there? What are their strengths and limitations?

This section will address these two questions.

4.1 Route-planning algorithms **HMM** | *NH*

Route or path planning algorithms are algorithmic ways of arriving at the fastest or most efficient way to for an individual to travel through all its designated way points. Although it is quite possible for humans to draw up a route intuitively, some research has shown that humans seldom choose the shortest possible path [49], especially when the total distance between origin and destination increases [50]. Wholly manual path planning is, however, neither scalable nor feasible in the context of Unmanned Aerial Vehicle (UAV) swarms, and computer-generated routes are thus an important part of the aspect of autonomy necessary for the route planning software.

It is possible to roughly divide path-planning algorithm into different categories, based on the approaches taken to calculate the optimal path. In their survey on different algorithms for agricultural use, Basiri et. al. define the four categories into algorithms using grid-based techniques, sampling-based techniques, artificial intelligence and cooperative techniques [51].

In grid-based techniques, the programmer uses a grid where the various points on the grid correspond to possible positions of the vehicle, and the vehicle can move freely between adjacent points. Some times, the links between two neighboring points (called edges) can have different costs associated with them. Examples of this kind of algorithm is the A* algorithm [52] and its variations, such as Theta* [53], or Dijkstra's Algorithm [54]. Such algorithms have been applied to unmanned vehicles to generate paths and are often simple to implement, which can be seen in [55] where the author used a modified A* algorithm to plan routes for an unmanned surface vehicle.

Sample-based techniques base themselves on the principle that not every path needs to be tested to find an optimal one. Instead, paths will randomly be sampled, with the global optimal path being saved. This global optimal path can be saved to guide the next random sampling until the best path has been found. Examples of this kind of approach include particle swarm optimization [56] and rapidly exploring random trees [57]. Particle swarm optimization, in particular, has seen frequent use in flight and surface path planning experiments [58][59][60][61].

Artificial Intelligence (AI) is another method that has been used for route planning, and has the advantage of being able to infer information based on previous experience, which may reduce the number of sensors needed to be mounted on drones [62]. Examples of AI methods that have been used in path-planning problems include neural networks [63], ant colony optimization [64] and genetic algorithms [65].

Finally, the cooperative techniques encompass a large number of different models that can be used to generate paths for robots, including machine learning models, mathematical models, multi-objective optimization models and bio-inspired models [66]. These techniques include Bezier curves, which can be used to plan out smooth the curves of the path, an example of which can be seen in [67].

Different algorithms may perform better in different circumstances. In using path planning algorithms to route end-to-end data transfers between UAVs, one study found the A* algorithm to perform the best [68]. This is also the algorithm that was more efficient in another survey related to path planning in 3D for agricultural purposes [51]. In another project, which compared algorithms for the purpose of finding the path with the minimum cost in an environment with dynamic obstacles, the bug algorithm was the most successful [69].

4.2 Existing solutions for controlling drones and drone swarms

HMM | *NH*

A number of commercial software solutions for UAV flight planning and simulation already exists. These often allow the use of waypoints to define a rough route, such as UgCS [70], and are often designed for surveying an area (such as the DJI route planner or the Orbit Logic UAV Planner)[71][72], or performing surveillance in an area, in addition to flyover route planning for mapping purposes, such as DroneDeploy's planning solution [73]. Such terrain mapping solutions also feature terrain avoidance functionality, and often contain a number of intelligent routes for surveying an area. These solutions, however, do not have built-in support necessary for light shows, such as controlling the direction of spotlights attached to the drones being controlled. They are also often expensive (790 to 4390 EUR for the UgCS software [70]), which poses a challenge for a small bachelor's research project or hobby pilot. Some extant software is also not model agnostic, and may only support planning for one kind of drone (such as DJI's planning software) [71]. There are open source solutions available that could be adapted to allow for spotlight control, such as QGroundControl [74] and the PaparazziUAV software [75].

Different frameworks, software or systems for UAV mission planning have also been suggested in academic literature. In [76], the complete software with the aim to optimalize energy expenditure over a route was proposed for multiple UAVs. In [77], software for flight management in the context of agricultural image processing was proposed using MATLAB. Researchers have also proposed full control station software for single [78] and multiple [79] vehicles, in general for the purpose of surveillance. Finally, work is also being done in pursuit of developing better, more realistic simulations for multiple UAVs. An example of this is seen in [13].

Some architectures have already been proposed that fall within a similar problem domain.

In [80], a framework for an swarm intelligence system using machine learning is proposed. Swarm intelligence is also used in [81] to implement a software architecture specific to UAV swarms for firefighting purposes.

Evidently, the field of research related to UAVs, route planning and swarms is vibrant and diverse, and in many cases, the jury is still out on exactly which approaches are optimal in the different scenarios that such technologies can be used. Hivemind will cover only a tiny scope of the body of work as a whole. While not necessarily able to do so during the course this bachelor's project, the aspiration is that the Hivemind software will be able to bolster the somewhat underdeveloped area of research that encompasses software and techniques related to UAV light shows. In time, this software should also be able to perform repeatable experiments and demonstrations in relation to drone light shows, to further enrich current empirical research on drone swarms, light shows and optimal path planning.

The next section of this report will start by explaining how the Hivemind project group was managed, before moving onto the more technical aspects of developing the Hivemind software.

5 Project management

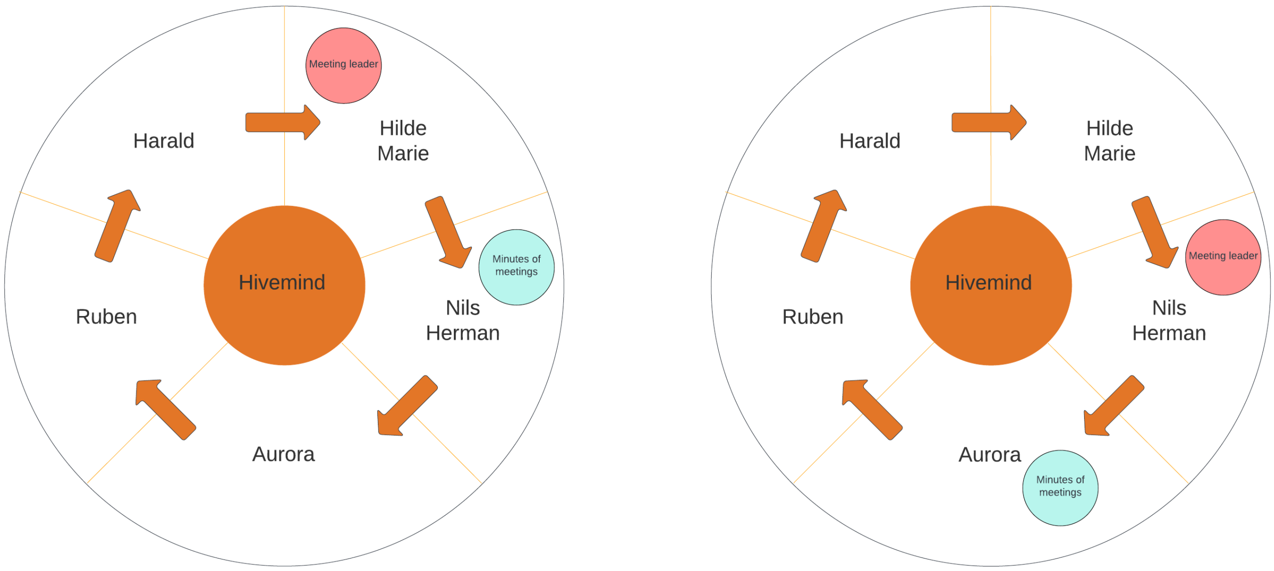
This section deals with how the organizational and administrative tasks, an important requirement to the bachelor's project, were solved. This includes how administrative roles were divided, how Hivemind communicated with its external and internal supervisors, how risk was considered for the project, and finally how the Hivemind website (a university-set requirement) was implemented.

5.0.1 Flat leadership structure

NH | *AM*

A flat leadership structure was chosen early in the project to make sure the burden of administrative tasks was evenly distributed among the group members. In practice, this meant that no one person was given overall responsibility for the project as project leader, and that administrative roles such as meeting leader and secretary was rotated weekly.

A wheel was made to keep track of whose turn it was for an administrative role(fig. 4a). Hilde Marie was the first to take on the role as meeting leader, while Nils Herman acted as secretary. The next week, whoever acted as secretary would take on the role of meeting leader, while the person after them in the wheel would act as that week's secretary(fig. 4b).



(a) Meeting leader wheel

(b) Meeting leader wheel progress

Figure 4: Meeting leader rotation

In addition to spreading the administrative work across the individual Hivemind members, this leader wheel solution provided an opportunity for each member to gain experience performing administrative tasks. It also ensured that each member needed to keep up-to date and involved with the project as a whole, instead of just focusing on their own tasks.

Throughout the project, this way of managing the project proven very successful. The flat leadership structure proved to be flexible, and encourage both communication and initiative from all the group members. The ease of communication in a flat structure facilitated communication and made sure every member felt a sense of ownership of the task at hand.

5.0.2 Team building

NH | *AM*

Every Friday, provided the group was on schedule to reach that week's sprint goal, one hour was dedicated to team building activities. A large variety of activities were covered throughout the project period, determined each week by the designated meeting leader, and included playing games, singing karaoke, and enjoying ice cream together. The inclusion of these social activities served as an incentive for the group to work hard throughout the week, ensuring that all sprint objectives were accomplished. It allowed the group to have a dedicated time for team bonding and facilitated the development of positive relationships within the group.

5.0.3 Seating arrangement

AM | *NH*

The Hivemind workspace had three desks. All tables were set against the walls to maximize space. In addition to the desks, the room included a book cabinet for the green binders and two armchairs for supervisors. This layout made it easy for the group to communicate, the room felt less crowded, it was possible to relax in the armchairs during breaks, and the supervisors also had dedicated seating during meetings(fig. 5). If you would like to see the development of the seating arrangement you can see Appendix N.

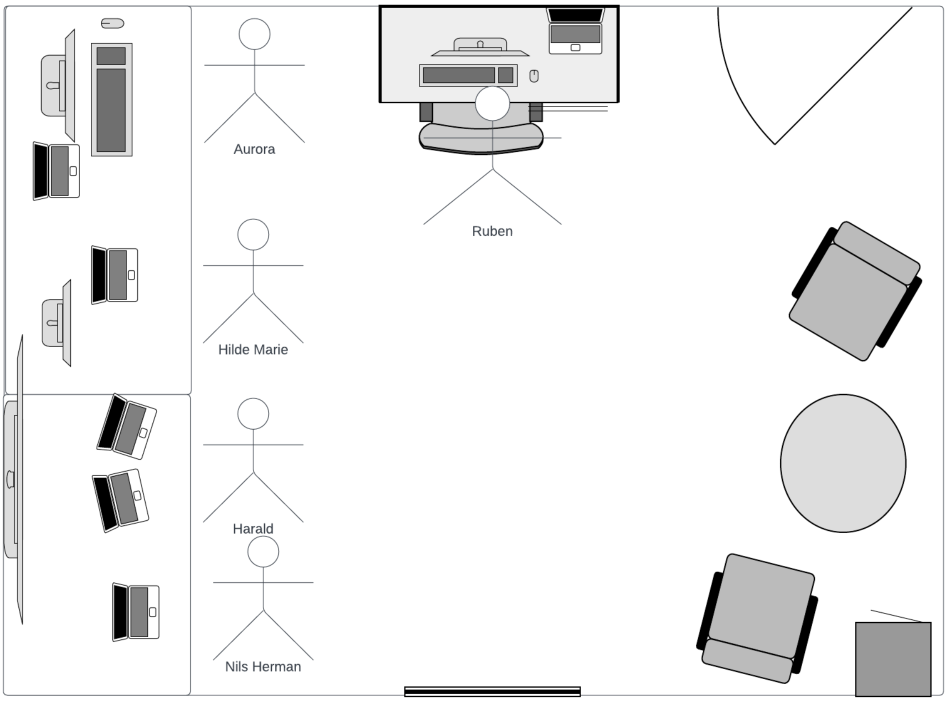


Figure 5: Workspace layout

5.1 Supervisor communication

NH | *HM*

There were two supervisory roles in the Hivemind, the internal and the external supervisor. The internal supervisor's main task was to ensure the group provided all the necessary documentation, give guidance on project methodology and when to do what, and to assist if there were any conflicts that arose during the project. The external supervisor role entails technical guidance for the project, especially in the case where a given assignment might involve very specialized or proprietary software. The client (KDA) for this project also doubled up as one of Hivemind's external supervisors.

5.1.1 External supervisor (client) **NH** | *AM*

All communication and interaction with the external supervisor occurred through meetings and email correspondence. Weekly meetings were scheduled every Friday at 13:00 in the Hivemind workspace. In these meetings, the client was able to monitor the progress of the project, provide guidance, and also give technical advice as the external supervisor. Risk analysis was also a part of these meetings, which helped inform the project and product risk analysis.

These meetings were crucial in ensuring that Hivemind continued to progress in the right direction, and for members to receive invaluable help in terms of techniques and new technologies that could be utilized to solve present and future problems.

5.1.2 Internal supervisor **NH** | *AM*

Hivemind also met weekly with the internal supervisor to discuss the progress of the project. During these meetings, the internal supervisor offered feedback and guidance. The internal supervisor focused on the academic aspects of the project and how Hivemind could meet the university's requirements for a bachelor's thesis. To aid in this work, Hivemind also submitted regular status update documents (called follow-up documents) that described the overall status of the project and updates from individual group members, containing work done and challenges encountered.

In addition, the group provided its internal supervisor with a comprehensive follow-up document that included the overall status of the project as well as individual updates from each group member. These updates contained all accomplishments, highlighted any challenges encountered that week, and described the strategies employed to overcome those challenges.

5.2 Project risk analysis **NH** | *AM*

Risk analysis is an essential aspect of any engineering project. Effective risk analysis can help the project group identify potential threats and opportunities early, enabling proactive steps to be taken to mitigate risks and minimize project disruption. The risk analysis can be found in appendix J.

5.2.1 Definitions and risk matrix **NH** | *AM*

In this report, risk has been separated into project and product risk analysis. The project risk refers to risk that are associated with the management and execution of the project, such as schedule delays and team conflicts. Product risk refers to the expected risks related to using Hivemind, and will be covered in a later section.

In order to manage risks effectively, it is crucial to have a clear understanding of the potential risks that exist, their likelihood of occurrence(tab. 2), and their potential impact on the project and the product(tab. 3).

Definition of probability		
Degree of probability	Frequency	Interval
1	Very low	Happens very rarely
2	Low	Happens rarely
3	Medium	Happens sometimes
4	High	Happens often

Table 2: Definition of probability

Definition of degree of consequence for project		
Degree of consequence		Outcome
1	Insignificant	Project continues as normal.
2	Small	Project becomes delayed slightly, but minimal effect on end result.
3	Considerable	Project becomes stagnant, measures required.
4	Serious	Project stops, critical measures required.
5	Disastrous	Project cancelled.

Table 3: Degree of consequence

Risk is evaluated by multiplying the probability of some event occurring and the degree of consequences should this event occur(fig. 6).

Risk = degree of probability x degree of consequence						
Degree of probability	4	4	8	12	16	20
	3	3	6	9	12	15
	2	2	4	6	8	10
	1	1	2	3	4	5
		1	2	3	4	5
		Degree of consequence				

Figure 6: Risk matrix

Once the risk level has been calculated, it is used to determine an appropriate mitigation strategy. Events at a high risk level requires mitigation, either by reducing the likelihood of the event occurring or by reducing the severity of the consequences. Risks with medium risk level warrant ongoing monitoring and periodic reassessment, while risks with a low risk level may be acceptable without any further action.

Risks are further divided into internal and external risks. Internal risks are risks that come from inside the group such as a group member falling ill, while an external risks refer to external factors such as a pandemic. Both types of risk will impact the project, but internal risks are easier to mitigate the probability for. In the case of external risks, there is no course of action other than attempting to anticipate and find measures to mitigate the severity of the consequences of these events.

5.2.2 Continuous analysis

NH | *AM*

Risk was continuously monitored and updated as Hivemind evolved. This work included updating probabilities, degrees of consequence and mitigation, but also adding new risks.

One example of this is the updated mitigation policy for cases where multiple members fall ill simultaneously. At one point in the project, two members participated in the first presentation while ill. This caused the rest of the group to fall ill like dominoes in the following weeks, causing delays in Hivemind's progress. Consequently, a new stipulation was added to require mask-wearing of any member who chooses to still come to the workspace while ill.

Point 18(appendix J) was also added during the project as opposed to in the initial risk analysis, because some tasks were failed to be finished on time. This caused a week long delay in the project. To mitigate this, a new mid-week addition of each member evaluating if they could finish their tasks on time, and if not, ask for assistance, was implemented. This mitigation has allowed the risk likelihood related to delays in the project to fall, because other members were able to assist with any task that took longer than anticipated.

5.3 Website

NH | *AM*

Implementing a website for Hivemind was one of the requirements for the bachelor's thesis set by the university. This website was to contain:

- detailed information about each group member
- an overview of the project
- summaries of each work week

The website is hosted on both a University of South-Eastern Norway (USN) server as well as a private server, which operate independently to ensure availability. Both servers are connected to the same database, ensuring consistent content across both websites.

The website employs PHP for easy access to the SQL database that serves as the repository for all information on the site. This approach also enables securing the database username and password from potential web-based access. To enhance the websites design and ensure it is portable across multiple devices, including phones, tablets and PCs, the "Bootstrap" library was used. Bootstrap offers a multitude of pre-established classes, which simplified the design process and ensured scalability.

Hivemind maintained a separate repository for the website on AzureDevops, a tool for software development and project management, which will be described in further detail in section 6.2. AzureDevops allowed for dynamically updating the website each time any part of the project was updated, which significantly simplified website updates.

Information on the website dynamically updated also included the documentation for the Hivemind source code. Each time a feature was completed and updated in the AzureDevops repository, the documentation on the website would also update. This arrangement provides straightforward access to the various segments of code documentation and the coding standards implemented by our group.

Updates to the website introduced new features, such as modals for each group member. The modals operate by storing all relevant information queried from the database on the button for each group member. Using JavaScript, the data from each button is then allocated to its corresponding location within the modal. This functionality allows us to maintain a single dynamic modal instead of generating a separate modal for each group member.

The website is designed to be dynamic, pulling all content directly from the database. This approach offers significant advantages, particularly in terms of scalability and content management. Adding new content to the website is a seamless process, as it simply involves adding a new entry to the database. This eliminates the need for manual coding or modifying the website structure.

The database itself is structured following the principles of normalization. By adhering to these normalization rules, the database is optimized for storage and retrieval of information, resulting in a well-structured and efficient system.

The website implementation is an embodiment of how Hivemind implemented smart work methods and thorough preparatory work in order to spend the time period designated for development work to focus purely on coding and implementation. The next few sections of this report will focus on the technical preparatory work that built the flexible foundation for Hivemind.

6 Software development process

A clearly defined software development process will help ensure the project progresses smoothly, and that all work done is done with the common purpose of completing the intended product. This section will outline the development process for Hivemind, including the chosen project model and choice of design languages and methods for representing the abstract design of the software. The technologies used that assisted in software development will also be introduced, before outlining how verifying requirements were systematically planned to ensure the final product delivered conformed with the client's requests.

6.0.1 Overview of the development process

AM | *RS*

As shown in fig. 7, the first step in developing was meeting the client and determining their requirements for the software. Using this information as a starting point, user stories were defined, which served as the basis for making the use case descriptions. The requirements were made through an analysis of the use cases. When the requirements had been defined, the scope of the Minimum Viable Product (MVP) was determined and the list of requirements cut down to only reflect the MVP.

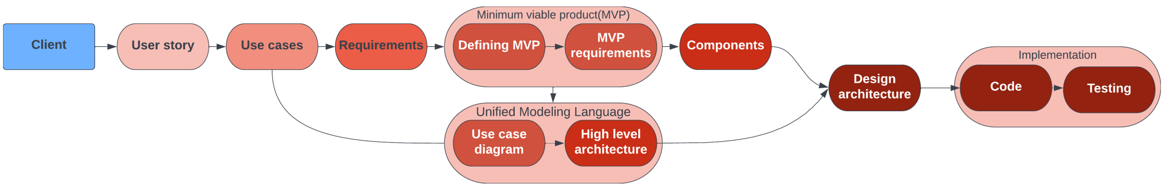


Figure 7: The "red thread"

After establishing the definition of the MVP and its corresponding requirements, all the project software components necessary for the MVP based on the requirements were identified. These software components show the systems essential functionalities.

Various design approaches were considered for the Hivemind software architecture. To evaluate the effectiveness of each model, architectural criteria were established. The layered architecture was eventually determined as the most suitable model for Hivemind. A use case diagram was made with an actor that interacts with the system, and the software components were integrated into the layered architecture.

After establishing the software architecture, work on the software components was started using an agile development approach. Each software component was coded, tested, and reviewed before being integrated into the system. Continuous integration and deployment was used in this process to ensure that the software components worked seamlessly together.

Throughout the coding process, regular testing was carried out to ensure that the system met the requirements and performed as expected. The testing involved both manual and automated tests, including functional and integration tests. Any issues or bugs discovered during testing were promptly addressed, and the software components were retested to ensure that they functioned as expected.

After the initial coding and testing phase for the MVP, additional advanced features necessary for the product to fully meet the client's requirements were added. This meant going back to the requirements and expanding or modifying the software architecture before coding could continue.

To implement these new features, new requirements were identified and and the necessary software components distilled from this. Then, the current software architecture was evaluated to determine any changes needed to accommodate the new features. Once the architecture was updated, coding and testing resumed until the advanced features were completed.

6.1 Methodology **NH** | *AM*

There were three pillar's to Hivemind's software development process. The first was to use an Agile methodology throughout the project to remain flexible, and to prevent the project from spending too much time on aspects of the software that turned out to be fruitless. The second was to use Unified Modeling Language (UML) to design and represent the various parts of the software before starting any coding, ensuring that the resulting architecture was clear and flexible. Finally, implementation of all software components followed an iterative approach, with the simplest and most essential functionality being added and tested before creating more advanced, but non-essential, features.

6.1.1 Project model **NH,HMM** | *AM,NH*

Hivemind practiced an Agile method based on Scrum. Scrum is an agile framework for managing and organizing projects which provides a flexible and iterative approach to software development. The major difference between the Hivemind Agile method and Scrum was that core roles within a Scrum framework, such as product owner and Scrum master were missing. Hivemind did, however, make use of a product backlog, organize work in sprints, establish daily stand-up meetings and held both sprint reviews and sprint retrospectives after each ended sprint.

Hivemind's practiced week-long sprints. Each day was started with a stand-up meeting, during which group members shared work done the previous day, outlined their plans for the current day, expressed how they were feeling, and (after this was added) discussed any difficulties faced. The segment in the stand-up where members expressed how they were feeling that day also diverges from traditional Scrum practices, and was a deliberate addition to stand-ups to help provide insight into each member's well-being. At the conclusion of each sprint, a retrospective was conducted to reflect on whether the previous week's goals were met, identify areas for improvement, and acknowledge which aspects of the sprint were successful. The task board was also reviewed to decide whether any incomplete tasks were to be carried over to the next sprint, or placed in the development backlog.

Each retrospective was followed by a sprint planning meeting to define goals and set task for the upcoming sprint. Administrative tasks were also added to the sprint board, even though this is not common, as this made it easier to visualize each member's work load and to remember to do these administrative tasks in addition to technical work.

The sprint task board employed the following categories:

- Sprint Backlog
- Active
- Resolved
- Dropped
- Closed

The Dropped category is not a normal addition to task boards, but was added for a visual representation of tasks that were deemed unnecessary to complete for various reasons. The Sprint Backlog contains new tasks that have not yet been initiated, while the Active column houses tasks that are currently in progress, but not yet completed. The Resolved column contained any tasks that a member had finished, but that still required verification by someone else. Finally, the Closed column was reserved for any tasks that were completely finished.

6.1.2 Design language and software models **HMM** | *NH*

UML was used to define the Hivemind software models. This is a general-purpose design language that can be used to create and visualize aspects of a system's design. In Hivemind, this was used in particular to create use case diagrams and the software architecture. The process through which this was done will be outlined in a later section of this report.

6.1.3 Implementation of software components **HMM** | *NH*

Software components were implemented in an iterative fashion once the architecture had been finished and interfaces determined. In general, one designated Hivemind team member was given responsibility for each software component, though it was acceptable and encouraged for members to work together on developing software components. This was first, to ensure that each member had at least one technical software component they were wholly in charge of, second, to ensure that asking another member for help did not risk the other member taking over the technical work for this software component, and finally, in this way encourage helpful co-operation.

All software components had their own list of necessary functionality that needed to be implemented for Hivemind to function as required. For many software components, this meant the functionality required to interface correctly with the rest of the software. Some software components also had some advanced functionality that could be implemented when the basic version of Hivemind was functional. This included dynamic height data updating for the Height Manager and improvements to the Graphical User Interface (GUI) that enhanced user friendliness. This demonstrates the iterative implementation of software components.

6.1.4 Project Timeline

NH | *AM*

In the intitial stages of the project, a project timeline was developed that served as a flexible framework outlining which sprints would be used for planning, preparation, coding, and work on this final report and the thesis presentation. The timeline provided a loose guideline for when certain aspects of Hivemind should be completed. This document was intended as a flexible plan, which might need tuning as the project developed. This was not needed, however, as the timeline has remained consistently accurate throughout the project. It ensured sufficient time was given to each stage of the project. The timeline can be seen in appendix M

6.2 Technologies used

6.2.1 Programming language

RS | *HM*

Hivemind was developed in C++ 17 with an object-oriented approach. Both the university and the local industry focuses on C++, and as C++ allows for a focus on efficiency and optimization on a high level, it was logical to utilize it for an algorithm-heavy software. C++ has a lot of features, and can be programmed in many styles. During development, the usage of the singleton pattern[82] proved useful for components that needed to have a global state. A set of coding standards were therefore set in place, to make sure the codebase remained consistent. The coding standards were made part of the code documentation, and can be viewed both online, and in appendix O.

6.2.2 Development/Target platform

RS | *HM*

Ubuntu Linux was chosen as the Hivemind development platform. This is also the platform targeted in terms of release. This is because the Hivemind software will be intended for use with drones through the Robot Operating System (ROS) libraries and tools, and Ubuntu Linux is the target platform for these.
In order to ensure a uniform development platform for all team members, a virtual machine was created with all the packages and tools needed to develop Hivemind pre-installed. The virtual machine was then installed on each team member's computer, allowing each member to customize their platforms as they saw fit. Setting things up this way ensured each team member develop Hivemind using the same versions of dependencies.

For easier maintainability and flexibility, using a technology like Vagrant should be considered in the future. Vagrant allows us to configure a virtual machine and its dependencies and setup using one or several configuration files. This would be committed to version control, and other team members could retrieve updated dependencies easily by updating the virtual machine through Vagrant.

6.2.3 Azure Devops

RS | *HM*

The client, KDA, provided Hivemind with access to Azure Devops, a platform for handling software development throughout the whole life-cycle. Azure Devops features tasks management in sprint boards and a common platform for handling version control with git, automated Continuous integration and Continuous delivery/deployment tasks with Azure Pipelines, as well as a Wiki for Hivemind to keep documents, notes and personal diaries centralized.

6.2.4 Google Drive

NH | *AM*

Hivemind utilized Google Drive to store all its project documents. The Drive was organized into distinct folders for technical work, administrative work, presentations, reports and personal files. This clear folder structure enabled easy navigation and swift access to the required resources.

6.2.5 Version control

RS | *HM*

For version control of Hivemind's codebase, git was used. There are several great Version Control System (VCS) available, but the rationale for using git is that it is widely used and understood in today's software industry. It is also the only VCS the entire team had experience with. Although git is a distributed VCS, Hivemind worked with a central remote repository hosted in Azure Devops. This makes it much easier to handle merging of several branches in one place, minimizing the amount of conflicts.

6.2.6 Doxygen

RS | *HM*

Providing well-documented code was considered vital to Hivemind from the start of the project. As such, the use of Doxygen was deployed to handle code documentation. Doxygen enabled the direct writing of code documentation into source code through code comments. This simplifies the process of keeping code documentation synchronized with updates to the codebase. The Doxygen comments are compiled by Doxygen to HTML, creating an interactive website for the code documentation with separate pages for each class and file. A requirement was set for the Hivemind team to document each class thoroughly to make it as easy as possible for future developers to dive into the codebase.

6.2.7 Continous integration & continuous deployment

RS | *HM*

In modern development there is a large focus on Continuous integration & Continuous delivery/deployment (CI/CD)[83]. Continuous integration relates to continuously building, testing and merging features during development to keep the central repository up to date and stable. Continuous delivery/deployment both relate to the deployment of the integrated project in production. They differ in that continuous deployment is generally an automated process, whereas continuous delivery required more manual work[84].

In terms of CI, strict rules were followed in terms of which branches to use for development. Hivemind operated with two persistent branches: The *Main* branch, and the *Development* branch. The main branch is never touched directly, but is rather merged with the development branch when the current state of the development branch has been thoroughly tested and is considered stable. The development branch is also not touched directly. Instead, when a team member is developing a specific feature, they will branch out from the development branch in to a *feature branch*. They can work on this feature branch until it is considered at least partially implemented. A pull request is then made to merge it into the development branch. A reviewer reviews the pull request, and upon approval the merge is completed and any merge conflicts handled. A similar process is followed for implementing updates to existing features as well as bug-fixes. The branching process can be seen in fig. 8.

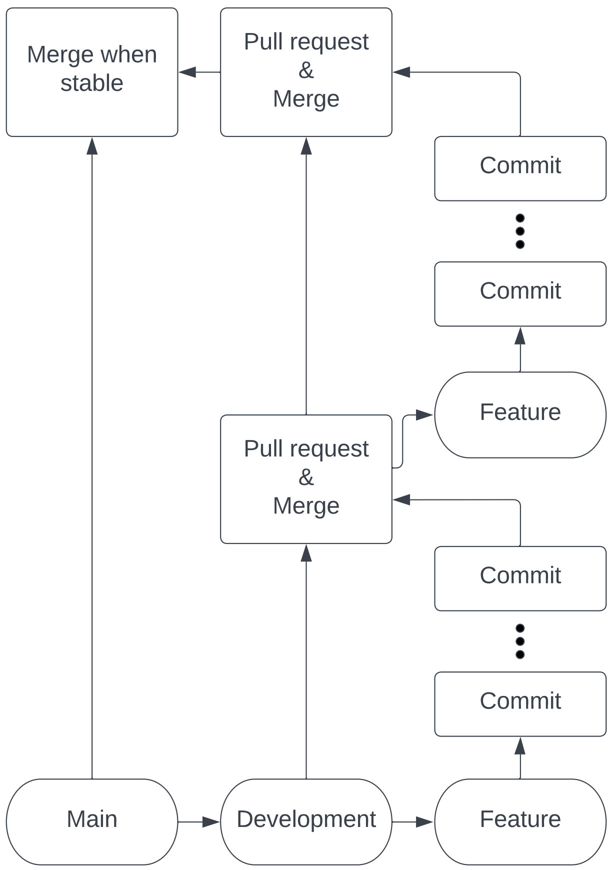


Figure 8: Git branching and merging

Following these strict rules minimizes the amount of merge conflicts on a local level. Optimally, any merge conflicts will be isolated in Azure meaning they will only need to be handled once by the appropriate reviewer.

In terms of continuous deployment, an Azure Pipeline was created to automatically build and publish code documentation online. It is set up so that anytime anytime the main branch of Hivemind is updated, it uses Doxygen to compile the code documentation and publish it at at https://itfag.usn.no/grupper/D01-23/docs. The currently most up-to-date code documentation has also been attached as appendix O. The automatic generation of code documentation, as well as the automatic publishing, ensures that the publicly available code documentation is always up to date with the stable release of Hivemind.

6.3 Verification

HM | *HMM*

To ensure that the system operates as intended, it is important to continuously verify that the software can meet given requirements during the development process. Requirements could include the response time of a function to be low, or precision of map data down to a certain amount of meters. These requirements must be possible to verify.

6.3.1 Methods for testing

HM | *HMM*

When developing a system or deriving requirements for software, any requirements set must also be verifiable. There are 4 main ways to verify a system[85]:

Inspection is examining the system and verifying that functionality is present. In a software system inspection can be performed by looking at the code and verifying that the software has the necessary inputs and functions that are required for the system to work.

Demonstration is verifying the system through manipulation. This is done by verifying that the expected results are acquired when the system is used as intended. In software, demonstration can be done by clicking on a button and checking if the system responds according to expectations.

Testing is verifying that the system operates as intended through using a predefined set of data and inputs, as well as knowing the expected output from the system when using those data and inputs. This type of verification is possible to automate.

Analysis is the final method used to verify a system. This is done by creating models of the system, using equipment to test parts of the system if possible or calculations, if there is a complex function or algorithm in the system.

6.3.2 Unit testing

HM | *HMM*

In testing Hivemind, the principle of unit testing was explored, which entails independent verification of the different components of the software. A major benefit of following unit testing is that this simplifies automated testing through the use of for example GoogleTest and Azure Pipelines. Although a unit test is usually easy to automate, this does not mean all unit tests need to be automated. The project also utilized unit testing for the GUI, through interaction of individual components to verify that these operated as intended. Making use of unit testing allowed the project to have an easier time debugging. When testing only a small part of the software, it is easy to identify where bugs exist than if the entire software was tested at the same time.[86]

6.3.3 Documentation of verification

HM | *HMM*

To document the verification process, a table that contains all the necessary information was used(tab. 4).

Index	The test identification (T.1.1)
Approved by	Name of the person who approved the test
Done by	Name of the person who performed the test
Methods	List of the method that were used to perform the test
Prerequisites	What has to be in place to be able to perform the test
Data	List of any example data that were used to perform the test if you used any example data
Description	Description of the test as well as a step by step guide on how you performed the test
Success criteria	What are the criteria for the test to be successful
Failed test	If the test fails you need to provide a description of the error so that other people can reproduce the same error at a later date

Table 4: Template for verification

A solid foundation in terms of software development process enables the development of a stable and flexible final product. The achievement of this can be done with the help of good tools. The next section of this report will detail the development of the software model, the final crucial step to any software development process before any coding can begin.

7 Proposing a conceptual software model

This section handles how Hivemind's stable conceptual software model was derived. It first describes the use cases that were developed from the requirements of the client. Then, a generic architecture is presented which encapsulates the common functions of most route planning software. This generic architecture is then decomposed into software components and their interfaces, which helps guide the code implementation of Hivemind. Later in this report, the term "agent" will be used to refer to a specific instance of an Unmanned Aerial Vehicle within the software.

7.1 Use cases

AM | *RS*

Use case diagrams are a helpful tool in software development, as they allow stakeholders to visualize and understand how the system will be used in the real world. By depicting the actors and their interactions with the system, use case diagrams provide a high-level overview of the system's functionality and role in supporting the user's goals. Overall, use case diagrams are an essential part of the requirements gathering and design process, as they ensure that the system meets the needs of it's users.

Using the requirements, the use cases for Hivemind is:

- Design settings; processing the necessary data from the user for a scenario.
- Generate scenario; generating a scenario at a specific moment where decisions are made on how agent should move.
- Load scenario; loading an existing scenario the user has previously saved.

Using these use cases, a use case diagram was created, depicted in fig. 9. This use case diagram is very generic by design as it makes it possible to use it for route planning for any type of agent, not only for UAVs. It illustrates the main functionalities of software for route planning for UAVs.

The use case diagram does not provide explicit information about the specific data involved in each use case, allowing for potential data sharing among use cases if needed. The content of the use cases reveals what is needed to develop software applications and how data is accessed and updated.

By analyzing the content of the use cases, one can observe the composition between them. *Generate scenario* is the most complex use case that performs the most significant computations, while *Load Scenario* depends on the reliability of the generated Scenario. *Design Settings* relies on gathering data through human interaction, although the model does not explicitly define the interactions involved. The value of this use case diagram is in the separation of these three significantly different functionalities, and the software should not mix them together. The separation between the use cases is based on important principles in software design. Each use case has different access to data and performs distinct tasks related to data entry and updating.

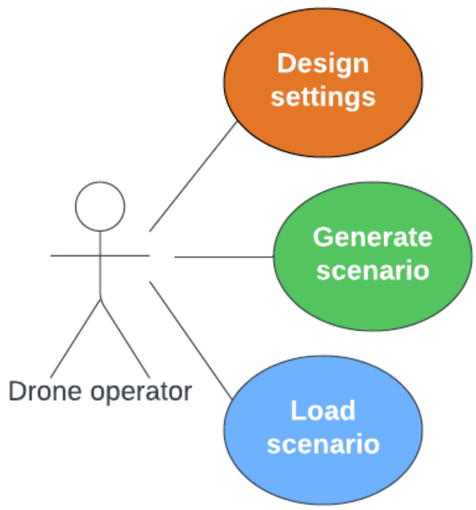


Figure 9: Use case diagram.

This is important to ensure proper authentication, data access, and role distribution within the software application.

7.2 Generic software architectural model

AM | *RS*

The architecture utilized for Hivemind is a three-layered software architecture. A layered software architecture is an architecture that organizes a system into hierarchical layers, where each layer represents a specific responsibility and offers a particular functionality. The three layers of the architecture of Hivemind are:

- User interface layer (top layer)
- Computational layer (middle layer)
- Data layer (bottom layer)

The user interface layer handles the user interface and receives user input. The user interface layer in Hivemind is designed to present the functionality of the system to the user in a clear and intuitive way. It helps to separate the presentation logic from the computational logic, which can improve the maintainability and flexibility of the system.

The computational layer is the middle layer in the three layer architecture. It is responsible for processing data received from the user interface layer and the data layer. This layer contains the core logic and algorithms that enable Hivemind to perform its functionality. It is important to note that the boxes within the computational layer should not communicate with each other directly. By avoiding direct communication between boxes, Hivemind can be more easily scaled, adapted, and maintained over time.

The data layer is responsible for managing the storage and retrieval of data used by the Hivemind system. This includes the handling of data from various sources such as user input, external sources, and data generated by the system itself.

This type of layered architecture activates the main criteria for an architecture for Hivemind. The criteria for an appropriate architectural pattern for this project are, therefore (in no particular order):

- Scalability
- Clarity
- Adaptability
- Stability

Firstly, scalability. By utilizing a layered architecture, specific layers could be scaled independently of one another, enabling the architecture to handle increasing amounts of traic or data without impacting the performance of other layers [87]. By organizing the application into distinct layers, each with a well-defined responsibility and interaction with other layers, the architecture could be easily understood and maintained by both current and future developers, making the architecture clear.

By isolating errors or failures to specific layers, alternative implementations could be swapped in without impacting the remainder of the application, providing flexibility in responding to changing requirements and technologies. For example, there is a clear separation between the User Interface (UI), computational layer, and data layer. In that case, updating or changing the UI layer without modifying the other layers is possible.

This adaptability makes responding to changing requirements or technologies easier without redesigning the entire application.

To ensure stability, retrieving data from an existing database when the project expands without modifying it is possible. This made it possible to adapt to changes in the database schema and recover from errors or failures without affecting the rest of the architecture.

A layered architecture is one that meets the criteria, scalability, clarity, adaptability, and stability we have set for the architecture. The architecture enabled the development of a flexible, scalable, and maintainable architecture that could adapt to changing requirements and support the project's long-term goals.

One of the primary benefits of this type of architecture is that it promotes a modular and scalable system design. Having a layered architecture enables different layers to be developed and tested independently. This means that changes made to one layer will not affect the others, reducing the risks associated with software development. For example, if a developer needs to make changes to the user interface, they can do so without worrying about disrupting the underlying code that drives the application's functionality.

7.2.1 Software architecture of Hivemind

AM | *RS*

Fig. 10 is a generic model of the software architecture to Hivemind. The components in this architecture are illustrated as coloured boxes and cylinders into appropriate layers.

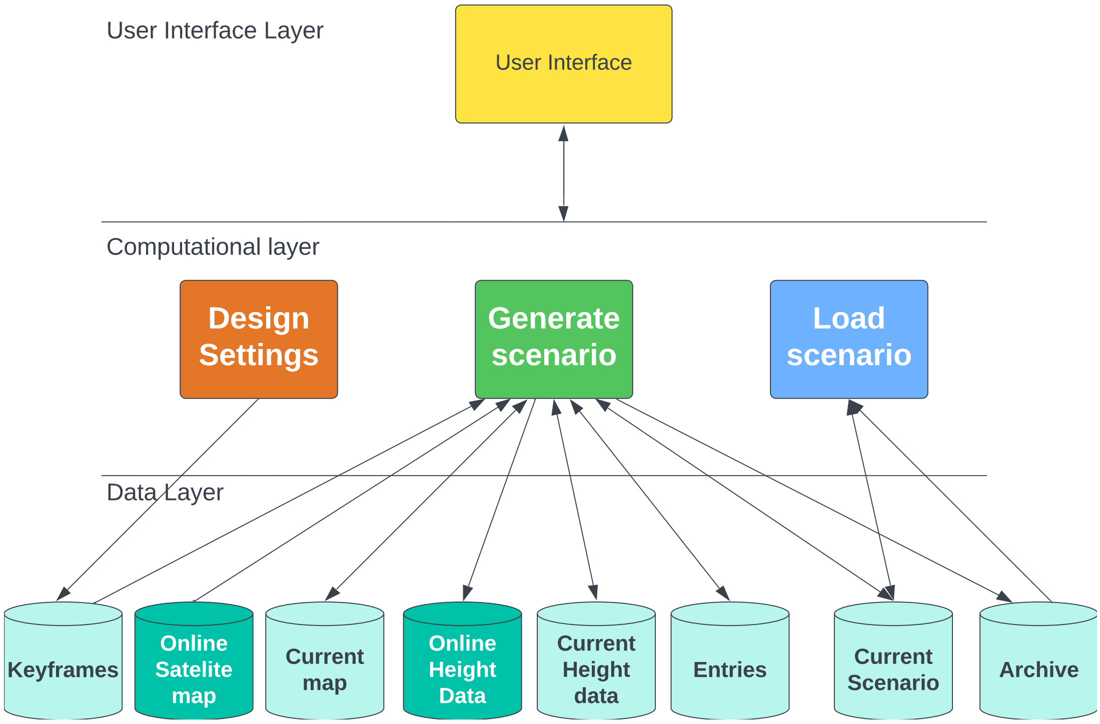


Figure 10: Logical architecture.

The software abstractions, presented as software components in fig. 10 are derived from the use case model in fig. 9 and user requirements defined in section 3.2 of this project. The functionality defined in the use case model is visible in fig. 10 through the imaginative vertical lines which separate *Design Settings* (amber), *Generate Scenario* (green) and *Load scenario* (blue).

The choice of data sources in the data layer and their usage in the computational layer indicate where data is being updated, read/retrieved, or entered. The arrows between computational and data components illustrate how the data is manipulated and which computer programs are responsible for it. Two-way arrows indicate that the same computing program both updates and retrieves the data.

7.3 Decomposing the software architecture

AM | *RS*

The design architecture focuses on the physical implementation of the system and shows the technical details of how the system is built. It provides guidelines for developers on how to implement the system in a way that is consistent. This helps to ensure that the system is built in a way that is scalable, clear, adaptable and stable.

In addition, the design architecture enables developers to work more efficiently by breaking down the system into smaller software component and defining their interactions and interfaces. This promotes modularity and reusability, which will save time and effort in the development process.

The design architecture is developed after the generic architecture in fig. 10 has been created, using the software component and interactions defined in the logical architecture. It provides a more detailed view of the system's physical implementation, specifying technical details. It enables the system to be broken down into smaller software component, which can be developed and tested independently.

In the design architecture, the software component are placed in a way that there is no direct communication between software component that belong to different use cases within the computational layer. This is important because it helps to ensure that the software component are loosely coupled, and changes to one component do not affect the others in other use cases. By making the software component independent, it becomes easy to locate the software component in the architecture within the code.

There are multiple solutions in this area that can address the same problem. Fig. 11 represents the outcome of several iterations exploring different software models. The generic software architecture model was selected as the most suitable for the project and the design architecture was developed from this. You can find the various iterations of the software architecture in Appendix H.

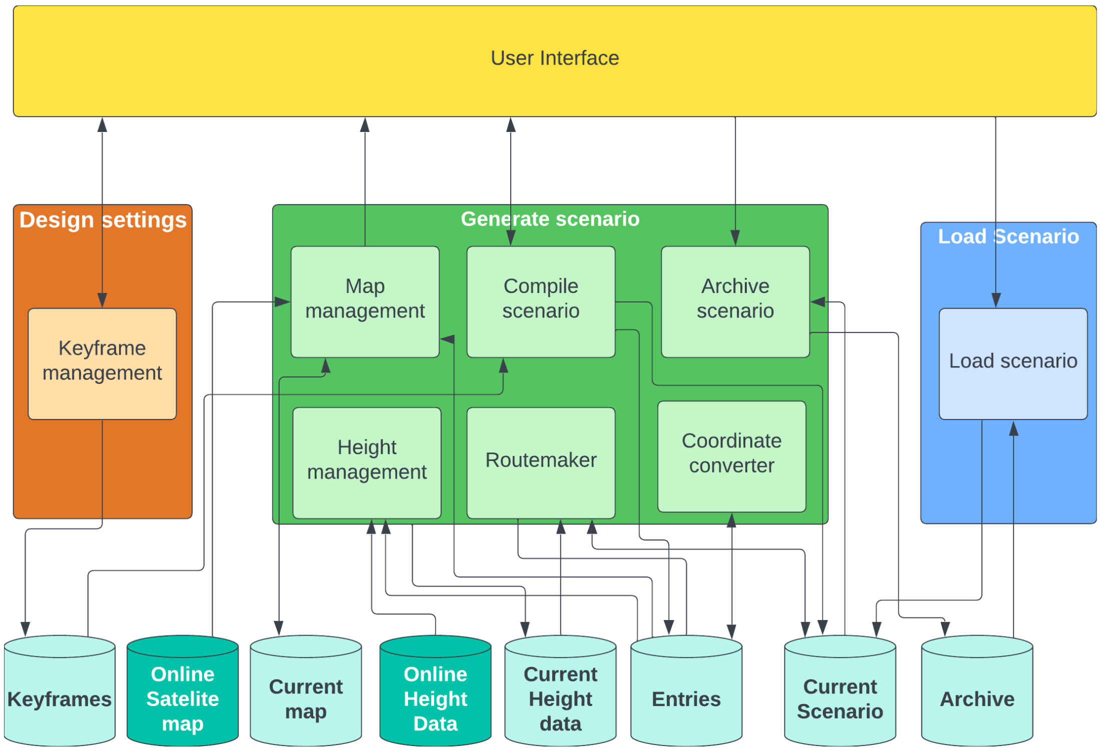


Figure 11: Design architecture.

7.3.1 Software component diagram & interfaces

NH | *AM*

Hivemind can be broken down into various software component that work together to provide the desired functionality. Breaking down these software component and displaying their respective inputs and outputs can simplify the coding process and enhance the comprehensibility of the software. Software components were distilled from the requirements. The information presented in tab. 5 indicates which software component corresponds to the requirements. This highlights that every requirement is addressed by at least one component and that each component is designed to fulfill one or more requirements. These software components have changed names during the course of the project, but their functionality has remained largely the same.

User Interface	R.1.1, R.1.2, R.6.1, R.6.5, R.6.8, R.6.9, R.6.11, R.6.12, R.11.3, R.11.4, R.11.5, R.11.7, R.3.3, R.6.16
Keyframe Manager	R.6.9, R.11.6, R.11.8, R.6.10
Height manager	R.3.5, R.3.3, R.3.4, R.6.8
Routemaker	R.11.9, R.11.10, R.11.11, R.6.8, R.6.14
Compile scenario	R.6.14, R.6.15, R.11.2
Save scenario	R.6.2
Map manager	R.2.1, R.6.8
Coordinate converter	R.8.1
Load scenario	R.6.5, R.6.6

Table 5: Software component

Integration Definition for Process Modelling (IDEF0) was used to represent the software components and their interfaces, as this type of representation allows for a hierarchical decomposition of a system. This allows it represent a complex system in an organized and structured way[88]. These diagrams aided in the comprehensive understanding of the inputs, outputs, and destinations associated with each component. Moreover, they have allowed for the creation of an expanded hierarchy, accommodating varying levels of detail. This hierarchical approach proves particularly advantageous in scenarios where the software expands beyond the MVP stage and additional software component are introduced.

IDEF0 works by representing each component as a box and defining its inputs, outputs, controls and mechanisms (fig. 12). It is possible to zoom in on one software component and view a software component by its subcomponents, which further enhances the detail by which software components can be illustrated.

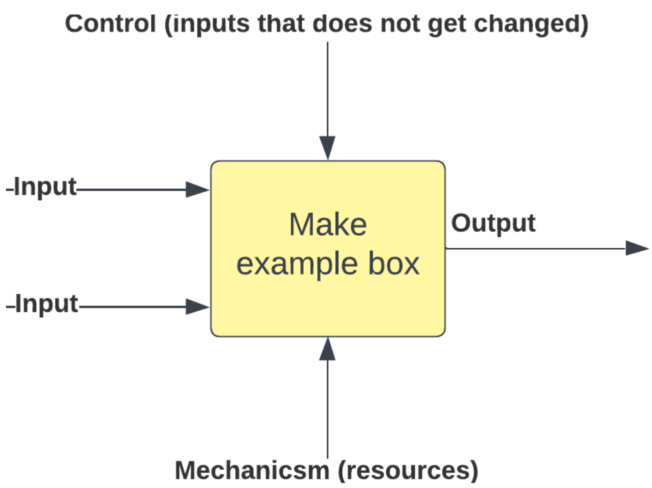


Figure 12: IDEF0: Example

This is an example of the IDEF0 diagram for the generate scenario component (fig. 13). This diagram shows the various software subcomponents within the "generate scenario" software component and demonstrates how they interface with each other.

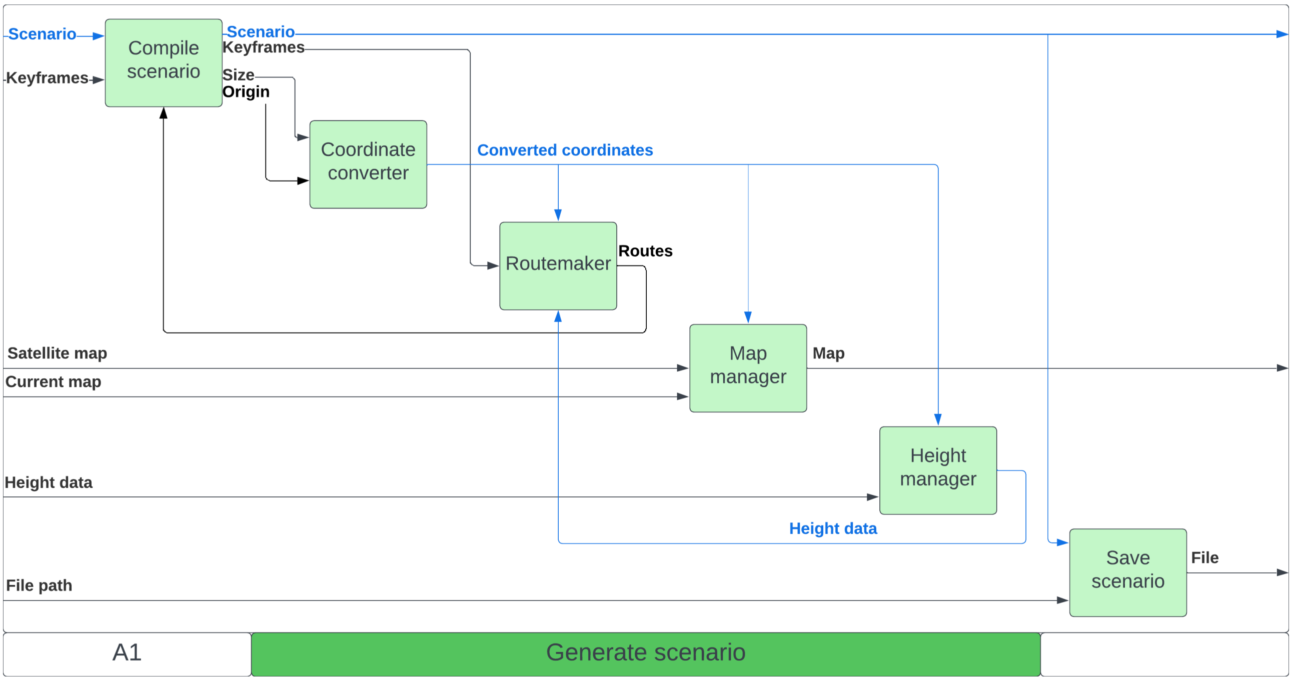


Figure 13: IDEF0: Generate scenario

After a stable software architecture had been created and decomposed into well-defined software components, the actual creation of the Hivemind software could commence. A good deal of time was spent on the architecture, to make sure it fulfilled the requirements of scalability, clarity, adaptability and stability. The next section will illustrate how this software model was implemented in various software components to create Hivemind.

8 Implementation

After creating the software model and defining components and their interfaces, the physical implementation of Hivemind could commence. Note while reading that Hivemind's team continued to operate following an Agile methodology - although the software model acted as a launching pad for further development activities, the realities of implementing and integrating the software often lead to the discovery of new opportunities or previously-unknown limitations. Accordingly, as the actual implementation of Hivemind continued and diverged from the original software model, the software model itself was also updated to reflect the new changes. As a reminder, the evolution of the software architecture can be seen in appendix H.

This section will present the concrete implementation of each component of Hivemind. Most of these are directly translated from the diagram in fig. 11, with the exception of the Serializer. This component encapsulates both the Archive and Load scenario functionalities from the architecture. A brief introduction of the libraries used to realize the software architecture is also presented.

8.1 Technology-specific software components

8.1.1 Geospatial Data Abstraction Library (GDAL)

HMM | *NH*

The GDAL (Geospatial Data Abstraction Library) is free open source software to use with geospatial data formats, including GeoTIFFs [89]. In this project, we make use of GDAL to extract height data over an area too large for individual geographic point Application Programming Interface (API) requests. This height data is necessary for the proper and safe functioning of Hivemind's route planning function. To understand the component that extracts this height data, it is therefore important to understand what specifically these GDAL functions do.
However, first, it is necessary to understand the GeoTIFF format. This format is an extension of the TIFF format [90], which is a type of layered image where each layer contains different kind of information. In the case of GeoTIFFs, this contains a large number of geographic image data used for spatial referencing, including image data, height data and much more. Numerical tags are used to indicate what information each GeoTIFF contains.

The data laid on top of each other in structures called raster bands [91]. For a GeoTIFF containing digital elevation data, the height data we are interested in is according to the GeoTIFF convention found in band 1. The current maintainers of the GeoTIFF format is the Open Geospatial Consortium [92].

In the case of the height data accessed for Hivemind, each GeoTIFF is organized into cells of a given pixel size. The GeoTIFF sample used by default in Hivemind is downloaded from Kartverket [93], where it is possible to choose the resolution (i.e. pixel dimensions) of a dataset before downloading. For ease of calculation and accuracy, the dataset has 1m x 1m resolution.

The dataset is then composed of rows and columns of these 1m x 1m cells, called a grid. The dataset also uses Universal Transverse Mercator (UTM) 33N coordinates (EPSG:25833), which is a projection that covers the entirety of Norway. These coordinates are already in meters, which means that each row/column of a dataset will cover each easting and northing coordinate of a GeoTIFF subset. Finally, each cell (whose geographic UTM coordinate is defined by the easting and northing given by the intersection of a row and column) will contain one height value. This is visualised in figure 14.

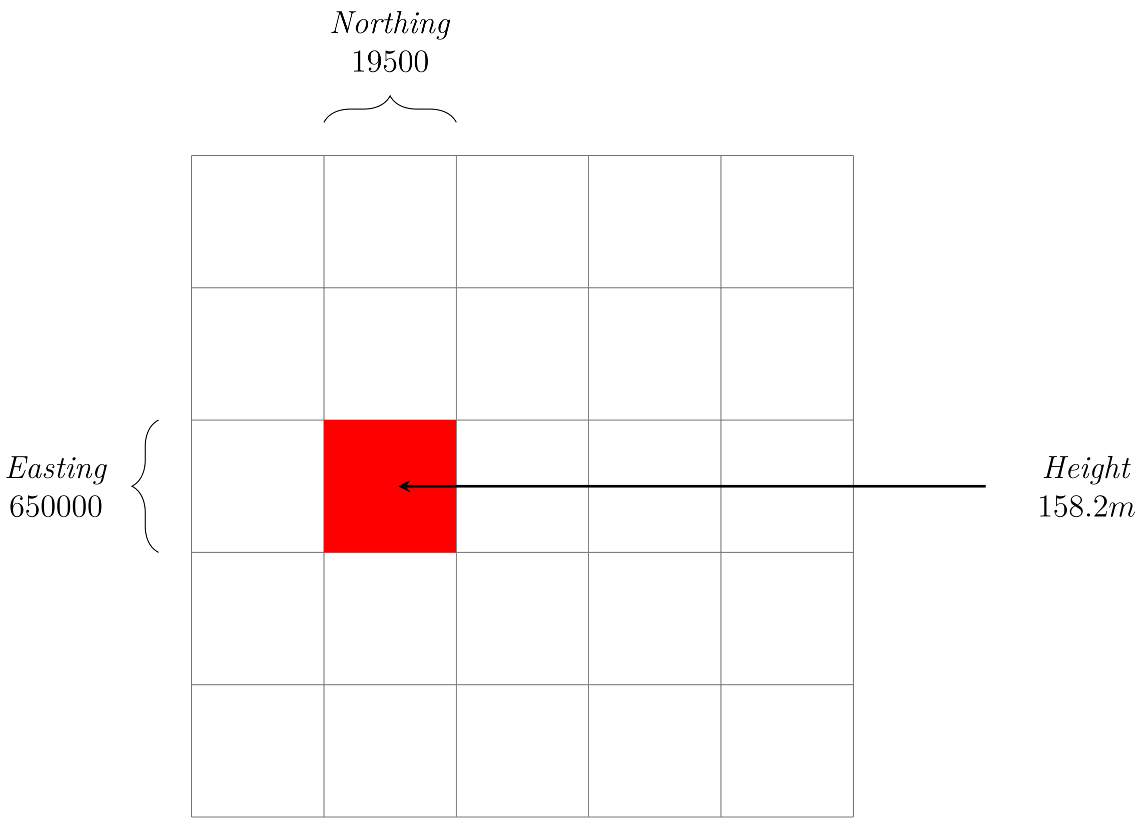


Figure 14: Height data is organized in a two-dimensional grid, where each cell has one height value. The east/west coordinates are defined by the two dimensions in the grid.

When opening a GeoTIFF using GDAL, the first function that must be run is GDALAllRegister(). Each file format GDAL is able to operate on has its own driver which contains information and specifications about this format [94]. Running this method therefore enables the rest of the code to properly read and operate on the input file. After opening the dataset containing height data, the next thing that needs to be done is to retrieve the part of the data that contains height data. In the default dataset (and in general), this is found on raster band 1. The function to retrieve this is GetRasterBand(1).

After preparing the height data, GetGeoTransform() is run on the dataset to transform it from the row/column referenced system wherein the height is located to using geographic coordinates. The 6 coefficients of the resulting geotransform is as follows [95]:

- `[0]`: x-coordinate of the upper-left corner of the upper-left pixel.
- `[1]`: w-e pixel resolution / pixel width.
- `[2]`: row rotation (typically zero).

- `[3]`: y-coordinate of the upper-left corner of the upper-left pixel.
- `[4]`: column rotation (typically zero).
- `[5]`: n-s pixel resolution / pixel height (negative value for a north-up image).

Finally, the RasterIO(flag to use read or write operations, x offset, y offset, x size, y size, output data buffer, size of data buffer in x direction, size of data buffer in y direction, output data type, spacing between data, extra arguments) function can be used to iterate through the data from a chosen offset to extract the individual heights. From the function call, it is apparent that a lot of customization is possible to change which data is read.

8.1.2 Geographiclib

AM | *RS*

GeographicLib is an open-source C++ library developed and maintained by Charles F. F. Karney that provides the functionality to calculate the exact geographical coordinate, distances, and directions between points on the earth's surface. The library supports several geodetic models i.e WGS84[96]. It also provides support for coordinate systems such as UTM and MGRS.

Conversion error: Every time a coordinate is converted from one coordinate system to another, there will be a certain error, especially if the reference for the coordinate systems is different. For Hivemind, in order to have as few errors as possible, Geographiclib is used to convert coordinates. Karney has shown that his method of converting between coordinate systems results in negligible error (approximately $5nm$ within $3900km$ of the central meridian)[97].

GeographicLib::LocalCartesian Class Reference: The class *LocalCartesian* converts geographical coordinates to a local cartesian coordinate. The constructor to the class takes in a geographical coordinate that is the reference point to the local cartesian coordinate system when it converts, meaning the specified geographical coordinate is used as the Cartesian space's origin. The *Reset* method resets the origin to a new geographical coordinate.

```
void GeographicLib::LocalCartesian::Reset(double lat0, double lon0,
                                          double h0 = 0)
```

To convert from a geographical coordinate to a cartesian coordinate, the *Forward* method is used. This takes in a geographical coordinate and returns a coordinate in Cartesian space relative to the origin.

```
void GeographicLib::LocalCartesian::Forward(double lat, double lon, double h,
                                            double & x, double & y, double & z)
```

To convert from a cartesian coordinate to a geographical coordinate, the method *Reverse* is used. This takes in a cartesian coordinate and returns the corresponding geographical coordinate.

```
void GeographicLib::LocalCartesian::Reverse(double x, double y, double z,
                                            double & lat, double & lon, double & h)
```

GeographicLib::UTMUPS Class Reference: The class UTMUPS converts between geographical coordinates and UTM coordinates. The class' constructor takes in a geographical coordinate and defines the zone for the UTM coordinate. However, in some cases, such as when working in a specific UTM zone, it may be necessary to specify the UTM zone manually. This is particularly relevant in cases where the default UTM zone does not correspond to the actual area being worked on, as in the case of working in UTM 33N, while the default is UTM 32N.

The method *Reverse* converts from UTM coordinates to geographical coordinates by taking a UTM coordinate and giving the values to lat and lon parameters. When the specified UTM coordinate is at the north side of the equator, *northp* is true; when the UTM coordinate is at the south side of the equator, *northp* is false.

```
void GeographicLib::UTMUPS::Reverse(int zone, bool northp,
                                    double x, double y, double & lat, double & lon,
                                    double & gamma, double & k)
```

The method *Forward* converts from geographical to UTM coordinates by taking a geographical coordinate and giving the value to *zone*, *northp*, *x* and *y*.

```
static void GeographicLib::UTMUPS::Forward(double lat, double lon,
                                           int & zone, bool & northp,
                                           double & x, double & y,
                                           int setzone = -1)
```

8.1.3 Qt

RS | *HM*

Qt is a cross-platform, multi-language set of libraries and tools for creating Graphical User Interfaces (GUI) and applications. In addition to tools for building graphical user interfaces, Qt provides networking functionality, simple multi-threading and Inter-process communication (IPC) interfaces and more. At the time of writing, the latest major version of Qt is Qt6, and this is the version used in Hivemind. Qt was chosen for its GUI functionality, but some of it's networking tools were also found to be useful for some of the components.

There are several ways of building a GUI with Qt, such as interactively designing it using QDesigner and automatically generating the code. The method employed in Hivemind, however, is manually creating the GUI programmatically directly in our codebase. Although this method is a more tedious than using QDesigner, it provides greater control in structuring the code. QDesigner also tends to create more bloated code than necessary, so manual coding removes some overhead.

Qt's GUIs are generally structured as a tree of widgets, with a root widget on top representing the window. A widget may be anything from a container, a button or an image. A widget has a variable amount of child widgets which again may contain even more widgets. This tree-like datastructure makes for easy creation of new, reusable widgets that can be moved and positioned as we want.

Qt provides a lot of pre-made widgets out of the box, such as push buttons, text boxes and dialog boxes. All widgets are classes inheriting from *QWidget* or a sub-class of *QWidget*. This means that creating custom widgets is as simple as inheriting from *QWidget* or a sub-class of it ourselves, and overriding any methods we need, such as rendering and event methods.

Qt provides a simple way of connecting functionality to triggers such as a click on a button, or an updated combo-box. This works by defining so-called *signals*, which are *emitted* from widgets, to *slots*, which are methods defined in other widgets. This means that opening a dialog box when a user clicks on a button is as simple as connecting the button's *clicked* signal, with the dialog box's *open* slot.

8.1.4 GoogleTest

HM | *HMM*

GoogleTest is a framework for testing in C++ code. It makes testing easier to perform and debug by giving the tester as much feedback as possible when a test fails. This is possible because testing is set to run on different objects, ensuring that each test can be run every time and not be dependent on other tests being successful. This means that even if a test fails, GoogleTest will still run all the other tests instead of stopping at the test which failed.[98]

8.1.5 Bootstrap

NH | *AM*

Bootstrap is a free and open-source framework used in web development. It provides ready-to-use components, CSS and HTML templates, JavaScript plugins, and other tools that simplify the web development process [99]. A key feature of Bootstrap is its responsive grid system, which ensures proper layout on various screen sizes. It also offers compatibility with modern web browsers.

8.1.6 Additional Explored Libraries

AM | *RS*

Additionaly, a number of other libraries were explored that ultimately remained unused in Hivemind. Some of these libraries were found unsuitable for Hivemind, while others had limited resources or outdated content.

QGIS is a free and open-source software for Geographical Information Science (GIS) [100]. It is a software used to visualize geographic information in an intuitive and understandable way. It has many different tools and functions that can be used to make customized maps, analyze data, and create visualizations and presentations.

To display a map in QGIS it is possible to use the Web Map Service (WMS). By using the URL for the WMS service form Geonorge it is possible to generate a map and it will be shown in the QGIS GUI. QGIS API allows integration of the software with other applications and user interfaces, enabling the creation of custom GUIs and tools tailored to specific needs[101]. This feature was relevant for Hivemind, as Hivemind required a customized interface to display the map and route planning tools. QGIS was discarded as an option for the dynamic map visualisation, because it may be too complex to use the software for a limited purpose within the available time. QGIS is a large and comprehensive software that may take time to learn and customize for specific needs. For a smaller feature, it may be more appropriate to choose a more specialized software or develop a smaller customized solution.

Robot Operating System (ROS) is an open-source framework for building robotic applications [102]. It provides a collection of software libraries and tools that enable developers to create robotic systems. Robot Operating System is designed to be modular, and it provides a messaging system for communication between different parts of a robotic system and built-in tools for visualization. ROS has support for multiple programming languages like Python and C++. Although Hivemind will most likely make use of this once its functionality is extended to include real-time communications with UAVs, this was not within the scope for this project. The library was nevertheless explored so the basic components of Hivemind could be designed with future use of ROS in mind.

ROS Visualization (Rviz) is a 3D visualization tool that is part of the ROS software[103] suite. It allows users to display and interact with various data types in a virtual environment, including point clouds, maps, and sensor data. Rviz provides visualization options, including 3D models, grid maps, and camera images.

Librviz is a library that provides access to the functionality of the Rviz visualization tool within a user's own application. By linking to the librviz library, developers can incorporate the rich 3D visualization capabilities of Rviz into their own GUI. The use of librviz has the potential to greatly enhance the capabilities of custom robot control and monitoring systems by leveraging the powerful visualization features of Rviz.

Point Cloud Libraries (PCL) is an open-source library for working with 3D point cloud data[104]. This is a library that can convert the height data from the height manager to a point cloud. Issues were encountered linking it with PCL as it depends on Qt5[105]. Hivemind itself depends on Qt6, so as a result several linking collisions occurred when attempting to link with PCL.

To address this issue, the compilation of PCL from source was attempted. Upon observing the source code, it was discovered that PCL has some compile flags that can be set to link with Qt6, rather than Qt5. However, even when setting these flags, Hivemind failed to properly link PCL with Qt6.

8.2 Coordinate Converter

AM | *RS*

A coordinate converter is an essential component in any system dealing with different coordinate systems. Hivemind uses a unified cartesian coordinate system to represent the physical space in which the agents operate. However, the input data comes in various coordinate systems, such as geographical coordinates and UTM coordinates. Therefore, a coordinate converter is necessary to convert these different types of coordinates into a unified system that the rest of the system can use. This ensures consistency and accuracy in the spatial representation of the environment and the agents' movements.

The coordinate converter in Hivemind is implemented as a software component that performs specific computations necessary for the system's overall functionality. The coordinate converter is designed as a singleton object, meaning it is only instantiated once and can be accessed globally in other parts of the code.

The coordinate converter in Hivemind converts various types of coordinates into a unified Cartesian coordinate system used by the system using *GeographicLib*. The coordinate converter is a versatile tool that can perform several types of conversions between different coordinate systems. It can convert from geographical coordinates (longitude and latitude) to the local Cartesian coordinate system used by Hivemind, as well as from UTM coordinates to the same local Cartesian system. Additionally, it can perform conversions from UTM to geographical coordinates and from geographical coordinates to UTM. It also maintains the context of the conversions and utilizes the WGS84 geodetic model. One additional feature of the coordinate converter is its ability to perform conversions between symmetric and asymmetric cartesian coordinate systems.

8.2.1 Symmetric and Asymmetric converting

AM | *RS*

One feature of the coordinate converter is its ability to perform conversions between symmetric and asymmetric cartesian coordinate systems. Geographiclib makes a symmetric Cartesian system. Every time Hivemind starts up, the program starts with a map of a size of $3x3km$. This is the size of the coordinate system. To convert a coordinate from a symmetric system to an asymmetric system, the converter uses this calculation:

$$\text{AsymmetricX} = \text{symmetricX} + \frac{\text{size}}{2} \tag{1}$$

$$\text{AsymmetricY} = -\text{symmetricY} + \frac{\text{size}}{2} \tag{2}$$

This calculation moves the origin from the middle of the Cartesian system to the upper left corner for the Cartesian system.

To convert a coordinate from an asymmetric system to a symmetric system, the converter uses this calculation:

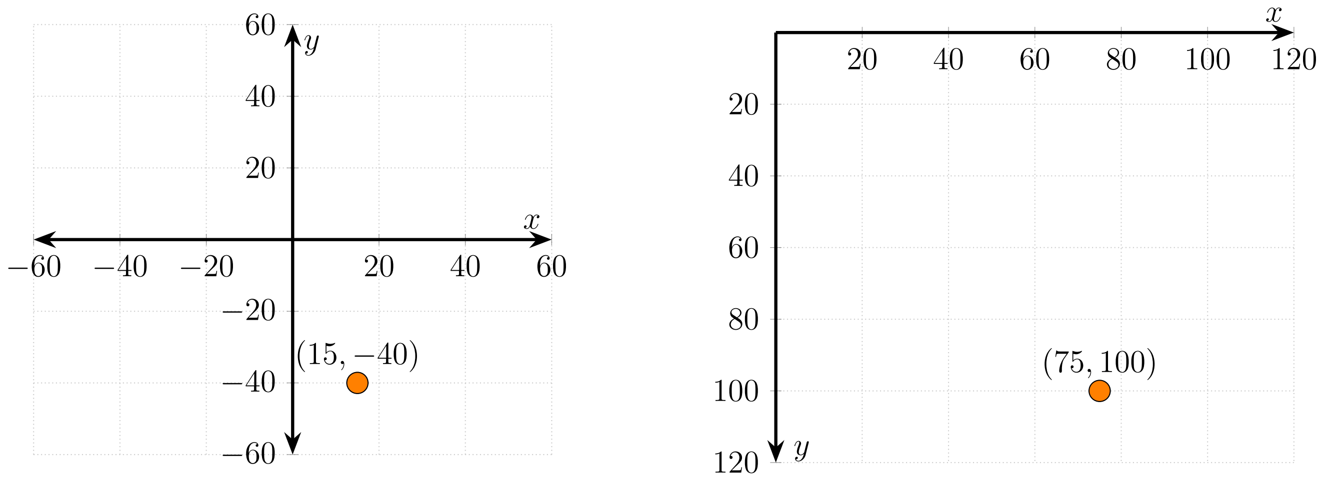
$$\text{SymmetricX} = \text{AsymmetricX}\frac{\text{size}}{2} \tag{3}$$

$$\text{SymmetricY} = \text{AsymmetricY} + \frac{\text{size}}{2} \tag{4}$$

Example: Fig. 15 is a symmetric coordinate system with a size of 120. It has a point at $(-40, 40)$. To convert this point to an asymmetric coordinate system, the following calculation is done:

$$\text{AsymmetricX} = \text{symmetricX} + \frac{\text{size}}{2} = 15 + \frac{120}{2} = \underline{\underline{75}} \tag{5}$$

$$\text{AsymmetricY} = -\text{symmetricY} + \frac{\text{size}}{2} = -(-40) + \frac{120}{2} = \underline{\underline{100}} \tag{6}$$



(a) Symmetric cartesian coordinate system

(b) Asymmetric cartesian coordinate system

Figure 15: cartesian coordinate systems

8.3 Height Management

8.3.1 Preliminary work

HMM | *NH*

When making a route planning software for unmanned aerial vehicles, the route making algorithm will need to take into account the height at each coordinate it considers passing through. In order to create routes wherein the drones will not collide with buildings, it is therefore crucial that the route planner can access the height data of all coordinates visited. The HeightMap class fulfills this purpose through the GetVertex function. An earlier component that fulfilled this function was the HeightData class, which could be used to send individual Get request over the network to an online API supplied by Kartverket [106] . For a small algorithm that only has to query a handful of points, this would have been a simple and elegant solution - requesting one point might take around 0.35s, based on informal tests of the API, see fig. 16. It was quickly discovered, however, that this might not be sufficient in implementing more advanced features such as route planning for multiple drones, 3D visualization and longer routes. To illustrate this, let us consider 3D visualization. To construct a height map, the system must gather the heights of all points within a certain area. For the minimum viable product, this is 500 x 500 meters (250 000 points, with a resolution of 1 meters). Simply populating this small height map would require approximately 24.5 hours (See eq. 7), based on the time taken to perform one request seen in fig. 16.

$$0.35s * 250000\text{points} * \frac{1}{60 * 60} \approx 24.5\text{hours} \tag{7}$$

For a slower HTTP request speed of 0.5 seconds or even 1 seconds, which could be more realistic during in-field operations of the route planning software, this population would take somewhere between 35 and 70 hours. The API does support querying 50 points at a time, but

	Time	Source	Destination	Proto	Length	Info
10	2023-04-17 20:10:32.691585	10.0.0.6	148.122.164.253	DNS	74	Standard query 0xa72f A ws.geonorge.no
11	2023-04-17 20:10:32.691675	10.0.0.6	148.122.164.253	DNS	74	Standard query 0xf710 AAAA ws.geonorge.no
12	2023-04-17 20:10:32.691748	10.0.0.6	148.122.164.253	DNS	74	Standard query 0x7aa8 HTTPS ws.geonorge.no
13	2023-04-17 20:10:32.694368	148.122.164.2…	10.0.0.6	DNS	136	Standard query response 0xf710 AAAA ws.geonorge.no SOA ns1.statkart.no
14	2023-04-17 20:10:32.695256	148.122.164.2…	10.0.0.6	DNS	90	Standard query response 0xa72f A ws.geonorge.no A 159.162.23.38
15	2023-04-17 20:10:32.695383	148.122.164.2…	10.0.0.6	DNS	136	Standard query response 0x7aa8 HTTPS ws.geonorge.no SOA ns1.statkart.no
16	2023-04-17 20:10:32.695527	10.0.0.6	159.162.23.38	TCP	66	63252 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
17	2023-04-17 20:10:32.699870	159.162.23.38	10.0.0.6	TCP	66	443 → 63252 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
18	2023-04-17 20:10:32.699896	10.0.0.6	159.162.23.38	TCP	54	63252 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
19	2023-04-17 20:10:32.699996	10.0.0.6	159.162.23.38	TLS…	571	Client Hello
20	2023-04-17 20:10:32.704291	159.162.23.38	10.0.0.6	TCP	60	443 → 63252 [ACK] Seq=1 Ack=518 Win=30336 Len=0
21	2023-04-17 20:10:32.704406	159.162.23.38	10.0.0.6	TLS…	230	Server Hello, Change Cipher Spec, Encrypted Handshake Message
22	2023-04-17 20:10:32.704512	10.0.0.6	159.162.23.38	TLS…	129	Change Cipher Spec, Encrypted Handshake Message
23	2023-04-17 20:10:32.704603	10.0.0.6	159.162.23.38	TLS…	971	Application Data
24	2023-04-17 20:10:32.708293	159.162.23.38	10.0.0.6	TCP	60	443 → 63252 [ACK] Seq=177 Ack=1510 Win=32128 Len=0
25	2023-04-17 20:10:32.934758	159.162.23.38	10.0.0.6	TLS…	603	Application Data
26	2023-04-17 20:10:32.987882	10.0.0.6	159.162.23.38	TCP	54	63252 → 443 [ACK] Seq=1510 Ack=726 Win=261888 Len=0
27	2023-04-17 20:10:32.996366	10.0.0.6	159.162.23.38	TLS…	875	Application Data
28	2023-04-17 20:10:33.000582	159.162.23.38	10.0.0.6	TLS…	795	Application Data
31	2023-04-17 20:10:33.043743	10.0.0.6	159.162.23.38	TCP	54	63252 → 443 [ACK] Seq=2331 Ack=1467 Win=262656 Len=0

Figure 16: Packet capture showing the speed of a single Get request toward Kartverket's API. The circled timestamps show when the HyperText Transfer Protocol (HTTP) request was sent (top) and when the transfer had completed (bottom).

even if these were received as quickly as a single point is (about 0.3s, based on the test of a single point request toward the API), populating the entire height map would still take around 30 minutes. As a result, it was determined that using Kartverket's own API for requesting height data was simply not a feasible solution.

An alternative and more scalable solution was making use of Kartverket's GeoTIFF files to extract height data. This provided for a flexible solution wherein new height maps could easily be populated by downloading new GeoTIFF files. A great deal of work was unexpectedly necessary to find a suitable library to successfully extract height data from GeoTIFF files programmatically. The first library considered was GDAL (Geospatial Data Abstraction Library) [89], which contains all the tools required for the HeightMap and more. Unfortunately, there is no binary file to install GDAL and all its dependencies. On the official site, for a Windows based operating system, you are presented with the following option:

- Source files to build project using cmake
- Download and installation via Conda
- Download and installation via vcpkg

All three methods were attempted in multiple Windows environments with various amounts of success, with nothing quite successful enough to build Hivemind and program with GDAL in a Windows environment. A good 20+ hours was spent attempting this. In between these attempts, investigations were also made into the TIFF [107] and RasterIO [108] libraries to attempt to open and extract heights from a GeoTIFF file.

TIFF was easy to install and user friendly, but did not work with the GeoTIFF file used for testing. An unconfirmed suspicion is that TIFF is not designed for GeoTIFFs, but simply TIFFs.

RasterIO, similarly, was easy to install but did not result in any successful extraction of height data, though it is important to note that this method is in fact integrated with GDAL

In the end, the final solution was to install and set up GDAL in the virtual machine used for the project. The most difficult aspect of this was now not installing GDAL, but to integrate GDAL and the prerequisite library paths into the cmake file of the Hivemind project. The documentation on GDAL for C++ in general is not very beginner friendly, and in the end, a solution was found through digging through the GDAL source code and seeing how the creators themselves had built their own project using cmake.

After setting up the development environment to run with GDAL, the next and final step was actually coding the HeightMap class. This also included a good deal of research into the GeoTIFF format, trial and error and testing to ensure the correct values were returned.

One thing not necessarily part of the minimum viable product but that was attempted to be added into the class regardless was dynamic download of GeoTIFF files based on the user's input. This turned out to be less straightforward than anticipated. Kartverket has its own API to accommodate WCS (Web Coverage Service) requests [109], which in itself is rather poorly documented but is a method to download whole or parts of GeoTIFF files from the internet

After composing a GET request through trial and error that actually returned data, the project was faced with the challenge that the downloaded file was not in fact a GeoTIFF. Though WCS does include a parameter for specifying type of file downloaded, Kartverket specifies on their brief user instructions page that it is only possible to download GML (Graphical Markup Language) format [110].

The following GET Request successfully returned a file:

`https://wcs.geonorge.no/skwms1/wcs.hoyde-dom-nhm-25833?service=WCS& version=2.0.1&request=GetCoverage&coverageId=NHM_DOM_25833& format=image/geotiff&subset=x(197332,200335)& subset=y(6624844,6627847)&outputCRS=urn:ogc:def:crs:EPSG::25833& scaleSize=x(500),y(500)`

The downloaded file contained a number of Extensible Markup Language (XML) headers indicating that this was a GML file and that a GeoTIFF had been downloaded. There was also a large amount of encoded data that that numerous trials and errors proved unable to decode.

The reigning theory was and still is that the resulting file is somehow a GML file that contains the requested GeoTIFF. Performing the request and opening the file using a desktop program for geographic data (QGis) was successful, indicating that the height data is present. The Content-Type: header also confirms that the data at the bottom is in the TIFF format.

Attempts were made to strip the GML headers and opening the resulting file as a GeoTIFF, with and without steps for decoding, but to no avail. A more detailed discussion on how this could be solved in the future will follow in the future work section of this report.

In the end, the HeightData class as it stands is able to load height data successfully from a cached GeoTIFF file of Kongsberg, which covers about 3km x 3km of area, and can also be used with any other GeoTIFF file given the resolution of the data is 1m. An if test in the code tests whether any given origin point will fit into the selected data set and is the natural place to implement the dynamic downloading of TIFF files in the future.

8.3.2 Class flow chart

HMM | *NH*

Usage of the HeightMap class is designed to be as simple as possible for the end user. In the simplest scenario where the user will use the integrated file of Kongsberg city for route planning, starting a new project will construct a HeightMap class. After a new instance of HeightMap has been instantiated, the user will then be able to enter origin coordinates. This, in turn will lead to the member variable of that instance of HeightMap to be populated with the heights for each point within the selected subset. See fig. 17 for an illustration of this. Initially, the size of the subset was hard-coded to be 500 x 500 meters. After the MVP had been finished, HeightMap was updated to also allow for dynamic size of the generated height map, through adding an argument for selection size in the UpdateOrigo() method.

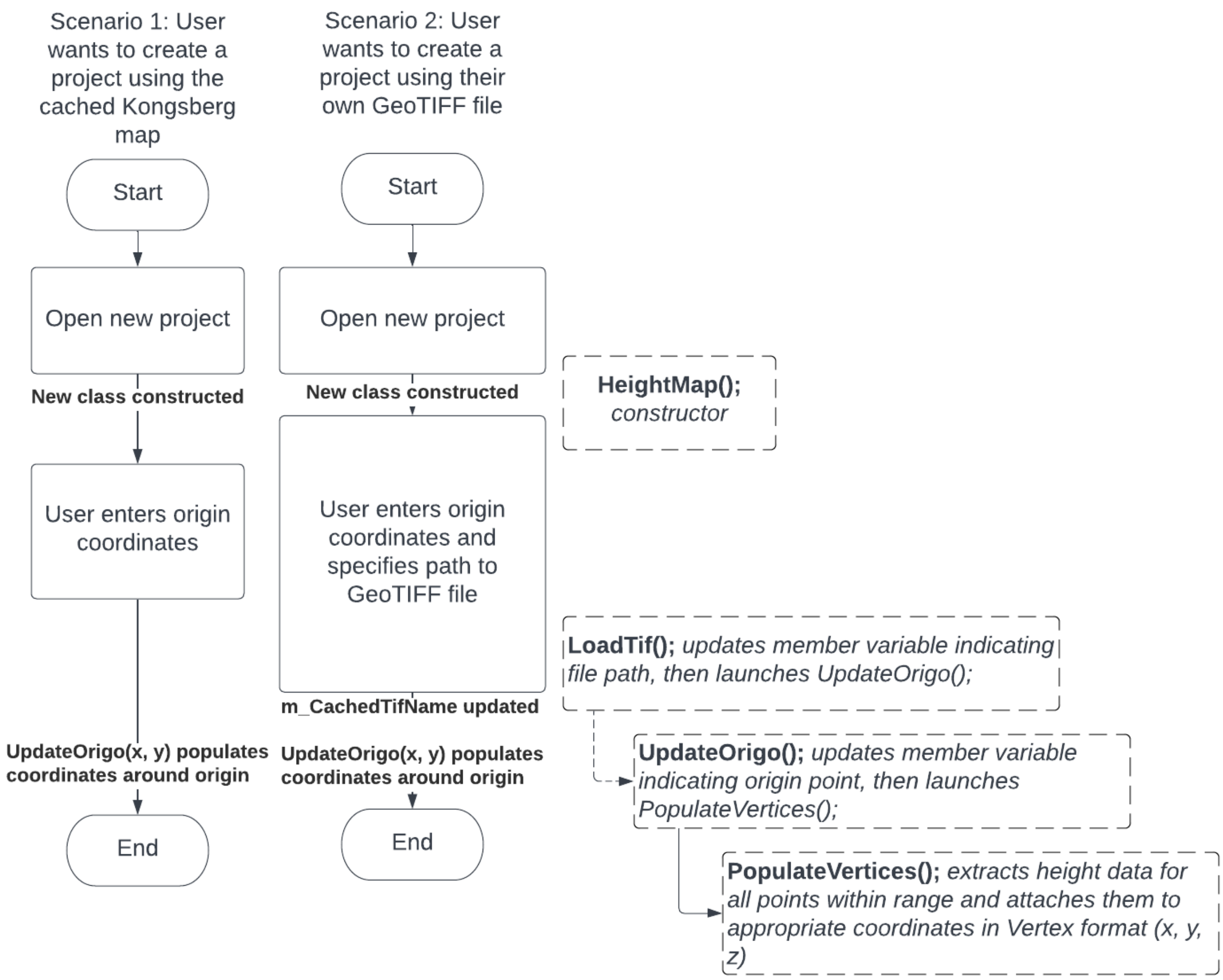


Figure 17: Flow diagram showing two possible usage scenarios for HeightMap

In the event that the user wants to populate height data from a different map, another method (LoadTif()) can be used to specify the path to another GeoTIFF file. The LoadTiff() method will then in turn run the UpdateOrigo() method and populate the HeightMap with height data. In general, it does not matter what coordinate system the new GeoTIFF file uses, as long as the user is consistent in using this system in other parts of the program. The resolution of the file, however, should be 1m.

Several methods have also been made to extract heights from the HeightMap. The most important one is the GetHeight(x, y) method, which takes in the relative X and Y coordinates (where 0, 0 is the top left corner coordinate) and returns a float containing the height for that given point. This has been illustrated in a simple flow chart, seen in fig. 18. Other methods take in geographic coordinates and return height, or height and x, y coordinates, take in relative coordinates and return x, y coordinates and height. These methods are not used in the Routemaker component. Instead, the GetHeight() method is used. The other methods to fetch height are nevertheless extant in the HeightMap class, in case they need to be used for testing or if HeightMap is to be used by itself.

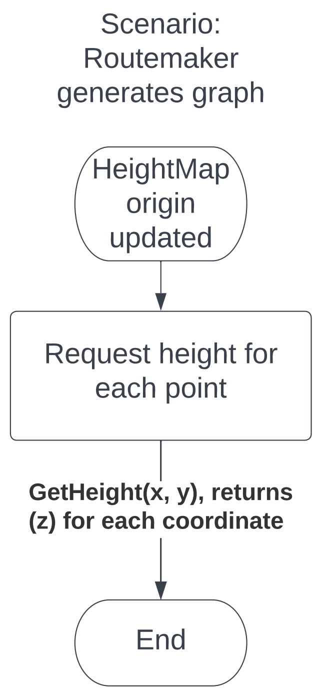


Figure 18: All methods for retrieving height rely on just receiving X and Y coordinates.

8.4 Map Management **AM** | *RS*

The map manager receives the origin and size information of the map from the GUI, through the compile scenario component. After processing this information, the map manager updates the map data which is used in the GUI for visualization. It acts as a mediator between the GUI and the map-related functionality in the system, ensuring that the map is always up-to-date and correctly displayed to the user.

8.4.1 HTTP request **AM** | *RS*

The *GetMap* method sends an HTTP request to retrieve map data from a Web Map Service (WMS) server provided by Kartverket. The request is constructed using several parameters, each specifying a different aspect of the requested map image. The request parameter specifies the type of request being made, which is a "GetMap" request in this case.

The *service* parameter specifies the requested service type, in this case, a WMS service.

The *version* parameter specifies the version of the WMS protocol being used, which is 1.3.0 in this case.

The *layers* parameter specifies the data layers to be included in the map image. Hivemind needs to have a map that includes height data, land cover data, water data, transportation data, and building data; therefore, this is specified in the request.

The *styles* parameter specifies the style to be used for the requested layers, which is set to the default style in this case.

The *format* parameter specifies the response format from the server, which is set to a PNG image in this case.

The *crs* parameter specifies the coordinate reference system (CRS) to be used for the map data, which is EPSG:25833 in this case. This is the UTM zone 33N coordinate reference system used in Norway.

The *bbox* parameter specifies the bounding box of the map image to be requested. The bounding box is calculated by the CalculateCornerCoordinates function in the code, which takes a UTM coordinate and a size parameter and calculates the bounding box based on those values. The width and height parameters specify the width and height of the requested map image in pixels.

Once the request is constructed with these parameters, it is sent using an instance of the QNetworkAccessManager class, which is part of the Qt Network module and handles the communication with the WMS server. When a response is received from the server, it is processed by a callback function that extracts the image data from the response and stores it in the *m_Data* member variable of the SatelliteMap class. The *GotImage* signal is then emitted, indicating that the image data is ready to be displayed or further processed.

8.4.2 Signals and slots **AM** | *RS*

The Map Manager in Hivemind utilizes the Signal-Slot mechanism provided by Qt to communicate with the GUI and update the displayed map. When the map is ready to be displayed, the Map Manager emits a *GotImage* signal which is connected to a slot in the GUI. Upon receiving the signal, the GUI updates the displayed map with the newly obtained image data. This approach allows for decoupling the GUI and Map Manager, enabling them to work independently of each other while still maintaining effective communication. Additionally, the Signal-Slot mechanism provided by Qt ensures a thread-safe implementation of the communication between the GUI and Map Manager.

8.4.3 Calculating the bounding box **AM** | *RS*

The Map Manager has a *method* for calculating the corner coordinates of the area to be fetched, which are used in constructing the WMS request. It takes in a UTMCoordinate and a size parameter, which specifies the size of the map image to be requested. The function first calculates the minimum and maximum x and y coordinates of the bounding box by subtracting and adding half of the size to the easting and northing coordinates of the input UTMCoordinate, respectively.

The method then creates a QStringList containing the four bounding box coordinates in the order of minX, minY, maxX, and maxY. These values are converted to strings using the QString::number() function. Finally, the function joins the four coordinates into a single string using a comma separator and assigns the resulting string to the *m_Area* member variable of the SatelliteMap singleton instance. The *m_Area* string is later used to construct the HTTP request URL in the *GetMap* method.

8.5 Keyframe Management

NH | *AM*

In the development of our route planner, the need for points to establish a route between two or more locations was identified. To address this requirement, the concept of keyframes for a specific agent at a given time was introduced. Each keyframe comprises an agent ID, a timestamp, and a position represented by a cartesian coordinate.

The KeyframeManager class was designed to handle the management of these keyframes. It employs a singleton pattern to ensure that there is only one instance of the KeyframeManager in the entire application. The keyframes are stored in a vector, which allows for efficient access, addition, and deletion of keyframes.

Methods to add keyframes to the KeyframeManager were implemented in multiple ways. Users can input keyframes by providing individual parameters such as agent ID, timestamp, and cartesian coordinates, or they can add a fully constructed keyframe object. This flexibility ensures the KeyframeManager can accommodate various input scenarios.

To facilitate the deletion of keyframes, function that removes a specific keyframe from the vector by finding and matching it against the provided reference was implemented. This function is particularly useful in conjunction with the graphical user interface elements, where users can select keyframes for deletion from a list. However, searching through the entire vector for the exact keyframe to delete is not the most efficient solution. As the number of keyframes grows, this approach could lead to performance issues.

To improve the efficiency of the keyframe deletion process, alternative data structures and algorithms that would allow for quicker identification and removal of keyframes need to be considered. One potential solution could involve using a more sophisticated data structure, such as a balanced search tree, that maintains the sorted order of keyframes and allows for faster searching and deletion operations.

8.6 Routemaker

RS | *HM*

For a route-planning system to work, there needs to be some way of actually creating routes based on some input. This is the responsibility of the *Routemaker* component. It was decided that the general usage of the Routemaker component was to supply it with two inputs representing two locations, and it returning a list of points defining an optimal route between them. The routes generated should take terrain and buildings into account in order to avoid collisions.

Route generation is not a new idea, and there is much research on the topic. In the world of algorithms and graph theory it is often referred to as path-searching or graph-searching. A simple A* algorithm was chosen in order to quickly get a working product. A* is an efficient best-search-first algorithm for finding the *cheapest* path between two nodes in a graph, where the cost is defined by a heuristic. In Hivemind, the heuristic is defined by the distance between nodes, meaning the cheapest path will be the shortest one.

8.6.1 Graph class

RS | *HM*

To get started, a *graph abstract data type* was developed. It was implemented by making a simple graph interface that has several methods for working with nodes. The nodes hold some information needed by the A* algorithm. They also hold some abstract data, made possible by C++'s template system. The rationale for implementing an abstract interface was to make it as flexible as possible. The A* algorithm is not aware of the underlying data or use-case, it just works on a graph. To use it, one would create a sub-class of Graph, and implement a few methods needed by the A* implementation.

To test the A* implementation, the team created a 2D Grid class inheriting from the Graph interface. Each cell in the grid can either be occupied or not occupied. After the required methods were implemented, finding the shortest path between two cells in the grid was as simple as calling the *SolveAStar* method. One of the methods sub-classes of Graph must implement is *GetNeighbors*, which returns a list of all neighbors of the provided node. Since the *GetNeighbors* method is implemented by the Grid class, not allowing the path to cross occupied cells is simple; just avoid including occupied cells as neighbors. Fig. 19 shows a path generated on the grid.

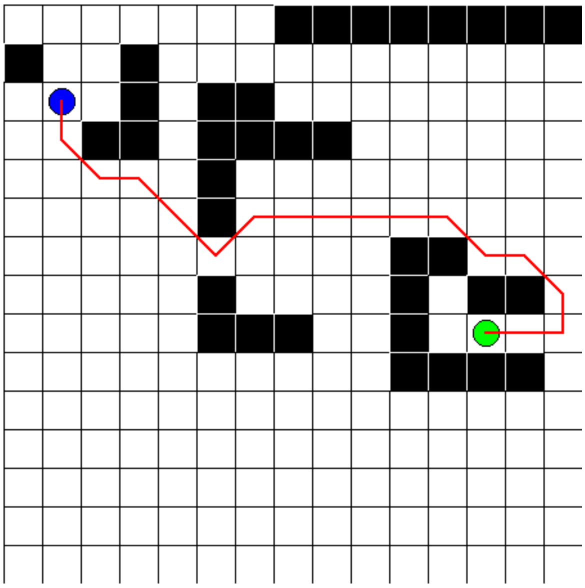
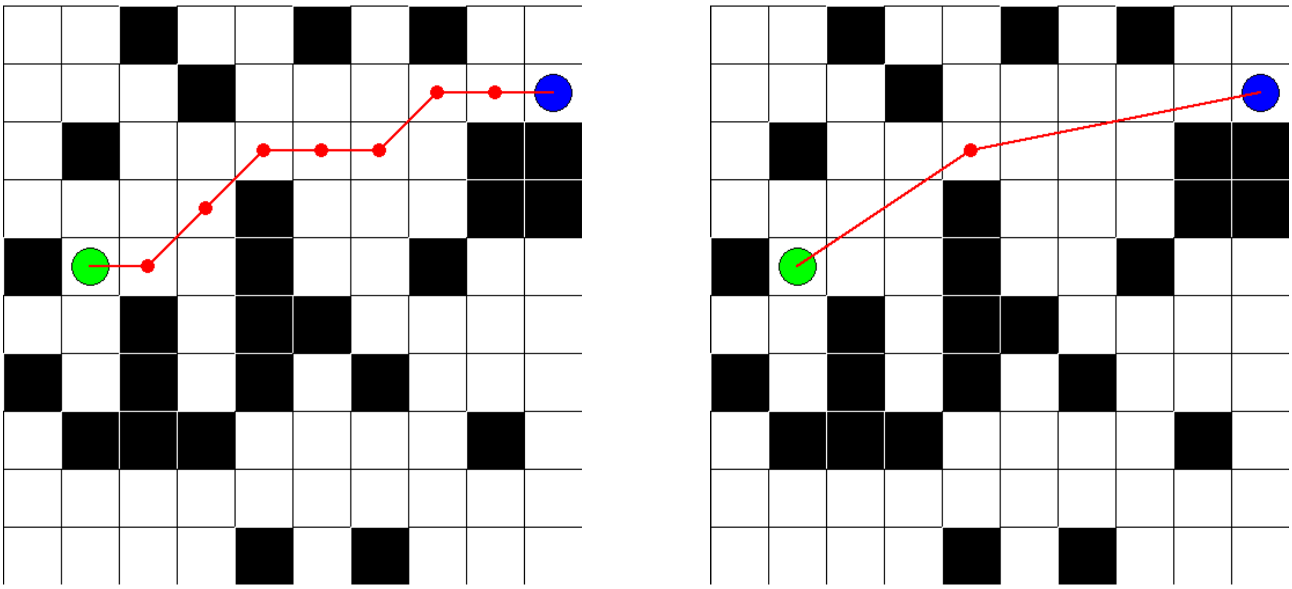


Figure 19: Simple path generated by A*

8.6.2 Post-smooth process

RS | *HM*

Fig. 19 highlights a weakness of the path-finding algorithm: As A* only considers direct neighbors of any given node when exploring the graph, it ends up generating quite a rough path. In the context of a grid such as this one, it means it is locked to 45 degree movements. One solution to this problem would be to consider an any-angle algorithm instead, such as Theta*[53]. This ends up adding quite a bit of complexity though, and since the focus of Hivemind was on creating a proof of concept, it was decided to continue using A*, but also implement a simple path-smoothing algorithm which runs after the A* algorithm finishes. After the A* algorithm finishes, the path is defined by parent-child relationships in the nodes. Each node has a pointer to it's parent, so starting at the end node and following the parent recursively, will eventually lead back to the origin point. The post-smoothing simply starts at the end node and checks if it has a direct line of sight to its grandparent. If it does, it makes its grandparent its parent instead. Then it checks again. If it does not have a direct line of sight to the grandparent, it moves on to the parent, and starts checking for that node. It keeps going until it reaches the start node. Not only does this smooth out the path, it also potentially results in a shorter path. The resulting path also consists of fewer nodes, making it more memory-efficient. Figure 20 compares the paths generated from two points on a grid before and after smoothing. Note the significant reduction in amount of nodes that define the path in fig. 20b as compared to fig. 20a.



(a) Path before smoothing

(b) Path after smoothing

Figure 20: Comparison of paths generated with and without smoothing

8.6.3 Bresenham's line algorithm

RS | *HM*

As previously mentioned, the Graph interface requires sub-classes to implement a method that determines whether or not two nodes have a direct line of sight. When considering a 3D environment, we may have to look into ray-casting for doing this efficiently. However, as we are currently still in 2D we may use something a little simpler, like Bresenham's line algorithm[111][112]. Bresenham's line algorithm is often used in the context of 2D raster images, when one needs to compute which integer pixel indices a line intersects with. It takes two end-points, and computes all integer coordinates that make up the line segment between them. This means we can determine line of sight between two nodes by iterating over all the nodes in the positions calculated by Bresenham's line algorithm and check if any of the nodes are occupied. If any of them are, there is not a line of sight between the nodes. Fig. 21 illustrates bresenham's line algorithm.

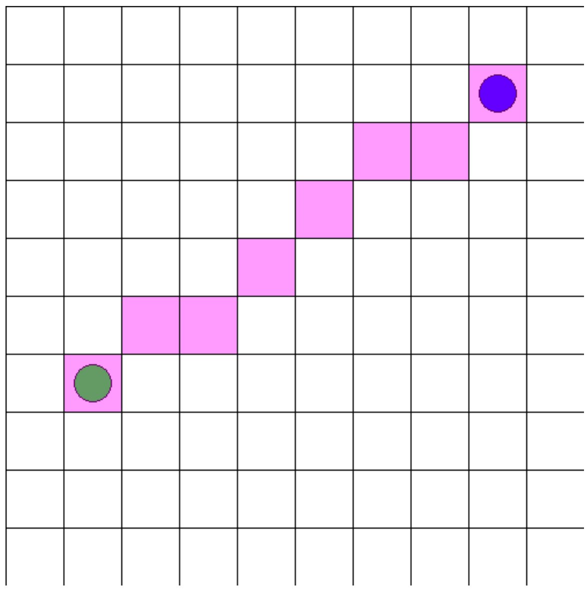


Figure 21: Bresenham's line algorithm

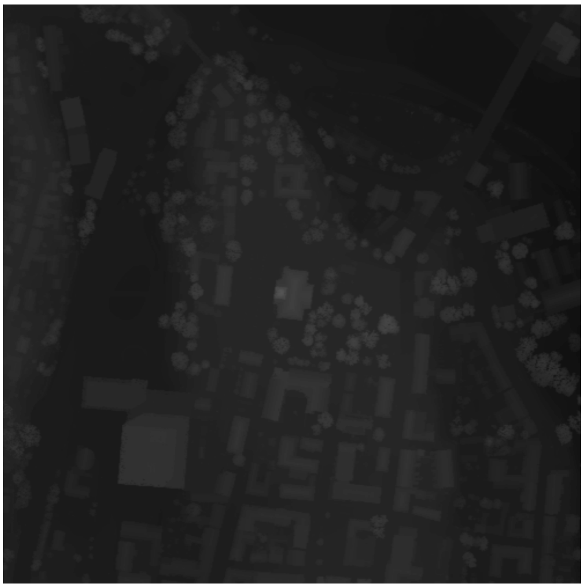
8.6.4 Routemaker implementation

RS | *HM*

Having implemented and tested the A* implementation and post-smoothing algorithm, the actual Routemaker class could then be implemented. Optimally, the Routemaker should operate in a 3D environment, but to simplify the initial implementation, a 2D system was chosen. This means a fixed height for the drones to fly at was defined, while the graphical representation maintains a top-down view. A lot of the logic for route generation was already in place since with a 2D environment, the resultant point of view is essentially a grid similar to the testing class used earlier.

It is important to note that the abstract graph interface makes for a very adaptable Routemaker. Even though Hivemind is only considering a 2D environment at this time, moving into 3D is easy in terms of the Routemaker implementation; by adjusting the *GetNeighbors* method to account for neighbors in the vertical axis. Additionally, since the graph base class is a standard graph datastructure, any other graph-searching algorithms can be implemented to improve upon the system or simply to compare with the A* implementation.

Up until now, the grid has been randomly generated for testing purposes. However, as the purpose is to generate routes for drones in the real world, grids that represent the real world must be generated. To do this, the Routemaker uses the Heightmap component to query the height data over the terrain. It then generates a grid based on this. A height threshold is defined, representing the drones' flight height, and if the height in the height data at this point is larger than the height threshold, the corresponding node is defined as occupied. Fig. 22 shows a heightmap and the corresponding grid that Routemaker creates. The flight height in this example is 175 Meters above mean sea level (MAMSL).



(a) Heightmap. Brightness and contrast adjusted for visibility

(b) Routemaker's grid based on heightmap

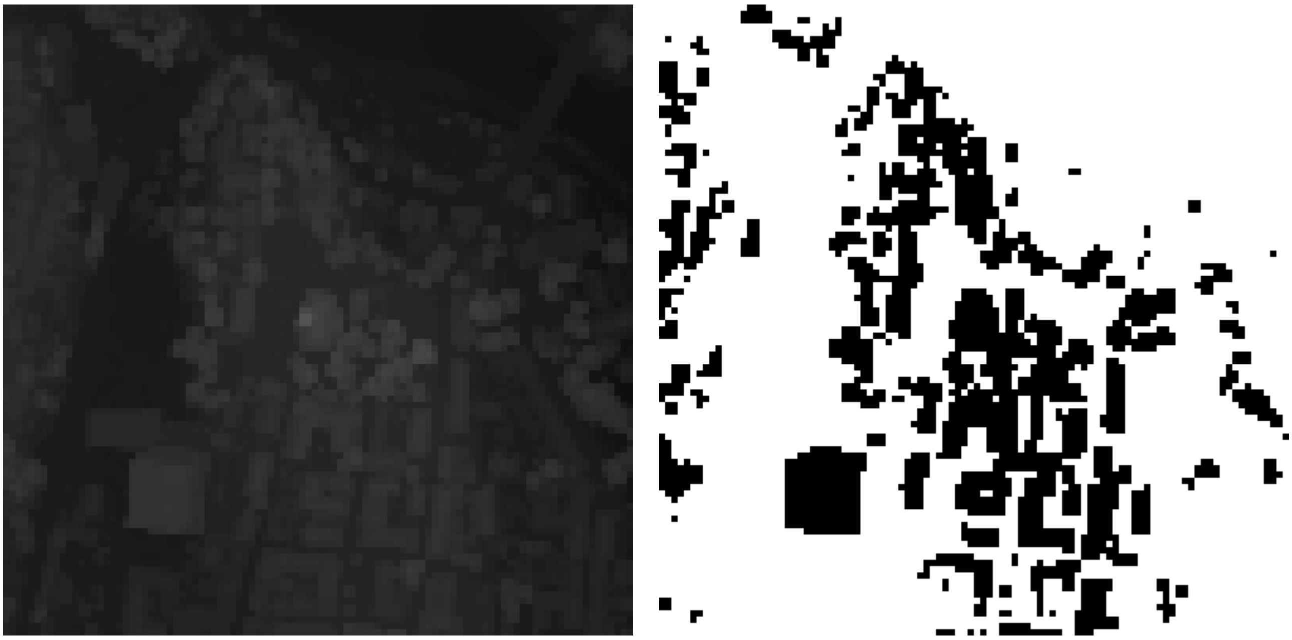
Figure 22: Heightmap and corresponding routemaker grid with a flight height of 175 MAMSL

As for the interface when generating routes, the Routemaker class has a *MakeRoute* method, which takes two keyframes as arguments. The positions defined in the keyframes are in a symmetrical cartesian space, but the Routemaker's grid uses an asymmetrical cartesian coordinate system. Because of this, all keyframe positions are transformed using the Coordinate Converter class before the path-finding starts. Additionally, after the path has been generated, all coordinates that define the path are transformed back to a symmetrical Cartesian space before returning the path. This means that from the outside of the Routemaker class, there is no need to consider an asymmetrical Cartesian coordinate system. Both the inputs and outputs use symmetrical cartesian coordinates.

8.6.5 Resolution

RS | *HM*

An issue that quickly became evident was computation time. The height data has a resolution of 1 meter per measurement. By default, the Routemaker grid has the same resolution. This is fine for smaller areas, such as a 200x200m area. However, when wanting to create a scenario on a larger scale, like 2x2km, the search complexity of the A* algorithm increases exponentially. To mitigate this issue, functionality for reducing the resolution of the Routemaker was implemented. To do this, during the building of the Routemaker grid, the largest measurement from the height data for each *block* of measurements is taken and used to determine whether or not a node is occupied. Also, when given keyframes to use for path-finding, the keyframe positions need to be divided by the resolution to further transform them to the Routemaker's space. Before returning the path, the positions now need to be multiplied by the resolution. Figure 23 illustrates the reduced height data and corresponding generated grid when using a resolution of 5 meters per measurements.



(a) Heightmap: 5 meter resolution (b) Grid: 5 meter resolution

Figure 23: Heightmap parsed by Routemaker with resolution of 5m and resulting grid

When comparing fig. 23 to fig. 22 it is obvious the data is still valid after reducing the resolution, but that details have been lost.

8.7 Serializer (Load and Archive Scenario) **HM** | *HMM*

To simplify the technical work related to saving and loading, a data driven interface for the Serializer was developed. This interface is only possible to implement if it is able to keep references of the actual values to be stored in JavaScript Object Notation (JSON) format. It also has to be able to keep track of where the different value belong in the data structure, the ability to do this is called reflection[113]. Reflection as a concept is not supported by C++, but in many other programming languages like JavaScript, python or C# it is.In practice, this was solved by creating an abstract class in which a method called GetProperty was defined. All classes that are serialized in Hivemind will then need to inherit this class and implement the GetProperty method.

8.7.1 Implementation:ISValue **HM** | *HMM*

All other elements of the Serializer is based on an abstract class called IValue. ISvalue implements some functionalities that have to be in place for the Serializer to work both for serializing and deserializing. The first thing ISValue does is determines whether the object to be serialized is a composite object or a primitive object. Specifying the type of object is done through an argument specified when calling the Serializer to serialize an object.

When called with a primitive type such as an integer, it will store the name of the variable as well as a reference to its value. If called with a composite type, like an object, it will retrieve the specified primitive type stored in the object instead. Retrieving one value from within a composite type variable is done through the GetProperty function, which creates a map of all the member variables of the composite type.[114]

ISValue is implemented so that it can support other file formats if it is deemed necessary to implement another one like XML or a database. The only requirement for the chosen format is that it supports the third level of sophistication.

8.7.2 Implementation: Types

HM | *HMM*

Each type the Serializer needs to be able to handle must be implemented separately. The Hivemind Serializer is compatible with the following types:[6]

- Integers
- Floats
- Doubles
- Strings
- Bools
- Objects
- Members
- Integer vectors
- Float vectors
- Double vectors
- Object vectors
- Member vectors
- Object vector vectors
- Member vector vectors

For all the primitive types the Serializer takes the name of the variable and a reference to the value of the variable and stores it as a name and value pair which gets pushed straight to or from the JSON file through the use of the RapidJSON library. To get data to the JSON file, the ToDom function is called. This stores the value of a member in the RapidJSON document, which after serialization will be stored in a JSON file.

When retrieving stored data, a RapidJSON document is populated with the specified JSON file. Calling the FromDom function on this JSON file will retrieve the value of specified the member in the JSON file, and store it in an object similar to the one that was initially serialized.

To serialize composite types, like members and objects, a JSON object is created and all the primitive types to be serialized added to the object as member values.

In order to serialize vectors, a JSON array is created, and all the values to be serialized are added to the JSON array.

8.7.3 Implementation: ISProperty

HM | *HMM*

ISProperty is a struct which forms the basis for the entire Serializer. Its main function is to enable the Serializer to replicate the name and value pair structure of a JSON file.

8.7.4 Implementation: Macros

HM | *HMM*

The macros created for the Serializer are there to improve workflow for other application programmers when creating objects with persistence in Hivemind.

8.7.5 Persistence in C++

HM | *HMM*

Persistence is the ability to store data beyond the lifetime of the program. In C++, this can be achieved persistence by storing data in a file on the disk or a database.

Having the opportunity to make data persistent is useful since it makes for multiple users to access the same data, or for the program to reuse the same data at a later date. Enabling saving and loading is a core requirement for Hivemind because it allows the user to create a scenario and come back to it later, avoiding having to make the same scenario on multiple occasions.

In C++, it is possible to achieve persistence through a number of different mechanisms. Examples of these are making use of serialization frameworks, databases or input/output operations. Serialization frameworks like Boost.Serialization[115], cereal or RapidJSON[116] make it easier to develop C++ programs that require persistent data.

Developing functionality for persistent data in C++ is not without its challenges. The developer will need to keep in mind the format of the stored data, the structure of the data in the program as well as how determining how to access the data both when saving it and when rebuilding the data structure at a later date.[114]

Challenges to achieving persistence in C++ The big challenges with persistence in C++ is that since it is a low level language, the developer needs to pay special attention to memory management as well as keeping track of objects lifetime, in addition to making sure steps are taken to guarantee data consistency.

When it comes to memory management in C++ it can be difficult to manage data that has to persist. When objects have to persist beyond the lifetime of the program, it is important to be able to ensure the intended data is saved. In C++ objects are usually saved on the heap or on the stack. As a result, the location of the data intended for storage may contain something completely different, if the original data was overwritten. This means the developer needs to be particularly careful in making sure the desired data is the one that actually exists in a certain location before saving it.

Object serialization is the ability to convert objects to a format that can be written to a file on the disk or some other non-volatile storage media. When developing functionality for serialization and deserialization, it is crucial to handle undefined behavior and make sure there are no data leaks.

One final, but significant, reason that dealing with persistent data in C++ is that it is difficult to guarantee that the data stays consistent from one execution of the program until the next execution. This leads to requirements for the data to be stored in a specific format, having a way for data encoding, needing to be prepared for error handling, data validation before storage, and concurrency of the data.[117][113]

Benefits of persistence: There are many benefits to having persistent data when making a software application. Being able to preserve data between different executions of the program is helpful, for example in the situation where a user is planning a drone show with a large number of drones. Being able to store the data and start up again means the user will never have to redo their work, unless they want to.

Persistence also enables the sharing of data. Because a save file can be sent between users without data degradation, multiple people are able to share and collaborate on the same project. [117][114]

Why JSON: The JSON format was chosen for data storage in Hivemind because it is compatible with the ROS, which it was assumed the drones Hivemind would be controlling would be using. Emphasis was also placed on the ability to store data in a human readable format, in order to make it easy to understand what the drones were doing even when not working directly in the Hivemind user interface. Since JSON is easy to read and understand for humans as well as being a format the drone can work with, rapidJSON[116] was chosen. Another option for storage briefly considered was the XML format, but since data eventually would need to be in JSON regardless, it was decided to only use this format for serialization and data storage.[6]

The Document Object Model: A Document Object Model (DOM) is an interface that is used to ensure data is stored in a way that makes it possible to restore the same data structure at a later date[118]. It has a structure that resembles a tree where the data is stored in nodes and objects. When making a HTML and XML document, DOM is communicated through to add elements, remove elements as well as change elements that are already there. To store C++ data in a JSON file, the C++ library called RapidJSON was used, which has a DOM style API for parsing and generating C++ data in a JSON file.[118]

Levels of sophistication: When serializing data in C++ objects, it is also important to consider how complex the objects to be stored are. Only when the complexity of these object has been determined is it possible to choose the technique(s) used for serialization.[114]

The lowest level of sophistication is used when the data to be serialized does not contain pointers to other objects and they are not part of an inheritance hierarchy. When using this technique every class is responsible for their own serialization and if they contain other classes they should only call the serialization function contained in that class.

The second level of sophistication can be used serialization is to be performed on a data structure with objects that are a part of an inheritance hierarchy, but the objects do not have pointers that point to different objects. This is the technique to be used when there are multiple classes that are derived from a more abstract class. When serializing some data the first thing that is needed is the name or identity of the object.

The third level of sophistication is used when a class contains pointers to different objects. In this level of sophistication the pointers can not form cycles or joins. This leads to the data structure looking like a tree. No cycles means that following the pointers from object to object will never lead to an object that has already been visited.

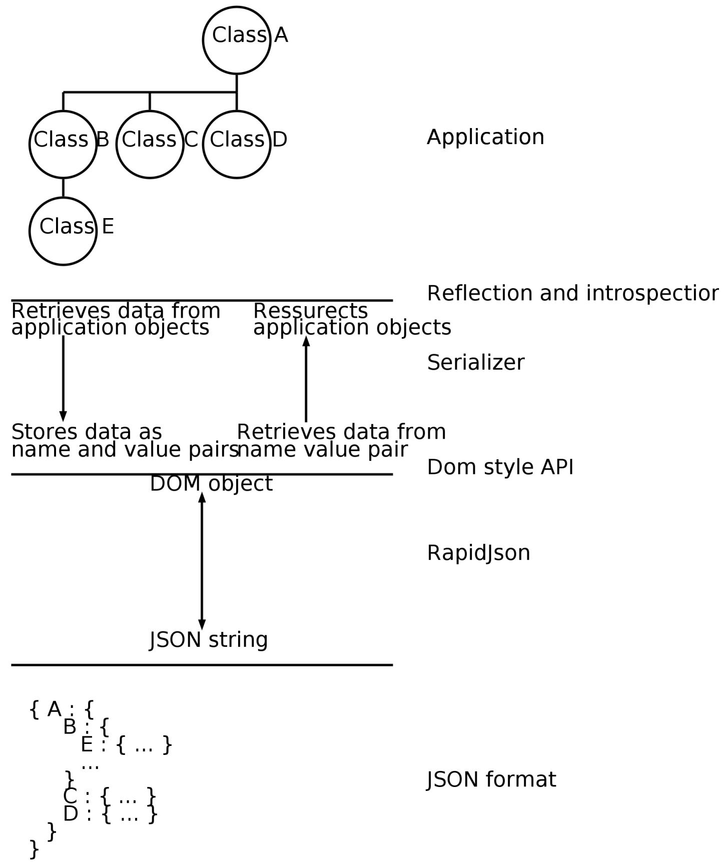


Figure 24: Architecture and dataflow for serializer

This is achieved with a recursive algorithm that will serialize objects that are pointed to when it gets to them in the serialization process. When using this technique, it is important to focus on the contents of the objects. If the object has a pointer, a string and another pointer, then the first step is to dive into the first pointer and serialize all the data and pointers it contains. After this has been done, the string is serialized, before lastly following the second pointer and serializing everything that pointer points to. To avoid memory leaks when using this technique, it is crucial to only use smart pointers.

The fourth level of sophistication is mostly the same as the third level, but serialized objects can contain pointers that point to the leaves of a different tree. The serialization part is done the exact same way as at the third level, meaning the difference lies in wanting to generate the original data structure from serialized data when deserializing. This is solved by creating a look-up table that contains the variables serialized and the nodes they belong to. When deserializing, any variable that has already been deserialized is skipped.

The fifth and most sophisticated level of serialization is used when the objects have pointers to different objects, those pointers form a tree that can contain cycles and joins between trees are not only in the leaves. This can be achieved if infinite loops are successfully avoided. A way of achieving this is by creating an object-ID map that is built by serializing the objects in the same recursive manner as in sophistication levels three and four. The main difference is that a check of whether the node is already on the object-ID map is performed at each node. When all nodes have been iterated through, a second pass is done, at which time the contents of the nodes are written along with the ID of each node. This time, the process does not recursively dive into the tree, instead following the object-ID map.

In Hivemind, the third level of sophistication has been chosen because the data that will be serialized can easily be represented in a tree form with no joins and no cycles.

Human readable vs non-human readable: When choosing between storing data in a human readable format or a non-human readable format there are several questions you need to answer. Is it important to be able to read and understand the data when it is stored? If so then you should choose a human readable format. A binary format can be slightly faster but it is not a very relevant bottleneck unless your application is using 100% of your cpu and a significant portion of that is used on your Serializer. For the Hivemind application we chose a human readable format mostly because we wanted to use JSON and never really considered going with a binary data format.[114]

8.8 Compile Scenario

AM | *RS*

The Compile scenario component is an essential part of Hivemind. This is the component is responsible for updating the area and generating the scenario. The software component generates scenarios using the predefined keyframes from the user. The compile scenario instantiates a Routemaker object which subsequently instantiates a HeightMapg object.

When the user sets the origin and size of the area in the GUI, this information is sent to the Compile Scenario component. The Compile Scenario component updates the origin and size throughout the system. Fig. 25 illustrates how the Compile Scenario component creates the Routemaker, which in turn generates the HeightMap and updates the Map Management and Coordinate Converter. By doing so, the Compile Scenario component always updates the origin and size parameters of these components when the data retrieves the GUI. Additionally, the figure demonstrates that the Compile Scenario component also updates the Map Management.

In the Compile Scenario component, the routes generated by the Routemaker are stored in a map where the agentID serves as the key, and a vector of Cartesian coordinates represents the value. The routes obtained from the Routemaker are then appended to the corresponding

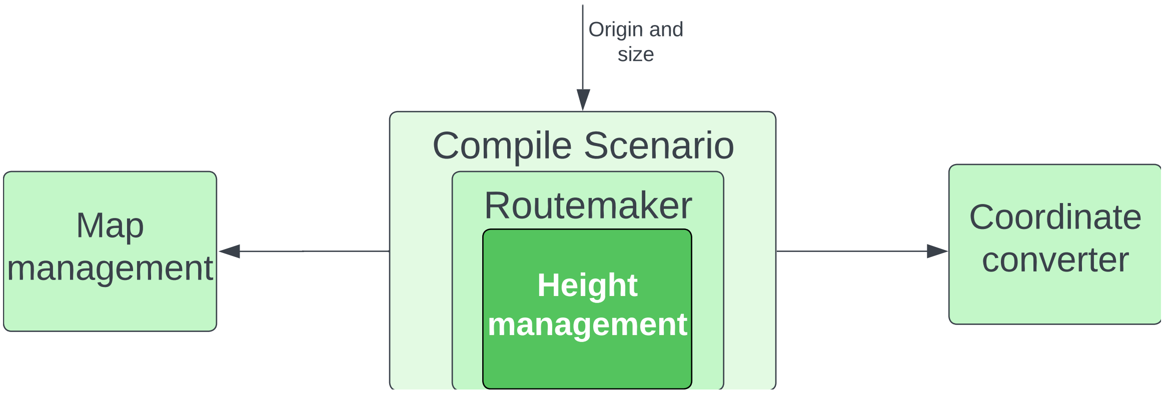


Figure 25: Origin and size update pattern.

vector based on the agent ID.

When the *compile* method is called, it first checks if any keyframes have been defined. If not, it simply returns the empty map with no routes. Otherwise, it sorts the keyframes by agent ID and timestamp and then iterates over them. It checks the agent ID for each keyframe and compares it with the previous one. If the agent ID is the same, the keyframe belongs to the same drone, and the component generates a route based on the two keyframes. If this is the first keyframe for the drone, the component creates a new vector for the routes belonging to that drone and adds the generated route to the vector. Otherwise, the route is added to the existing vector. This is shown in fig. 26.

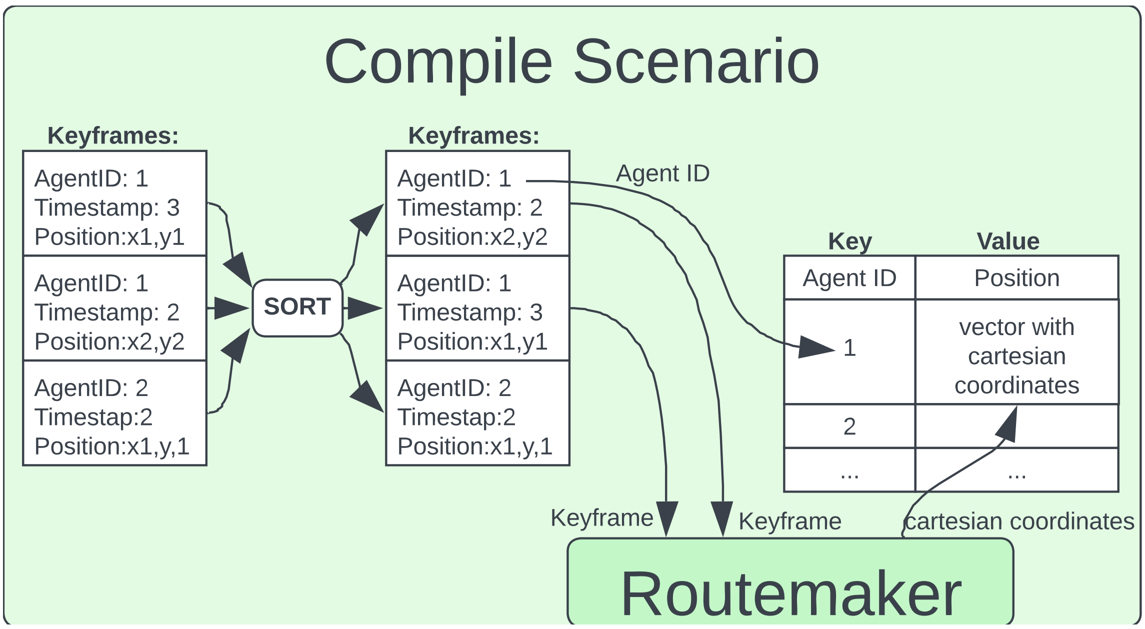


Figure 26: Compile scenario functionality

By keeping track of the agent ID of each keyframe and checking it against the previous keyframe, the Compile scenario component can associate each generated route with the right agent. This information is stored in a map, where the keys are the agent IDs, and the values are vectors of routes belonging to that agent. Finally, the generated routes are returned as a map of agent IDs and their corresponding routes.

To visualize the scenario in the GUI, the scenario returns the map with all the routes between the keyframes to each agent after the scenario has been compiled.

8.9 Graphical User Interface

8.9.1 The code structure of Hivemind's GUI **RS** | *HM*

The code structure for the GUI was implemented similarly to how Qt structures widgets internally - as a tree. The *MainWindow* class is located at the root of the tree. This class has two children; the menu bar at the top, and a custom widget called *MainContent*. The *MainContent* also has two children: the *Sidebar* and the *TabWidget*. Without diving further down, the sidebar can be summarized as a container for several tools available to the user. The tab widget, which as of now is focused on the planner tab, contains the map and timeline for the current scenario. Fig. 27 visualizes this tree structure of the widgets. The *simulator* and *launcher* widgets are both placeholders for future features, and are currently not in use.

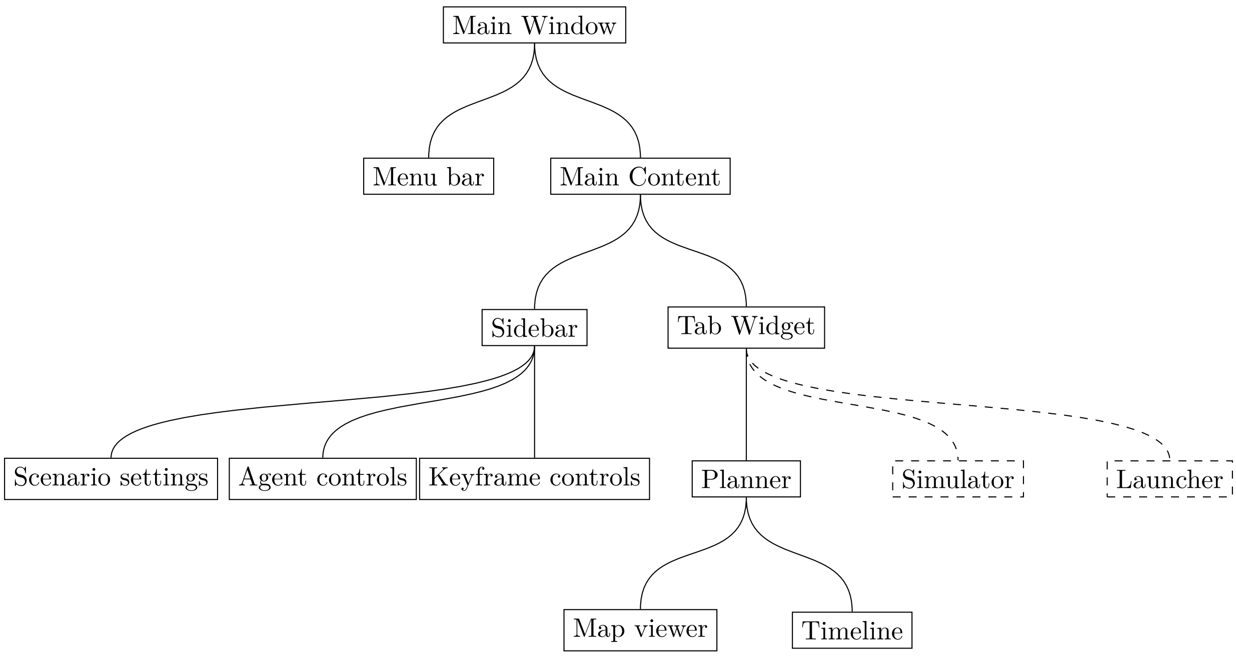


Figure 27: Graphical user interface tree

Each widget is responsible for defining their own signals and slots, as well as their own triggers for emitting their signals. The connections between these slots and signals are mostly handled in the *MainWindow* class. Originally, all the widgets made their own connections, but after moving the responsibility to the *MainWindow* class, the amount of cross-references between widgets has been significantly reduced.

8.9.2 Widgets in the Hivemind GUI **AM,RS,NH** | *RS*

Under scenario settings, the user can select a location with a geographical coordinate and size for visualization of the map to the scenario area. The agent controls widget allows the user to control the color of the agents, choose the active agent, and add new agents to the scenarios.

Finally, the keyframe controls provide an overview of all the keyframes in the scenario and a button to delete them.

As per fig. 27, the planner tab has two child widgets: The *map viewer*, and the *timeline*. The map viewer is an interactive visualization of the map contained in the *Map manager* component. In addition, all existing keyframes and compiled routes are visualized over the map, with unique colors for each agent. For improved interactivity, the *map viewer* is responsive to mouse presses, so adding new keyframes is as simple as pressing at a location in the map.

The timeline widget serves as a visual representation of time withing the scenario, offering users an easy way to navigate keyframes. The timeline operates using mouse clicks for addition or deletion of keyframes. The timeline's implementation follows a modular approach, ensuring easy reusability in different parts of the software. The timeline improves user-friendliness compared to previous methods, offering a streamlined experience for keyframe management.

The Graphical User Interface in it's current state can be seen in fig. 28, with a compiled scenario comprising three agents.

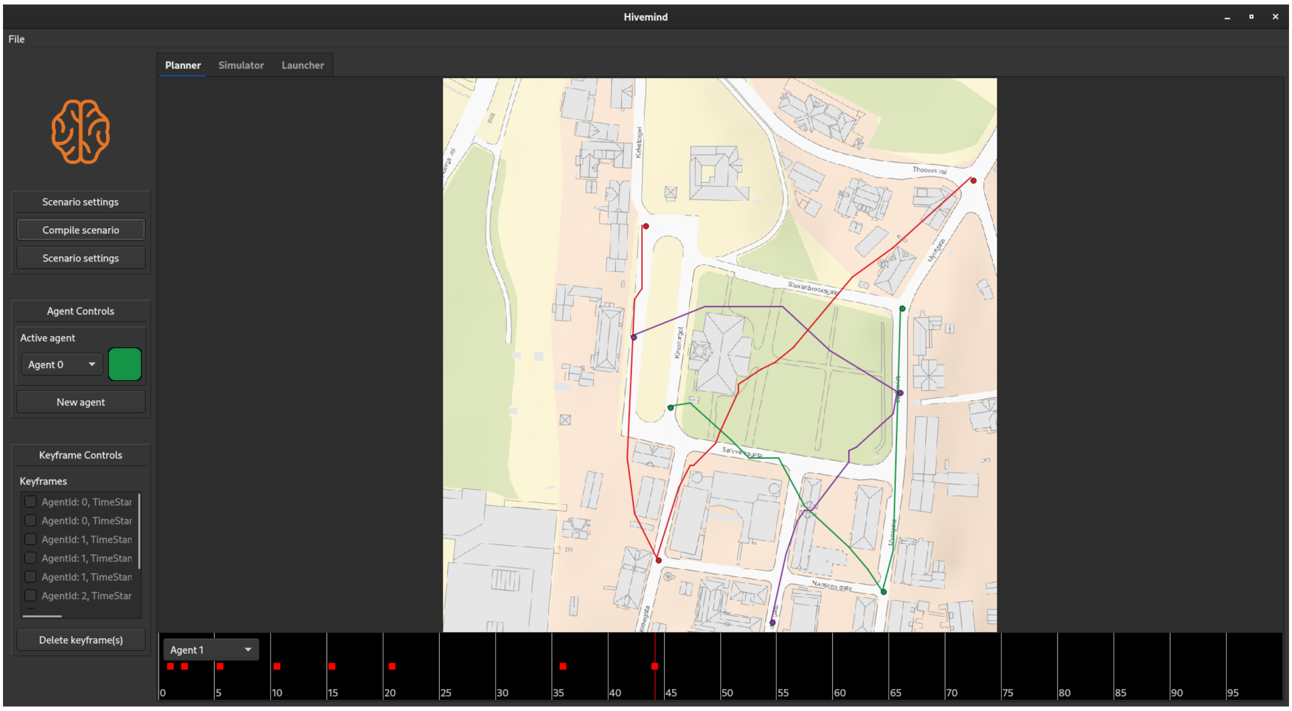


Figure 28: Graphical User Interface

All the software components for Hivemind were purposedly implemented in accordance with the developed software architecture. This section has described each of these components in detail, and shown how they work in tandem. An important part of implementation is continuous testing. Though not mentioned here, components were individually and continuously tested, before the entire software was integrated and tested as a whole. The next section will focus on these testing activities and explain how these were performed to ensure components functioned as intended, and that all implemented functionality met the requirements derived earlier in the project.

9 Testing

The project followed a method of continuous impromptu testing, as well as a structured testing scheme for verifying the software against the requirements (as described in the section for testing). Because the software contains a variety of different components, there was also a good deal of variation in how each component was developed and tested. This section will first explain how each of the software components were tested individually in simultaneously with development. It will then briefly present the testing regime for the software as a whole.

9.1 Testing of software components

9.1.1 Coordinate Converter

AM | *RS*

To test that the coordinate converter's functionality, each method that is used to convert coordinates was tested separately.
Reset origin: To test the function to reset the origin, the user sets a known origin with the *ResetOrigin* method and verifies that the origin is updated correctly.

Conversion between geographical coordinates and cartesian coordinates: To verify this, the same calculation is performed manually as done by the coordinate converter, and then the results are compared. Taking the example of 60 degrees north and 10 degrees east, where one minute of longitude is half the length of one minute of latitude (due to the sine of 60 degrees being 0.5), the conversion is validated.

Starting with the origin at 60N 10E, the coordinates 60'1N 10'1E are converted to cartesian coordinates. One minute northward corresponds to approximately 1.8507km, but it vary based on where it is on the earth[119]. At 60N 10E is one minute the same as 1/60 = 0.016667 in latitude.

Therefore, the calculation of latitude and longitude for the point are:

$$\begin{aligned} \text{latitude} &= 60 + 0.016667 = 60.0166667 \\ \text{longitude} &= 10 + 0.0166667 = 10.0166667 \end{aligned} \tag{8}$$

Using the *GeographicalToCartesian* method in the coordinate converter, the output should be that the x-axis is 0.5 of that of the y-axis. Therefore, the converter's output should be $y = 1850.7$ and $x = 925.35$

The output from the coordinate converter is $(929.532, 1856.99)$. Since the x-axis is approximately half of the y-axis which is around one minute, and the Coordinate Converter is using a recognized and mature library, we can assume that it is correctly converted.

Conversion between Symmetric and Asymmetric: Tested by validating the conversion results in all quadrants. To validate the conversion between asymmetric to symmetric, eq. 3 was used for the x-axis and the eq. 4 for the y-axis, and verified that is is the same point. To validate the conversion between symmetric to asymmetric, eq. 1 was used for the y-axis and the eq. 2 for the y-axis, and verified that it is the same point. An example of this is in the eq. 5 for the x-axis and in the eq. 6 for the y-axis. The result is shown in fig.15.

Conversion between Geographic and Universal Transverse Mercator (UTM): This conversion was tested in two steps. To validate the accuracy of the *GeographicToUTM* method, a specific geographic coordinate is selected on Google Maps, converted to UTM coordinates using the *GeographicToUTM* method, and compared with the corresponding location on Norgeskart. By confirming that the locations align precisely, this test confirms the reliability of the conversion.

To validate the accuracy of the *UTMToGeographic* function a specific UTM coordinate is selected on NorgesKart, converted to geographic coordinates using the *UTMToGeographic* method, and compared with the corresponding location on Google Maps. By confirming that the locations align precisely, this test confirms the reliability of the conversion.

To verify the CartesianToGeographical conversion, the same methodology was applied in reverse.

9.1.2 Testing of Serializer

HM | *HMM*

The Serializer needed to be tested a number of times during development. To perform the necessary tests, a test file containing multiple types to be serialized was developed. Every time a new type was added to the Serializer's capabilities, this test document was updated to include a new test for this type. After automated testing had been implemented, the test file was ported to the GoogleTest format so that tests could be run automatically.

9.1.3 Map manager

AM | *RS*

To ensure that the map manager displays the correct map, it can be tested by verifying that the coordinates for the origin and the size are correctly converted and calculated. This can be done by manually calculating the corner coordinates of the map and verifying that it represents the same area as shown in the GUI.

The following geographical coordinate and size were used in testing:

$$\begin{aligned} \text{Latitude} &= 59.66472311214873 \\ \text{Longitude} &= 9.644727959269787 \\ \text{Size} &= 200m \end{aligned} \tag{9}$$

These coordinates correspond to easting 536324.30, northing 6614249.51 in UTM 33N coordinates. As this is the centre of the map, each corner coordinate of the map is calculated by adding or subtracting half of the width and and half of the height to or from the origin. The calculated corner coordinates are as follows:

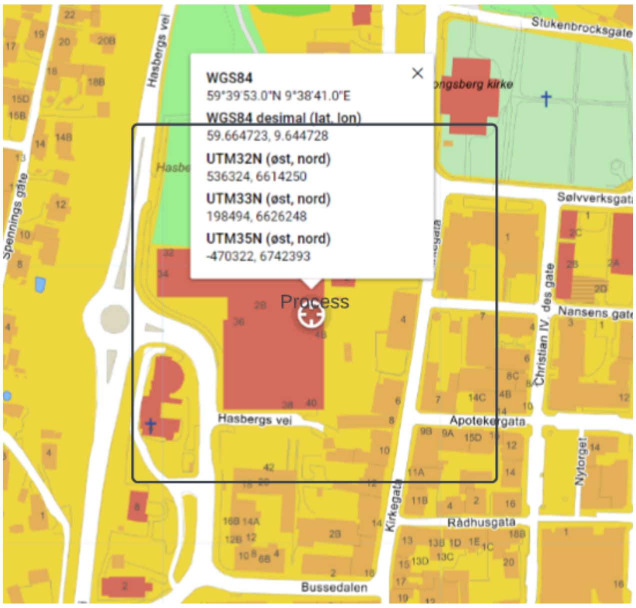
$$\begin{aligned} \text{West} &= 536324.30 - 100 = \underline{536224.30} \\ \text{East} &= 536324.30 + 100 = \underline{536424.30} \\ \text{North} &= 6614249.51 + 100 = \underline{6614349.51} \\ \text{South} &= 6614249.51 - 100 = \underline{6614149.51} \end{aligned} \tag{10}$$

This gives us the following corner coordinates:

$$\begin{aligned} \text{Top left} &= (536224.30, 6614349.51) \\ \text{Bottom right} &= (536424.30, 6614149.51) \end{aligned} \tag{11}$$

These corner coordinates can be used to find the map in external source. In this test it has been used GuleSider[120]. It was verified that the corner in the map in GuleSider corresponded to the corner in the screenshot from the GUI. Fig. 29 compares the map from Hivemind with the map from GuleSider. The figure demonstrates that the map in Map Manager is correct.

(a) Screeenshot of map in Graphical User Interface



(b) Screenshot of map from GuleSider.

Figure 29: Comparison of the map in Map Manager with the map from GuleSider.

9.1.4 Height Manager

HMM | *NH*

Developing the Height Manager was done in two distinct steps. First, to determine how to and successfully read height data from a GeoTIFF file and save this into a sensible data structure, then second, to develop methods to successfully extract the correct height for any given coordinate.

For the first step, the data extracted from the GeoTIFF file (once this was possible through code) was inspected to see if it made sense, and then cross-referenced with heights at corresponding coordinates from `https://hoydedata.no/LaserInnsyn2/` (which uses the same data set as the one used for testing and cached in the Hivemind software).

Once the integrity of the data extracted from the file had been confirmed, the methods to store and fetch height data for any given coordinate were tested. Methods for testing using UTM33N coordinates were also made, which was used to again cross-reference heights from a known coordinate on `https://hoydedata.no/LaserInnsyn2/` with the same coordinate in the HeightMap class. To confirm that methods to get height using relative coordinates, known heights of the origin of the selection and the calculated corner coordinates were compared with the Høydedata website. This was repeated with a number of different coordinates.

9.1.5 Keyframe Manager

NH | *AM*

The KeyframeManager was initially tested by implementing hard-coded keyframes, followed by verification of their successful addition to the vector through printing them in the terminal. Manual addition of keyframes through the terminal was next, and once that had succeeded a QDialogBox was developed. In order to verify that the QDialogBox functioned as expected, debugging mechanisms were implemented to print each added keyframe to the terminal. Once this functionality was confirmed, a delete keyframe dialog was introduced, which dynamically displayed all existing keyframes whenever opened. This feature allowed for easy verification of keyframe additions and deletions by inspecting the delete keyframe dialog.

9.1.6 Timeline

NH | *AM*

The timeline underwent manual testing throughout its development process, utilizing various methods of adding keyframes. Initially the add keyframe dialog was opened by clicking on the timeline, with the timestamp being automatically filled in from the timeline. Users were then required to input the coordinates and agent ID manually. The success of these keyframe additions was verified by examining the delete keyframe dialog, but for the final version a user clicks on the timeline, and then on the map and all the information is filled in for the user. The keyframe also showes up both on the map and on the timeline.

Once the keyframes could be added successfully via the timeline, the implementation of visual representation on the timeline itself was introduced. Testing involved adding keyframes and confirming their appearance on the timeline in the correct position.

Subsequently, a right-click function was implemented on each keyframe, triggering a dialog box to prompt the user for confirmation before deleting the selected keyframe. To validate this functionality, keyframes were added, deleted using the timeline, and then cross-verified with the delete keyframe dialog to ensure the correct keyframe was removed.

9.1.7 Automated Testing

HM | *HMM*

There were two steps involved in properly implementing GoogleTest in the Hivemind development pipeline. The first was successfully using GoogleTest locally on a computer, and the second was to create an Azure Pipeline able to run all tests in the system automatically whenever the repository was updated.[98][121]

Succeeding in running GoogleTest locally was straightforward, as the GoogleTest maintainers provide a step by step guide on installing the library and running sample tests. Setting up the Azure Pipeline to run tests automatically was done through creating a pipeline that followed the same step by step guide used to run tests locally. Finally, the trigger of the pipeline was defined as:

```
Trigger:
    Branch:
        Include:
            -'*'
```

This enabled the pipeline to run all tests when any branch is pushed.

9.1.8 Routemaker

RS | *HM*

The *Routemaker* component's individual features were thoroughly tested from the start. As the graph abstract data type and graph-searching algorithms were defined as abstract interfaces, a debug sub-class of the *Graph* class was implemented along-side them. Along with this, a debug Qt widget was also implemented to visualize and verify the functionalities of the component in the GUI.

The debug visualizer in the GUI proved very useful. Being able to visually differentiate an occupied node from an unoccupied node, and highlight the neighbors of certain nodes made the verification of smaller parts of the component efficient.

9.1.9 Compile scenario

AM | *RS*

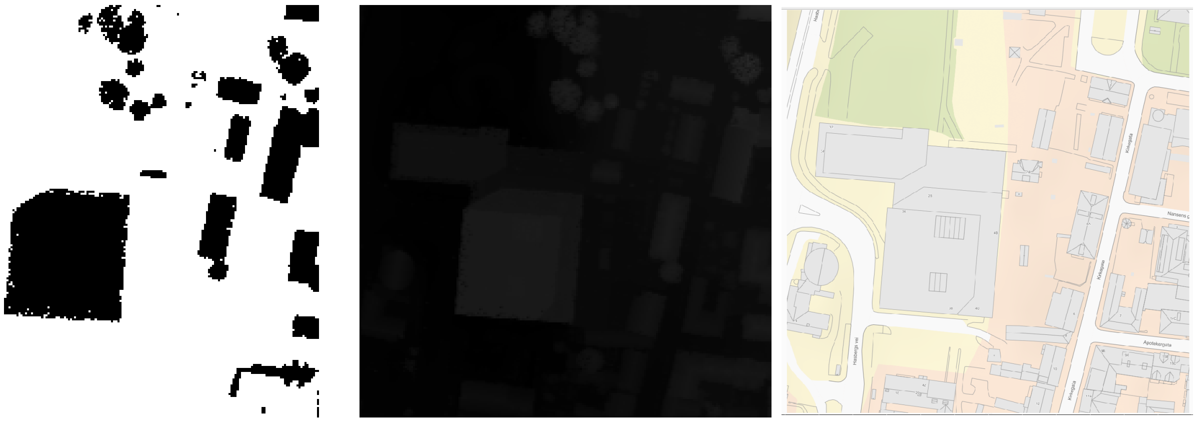
There are three important functions to test in the compile scenario component:

- Verify that the origin and size are correctly sent to the system so that all components that require them receive the correct information.
- Verify that the agents are sorted correctly based on timestamp and agent ID.
- Verify that the routes returned from the routemaker are assigned to the correct agent ID and placed in the appropriate position in the vector.

To check if the origin and size are correctly sent to the entire system, the map from the Map Manager is compared with the map in the Routemaker and the height data. The comparison can be seen in fig. 30. If the size is the same and it represents the same area, the compile scenario has successfully distributed the data correctly.

To verify that the keyframes are sorted correctly based on time and agent ID, the debug mode is used to step through the function that sorts the keyframes. This allows for checking that the vector is updated correctly and that all completed keyframes are placed in the correct position.

To check if the routes returned from the Routemaker are properly placed in the corresponding location in the map associated with the correct agent ID, the debug mode is used to verify



(a) Map from routemaker (b) Map form height manager (c) Map from map manager

Figure 30: Comparison of the map in Map Manager, height manager and routemaker.

that the returned route is placed in the same position in the map as the agent it belongs to. It is also possible to verify this from the GUI by adding several keyframes to more than one agent in different orders and compiling the scenario. Then the routes to each drone will be painted between the keyframes. Fig. 31 shows two agent with three keyframes each at different timestamps, generating their respective routes.

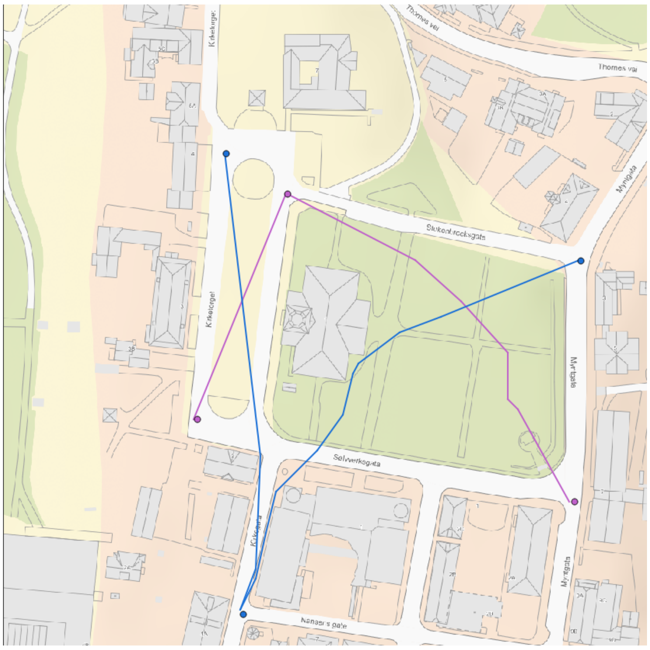


Figure 31: Compile scenario with multiple agents.

9.1.10 Automated tasks

RS | *HM*

As mentioned in section 6.2.7, Azure Pipelines were employed for automated tasks. This includes automatically publishing the website online whenever the repository of the website is updated, as well as automatically publishing the code documentation generated by Doxygen whenever the main branch of Hivemind is updated.

Testing these pipelines is simple, but time-consuming. Verifying that the pipelines were working properly was done by repeatedly running the pipelines, both by manually triggering them, and by updating different branches and verifying that they only automatically trigger on updates in the defined branches. Each time a pipeline is run, Microsoft spins up a new virtual machine on their servers, installs all packages and dependencies of Hivemind, and then performs the task. This is both computationally expensive and time-consuming. In hindsight, utilizing Azure's support of locally hosted pipeline agents would be much more efficient in terms of computational and time cost.

9.2 Testing of Hivemind

HMM | *NH*

When each component had been verified to function correctly in isolation, they were merged into the development branch of Hivemind for integration. After integrating these components with the Graphical User Interface, the formal testing of the software against requirements could occur. All requirements for the final product were tested and recorded in the document in appendix L.

9.2.1 Verification of the Graphical User Interface

HM | *HMM*

To test the Graphical user interface as a whole system, a scenario was made. The first step was to run the executable. When the GUI had finished loading the map, the following steps were taken to verify that the software operated as expected:

- Set location to the church in Kongsberg with size as 250 meters
- Added 3 agents
- Made 15 key frames(5 for each agent)
- Compiled scenario
- Deleted 5 keyframes(chosen at random)
- Compiled scenario
- Added 5 new keyframes(so that agent ended up with 5 each)
- Compiled scenario
- Saved scenario
- Restarted Hivemind
- Loaded the scenario
- Compiled scenario
- Added a fourth agent
- Deleted 5 Keyframes

- Saved scenario
- Closed Hivemind

From doing this some bugs where made visible. One bug occurred when loading a scenario and adding a new agent, wherein the new agent would get agentid 0 - the same id as previously existing agents. This is confusing to the user when working on an oldscenario.

Testing is an important part of the software development process. In Hivemind, testing was conducted continuously, with the final product thoroughly tested when all software components had been integrated with the GUI. The final two sections of this report will illustrate the risk analysis for the Hivemind software.

10 Product Risk Analysis

Thorough risk analysis was considered a vital part of Hivemind because of the client specifications that the software will control drones in the real world without their own sensors or ability to avoid obstacles, including human ones. For the sake of safety, it was crucial that risks and the severity of allowing unsafe and untested code to slip through the cracks into the final product was constantly kept in mind. You can find the risk analysis in appendix J.

10.1 Definitions and risk matrix

NH | *AM*

The logic behind the product is much the same as the project sik analysis. The definitions for the degree of consequence for the product are defined in the tab 6. While the likelihood of occurrence and the risk matrix itself is the same as in section 5.2 (fig. 6) (tab. 2).

Definition of degree of consequence for product	
Degree of consequence	Outcome
1 Insignificant	Product works as normal.
2 Small	Product stops working.
3 Considerable	Product stops working, and won't start working again, even with restart.
4 Serious	Products works, but not as intended.
5 Disastrous	Product stops working and drones start crashing.

Table 6: Consequence for product

10.2 Client interaction and risk identification

NH | *AM*

Risk was often discussed in the regular meetings with the external supervisor. As the scope of the project revealed itself, it was necessary to discuss on whether to include external risks, such as birds and weather, into risk considerations. As a result, the final risk matrices only include internal and external risks which Hivemind has some ability to influence. For example, birds are not included in the risk considerations.

On the other hand, the possibility of signal/GPS jamming will be included in the final risk analysis as a security consideration to be taken into account while operating Hivemind. Some external risks have not been added yet, as they concern the real-time part of the project which is not included in our scope of the project.

What has been added is a "future risks" section which is divided into "operational" and "software" risks. These are some of the risks that were considered, but ultimately fell outside the scope of the project. They should nevertheless be mentioned in the risk analysis.

10.3 Risk mitigation strategies

NH | *AM*

To address product risks, Hivemind has implemented several mitigation strategies, such as rigorous testing and validation of the software, ensuring that safety features are designed and implemented correctly. Hivemind has also committed to revisiting and expanding the risk analysis once the MVP was completed to ensure that all relevant risks are considered and appropriate mitigation strategies are in place, which was done as planned.

10.4 Encountered risks

NH | *AM*

The integration of RVIZ and Qt6 for 3D visualization proved to be more challenging than anticipated, primarily due to the limited documentation and support for seamless integration or viable workarounds. Despite employing extensive testing to mitigate this issue, none of the methods we attempted yielded a result.

Debugging Qt in the compilers we used posed difficulties, as breakpoints did not function as expected. The reason for this is that Qt generates code automatically, and the debuggers in Visual Studio Code and CLion do not account for this autogenerated code. This issue was encountered by multiple team members on separate occasions. However, after finishing the coding phase of the project, we discovered that debugging in Qt Creator would be an effective mitigation strategy. Qt Creator considers the autogenerated code, providing an effective debugging environment.

This chapter has briefly explained how product risk analysis was considered and absorbed into the software development process. Having completely covered all aspects of the Hivemind implementation, the next section will evaluate the success of Hivemind in meeting the requirements set at the onset of the project.

11 Evaluation

For Hivemind, there are two main aspects that merit evaluation. First, whether the software delivered in fact does what the client requested when they outlined the project. Second, how well the project model and work methods worked, and to what extent these supported a successful and untroubled software development process. This section will deal with these considerations.

11.1 Have we met the requirements?

HMM | *NH*

Tab. 7 shows a list of all the requirements for the minimum viable product and their completion statuses. It demonstrates that all the functionality that was agreed upon by the customer for the minimum viable product has been successfully implemented in Hivemind. In addition, three extra functions have been implemented: support for route planning of multiple agents, a timeline widget for the user interface that provides a more intuitive way of adding keyframes, and the visual representation of the calculated routes in the form of a line on the map.

Requirement	Status	Notes
GUI with tabs and options	Completed	
Save/load scenarios	Completed	
Map data query	Completed	
Height data query	Functional, but not completed	Height data is interchangeable, but not dynamically downloaded from internet
Coordinate system converter	Completed	
Basical graphical representation of key frames in GUI	Completed	
Single drone route planning	Completed	
Additional functionality		
Support for multiple agents	Functional, but not completed	Anti collision functionality has not been added, so these routes are not considered safe
Timeline	Completed	
Visual representation of calculated route	Completed	

Table 7: Table of revised requirements for product

Although the wording of some requirements for the minimum viable product have been adjusted due to changing the practical implementation of some functionality, the essence of these requirements remained the same. Some requirements and tests were also added for the final product because advanced, additional functionality was added. All of the requirements for the final version of Hivemind, including those for the minimum viable product, were met and tested. These can be studied in detail in appendix F. This document also shows the added and tested requirements for additional advanced functions that were added to the software after finishing the minimum viable product.

11.2 Practical evaluation

HMM | *NH*

In general, as is evidenced from the previous section, the group's agile working methods enabled it to face most of the practical challenges that appeared during the course of the project. The short sprint lengths meant that there was a continuous evaluation of whether or not the project was headed in the right direction, and whether the group was focusing on the right things. This also meant that the project itself was continuously evaluated and adjusted, which is evident from the development seen in the documentation such as the architecture.

When members were absent or fell ill, it was easy to adjust the tasks to be completed in the sprint and re-assign work to make sure everything was completed. The steady progress, and meeting the most important deadlines at first and second presentations, along with semi-regular team building and cake celebrations helped keep the group motivated and positive.

Finally, great organization and thorough planning made sure the project progressed smoothly despite many regular absences. In fact, the project has largely been able to follow the timeline that was proposed for the first presentation, showing that the entire process has been well-thought out and planned bearing in mind what the group could realistically achieve.

12 Conclusion

12.1 Challenges

HMM | *NH*

Hivemind faced a number of challenges throughout the project. These have been divided into academic and administrative challenges. The academic challenges were ones related to technical work and requirements. A number of these have already been mentioned briefly in previous sections, but this section will re-iterate these. The administrative challenges faced by Hivemind generally involve those that made co-operation and meeting deadlines more difficult, such as absences and illness. They will be described in this section.

12.1.1 Academic challenges

HMM | *NH*

Distilling requirements Hivemind was not presented with a neat list of requirements for the project. Instead, these were distilled over a long time period, starting with the first planning meeting with the client/external supervisor. A large amount of effort was expended to develop clear and concise requirements, which included numerous meeting with the client to clarify their requests and tailoring already-defined requirements to the client's actual needs. Work on requirements also relates to the very specific table of requirements, and tests which was devised.

Scope of project Initially, the scope of the project was very large, including not only the planning functionality, but also simulation and real-time control and monitoring modes. Early on in the project, this meant that the group was spread a little bit thin trying to research and plan out all the different components that would need to be implemented.

Around one month into the project, it gradually became clear that the scope of the project had become too large to realistically (and responsibly) be completed. Through discussions with the group's external supervisor (and client), the large-scale planned software was etched down to the minimum viable product that was ultimately delivered.

Changing the direction of the project after one month of a five month project run poses a significant challenge, especially when considering that this could mean having to discard already completed work that is no longer relevant for the new minimum viable product. Luckily, the group handled this flexibly and was able to re-purpose most of the work already completed. In fact, perhaps the only piece of documentation that does not hold a function in the minimum viable product are the requirements that are only relevant for the whole, complete software. Even this, however, has a use in giving the future developers of Hivemind an idea of how the advanced functionality can be tested and what it should do. This demonstrates that the group was able to pivot from one direction of the project to another, and that the work already done was done in such a way that it was useful regardless of the scope of the project.

Determining a software development methodology and developing a flexible software model This project posed a particularly difficult challenge in terms of software modelling, as the software needed to be able to handle constantly changing environment. The core software should remain resilient and unchanging regardless of what other functionalities were added, while still allowing for extensions when new technologies or methods are to be combined with Hivemind. Because of this, it was crucial to determine a software development model that was uniquely suited to the problem solved in Hivemind.

Similarly, a very large section of the project was used to develop this flexible software model. This is indicative of the large challenge related to software development and the creation of the software model, as numerous different iterations were created before the final, stable version of Hivemind's architecture was finished.

Difficulty using the shared development environment Because it was envisioned early on that the Hivemind software would need to run on a Linux system to make use of the Robot Operating System (ROS), it was agreed that the project would also be developed in Linux. Unfortunately, many in the group only had Windows computers, which led to the necessity of setting up identical virtual machines. The hardware limitations of many in the group led to many virtual machines to run slowly, triggering some members to exchange their operating systems with Linux, while others wrote code and compiled code in a very relaxed pace.

This also meant that whenever any member downloaded and installed a new library to use in their components, every other member would need to install this in their machine to properly run the Hivemind software for testing. Often, installing many of these libraries was a challenge in itself (which will be detailed in the next subsection), leading to errors and large amounts of frustration for members. This particular challenge was solved by Ruben, who took upon himself the role as general IT support and wrote scripts to install all dependencies and build the development environment, making it easy for the members to update their libraries.

Finally, CMake is used to build the software. This is a new technology for many of the group members, and is not very intuitive to use without some guidance. Some members used tens of hours trying to successfully install new libraries and include them with CMake. Again, this challenge was generally addressed with scripts and assistance from Ruben.

Poorly documented libraries One significant challenge technically in this project was installing and successfully being able to use methods of the many libraries necessary in Hivemind, such as GeographicLib, GDAL and RapidJSON.

In many cases, successfully importing these into the integrated development environment was a result of trying and failing, intensive online research and caffeine. For many of these libraries, the documentation of installing and importing is sadly lacking, or very inaccessible to people not very familiar with working in this way already. In the case of GDAL, for example, this was only successfully imported into Hivemind after looking up the source code of GDAL's own implementation of itself, to see how their CMake file was configured.

Unfortunately, there was no smart way of overcoming these obstacles, but instead was a good exercise in tenacity. This exercise is a good demonstration of the excellent work ethic of the Hivemind team, and how hard work can solve difficult technical challenges.

12.1.2 Administrative challenges

HMM | *NH*

Ruben's absence at start of project At the start of the project, Ruben was still completing his exchange semester in Belgium. He was physically not present in Kongsberg for about the first month of the project. This led to some logistical challenges in terms of group meeting participation and co-operation.

While it is always the most ideal to be able to partake in meetings fully in-person, we nevertheless addressed this problem by (1) planning the project early, (2) having regular Zoom meetings to discuss our progress and the way forward and (3) making sure the tasks Ruben was assigned to do the first month were tasks he could complete entirely by himself without much input from anyone else.

The group was formed before the summer of 2022, which means we were able to gradually agree on working methods, project methodology and other organizational details before the project period formally started in January 2023. As a result, the group already knew how it wanted to work, which values it wanted to follow, and which roles each member would have before Ruben even left for Belgium. As a side note, Nils Herman was not part of the group at this time, but joined when the project started in earnest January 2023.

Zoom meetings were decided early on as crucial for smoothly progressing the project while Ruben was away. We had daily stand-up meetings on Zoom so Ruben always knew what the rest of the group was working on, while he was also able to update the group on his own progress. Semi-regular social activities on Fridays was early on decided to be desireable for team-building purposes, and while Ruben was in Belgium, these were also digital, with activities such as quizzes and GeoGuessr tournaments. Ruben also participated online in the weekly Sprint planning and retrospective meetings.

Finally, Ruben was assigned tasks that were suitable to complete alone. This included the work of determining coding standards, some research and starting to set up the group development environment.

Regular absences Two group members in particular had regular or semi-regular absences. Hilde Marie worked part-time in Oslo throughout the project period (in general, two days a week), while Nils Herman had irregular days where he was partially or entirely absent due to work or engagements in various volunteer organizations. The main challenges related to this are two-fold.

First, the bachelor's project is highly co-operative, and absences from core hours can impair co-operation. In many cases, even though Hilde Marie and Nils Herman were able to make up the time they missed, their tasks could have been more efficiently completed with the help of other group members in a collaborative setting. At the same time, the rest of the group missed out on helpful input or assistance from these two members while they were away. Many of the programming tasks or technical work that needed to be done was co-operative in nature, for example in the case where two components are highly intertwined.

Second, having to catch up with project work in the evenings after work or during the weekends instead of resting, in addition to commuting, was an additional source of stress for the members that had to do this. This was something that needed to be closely monitored in order to be able to address this stress before it lead to burnout.

The first problem was dealt with through planning and flexibility. In the case of Hilde Marie, the days she worked was decided together with the group ahead of time so her absences were predictable. Tuesday and Thursday were chosen, so Hilde Marie was able to attend the Sprint planning on Monday and the final stand-up meeting on Friday, in addition to the group's regular meetings with supervisors on Wednesday and Friday. In addition, while she co-operated with the group when she was present in school, the tasks she was assigned were in general tasks she could complete from home during the weekends if necessary. The HeightMap components interfaces with few other components, and was therefore an excellent choice for individual work. She also took responsibility for the literature review, which could successfully be completed anywhere with little input from the rest of the group.

For Nils Herman, his absences were fewer and more irregular. The group maintained flexibility through frequent communication on progress and problems using the Discord channel, and Nils Herman participated on meetings through Zoom when possible. The group established a "Help Me" channel on Discord which ensured it was easy for group members to ask for help and to assist each other remotely. Nils Herman was also given tasks that could be completed somewhat individually, such as devising the scheme for the project and product risk analysis, and developing the timeline component.

Sickness There were two significant bouts of illness that affected the Hivemind project. The first was sickness at the start of the project, where all members fell ill at different times within a three-week period, leading to bouts of sickness of anything between one day and two weeks. The second was Ruben falling ill the week of the final technical sprint.

The first bout of illness affected each individual differently, with the member falling ill the longest being ill for two weeks. In general, this did not affect the progress of the project as a whole for several reasons. First, because the project had weekly sprint planning meetings and daily stand-up meetings, it was easy to re-assign work and to see when someone else needed to take over work for someone feeling ill. Second, as it was early on in the project and most of the work being done was related to research and planning the product, there was some leeway in when tasks needed to be done, allowing the group to accommodate for the absences. Finally, because not everyone was sick at the same time, and because most of the group was indisposed for less than a week, the group still had resources to complete their tasks.

The second bout of illness could have severely impacted the completion of the project, or shifted the timeline of the project, leading to less time to write the report. One member fell ill during the final technical sprint, which was a potential challenge because their tasks needed to be finish before the project could be considered complete. This hurdle was passed without much turbulence, however, as other members were able to take over this member's tasks, and the rest of the tasks to be done were adjusted to make sure Hivemind could be completed. As a result, the timeline for the project did not need to be shifted, and the advanced functionality of the software was able to be integrated into the software as hoped.

Three members from the same family One important concern from the start of the project was to make sure Ruben and Nils Herman felt comfortable in the group and not like outsiders, despite the fact that Aurora, Harald and Hilde Marie are cousins. This was something the entire group was reminded of early on in the project, and implementing semi-regular team building sessions was one method meant to target this potential issue. As a final evaluation of this, however, it is important to note that Ruben and Nils Herman never felt like outsiders. This demonstrates this challenge was very successfully addressed.

12.2 Future work

Although Hivemind is fully functional, there are a number of features that could be added to the route planning mode to improve the user experience. In addition, there are two more modes to be added before Hivemind can be considered "completed". This section will address these.

12.2.1 Planning mode

HMM | *NH*

The planning mode still lacks dynamically updated height data. When envisioned, the Vertex Manager component would be able to query a WCS when the user selected a new origin coordinate and download the selected sized map around this origin point. As detailed in the section for the Vertex Manager, the group was unable to devise a way to open the GML file, in part due to file constraints. Though the software will function perfectly fine using manually downloaded and selected GeoTIFF files, being able to dynamically update the height data given an origin point and size would greatly improve the user experience.

As of right now, when calculating routes for multiple agents, the routemaker makes sure each agent avoids the surrounding terrain and buildings, but not other agents. As a result, the scenario cannot be considered safe to test with real drones. Any future work on Hivemind should improve additional work on the algorithm to calculate routes that will not intersect close enough in time that a collision is likely. One way to accomplish this would be for the routemaker to mark the cells between keyframes as occupied in the time interval the keyframes define. This way, when planning routes for other agents, those cells could be avoided.

A number of visual tweaks remains to be done to the GUI. In particular, the GUI would be easier to use if (1) the colours on the timeline corresponded to the different colour of each agent involved in the scenario and (2) the different agents were placed at different heights on the timeline, so it is easy to distinguish between them.

At present, if creating a scenario, all keyframes are saved as relative points to the selected origin. If the origin is changed after creating keyframes, the routes calculated by compiling the scenario will be incorrect. This means a user would need to delete all keyframes present after changing the origin, posing a problem if an operator plans a route and tests it at one safe geographic location and then brings it somewhere else to perform a light show.

While planning, coordinates occupied by buildings, trees and other terrain are not highlighted, which means it is not evident to the planner if keyframes will hit anything before scenario is compiled. Improvements to the GUI necessarily includes a visual representation of obstacles based on the HeightMap.

In addition, though the Load scenario and Save as buttons from the file menu in the GUI function as intended, the Save and New project buttons are still dummies that do nothing when clicked. When continuing development on the software, adding functionality to the Save and New project buttons will ensure making multiple projects is a more streamlined experience for the user.

Finally, at present, the timeline only covers 100 seconds. This is not dynamic and cannot be changed without editing the source code. To make it easy to create longer shows by using the graphical user interface, Hivemind should be augmented to allow for selecting longer lengths and dynamically updating the timeline.

12.2.2 Simulation and Launch modes **HMM** | *NH*

The Hivemind Simulation and Launch modes were not implemented during this run of the project. Both of these functions are integral to creating the software that the client initially envisioned. In terms of Simulation, some work has already been done looking at 3D visualisation, whose lessons can be applied to expanding the Hivemind software.

The Launch mode will require further knowledge of the drones that will ultimately be used with the software, and will need to implement a large number of components to handle different sensor data such as battery status, positional data, as well as adding emergency landing and dynamic re-routing functionality.

12.3 Contribution **HMM** | *NH*

The primary goal of this project was to produce a piece of software that could be of use to the client. Because it was determined that there would be insufficient time to complete all of the planning, simulation and launch modes during the project period, the focus was therefore to produce route planning software with a functional planning mode, and that was scalable and flexible enough that the client could add the rest of the functionality at a later time without making significant changes to the software backbone. In this regard, the project has succeeded and contributed towards the goal of the client of having their own proprietary route planning software for light shows.

The software has a fully functional planning mode, which can program routes for a practically unlimited amount of agents. The resulting routes take into consideration the local terrain, buildings and trees, and will therefore be able to avoid collisions with static objects. Although the software takes Kongsberg as a starting point (as this was part of the client's original request), the map used by the software can be easily exchanged by a user, and any new GeoTIFF file of Norway will function with the software given the coordinates are UTM33N. The GUI is intuitive and user-friendly, with visualisations of the route planned and an interactive timeline.

Finally, great care has been taken in the design process of the software, which has ensured that the architecture allows for the addition of further functionality without having to change any of the existing components.

In terms of academics, Hivemind may not have provided any new insights itself through experiments and simulations, but it does provide a novel flexible, open-source solution that can be augmented by future researchers interested in a variety of different topics related to UAVs. Hivemind is perfectly designed so that new technologies in terms of software and drones can easily be added to the software. Finally, the architecture of Hivemind is different from many previously proposed as the algorithm that the route planning software is run on is easily replaced. In addition to the intended functionality for light shows, some applications for extensions of Hivemind could include comparison of different algorithms, a planning software for real routes (given the agents that will fly them have their own anti-collision solutions) or functionality added for agriculture and cinematography.

References

[1] S. of Automotive Engineers International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles." [Online]. Available: https://www.sae.org/standards/content/j3016_202104/

[2] E. Anderson, T. Fannin, and B. Nelson, "Levels of aviation autonomy," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, 2018, pp. 1–8.

[3] H.-M. Huang, E. R. Messina, and J. S. Albus, "Toward a generic model for autonomy levels for unmanned systems (alfus)," *NIST*, Aug 2003. [Online]. Available: https://www.nist.gov/publications/toward-generic-model-autonomy-levels-unmanned-systems-alfus

[4] F. Hu, X.-L. Huang, and D. Ou, *Communication Topology Analysis upon a Swarm of UAVs: A Survey*, 1st ed. Boca Raton; London; New York: CRC Press, 2021.

[5] [Online]. Available: https://earth-info.nga.mil/index.php?dir=coordsys&action=coordsys

[6] "json," 1999. [Online]. Available: https://www.json.org/json-en.html

[7] [Online]. Available: https://earth-info.nga.mil/index.php?dir=coordsys&action=coordsys

[8] [Online]. Available: https://www.geonorge.no/en/references/references/coordiante-systems/

[9] [Online]. Available: https://earth-info.nga.mil/index.php?dir=wgs84&action=wgs84

[10] J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*, ser. The Morgan Kaufmann series in evolutionary computation. San Francisco: Morgan Kaufmann Publishers, 2001.

[11] "Swarm definition," Apr 2023. [Online]. Available: https://www.merriam-webster.com/dictionary/swarm

[12] E. Teague and R. H. Kewley, "Swarming unmanned aircraft systems," *Defense Technical Information Center*, 9 2008. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA489366

[13] P. Cybulski, "A framework for autonomous uav swarm behavior simulation," in *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2019, pp. 471–478.

[14] M. C. Jeffrey, S. Subramanian, C. Yan, J. Emer, and D. Sanchez, "A scalable architecture for ordered parallelism," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 228–241.

[15] R. Arnold, K. Carey, B. Abruzzo, and C. Korpela, “What is a robot swarm: A definition for swarming robotics,” in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2019, pp. 0074–0081.

[16] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-de Dios, and A. Ollero, “Experimental results in multi-uav coordination for disaster management and civil security applications,” *Journal of Intelligent & Robotic Systems*, vol. 61, no. 14, p. 563585, Jan 2011. [Online]. Available: http://link.springer.com/10.1007/s10846-010-9497-5

[17] H. X. Pham, H. M. La, D. Feil-Seifer, and M. C. Deans, “A distributed control framework of multiple unmanned aerial vehicles for dynamic wildfire tracking,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1537–1548, 2020.

[18] W. Zhou, Z. Liu, J. Li, X. Xu, and L. Shen, “Multi-target tracking for unmanned aerial vehicle swarms using deep reinforcement learning,” *Neurocomputing*, vol. 466, pp. 285–297, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221014223

[19] N. A. Kyriakakis, M. Marinaki, N. Matsatsinis, and Y. Marinakis, “Moving peak drone search problem: An online multi-swarm intelligence approach for uav search operations,” *Swarm and Evolutionary Computation*, vol. 66, p. 100956, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650221001188

[20] Z. Zhen, D. Xing, and C. Gao, “Cooperative search-attack mission planning for multi-uav based on intelligent self-organized algorithm,” *Aerospace Science and Technology*, vol. 76, pp. 402–411, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963817301736

[21] M.-H. Kim, H. Baik, and S. Lee, “Response threshold model based uav search planning and task allocation,” *Journal of Intelligent & Robotic Systems*, vol. 75, no. 3, p. 625640, Sep 2014. [Online]. Available: https://doi.org/10.1007/s10846-013-9887-6

[22] D. Hambling, “The us navy wants swarms of thousands of small drones.” [Online]. Available: https://www.technologyreview.com/2022/10/24/1062039/us-navy-swarms-of-thousands-of-small-drones/

[23] K. C. W. Goh, R. B. C. Ng, Y.-K. Wong, N. J. H. Ho, and M. C. H. Chua, “Aerial filming with synchronized drones using reinforcement learning,” *Multimedia Tools and Applications*, vol. 80, no. 12, p. 1812518150, May 2021. [Online]. Available: https://doi.org/10.1007/s11042-020-10388-5

[24] X. Jin, S. Liu, F. Baret, M. Hemerlé, and A. Comar, “Estimates of plant density of wheat crops at emergence from very low altitude uav imagery,” *Remote Sensing of Environment*, vol. 198, pp. 105–114, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425717302651

[25] P. Jiang, X. Zhou, T. Liu, X. Guo, D. Ma, C. Zhang, Y. Li, and S. Liu, "Prediction dynamics in cotton aphid using unmanned aerial vehicle multispectral images and vegetation indices," *IEEE Access*, vol. 11, pp. 5908–5918, 2023.

[26] R. Gupta, P. Bhattacharya, S. Tanwar, N. Kumar, and S. Zeadally, "Garuda: A blockchain-based delivery scheme using drones for healthcare 5.0 applications," *IEEE Internet of Things Magazine*, vol. 4, no. 4, pp. 60–66, 2021.

[27] A. Staff, Jun 2022. [Online]. Available: https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries

[28] S. J. Plathottam and P. Ranganathan, "Next generation distributed and networked autonomous vehicles: Review," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, 2018, pp. 577–582.

[29] W. Chen and Z. Jin, *1. Communication Topology Analysis upon a Swarm of UAVs: A survey.* Boca Raton: CRC Press, Dec 2020.

[30] A. Bürkle, F. Segor, and M. Kollmann, "Towards autonomous micro uav swarms," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, p. 339353, Jan 2011. [Online]. Available: https://doi.org/10.1007/s10846-010-9492-x

[31] lker Bekmezci, O. K. Sahingoz, and amil Temel, "Flying ad-hoc networks (fanets): A survey," *Ad Hoc Networks*, vol. 11, no. 3, pp. 1254–1270, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870512002193

[32] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: a review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, 2019. [Online]. Available: https://doi.org/10.1139/juvs-2018-0009

[33] S.-C. Choi, J.-H. Park, and J. Kim, "A networking framework for multiple-heterogeneous unmanned vehicles in fanets," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, 2019, pp. 13–15.

[34] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 2534, aug 1987. [Online]. Available: https://doi.org/10.1145/37402.37406

[35] J. B. Clark and D. R. Jacques, "Flight test results for uavs using boid guidance algorithms," *Procedia Computer Science*, vol. 8, pp. 232–238, 2012, conference on Systems Engineering Research. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091200049X

[36] J. Wu, C. Luo, Y. Luo, and K. Li, "Distributed uav swarm formation and collision avoidance strategies over fixed and switching topologies," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 969–10 979, 2022.

[37] W. Ren, "Consensus based formation control strategies for multi-vehicle systems," in *2006 American Control Conference*, 2006, pp. 6 pp.–.

[38] X. Dong, B. Yu, Z. Shi, and Y. Zhong, "Time-varying formation control for unmanned aerial vehicles: Theories and applications," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 340–348, 2015.

[39] X. Ge and Q.-L. Han, "Distributed formation control of networked multi-agent systems using a dynamic event-triggered communication mechanism," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 10, pp. 8118–8127, 2017.

[40] H. Liu, T. Ma, F. L. Lewis, and Y. Wan, "Robust formation control for multiple quadrotors with nonlinearities and disturbances," *IEEE Transactions on Cybernetics*, vol. 50, no. 4, pp. 1362–1371, 2020.

[41] W. Jasim and D. Gu, "Robust team formation control for quadrotors," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1516–1523, 2018.

[42] X. Liang, Y.-H. Liu, H. Wang, W. Chen, K. Xing, and T. Liu, "Leader-following formation tracking control of mobile robots without direct position measurements," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 4131–4137, 2016.

[43] X. You, C. Hua, and X. Guan, "Self-triggered leader-following consensus for high-order nonlinear multiagent systems via dynamic output feedback control," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2002–2010, 2019.

[44] K.-H. Tan and M. Lewis, "Virtual structures for high-precision cooperative mobile robotic control," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, vol. 1, 1996, pp. 132–139 vol.1.

[45] D. Zhou, Z. Wang, and M. Schwager, "Agile coordination and assistive collision avoidance for quadrotor swarms using virtual structures," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 916–923, 2018.

[46] [Online]. Available: https://www.kongsberg.com/kda/Who-we-are/200-years-of-excellence/

[47] [Online]. Available: https://www.kongsberg.com/no/newsandmedia/news-archive/20202/forsvarsmateriell-velger-kongsberg-for-leveranse-av-nytt-taktisk-radiolinje-system/

[48] J. Finborud, "Kongsberg gruppen sikrer ny milliardavtale med det amerikanske forsvaret," May 2021. [Online]. Available: https://www.finansavisen.no/nyheter/industri/2021/05/11/7671965/kongsberg-gruppen-sikrer-ny-milliardavtale-med-det-amerikanske-forsvaret

[49] S. Zhu and D. Levinson, "Do people use the shortest path? an empirical test of wardrops first principle," *PLOS ONE*, vol. 10, no. 8, p. e0134322, Aug 2015. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0134322

[50] C. Bongiorno, Y. Zhou, M. Kryven, D. Theurel, A. Rizzo, P. Santi, J. Tenenbaum, and C. Ratti, "Vector-based pedestrian navigation in cities," *Nature Computational Science*, vol. 1, no. 10, p. 678685, Oct 2021. [Online]. Available: https://www.nature.com/articles/s43588-021-00130-y

[51] A. Basiri, V. Mariani, G. Silano, M. Aatif, L. Iannelli, and L. Glielmo, "A survey on the application of path-planning algorithms for multi-rotor uavs in precision agriculture," *The Journal of Navigation*, vol. 75, no. 2, p. 364383, 2022.

[52] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[53] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, vol. 39, p. 533579, Oct 2010, arXiv:1401.3843 [cs]. [Online]. Available: http://arxiv.org/abs/1401.3843

[54] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, p. 269271, Dec 1959. [Online]. Available: http://link.springer.com/10.1007/BF01386390

[55] Y. Wang, X. Liang, B. Li, and X. Yu, "Research and implementation of global path planning for unmanned surface vehicle based on electronic chart," in *Advances in Intelligent Systems and Computing*. Springer International Publishing, nov 2017, pp. 534–539. [Online]. Available: https://doi.org/10.1007%2F978-3-319-65978-7_80

[56] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[57] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, p. 378400, May 2001. [Online]. Available: http://journals.sagepub.com/doi/10.1177/02783640122067453

[58] X. Li, Y. Zhao, J. Zhang, and Y. Dong, "A hybrid pso algorithm based flight path optimization for multiple agricultural uavs," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2016, pp. 691–697.

[59] H. S. Dewang, P. K. Mohanty, and S. Kundu, "A robust path planning for mobile robot using smart particle swarm optimization," *Procedia Computer Science*, vol. 133, pp. 290–297, 2018, international Conference on Robotics and Smart Manufacturing (RoSMa2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918309815

[60] X. Peng, F. Guan, Z. Wang, and S. Gao, "Simulation and optimization of multi-uav route planning based on hybrid particle swarm optimization," in *Advances in Simulation and Process Modelling*, Y. Li, Q. Zhu, F. Qiao, Z. Fan, and Y. Chen, Eds. Singapore: Springer Singapore, 2021, pp. 223–232.

[61] C.-L. Huo, T.-Y. Lai, and T.-Y. Sun, "The preliminary study on multi-swarm sharing particle swarm optimization: Applied to uav path planning problem," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 1770–1776.

[62] A. Puente-Castro, D. Rivero, A. Pazos, and E. Fernandez-Blanco, "A review of artificial intelligence applied to path planning in uav swarms," *Neural Computing and Applications*, vol. 34, no. 1, p. 153170, Jan 2022. [Online]. Available: https://link.springer.com/10.1007/s00521-021-06569-4

[63] L. He, N. Aouf, and B. Song, "Explainable deep reinforcement learning for uav autonomous path planning," *Aerospace Science and Technology*, vol. 118, p. 107052, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963821005629

[64] S. Perez-Carabaza, E. Besada-Portas, J. A. Lopez-Orozco, and J. M. de la Cruz, "Ant colony optimization for multi-uav minimum time search in uncertain domains," *Applied Soft Computing*, vol. 62, pp. 789–806, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617305483

[65] G. Zhang, R. Wang, H. Lei, T. Zhang, W. Li, and Y. Song, "Uav path planning based on variable neighborhood search genetic algorithm," in *Advances in Swarm Intelligence*, Y. Tan and Y. Shi, Eds. Cham: Springer International Publishing, 2021, pp. 205–217.

[66] S. Aggarwal and N. Kumar, "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges," *Computer Communications*, vol. 149, pp. 270–299, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419308539

[67] X. Luo, X. Li, Q. Yang, F. Wu, D. Zhang, W. Yan, and Z. Xi, "Optimal path planning for uav based inspection system of large-scale photovoltaic farm," in *2017 Chinese Automation Congress (CAC)*, 2017, pp. 4495–4500.

[68] B. M. Sathyaraj, L. C. Jain, A. Finn, and S. Drake, "Multiple uavs path planning algorithms: a comparative study," *Fuzzy Optimization and Decision Making*, vol. 7, no. 3, p. 257267, Sep 2008. [Online]. Available: https://doi.org/10.1007/s10700-008-9035-0

[69] F. Haro and M. Torres, "A comparison of path planning algorithms for omni-directional robots in dynamic environments," in *2006 IEEE 3rd Latin American Robotics Symposium*, 2006, pp. 18–25.

[70] UgCS, "UgCS Commander: Manage multiple UAVs." [Online]. Available: https://www.ugcs.com/page/ugcs-commander-manage-multiple-uavs

[71] DJI, "Flight planning software for dji drones." [Online]. Available: https://www.djiflightplanner.com/

[72] Orbit Logic. [Online]. Available: https://www.orbitlogic.com/uav-planner.html

[73] DroneDeploy, "Desktop flight planning." [Online]. Available: https://help.dronedeploy.com/hc/en-us/articles/1500004861101-Desktop-Flight-Planning

[74] QGroundControl, "Qgroundcontrol user guide." [Online]. Available: https://docs.qgroundcontrol.com/master/en/index.html

[75] PaparazziUAV, "Overview - paparazziuav." [Online]. Available: https://wiki.paparazziuav.org/wiki/Overview

[76] Y. Bouzid, Y. Bestaoui, and H. Siguerdidjane, "Guidance-control system of a quadrotor for optimal coverage in cluttered environment with a limited onboard energy: Complete software," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 2, p. 707730, Aug 2019. [Online]. Available: https://doi.org/10.1007/s10846-018-0914-5

[77] N. Aliane, C. Q. G. Muñoz, and J. Sánchez-Soriano, "Web and matlab-based platform for uav flight management and multispectral image processing," *Sensors*, vol. 22, no. 11, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/11/4243

[78] S. Romaniuk, Z. Gosiewski, and L. Ambroziak, "A ground control station for the uav flight simulator," *Acta Mechanica et Automatica*, vol. 10, no. 1, pp. 28–32, 2016. [Online]. Available: https://doi.org/10.1515/ama-2016-0005

[79] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, "A ground control station for a multi-uav surveillance system," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1, p. 119130, Jan 2013. [Online]. Available: https://doi.org/10.1007/s10846-012-9759-5

[80] O. Zedadra, C. Savaglio, N. Jouandeau, A. Guerrieri, H. Seridi, and G. Fortino, "Towards a Reference Architecture for Swarm Intelligence-based Internet of Things," in *International Conference on Internet and Distributed Computing Systems*, Mana Island, Fiji, Dec. 2017. [Online]. Available: https://hal.science/hal-02317288

[81] . Madridano, A. Al-Kaff, P. Flores, D. Martín, and A. de la Escalera, "Software architecture for autonomous and coordinated navigation of uav swarms in forest and urban firefighting," *Applied Sciences*, vol. 11, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/3/1258

[82] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995, pp. 127–134.

[83] "What is CI/CD?" may 2022. [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[84] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. PP, 03 2017.

[85] “What are the four fundamental methods of requirement verification?” [Online]. Available: https://www.modernanalyst.com/Careers/InterviewQuestions/tabid/128/ID/1168/What-are-the-four-fundamental-methods-of-requirement-verification.aspx

[86] “Unit test,” 2013. [Online]. Available: https://wiki.c2.com/?UnitTest

[87] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed. MTM, 2013.

[88] C. Feldmann, *The practical Guide to Business Process Reengineering Using IDEFO*. Dorset House Publishing, 2013.

[89] “GDAL Documentation.” [Online]. Available: https://gdal.org/

[90] Library of Congress, “Sustainability of digital formats: Planning for library of congress collections,” May 2020. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000279.shtml

[91] “Raster bands.” [Online]. Available: https://help.arcgis.com/en/geodatabase/10.0/sdk/arcsde/concepts/rasters/entities/rasterbands.htm

[92] Open Geospatial Consortium, “Ogc geotiff standard,” Sep 2019. [Online]. Available: https://docs.ogc.org/is/19-008r4/19-008r4.html

[93] Kartverket, “Høydedata laserinnsyn2.” [Online]. Available: https://hoydedata.no/LaserInnsyn2/

[94] “GDALDriver class.” [Online]. Available: https://gdal.org/api/gdaldriver_cpp.html

[95] “GDAL Geotransform Tutorial.” [Online]. Available: https://gdal.org/tutorials/geotransforms_tut.html

[96] C. Karney, “Geographiclib.” [Online]. Available: https://geographiclib.sourceforge.io/2009-03/index.html

[97] C. F. Karney, “Transverse mercator with an accuracy of a few nanometers,” *Journal of Geodesy*, vol. 85, no. 8, p. 475485, 2011.

[98] “Googletest,” 2023. [Online]. Available: http://google.github.io/googletest/

[99] “Bootstrap documentation.” [Online]. Available: https://getbootstrap.com

[100] “Qgis documentation.” [Online]. Available: https://docs.qgis.org/3.28/en/docs/index.html

[101] N. Nolde, "Qgis 3 plugin tutorial - qt designer explained " gisops," Jan 2021. [Online]. Available: https://gis-ops.com/qgis-3-plugin-tutorial-qt-designer-explained/

[102] "Robot operating system." [Online]. Available: https://www.ros.org/

[103] "Rviz." [Online]. Available: http://wiki.ros.org/rviz

[104] "Point clouds." [Online]. Available: https://pointclouds.org/

[105] PointCloudLibrary, "Find_qt.cmake - pointcloudlibrary/pcl." [Online]. Available: https://github.com/PointCloudLibrary/pcl/blob/master/cmake/pcl_find_qt.cmake

[106] "Åpent API for høyde- og dybdedata fra Kartverket." [Online]. Available: https://ws.geonorge.no/hoydedata/v1/

[107] "Tiff library and utilities." [Online]. Available: http://www.libtiff.org/

[108] "Rasterio Documentation." [Online]. Available: https://rasterio.readthedocs.io/en/stable/

[109] "Nasjonal høydemodell Digital overflatemodell 25833 WCS." [Online]. Available: https://kartkatalog.geonorge.no/metadata/nasjonal-hoeydemodell-digital-overflatemodell-25833-wcs/e36ea427-13a1-4d7c-be82-977068dfc3e3

[110] "Brukerveiledning." [Online]. Available: https://www.geonorge.no/aktuelt/om-geonorge/brukerveiledning/

[111] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[112] C. Flanagan, "The bresenham line-drawing algorithm." [Online]. Available: https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

[113] "Programming concepts: Type introspection and reflection," 2016. [Online]. Available: https://thecodeboss.dev/2016/02/programming-concepts-type-introspection-and-reflection/

[114] "Serialization and unserialization," 2023. [Online]. Available: https://isocpp.org/wiki/faq/serialization

[115] "Boost serialization," 2004. [Online]. Available: https://www.boost.org/doc/libs/1_81_0/libs/serialization/doc/index.html

[116] "Rapidjson," 2015. [Online]. Available: http://rapidjson.org/

[117] "A practical guide to cplusplus serialization," 2011. [Online]. Available: https://www.codeproject.com/articles/225988/A-practical-guide-to-Cplusplus-serialization

[118] "Dom," 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[119] [Online]. Available: https://www.usgs.gov/faqs/how-much-distance-does-a-degree-minute-and-second-cover-your-maps

[120] [Online]. Available: https://kart.gulesider.no/

[121] "Azure pipeline documentation," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops

Bibliography

[Bib1] H. Hamann, *Swarm Robotics: A Formal Approach*, 2018.

[Bib2] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019.

Appendices

Appendix A

GML file returned by WCS request

```
--wcs
Content-Type: text/xml
Content-ID: GML-Part
  <gml:boundedBy>
    <gml:Envelope srsName="http://www.opengis.net/def/crs/EPSG/0/25833" axisLabels="x y"
    ↪  uomLabels=" " srsDimension="2">
      <gml:lowerCorner>197332.000000 6624844.000000</gml:lowerCorner>
      <gml:upperCorner>200335.000000 6627847.000000</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <gml:domainSet>
    <gml:RectifiedGrid dimension="2" gml:id="grid_Coverage1">
      <gml:limits>
        <gml:GridEnvelope>
          <gml:low>0 0</gml:low>
          <gml:high>499 499</gml:high>
        </gml:GridEnvelope>
      </gml:limits>
      <gml:axisLabels>band_1</gml:axisLabels>
      <gml:origin>
        <gml:Point gml:id="grid_origin_Coverage1"
        ↪  srsName="http://www.opengis.net/def/crs/EPSG/0/25833">
          <gml:pos>197332.000000 6624844.000000</gml:pos>
        </gml:Point>
      </gml:origin>
      <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/25833">6.006000 0
      ↪  </gml:offsetVector>
      <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/25833">0 6.006000
      ↪  </gml:offsetVector>
    </gml:RectifiedGrid>
  </gml:domainSet>
  <gml:rangeSet>
    <gml:File>
      <gml:rangeParameters xlink:href="cid:Coverage1.tif"
      ↪  xlink:role="http://www.opengis.net/spec/WCS_coverageencoding_geotiff/1.0/"
      ↪  xlink:arcrole="fileReference"/>
      <gml:fileReference>cid:Coverage1.tif</gml:fileReference>
      <gml:fileStructure/>
      <gml:mimeType>image/tiff</gml:mimeType>
    </gml:File>
  </gml:rangeSet>
  <gmlcov:rangeType>
    <swe:DataRecord>
      <swe:field name="band_1">
        <swe:Quantity>
          <swe:description>band_1</swe:description>
          <swe:uom code="unknown"/>
          <swe:constraint>
```

```
            <swe:AllowedValues>
              <swe:interval>3.4E-38 3.4E+38</swe:interval>
            </swe:AllowedValues>
          </swe:constraint>
        </swe:Quantity>
      </swe:field>
    </swe:DataRecord>
  </gmlcov:rangeType>
</gmlcov:RectifiedGridCoverage>
--wcs
Content-Type: image/tiff
Content-Description: coverage data
Content-Transfer-Encoding: binary
Content-ID: dom_25833.tif
Content-Disposition: inline
II*

[encoded data]

<GDALMetadata>
  <Item name="DataType">Generic</Item>
  <Item name="SourceBandIndex" sample="0">0</Item>
</GDALMetadata>


[about 4500 more lines of encoded data]
```

Appendix B

Flowchart

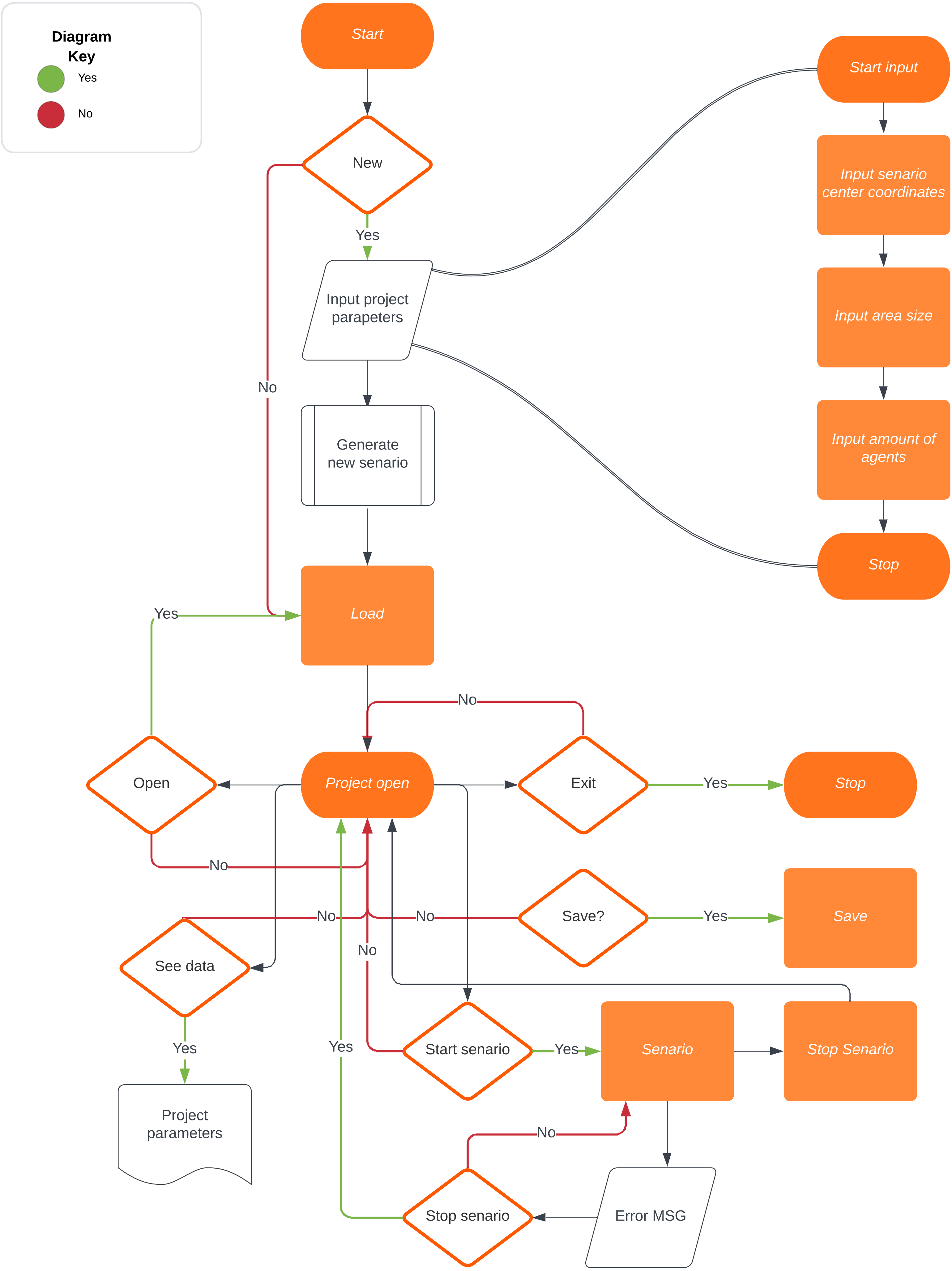


Diagram Key
Yes
No
Start
New
Yes
No
Input project parapeters
Generate new senario
Load
Start input
Input senario center coordinates
Input area size
Input amount of agents
Stop
Yes
No
Open
Project open
Exit
Yes
Stop
No
No
No
Save?
Yes
Save
See data
No
Yes
Yes
Start senario
Yes
Senario
Stop Senario
Project parameters
No
Error MSG
Stop senario

Appendix C

Requirements: User stories

Planning

1. Drone Overview
As a drone operator **I want** to see a map **So that** I know where all the drones are
ACCEPTANCE CRITERIA
Given that the drones are connected to GPS **And** that the drone are connected to the computer **When** the drone operator opens the map on the computer **Then** they should see all the drones that are connected to the computer on the map with GPS coordinates

2. Restricting Flight Zone
As a drone operator **I want** a map where I can select an area **So that** the drones will be restricted to flying within the selected area
ACCEPTANCE CRITERIA
Given that the computer is connected to GPS **And** that the computer is connected to the drones **When** the drone operator wants to restrict where the drone can fly **Then** they should be able to specify, either with start and end coordinates or through click and drag, an area where the drones are allowed to fly

3. Terrain Overview
As a drone operator **I want** to see the buildings in the area **So that** I can plan a route that goes around the buildings
ACCEPTANCE CRITERIA
Given that the computer has connection to the GPS network **When** the drone operator is planning a scenario **Then** the operator should be notified if the route goes too close to a building

4.Safe Landing Zones
As a drone operator **I want** to select an area **So that** the drones can land in a safe spot if needed
ACCEPTANCE CRITERIA
Given that the computer is connected to GPS **And** that the computer is connected to the drones **When** the drone operator wants to select an area for safe landings **Then** they should be able to specify, either with start and end coordinates or through click and drag, an area where the drones can perform a safe landing

5.Spotlight
As a drone operator **I want** to be able to select a point on a map **So that** I can make the drones point their lights at the point
ACCEPTANCE CRITERIA
Given the computer is connected to GPS **And** the computer is connected to the drones **And** the drones are connected to GPS **When** the operator is planning a scenario **Then** they have to be allowed to select a point on the map so that all the drones will point towards the selected point.

6. Creating, Saving and Loading Scenarios
As a drone operator **I want** to be able to create, save and load scenarios I make **So that** I can continue working on a scenario at a later date
ACCEPTANCE CRITERIA
When the operator is planning a scenario

Then there should be options to create a new scenario, load an old scenario or save the current scenario.

7. Simulation
As a drone operator **I want** to simulate the scenario that is selected **So that** I can check that the drones will not collide
ACCEPTANCE CRITERIA
Given that the operator has selected a scenario **Then** the operator should be able to simulate the route for all the drones for the specified scenario

8. Cartesian Coordinates
As a drone operator **I want** to be able to use meters and cartesian coordinates to tell the drones to move **So that** the drones will move a specified distance in a specified direction
ACCEPTANCE CRITERIA
When planning a scenario the operator should give the drones directions in cartesian coordinates

Then the drones should move the correct distance in the correct direction

During Flight

9. Drone status
As a drone operator **I want** to see each drone's status **So that** I can see which drones are having problems and where the drones are and how much time they have left to fly
ACCEPTANCE CRITERIA
Given that the drones are operating **And** connected to the computer **When** I look at the statuspage **Then** I want to be able to see if a drone is having problems and remaining expected flight time

10. Abort Flight
As a drone operator **I want** to be able to abort the flight if a drone has critical errors **So that** I can avoid having the drones crash landing
ACCEPTANCE CRITERIA

Given that the drone has a fault notification **When** the drone notices something is wrong **Then** I want to be able to press a button that aborts the flight and makes the drone perform an emergency landing in a given emergency landing area.

11. Controlling Multiple Drones
As a drone operator **I want** to be able to control multiple drones at the same time **So that** the drones follow their assigned paths
ACCEPTANCE CRITERIA
Given that all the drones are connected to the GPS network **And** all the drones are connected to the computer **When** the operator opens the program **Then** they have to be able to assign paths to multiple drones in the same scenario.

Appendix D

Requirements: Use cases

Name, description nr. 1.1	Drone overview, the drone operator wants to get an overview of all the drones and where they are.
Actors	Drone operator
Pre-condition	The drone operator must have placed the drones they want to use on the ground and turned them on.
Post-condition	The drone operator can see a map of all the drones.

Main success Path (primary flow)	The Drone operator starts the hivemind program on their computer. Then the drone operator connects to all the drones in the area and clicks on the map button. The hivemind program will then show a map of the surrounding area and where the drones are.
Actor actions	**System responses**
1. Opens the hivemind program.	1.1 Start the GUI.
2. Clicks on the find drone button.	2.1 Checks if the computer is Connected to the GPS network. 2.2 Tries to connect the drones in the area. 2.3 Checks if the drones are connected to the GPS network.
3. Opens the map in the Hivemind program.	3.1 The Hivemind program gets GPS positions from all the drones that are connected. 3.2 The Hivemind program will show all the drones it got GPS positions from and the map. 3.3 The drones that could not send their GPS position will be listed.

Name, description	Restricting the flight zone, the drone operator wants to be able to select an area on a map and prevent drones from flying in the selected area.
Actors	Drone operator
Pre-condition	The drone operator must have the Hivemind program open and be connected to the GPS network as well as all the drones that are going to fly in the area.
Post-condition	The drones will stay within the area selected by the drone operator.

Main success Path (primary flow)	The drone operator opens the map in the Hivemind program and selects the area they want the drones to fly within.
Actor actions	**System responses**
1. Click the button to open the map.	1.1 Checks if the computer has a connection with GPS. 1.2 Shows a map of the surrounding area.
2. Selects an area on the map.	2.1 Selected area is graphically indicated on GUI. 2.2 A range of forbidden coordinates is generated. 2.3 Range of forbidden coordinates added into flight plan constraints.

Name, description	Terrain overview, the drone operator wants to get an overview of all the buildings in the area.
Actors	Drone operator
Pre-condition	The computer must be turned on and connected to the GPS network
Post-condition	The drone operator can see a map where he can click on a building and get relevant information about the selected building.

Main success Path (primary flow)	When the drone operator is looking at the map in the Hivemind program and clicks on a building a small window opens that shows information about the height of the building.
Actor actions	**System responses**
1. Opens the map in the Hivemind program.	1.1 Shows the map
2. Clicks on a building on the map.	2.1 Checks if there is any available data for the building. 2.2 Shows the information it gathered. 2.3 If it can't find any information about the building it will show an error message instead.

Name, description	Assigning safe landing zones, The drone operator wants to be able to assign a safe landing area
Actors	Drone operator
Pre-condition	The computer must be turned on and connected to the GPS network
Post-condition	The drone operator can see a map and select an area and assign it as a safe landing spot

Main success Path (primary flow)	When the drone operator is looking at the map in the Hivemind program and selects an area. They get the option to assign the area as a safe landing area.
Actor actions	**System responses**
1. Opens the map in the Hivemind program.	1.1 Shows the map.
2. Selects an area on the map.	2.1 Highlights the selected area 2.2 Ask what the operator wants to do with the area.
3. Assign it as a safe landing zone.	3.1 Send the coordinates of the assigned safe landing zones to the drones.

Name, description	Spotlight, the drone operator wants the drone to point at the same spot.
Actors	Drone operator
Pre-condition	The computer and the drones must be turned on and connected to the GPS network
Post-condition	The drone operator can see a map and select a spot on the map and tell all the active drones to point at the same spot

Main success Path (primary flow)	The drone operator is planning a scenario and selects a specific time in the run. They also select a spot on the map where he wants all the drones to point their lights. When the drones follow the route they will at the specified time point their light in the correct direction.
Actor actions	**System responses**
1. When planning a scenario the drone operator selects a point on the map.	1.1 The program gets the GPS coordinates for the selected point.
2. Assign the point as somewhere they want the drones to point.	2.1 Sends instructions to the drones when and where they should point.

Name, description	Creating, saving and loading scenarios, the drone operator should have the options to create new scenarios, save scenarios and load previous scenarios.
Actors	Drone operator
Pre-condition	The Hivemind app has to be open on the computer
Post-condition	The Hivemind program has loaded a previously made scenario or saved the current scenario.

Main success Path (primary flow)	When making a scenario for the drones the drone operator wants to save the scenario so that they can keep working on it later. They click the save scenario button and the program saves the scenario as a XML/JSON file.
Actor actions	**System responses**
1. Clicks on the save scenario button.	1.1 Asks what the scenario should be called.
2. Inputs a name.	2.1 Saves the scenario as a XML/JSON file.

Alternative success Path (secondary flow)	When making a scenario for the drones the drone operator wants to load a scenario they already made.
Actor actions	**System responses**
1. Clicks on the load scenario or create new scenario buttons.	1.1 Open the directory where scenarios are stored.
2. Either selects the file they want to open or where they want to store a new scenario.	2.1 Loads the scenario.

Name, description	Simulation, the drone operator should have the option to simulate the scenario when they have completed making a scenario or while making a scenario. They should see the surrounding area.
Actors	Drone operator
Pre-condition	The Hivemind app is open on the computer and the drone operator has selected a scenario to simulate.
Post-condition	The Hivemind program has simulated the scenario so that the drone operator can see that the scenario works in theory.

Main success Path (primary flow)	When the drone operator wants to simulate a scenario they need to first select the scenario they want to simulate and give a start position for the scenario.
Actor actions	**System responses**
1. Clicks on the "simulate scenario" button	1.1 Asks the operator to choose which scenario to simulate. 1.2 Asks for the start coordinates for the scenario.
2. Clicks on the "start simulation" button	2.1 Shows a visual representation of the scenario.

Name, description	Cartesian coordinates, while working in Hivemind all coordinates should be converted to cartesian coordinates
Actors	Drone operator
Pre-condition	The Hivemind app is open on the computer and the drone operator is planning a scenario.
Post-condition	The drone operator is done planning a scenario.

Main success Path (primary flow)	The drone operator is planning a scenario. They choose a drone they want to move and tells it to move in a specified direction and a specified distance.
Actor actions	**System responses**
1. Clicks on a drone to give it instruction.	1.1 Highlights the chosen drone.
2. Inputs the direction and distance the drone has to move.	1.2 Moves the drone as specified and shows the drone's new location.

Name, description	Drone status, while the drones are in flight the drone operator wants to see the status of their connectivity and the status of their batteries as well as any issues the drones are encountering.
Actors	Drone operator
Pre-condition	The drones are in flight
Post-condition	The drone operator can see the drones status in the Hivemind program

Main success Path (primary flow)	The drones send information about their status and the Hivemind program organizes the data in an intuitive way.
Actor actions	**System responses**
1. Opens the “Drone Status” window.	1.1 Lists all connected drones and information about their status. 1.2 Highlights any drones that are having issues.

Name, description	Abort Flight, while the drones are in flight they might encounter a critical issue. If this happens the drone operator needs to be notified.
Actors	Drone operator, Drone
Pre-condition	The drones are in flight and having an issue
Post-condition	The drone operator can see the drones issue in the hivemind software and give it new instruction if it is needed.

Main success Path (primary flow)	The drones send notifications to the Hivemind software whenever they encounter anything abnormal in flight. And receives new instructions from the drone operator.
Actor actions	**System responses**
1. Drone sends a notification to the Hivemind software.	1.1 Shows the issue the drone is having. 1.2 Asks if it should continue as planned.
2. Drone operator inputs new instructions for the drone.	2.1 Sends the new instruction to the drone.

Alternative success Path (secondary flow)	The drone is told to continue Following the planned route.
Actor actions	**System responses**
1. Drone operator tells the drone to continue as planned.	1.1 System continues as planned.

Name, description	Controlling multiple drones, the drone operator wants to be able to control multiple drones at the same time.

Actors	Drone operator
Pre-condition	The drones are in flight
Post-condition	The drone operator has assigned routes to multiple drones so that they will follow their paths at the same time.

Main success Path (primary flow)	The drone operator creates routes for different drones and when the drone operator runs the program the drones have to follow their own path to a synchronized clock.
Actor actions	**System responses**
1. Creates keyframes for multiple drones	1.1 Handles the keyframes in a way where Hivemind can keep track of which keyframes belong to which drone.

Appendix E

Requirements table

Priority colours	Necessary - part of minimum viable product		Optional - nice to have		
Status colours	Not started	Ongoing	Testing	Completed	Not Viable
User story, requirements, testing colours	Colours will be set to coordinate with colours in our architecture in order to easily track which requirements relate to which component of our system.				

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
1. Drone overview	**1.1.** The drone operator wants to get an overview of all the drones and where they are.	Initiates the Hivemind software.	Launch the graphical user interface (GUI)	**R1.1.** Graphical user interface correctly launches upon software start-up.	**A**	**T.1.1.** Launch software, verify that GUI is visible and can be interacted with.	Completed
		Selects a the tab they want	Shows the corrects tab	**R.1.2.** Upon start-up, the user will be able to select if they want to load or start a new scenario.	**A**	**T.1.2.** Verify that the selected mode corresponds to the actual mode launched by software.	On going
		Selects a tab with a map	Software initiates computer GPS network connection	**R.1.3.** Computer will connect to the GPS network.	**C**	**T.1.3.** Verify using a GUI map that the given GPS location is correct.	Not started
			Connects to the compatible drones in the area.	**R.1.4.** Computer is communicating on the correct frequency/channel.	**C**	**T.1.4.** Test that computer receives a "sign-of-life" signal from a dummy drone (testing module).	Not started
				R.1.5. Computer successfully connects to all detected drones.	**C**	**T.1.5.** Successfully ping all drones connected on specified frequency/channel.	Not started
			Checks if the drones are connected to the GPS network.	**R.1.6.** Successfully query the drone for GPS connection status.	**C**	**T.1.6.** Verify that the GPS status of a connected dummy drone can be queried by software.	Not started
		Selects a tab with a map	Software receives the GPS position of all connected drones.	**R.1.7.** Computer is able to receive GPS data from a connected drone.	**C**	**T.1.7.** Verify that the computer can receive GPS data from a connected drone (dummy	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
						drone), and that GPS data received is the same that is sent from a drone (dummy drone).	
			All connected drones with GPS data made visible on GUI	**R.1.8.** Each connected drone that was connected to GPS is visible on a map as a dot or number.	**C**	**T.1.8.** Verify that all connected drones with GPS data appear on map, and that no connected drones without GPS data appears on map.	Not started
			List drones that did not send GPS positions.	**R.1.9.** Each drone without GPS position will be made visible in a designated part of the GUI with drone identifier.	**C**	**T.1.9.** Connect a dummy drone without GPS connection, verify that it appears in the designated part of the GUI.	Not started
2. No fly zones	**2.1.** The drone operator wants to restrict the drones to flying within a selected area.	Selects a tab with a map	Map appears on screen	**R.2.1.** When Opening the planner tab, map appears on software GUI	**B**	**T.2.1.** Pressing the map button correctly launches the map	Completed
		Click the "find computer" button	Software confirms computer GPS connection	**R.1.3.** Computer will connect to the GPS network.	**C**	**T.2.2** Verify that clicking the button will check GPS connection	Not started
						T.1.3. Verify using a GUI map that the given GPS location is correct.	Not started
			Map relocates to computer GPS location	**R.2.2.** Visible map on GUI is centred upon the computer's location	**C**	**T.2.3.** Given the computer's current location is known, verify visually that the map has been centred on the correct location	Not started
		Selects an area on the map	User can select an area on the map	**R.2.3.** Software allows for the selection of an area on the map	**C**	**T.2.4.** Select an area and confirm that this area is accurately reflected in the selection on the GUI	Not started
			Selected area is graphically indicated on GUI	**R.2.4.** The area selected will be graphically indicated	**C**		Not started
			User should be able to fine-tune selected area	**R.2.5.** When an area is selected, the user will be able to fine-tune selection using either drag-and drop	**C**	**T.2.5.** When fine-tuning, user commands are accurately	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
				mechanics, incremental arrow commands or to enter their own coordinates.		reflected in the updated restricted area.	
			Dropdown menu with further actions appears	**R.2.6.** When an area is selected, a menu of options for selections will appear.	**C**	**T.2.6.** Selecting an area correctly triggers the selection menu to appear	Not started
				R.2.7. A selection for making a restricted flight zone is available	**C**	**T.2.7.** The restrict flight zone option appears on the menu when an area is selected on the map.	Not started
		User clicks/selects "restrict flight zone" option	System enters restriction selection mode	**R.2.8.** Selecting the restricted flight zone option successfully allows for the definition of a no-fly zone.	**B**	**T.2.8.** Test that selecting restrict flight zone option successfully changes mode.	Not started
		Confirms selection	A range of forbidden coordinates is generated	**R.2.9.** A range of forbidden coordinates will be generated when area has been selected	**B**	**T.2.9.** Select known "no flight zone", cross-reference the resulting coordinates with the known ranges for the "no flight zone" and ensure they are the same.	Not started
			Range of forbidden coordinates added into scenario constraints	**R.2.10.** User is not allowed to create scenarios including any drone at any of the restricted coordinates	**B**	**T.2.10.** When trying to make a scenario including any restricted coordinates, the system will return an error and not generate a scenario.	Not started
3. Terrain Overview	**3.1** The drone operator wants to have an overview of the buildings in the area.	Selects a tab with a map	Map appears on screen	**R.2.1** When map button is pressed, map appears on software GUI	**A**	**T.2.1** Pressing the map button correctly launches the map	Not started
		User clicks on a point on the map	Dropdown menu of available actions appears	**R.3.1** When user clicks on the map in the GUI, a dropdown menu of available actions appears	**C**	**T.3.1** Clicking the map will successfully show the dropdown menu	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		User clicks a point on the map	Software checks if there is available data for point	**R.3.2** Height data is queried for data on a given coordinate. It will return either the height data or a NULL value.	**A**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Testing
						T.3.3. Test selecting a coordinate with no known height data to verify that the returned data is correct.	Testing
			Software displays available information	**R.3.3** Hivemind will display available height data for the selected point.	**B**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Not started
			If no data is available, an error message is shown	**R.3.4** Software will display an error message and no more data is returned.	**B**	**T.3.4.** Test selecting coordinate with known data NULL, verify the software returns an error message and no further action is taken.	Testing
		Sets a start location for the drones	Loads topographical information about the selected point	**R.3.5.** The software needs to be able to collect data and information about buildings and the landscape of specified locations	**A**	**T.3.5.** The software try to select a point where you want to start and make sure the software collects information about the correct area	Completed
	3.2 User wants to be able to view terrain in 3D (24/04/2023)	User selects 3D mode	2D map is exchanged with 3D visualisation of terrain	**R.3.6.** When 3D mode is chosen, the 2D map is successfully replaced with 3D visualisation	**C**	**T.3.6.** Test selecting 3D mode and confirm that 3D visualisation is loaded	Ongoing
						T.3.7. Confirm that 3D visualisation corresponds to height data and selected origin of 2D map	Ongoing
		User clicks and drags screen (or clicks on GUI element) to	Point of view rotates in accordance with user selection	**R.3.7.** The user is able to rotate 3D visualisation using GUI elements or mouse actions.	**C**	**T.3.8.** Test through interaction that 3D visualisation is rotated.	Ongoing

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		change point of view					
		User scrolls mousewheel (or clicks on GUI element) to zoom in or out	Visualisation zooms in	**R.3.8.** The user is able to zoom in or out on visualisation using GUI elements or mouse actions.	**C**	**T.3.9.** Test through interaction that 3D visualisation zooms in/out.	Ongoing
4. Safe Landing Zones	**4.1** The drone operator wants to select a spot within which the drones can perform safe landings.	User selects area on map	Selected area is graphically indicated on GUI	**R.2.4** The area selected will be graphically indicated	**C**	**T.2.4** Select an area and confirm that this area is accurately reflected in the selection on the GUI	Not started
			Dropdown menu with further actions appears	**R.2.6.** When an area is selected, a menu of options for selections will appear.	**C**	**T.2.6.** Selecting an area correctly triggers the selection menu to appear	Not started
				R.4.1. The user will be able to select a "designate safe landing zones" option	**C**	**T.4.1.** Verify that the option to designate a safe landing zone appears in the menu when user selects an area	Not started
			User should be able to fine-tune selected area	**R.2.5.** The user will be able to fine-tune selection	**C**	**T.2.5.** When fine-tuning, user commands are accurately reflected in the updated restricted area.	Not started
		User selects "designate safe landing zones" option	Range of coordinates for safe landing zone is displayed to the user.	**R.4.2.1.** Safe landing zone coordinates are displayed.	**C**	**T.4.2.** Test that the selected area corresponds to a populated list of coordinates.	Not started
				R.4.2.2. Safe landing zones are populated based on user selection.	**C**		
		Confirms selection	A range of safe landing zones is generated	**R.4.3.** A range of safe landing zone coordinates will be generated when area has been selected	**C**	**T.4.3.** Test that the scenario correctly includes the range of safe landing zones selected by the user.	Not started
5. Spotlight	**5.1** The drone operator wants to select a point on a map that	User clicks a point on map	Dropdown menu of available actions appears	**R.3.1** When user clicks on the map in the GUI, a dropdown menu of available actions appears	**C**	**T.3.1** Clicking the map will successfully show the dropdown menu	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
	will be targeted for illumination.	User selects "Illuminate this point"	System graphically marks point as target of illumination	**R.5.1.** When user clicks on the "Illuminate this point" option, the point is marked on the GUI map.	**C**	**T.5.1.** Test that the point is marked with an icon when "Illuminate this point" is selected.	Not started
						T.5.2. Verify that the marked point is the same as the selected point.	Not started
			System gets the coordinate for selected point	**R.5.2.** System can query map database for coordinates of the selected point.	**C**	**T.5.3.** Verify that the returned coordinates are the same as a known test coordinate.	Not started
			Software checks if there is available height data for point	**R.3.2** Height data is queried for data on a given coordinate. It will return either the height data or a NULL value.	**A**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Testing
						T.3.3. Test selecting a coordinate with no known height data to verify that the returned data is correct.	Testing
			Software displays available information	**R.3.3** Software will display available height data for the selected point.	**A**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Testing
			Planning space appears with available coordinate and height data with space to enter desired height for illumination.	**R.5.3.** Queried information on coordinates and height will be visible in a separate planning space (pane, window, popup), where the user can enter information.	**C**	**T.5.4.** Verify that planning space successfully appears when a user requests to illuminate a point, and that data it is populated with is accurate.	Not started
		User enters a height selection for a point.	Illumination target coordinates updated with height.	**R.5.4.** When height data is entered into the planning pane, the backend planning data is successfully updated to reflect this change.	**C**	**T.5.5.** Verify that the entered height data is the same as what appears in the route.	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		Users confirms selection	Route to illuminate point at selected height generated.	**R.5.5.** System generates a route based on desired coordinates and height of illumination.	C	**T.5.6.** Simulate route and verify that drone reaches desired coordinates and height.	Not started
6. Saving, Loading and Creating Scenarios	**6.1** The drone operator wants to be able to save routes/scenarios created to be able to load it later.	(When in an open scenario) User clicks the "Save Scenario" button on the GUI	System prompts the user to name the new scenario..	**R.6.1.** Clicking the save scenario button prompts a dialogue box for saving the scenario.	A	**T.6.1** Verify that clicking the "Save" button will launch the save dialogue box.	Completed
		User enters a name and clicks enter.	All scenario data is serialised to an XML file.	**R.6.2.** The current scenario will be recorded accurately in a file.	A	**T. 6.4.** Verify that the user can successfully save a file in a custom location.	Completed
						T.6.3. Load file and verify that it is identical to the plan saved earlier.	Completed
		User writes desired location of saved file/selects it using a directory explorer.	If using directory explorer, open explorer GUI to allow the user to select desired location.	**R.6.3.** The user is allowed to define where the file should be saved.	A	**T. 6.4.** Verify that the user can successfully save a file in a custom location.	Completed
	6.2. The drone operator wants to load a previously created scenario.	(When in any part of the Hivemind UI) Click the "Load" button.	Opens a list of recent saved scenarios to allow the user to select from	**R.6.4.** Recent saved scenarios are listed out for the user to select from.	C	**T.6.5.** When "Load" is clicked, a list of recent scenarios will be made available.	Ongoing
			Allows the user to explore file structure to find the desired scenario.	**R.6.5.** The user can browse through files to find the correct scenario.	A	**T.6.6.** When "Load Scenario" is pressed, verify that the explorer window appears.	Completed
		User selects desired scenario	Hivemind opens scenario planning mode.	**R.6.6.** When a scenario has been selected, Hivemind will open the scenario.	A	**T.6.7.** Verify that the correct scenario opens when selected.	Completed
			Current scenario populated with data from file.	**R.6.7.** When a scenario is opened, all saved data such as chosen options and keyframes populate relevant fields in the planning panes.	A	**T.6.8.** Verify that the populated options and coordinates	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
						match with a given, known test scenario.	
	6.3 The drone operator wants to create a simple scenario containing two points	(When in the planner view) User selects "Set location"	System opens dialogue box allowing for entering of scenario coordinates	**R.6.8.** When user selects set location, some UI element allowing them to enter scenario coordinates should appear	**A**	**T.6.9.** Press the set location, and verify that the dialogue box to enter coordinates appears as expected.	Completed
		User presses "add keyframe" and inputs keyframe information	A new keyframe is added, and the keyframe is added to the keyframe list	**R.6.9.** When a keyframe is entered, a new keyframe is successfully added to graphical user interface	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.	Completed
				R.6.10. When a keyframe is entered, a new keyframe appears in the keyframe list corresponding to entered coordinates.	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.	Completed
			The new keyframe is added to the timeline	**R.6.11.** When a coordinate is added, a new keyframe appears on the timeline corresponding to the relative timestamp of the keyframe	**C**	**T.6.12.** Verify that added keyframe is correctly assigned on the timeline	Completed
				R.6.12. The position of keyframes on the timeline should correspond to the length of the scenario and update dynamically	**C**	**T.6.13.** Add a coordinate, and verify that the distribution of keyframes on timeline dynamically updates.	Completed
		User clicks add keyframe	Dialogue box for new keyframe appears	**R.6.13.** When user clicks add keyframe, dialogue box for new keyframe appears	**A**	**T.6.14.** Press add keyframes, and verify that the dialogue box appears as expected.	Completed
		User enters coordinates and timestamp and presses enter	New keyframe is added, and the keyframe is added to the list of keyframes.	**R.6.9.** When a keyframe is entered, a new keyframe is successfully added to graphical user interface	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
						dialogue box, and that timestamp and location is correct.	
				R.6.10. When a keyframe is entered, a new keyframe appears in the keyframe list corresponding to entered coordinates	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.	Completed
			The new keyframe is added to the timeline	**R.6.11.** When a is added, a new keyframe appears on the timeline corresponding to the relative timestamp of the keyframe	**C**	**T.6.12.** Verify that added keyframe is correctly assigned on the timeline	Completed
				R.6.12. The position of keyframes on the timeline should correspond to the length of the scenario and update dynamically	**C**	**T.6.13.** Add a coordinate, and verify that the distribution of keyframes on timeline dynamically updates.	Completed
		User presses compile scenario	All key frames are generated into a route and all routes are compiled into a scenario.	**R.6.14.** Key frames for drones are generated into a route.	**A**	**T.6.15.** Press compile scenario, and verify that a route is created, and that information in route is correct.	Completed
				R.6.15. All routes are successfully compiled into a scenario.	**A**		
	6.4 The operator wants to review the scenario using a timeline	User interacts with timeline to see the keyframes appear	As the user interacts with timeline, keyframes appears on map in correct order	**R.6.16.** The animation of keyframes on the map corresponds to their timestamps.	**C**	**T.6.16.** Interact with the timeline and verify that the animations correspond to the keyframes.	Not started
7. Simulation	**7.1** The drone operator wants to simulate the selected route/scenario.	Selects the simulation mode in the software	Shows the simulation GUI on the screen	**R.7.1.** The software needs a GUI for a simulation mode	**B**	**T.7.1.** When the software is open try to select simulation mode and make sure the software enters the desired state	Not started
		(When in any part of the Hivemind UI)	Opens a list of saved route plans to allow the user to select from	**R.6.4.** Saved routes are listed out for the user to select from.	**C**	**T.6.5.** When “Load Route” is clicked, a list of routes will be made available.	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		Click the “Load” button.	Allows the user to explore file structure to find the desired route.	**R.6.5.** The user can browse through files to find the correct scenario.	**A**	**T.6.6.** When “Browse” is pressed, verify that the explorer window appears.	Completed
		User selects desired route	Hivemind opens route planning mode.	**R.6.6.** When a scenario has been selected, Hivemind will open the scenario.	**A**	**T.6.7.** Verify that the correct scenario opens when selected.	Completed
			Current scenario populated with data from file.	**R.6.7.** When a scenario is opened, all saved data such as chosen options and current list of coordinates populates relevant fields in the planning panes.	**A**	**T.6.8.** Verify that the populated options and coordinates match with a given, known test scenario.	Completed
		Sets a start location for the drones	Loads topographical information about the selected point	**R.3.5.** The software needs to be able to collect data and information about buildings and the landscape of specified locations	**A**	**T.3.5.** When in the simulation mode of the software try to select a point where you want to start and make sure the software collects information about the correct area	Completed
		Start simulation	Simulates the scenario	**R.7.3.** The software needs the ability to simulate the planned scenario in a visual representation	**C**	**T.7.3.** When in the simulation mode of the software try to start a simulation and make sure it uses the correct start location and behave as expected	Not Started
8. Coordinate Converting	**8.1** The drone operator wants to convert between different coordinate systems.	When the drone operator enters a coordinate the software can convert that coordinate to a different coordinate system where it is needed.	The software converts coordinates whenever it is needed.	**R.8.1.** The software has to be able to convert cartesian coordinates to geographical coordinates as well as geographical coordinates to cartesian coordinates.	**A**	**T.8.1.** Enter a coordinate and check that you get expected data in return.	Completed
9. Drone Status	**9.1** The drone operator wants to be able to	Select the drone overview mode in the software	List all the connected drones with a unique identification	**R.9.1.** The software has to be able to keep track of which drone is which	**B**	**T.9.1.** Connect several drone and have them do different movements to make sure it	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
	see health metrics of each drone and related warnings.					is consistent with what shows up in the software	
			if the software receives information from a drone that is having problems it highlights which drone is having issues	**R.9.2.** The software has to be able to highlight drones that are having problems so that they are easy to identify for the drone operator	**C**	**T.9.2.** Have a dummy drone simulate having trouble by sending a signal to the software as if the drone was having an issue and make sure the correct drone is highlighted in the software	Not started
			Displays information from each drone.	**R.9.3.** Has to be able to receive information from the drones and keep track of which drone sent the information	**C**	**T.9.2.** Have a dummy drone simulate having trouble by sending a signal to the software as if the drone was having an issue and make sure the correct drone is highlighted in the software.	Not started
10. Abort Flight	**10.1** The drone operator wants to be able to safely abort the flight of one or more drones.	A drone sends a message to the software that it is struggling	Highlight the drone that is struggling and what it is struggling with	**R.10.1.** Has to be able to receive information from the drones and keep track of which drone sent the information	**C**	**T.9.2.** Have a dummy drone simulate having trouble by sending a signal to the software as if the drone was having an issue and make sure the correct drone is highlighted in the software.	Not started
		The drone operator clicks the struggling drone	Shows a menu/pop-up window where the drone operator can either tell the drone to land or to continue as planned	**R.10.2.** Has to be able to show a menu/pop-up window when the drone operator clicks on a highlighted drone	**C**	**T.10.1.** Click on a drone when it is highlighted and make sure you get the options to either emergency land it or to let it keep doing the planned route	Not started
		The drone operator selects the emergency landing option	Sends instructions to the struggling drone to land at the closest emergency landing	**R.10.3.** Has to be able to overwrite the planned route for the drones	**C**	**T.10.2.** Have a dummy drone pretend to fly a route and try to make it abort it's route and land in the emergency landing zone	Not started
		The drone operator selects	The system keeps going as planned	**R.10.4.**	**C**	**T.10.3.**	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		the keep flying option		The software has to be able to let drones keep flying even if they have certain faults		Make a dummy drone simulate having a problem and ignore it in the software	
11. Controlling multiple agents	**11.1** The drone operator wants a mode wherein he can indicate a lead drone and formation and have the rest of the drones follow the leader in a selected route.	Selects a drone	The software highlights the selected drone	**R.11.1.** Has to be able to highlight a drone when it is selected	**C**	**T.11.1.** When planning a scenario you select a drone and make it the leader.	Not started
			The software makes routes for the rest of the drone so that they follow the leader in a formation	**R.11.2.** The software has to be able to make routes for several drones	**B**	**T.11.2.** After adding a different agent ID the software will make paths for the different drones based on the agent ID of the keyframe.	Completed
	11.2 The drone operator wants to plan the routes individually for several agents manually	User adds a new agent through user interface actions	A new agent is listed in the user interface along with any previous agents	**R.11.3.** The user can add new agents through the interface.	**B**	**T.11.3.** Test that adding new agent through interface causes the new drone to correctly appear	Completed
				R.11.4. The new agent is correctly displayed in the user interface.	**B**		Completed
				R.11.5 All added agents should be uniquely identified in the interface.	**B**	**T.11.3.** Test that adding new agent through interface causes the new drone to correctly appear	Completed
		User changes active agent	System changes active agent	**R.11.6.** Active agent is changed in system back-end	**B**	**T.11.3.** Test that adding new agent through interface causes the new drone to correctly appear	Completed
			Active agent is displayed in user interface	**R.11.7.** Active agent is indicated in user interface	**B**		
		User adds keyframe for active agent	System ascribes keyframe to active agent	**R.11.8.** The agent ID in the new keyframe is the same as the active drone.	**B**		
		User compiles scenario	System performs route planning to avoid collisions	**R.11.9.** When generating a route for multiple agents, the algorithm shall ensure the drones do not collide.	**B**	**T.11.6.** Try compiling a route where the agents are on a clear collision course and confirm that collisions are avoided through adjusting time stamps or routes.	Not started

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
				R.11.10. If the algorithm is unable to prevent a collision, the route should not be generated and an error returned.	**B**	**T.11.7.** Try setting up a route where there is no way to avoid a collision and confirm that	Not started
			Compiles scenario containing all agent route	**R.11.11.** The compiled scenario should include all agent routes.	**B**	**T.11.8** Compile scenario and ensure all agent routes are included.	Completed

Appendix F

Final Hivemind product requirements

Priority colours	Necessary - part of minimum viable product		Optional - nice to have		
Status colours	Not started	Ongoing	Testing	Completed	Not viable
User story, requirements, testing colours	Colours will be set to coordinate with colours in our architecture in order to easily track which requirements relate to which component of our system.				

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
1. Drone overview	**1.1.** The drone operator wants to get an overview of the area the scenario is being planned for	Initiates the Hivemind software.	Launch the graphical user interface (GUI)	**R1.1.** Graphical user interface launches upon software start-up.	**A**	**T.1.1.** Launch software, verify that GUI is visible and can be interacted with.	Completed
		Selects a the tab they want	Shows the corrects tab	**R.1.2.** Upon start-up, the user shall be able to select if they want to load or start a new scenario.	**B**	**T.1.2.** Verify that the selected scenario corresponds to the actual scenario launched by software.	Not started
3. Terrain Overview	**3.1** The drone operator wants to have an overview of the buildings and the terrain in the area	Selects a tab with a map	Map appears on screen	**R.2.1** When opening the planner tab, map appears on software GUI	**A**	**T.2.1** Open the planner tab, and verify that the map for the given area appears on screen.	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		User clicks a point on the map	Software displays available information	**R.3.3** Hivemind will display available height data for the selected point.	**B**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Not started
			If no data is available, an error message is shown	**R.3.4** Software will display an error message and no more data is returned.	**B**	**T.3.4.** Test selecting coordinate with known data NULL, verify the software returns an error message and no further action is taken.	Testing
		Sets a start location for the drones	Loads topographical information about the selected point	**R.3.5.** The software needs to be able to collect data and information about buildings and the landscape of specified locations	**A**	**T.3.2** Test selecting a coordinate with known height data to verify the returned data is correct.	Completed
	3.2 User wants to be able to view terrain in 3D (24/04/2023)	User selects 3D mode	2D map is exchanged with 3D visualisation of terrain	**R.3.6.** When 3D mode is chosen, the 2D map is successfully replaced with 3D visualisation	**C**	**T.3.6.** Test selecting 3D mode and confirm that 3D visualisation is loaded	Not viable
						T.3.7. Confirm that 3D visualisation corresponds to height data and selected origin of 2D map	Not viable
		User clicks and drags screen (or clicks on GUI element)	Point of view rotates in accordance with user selection	**R.3.7.** The user is able to rotate 3D visualisation using GUI elements or mouse actions.	**C**	**T.3.8.** Test through interaction that 3D visualisation is rotated.	Not viable

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		to change point of view					
		User scrolls mousewheel (or clicks on GUI element) to zoom in or out	Visualisation zooms in	**R.3.8.** The user is able to zoom in or out on visualisation using GUI elements or mouse actions.	**C**	**T.3.9.** Test through interaction that 3D visualisation zooms in/out.	Not viable
6. Saving, Loading and Creating Scenarios	**6.1** The drone operator wants to be able to save routes/scenarios created to be able to load it later.	(When in an open scenario) User clicks the "Save Scenario" button on the GUI	System prompts the user to name the new scenario..	**R.6.1.** Clicking the save scenario button prompts a dialogue box for saving the scenario.	**A**	**T.6.1** Verify that clicking the "Save" button will launch the save dialogue box.	Completed
		User enters a name and clicks enter.	All scenario data is serialised to an XML file.	**R.6.2.** The current scenario will be recorded accurately in a file.	**A**	**T. 6.4.** Verify that the user can successfully save a file in a custom location.	Completed
						T.6.3. Load file and verify that it is identical to the previously saved plan	Completed
	6.2. The drone operator wants to load a previously created scenario.	(When in any part of the Hivemind UI) Click the "Load" button.	Opens a list of recent saved scenarios to allow the user to select from	**R.6.4.** Recent saved scenarios are listed out for the user to select from.	**C**	**T.6.5.** When "Load" is clicked, a list of recent scenarios will be made available.	Not started
			Allows the user to explore file	**R.6.5.**	**A**	**T.6.6.**	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
			structure to find the desired scenario.	The user can browse through files to find the correct scenario.		When "Load Scenario" is pressed, verify that the explorer window appears.	
		User selects desired scenario	Hivemind opens scenario planning mode.	**R.6.6.** When a scenario has been selected, Hivemind will open the scenario, loading all saved data.	**A**	**T.6.7.** Verify that the correct scenario opens when selected.	Completed
	6.3 The drone operator wants to create a simple scenario containing two points	(When in the planner view) User selects "Set location"	System opens dialogue box allowing for entering of scenario coordinates	**R.6.8.** User shall be able to specify location and size for a scenario	**A**	**T.6.9.** Set location and verify that correct location and data is loaded.	Completed
		User presses "add keyframe" and inputs keyframe information	A new keyframe is added, and the keyframe is added to the keyframe list	**R.6.9.** When a keyframe is entered, a new keyframe is successfully added to graphical user interface	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.	Completed
				R.6.10. When a keyframe is entered, a new keyframe appears in the keyframe list corresponding to entered coordinates.	**A**	**T.6.10.** Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
			The new keyframe is added to the timeline	**R.6.11.** When a coordinate is added, a new keyframe appears on the timeline corresponding to the relative timestamp of the keyframe	**C**	**T.6.12.** Verify that added keyframe is correctly assigned on the timeline	Completed
				R.6.12. The position of keyframes on the timeline should correspond to the length of the scenario and update dynamically	**C**	**T.6.13.** Add a coordinate, and verify that the distribution of keyframes on timeline dynamically updates.	Completed
		User presses compile scenario	All key frames are generated into a route and all routes are compiled into a scenario.	**R.6.14.** Key frames for drones are generated into a route.	**A**	**T.6.15.** Press compile scenario, and verify that a route is created, and that information in route is correct.	Completed
				R.6.15. All routes are successfully compiled into a scenario.	**A**		
	6.4 The operator wants to review the scenario using a timeline	User interacts with timeline to see the keyframes appear	As the user interacts with timeline, keyframes appears on map in correct order	**R.6.16.** The animation of keyframes on the map corresponds to their timestamps.	**C**	**T.6.16.** Interact with the timeline and verify that the animations correspond to the keyframes.	Not started
8. Coordinat e Convertin g	**8.1** The drone operator wants to convert between different coordinate systems.	When the drone operator enters a coordinate the software can convert that coordinate to a different coordinate	The software converts coordinates whenever it is needed.	**R.8.1.** The software has to be able to convert between coordinates in geographical coordinate space, Universal Transverse Mercator coordinate space and cartesian space.	**A**	**T.8.1.** Enter a coordinate and check that you get expected data in return.	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
		system where it is needed.					
11. Controlling multiple agents	**11.1** The drone operator wants a mode wherein he can indicate a lead drone and formation and have the rest of the drones follow the leader in a selected route.	Selects a drone	The software highlights the selected drone	**R.11.1.** The software shall be able to highlight an agent when it is selected	**C**	**T.11.1.** Select an agent and verify that it is highlighted.	Not started
			The software makes routes for the rest of the drone so that they follow the leader in a formation	**R.11.2.** The software shall be able to make routes for several agents	**B**	**T.11.2.** Verify that each generated route is separated by an agent ID.	Completed
	11.2 The drone operator wants to plan the routes individually for several agents manually	User adds a new agent through user interface actions	A new agent is listed in the user interface along with any previous agents	**R.11.3.** The user can add new agents through the interface.	**B**	**T.11.3.** Test that adding new agent through interface causes the new drone to correctly appear	Completed
				R.11.4. The new agent is correctly displayed in the user interface.	**B**		Completed
				R.11.5 All added agents should be uniquely identified in the interface.	**B**	**T.11.3.** Test that adding new agent through interface causes the new drone to correctly appear	Completed
		User changes active agent	System changes active agent	**R.11.6.** Active agent is changed in system back-end	**B**	**T.11.3.** Test that adding new agent through interface	Completed

User story	Use case	Actor actions	System response	Derived requirement	Priority	Test method	Status
			Active agent is displayed in user interface	**R.11.7.** Active agent is indicated in user interface	**B**	causes the new drone to correctly appear	
		User adds keyframe for active agent	System ascribes keyframe to active agent	**R.11.8.** The agent ID in the new keyframe is the same as the active drone.	**B**		
		User compiles scenario	System performs route planning to avoid collisions	**R.11.9.** When generating a route for multiple agents, the algorithm shall ensure the drones do not collide.	**C**	**T.11.6.** Try compiling a route where the agents are on a clear collision course and confirm that collisions are avoided through adjusting time stamps or routes.	Not started
				R.11.10. If the algorithm is unable to prevent a collision, the route should not be generated and an error returned.	**C**	**T.11.7.** Try setting up a route where there is no way to avoid a collision and confirm that no routes are generated and an error is returned.	Not started
			Compiles scenario containing all agent route	**R.11.11.** The compiled scenario should include all agent routes.	**B**	**T.11.8** Compile scenario and ensure all agent routes are included.	Completed

Appendix G

Original test table

Priority colours	Necessary - part of minimum viable product	Optional - nice to have

Test	Sub-tests	Type of test	Requirements tested
T.1.1. Launch software, verify that GUI is visible and can be interacted with.	**T.1.1.1.** See that GUI is visible	Inspection	**R1.1.** Graphical user interface correctly launches upon software start-up.
	T.1.1.2. Verify GUI is interactive	Demonstration	
	T.1.1.3. Verify that interacting with GUI elements will trigger the right process (i.e. "load" successfully triggers the loading operations)	Demonstration	
T.1.2. Verify that the selected mode corresponds to the actual mode launched by software.	**T.1.2.1.** Verify that selecting load will trigger the loading operation.	Demonstration	**R.1.2.** Upon start-up, the user will be able to select if they want to load or start a new scenario. **R.1.1.**
	T.1.2.1. Verify that selecting new scenario will open the software in planning mode.	Demonstration	
T.1.3. Verify using a GUI map that the given GPS location is correct.	**T.1.3.1.** Verify that the computer connects to the GPS network	Demonstration/ Analysis	**R.1.3.** Computer will connect to the GPS network.
	T.1.3.2. Verify that the given GPS location is correct	Inspection/ Demonstration	
T.1.4. Test that computer receives a "sign-of-life" signal from a dummy drone (testing module).		Demonstration/ Analysis	**R.1.4.** Computer is communicating on the correct frequency/channel.
T.1.5. Successfully ping all drones connected on specified frequency/channel.		Demonstration	**R.1.5.** Computer successfully connects to all detected drones.
T.1.6. Verify that the GPS status of a connected dummy drone can be queried by software.		Demonstration	**R.1.6.** Successfully query the drone for GPS connection status.
T.1.7. Verify that the computer can receive GPS data from a connected drone (dummy drone), and that GPS data received is the same that is sent from a drone (dummy drone).	**T.1.7.1.** Verify that computer can receive GPS data from a connected drone	Analysis	**R.1.7.** Computer is able to receive GPS data from a connected drone.
	T.1.7.2. Verify that GPS data received is the same that was sent from drone	Testing/ Demonstration	
T.1.8.	**T.1.8.1.** Verify that all connected drones with GPS data appear on map.	Demonstration	**R.1.8.**

Test	Sub-tests	Type of test	Requirements tested
Verify that all connected drones with GPS data appear on map, and that no connected drones without GPS data appears on map.	**T.1.8.2.** Verify that no drones without data appear on the map.	Demonstration	Each connected drone that was connected to GPS is visible on a map as a dot or number.
T.1.9. Connect a dummy drone without GPS connection, verify that it appears in the designated part of the GUI.		Demonstration	**R.1.9.** Each drone without GPS position will be made visible in a designated part of the GUI with drone identifier.
T.2.1. Open the planner tab, and verify that the map for the given area appears on screen.		Demonstration	**R.2.1.** When opening the planner tab, map appears on software GUI
T.2.2 Verify that clicking the button will check GPS connection		Analysis/ Demonstration	**R.1.3.** Computer will connect to the GPS network.
T.2.3. Given the computer's current location is known, verify visually that the map has been centred on the correct location.		Demonstration	**R.2.2.** Visible map on GUI is centred upon the computer's location
T.2.4. Select an area and confirm that this area is accurately reflected in the selection on the GUI.	**T.2.4.1.** Verify that an area on the map can be selected.	Demonstration	**R.2.3.** Software allows for the selection of an area on the map **R.2.4.** The area selected will be graphically indicated
	T.2.4.2. Verify that the highlighted area is the same as the one that was actually selected.	Analysis	
T.2.5. When fine-tuning, user commands are accurately reflected in the updated restricted area.		Analysis/ Demonstration	**R.2.5.** When an area is selected, the user will be able to fine-tune selection using either drag-and drop mechanics, incremental arrow commands or to enter their own coordinates.
T.2.6. Selecting an area correctly triggers the selection menu to appear.		Demonstration	**R.2.6.** When an area is selected, a menu of options for selections will appear.
T.2.7. The restrict flight zone option appears on the menu when an area is selected on the map.		Demonstration	**R.2.7.** A selection for making a restricted flight zone is available
T.2.8. Test that selecting restrict flight zone option successfully changes mode.		Demonstration	**R.2.8.** Selecting the restricted flight zone option successfully allows for the definition of a no-fly zone.
T.2.9.		Demonstration	**R.2.9.**

Test	Sub-tests	Type of test	Requirements tested
Select known "no flight zone", cross-reference the resulting coordinates with the known ranges for the "no flight zone" and ensure they are the same.			A range of forbidden coordinates will be generated when area has been selected
T.2.10. When trying to make a route including any restricted coordinates, the system will return an error and not generate a route.	**T.2.10.1.** Making a route including restricted coordinates returns an error.	Demonstration	**R.2.10.** User is not allowed to create routes including any drone at any of the restricted coordinates
	T.2.10.2. Making a route including restricted coordinates does not generate a route.	Demonstration	
T.3.1 Clicking the map will successfully show the dropdown menu		Demonstration	**R.3.1** When user clicks on the map in the GUI, a dropdown menu of available actions appears
T.3.2 Test selecting a coordinate with known height data to verify the returned data is correct.		Demonstration	**R.3.2** Height database is queried for data on a given coordinate. It will return either the height data or a NULL value. **R.3.3** Software will display available height data for the selected point.
T.3.3. Test selecting a coordinate with no known height data to verify that the returned data is correct.		Demonstration	**R.3.2.**
T.3.4. Test selecting coordinate with known data NULL, verify the software returns an error message and no further action is taken.		Analysis/ Demonstration	**T.3.4.** Software will display an error message and no more data is returned.
T.3.5. When in the simulation mode of the software try to select a point where you want to start and make sure the software collects information about the correct area		Testing/ Analysis	**R.3.5.** The software needs to be able to collect data and information about buildings and the landscape of specified locations
T.4.1. Verify that the option to designate a safe landing zone appears in the menu when user selects an area		Demonstration	**R.4.1.** The user will be able to select a "designate safe landing zones" option
T.4.2. Verify that the selected area corresponds to a populated list of coordinates.		Demonstration	**R.4.2.1, R.4.2.2** Safe landing zone coordinate are displayed and populated based on user selection

Test	Sub-tests	Type of test	Requirements tested
T.4.3. Verify that the scenario correctly includes the range of safe landing zones selected by the user.		Inspection	**R.4.3.** A range of safe landing zone coordinated coordinates will be generated when area has been selected
T.5.1. Test that the point is marked with an icon when "Illuminate this point" is selected.		Demonstration	**R.5.1.** When user clicks on the "Illuminate this point" option, the point is marked on the GUI map.
T.5.2. Verify that the marked point is the same as the selected point.		Inspection	**R.5.1.** When user clicks on the "Illuminate this point" option, the point is marked on the GUI map.
T.5.3. Verify that the returned coordinates are the same as a known test coordinate.		Testing/ Analysis	**R.5.2.** System can query the map database for coordinates of the selected point.
T.5.4. Verify that information successfully appears when a user requests to illuminate a point, and that data it is populated with is accurate.	**T.5.4.1.** Verify that information successfully appears when user requests to illuminate point (in a pane on the side, window or similar)	Demonstration	**R.5.3.** Queried information on coordinates and height will be visible in a separate planning pane, where the user can enter information
	T.5.4.2. Verify that information populated is accurate.	Demonstration	
T.5.5. Verify that the entered height data is the same as what appears in the route.		Analysis/ Demonstration	**R.5.4.** When height data is entered into planning pane, the backend planning data is successfully updated to reflect this change.
T.5.6. Simulate route and verify that drone reaches desired coordinates and height.		Demonstration	**R.5.5.** System generates route based on desired coordinates and height of illumination.
T.6.1 Verify that clicking the "Save" button will launch the save dialogue box.		Demonstration	**R.6.1.** Clicking the save route button prompts a dialogue box for saving the route.
T.6.2. Verify that file is created.		Inspection	**R.6.2.** The current scenario will be recorded accurately in a JSON file.
T.6.3. Load file and verify that it is identical to the scenario saved earlier.		Testing/ Inspetion	**R.6.2.** The current scenario will be recorded accurately in a JSON file.
T. 6.4. Verify that the user can successfully save a file in a custom location.		Demonstration/ Testing	**R.6.3.** User is allowed to define where the file should be saved.
T.6.5. When "Load" is clicked, a list of routes will be made available.		Demonstration	**R.6.4.** Saved routes are listed out for the user to select from.

Test	Sub-tests	Type of test	Requirements tested
T.6.6. When "Browse" is pressed, verify that explorer window appears.		Demonstration	**R.6.5.** User can browse through files to find correct route.
T.6.7. Verify that the correct scenario opens when selected.		Demonstration	**R.6.6.** When a scenario has been selected, Hivemind will open the scenario.
T.6.8. Verify that the populated options and coordinates match with a given, known test route.		Analysis/ Demonstration	**R.6.7.** When a scenario is opened, all saved data such as chosen options and current list of coordinates populates relevant fields in the planning panes.
T.6.9. Press "new scenario", and verify that the dialogue box to enter coordinates appears as expected.		Demonstration	**R.6.8.** When user selects new scenario, some UI element allowing them to enter start coordinates should appear
T.6.10. Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.		Demonstration	**R.6.9.** When a keyframe is entered, a new keyframe is successfully added to graphical user interface **R.6.10.** When a keyframe is entered, a new keyframe appears in the keyframe list corresponding to entered coordinates.
T.6.12. Verify that added keyframe is correctly assigned on the timeline		Demonstration/ Analysis	**R.6.11.** When a coordinate is added, a new keyframe appears on the timeline corresponding to the relative timestamp of the keyframe
T.6.13. Add a coordinate, and verify that the distribution of keyframes on timeline dynamically updates.		Demonstration	**R.6.12.** The position of keyframes on the timeline should correspond to the length of the scenario and update dynamically
T.6.15. Press compile scenario, and verify that a route is created, and that information in route is correct.		Demonstration	**R.6.14.** Key frames for drones are generated into a route. **R.6.15.** All routes are successfully compiled into a scenario.
T.6.16. Interact with the timeline and verify that the animations correspond to the keyframes.		Demonstration	**R.6.16.** The animation of keyframes on the map corresponds to their timestamps.

Test	Sub-tests	Type of test	Requirements tested
T.7.1. When the software is open try to select simulation mode and make sure the software enters the desired state		Demonstration	**R.7.1.** The software needs a GUI for a simulation mode
T.7.3. When in the simulation mode of the software try to start a simulation and make sure it uses the correct start location and behave as expected		Demonstration	**R.7.3.** The software needs the ability to simulate the planned route in a visual representation
T.8.1. Enter a coordinate and check that you get expected data in return.		Demonstration	**R.8.1.** The software has to be able to convert between coordinates in geographical coordinate space, Universal Transverse Mercator coordinate space and cartesian space.
T.9.1. Connect several drone and have them do different movements to make sure it is consistent with what shows up in the software		Demonstration	**R.9.1.** The software has to be able to keep track of which drone is which
T.9.2. Have a dummy drone simulate having trouble by sending a signal to the software as if the drone was having an issue and make sure the correct drone is highlighted in the software		Demonstration	**R.9.2.** The software has to be able to highlight drones that are having problems so that they are easy to identify for the drone operator. **R.9.3.** Has to be able to receive information from the drones and keep track of which drone sent the information
T.10.1. Have a dummy drone send a signal and pretend like it is a struggling drone and make sure the software is able to highlight the drone		Demonstration	**R.10.1.** Has to be able to receive information from the drones and keep track of which drone sent the information
T.10.2. Click on a drone when it is highlighted and make sure you get the options to either emergency land it or to let it keep doing the planned route		Demonstration	**R.10.2.** Has to be able to show a menu/pop-up window when the drone operator clicks on a highlighted drone
T.10.3. Have a dummy drone pretend to fly a route and try to make it abort it's route and land		Demonstration	**R.10.3.** Has to be able to overwrite the planned route for the drones
T.10.4.		Demonstration	**R.10.4.**

Test	Sub-tests	Type of test	Requirements tested
Make a dummy drone simulate having a problem and ignore it in the software			The software has to be able to let drones keep flying even if they have certain faults
T.11.1. When planning a scenario you select a drone and make it the leader.		Demonstration	**R.11.1.** Has to be able to highlight a drone when it is selected
T.11.2. After making a drone the leader tell the software you have more drones and tell it you want a specific formation and make sure the scenario is doable by simulating the routes it made		Demonstration	**R.11.2.** The software has to be able to make routes for several drones
T.11.3. Test that adding new agent through interface causes the new drone to correctly appear		Demonstration	**R.11.3.** The user can add new agents through the interface. **R.11.4.** The new agent is correctly displayed in the user interface. **R.11.5** All added agents should be uniquely identified in the interface. **R.11.6.** Active agent is changed in system back-end **R.11.7.** Active agent is indicated in user interface **R.11.8.** The agent ID in the new keyframe is the same as the active drone.
T.11.6. Try compiling a route where the agents are on a clear collision course and confirm that collisions are avoided through adjusting time stamps or routes.		Demonstration	**R.11.9.** When generating a route for multiple agents, the algorithm shall ensure the drones do not collide.
T.11.7. Try setting up a route where there is no way to avoid a collision and confirm that no routes are generated and an error is returned.		Demonstration	**R.11.10.** If the algorithm is unable to prevent a collision, the route should not be generated and an error returned.
T.11.8 Compile scenario and ensure all agent routes are included.		Demonstration	**R.11.11.** The compiled scenario should include all agent routes.

Appendix H

Development of the software architecture

Developing a stable architecture has been a continuous process since the end of the first presentation. The architecture has gone through several iterations, developed using various methods and envisioned to meet different goals before arriving at its current version. This section will detail the considerations made before embarking upon the architectural design process, the different versions of the architecture and how we arrived on them, and the final, stable version of our architecture for the minimum viable product.

1 Initial Architectural Design

HMM | *AM*

The Hivemind software will in time be able to serve a significant number of agents, and should be able to monitor their process in real-time as well as give emergency commands with low latency. As a result, Hivemind will need to be stable and efficient. The architecture should facilitate the development of software that can meet these requirements.

In addition, there are two considerations that give rise to more requirements for the chosen architecture. First, due to time constraints, it is unlikely that the project will be able to complete the full software as imagined by the client. Instead, a minimum viable product will be completed first, which means that the architecture should facilitate the ability to add new features to the software and to increase the amount of agents it can handle as development progresses.

Second, the project will ultimately not be finished by this group, but will be continued on by another team in the future. This means the architecture should be easy to understand and easy to maintain, even for someone who has not been a part of the project from its inception. The architecture should also facilitate the software development process and help logically divide the entire software into programmable components.

The criteria for an appropriate architectural pattern for this project is therefore (in no particular order):

- Scalability
- Clarity
- Adaptability
- Stability

At the beginning of the architectural development process, we also spent some time considering different architectural patterns. The group has the most exposure to layered architecture, and therefore compared other architectural patterns, such as event-driven architecture, with layered architectures and considered both against our needs. In the end, it was determined that a layered architecture was the most appropriate choice to describe the high level functions that the software will be divided into, though it is apparent in the evolution of the architecture that will be outlined below that the group was influenced by other ways of presenting software.

1.1 Initial component diagram

HMM | *AM*

The first diagram that was created was a simple component diagram, containing five main components - Graphical user interface (GUI), Status, Routeplanning, Simulation and Execution. This was meant as a component diagram for the entire software, see figure H.1.

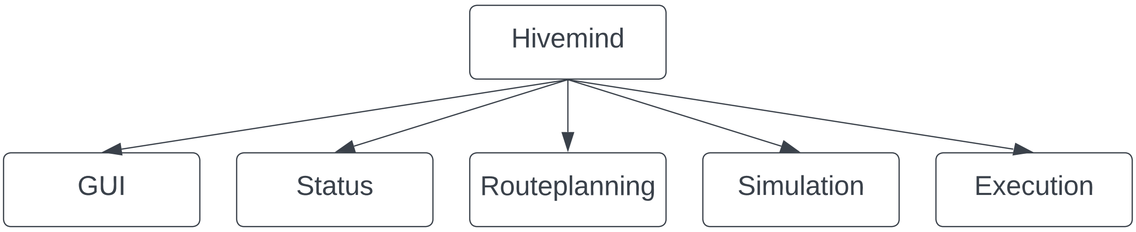


Figure H.1: The initial component diagram

This was further broken down into sub-components and functions, and included annotations indicating which of the requirements each sub-component or function would fulfil (the full-scale diagram is attached in the appendix), see figure H.2.

Although little of this initial diagram has made it into the final, stable architecture, it was nevertheless a crucial step in starting to considering what the Hivemind software is meant to do, and how it could do this. It was decided at this point that the minimum viable product would not include sending new routes to drones in-flight or emergency landings.

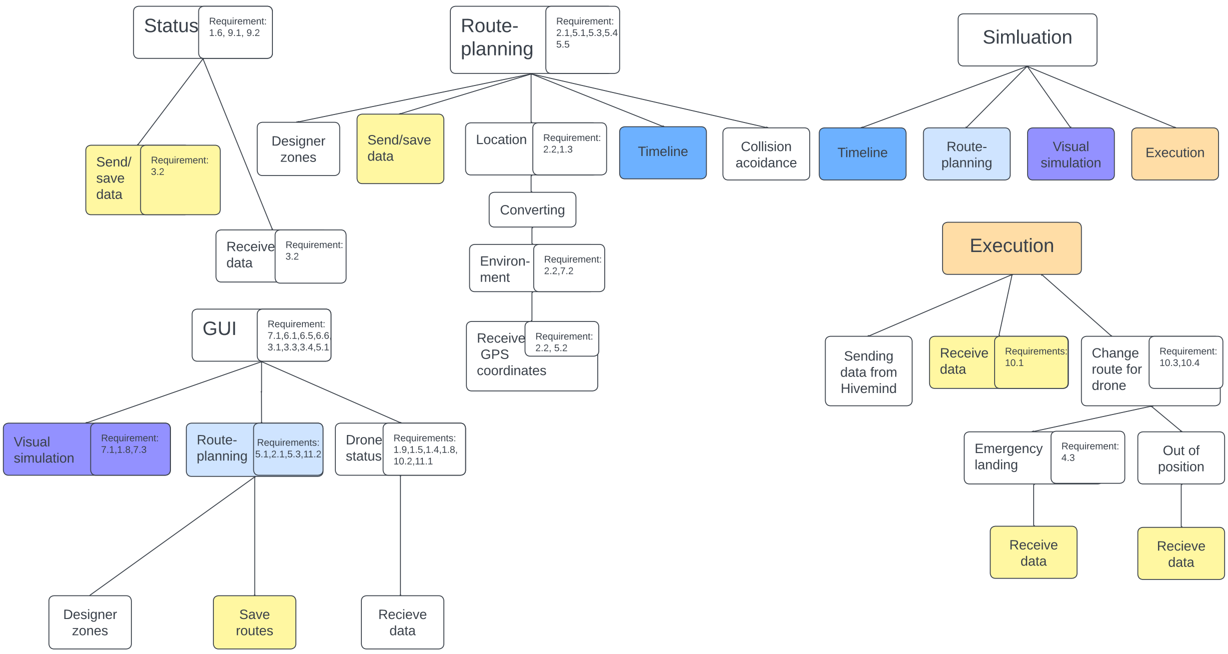


Figure H.2: The initial component diagram

1.2 First architecture & model-view-controller diagram

HMM | *AM*

After mapping out components, the group attempted to create a more functional view of the software through a layered architecture, figure H.3, and a model-view-controller diagram, figure H.4.

Mapping these out yielded two results: 1, we realized that the minimum viable product was still too large for this product and needed to be reduced in scale and 2, while stile not good ways of illustrating the functionality of our software, it helped further crystallize our understanding of which functions are necessary, and what they would do.

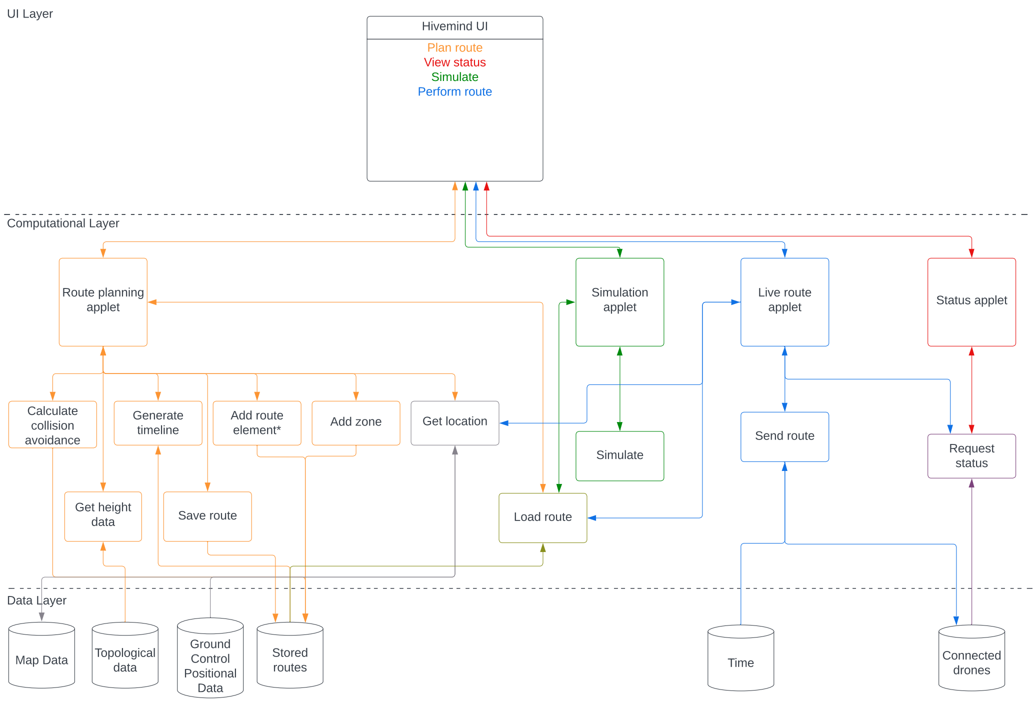


Figure H.3: The first layered architecture

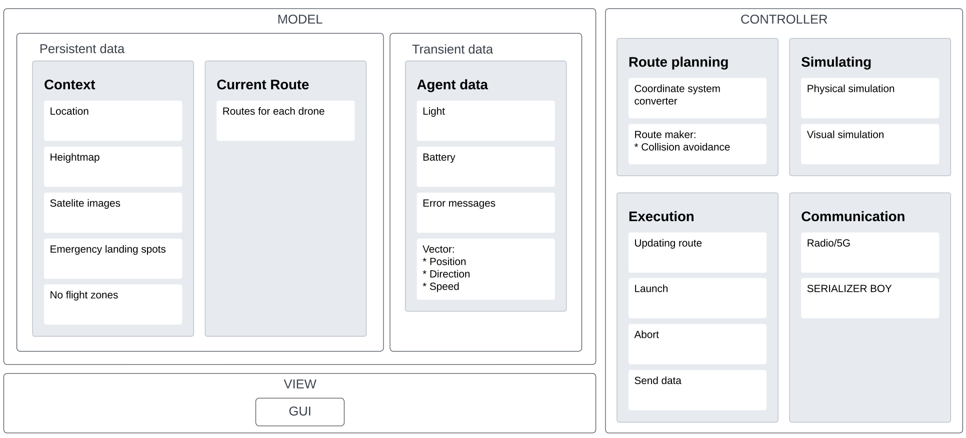


Figure H.4: The derived MVC diagram

1.3 Second layered architecture and use cases **HMM** | *AM*

The next iteration of our architecture came after redefining the minimum viable product and deciding deciding to use a layered architecture to represent the functionality of the software. This version of the architecture features functions divided into coloured boxes indicating which main functionality they belonged to, and horizontal lines indicating interactions between functions, see figure H.5.

At this point, all the functions and their interactions are becoming more clear and defined, though the architecture itself is still somewhat hard to read. Further work to simplify and present the abstract functionality of the software was therefore necessary.

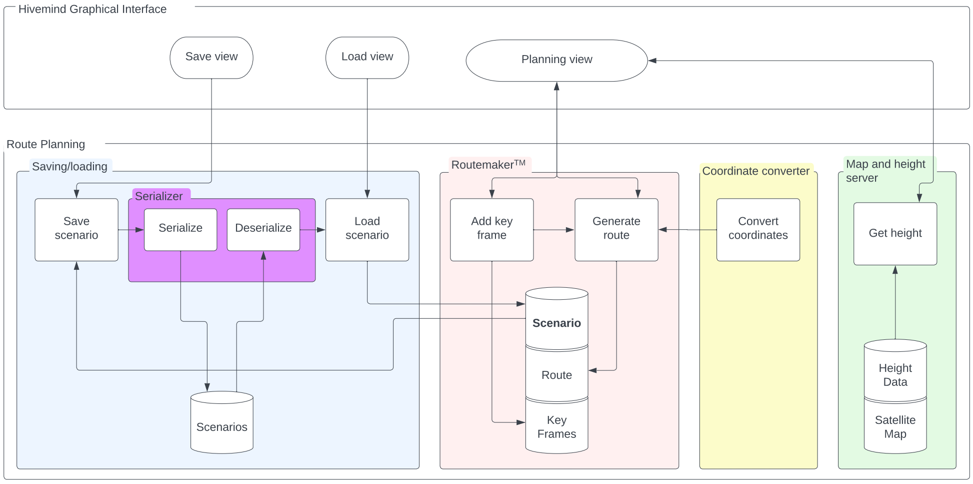


Figure H.5: The second layered architecture

1.4 Third layered architecture **HMM** | *AM*

When making the third layered architecture, we first started defining use cases, see figure H.6, and then derived the architecture from these use cases, see figure H.7. In these use cases, we imagine that the actor (or route planner) has only three options when interacting with the Hivemind software: save scenario, load scenario, or add key frame. All other actions are derived from these three.

As is apparent, the third architecture is much improved in terms of readability and clarity of the main abstract functionality of Hivemind. Taking the knowledge gained from this iteration of the architecture, the group re-did the architecture a final time, landing on the stable architecture that is currently being used for future development. That being said, the architecture of Hivemind remains an element of paramount importance, and is still being considered a living document, and as more is discovered about limitations and opportunities derived from the functions of the minimum viable product, the architecture itself is also subject to adjustments.

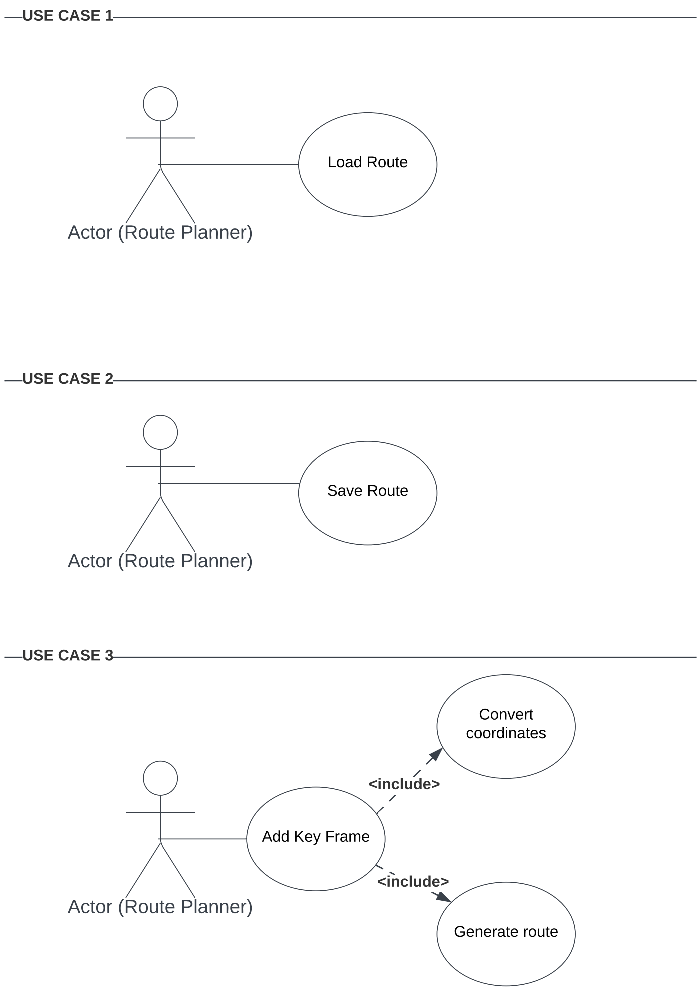


Figure H.6: Use cases for the third layered diagram

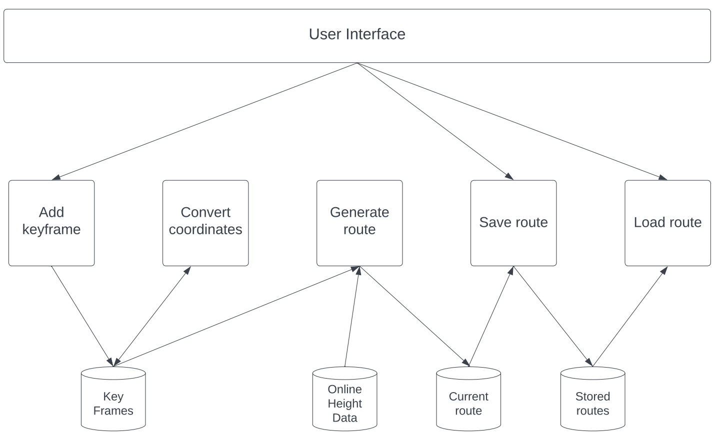


Figure H.7: The third layered architecture

1.5 Final layered architecture before start coding **AM** | *RS*

In our efforts to determine the architecture for our software system, we adopted an approach consisting of a logical architecture and a design architecture. Our chosen architectures are based on a three-layered architecture, where the logical architecture is built upon the use case diagram that we have developed. In contrast, the design architecture is based on the logical architecture and the components we have created. The logical architecture serves as the foundation of our software system, shown in fig. H.8. This architecture provides a high-level view of the system, describing the absolute major components, their interactions, and the data flow between them. The logical architecture primarily concerns the system's behaviour, functionality, and performance.

On top of the logical architecture, we developed a design architecture that focuses on the physical implementation of the system, shown in fig. H.9. This architecture defines the detailed structure and organization of the system's components, their relationships, and their interactions. The design architecture considers the constraints of the underlying technology and aims to optimize the system's performance, scalability, and maintainability.

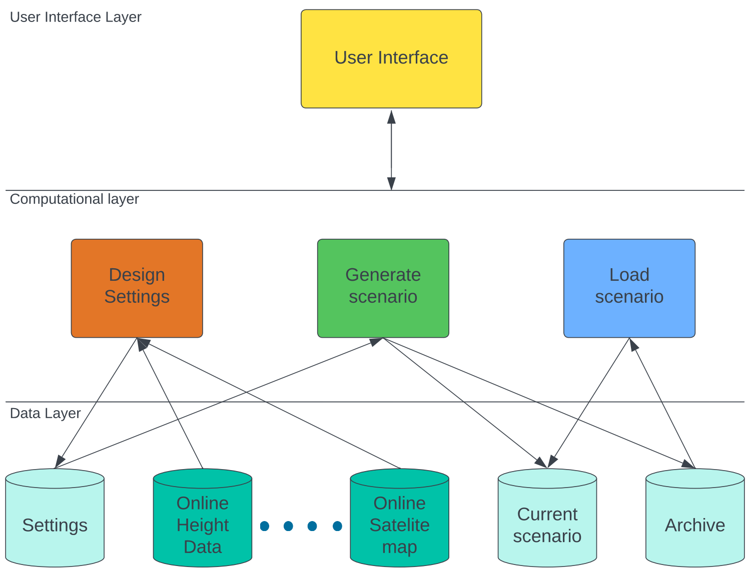


Figure H.8: The first high level architecture

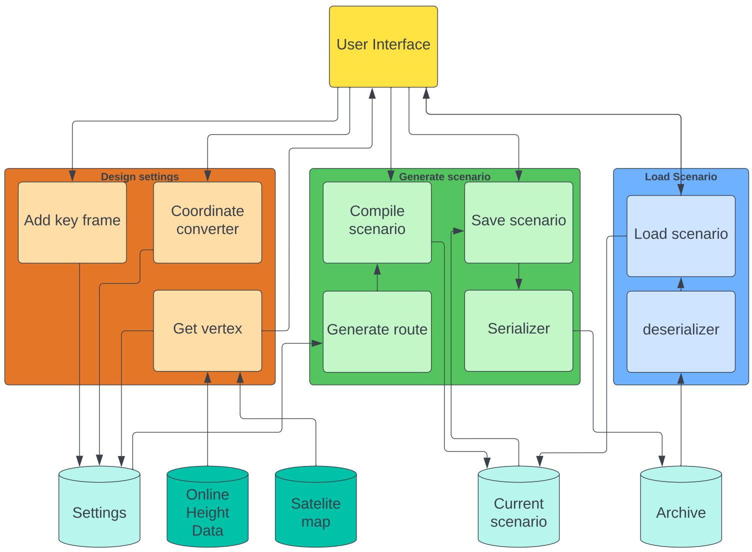


Figure H.9: The first desgin architecture

2 Adapting Architecture for Coding Challenges & Requirements

AM | *RS*

During the coding phase, several challenges arose, leading to changes in the architecture. The implementation revealed that there were several architectural modifications that could have been made to make the system more robust. This chapter will show how our architecture has developed during the implementation phase, and explain each change that has been introduced.

In the initial coding components developed were the serializer and deserializer two separate components. It became apparent that these components were responsible for much of the saving and loading functionality. As a result, it was decided to merge them into two single components instead of four separate ones, and therefore save and serializer one component(save), and deserializer and load were one component(load) as in fig. H.10.

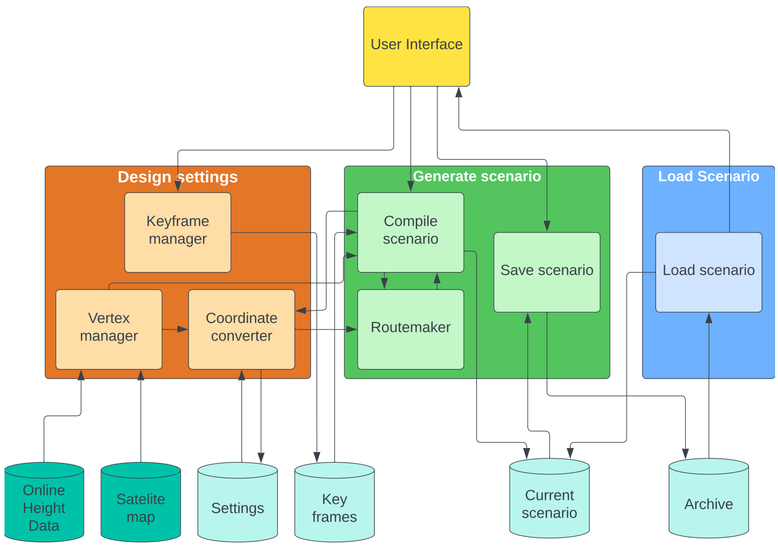


Figure H.10: Merge the serializer and deserializer components with the save and load components.

After merging the serializer and deserializer with save and load, it was important to establish proper connections between the components and the data layer. Fig. H.11 illustrates one of the attempts made.

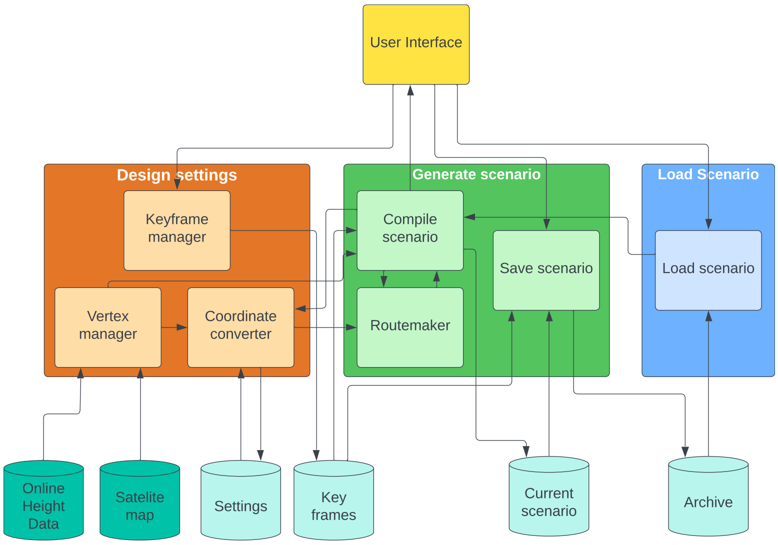


Figure H.11: An attempt to connect the data layer and components together.

When coding the following components, such as the keyframe manager, vertex manager, route maker, and compile scenario, many of these components utilized the coordinate converter. It was determined that treating the coordinate converter as a utility function rather than a separate component would be more efficient. Therefore, it was removed from the architecture and it was regarded as a utility function, see fig. H.12.

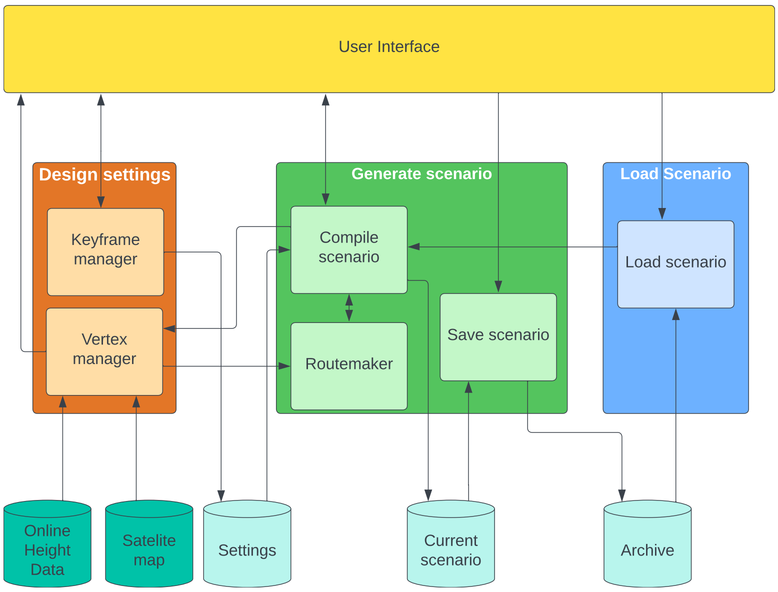


Figure H.12: Removed the coordinate converter from the architecture.

During the development of Hivemind, one of the challenges encountered was visualizing the data layer effectively. Several changes were made to the data layer throughout the architecture's development. The objective was to eliminate any interfaces between the use case boxes. Although fig. H.13 attempted to address this issue, there were still some remaining interfaces between these components.

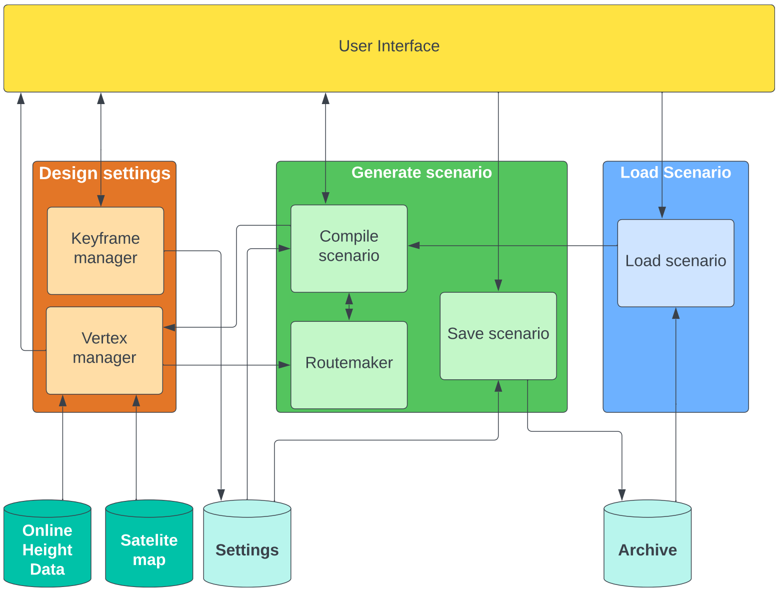


Figure H.13: Removed horizontal lines between the use cases.

During the development of Hivemind, when the architecture was closely aligned with the system, the team split the vertex manager into two separate components. Initially, to height and map data, which has different functionality. It was determined that separating them into distinct components would be better. As a result, the map manager and height manager were developed, as shown in fig. H.14.

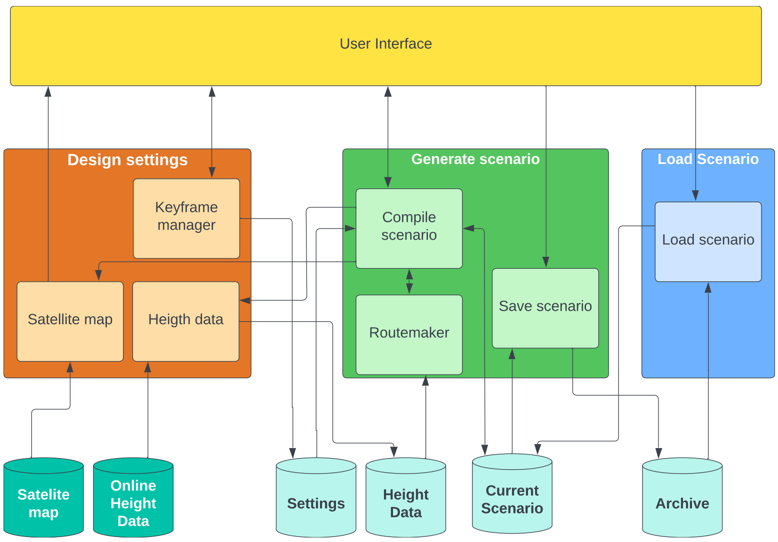


Figure H.14: Vertex manager splited into map manager and height data.

Once the components were in place, it was essential to structure the architecture in line with the system and make it easier to understand how the system was built. Since the code was organized such that the map and height data were part of a scenario, these components were moved into the "generate scenario" use case. This change also eliminated the horizontal lines between the use cases. Please refer to Fig. H.15 for visualization. The name for the map manager was changed to map management, height manager was changed to height management and save scenario was changes to archive scenario. The team did this to ensure that the name of each component aligned better with its actual functionality.

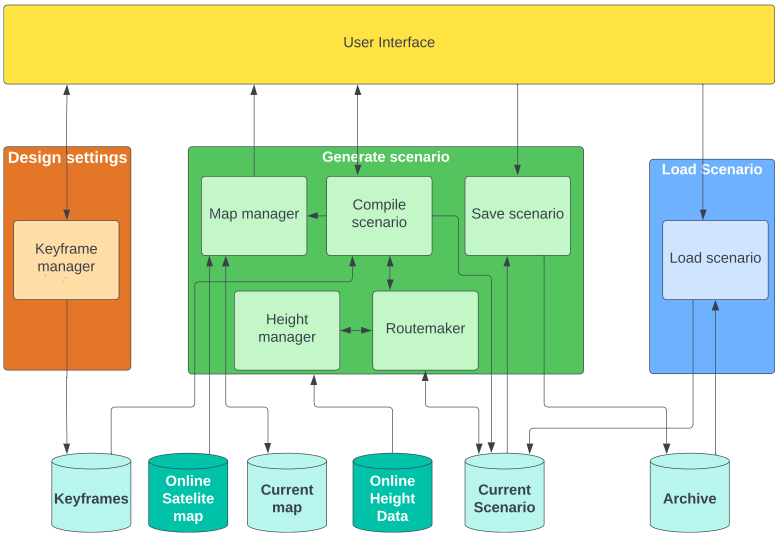


Figure H.15: The placements of some of the components have been rearranged.

The team realized that it was not appropriate to have the coordinate converter as a global function. Therefore, it was reintegrated into the architecture. This also resolved the issue of horizontal arrows between the components as shown in fig. H.16

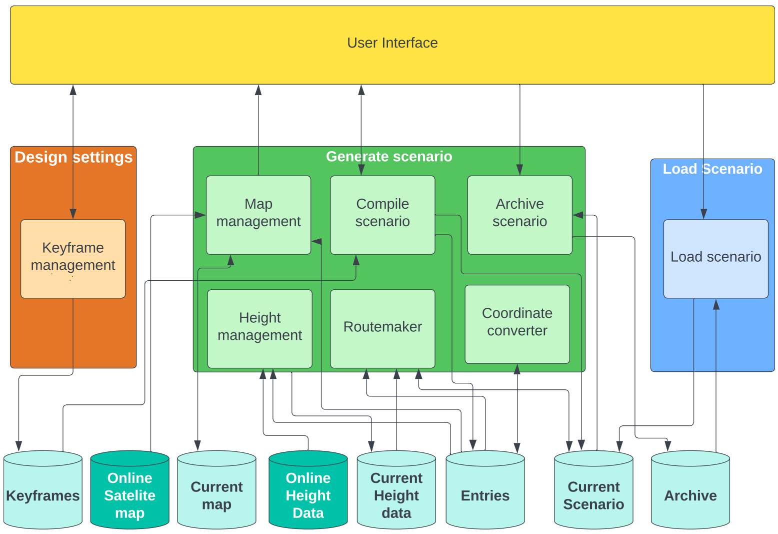


Figure H.16: Reintegrated the coordinate converter.

When the architecture containing the components aligned with Hivemind, the data layer in the high-level architecture was updated as shown in the fig. H.17.

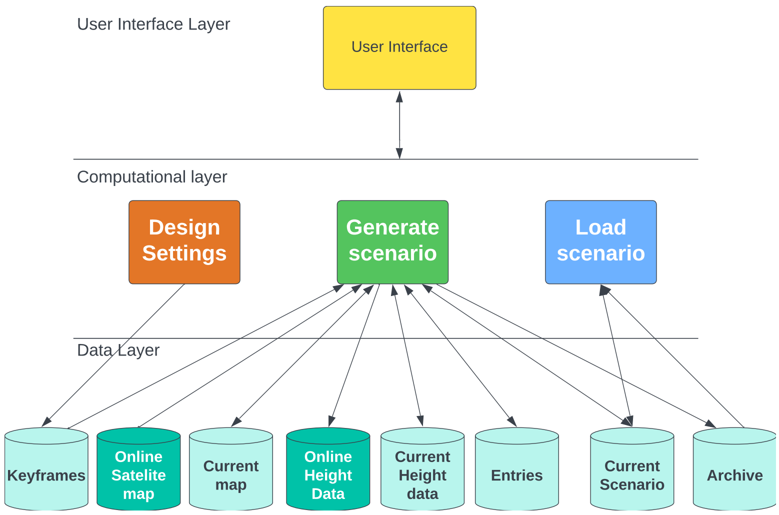
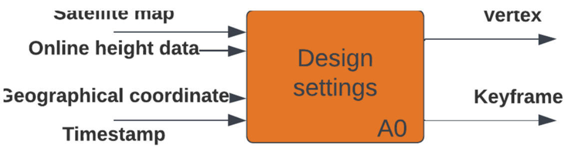
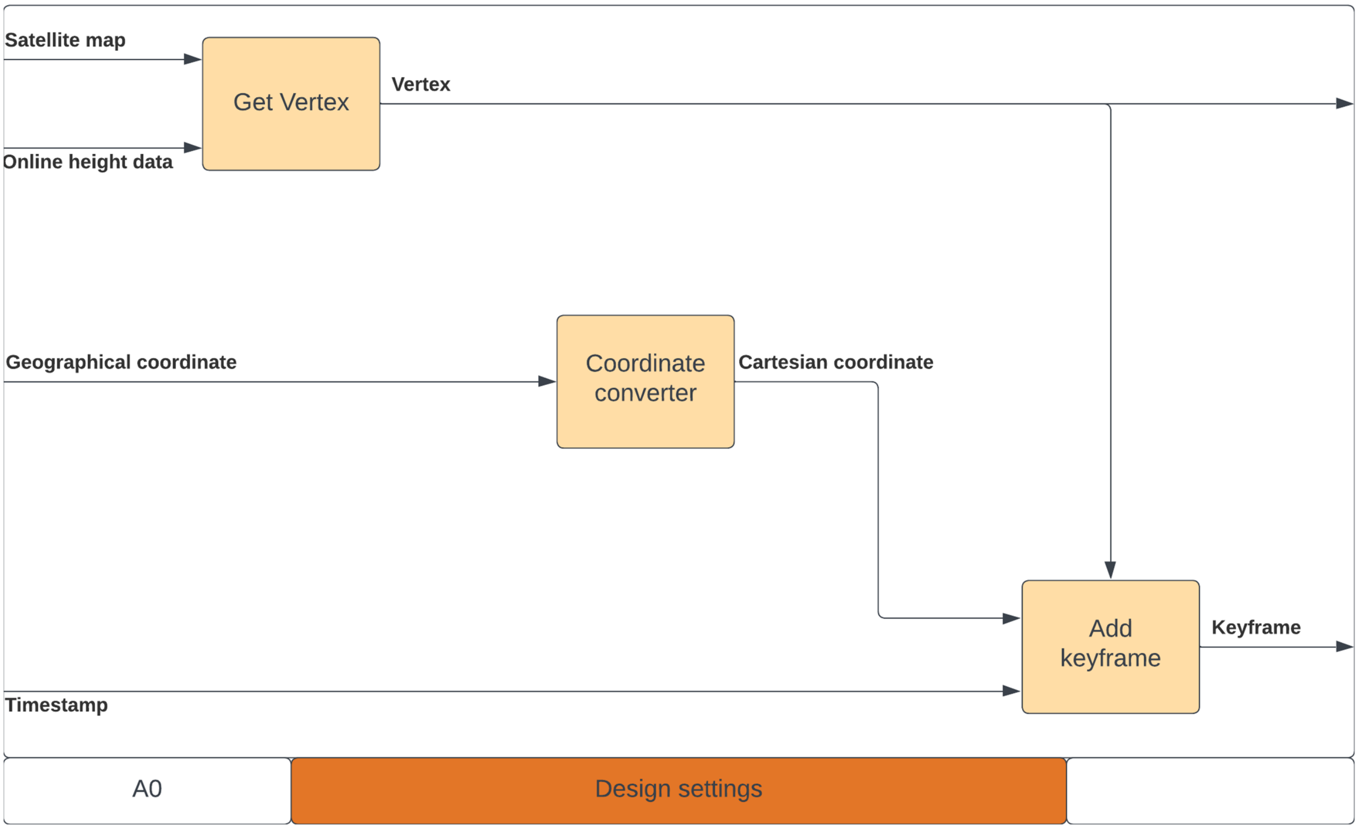


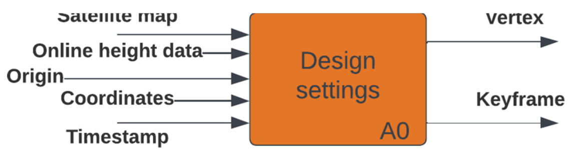
Figure H.17: Final high-level architecture.

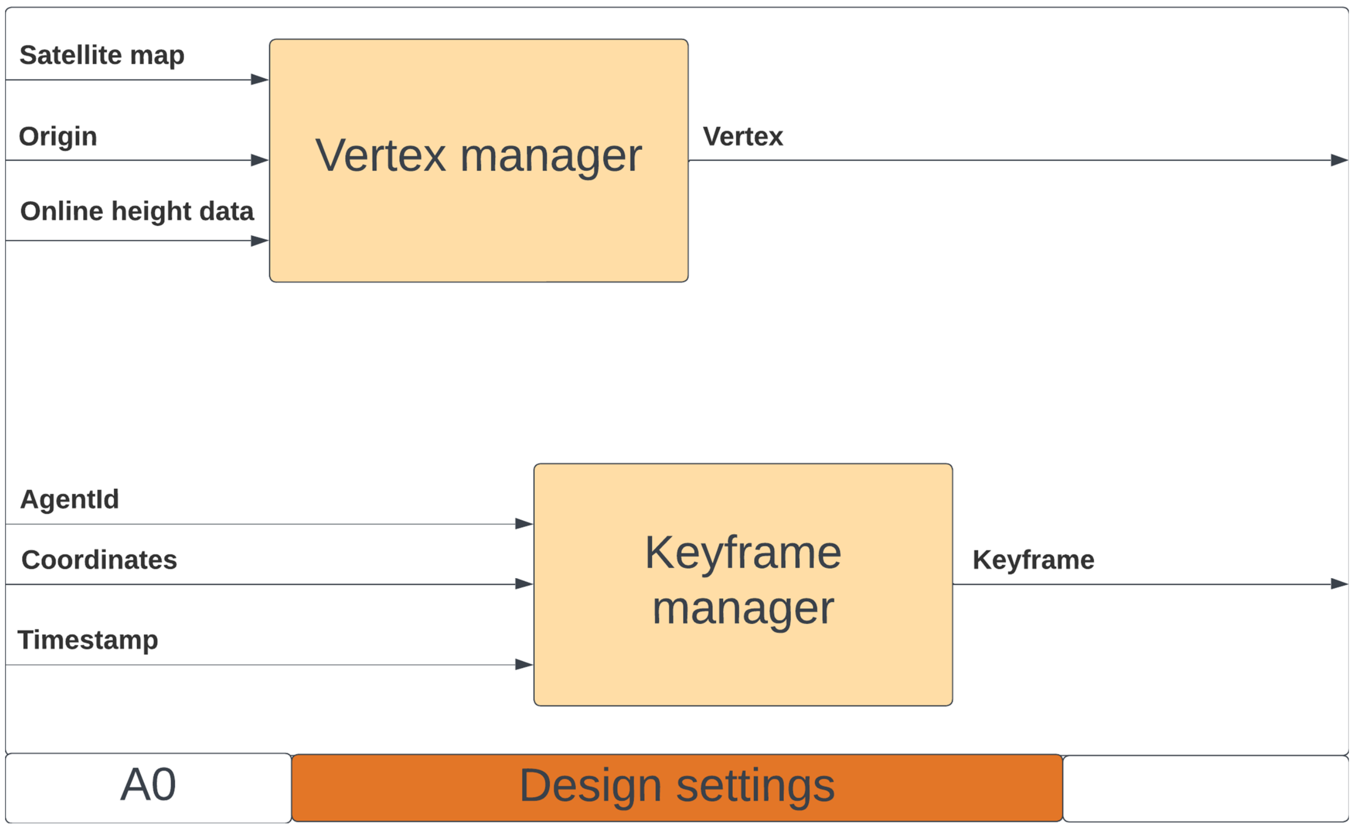
Appendix I

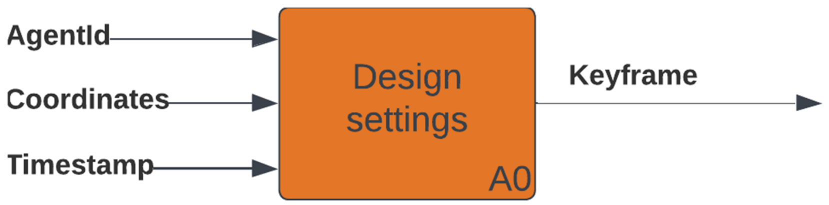
IDEF0

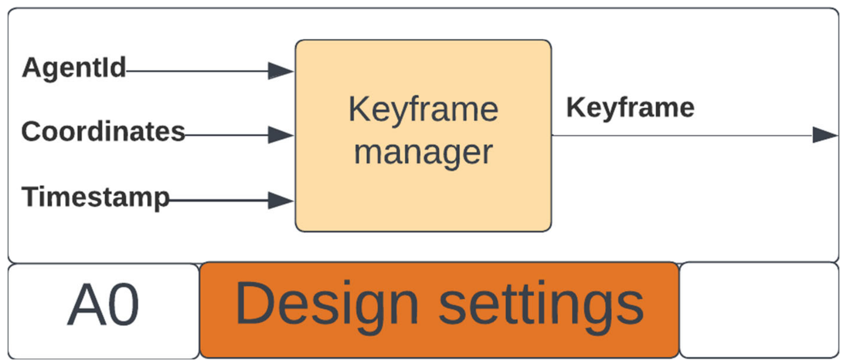

Online height data
Timestamp
Design settings
A0
Keyframe

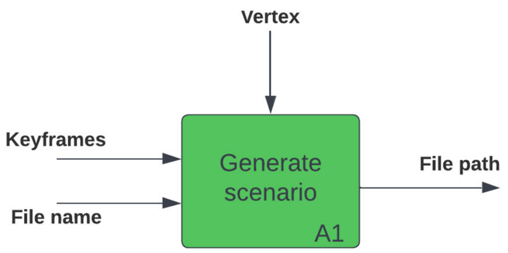

Satellite map
Online height data
Get Vertex
Vertex
Geographical coordinate
Coordinate converter
Cartesian coordinate
Add keyframe
Keyframe
Timestamp
A0
Design settings

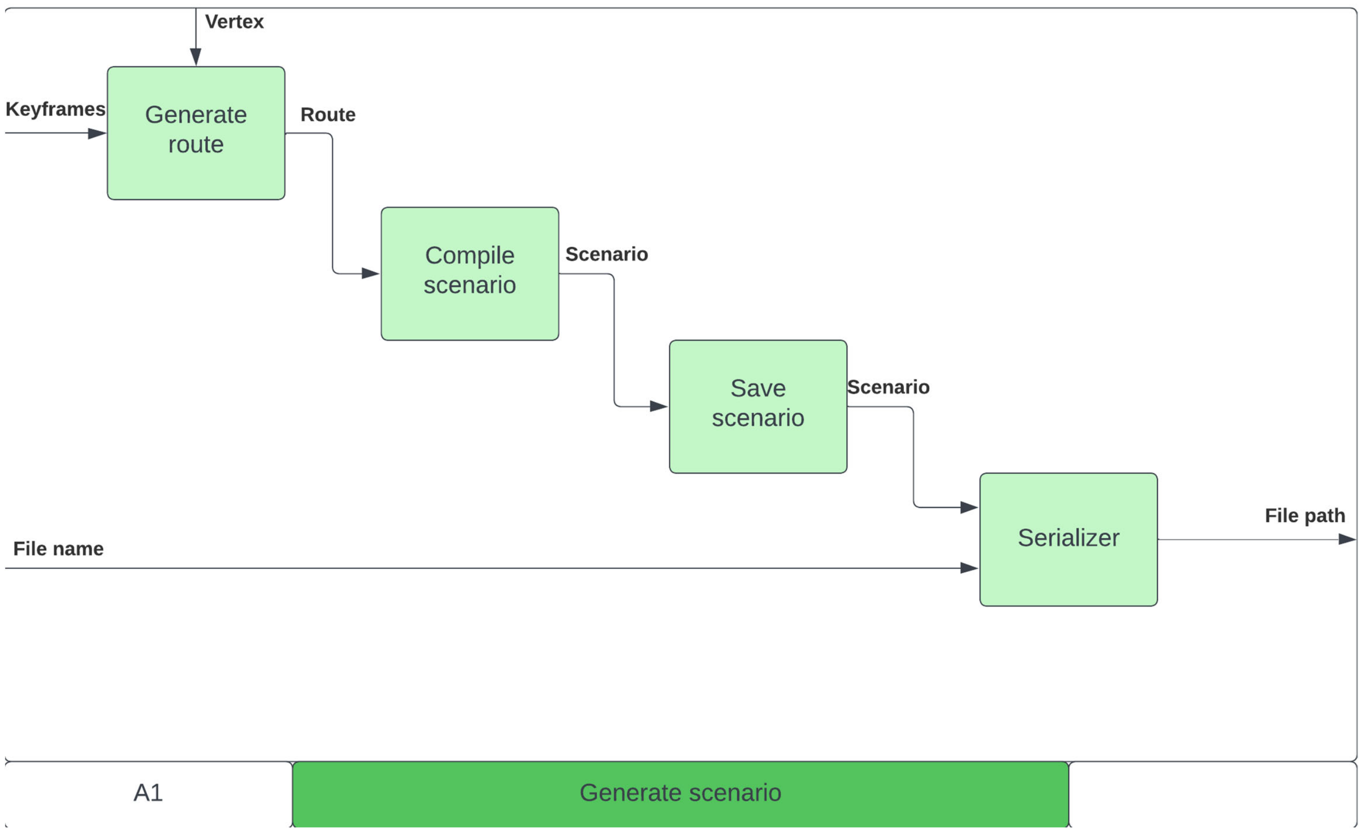

Online height data
Origin
Coordinates
Timestamp
Design settings
A0
Keyframe

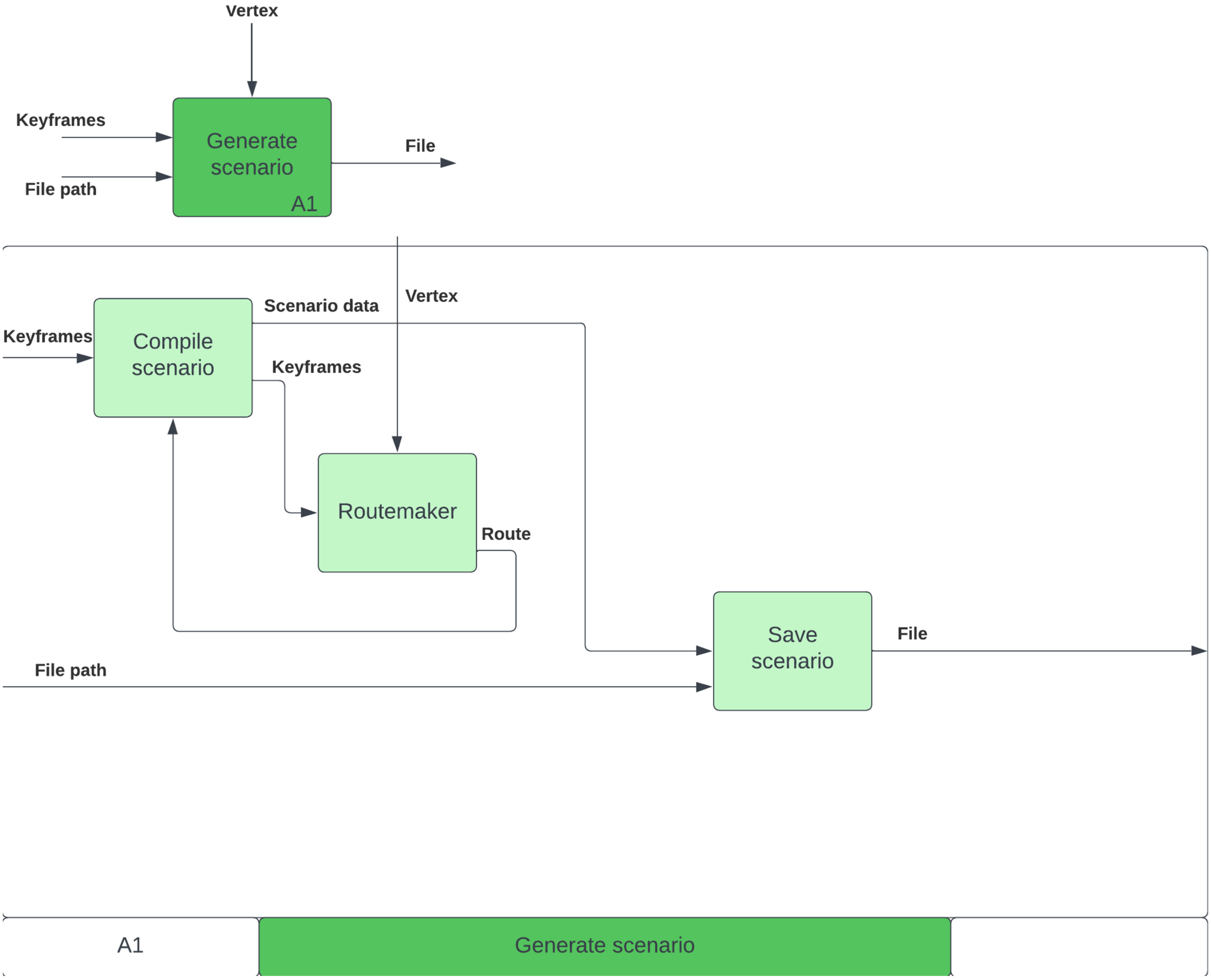

Satellite map
Origin
Online height data
Vertex manager
Vertex
AgentId
Coordinates
Timestamp
Keyframe manager
Keyframe
A0
Design settings

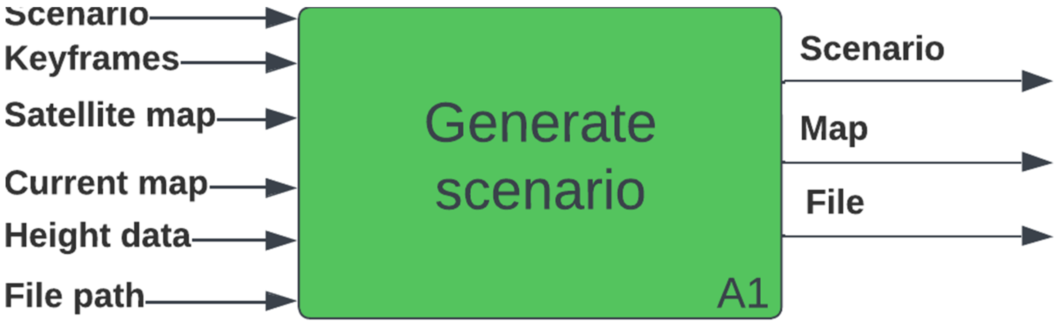

AgentId
Coordinates
Timestamp
Design settings
A0
Keyframe

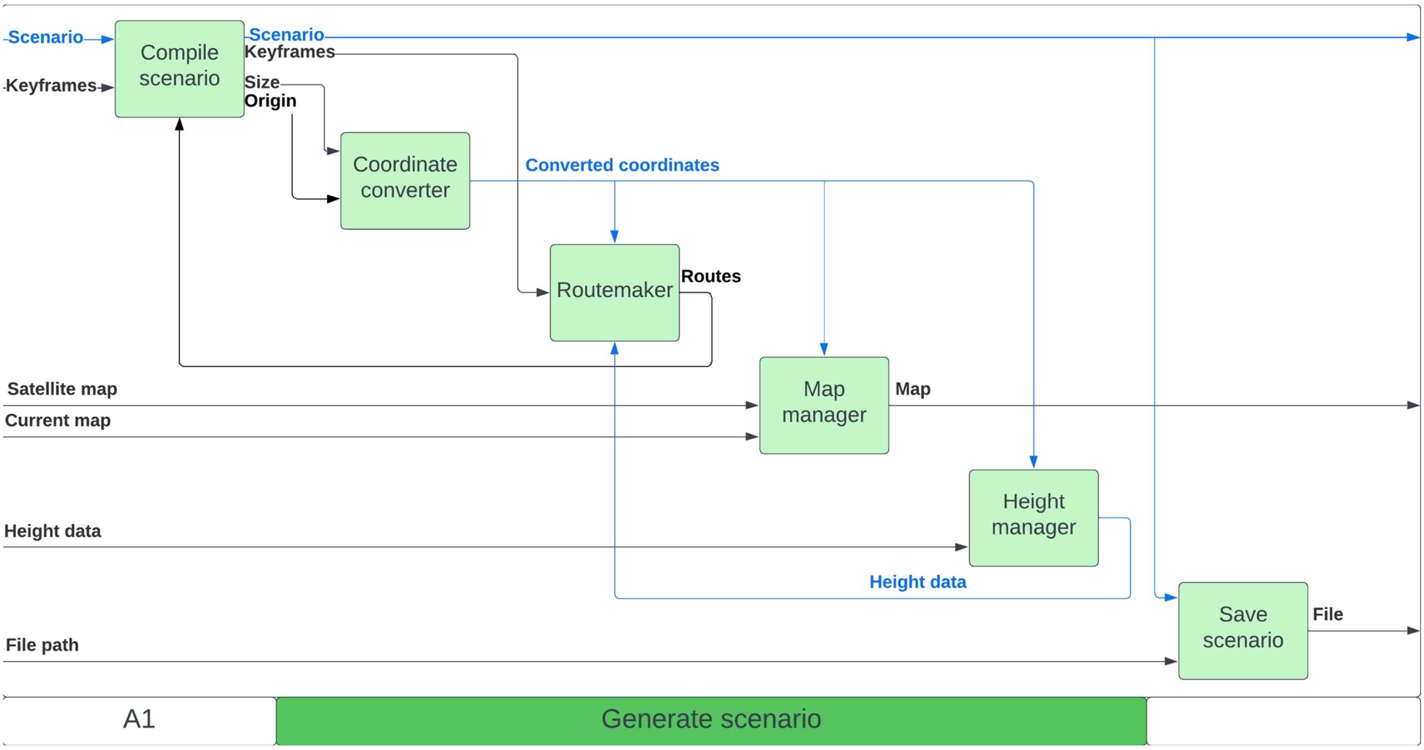

AgentId
Coordinates
Timestamp
Keyframe manager
Keyframe
A0
Design settings

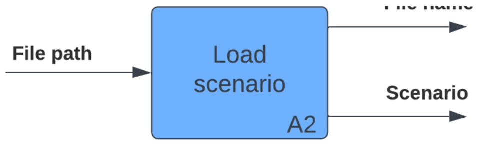

Vertex
Keyframes
File name
Generate scenario
A1
File path

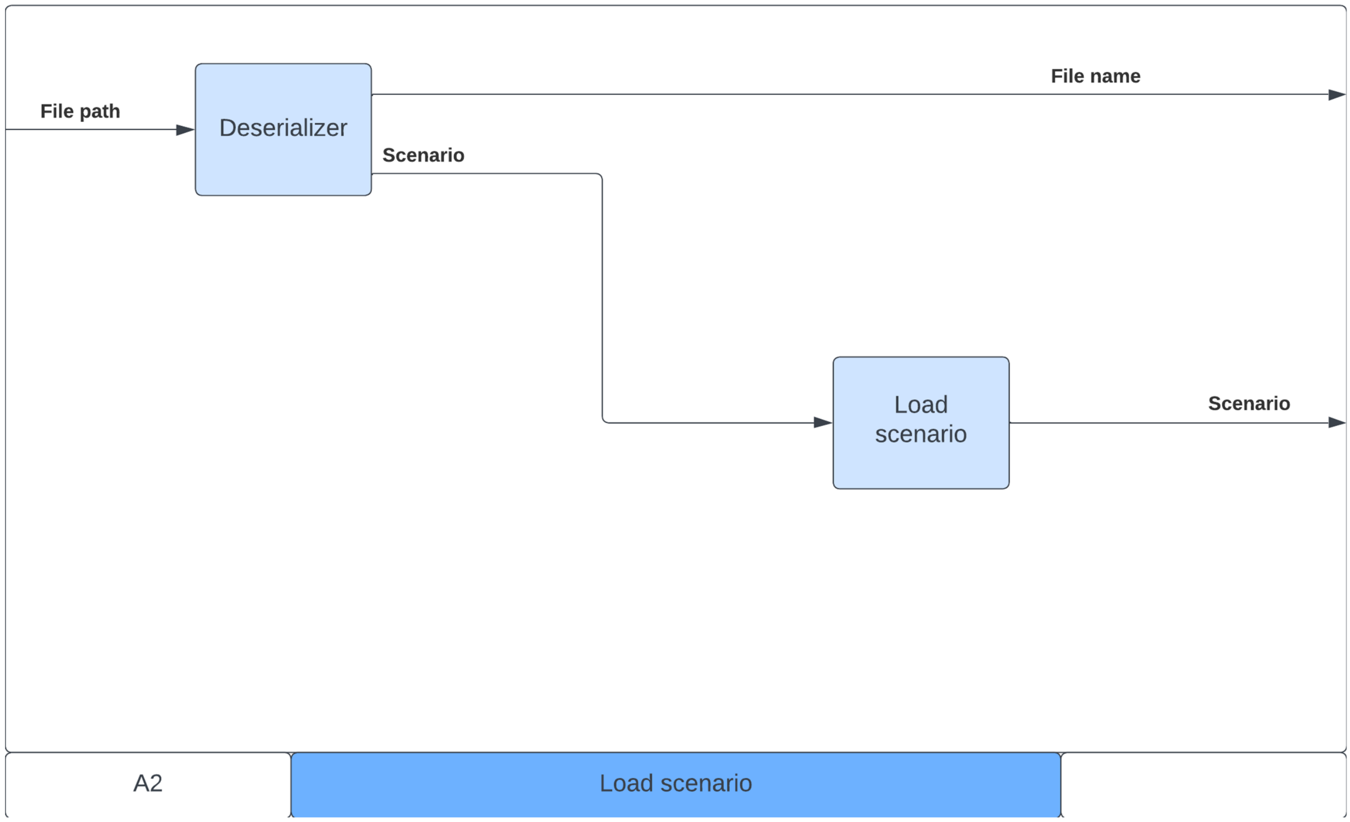

Vertex
Keyframes
Generate route
Route
Compile scenario
Scenario
Save scenario
Scenario
Serializer
File name
File path
A1
Generate scenario

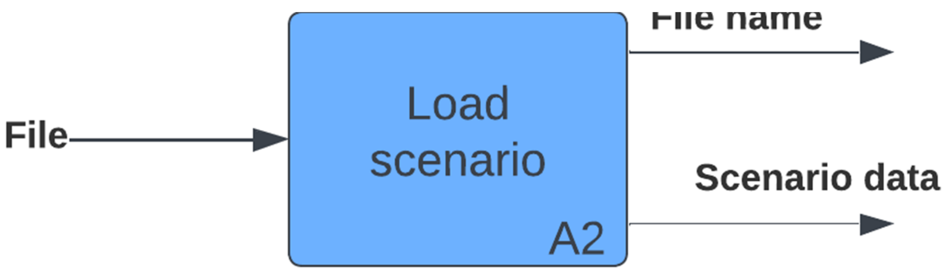

Vertex
Keyframes
Generate scenario
A1
File
File path
Scenario data
Vertex
Keyframes
Compile scenario
Keyframes
Routemaker
Route
Save scenario
File
File path
A1
Generate scenario

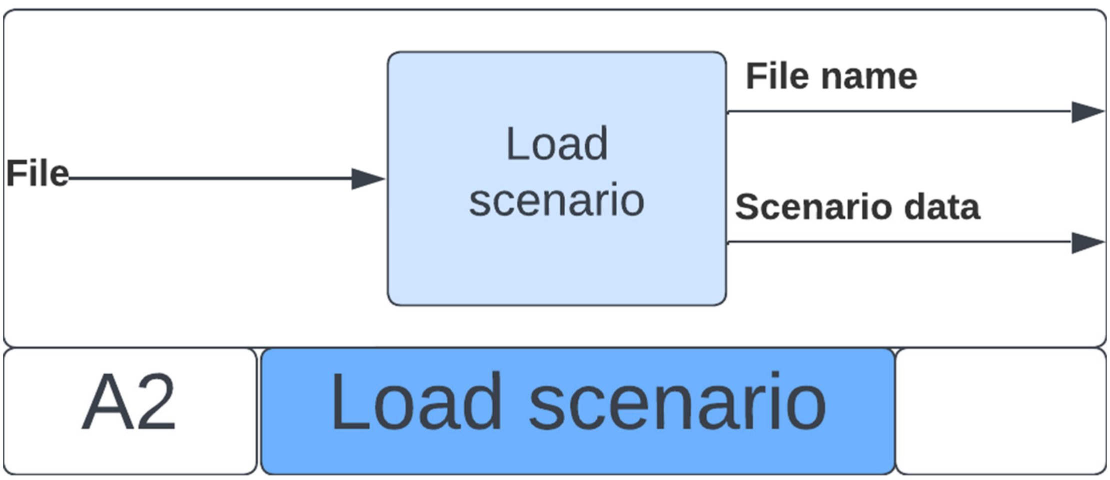

Scenario
Keyframes
Satellite map
Current map
Height data
File path
Generate scenario
A1
Scenario
Map
File


Scenario
Keyframes
Compile scenario
Scenario
Keyframes
Size
Origin
Coordinate converter
Converted coordinates
Routemaker
Routes
Satellite map
Current map
Map manager
Map
Height data
Height manager
Height data
File path
Save scenario
File
A1
Generate scenario


File path
Load scenario
A2
Scenario


File path
Deserializer
File name
Scenario
Load scenario
Scenario
A2
Load scenario


File
Load scenario
A2
File name
Scenario data


File
Load scenario
File name
Scenario data
A2
Load scenario

Appendix J

Risk analysis

Definition of probability		
Degree of probability	Frequency	Interval
1	Very low	Happens very rarely
2	Low	Happens rarely
3	Medium	Happens sometimes
4	High	Happens often

Definition of degree of consequence for project		
Degree of consequence		Outcome
1	Insignificant	Project continues as normal.
2	Small	Project becomes delayed slightly, but minimal effect on end result.
3	Considerabl	Project becomes stagnant, measures required.
4	Serious	Project stops, critical measures required.
5	Disastrous	Project cancelled.

Definition of degree of consequence for product		
Degree of consequence		Outcome
1	Insignificant	Product works as normal.
2	Small	Product stops working.
3	Considerabl	Product stops working, and won't start working again, even with restart
4	Serious	Products works, but not as intended.
5	Disastrous	Product stops working and drones start crashing.

Risk = degree of probability x degree of consequence						
Degree of probability	4	4	8	12	16	20
	3	3	6	9	12	15
	2	2	4	6	8	10
	1	1	2	3	4	5
		1	2	3	4	5
		Degree of consequence				

High risk	Measures to reduce risk has to be implemented.
Medium risk	Measures to reduce risk has to be considered.
Low risk	Measures to reduce risk not required.

Index	Internal/ext	Event	P	C	R	Consequence	Measures to reduce risk
1	Internal	A group member becomes sick	4	1	4	The group member has to catch up with their work or another group member has to take responsibility the tasks.	Actively use anti-bac and hand washing.
2	Internal	A group member becomes sick over a longer period of time	3	3	9	The group has to take responsibility for that the tasks of the sick group member is followed up and taken care of.	Actively use anti-bac and hand washing.
3	Internal	Multiple group members becomes sick	3	2	6	The group members has to catch up or another group member has to take responsibility for the tasks.	Actively use anti-bac and hand washing. The group members should also avoid coming to campus while sick. If they do decide to come in then they need to wear a mask.
4	Internal	Multiple group members becomes sick over a longer period of tin	2	3	6	The remaining group members have to reevaluate the project timeline and see if we can do everything originally planned. Tasks should then be divided between the remaining group members to ensure that everything that has to be done is still covered by someone not sick.	Actively use anti-bac and hand washing. The group members should also avoid coming to campus while sick. If they do decide to come in then they need to wear a mask.
5	Internal	The level of competence in the group is too low	4	3	12	The group member or group as a whole has to learn the relevant subject, or that part of the project has to be dismissed because the level of competence is too low.	Read up on relevant subjects early in the project and try to learn as much as possible before we start producing the product.
6	External	KDA goes bankrupt	1	1	1	The group has to talk to USN about a plan that allows us to finish the product and the bachelor's degree.	Nothing the group can do to reduce this risk.
7	Internal	Project files is lost/deleted	1	4	4	The group has to begin making the files again, but with the expirience of having done it once.	Create backups of everything.
8	External	Pandemic	2	2	4	The group has to switch to working digitally, with the expirience from Covid-19 this switch should be okay. This will affect the group socially which could have an effect on the final product.	Not possible for our group to stop a pandemic.
9	Internal	Disagreement causes group cooperation to falter	2	4	8	Two or more members are unable to cooperate and causes group work and meetings to go awry. The group members that are not in the disagreement needs to fix the dynamics in the group again with help for our internal supervisor.	Tell the other members in the group if something annoys us, and try to deal with disagreements as soon as possible and not let anything build up.
10	External	Group room burns down	1	2	2	We will lose all our physical work and potenially lose computers/laptops. All digital files however are backed up digitally and will still be available. May cause the continuation of the work to happen via Zoom.	No open flames in the group room and keep materials away from heat oven.
11	Internal	Inefficient use of time	4	3	12	Spending too much time on things that do not affect the grade. This use of time could have been used on something else that would be more efficient for a better end result.	Keep time spent on unrelated work to a minimum
12	Internal	Wrong focus (spend too much time on something that doesn't giv	3	3	9	Spending too much time going down a direction that in the end is not useful for our project.	Ensure that all the work we do are directly related to a user story or is necessary for the project documentation.
13	Internal	People are regularly delayed to core time	3	2	6	May cause irritation in group and also make it difficult to start the stand up on time which will delay the other people from starting to work.	Assume that the day starts at 08:30 so you have enough time to get ready and leave in time so that you're always ready before core time starts at 09.00. After a couple of weeks and continued late arrival, we added a rule that whoever is late has to buy cookies for the group to mitigate some of the annoyance of people coming late and also giving a bigger incensitve to be on time.

14	External	External sources causing stress	3	2	6	Personal economics, exams, part time jobs and so on are all sources of stress. That included with the bachelor project itself may cause a person to have a lack of energy. Which in turn makes work hours inefficient and may affect the quality of the work as well.	Talk to the group members if you are feeling stressed and maybe take a day of home office to try to recharge a little.
15	Internal	Burnout	1	3	3	Burnout may cause a lack of motivation and feeling of helplessness and hopelessness. May cause the group member to be away for a time to recover.	Tell the other members in the group and try to reach some sort of agreement on workload and maybe some time to recharge so they can get back earlier than with full burnout.
16	Internal	Group member leaves group	1	4	4	The workload that was for 5 group members have to be divded between 4 group members, and we also have to reevaluate the timeline and maybe reduce the workload in some places so it's possible to finish in time.	The reason for leaving may be many, but keeping internal conflicts to a minimum may help.
17	External	Death in the family or close friends	1	4	4	The group member(s) may need to take some time off to grieve.	Few things we can do to affect this point
18	Internal	Tasks are not finished within deadline	3	3	9	Tasks not being finished within deadline will delay the entire project and give us less time to make advanced modules	Everyone has agreed to get better at asking for help. We've also implemented a status meeting at wednesdays where everyone does a calculated satus update on wether or not they will finish their tasks within the current sprint.
19	External	VM refuses to work correctly, or is too slow to work effectively	3	2	6	VSC in the virtual machine refuses to build and run the program. Making testing our new code more work. It also has a tendency to be very slow on laptops, or flat out refusing to boot.	Unsure what exactly makes VSC stop building and running the software. Unsure what makes it slow, but one group member has used a laptop that has Ubuntu natively, and another group member has added a dual boot so on their laptop so they have both Windows and Ubuntu

Index	Event	P	C	R	Consequence	Measures to reduce risk
101	Routemaker™ fails to take account for something in real life	2	4	8	The drones may crash with physical objects in the real world that the pathfinding failed to take into account when making the route.	Insert costum objects in the software that is taken into account when pathfinding.
102	Program/computer crashes while planning route	3	2	6	May lose unsaved files, and also needs to reboot the program.	Add autosaving so that the potentially lost work will be kept to a minimum.
103	Coordinate converter isn't working properly	1	5	5	Drones will be using the wrong GPS coordinates which may cause crashing into buildings, trees, the ground and even people.	Extensive testing
104	Files saves incorrectly/becomes corrupt	2	2	4	File will be unusable and the user will have to make the scenario again.	Extensive testing
105	Loses internet connection	4	1	4	As long as height data is fully downloaded is will pose no problem. If it is not fully downloaded then first run the algorithm for pathfinding after internet connection is reestablished	Allow user to continue adding key frames and get height data for all points when internet returns to reduce disruptions. Save all height API data to disk along with scenario when scenario is first created.
106	Operator designates a route that is not possible	2	1	2	The drones may crash.	Add a check that checks if the drone will crash into anything trying to fly the designated route. If yes then add a popup that notifies the operator that this route is not possible and makes the operator redo the keyframe.
107	Compability issues with different software integrations	3	2	6	There may be some compability issues between different softwares for the different components. Which causes components to not work together with everything else even though they work individually	Extensive testing.
108	Qt debugging	4	2	8	Debugging Qt can be a challenge as it automatically generates code which Visual Studio Code and CLion debugger does not take into account when inserting breakpoints.	Debug in Qt creator

Index	Software/operational	Event	P	C	R	Consequence	Measures to reduce risk
201	Software	Routemaker algorithm will not be fast enough for live updates	3	5	15	If we can't do live updates to the drones concerning pathfinding, then the product will be unsafe for flying in populated enviornments	Modular architecture
202	Operational	GPS jammer	2	5	10	Someone brings out a GPS jammer to stop the drones being able to position themselves.	GPS jammers are difficult to deal with as they make a lot more noise than the satellite does. However they are not very common. The biggest risk for our project is someone using a GPS jammer in a car for various reasons, can have a range of a few meters to a couple of hundred meters, so reducing the risk on this point would be difficult. The best option would be to make the GPS signal strong enough to that a signal jammer would be ineffective
203	Operational	Helicopter flies over town (especially ambulance helicopter)	2	4	8	If a helicopter flies over Kongsberg as our lightshow is going every drone needs to do an emergency landing.	Talk to Kongsberg kommune about flight permissions. And maybe try to make them disallow flying over the city center while the lightshow is ongoing. However the ambulance helicopter has priority anyways. Establish good communications with the hospital so that we can abort the show long before the helicopter arrives.

Appendix K

Technical contributions

3D visualization	
Responsible member. Aurora Moholth	
Task	Person
Research how to use Rviz as a 3D visualization	Ruben Sørensen, Aurora Moholth
Research how to implement Rviz with librviz in GUI	Ruben Sørensen, Aurora Moholth
Deep-dive librviz	Aurora Moholth
Research how to convert hightdata to Cloudpoint2	Aurora Moholth

Satellite map	
Responsible member. Aurora Moholth	
Task	Person
HTTP request	Ruben Sørensen
Made the HTTP request dynamically retrieve the map information.	Aurora Moholth
Corner coordinates calculation (CCC)	Hilde Marie Moholth
Research how to implementing QGIS	Aurora Moholth
Research and testing related to determining appropriate GIS library	Aurora Moholth
Research ArcGIS	Aurora Moholth
Research how to implementing QGIS in our own GUI	Aurora Moholth
Research how to get right map with QGIS and API from Geonorge	Aurora Moholth

HeightMap	
Responsible member. Hilde Marie Moholth	
Task	Person
Research and testing related to determining appropriate GeoTIFF library	Hilde Marie Moholth
Method to extract height data from GeoTIFF file	Hilde Marie Moholth
Methods to retrieve height data using both UTM33 east, north coordinates and using relative coordinates	Hilde Marie Moholth
Various methods necessary for the dynamic updating of member variables such as the path of the GeoTIFF file	Hilde Marie Moholth
Work and testing related to dynamic update of GeoTIFF file (unfinished)	Hilde Marie Moholth
Continuous testing	Hilde Marie Moholth
Necessary adjustments in methods and variables for integration purposes	Aurora Moholth, Ruben Sørensen

Compile Scenario	
Responsible member. Aurora Moholth	
Task	*Person*
Researching methods for implementing multi drones	Aurora Moholth, Ruben Sørensen
Implementing multi drones and enable sorting by agentID	Aurora Moholth
Dynamically change size and origin	Aurora Moholth

Serialization	
Responsible member. Harald Moholth	
Task	Person
Implementing serilization	Harald Moholth
Macros	Harald Moholth

Testing	
Responsible member. Harald Moholth	
Task	Person
Making rules for how to test	Harald Moholth
Implementing Gtest	Harald Moholth
Making test overview	Harald Moholth & Hilde Marie Moholth
Testing Serializer	Harald Moholth
Make document for documenting tests	Nils Herman Lien Håre
Verify tests against requirements	Aurora Moholth

Requirements	
Responsible member. Harald Moholth	
Task	Person
User stories	Harald Moholth
Use Cases	Harald Moholth
Derived requirements	Harald Moholth
Verified components against requirements	Aurora Moholth, Ruben Sørensen

Coordinate converter

Responsible member. Aurora Moholth	
Task	Person
Converting between geographic and cartesian	Aurora Moholth
Converting between geographic and UTM	Aurora Moholth
Converting between asymmetric and symmetric cartesian	Aurora Moholth
Calculation between asymmetric and symmetric	Aurora Moholth, Ruben Sørensen
Continuous testing	Aurora Moholth
Research libraries for coordinate systems	Ruben Sørensen
Deep-dive GeograpicLib	Aurora Moholth

Architecture	
Responsible member. Aurora Moholth	
Task	Person
Brainstorming architectures	Everyone
Work on initial drafting of architectures	Aurora Moholth, Hilde Marie Moholth
Verify the design architecture against the code	Aurora Moholth, Ruben Sørensen
Continuous updated the architectures in line with the implementation of the system	Aurora Moholth
Finalizing the architecture	Aurora Moholth
Differentiate between logical and design architecture	Aurora Moholth
Developed use case diagram	Aurora Moholth

Routemaker	
Responsible member: Ruben Sørensen	
Task	Person
Implement abstract graph interface	Ruben Sørensen
Implement A* path-finding algorithm	Ruben Sørensen
Implement Bresenham's line algorithm	Ruben Sørensen
Implement post-smoothing of paths	Ruben Sørensen
Make Routemaker resolution adjustable	Ruben Sørensen
Dynamically change Routemaker size, origin and resolution	Aurora Moholth

Wiki Page	
Responsible member: Hilde Marie Moholth	
Task	Person
Defining Wiki structure and contents	Hilde Marie Moholth
Updating Wiki	Hilde Marie Moholth, Nils Herman Lien Håre

Literature Review	
Responsible member: Hilde Marie Moholth	
Task	Person
Read and summarize articles in separate notes scheme	Hilde Marie Moholth
Synthesize academic papers into literature review	Hilde Marie Moholth

Project Plan	
Responsible member: Hilde Marie Moholth	
Task	Person
Planned and drew up proposal for project timeline	Hilde Marie Moholth
Planned and drew up proposal for project sprint calendar	Hilde Marie Moholth
Proposal for detailed plan of final 8 sprints	Aurora Moholth, Hilde Marie Moholth, Ruben Sørensen

Proofreading	
Responsible member: Hilde Marie Moholth	
Task	Person
Overall read-through and editing for fluency and ease of reading	Hilde Marie Moholth
Proofreading each section	Every member
Appendices	Harald Moholth
Glossary and acronyms	Aurora Moholth
Tables and figures	Nils Herman Lien Håre
General appearance of final report and resources	Ruben Sørensen

Keyframe manager	
Responsible member: Nils Herman Lien Håre	
Task	Person

Singleton	Nils Herman Lien Håre
Keyframe manager header-file	Ruben Sørensen, Nils Herman Lien Håre
Handle keyframe with same agentID and Timestamp	Aurora Moholth

Website	
Responsible member: Nils Herman Lien Håre	
Task	Person
Design	Nils Herman Lien Håre
Convert from HTML to PHP, and separate files	Ruben Sørensen
Database queries	Ruben Sørensen
Pipeline for automatic updates when merging into main	Ruben Sørensen
Database	Nils Herman Lien Håre
Dynamic modals	Nils Herman Lien Håre

Risk analysis	
Responsible member: Nils Herman Lien Håre	
Task	Person
Create risk matrix	Nils Herman Lien Håre
Risk evaluation	Everyone

Interfaces	
Responsible member: Nils Herman Lien Håre	
Task	Person

Create IDEF0-diagrams	Nils Herman Lien Håre
Initial interfaces draft	Ruben Sørensen, Nils Herman Lien Håre, Harald Moholth
Define updated interfaces	Ruben Sørensen, Aurora Moholth
Verify updated interfaces against architecture	Ruben Sørensen, Aurora Moholth
Verify updated interfaces against code	Ruben Sørensen, Aurora Moholth

Integration	
Responsible member: Ruben Sørensen	
Task	*Person*
Define coding standard	Ruben Sørensen
Create Azure Pipelines for building and publishing code documentation online	Ruben Sørensen
Integrate MVP components	Ruben Sørensen
Verify code against components	Ruben Sørensen, Aurora Moholth
Verify code against architecture	Ruben Sørensen, Aurora Moholth
Integrate advanced components	Aurora Moholth, Ruben Sørensen
General types header-file	Ruben Sørensen, Aurora Moholth

MVP definition	
Responsible member: Everyone	
Task	Person
Develop list of required functionality for MVP	Hilde Marie Moholth
Determining requirements list for MVP	Everyone

Version control	
Responsible member. Ruben Sørensen	
Task	*Person*
Define branch rules	Ruben Sørensen
Handle merging and conflicts	Ruben Sørensen
Temporary responsibility for merging and conflicts when Ruben was ill	Aurora Moholth

GUI	
Responsible members: Ruben Sørensen, Aurora Moholth, Nils Herman Lien Håre	
Task	*Person*
Qt6 research	Ruben Sørensen
GUI Layout	Ruben Sørensen
Code structure for GUI	Ruben Sørensen
Timeline widget - Visualization and interactivity	Nils Herman Lien Håre
Initial keyframe adding and removal dialog boxes	Nils Herman Lien Håre
Planning view - Map display	Aurora Moholth
Set Location dialog box	Aurora Moholth
Planning view - Routes visualized	Aurora Moholth, Ruben Sørensen
Planning view - Mouse picking, click map to add keyframes	Ruben Sørensen

Documentation

Responsible member: Hilde Marie Moholth	
Task	*Person*
Content structure of documentation	Hilde Marie Moholth
Create LaTeX project structure for documentation	Ruben Sørensen
Set up custom LaTeX commands for displaying the authors and collaborators of sections, subsections and subsubsections	Ruben Sørensen
Create various Tikz figures	Ruben Sørensen
Software user guide	Aurora Moholth
Development environment setup guide	Ruben Sørensen

Appendix L

Updated testing documentation

Index	T.1.1
Approved by	Nils Herman
Done by	Harald (approved)
Method	Demonstration
Prerequisites	Have every necessary library installed
Data	None
Description	Launch software, verify that GUI is visible and can be interacted with.
Steps	Run the hivemind executable to see if the hivemind window appears.
Success criteria	The hivemind window appears and can be interacted with.

Index	T.2.1
Approved by	Aurora Moholth
Done by	Nils Herman (Approved)
Method	Demonstration
Prerequisites	Launch Hivemind
Data	None
Description	Press the planner button, and verify that the map for the given area appears on screen.
Steps	Press the “Planner” tab on the top left.
Success criteria	The map appears under the planner tab.

Index	T.3.2
Approved by	Ruben Sørensen
Done by	Nils Herman (approved)
Method	Demonstration

Prerequisites	Set location to “61.636740010738535, 8.312417773829353”
Data	AgentId: 1, TimeStamp: 0, X: -11.7224, Y: 40.0418, Z: 0 AgentId: 1, TimeStamp: 6, X:41.6667, Y: -54.4336, Z: 0
Description	Test selecting a coordinate with known height data to verify the returned data is correct.
Steps	Add the two keyframes, and check if the routemaker avoids the church. If successful then it takes church into account when planning routes which means the height data must be correct.
Success criteria	Planned route avoids the church

Index	T.3.2
Approved by	Ruben Sørensen
Done by	Hilde Marie (approved)
Method	Inspection
Prerequisites	None
Data	UTM33N coordinate: (6626362, 198592)
Description	Test selecting a coordinate with known height data to verify the returned data is correct.
Steps	Query HeightMap for height at specified UTM coordinate. Verify with an external source that height data is correct.
Success criteria	Queried height data matches external data

Index	T.3.4
Approved by	
Done by	Nils Herman (failed)
Method	Demonstration
Prerequisites	Set location to “1, 2”
Data	None

Description	Test selecting coordinate with known data NULL, verify the software returns an error message and no further action is taken.
Steps	Set the location data.
Success criteria	If an error shows up when you insert the coordinates, and no further action is taken.
If failed, why?	An error message comes up in the terminal, and the program crashes.

Index	T.6.1
Approved by	Harald
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Launch Hivemind
Data	None
Description	Verify that clicking the “Save” button will launch the save dialogue box
Steps	Press “File” and “Save as”. See if you get the directory for choosing location and saving file.
Success criteria	The dialogue for the directory appears as expected.

Index	T.6.3
Approved by	Ruben Sørensen
Done by	Nils Herman (Approved)
Method	Demonstration
Prerequisites	A saved file
Data	AgentId: 0, TimeStamp: 0, X: -83.9833, Y: 126.602, Z: 0
Description	Load file and verify that it is identical to the previously saved plan
Steps	Open the file in notepad or similar. Check that the data in the file is correct.
Success criteria	The data in the file is correct.

Index	T.6.4
Approved by	Ruben Sørensen
Done by	Nils Herman (Approved)
Method	Demonstration
Prerequisites	Launch Hivemind and add a keyframe for the serializer to save.
Data	AgentId: 0, TimeStamp: 0, X: -83.9833, Y: 126.602, Z: 0
Description	Verify that the user can successfully save a file in a custom location.
Steps	Press “save as”, find a location in the directory, press “save” and verify that a file is created
Success criteria	A file is created with the name and location specified by the user

Index	T.6.6
Approved by	Harald
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Launch Hivemind
Data	None
Description	When “Load Scenario” is pressed, verify that the explorer window appears.
Steps	Press “File” and “Open…” and check if directory shows up
Success criteria	The directory dialogue for opening a saved file shows up

Index	T.6.7
Approved by	Ruben Sørensen
Done by	Nils Herman (Approved)
Method	Demonstration

Prerequisites	Launch Hivemind and have a test file prepared for loading.
Data	Test file containing the following keyframes: AgentId: 0, TimeStamp: 0, X: -853.621, Y: -111.56, Z: 0 AgentId: 0, TimeStamp: 6.65709, X: -226.88, Y: -264.485, Z: 0
Description	Verify that the correct scenario opens when selected.
Steps	Press “File”, “Open…” and choose the file you wish to open. Then click “Open” and check if the data is correct.
Success criteria	A file is loaded, and the relevant areas are populated with data.

Index	T.6.9
Approved by	Harald
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Launch Hivemind
Data	None
Description	Set location and verify that correct location and data is loaded.
Steps	Launches Hivemind. Press “Set location” and then verify that the box appears as it’s supposed to.
Success criteria	The dialogue box appears as expected.

Index	T.6.10 (tested on older version)
Approved by	Harald
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Opened the “Add Keyframe” dialogue and entered exampledata
Data	agentId: 1, timestamp: 1, x coordinate: 1, y coordinate: 1, z coordinate: 1

Description	Verify that the keyframe is added to the graphical user interface when enter is pressed in the keyframe dialogue box, and that timestamp and location is correct.
Steps	Have the dialogue box for keyframes open, enter the example test and press enter. Verify that the keyframe is added to the vector that stores keyframes.
Success criteria	The keyframe is stored in the keyframe vector

Index	T.6.12
Approved by	Aurora Moholth
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Keyframes stored in the vector
Data	AgentId: 0, TimeStamp: 0, X: -387.326, Y: -121.588, Z: 0 AgentId: 0, TimeStamp: 6.03448, X: -266.992, Y: -176.741, Z: 0
Description	Verify that added keyframe is correctly assigned on the timeline
Steps	Add two keyframes and see if they show up on the timeline and if they show up in the right order with the right coordinate.
Success criteria	The keyframes show up on the timeline in the right order and the right coordinate.

Index	T.6.13
Approved by	Ruben Sørensen
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Keyframes added to the timeline
Data	AgentId: 0, TimeStamp: 0, X: -387.326, Y: -121.588, Z: 0 AgentId: 0, TimeStamp: 6.03448, X: -266.992, Y: -176.741, Z: 0 AgentId: 0, TimeStamp: 3.35249, X: -377.298, Y: -251.95, Z: 0

Description	Add a coordinate, and verify that the distribution of keyframes on timeline dynamically updates.
Steps	Have the two first keyframes added. Add the third keyframe and see if the timeline updates with the third keyframe in the middle of the first two
Success criteria	The third keyframe shows up on the timeline between the first and second keyframe

Index	T.6.15
Approved by	Nils Herman
Done by	Aurora (approved)
Method	Demonstration
Prerequisites	Launch Hivemind, and set location to “59.665819782515435, 9.646190995911908”
Data	AgentId: 0, TimeStamp: 0, X: -40.9703, Y: 29.364, Z: 0 AgentId: 0, TimeStamp: 9.48276, X: 97.1449, Y: -24.7215, Z: 0 AgentId: 0, TimeStamp: 15.8525, X: 109.912, Y: 52.8087, Z: 0
Description	Press compile scenario, and verify that a route is created, and that information in route is correct.
Steps	Add keyframes to the scenario and compile. Verify visually that the routes are correct.
Success criteria	The routes have to avoid buildings and other objects.

Index	T.8.1
Approved by	Ruben Sørensen
Done by	Aurora (approved)
Method	Demonstration
Prerequisites	Launch coordinate converter
Data	Geographical coordinate from google maps: 59.66465552506008, 9.645717559340614

	UTM coordinate returned from Hivemind: NORTH 6626236.65 EAST 198547.51
Description	Enter a coordinate and check that you get expected data in return.
Steps	Use google maps and find a geographical coordinate. Used the coordinate converter to convert from geographic to UTM coordinate. Check at norgesdata.no that the UTM coordinate is the same place as in google maps. Use the UTM coordinate and convert back to geographical coordinate.
Success criteria	The coordinate that we convert and convert back again is the same and corresponds to the data form norgeskart and google maps.

Index	T.11.2
Approved by	Nils Herman
Done by	Aurora (approved)
Method	Demonstration
Prerequisites	Launch Hivemind and add keyframes with different agent IDs
Data	agentId: 1, timestamp: 1, x coordinate:-40, y coordinate: 0, z coordinate: 1 agentId: 1, timestamp: 3, x coordinate: 0, y coordinate: 20, z coordinate: 3 agentId: 2, timestamp: 1, x coordinate: -45, y coordinate: 40, z coordinate: 2 agentId: 2, timestamp: 3, x coordinate: 22, y coordinate: 50, z coordinate: 4
Description	Verify that each generated route is separated by an agent ID.
Steps	Make sure the drones generate different paths between each keyframe and that they are not connected to each other.
Success criteria	The routemaker creates different paths for each unique agent ID

Index	T.11.3 (tested on older version)
Approved by	Aurora

Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Launch Hivemind and add two keyframes with different agentId
Data	agentId: 1, timestamp: 1, x coordinate: 1, y coordinate: 1, z coordinate: 1 agentId: 1, timestamp: 2, x coordinate: 9, y coordinate: 3, z coordinate: 1 agentId: 2, timestamp: 1, x coordinate: 1, y coordinate: 1, z coordinate: 1 agentId: 2, timestamp: 3, x coordinate: 3, y coordinate: 3, z coordinate: 1
Description	Test that adding new agent through interface causes the new agent to correctly appear
Steps	Check if the keyframes are stored with different agent IDs
Success criteria	The keyframes are stored with different agent IDs

Index	T.11.8
Approved by	Aurora
Done by	Nils Herman (approved)
Method	Demonstration
Prerequisites	Launch Hivemind and set location to “59.66579762399427, 9.646237795427599”
Data	AgentId: 0, TimeStamp: 0, X: 102.02, Y: 19.6147, Z: 0 AgentId: 0, TimeStamp: 5.17241, X: 97.6091, Y: -62.558, Z: 0 AgentId: 1, TimeStamp: 5.17241, X: -38.1848, Y: 43.0594, Z: 0 AgentId: 1, TimeStamp: 14.5594, X: -44.6843, Y: -11.4902, Z: 0
Description	Compile scenario and ensure all agent routes are included.
Steps	Add the 4 keyframes and press “compile scenario”
Success criteria	Verify that the routes are created and independent of each other.

Appendix M

Project timeline

Project Timeline

March 15, 2023

Phase	Sprints	Dates	Goals
Planning	Pre sprint	24.01 - 31.01	*Second draft of requirements table, including design choices* *Formulate project plan* *Finalise code standard*
	Sprint 1	31.01 - 07.02	*Complete PowerPoint presentation* *Have completed requirements table, including design choices, and project plan* ***First presentation (7/2)***
Preparation	Sprint 2 Sprint 3 Sprint 4	07.02 - 28.02	*Design choices locked in* *Finish architecture*
	Sprint 5 Sprint 6 Sprint 7 Sprint 8	28.02 - 29.03	*Complete risk analysis of project* *Have started programming some modules of minimum viable product (testing continuously)* ***Second presentation (28/3)***
Coding	Sprint 9 Sprint 10 Sprint 11	31.03 - 24.04	*Minimum viable product assembled and tested*
	Sprint 12 Sprint 13	24.04 - 08.05	*Add more advanced modules* *Complete risk analysis for product*
Report	Sprint 14	08.05 - 23.05	*Write report* *Edit report, ensure proper citaiton and formatting* *Prepare EXPO booth* ***EXPO (23/05)***
Present	Sprint 15	23.05 - 25.05	*Prepare third presentation* ***Third presentation (30/05)***

Appendix N

Seating arrangements

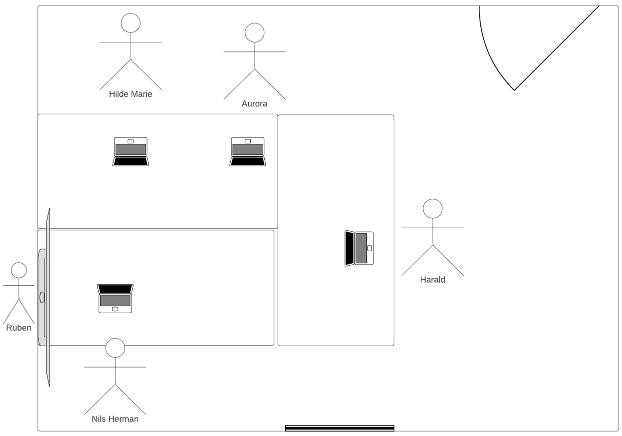
1 Seating arrangement

NH | *AM*

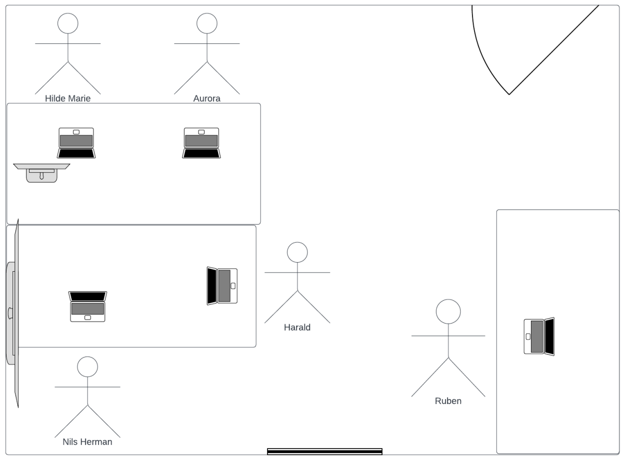
Initially the group's designated work space contained three desks; two were positioned with their short sides against the wall, while the third was centrally located, abutting the other two. This layout (fig. N.1a) was maintained during the initial weeks, primarily due to Ruben's temporary absence while participating in exchange studies in Belgium. His participation in meetings was facilitated via Zoom on the TV screen positioned in the bottom left corner of the room. However, the central table's instability made the group relocate it to the bottom right corner, where it remained unused.

Upon Ruben's return, he utilized the unstable table briefly (fig. N.1b). However, the decision was made to replace this table with Ruben's personal desk. With USN's agreement, the group removed the unstable table and introduced Ruben's desk. This also triggered the group to reconsider the configuration of furniture in the room. The two previously conjoined tables were separated, repositioning their occupants to face the wall, fostering enhanced openness and easier communication among group members. We also introduced second-hand furniture, including a cabinet for the storage of books, various teas, and documentation, along with an armchair offering seating for our supervisors and a relaxation space for the group and any visitors.

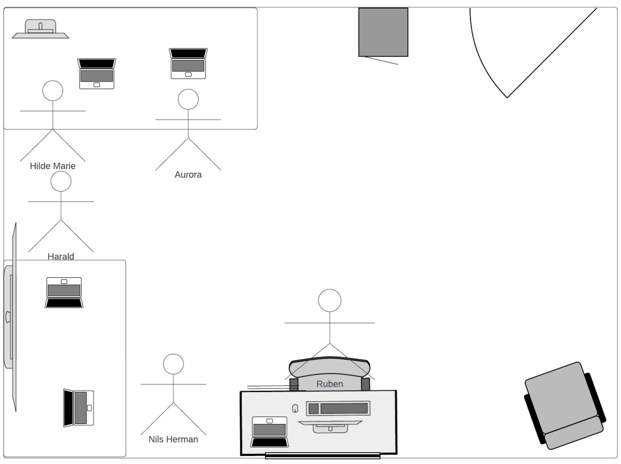
After making these improvements, the rooms ventilation became an issue. After occupying the room in its current state (fig. N.1c) for several weeks, the group decided to reposition the tables to make access to the window for ventilation purposes easier. The group aligned all tables against one wall, relocating Ruben's desk to the opposite side. An additional tea table and an armchair were introduced to accommodate the group's two supervisors. This layout (fig. N.1d) proved effective, cultivating an environment that encouraged easy communication, provided ample movement space, and included a relaxation corner for the group when necessary, and seating for supervisors during meetings.



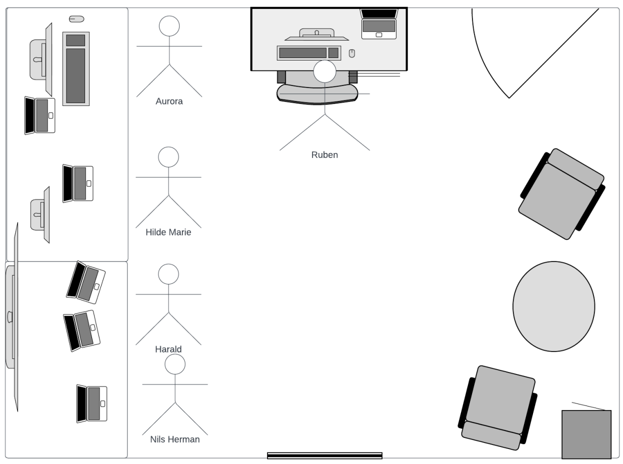
(a) Inital workspace configuration



(b) Workspace layout upon Ruben's return



(c) Workspace layout after introduction of personal desk



(d) Final workspace layout

Figure N.1: Seating arrangements

Appendix O

Code documentation

hivemind

1.0.0

Generated by Doxygen 1.9.6

Chapter 1

Hivemind

1.1 About

Hivemind is a route-planning software for drone swarms. It is currently in the early stages of development. It currently serves as the final project of the developer team's bachelor's degrees.

1.2 Where to start?

- Take a look at Get started to get the development environment up and running.
- For a guide to using the software, take a look at the user guide.
- For an insight into the testing methods Hivemind utilizes, head over to Testing standard.
- For developers, please familiarize yourself with Hivemind's coding standards.

Chapter 2

Coding Standards

2.1 Introduction

This document serves as both a guide to the developers and maintaners of Hivemind, and an explanation of code design and architecture choices for other actors who needs to or wants to look at the source code of Hivemind.

2.2 Semantics and coding style issues

2.2.1 Treat compiler warnings as errors

Compiler warnings are useful hints to improve code. Generally, if the compiler issues a warning, adjust the code to suppress this.

2.2.2 Object oriented programming

Generally, developers of Hivemind should adhere to an object oriented programming (OOP) style in Hivemind's codebase, especially when implementing top-level interfaces of major components. It is important to note, however, that it is encouraged avoid OOP when moving into deeper implementation details. Being too strict on an object oriented approach often leads to unnecessary complications.

2.2.3 Assert extensively

The use of assertions in code is encouraged. Assertions are not only very useful for verifying states and data, they also document expected behaviour for other developers looking at the codebase.

2.3 Source Code Formatting

2.3.1 Clang Format

The root of the project contains a `.clang-format` configuration file. This should be used to format all source files to maintain consistent formatting throughout the codebase, and to prevent unnecessary changes to untouched code cluttering the version control history.

The control comments `clang-format off` and `clang-format on` *can* be used for specific code blocks where retaining a specific format is preferable, but this should be used sparingly.

2.3.2 Commenting

Comments are useful for documenting source code and provides improved readability and maintainability. Comments should provide an explanation of the code's purpose rather than an explanation of *how* it is done; the code documents the process itself.

2.3.2.1 Class definitions

Proper documentation of class definitions are expected. The purpose of a class and how it works should be explained with Doxygen comments to keep the docs as up to date as possible. These comments should be located in the header file of the class.

2.3.2.2 Comment formatting

Generally, prefer C++-style comments rather than C-style. For normal comments, this means using `//`, and using `///` for doxygen comments.

2.3.2.3 Doxygen comments

Prefer using *triple-slash* (`///`) comments for doxygen documentation. Prefer using *backslash* (`\`) over *at* (`@`) for doxygen tags such as `param`, `file` and `returns`.

Prefer:

```
///
/// \brief Function used to create Bar
///
/// \param id Unique ID that represents Bar
/// \returns Bar object
///
Bar Foo(int id);
```

Avoid:

```
/**
 *  @brief Function used to create Bar
 *
 *  @param id Unique ID that represents Bar
 *  @returns Bar object
 */
Bar Foo(int id);
```

2.3.3 White space

Prefer spaces over tabs. There are valid arguments for both the use of spaces and tabs, but a mixture of both of them should not be used. Therefore, a standard of using spaces is preferred in this project. Clang Format **should** ensure that tabs are converted to spaces.

2.3.4 Column width

If a maximum column width is going to be defined, using a standard width makes sense. Therefore, a maximum column width of 80 has been set. This **should** be enforced by Clang Format. There are exceptions to this rule, and these are generally related to comments or strings that have a specific format that makes more sense than the one enforced by Clang Format. In these cases, the use of `clang-format off` and `clang-format on` are allowed.

2.4 Language specifics

For now, all the source code of Hivemind is written in C++. When we start to use ROS as part of the system, there are plans to experiment with the feasibility of implementing modules in Python.

2.4.1 C++

2.4.1.1 Standard version

Hivemind uses C++17.

Although C++20 is both feature-complete and mostly supported by the major compilers, our preferred build toolchain, CMake, only supports some features through the use of experimental flags. As such, we currently view C++20 as not fully supported and not a viable option. This may change in the future.

2.4.1.2 Standard library

Generally prefer to use the data structures, algorithms and functions available in the C++ standard library rather than implementing custom solutions. The standard library is mature, robust, extensively tested and highly optimized.

2.4.1.3 Naming convention

Maintaining a uniform naming convention throughout the codebase helps increase readability. As such, we use a well-defined naming convention that must be adhered to when writing C++ code.

- Namespaces, classes, structs and enums should all be named using `PascalCase`.
- Macros and enum values should be named using `SCREAMING_SNAKE_CASE`.
- Local variables and functions outside classes/structs should be named using `camelCase`.
- Members and attributes of classes/structs should be named using `PascalCase`, but private attributes should be pre-fixed with `m_`.

2.4.1.4 Classes and structs

In C++, classes and structs are essentially the same thing and they can generally be used interchangeably, given that you take access specifiers into account.

We define a semantic difference in our codebase: Classes are to be used for more complex data objects with attributes and members of both private and public access. Structs are to be used for more simple data objects where all attributes are public.

When defining classes, the attributes and members of different access specifiers should be defined in the following order:

1. public
2. protected
3. private

The rationale for this is that if someone looks at the header file of a class to see what attributes and members they can access, they will not care about private members and implementation details. They want to know which members and attributes they can actually use.

2.4.1.5 Include style

At the top of the file, below the header guard in the case of header files, should the includes required by the file be listed. They should be ordered as follows:

1. Main module header
2. Project headers
3. Library headers
4. System headers

The *main module header* only applies to .cpp files with a header files whose classes and functions it implements. The *project headers* refer to other header files part of the Hivemind project that the file depends on. The *library headers* refer to dependant header files from external libraries such as *QT* headers. Finally, *system headers* generally refer to headers that are part of the C standard library and C++ standard library.

The main module header and project files should be include with the **double-quote** style, and library files and system files should be included with the **angled brackets** style.

Header files should only include other header files that it **strictly** needs. If the include can be moved to the corresponding .cpp file instead, it should.

Example:
```
// foo.cpp

#include "foo.h"

#include "hivemind_core.h"
#include "hivemind_gui.h"

#include <QWidget>
#include <QMath>

#include <vector>
#include <map>
...
```

2.4.1.6 Header Guards

Header files should be protected using `#pragma once` rather than traditional header guards. `#pragma once` is technically not standard but it is widely supported and provides several advantages including less code, less risk of name clashing and potentially improved compilation speed.

Prefer:
```
// foo.h

#pragma once

class Foo
{};
```

Avoid:
```
// foo.h

#ifndef FOO_H
#define FOO_H

class Foo
{};

#endif // FOO_H
```

2.4.1.7 Use of the auto keyword

The use of the `auto` keyword should be reserved for cases where the type can be deduced from the context. An example of this is when casting a variable to another type. The cast operation will specify the type, so it is easily deduces.

Example:
```
// It is obvious the resulting type will be Foo
auto foo = static_cast<Foo>(bar);
```

The `auto` keyword can sometimes also be used to increase readability of the codebase. Examples of this is when using the chrono library in the std namespace. It is extremely verbose, and using auto can help with readability.

Example:
```
// The following assignments are equivelant, but one is arguibly more readable.

std::chrono::time_point<std::chrono::steady_clock> start = std::chrono::steady_clock::now();

auto start = std::chrono::steady_clock::now();
```

2.4.1.8 RAII

`RAII`, or *Resource Acquisition Is Initialization*, is a C++ programming technique which ensures that the life-cycle of a limited resource, such as heap memory or a locked mutex, is bound to the life-cycle of an object, meaning that the resource is accessible and usable as long as the object lives, and that it is automatically freed when the object is destroyed.

RAII is generally implemented by acquiring the needed resource in the constructor of a class, and freed in the destructor.

Prefer to use RAII where applicable.

Chapter 3

Get Started

3.1 Install dependencies

Hivemind has several dependencies. The following installation methods have been tested for Ubuntu 22.04. You may attempt other installation methods as well, but these are veryfied to be working.

3.1.1 Main dependencies

The main dependencies for building Hivemind are listed here.

Click each section to expand.

Make sure system is up-to-date

```
$ sudo apt-get -q update
```

Install build tools

```
$ sudo apt-get install -y cmake ninja-build make g++ rpm build-essential libgl1-mesa-dev
```

Install Qt6

```
$ sudo apt-get install -y qt6-base-dev
```

Install proj development package

```
$ sudo apt-get install -y libproj-dev
```

Install GeographicLib

```
$ wget -qO-
"https://downloads.sourceforge.net/project/geographiclib/distrib-C%2B%2B/GeographicLib-2.2.tar.gz?ts=gAAAAABkPnvtCqJ9K7pUSa
| tar xvz
$ mkdir GeographicLib-2.2/build/ && cd GeographicLib-2.2/build/
$ cmake ..
$ make -j`nproc`
$ sudo make install
```

Install GDAL

```
$ git clone https://github.com/OSGeo/GDAL.git
$ mkdir GDAL/build/ && cd GDAL/build/
$ cmake ..
$ cmake --build .
$ sudo cmake --build . --target install
```

Install RapidJSON

```
$ git clone --recursive https://github.com/Tencent/rapidjson/
$ mkdir rapidjson/build/ && cd rapidjson/build/
$ cmake ..
$ make -j`nproc`
$ sudo make install
```

3.1.1.1 Want to build the docs?

Hivemind's docs requires Doxygen 1.9.6. The following sections shows how to install Doxygen's dependencies and Doxygen itself.

Click each section to expand

Install dependencies

```
$ sudo apt-get install -y git graphviz wget
```

Install Doxygen

```
$ wget https://github.com/doxygen/doxygen/releases/download/Release_1_9_6/doxygen-1.9.6.linux.bin.tar.gz
$ tar -xvf doxygen-1.9.6.linux.bin.tar.gz
$ cd doxygen-1.9.6/
$ sudo make install
```

3.2 Build

3.2.1 Building Hivemind

Once all dependencies are installed, building Hivemind is simple.

From the project's root directory:

```
$ mkdir -p build/ && cd build/
$ cmake ..
$ make -j`nproc`
```

After building, Hivemind can be launched:

```
$ ./hivemind
```

3.2.2 Building the docs

If you want to build the docs, make sure you have installed Doxygen as shown above.

From the project's root directory:

```
$ mkdir -p build/ && cd build/
$ cmake ..
$ make docs
$ firefox docs/index.html # Replace firefox with your browser of choice
```

Chapter 4

Testing Standard

4.1 Introduction

To be able to develop a functioning software we have to have a standard for testing so that everyone agrees on when a component is done being developed.

This document will go throught the different ways of how you should go about verifying components for Hivemind and what is expected to be included in the verification document.

4.2 Unit testing

When testing Hivemind, the principle of unit testing should the followed. This means that when creating a test for a component that test should be independent of other tests or components. A major benefit of following unit testing is that it simplifies automation of test through use of azure pipelines and GoogleTest. This does not mean every test that follows unit testing needs to be automated since it is very hard to automate test for a Graphical User Interface (GUI), but you still want to follow the principles of unit testing when testing a GUI.

4.3 Methods of Verification

There are 4 methods that can be used to verify components for Hivemind. They are:

4.3.1 Inspection

Inspection is examining the system and verifying that functionality is present. In a software system inspection can be performed by looking at the code and veryfying that the software has the necessary inputs and function that are required for the system to work.

4.3.2 Demonstration

Demonstration is verifying the system through manipulation. This is done by verifying that the expected result are acquired when the system is used as intended. In software, demonstration can be done by clicking on a button and checking if the system responds according to expectation.

4.3.3 Testing

Testing is verifying that the system operates as intended through using a predefined set of data and inputs, as well as knowing the expected output from the system when using those data and inputs. This type of verification is possible to automate.

4.3.4 Analysis

Analysis is the final method used to verify a system. This is done by creating models of the system, using equipment to test parts of the system if possible or calculations, if there is a complex function or algorithm in the system.

4.4 Documentation of verification

To document the verification process, a table that contains all the necessary information should be used. It should include:

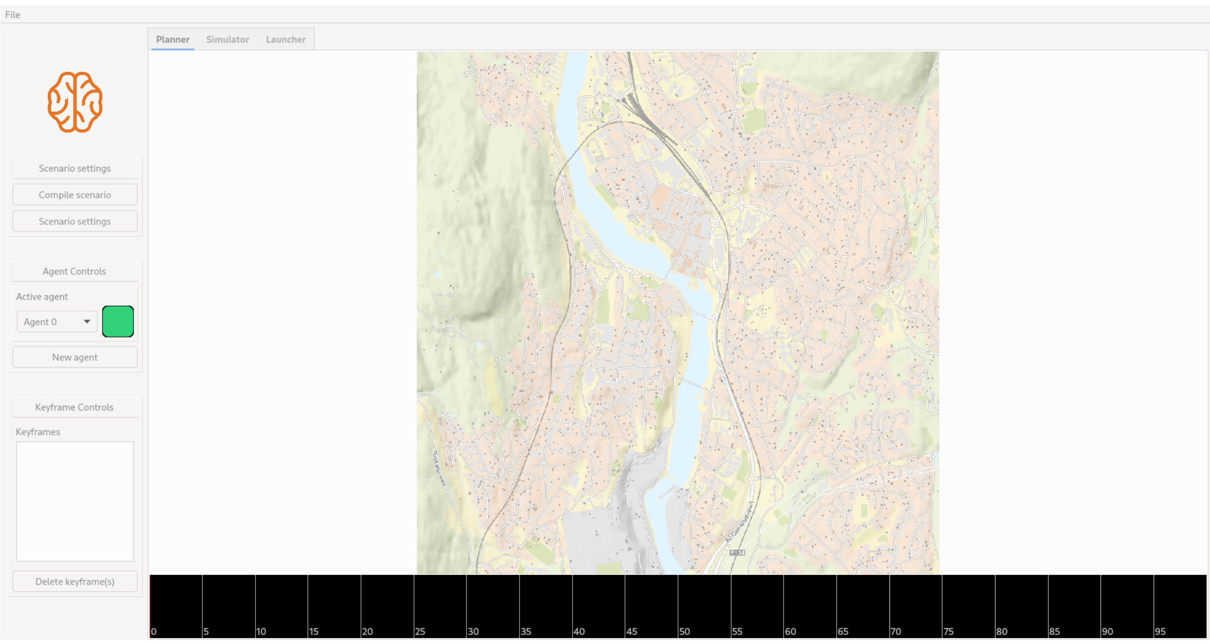
1. An index to identify which test is being done
2. Who approved test
3. Who did the test
4. Which methods were used to perform the test
5. What prerequisites has to be in place to be able to recreate the test
6. What data was used in the test
7. A description of the test
8. The success criteria for the test
9. If the test failed, a description of the error should be provided

Chapter 5

User Guide

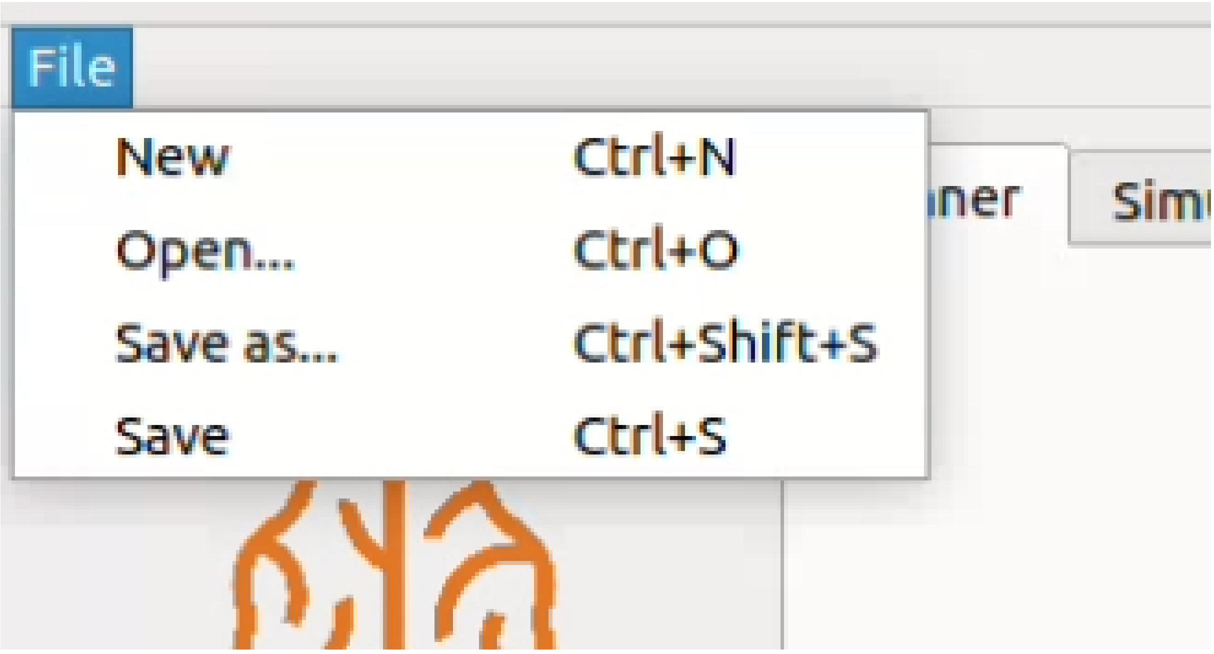
5.1 Graphical User Interface

The software has an intuitive user interface that makes it easy to navigate and perform tasks. Here is a description of the key elements in the user interface:



5.1.1 Menu bar

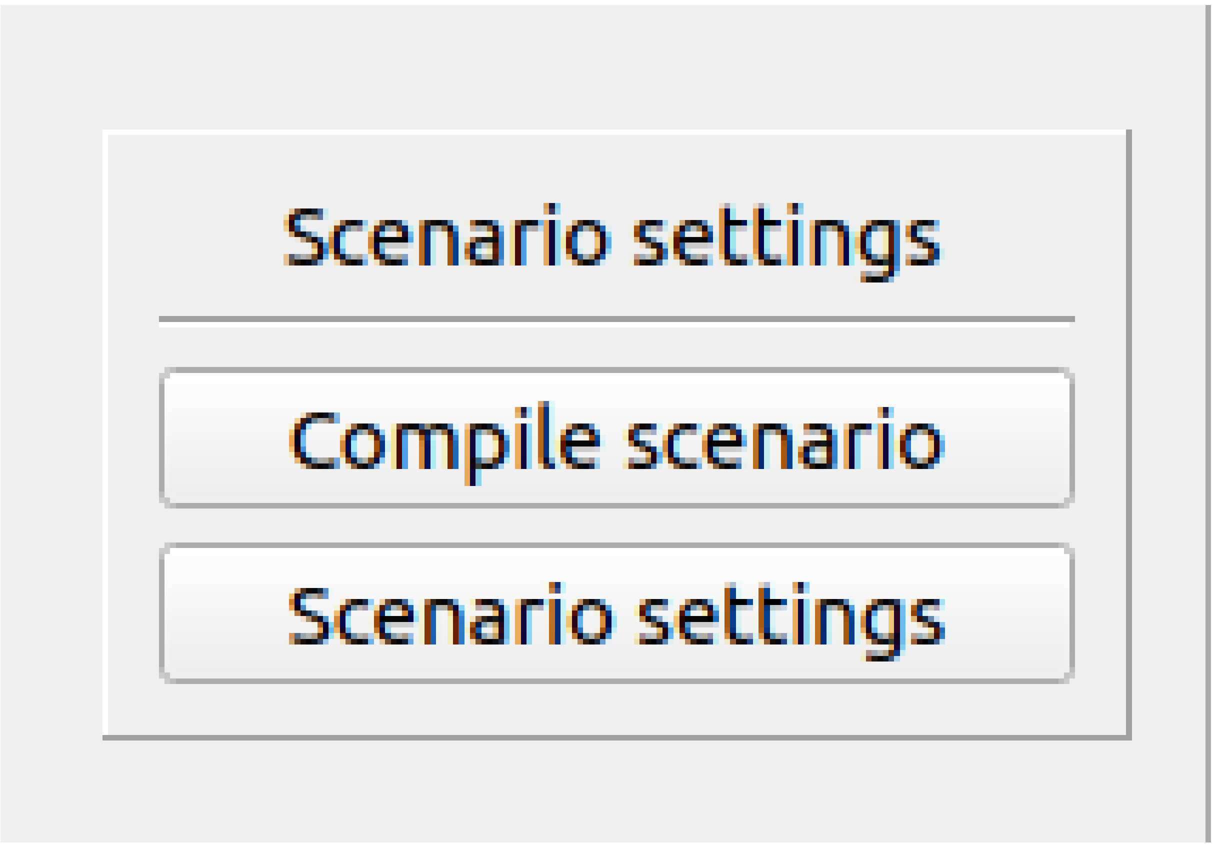
At the top of the window, you will find the menu bar. It includes a dropdown menu that allows you to manage scenarios by loading and saving them.



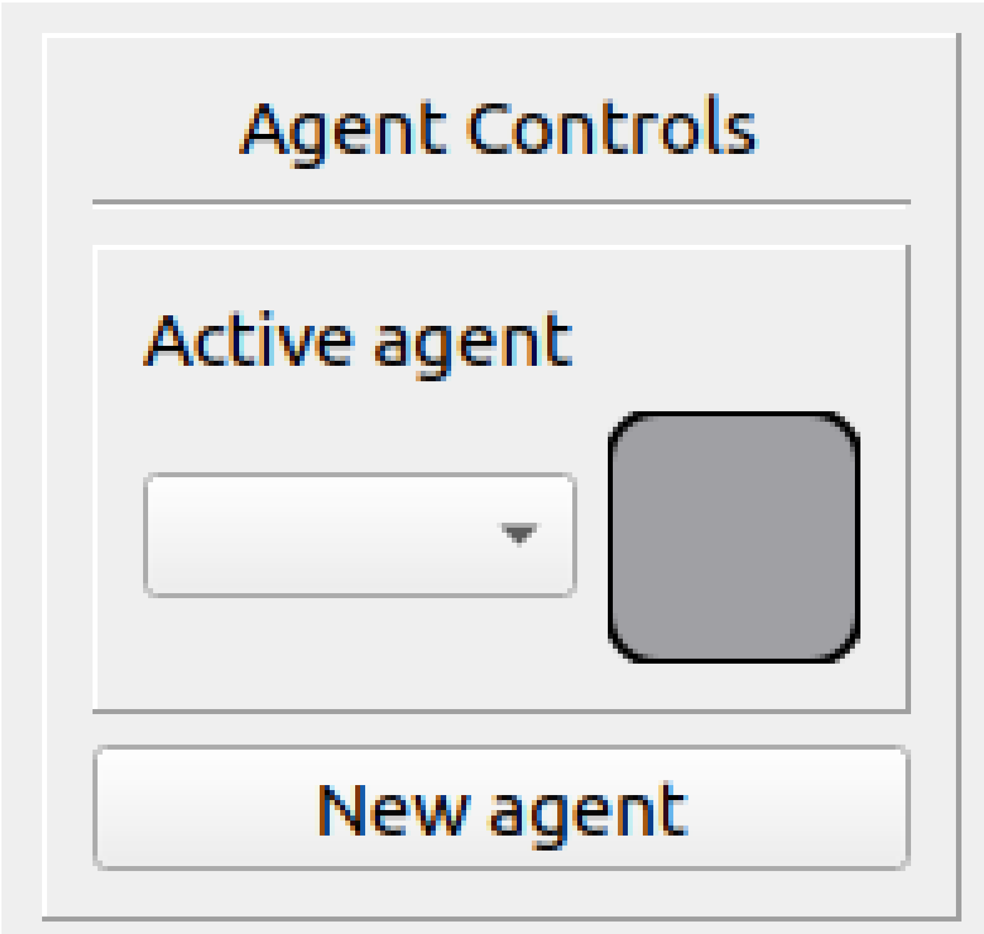
5.1.2 Sidebar

The sidebar is located on the left side of the window and provides quick access to different tools in the software. It contains the following sections.

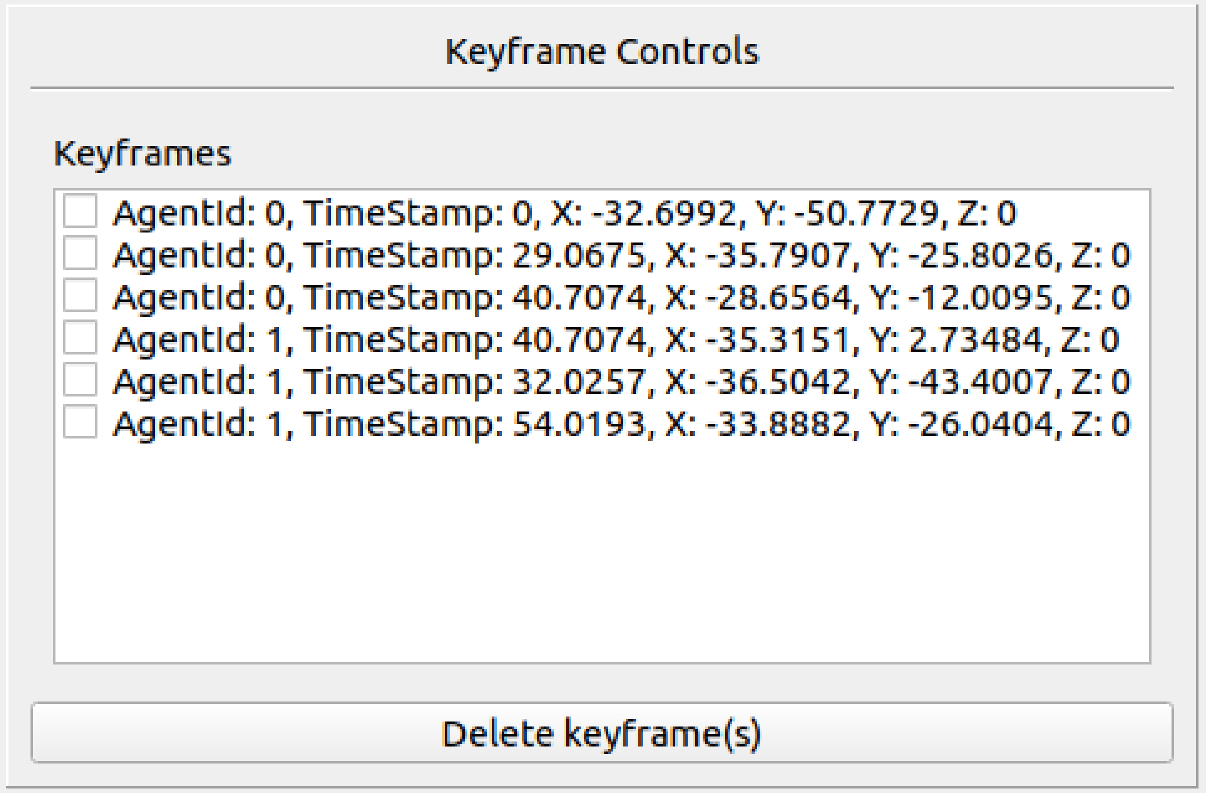
- ***Scenario Settings:*** This section allows you to define specific settings for the scenario, such as setting the location and size of the map.



- ***Agent Controls:*** In this section, you can manage the agents within the scenario. You have the ability to add new agents to the scenario, and changing the active agent between existing ones.



- ***Keyframe Controls:*** This section allows you to manage keyframes, which are specific points in time within the scenario that specify an agent's state.



5.1.3 The Planner tab

The **Planner** tab provides a visual representation of both a map and a timeline.

- The map display shows a graphical representation of the area.

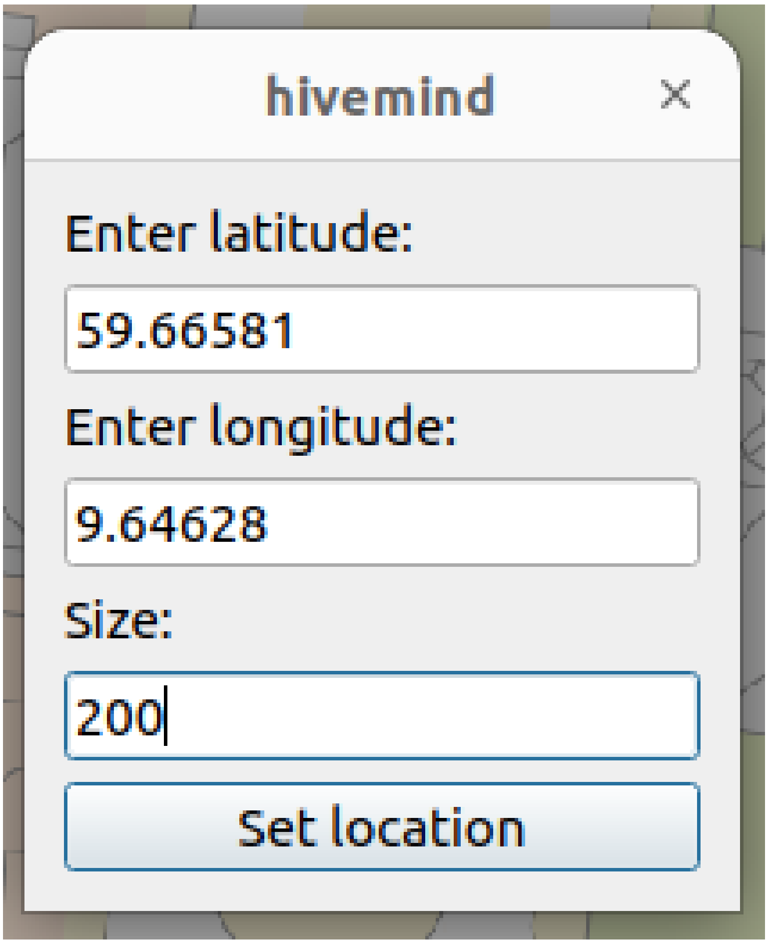
- The timeline displays the keyframes of all agents at the specified timestamps.

5.2 Functinality of Hivemind

5.2.1 Creating a scenario

5.2.1.1 Set scenario settings

1. Press scenario Settings button in the sidebar and a dialog box will pop up.
2. Specify the position on the map by entering geographical coordinates (latitude, longitude).
3. Determine the size of the map.
4. Click on the "Set Location" button to confirm the settings.



Landmark	Latitude	Longitude
Kongsberg church	59.66581	9.64628
Krona (University of South-Eastern Norway)	59.66471	9.64434
Hotel 1624	59.66944	9.65399
Nybrua	59.66761	9.64932
Gamlebrua	59.66265	9.65222
Train station	59.67221	9.65091

5.2.1.1.1 Popular Landmarks in Kongsberg

5.2.1.2 Add agents

1. Click on the *New Agent* button to create a new agent. This will make the newly created agent the active agent.

1. It is possible to select a color for the agent by choosing from the color options, located to the right in the Agent Control. This color will be used to visually identify the agent in the scenario.

1. If you want to switch to a previously created agent, simply click on the desired agent in the list or panel. This will make that agent the active agent, and you can view and modify its details as needed.

5.2.1.3 Add keyframes

1. To add keyframes, first select the agent for which you want to add keyframes.
2. Click on the timeline to set the desired timestamp for the keyframe.
3. Next, click on the map at the location where you want the keyframe to be associated.

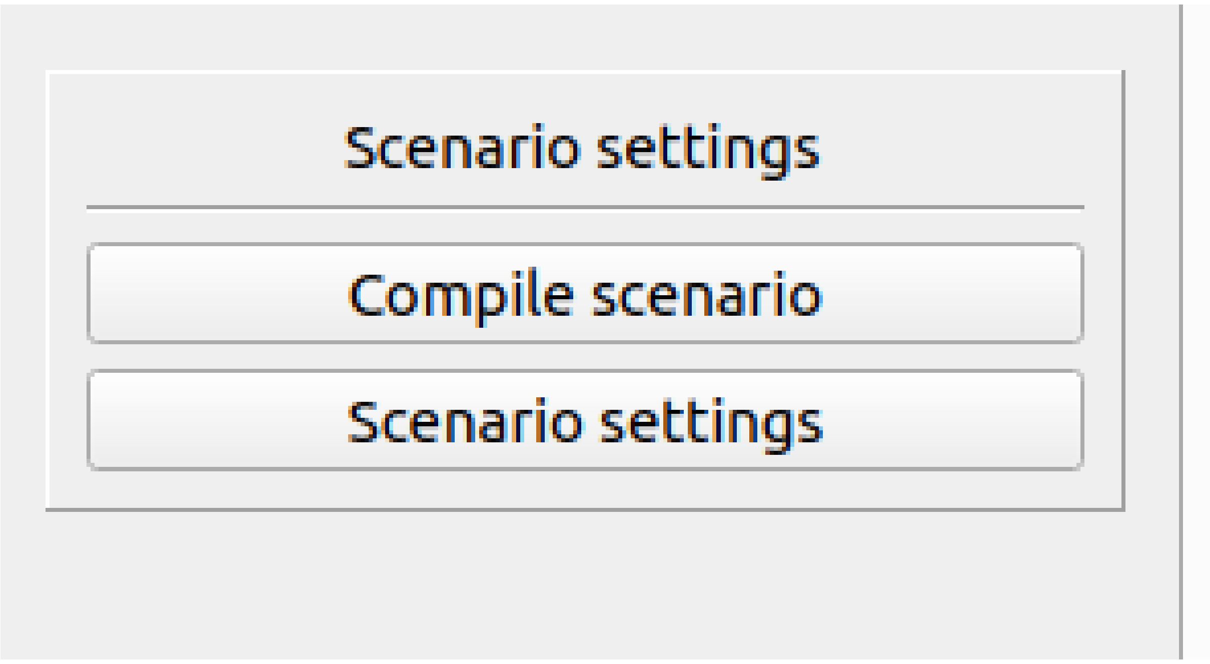
5.2.1.4 Delete keyframes

1. To delete keyframes, check the box(es) corresponding to the keyframe(s) you wish to remove in the keyframe controls panel.
2. Click on the "Delete Keyframes" button to delete the selected keyframes.

Alternatively, you can right click on any keyframe in the timeline to prompt deletion of that specific keyframe.

5.2.1.5 Compile Scenario

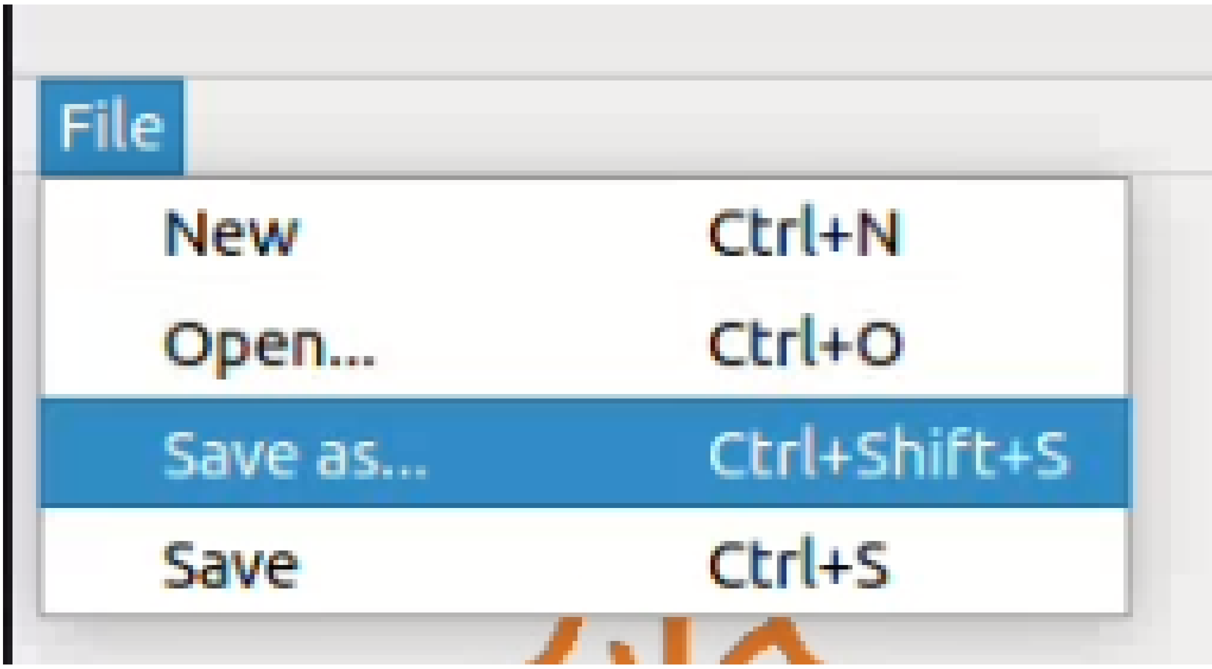
1. To compile the scenario, locate and click on the "Compile Scenario" button after you have Set scenario settings and added some keyframes to one or more drones.



1. The scenario will be displayed on the screen.

5.2.2 Saving Scenarios

1. Navigate to the *File* menu located in the top menu bar of the software.



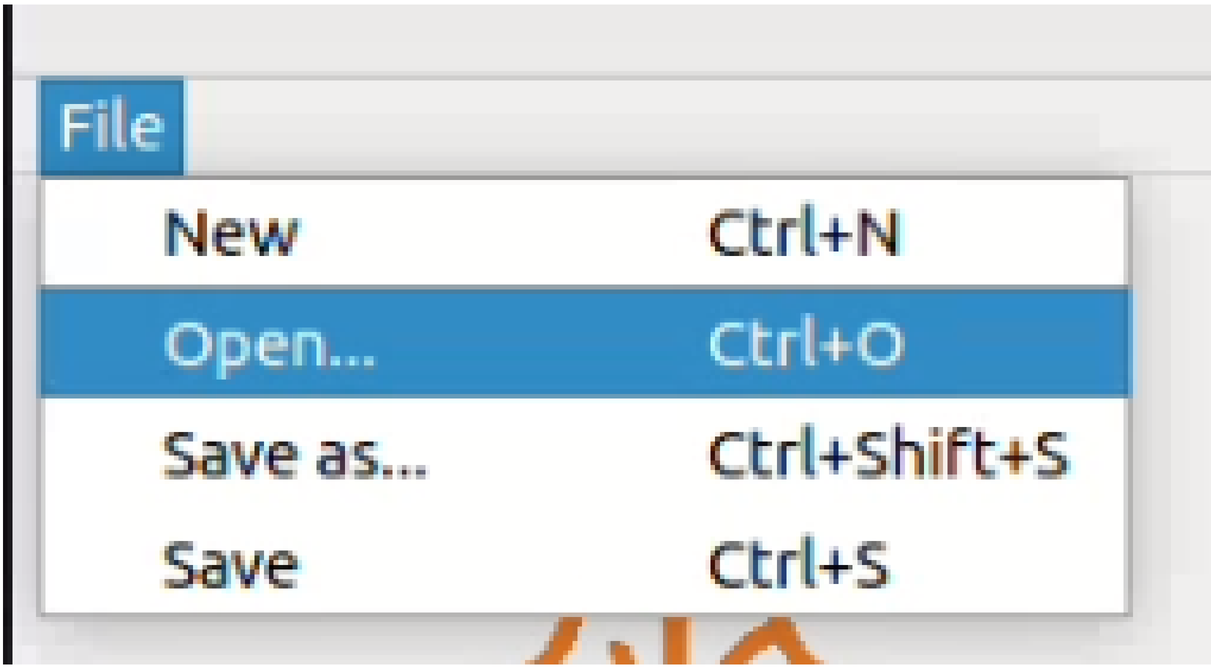
1. Click on *Save* from the dropdown menu. Alternatively, you can use the keyboard shortcut "Ctrl + shift + S".
2. A save dialog box will appear, allowing you to choose the location on your computer where you want to save the scenario.

3. Enter a file name for the scenario in the designated field. It is important to add the file extension ∗.hmsc∗. This is currently not added automatically, but if the proper extension is not added, you will not be able to load it again later.

1. Click the *Save* button to save the scenario with the specified name and format to the chosen location.

5.2.3 Loading Scenarios

1. Navigate to the *File* menu located in the top menu bar of the software.



1. Click on *Open* from the dropdown menu.
2. A file selection dialog box will appear. Navigate to the location where the saved scenario is stored.
3. Select the desired scenario file from the list or click on it to highlight it.

1. Click the *Open* button to load the selected scenario into the software.
2. Press the *Compile scenario* button and the scenario will be displayed in the map.

Chapter 6

Namespace Index

6.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 8

Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 9

File Index

9.1 File List

Here is a list of all files with brief descriptions:

Chapter 10

Namespace Documentation

10.1 CompileScenario Namespace Reference

Classes

- class Scenario

  The Scenario class represents a scenario with keyframes and routes.

10.2 CoordinateConverter Namespace Reference

Classes

- class CoordConv

  This is the class that performs coordinate conversions.

10.3 Core Namespace Reference

Classes

- struct Agent
- struct CartesianCoordinate

  A structure that represents a cartesian coordinate.
- struct GeographicalCoordinate

  A structure that represents a geographic coordinate.
- struct Keyframe

  A structure representing an agent's position in cartesian space at a given point in time.
- struct UTMCoordinate

  \ A structure that represents a coordinate in the Universal Transverse Mercator coordinate system

10.4 Gui Namespace Reference

Classes

- class Action

  Small wrapper around QAction.
- class AgentControls
- class ColorBox
- class KeyframeControls
- class KeyframeList
- class Launcher

  The launcher widget used to launch scenarios.
- class MainContent

  The main content of the main window.
- class MainWindow

  Handles the main window of Hivemind.
- class MapDialog

  The MapDialog class represents a dialog window for inputting map data.
- class MapViewer
- class MenuBar

  The main menubar of the user interface.
- class Planner

  The planner widget used for planning scenarios.
- class ScenarioControls
- class Sidebar

  The sidebar of the main window.
- class Simulator

  The simulator widget used to simulate scenarios.
- class TabWidget

  The tab widget of the main window.
- class Timeline

  A custom QWidget to represent a timeline with keyframes.

10.5 HeightManagement Namespace Reference

Classes

- class HeightManager

10.6 Json Namespace Reference

Classes

- class ISBool

  Implementation for bools.
- class ISConstructors

  Implemented for future expansion.

- class ISDouble

    Implementation for doubles.
- class ISDoubleVector

    Implementation for a vector with doubles.
- class ISFloat

    Implementation for floats.
- class ISFloatVector

    Implementation for a vector with floats.
- class ISInt

    Implementation for integers.
- class ISIntVector

    Implementation for a vector with integers.
- class ISMember

    Implementation for Members.
- class ISMemberVector

    Implementation for a vector with members.
- class ISMemVecVec

    Implementation for a vector with vectors with members.
- class ISObject

    Implementation for objects.
- class ISObjectVector

    Implementation for a vector with objects.
- class ISObjVecVec

    Implementation for a vector with vectors with objects.
- struct ISProperty

    Serializing and deserializing (persistent values) requires recflection which is a way for the programmer to ensure that the data you serialize will get back to the place you want it to be when you deserialize it later.
- class ISString

    Implementation for strings.
- class ISValue

    Rflection is made possible by the help of the ISValue class and the type classes.

Typedefs

- using ISValuePtr = std::shared_ptr< ISValue >
- using ISValues = std::vector< ISValuePtr >
- using ISProperties = std::vector< ISProperty >

    ISProperties is a vector with ISProperty.
- using ISIV = std::vector< int >
- using ISFV = std::vector< float >
- using ISDV = std::vector< double >

Functions

- void serialize (std::string filename, ISValue *p)

    Function to start serializing an onbject.
- void deserialize (std::string filename, ISValue *p)

    Function to start deserializing a file.

10.6.1 Typedef Documentation

10.6.1.1 ISDV

```
using Json::ISDV = typedef std::vector<double>
```

Definition at line 449 of file serializer.h.

10.6.1.2 ISFV

```
using Json::ISFV = typedef std::vector<float>
```

Definition at line 434 of file serializer.h.

10.6.1.3 ISIV

```
using Json::ISIV = typedef std::vector<int>
```

Definition at line 419 of file serializer.h.

10.6.1.4 ISProperties

```
using Json::ISProperties = typedef std::vector<ISProperty>
```

ISProperties is a vector with ISProperty.

Definition at line 34 of file serializer.h.

10.6.1.5 ISValuePtr

```
using Json::ISValuePtr = typedef std::shared_ptr<ISValue>
```

Definition at line 17 of file serializer.h.

10.6.1.6 ISValues

```
using Json::ISValues = typedef std::vector<ISValuePtr>
```

Definition at line 18 of file serializer.h.

10.6.2 Function Documentation

10.6.2.1 deserialize()

```
void Json::deserialize (
            std::string filename,
            ISValue * p )
```

Function to start deserializing a file.

Parameters

std::string	filename Name of the file you want to extract data from.
ISValue∗	p A pointer to the top object so it know where to start.

Definition at line 235 of file serializer.cpp.

References Json::ISValue::GetProperty().

Referenced by CompileScenario::Scenario::load().

10.6.2.2 serialize()

```
void Json::serialize (
            std::string filename,
            ISValue * p )
```

Function to start serializing an onbject.

Parameters

std::string	filename Name of the file you want to store the application data in.
ISValue∗	p A pointer to the object you want to serialize.

Definition at line 206 of file serializer.cpp.

References Json::ISValue::GetName(), and Json::ISValue::GetProperty().

Referenced by CompileScenario::Scenario::save().

10.7 KeyframeManagement Namespace Reference

Classes

- class KeyframeManager

  This is the class that manages keyframes.

10.8 MapManagement Namespace Reference

Classes

- class MapManager

  This is the class responsible for retrieving maps from Kartverket.

10.9 Routemaker Namespace Reference

Classes

- struct Cell2D
- class Graph

  Abstract graph interface optimized for path-finding.
- struct Node

  Represents a node in a Graph data structured made for path-finding.
- class Routemaker

  Main class responsible for handling creation of routes between keyframes.

Chapter 11

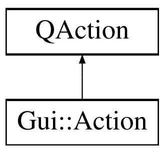
Class Documentation

11.1 Gui::Action Class Reference

Small wrapper around QAction.

```
#include <action.h>
```

Inheritance diagram for Gui::Action:



Public Member Functions

- Action (QWidget ∗parent, const QString &label, void(∗onClick)(void), const QKeySequence &shortcut=QKey↩ Sequence::UnknownKey)

    Constructs the Action widget.

11.1.1 Detailed Description

Small wrapper around QAction.

A tiny wrapper class around QAction that simply provides constructor arguments to add on-click functionality and keyboard shortcuts.

Definition at line 12 of file action.h.

11.1.2 Constructor & Destructor Documentation

11.1.2.1 Action()

```
Gui::Action::Action (
            QWidget * parent,
            const QString & label,
            void(*)(void) onClick,
            const QKeySequence & shortcut = QKeySequence::UnknownKey )
```

Constructs the Action widget.

Parameters

parent	The parent of the Action widget.
label	The label to be displayed in the action.
onClick	A function to call when the action is clicked.
shortcut	A keyboard shortcut to activate the action.

Typical usage:

```
Action* openAction = new Action(
    parent, QString::fromUtf8("Open..."),
    []() {
        QString fileName = QFileDialog::getOpenFileName(
            nullptr, QString::fromUtf8("Open Image"),
            QDir::currentPath(),
            QString::fromUtf8("Image Files (*.png *.jpg *.bmp)"));
        qInfo() « "File: " « fileName;
    },
    QKeySequence::Open);
```

Definition at line 9 of file action.cpp.

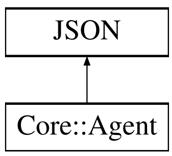
The documentation for this class was generated from the following files:

- include/gui/action.h
- src/gui/action.cpp

11.2 Core::Agent Struct Reference

```
#include <types.h>
```

Inheritance diagram for Core::Agent:



Public Member Functions

- Agent (int id=0, std::string name="Untitled Agent", std::string color="#FFFFFF")
- JSONSTART JSONINT (Id)
- JSONSTART JSONSTRING (Name)

Public Attributes

- int Id
- std::string Name
- std::string Color

11.2.1 Detailed Description

Definition at line 85 of file types.h.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 Agent()

```
Core::Agent::Agent (
            int id = 0,
            std::string name = "Untitled Agent",
            std::string color = "#FFFFFF" )  [inline]
```

Definition at line 87 of file types.h.

11.2.3 Member Function Documentation

11.2.3.1 JSONINT()

```
JSONSTART Core::Agent::JSONINT (
            Id )
```

11.2.3.2 JSONSTRING()

```
JSONSTART Core::Agent::JSONSTRING (
            Name )
```

11.2.4 Member Data Documentation

11.2.4.1 Color

```
std::string Core::Agent::Color
```

Definition at line 94 of file types.h.

Referenced by Gui::MapViewer::DrawKeyframes(), Gui::MapViewer::DrawRoutes(), Gui::AgentControls::SetActiveAgentIndex(), Gui::AgentControls::SetAgentColor(), and Gui::AgentControls::SyncColor().

11.2.4.2 Id

```
int Core::Agent::Id
```

Definition at line 92 of file types.h.

11.2.4.3 Name

```
std::string Core::Agent::Name
```

Definition at line 93 of file types.h.

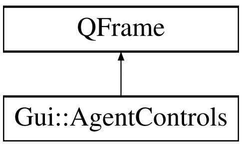
The documentation for this struct was generated from the following file:

- include/core/types.h

11.3 Gui::AgentControls Class Reference

```
#include <agent_controls.h>
```

Inheritance diagram for Gui::AgentControls:



Public Slots

- void UpdateAgents (std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator >)
- void SetActiveAgentIndex (int index)
- void SyncColor ()

Signals

- void AddAgent ()
- void AgentChanged (std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator >)
- void ActiveAgentChanged (int)

Public Member Functions

- AgentControls (QWidget ∗parent=nullptr)

Private Slots

- void SetAgentColor (QColor color)

Private Attributes

- QGridLayout * m_Layout
- QComboBox * m_ActiveAgentComboBox
- ColorBox * m_ActiveAgentColorBox
- QPushButton * m_NewAgentButton
- int m_ActiveAgentIndex
- std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator > m_Agents

11.3.1 Detailed Description

Definition at line 14 of file agent_controls.h.

11.3.2 Constructor & Destructor Documentation

11.3.2.1 AgentControls()

```
Gui::AgentControls::AgentControls (
            QWidget * parent = nullptr )  [explicit]
```

Definition at line 8 of file agent_controls.cpp.

References AddAgent(), m_ActiveAgentColorBox, m_ActiveAgentComboBox, m_Layout, m_NewAgentButton, SetActiveAgentIndex(), and SetAgentColor().

11.3.3 Member Function Documentation

11.3.3.1 ActiveAgentChanged

```
void Gui::AgentControls::ActiveAgentChanged (
            int  )  [signal]
```

Referenced by SetActiveAgentIndex(), and UpdateAgents().

11.3.3.2 AddAgent

```
void Gui::AgentControls::AddAgent ( )  [signal]
```

Referenced by AgentControls().

11.3.3.3 AgentChanged

```
void Gui::AgentControls::AgentChanged (
             std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >↩
::iterator >  )  [signal]
```

Referenced by SetAgentColor(), and UpdateAgents().

11.3.3.4 SetActiveAgentIndex

```
void Gui::AgentControls::SetActiveAgentIndex (
             int index )  [slot]
```

Definition at line 88 of file agent_controls.cpp.

References ActiveAgentChanged(), Core::Agent::Color, m_ActiveAgentColorBox, m_ActiveAgentIndex, m_Agents, and Gui::ColorBox::UpdateColor().

Referenced by AgentControls().

11.3.3.5 SetAgentColor

```
void Gui::AgentControls::SetAgentColor (
             QColor color )  [private], [slot]
```

Definition at line 54 of file agent_controls.cpp.

References AgentChanged(), Core::Agent::Color, and m_Agents.

Referenced by AgentControls().

11.3.3.6 SyncColor

```
void Gui::AgentControls::SyncColor ( )  [slot]
```

Definition at line 108 of file agent_controls.cpp.

References Core::Agent::Color, m_ActiveAgentColorBox, m_Agents, and Gui::ColorBox::UpdateColor().

11.3.3.7 UpdateAgents

```
void Gui::AgentControls::UpdateAgents (
            std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >↩
::iterator > agents )  [slot]
```

Definition at line 67 of file agent_controls.cpp.

References ActiveAgentChanged(), AgentChanged(), m_ActiveAgentColorBox, m_ActiveAgentComboBox, m_ActiveAgentIndex, and m_Agents.

11.3.4 Member Data Documentation

11.3.4.1 m_ActiveAgentColorBox

```
ColorBox* Gui::AgentControls::m_ActiveAgentColorBox  [private]
```

Definition at line 39 of file agent_controls.h.

Referenced by AgentControls(), SetActiveAgentIndex(), SyncColor(), and UpdateAgents().

11.3.4.2 m_ActiveAgentComboBox

```
QComboBox* Gui::AgentControls::m_ActiveAgentComboBox  [private]
```

Definition at line 38 of file agent_controls.h.

Referenced by AgentControls(), and UpdateAgents().

11.3.4.3 m_ActiveAgentIndex

```
int Gui::AgentControls::m_ActiveAgentIndex  [private]
```

Definition at line 42 of file agent_controls.h.

Referenced by SetActiveAgentIndex(), and UpdateAgents().

11.3.4.4 m_Agents

```
std::pair<std::vector<Core::Agent>::iterator, std::vector<Core::Agent>::iterator> Gui::↵
AgentControls::m_Agents [private]
```

Definition at line 46 of file agent_controls.h.

Referenced by SetActiveAgentIndex(), SetAgentColor(), SyncColor(), and UpdateAgents().

11.3.4.5 m_Layout

```
QGridLayout* Gui::AgentControls::m_Layout [private]
```

Definition at line 36 of file agent_controls.h.

Referenced by AgentControls().

11.3.4.6 m_NewAgentButton

```
QPushButton* Gui::AgentControls::m_NewAgentButton [private]
```

Definition at line 40 of file agent_controls.h.

Referenced by AgentControls().

The documentation for this class was generated from the following files:

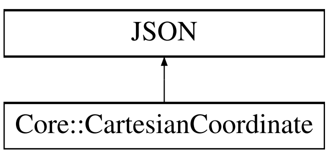
- include/gui/agent_controls.h
- src/gui/agent_controls.cpp

11.4 Core::CartesianCoordinate Struct Reference

A structure that represents a cartesian coordinate.

```
#include <types.h>
```

Inheritance diagram for Core::CartesianCoordinate:



Public Member Functions

- CartesianCoordinate (double x=0.0, double y=0.0, double z=0.0)
- JSONSTART JSONDOUBLE (X)
- JSONSTART JSONDOUBLE (Y)

Public Attributes

- double X
- double Y
- double Z

11.4.1 Detailed Description

A structure that represents a cartesian coordinate.

Definition at line 11 of file types.h.

11.4.2 Constructor & Destructor Documentation

11.4.2.1 CartesianCoordinate()

```
Core::CartesianCoordinate::CartesianCoordinate (
            double x = 0.0,
            double y = 0.0,
            double z = 0.0 )  [inline]
```

Definition at line 13 of file types.h.

11.4.3 Member Function Documentation

11.4.3.1 JSONDOUBLE() [1/2]

```
JSONSTART Core::CartesianCoordinate::JSONDOUBLE (
            X )
```

11.4.3.2 JSONDOUBLE() [2/2]

```
JSONSTART Core::CartesianCoordinate::JSONDOUBLE (
            Y )
```

11.4.4 Member Data Documentation

11.4.4.1 X

`double Core::CartesianCoordinate::X`

Definition at line 17 of file types.h.

Referenced by CoordinateConverter::CoordConv::AsymmetricToSymmetric(), CoordinateConverter::CoordConv::CartesianToGeogra
Gui::MapViewer::DrawRoutes(), Routemaker::Routemaker::MakeRoute(), KeyframeManagement::KeyframeManager::RemoveKeyfra
and CoordinateConverter::CoordConv::SymmetricToAsymmetric().

11.4.4.2 Y

`double Core::CartesianCoordinate::Y`

Definition at line 18 of file types.h.

Referenced by CoordinateConverter::CoordConv::AsymmetricToSymmetric(), CoordinateConverter::CoordConv::CartesianToGeogra
Gui::MapViewer::DrawRoutes(), KeyframeManagement::KeyframeManager::RemoveKeyframe(), and CoordinateConverter::CoordCo

11.4.4.3 Z

`double Core::CartesianCoordinate::Z`

Definition at line 19 of file types.h.

Referenced by CoordinateConverter::CoordConv::CartesianToGeographical(), and KeyframeManagement::KeyframeManager::Remo

The documentation for this struct was generated from the following file:

- include/core/types.h

11.5 Routemaker::Cell2D Struct Reference

`#include <routemaker.h>`

Public Attributes

- uint32_t X
- uint32_t Y
- bool Occupied

11.5.1 Detailed Description

Definition at line 14 of file routemaker.h.

11.5.2 Member Data Documentation

11.5.2.1 Occupied

`bool Routemaker::Cell2D::Occupied`

Definition at line 17 of file routemaker.h.

11.5.2.2 X

`uint32_t Routemaker::Cell2D::X`

Definition at line 16 of file routemaker.h.

11.5.2.3 Y

`uint32_t Routemaker::Cell2D::Y`

Definition at line 16 of file routemaker.h.

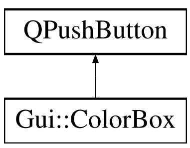
The documentation for this struct was generated from the following file:

- include/routemaker/routemaker.h

11.6 Gui::ColorBox Class Reference

`#include <color_box.h>`

Inheritance diagram for Gui::ColorBox:



Public Slots

- void UpdateColor (QColor color)

Signals

- void ColorUpdated (QColor color)

Public Member Functions

- ColorBox (QWidget ∗parent=nullptr)

Protected Member Functions

- void paintEvent (QPaintEvent ∗event) override
- void mousePressEvent (QMouseEvent ∗event) override

Private Slots

- void SelectColor ()

Private Attributes

- QColor m_Color
- QColorDialog ∗ m_ColorDialog

11.6.1 Detailed Description

Definition at line 9 of file color_box.h.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 ColorBox()

```
Gui::ColorBox::ColorBox (
            QWidget * parent = nullptr )  [explicit]
```

Definition at line 9 of file color_box.cpp.

References m_ColorDialog.

11.6.3 Member Function Documentation

11.6.3.1 ColorUpdated

```
void Gui::ColorBox::ColorUpdated (
            QColor color )  [signal]
```

Referenced by SelectColor().

11.6.3.2 mousePressEvent()

```
void Gui::ColorBox::mousePressEvent (
            QMouseEvent * event )  [override], [protected]
```

Definition at line 39 of file color_box.cpp.

References SelectColor().

11.6.3.3 paintEvent()

```
void Gui::ColorBox::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

Definition at line 21 of file color_box.cpp.

References m_Color.

11.6.3.4 SelectColor

```
void Gui::ColorBox::SelectColor ( )  [private], [slot]
```

Definition at line 52 of file color_box.cpp.

References ColorUpdated(), m_Color, and m_ColorDialog.

Referenced by mousePressEvent().

11.6.3.5 UpdateColor

```
void Gui::ColorBox::UpdateColor (
            QColor color )  [slot]
```

Definition at line 45 of file color_box.cpp.

References m_Color, and m_ColorDialog.

Referenced by Gui::AgentControls::SetActiveAgentIndex(), and Gui::AgentControls::SyncColor().

11.6.4 Member Data Documentation

11.6.4.1 m_Color

```
QColor Gui::ColorBox::m_Color  [private]
```

Definition at line 29 of file color_box.h.

Referenced by paintEvent(), SelectColor(), and UpdateColor().

11.6.4.2 m_ColorDialog

```
QColorDialog* Gui::ColorBox::m_ColorDialog  [private]
```

Definition at line 30 of file color_box.h.

Referenced by ColorBox(), SelectColor(), and UpdateColor().

The documentation for this class was generated from the following files:

- include/gui/color_box.h
- src/gui/color_box.cpp

11.7 CoordinateConverter::CoordConv Class Reference

This is the class that performs coordinate conversions.

```
#include <coordinate_converter.h>
```

Static Public Member Functions

- static void ResetOrigin (Core::GeographicalCoordinate geoCoord, int size)

    Sets the origin coordinate to use with relative coordinates.
- static Core::CartesianCoordinate GeographicalToCartesian (Core::GeographicalCoordinate geoCoord)

    Function used to convert a geographical coordinate to a cartesian coordinate.
- static Core::GeographicalCoordinate CartesianToGeographical (Core::CartesianCoordinate cartCoord)

    \biref Function used to convert a cartesian coordinate to a geograpical coordinate
- static Core::GeographicalCoordinate GetOrigin ()
- static Core::CartesianCoordinate SymmetricToAsymmetric (Core::CartesianCoordinate symmetric)

    Function used to convert a coordinate in a symmetric coordinate system to a coordinate in an asymmetric coordinate system.
- static Core::CartesianCoordinate AsymmetricToSymmetric (Core::CartesianCoordinate asymmetric)

    Function used to convert a coordinate in an asymmetric cooridnate system to a coordinate in a symmetric coordinate system.
- static Core::UTMCoordinate GeographicToUTM (Core::GeographicalCoordinate GeoCoord)

    Function used to convert a geographical coordinate to a UTM coordinate.
- static Core::GeographicalCoordinate UTMToGeographic (Core::UTMCoordinate UTMCoord)

    Function used to convert a UTM coordinate to a geographical coordinate.
- static int GetSize ()

Private Member Functions

- CoordConv ()

    The constructor is made private to adhere to the singleton pattern.

Static Private Member Functions

- static CoordConv & GetInstance ()

    Get the single instance of CoordConv.

Private Attributes

- Core::GeographicalCoordinate m_OriginGeographical
- GeographicLib::LocalCartesian m_Origin
- int m_Size

11.7.1 Detailed Description

This is the class that performs coordinate conversions.

Definition at line 13 of file coordinate_converter.h.

11.7.2 Constructor & Destructor Documentation

11.7.2.1 CoordConv()

```
CoordinateConverter::CoordConv::CoordConv ( )  [inline], [private]
```

The constructor is made private to adhere to the singleton pattern.

Definition at line 91 of file coordinate_converter.h.

11.7.3 Member Function Documentation

11.7.3.1 AsymmetricToSymmetric()

```
Core::CartesianCoordinate CoordinateConverter::CoordConv::AsymmetricToSymmetric (
            Core::CartesianCoordinate asymmetric )  [static]
```

Function used to convert a coordinate in an asymmetric cooridnate system to a coordinate in a symmetric coordinate system.

Parameters

asymmetric	Cartesian coordinate in an asymmetric coordinate system

Returns

The symmetric coordinate corresponds to the asymmetric coordinate

Definition at line 66 of file coordinate_converter.cpp.

References GetInstance(), Core::CartesianCoordinate::X, and Core::CartesianCoordinate::Y.

Referenced by Routemaker::Routemaker::MakeRoute(), and Gui::MapViewer::mousePressEvent().

11.7.3.2 CartesianToGeographical()

```
Core::GeographicalCoordinate CoordinateConverter::CoordConv::CartesianToGeographical (
            Core::CartesianCoordinate cartCoord )  [static]
```

\biref Function used to convert a cartesian coordinate to a geograpical coordinate

Parameters

cartCoord	Cartesian coordinate to convert

Returns

return a geographical point relative to origin and the cartesian coordinates.

Definition at line 33 of file coordinate_converter.cpp.

References GetInstance(), Core::CartesianCoordinate::X, Core::CartesianCoordinate::Y, and Core::CartesianCoordinate::Z.

11.7.3.3 GeographicalToCartesian()

```
Core::CartesianCoordinate CoordinateConverter::CoordConv::GeographicalToCartesian (
            Core::GeographicalCoordinate geoCoord )  [static]
```

Function used to convert a geographical coordinate to a cartesian coordinate.

Parameters

geoCoord	Geograhical coordinate to convert

Returns

return a cartesian point relative to origin

Definition at line 22 of file coordinate_converter.cpp.

References GetInstance(), Core::GeographicalCoordinate::Latitude, and Core::GeographicalCoordinate::Longitude.

11.7.3.4 GeographicToUTM()

```
Core::UTMCoordinate CoordinateConverter::CoordConv::GeographicToUTM (
            Core::GeographicalCoordinate GeoCoord )  [static]
```

Function used to convert a geographical coordinate to a UTM coordinate.

Parameters

GeoCoord	Geographical coordinate

Returns

UTM coordinate corresponds to the geographical coordinate

Definition at line 78 of file coordinate_converter.cpp.

References Core::UTMCoordinate::Easting, Core::UTMCoordinate::IsNorthHemisphere, Core::GeographicalCoordinate::Latitude, Core::GeographicalCoordinate::Longitude, Core::UTMCoordinate::Northing, and Core::UTMCoordinate::Zone.

Referenced by CompileScenario::Scenario::Scenario(), and CompileScenario::Scenario::SetOrigin().

11.7.3.5 GetInstance()

```
static CoordConv & CoordinateConverter::CoordConv::GetInstance ( )  [inline], [static], [private]
```

Get the single instance of CoordConv.

Returns

The single instance of CoordConv.

Definition at line 97 of file coordinate_converter.h.

Referenced by AsymmetricToSymmetric(), CartesianToGeographical(), GeographicalToCartesian(), GetOrigin(), GetSize(), ResetOrigin(), and SymmetricToAsymmetric().

11.7.3.6 GetOrigin()

```
Core::GeographicalCoordinate CoordinateConverter::CoordConv::GetOrigin ( )  [static]
```

Returns

The geographical coordinates to origin.

Definition at line 44 of file coordinate_converter.cpp.

References GetInstance().

11.7.3.7 GetSize()

```
static int CoordinateConverter::CoordConv::GetSize ( )  [inline], [static]
```

Definition at line 82 of file coordinate_converter.h.

References GetInstance(), and m_Size.

Referenced by Gui::MapViewer::DrawKeyframes(), Gui::MapViewer::DrawRoutes(), and Gui::MapViewer::mousePressEvent().

11.7.3.8 ResetOrigin()

```
void CoordinateConverter::CoordConv::ResetOrigin (
            Core::GeographicalCoordinate geoCoord,
            int size )  [static]
```

Sets the origin coordinate to use with relative coordinates.

Parameters

geoCoord	Geographical coordinate to be used as the origin of relative coordinates

Definition at line 10 of file coordinate_converter.cpp.

References GetInstance(), Core::GeographicalCoordinate::Latitude, and Core::GeographicalCoordinate::Longitude.

Referenced by CompileScenario::Scenario::Scenario(), and CompileScenario::Scenario::SetOrigin().

11.7.3.9 SymmetricToAsymmetric()

```
Core::CartesianCoordinate CoordinateConverter::CoordConv::SymmetricToAsymmetric (
            Core::CartesianCoordinate symmetric )  [static]
```

Function used to convert a coordinate in a symmetric coordinate system to a coordinate in an asymmetric coordinate system.

Parameters

symmetric	Cartesian coordinate in a symmetric coordinate system

Returns

The asymmetric coordinate corresponds to the symmetric coordinate

Definition at line 54 of file coordinate_converter.cpp.

References GetInstance(), Core::CartesianCoordinate::X, and Core::CartesianCoordinate::Y.

Referenced by Gui::MapViewer::DrawKeyframes(), Gui::MapViewer::DrawRoutes(), and Routemaker::Routemaker::MakeRoute().

11.7.3.10 UTMToGeographic()

```
Core::GeographicalCoordinate CoordinateConverter::CoordConv::UTMToGeographic (
            Core::UTMCoordinate UTMCoord )  [static]
```

Function used to convert a UTM coordinate to a geographical coordinate.

Parameters

UTMCoord	UTM coordinate

Returns

Geographical coordinate corresponds to the UTM coordinate

Definition at line 90 of file coordinate_converter.cpp.

References Core::UTMCoordinate::Easting, Core::UTMCoordinate::IsNorthHemisphere, Core::GeographicalCoordinate::Latitude, Core::GeographicalCoordinate::Longitude, Core::UTMCoordinate::Northing, and Core::UTMCoordinate::Zone.

11.7.4 Member Data Documentation

11.7.4.1 m_Origin

`GeographicLib::LocalCartesian CoordinateConverter::CoordConv::m_Origin [private]`

Definition at line 105 of file coordinate_converter.h.

11.7.4.2 m_OriginGeographical

`Core::GeographicalCoordinate CoordinateConverter::CoordConv::m_OriginGeographical [private]`

Definition at line 104 of file coordinate_converter.h.

11.7.4.3 m_Size

`int CoordinateConverter::CoordConv::m_Size [private]`

Definition at line 106 of file coordinate_converter.h.

Referenced by GetSize().

The documentation for this class was generated from the following files:

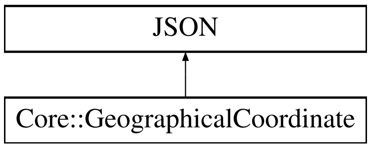
- include/coordinate_converter/coordinate_converter.h
- src/coordinate_converter/coordinate_converter.cpp

11.8 Core::GeographicalCoordinate Struct Reference

A structure that represents a geographic coordinate.

```
#include <types.h>
```

Inheritance diagram for Core::GeographicalCoordinate:



Public Member Functions

- GeographicalCoordinate (double lat, double lon)
- JSONSTART JSONDOUBLE (Latitude)

Public Attributes

- double Latitude
- double Longitude

11.8.1 Detailed Description

A structure that represents a geographic coordinate.

Definition at line 28 of file types.h.

11.8.2 Constructor & Destructor Documentation

11.8.2.1 GeographicalCoordinate()

```
Core::GeographicalCoordinate::GeographicalCoordinate (
            double lat,
            double lon )  [inline]
```

Definition at line 30 of file types.h.

11.8.3 Member Function Documentation

11.8.3.1 JSONDOUBLE()

```
JSONSTART Core::GeographicalCoordinate::JSONDOUBLE (
            Latitude  )
```

11.8.4 Member Data Documentation

11.8.4.1 Latitude

```
double Core::GeographicalCoordinate::Latitude
```

Definition at line 34 of file types.h.

Referenced by CoordinateConverter::CoordConv::GeographicalToCartesian(), CoordinateConverter::CoordConv::GeographicToUTM(CoordinateConverter::CoordConv::ResetOrigin(), and CoordinateConverter::CoordConv::UTMToGeographic().

11.8.4.2 Longitude

```
double Core::GeographicalCoordinate::Longitude
```

Definition at line 35 of file types.h.

Referenced by CoordinateConverter::CoordConv::GeographicalToCartesian(), CoordinateConverter::CoordConv::GeographicToUTM(CoordinateConverter::CoordConv::ResetOrigin(), and CoordinateConverter::CoordConv::UTMToGeographic().

The documentation for this struct was generated from the following file:

- include/core/types.h

11.9 Routemaker::Graph< T > Class Template Reference

Abstract graph interface optimized for path-finding.

```
#include <graph.h>
```

Public Types

- using NodePtr = std::shared_ptr< Node< T > >

  Helper alias to make code more readable.

Public Member Functions

- virtual std::vector< NodePtr > GetNeighbors (NodePtr node)=0

  Collects all neighbor nodes of `node`*.*
- virtual double GetCost (NodePtr a, NodePtr b)=0

  Returns the cost between `a` *and* `b`*.*
- virtual bool HasLineOfSight (NodePtr a, NodePtr b)=0

  Determines if there is a direct line of sight between node `a` *and node* `b`*.*
- virtual void ResetNodes (void)=0

  Resets all local and global goals and parent relationships of all nodes.
- void SolveAStar (NodePtr start, NodePtr goal)

  Finds cheapest path from `start` *to* `goal`*.*
- void PostSmooth (NodePtr start, NodePtr goal)

  Simplifies the path from `start` *to* `goal`*.*

11.9.1 Detailed Description

template<typename T>
class Routemaker::Graph< T >

Abstract graph interface optimized for path-finding.

Template Parameters

T	Type of user data to store in each node

This interface is designed to be flexible and scalable. The sub-classes are required to implement a few methods, such as Graph::GetNeighbors and Graph::GetCost for the A∗ path-finding algorithm to work.

Definition at line 73 of file graph.h.

11.9.2 Member Typedef Documentation

11.9.2.1 NodePtr

```
template<typename T >
using Routemaker::Graph< T >::NodePtr = std::shared_ptr<Node<T> >
```

Helper alias to make code more readable.

Definition at line 76 of file graph.h.

11.9.3 Member Function Documentation

11.9.3.1 GetCost()

```
template<typename T >
virtual double Routemaker::Graph< T >::GetCost (
            NodePtr a,
            NodePtr b ) [pure virtual]
```

Returns the cost between a and b.

Implemented by sub-classes of Graph. The a∗ path-finding algorithm uses cost to efficiently find the best path between two nodes. In order to do this, it requires some method of calculating the cost of moving between any two nodes. It is up to the sub-class to define how this is calulated. An example of this cost may be the euclidean distance between two nodes.

Parameters

a	Pointer to the first Node
b	Pointer to the second Node

Returns

Cost between node a and node b.

Implemented in Routemaker::Routemaker.

11.9.3.2 GetNeighbors()

```
template<typename T >
virtual std::vector< NodePtr > Routemaker::Graph< T >::GetNeighbors (
            NodePtr node ) [pure virtual]
```

Collects all neighbor nodes of node.

Implemented by sub-classes of Graph. The neighbor relationship between nodes define the edges of the graph. It is up to the subclass to define these relationships. For a 2D grid, the neighbors would simply be the nodes directly to the north, south, east and west, in addition to the corners between them. For a road network, the relationships may be more complex.

Parameters

node	A pointer to the node from which to collect all neighbors

Returns

A vector of pointers to all the neighbors of node

Implemented in Routemaker::Routemaker.

11.9.3.3 HasLineOfSight()

```
template<typename T >
virtual bool Routemaker::Graph< T >::HasLineOfSight (
            NodePtr a,
            NodePtr b ) [pure virtual]
```

Determines if there is a direct line of sight between node `a` and node `b`.

Implemented by sub-classes of Graph. The Graph::PostSmooth method traverses the already found path through the A∗ path-finding algorithm and simplifies it by using this method. In a graph representing a 2D grid, a Bresenham implementation or ray-casting can be used to determine line of sight.

Parameters

a	Pointer to the first Node
b	Pointer to the second Node

Returns

bool specifying whether or not there is a direct line of sight

Implemented in Routemaker::Routemaker.

11.9.3.4 PostSmooth()

```
template<typename T >
void Routemaker::Graph< T >::PostSmooth (
            NodePtr start,
            NodePtr goal )
```

Simplifies the path from `start` to `goal`.

Parameters

start	Pointer to the start node of the path
goal	Pointer to the end node of the path

Should be run on the same nodes as Graph::SolveAStar, and should only be called after Graph::SolveAStar has finished.

Definition at line 231 of file graph.h.

11.9.3.5 ResetNodes()

```
template<typename T >
virtual void Routemaker::Graph< T >::ResetNodes (
            void ) [pure virtual]
```

Resets all local and global goals and parent relationships of all nodes.

Implemented by sub-classes of Graph. In order to be able to re-use the same graph for several A∗ searches, the Graph::SolveAStar method needs to be able to reset all the nodes. As this interface does not contain the actual collection of nodes, this needs to be implemented in the sub-classes.

Implemented in Routemaker::Routemaker.

11.9.3.6 SolveAStar()

```
template<typename T >
void Routemaker::Graph< T >::SolveAStar (
            NodePtr start,
            NodePtr goal )
```

Finds *cheapest* path from `start` to `goal`.

Parameters

start	Pointer to the node to start the path from
goal	Pointer to the node to find the path to

Using the A∗ algorithm, this method explores the graph's nodes and updates their local and global goals, their visited flags, as well as their parent relationships.

When the algorithm finishes, given that a path exists between the nodes, the cheapest path between them is defined by the parent relationships. The path can be *extracted* by starting at the `goal` and following the Node::Parent pointers until `start` is reached, saving each node in a list and reversing it at the end.

Definition at line 165 of file graph.h.

The documentation for this class was generated from the following file:

- include/routemaker/graph.h

11.10 HeightManagement::HeightManager::heightdata Struct Reference

`#include <height_manager.h>`

Public Attributes

- double x
- double y
- double z

11.10.1 Detailed Description

Definition at line 14 of file height_manager.h.

11.10.2 Member Data Documentation

11.10.2.1 x

```
double HeightManagement::HeightManager::heightdata::x
```

Definition at line 16 of file height_manager.h.

Referenced by HeightManagement::HeightManager::GetHeightAbsolute(), HeightManagement::HeightManager::GetVertexAbsolute(), HeightManagement::HeightManager::PopulateVertices(), and HeightManagement::HeightManager::ValidInput().

11.10.2.2 y

```
double HeightManagement::HeightManager::heightdata::y
```

Definition at line 17 of file height_manager.h.

Referenced by HeightManagement::HeightManager::GetHeightAbsolute(), HeightManagement::HeightManager::GetVertexAbsolute(), HeightManagement::HeightManager::PopulateVertices(), and HeightManagement::HeightManager::ValidInput().

11.10.2.3 z

```
double HeightManagement::HeightManager::heightdata::z
```

Definition at line 18 of file height_manager.h.

Referenced by HeightManagement::HeightManager::GetHeight(), HeightManagement::HeightManager::GetHeightAbsolute(), HeightManagement::HeightManager::GetVertex(), HeightManagement::HeightManager::GetVertexAbsolute(), HeightManagement::HeightManager::PopulateVertices(), and HeightManagement::HeightManager::UpdateOrigin().

The documentation for this struct was generated from the following file:

- include/height_management/height_manager.h

11.11 HeightManagement::HeightManager Class Reference

```
#include <height_manager.h>
```

Classes

- struct heightdata

Public Member Functions

- HeightManager ()

  Constructor of HeightManager class.
- void UpdateOrigin (Core::UTMCoordinate UTMCoord, int size)

  Function to update the origin point.
- bool GetVertex (int inputRelativeX, int inputRelativeY, heightdata &vertex)

  Function to return the whole "height_management" for a given point.
- bool GetHeight (int inputRelativeX, int inputRelativeY, float &height)

  Function to return height, given relative coordinates (from a system where 0, 0 is in the upper left corner)
- bool GetVertexAbsolute (double inputX, double inputY, heightdata &vertex)

  Function to get the height_management of an absolute (geographic) coordinate, using the same coordinate system of the dataset.
- float GetHeightAbsolute (double inputX, double inputY)

  Function to get the height of an absolute (geographic) coordinate, using the same coordinate system of the dataset.
- void LoadTif (const char ∗filePath, double x, double y)

  Function to allow user to change GeoTiff file used in planning.

Private Member Functions

- void PopulateVertices ()

  Function that will open the GeoTiff file and extract all heights for the given subset of the dataset used.
- bool ValidInput (int x, int y)

  Function to test whether a point exists within the scope of the selected data subset.
- bool ValidInput (double x, double y)

  Function to test whether a point exists within the scope of the elected data subset.
- bool OrigoWithinBounds (double x, double y)

  Function that tests whether the selected origin point is within the bounds of the currently active data set, given the buffer size required to extract the subset.
- void UpdateCornerCoords ()

  Function to update the corner coordinates saved within the member instance of the chosen dataset.

Private Attributes

- const char ∗ m_CachedTifName = "../res/Kongsberg.tif"
- const char ∗ m_CoordinateSystem { "UTM33" }
- int m_Resolution { 1 }
- int m_Size
- long m_UpperLeftX
- long m_UpperLeftY
- long m_LowerRightX
- long m_LowerRightY
- heightdata ∗ m_Vertices
- heightdata m_Origo { 0, 0, 0 }
- heightdata m_SelectionCorner

11.11.1 Detailed Description

Definition at line 11 of file height_manager.h.

11.11.2 Constructor & Destructor Documentation

11.11.2.1 HeightManager()

```
HeightManagement::HeightManager::HeightManager ( )
```

Constructor of HeightManager class.

Returns

No object.

Definition at line 9 of file height_manager.cpp.

11.11.3 Member Function Documentation

11.11.3.1 GetHeight()

```
bool HeightManagement::HeightManager::GetHeight (
            int inputRelativeX,
            int inputRelativeY,
            float & height )
```

Function to return height, given relative coordinates (from a system where 0, 0 is in the upper left corner)

Parameters

inputRelativeX	The relative X value of the point.
inputRelativeY	The relative Y value of the point.

Returns

A float containing the height value of the point in metres.

Definition at line 163 of file height_manager.cpp.

References m_Size, m_Vertices, ValidInput(), and HeightManagement::HeightManager::heightdata::z.

11.11.3.2 GetHeightAbsolute()

```
float HeightManagement::HeightManager::GetHeightAbsolute (
            double inputX,
            double inputY )
```

Function to get the height of an absolute (geographic) coordinate, using the same coordinate system of the dataset.

Parameters

inputX	The absolute X value of the point.
inputY	The absolute Y value of the point.

Returns

A float containing the height of the point in metres.

Definition at line 148 of file height_manager.cpp.

References m_SelectionCorner, m_Size, m_Vertices, ValidInput(), HeightManagement::HeightManager::heightdata::x, HeightManagement::HeightManager::heightdata::y, and HeightManagement::HeightManager::heightdata::z.

Referenced by UpdateOrigin().

11.11.3.3 GetVertex()

```
bool HeightManagement::HeightManager::GetVertex (
            int inputRelativeX,
            int inputRelativeY,
            HeightManager::heightdata & vertex )
```

Function to return the whole "height_management" for a given point.

Parameters

inputRelativeX	The X coordinate in the relative system (where 0,0 is the top left corner of the system).
inputRelativeY	The Y coordinate in the relative system.

Returns

A height_management, containing the geographic (absolute) x, y and z coordinates.

Definition at line 116 of file height_manager.cpp.

References m_Size, m_Vertices, ValidInput(), and HeightManagement::HeightManager::heightdata::z.

11.11.3.4 GetVertexAbsolute()

```
bool HeightManagement::HeightManager::GetVertexAbsolute (
            double inputX,
            double inputY,
            HeightManager::heightdata & vertex )
```

Function to get the height_management of an absolute (geographic) coordinate, using the same coordinate system of the dataset.

Parameters

inputX	The absolute X value of the point.
inputY	The absolute Y value of the point.

Returns

A float containing the height of the point in metres.

Definition at line 131 of file height_manager.cpp.

References m_SelectionCorner, m_Size, m_Vertices, ValidInput(), HeightManagement::HeightManager::heightdata::x, HeightManagement::HeightManager::heightdata::y, and HeightManagement::HeightManager::heightdata::z.

11.11.3.5 LoadTif()

```
void HeightManagement::HeightManager::LoadTif (
            const char * filePath,
            double x,
            double y )
```

Function to allow user to change GeoTiff file used in planning.

If this function is not run, the user can still update the origin and Hivemind will run using the cached GeoTiff file.

Parameters

filePath	Complete file path of the file to be used.
x	X coordinate used for GeoTiff subset origin. Height data will be populated in a 500x500 pixel centered on the origin point. This is hard coded into the class.
y	Y coordinate used for GeoTiff subset origin.

Returns

No object, but will update the path for the cached tif.

Definition at line 12 of file height_manager.cpp.

References m_CachedTifName, m_Size, and UpdateOrigin().

11.11.3.6 OrigoWithinBounds()

```
bool HeightManagement::HeightManager::OrigoWithinBounds (
            double x,
            double y )  [private]
```

Function that tests whether the selected origin point is within the bounds of the currently active data set, given the buffer size required to extract the subset.

Parameters

x	The X value of the origin point.
y	The Y value of the origin point.

Returns

A bool indicating whether or not the origin point is within bounds.

Definition at line 195 of file height_manager.cpp.

References m_LowerRightX, m_LowerRightY, m_Size, m_UpperLeftX, and m_UpperLeftY.

Referenced by UpdateOrigin().

11.11.3.7 PopulateVertices()

```
void HeightManagement::HeightManager::PopulateVertices ( ) [private]
```

Function that will open the GeoTiff file and extract all heights for the given subset of the dataset used.

Returns

No object, but after this has run, all heights will have been imported into the instance of the class and the various GetHeight methods can be run.

Definition at line 46 of file height_manager.cpp.

References m_CachedTifName, m_Origo, m_SelectionCorner, m_Size, m_Vertices, HeightManagement::HeightManager::heightdata HeightManagement::HeightManager::heightdata::y, and HeightManagement::HeightManager::heightdata::z.

Referenced by UpdateOrigin().

11.11.3.8 UpdateCornerCoords()

```
void HeightManagement::HeightManager::UpdateCornerCoords ( ) [private]
```

Function to update the corner coordinates saved within the member instance of the chosen dataset.

Returns

No object, but the corner coordinates will be updated, given there were no problems opening the GeoTiff file.

Definition at line 206 of file height_manager.cpp.

References m_CachedTifName, m_LowerRightX, m_LowerRightY, m_UpperLeftX, and m_UpperLeftY.

Referenced by UpdateOrigin().

11.11.3.9 UpdateOrigin()

```
void HeightManagement::HeightManager::UpdateOrigin (
            Core::UTMCoordinate UTMCoord,
            int size )
```

Function to update the origin point.

Running this will also trigger the population of height data for the chosen subset of the GeoTiff file.

Parameters

x	X coordinate used for GeoTiff subset origin.
y	Y coordinate used for GeoTiff subset origin.

Returns

No object, but will update the origin for this instance of HeightManager and will populate the instance with height data.

Definition at line 21 of file height_manager.cpp.

References Core::UTMCoordinate::Easting, GetHeightAbsolute(), m_CoordinateSystem, m_LowerRightX, m_LowerRightY, m_Origo, m_Size, m_UpperLeftX, m_UpperLeftY, m_Vertices, Core::UTMCoordinate::Northing, OrigoWithinBounds(), PopulateVertices(), UpdateCornerCoords(), and HeightManagement::HeightManager::heightdata::z.

Referenced by LoadTif().

11.11.3.10 ValidInput() [1/2]

```
bool HeightManagement::HeightManager::ValidInput (
            double x,
            double y ) [private]
```

Function to test whether a point exists within the scope of the elected data subset.

Overloaded version of ValidInput() that takes doubles.

Parameters

x	The X value of the coordinate to be tested.
y	The Y value of the coordinate to be tested.

Returns

A bool indicating whether or not the input exists in the subset and is valid.

Definition at line 185 of file height_manager.cpp.

References m_SelectionCorner, m_Size, HeightManagement::HeightManager::heightdata::x, and HeightManagement::HeightManage

11.11.3.11 ValidInput() [2/2]

```
bool HeightManagement::HeightManager::ValidInput (
            int x,
            int y ) [private]
```

Function to test whether a point exists within the scope of the selected data subset.

Parameters

x	the X value of the coordinate to be tested.
y	the Y value of the coordinate to be tested.

Returns

A bool indicating whether or not the input exists in the subset and is valid.

Definition at line 178 of file height_manager.cpp.

References m_Size.

Referenced by GetHeight(), GetHeightAbsolute(), GetVertex(), and GetVertexAbsolute().

11.11.4 Member Data Documentation

11.11.4.1 m_CachedTifName

```
const char* HeightManagement::HeightManager::m_CachedTifName = "../res/Kongsberg.tif"  [private]
```

Definition at line 130 of file height_manager.h.

Referenced by LoadTif(), PopulateVertices(), and UpdateCornerCoords().

11.11.4.2 m_CoordinateSystem

```
const char* HeightManagement::HeightManager::m_CoordinateSystem { "UTM33" }  [private]
```

Definition at line 131 of file height_manager.h.

Referenced by UpdateOrigin().

11.11.4.3 m_LowerRightX

```
long HeightManagement::HeightManager::m_LowerRightX  [private]
```

Definition at line 136 of file height_manager.h.

Referenced by OrigoWithinBounds(), UpdateCornerCoords(), and UpdateOrigin().

11.11.4.4 m_LowerRightY

```
long HeightManagement::HeightManager::m_LowerRightY  [private]
```

Definition at line 137 of file height_manager.h.

Referenced by OrigoWithinBounds(), UpdateCornerCoords(), and UpdateOrigin().

11.11.4.5 m_Origo

```
heightdata HeightManagement::HeightManager::m_Origo { 0, 0, 0 }  [private]
```

Definition at line 139 of file height_manager.h.

Referenced by PopulateVertices(), and UpdateOrigin().

11.11.4.6 m_Resolution

```
int HeightManagement::HeightManager::m_Resolution { 1 }  [private]
```

Definition at line 132 of file height_manager.h.

11.11.4.7 m_SelectionCorner

```
heightdata HeightManagement::HeightManager::m_SelectionCorner  [private]
```

Definition at line 140 of file height_manager.h.

Referenced by GetHeightAbsolute(), GetVertexAbsolute(), PopulateVertices(), and ValidInput().

11.11.4.8 m_Size

```
int HeightManagement::HeightManager::m_Size  [private]
```

Definition at line 133 of file height_manager.h.

Referenced by GetHeight(), GetHeightAbsolute(), GetVertex(), GetVertexAbsolute(), LoadTif(), OrigoWithinBounds(), PopulateVertices(), UpdateOrigin(), and ValidInput().

11.11.4.9 m_UpperLeftX

```
long HeightManagement::HeightManager::m_UpperLeftX [private]
```

Definition at line 134 of file height_manager.h.

Referenced by OrigoWithinBounds(), UpdateCornerCoords(), and UpdateOrigin().

11.11.4.10 m_UpperLeftY

```
long HeightManagement::HeightManager::m_UpperLeftY [private]
```

Definition at line 135 of file height_manager.h.

Referenced by OrigoWithinBounds(), UpdateCornerCoords(), and UpdateOrigin().

11.11.4.11 m_Vertices

```
heightdata* HeightManagement::HeightManager::m_Vertices [private]
```

Definition at line 138 of file height_manager.h.

Referenced by GetHeight(), GetHeightAbsolute(), GetVertex(), GetVertexAbsolute(), PopulateVertices(), and UpdateOrigin().

The documentation for this class was generated from the following files:

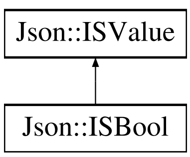
- include/height_management/height_manager.h
- src/height_management/height_manager.cpp

11.12 Json::ISBool Class Reference

Implementation for bools.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISBool:



Public Member Functions

- ISBool (bool &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- bool & value

11.12.1 Detailed Description

Implementation for bools.

Definition at line 121 of file serializer.h.

11.12.2 Constructor & Destructor Documentation

11.12.2.1 ISBool()

```
Json::ISBool::ISBool (
            bool & v ) [inline]
```

Definition at line 126 of file serializer.h.

11.12.3 Member Function Documentation

11.12.3.1 FromDom()

```
void Json::ISBool::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 73 of file serializer.cpp.

References value.

11.12.3.2 ToDom()

```
rapidjson::Value Json::ISBool::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 65 of file serializer.cpp.

References value.

11.12.4 Member Data Documentation

11.12.4.1 value

```
bool& Json::ISBool::value  [private]
```

Definition at line 123 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

- include/core/serializer.h
- src/core/serializer.cpp

11.13 Json::ISConstructors Class Reference

Implemented for future expansion.

```
#include <serializer.h>
```

Public Member Functions

- ISConstructors (const ISConstructors &)=delete
- void operator= (const ISConstructors &)=delete
- int AddConstructor (std::string name, ISValuePtr(∗creator)())
- ISValuePtr GetObject (std::string name)

Static Public Member Functions

- static ISConstructors & GetInstance ()

Private Member Functions

- ISConstructors ()

Private Attributes

- std::map< std::string, Json::ISValuePtr(∗)()> m_TheRegistry

11.13.1 Detailed Description

Implemented for future expansion.

Definition at line 476 of file serializer.h.

11.13.2 Constructor & Destructor Documentation

11.13.2.1 ISConstructors() [1/2]

```
Json::ISConstructors::ISConstructors ( )  [inline], [private]
```

Definition at line 488 of file serializer.h.

11.13.2.2 ISConstructors() [2/2]

```
Json::ISConstructors::ISConstructors (
            const ISConstructors & )  [delete]
```

11.13.3 Member Function Documentation

11.13.3.1 AddConstructor()

```
int Json::ISConstructors::AddConstructor (
            std::string name,
            ISValuePtr(*)() creator )
```

Definition at line 190 of file serializer.cpp.

References m_TheRegistry.

11.13.3.2 GetInstance()

```
static ISConstructors & Json::ISConstructors::GetInstance ( )  [inline], [static]
```

Definition at line 480 of file serializer.h.

11.13.3.3 GetObject()

```
ISValuePtr Json::ISConstructors::GetObject (
            std::string name )
```

Definition at line 198 of file serializer.cpp.

References m_TheRegistry.

11.13.3.4 operator=()

```
void Json::ISConstructors::operator= (
            const ISConstructors & )  [delete]
```

11.13.4 Member Data Documentation

11.13.4.1 m_TheRegistry

```
std::map<std::string, Json::ISValuePtr (*)()> Json::ISConstructors::m_TheRegistry  [private]
```

Definition at line 490 of file serializer.h.

Referenced by AddConstructor(), and GetObject().

The documentation for this class was generated from the following files:

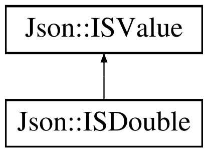
- include/core/serializer.h
- src/core/serializer.cpp

11.14 Json::ISDouble Class Reference

Implementation for doubles.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISDouble:



Public Member Functions

- ISDouble (double &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

    ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

    FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

    GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

    For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

    For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

    ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

    FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- double & value

11.14.1 Detailed Description

Implementation for doubles.

Definition at line 108 of file serializer.h.

11.14.2 Constructor & Destructor Documentation

11.14.2.1 ISDouble()

```
Json::ISDouble::ISDouble (
            double & v )  [inline]
```

Definition at line 113 of file serializer.h.

11.14.3 Member Function Documentation

11.14.3.1 FromDom()

```
void Json::ISDouble::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 58 of file serializer.cpp.

References value.

11.14.3.2 ToDom()

```
rapidjson::Value Json::ISDouble::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 48 of file serializer.cpp.

References debug, and value.

11.14.4 Member Data Documentation

11.14.4.1 value

```
double& Json::ISDouble::value [private]
```

Definition at line 110 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

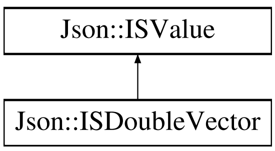
- include/core/serializer.h
- src/core/serializer.cpp

11.15 Json::ISDoubleVector Class Reference

Implementation for a vector with doubles.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISDoubleVector:



Public Member Functions

- ISDoubleVector (ISDV &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< double > & value

11.15.1 Detailed Description

Implementation for a vector with doubles.

Definition at line 454 of file serializer.h.

11.15.2 Constructor & Destructor Documentation

11.15.2.1 ISDoubleVector()

```
Json::ISDoubleVector::ISDoubleVector (
            ISDV & v )  [inline]
```

Definition at line 459 of file serializer.h.

11.15.3 Member Function Documentation

11.15.3.1 FromDom()

```
void Json::ISDoubleVector::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 181 of file serializer.cpp.

References value.

11.15.3.2 ToDom()

```
rapidjson::Value Json::ISDoubleVector::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 168 of file serializer.cpp.

References value.

11.15.4 Member Data Documentation

11.15.4.1 value

```
std::vector<double>& Json::ISDoubleVector::value  [private]
```

Definition at line 456 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

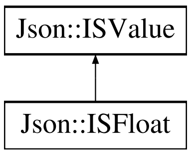
- include/core/serializer.h
- src/core/serializer.cpp

11.16 Json::ISFloat Class Reference

Implementation for floats.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISFloat:



Public Member Functions

- ISFloat (float &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- float & value

11.16.1 Detailed Description

Implementation for floats.

Definition at line 95 of file serializer.h.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 ISFloat()

```
Json::ISFloat::ISFloat (
            float & v )  [inline]
```

Definition at line 100 of file serializer.h.

11.16.3 Member Function Documentation

11.16.3.1 FromDom()

```
void Json::ISFloat::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 41 of file serializer.cpp.

References value.

11.16.3.2 ToDom()

```
rapidjson::Value Json::ISFloat::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 33 of file serializer.cpp.

References value.

11.16.4 Member Data Documentation

11.16.4.1 value

`float& Json::ISFloat::value [private]`

Definition at line 97 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

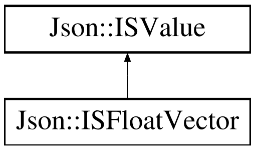
- include/core/serializer.h
- src/core/serializer.cpp

11.17 Json::ISFloatVector Class Reference

Implementation for a vector with floats.

`#include <serializer.h>`

Inheritance diagram for Json::ISFloatVector:



Public Member Functions

- ISFloatVector (ISFV &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< float > & value

11.17.1 Detailed Description

Implementation for a vector with floats.

Definition at line 439 of file serializer.h.

11.17.2 Constructor & Destructor Documentation

11.17.2.1 ISFloatVector()

```
Json::ISFloatVector::ISFloatVector (
            ISFV & v )  [inline]
```

Definition at line 444 of file serializer.h.

11.17.3 Member Function Documentation

11.17.3.1 FromDom()

```
void Json::ISFloatVector::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 159 of file serializer.cpp.

References value.

11.17.3.2 ToDom()

```
rapidjson::Value Json::ISFloatVector::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 146 of file serializer.cpp.

References value.

11.17.4 Member Data Documentation

11.17.4.1 value

```
std::vector<float>& Json::ISFloatVector::value [private]
```

Definition at line 441 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

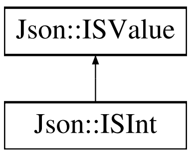
- include/core/serializer.h
- src/core/serializer.cpp

11.18 Json::ISInt Class Reference

Implementation for integers.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISInt:



Public Member Functions

- ISInt (int &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- int & value

11.18.1 Detailed Description

Implementation for integers.

Definition at line 82 of file serializer.h.

11.18.2 Constructor & Destructor Documentation

11.18.2.1 ISInt()

```
Json::ISInt::ISInt (
            int & v ) [inline]
```

Definition at line 87 of file serializer.h.

11.18.3 Member Function Documentation

11.18.3.1 FromDom()

```
void Json::ISInt::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d ) [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 26 of file serializer.cpp.

References value.

11.18.3.2 ToDom()

```
rapidjson::Value Json::ISInt::ToDom (
            rapidjson::Document & d ) [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 18 of file serializer.cpp.

References value.

11.18.4 Member Data Documentation

11.18.4.1 value

```
int& Json::ISInt::value [private]
```

Definition at line 84 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

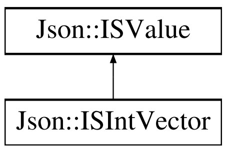
- include/core/serializer.h
- src/core/serializer.cpp

11.19 Json::ISIntVector Class Reference

Implementation for a vector with integers.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISIntVector:



Public Member Functions

- ISIntVector (ISIV &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< int > & value

11.19.1 Detailed Description

Implementation for a vector with integers.

Definition at line 424 of file serializer.h.

11.19.2 Constructor & Destructor Documentation

11.19.2.1 ISIntVector()

```
Json::ISIntVector::ISIntVector (
            ISIV & v )  [inline]
```

Definition at line 429 of file serializer.h.

11.19.3 Member Function Documentation

11.19.3.1 FromDom()

```
void Json::ISIntVector::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 137 of file serializer.cpp.

References value.

11.19.3.2 ToDom()

```
rapidjson::Value Json::ISIntVector::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 124 of file serializer.cpp.

References value.

11.19.4 Member Data Documentation

11.19.4.1 value

```
std::vector<int>& Json::ISIntVector::value [private]
```

Definition at line 426 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

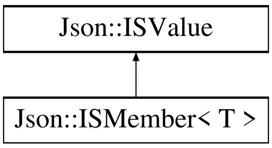
- include/core/serializer.h
- src/core/serializer.cpp

11.20 Json::ISMember< T > Class Template Reference

Implementation for Members.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISMember< T >:



Public Member Functions

- ISMember (T &v)
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.
- void CreateObject ()

  For future expansion.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- T & value

11.20.1 Detailed Description

template<typename T>
class Json::ISMember< T >

Implementation for Members.

Definition at line 334 of file serializer.h.

11.20.2 Constructor & Destructor Documentation

11.20.2.1 ISMember()

```
template<typename T >
Json::ISMember< T >::ISMember (
            T & v )  [inline]
```

Definition at line 339 of file serializer.h.

11.20.3 Member Function Documentation

11.20.3.1 CreateObject()

```
template<typename T >
void Json::ISMember< T >::CreateObject  [virtual]
```

For future expansion.

Reimplemented from Json::ISValue.

Definition at line 371 of file serializer.h.

11.20.3.2 FromDom()

```
template<typename T >
void Json::ISMember< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 355 of file serializer.h.

11.20.3.3 GetName()

```
template<typename T >
rapidjson::Value Json::ISMember< T >::GetName (
            rapidjson::Document & d )  [virtual]
```

For future expansion.

Typeid is mostly implemented for future expansion, but it helps with making the JSON file more readable for humans.

Reimplemented from Json::ISValue.

Definition at line 362 of file serializer.h.

11.20.3.4 ToDom()

```
template<typename T >
rapidjson::Value Json::ISMember< T >::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 348 of file serializer.h.

11.20.4 Member Data Documentation

11.20.4.1 value

```
template<typename T >
T& Json::ISMember< T >::value [private]
```

Definition at line 336 of file serializer.h.

The documentation for this class was generated from the following file:

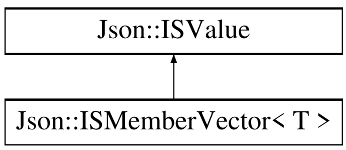
- include/core/serializer.h

11.21 Json::ISMemberVector< T > Class Template Reference

Implementation for a vector with members.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISMemberVector< T >:



Public Member Functions

- ISMemberVector (const ISMemberVector< T > &)
- ISMemberVector (std::vector< T > &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< T > & value

11.21.1 Detailed Description

template<typename T>
class Json::ISMemberVector< T >

Implementation for a vector with members.

Definition at line 380 of file serializer.h.

11.21.2 Constructor & Destructor Documentation

11.21.2.1 ISMemberVector() [1/2]

```
template<typename T >
Json::ISMemberVector< T >::ISMemberVector (
            const ISMemberVector< T > & ) [inline]
```

Definition at line 385 of file serializer.h.

11.21.2.2 ISMemberVector() [2/2]

```
template<typename T >
Json::ISMemberVector< T >::ISMemberVector (
            std::vector< T > & v ) [inline]
```

Definition at line 387 of file serializer.h.

11.21.3 Member Function Documentation

11.21.3.1 FromDom()

```
template<typename T >
void Json::ISMemberVector< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d ) [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 409 of file serializer.h.

11.21.3.2 ToDom()

```
template<typename T >
rapidjson::Value Json::ISMemberVector< T >::ToDom (
            rapidjson::Document & d ) [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 394 of file serializer.h.

References debug.

11.21.4 Member Data Documentation

11.21.4.1 value

```
template<typename T >
std::vector<T>& Json::ISMemberVector< T >::value [private]
```

Definition at line 382 of file serializer.h.

The documentation for this class was generated from the following file:

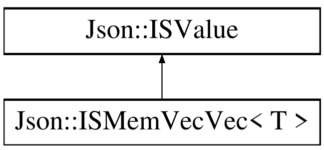
- include/core/serializer.h

11.22 Json::ISMemVecVec< T > Class Template Reference

Implementation for a vector with vectors with members.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISMemVecVec< T >:



Public Member Functions

- ISMemVecVec (std::vector< std::vector< T > > &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< std::vector< T > > & value

11.22.1 Detailed Description

template<typename T>
class Json::ISMemVecVec< T >

Implementation for a vector with vectors with members.

Definition at line 287 of file serializer.h.

11.22.2 Constructor & Destructor Documentation

11.22.2.1 ISMemVecVec()

```
template<typename T >
Json::ISMemVecVec< T >::ISMemVecVec (
            std::vector< std::vector< T > > & v )  [inline]
```

Definition at line 292 of file serializer.h.

11.22.3 Member Function Documentation

11.22.3.1 FromDom()

```
template<typename T >
void Json::ISMemVecVec< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 317 of file serializer.h.

11.22.3.2 ToDom()

```
template<typename T >
rapidjson::Value Json::ISMemVecVec< T >::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 299 of file serializer.h.

11.22.4 Member Data Documentation

11.22.4.1 value

```
template<typename T >
std::vector<std::vector<T> >& Json::ISMemVecVec< T >::value  [private]
```

Definition at line 289 of file serializer.h.

The documentation for this class was generated from the following file:

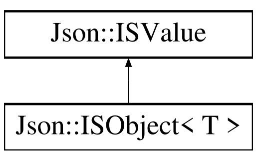
- include/core/serializer.h

11.23 Json::ISObject< T > Class Template Reference

Implementation for objects.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISObject< T >:



Public Member Functions

- ISObject (std::shared_ptr< T > &v)
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.
- void CreateObject ()

  For future expansion.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::shared_ptr< T > & value

11.23.1 Detailed Description

template<typename T>
class Json::ISObject< T >

Implementation for objects.

Definition at line 148 of file serializer.h.

11.23.2 Constructor & Destructor Documentation

11.23.2.1 ISObject()

```
template<typename T >
Json::ISObject< T >::ISObject (
            std::shared_ptr< T > & v )  [inline]
```

Definition at line 153 of file serializer.h.

11.23.3 Member Function Documentation

11.23.3.1 CreateObject()

```
template<typename T >
void Json::ISObject< T >::CreateObject  [virtual]
```

For future expansion.

Reimplemented from Json::ISValue.

Definition at line 191 of file serializer.h.

11.23.3.2 FromDom()

```
template<typename T >
void Json::ISObject< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 172 of file serializer.h.

11.23.3.3 GetName()

```
template<typename T >
rapidjson::Value Json::ISObject< T >::GetName (
            rapidjson::Document & d )  [virtual]
```

For future expansion.

Typeid is mostly implemented for future expansion, but it helps with making the JSON file more readable for humans.

Reimplemented from Json::ISValue.

Definition at line 182 of file serializer.h.

11.23.3.4 ToDom()

```
template<typename T >
rapidjson::Value Json::ISObject< T >::ToDom (
            rapidjson::Document & d ) [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 162 of file serializer.h.

11.23.4 Member Data Documentation

11.23.4.1 value

```
template<typename T >
std::shared_ptr<T>& Json::ISObject< T >::value [private]
```

Definition at line 150 of file serializer.h.

The documentation for this class was generated from the following file:

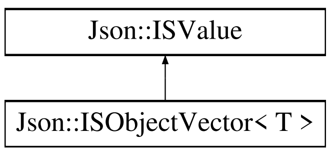
- include/core/serializer.h

11.24 Json::ISObjectVector< T > Class Template Reference

Implementation for a vector with objects.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISObjectVector< T >:



Public Member Functions

- ISObjectVector (std::vector< std::shared_ptr< T > > &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)
  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)
  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< std::shared_ptr< T > > & value

11.24.1 Detailed Description

template<typename T>
class Json::ISObjectVector< T >

Implementation for a vector with objects.

Definition at line 200 of file serializer.h.

11.24.2 Constructor & Destructor Documentation

11.24.2.1 ISObjectVector()

```
template<typename T >
Json::ISObjectVector< T >::ISObjectVector (
            std::vector< std::shared_ptr< T > > & v )  [inline]
```

Definition at line 205 of file serializer.h.

11.24.3 Member Function Documentation

11.24.3.1 FromDom()

```
template<typename T >
void Json::ISObjectVector< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d ) [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 225 of file serializer.h.

11.24.3.2 ToDom()

```
template<typename T >
rapidjson::Value Json::ISObjectVector< T >::ToDom (
            rapidjson::Document & d ) [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 212 of file serializer.h.

11.24.4 Member Data Documentation

11.24.4.1 value

```
template<typename T >
std::vector<std::shared_ptr<T> >& Json::ISObjectVector< T >::value [private]
```

Definition at line 202 of file serializer.h.

The documentation for this class was generated from the following file:

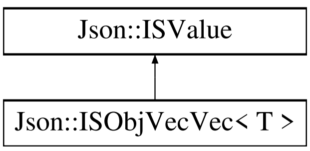
- include/core/serializer.h

11.25 Json::ISObjVecVec< T > Class Template Reference

Implementation for a vector with vectors with objects.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISObjVecVec< T >:



Public Member Functions

- ISObjVecVec (std::vector< std::vector< std::shared_ptr< T > > > &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::vector< std::vector< std::shared_ptr< T > > > & value

11.25.1 Detailed Description

template<typename T>
class Json::ISObjVecVec< T >

Implementation for a vector with vectors with objects.

Definition at line 239 of file serializer.h.

11.25.2 Constructor & Destructor Documentation

11.25.2.1 ISObjVecVec()

```
template<typename T >
Json::ISObjVecVec< T >::ISObjVecVec (
            std::vector< std::vector< std::shared_ptr< T > > > & v )  [inline]
```

Definition at line 244 of file serializer.h.

11.25.3 Member Function Documentation

11.25.3.1 FromDom()

```
template<typename T >
void Json::ISObjVecVec< T >::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 270 of file serializer.h.

11.25.3.2 ToDom()

```
template<typename T >
rapidjson::Value Json::ISObjVecVec< T >::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 252 of file serializer.h.

11.25.4 Member Data Documentation

11.25.4.1 value

```
template<typename T >
std::vector<std::vector<std::shared_ptr<T> > >& Json::ISObjVecVec< T >::value  [private]
```

Definition at line 241 of file serializer.h.

The documentation for this class was generated from the following file:

- include/core/serializer.h

11.26 Json::ISProperty Struct Reference

Serializing and deserializing (persistent values) requires recflection which is a way for the programmer to ensure that the data you serialize will get back to the place you want it to be when you deserialize it later.

```
#include <serializer.h>
```

Public Attributes

- std::string name
- ISValuePtr value

11.26.1 Detailed Description

Serializing and deserializing (persistent values) requires recflection which is a way for the programmer to ensure that the data you serialize will get back to the place you want it to be when you deserialize it later.

As this is not supported by C++ this is implemented by the ISProperty structure with the ISValue helper classes. The ISValue keeps the references to the actual values in the application. The ISProperty is the collection of all the application data.

Definition at line 26 of file serializer.h.

11.26.2 Member Data Documentation

11.26.2.1 name

```
std::string Json::ISProperty::name
```

Definition at line 28 of file serializer.h.

11.26.2.2 value

```
ISValuePtr Json::ISProperty::value
```

Definition at line 29 of file serializer.h.

The documentation for this struct was generated from the following file:

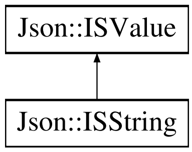
- include/core/serializer.h

11.27 Json::ISString Class Reference

Implementation for strings.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISString:



Public Member Functions

- ISString (std::string &v)
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Public Member Functions inherited from Json::ISValue

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Private Attributes

- std::string & value

11.27.1 Detailed Description

Implementation for strings.

Definition at line 134 of file serializer.h.

11.27.2 Constructor & Destructor Documentation

11.27.2.1 ISString()

```
Json::ISString::ISString (
            std::string & v )  [inline]
```

Definition at line 139 of file serializer.h.

11.27.3 Member Function Documentation

11.27.3.1 FromDom()

```
void Json::ISString::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented from Json::ISValue.

Definition at line 88 of file serializer.cpp.

References value.

11.27.3.2 ToDom()

```
rapidjson::Value Json::ISString::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented from Json::ISValue.

Definition at line 80 of file serializer.cpp.

References value.

11.27.4 Member Data Documentation

11.27.4.1 value

```
std::string& Json::ISString::value  [private]
```

Definition at line 136 of file serializer.h.

Referenced by FromDom(), and ToDom().

The documentation for this class was generated from the following files:

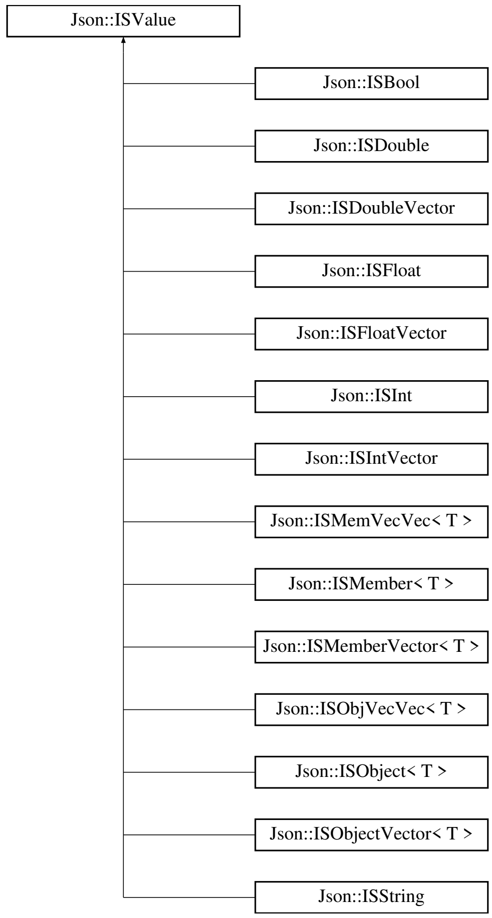
- include/core/serializer.h
- src/core/serializer.cpp

11.28 Json::ISValue Class Reference

Rflection is made possible by the help of the ISValue class and the type classes.

```
#include <serializer.h>
```

Inheritance diagram for Json::ISValue:



Public Member Functions

- virtual ISProperties GetProperty ()

  GetProperty enables the serializer to deal with composite type like objects and members.
- virtual void CreateObject ()

  For future expansion.
- virtual rapidjson::Value GetName (rapidjson::Document &d)

  For future expansion.
- virtual rapidjson::Value ToDom (rapidjson::Document &d)

  ToDom is the function that enables the serializer to take data from the application to the JSON file.
- virtual void FromDom (rapidjson::Value &v, rapidjson::Document &d)

  FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

11.28.1 Detailed Description

Rflection is made possible by the help of the ISValue class and the type classes.

Each type needs their own implementation for reflection to work. At the moment only JSON is supported by this library. Making the library work for other format than JSON would require implementing each type again for the new format by in theory would not impact the application programmers at all

Definition at line 41 of file serializer.h.

11.28.2 Member Function Documentation

11.28.2.1 CreateObject()

```
virtual void Json::ISValue::CreateObject ( )  [inline], [virtual]
```

For future expansion.

Reimplemented in Json::ISObject< T >, and Json::ISMember< T >.

Definition at line 55 of file serializer.h.

11.28.2.2 FromDom()

```
void Json::ISValue::FromDom (
            rapidjson::Value & v,
            rapidjson::Document & d )  [virtual]
```

FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.

Reimplemented in Json::ISInt, Json::ISFloat, Json::ISDouble, Json::ISBool, Json::ISString, Json::ISObject< T >, Json::ISObjectVector< T >, Json::ISObjVecVec< T >, Json::ISMemVecVec< T >, Json::ISMember< T >, Json::ISMemberVector< T >, Json::ISIntVector, Json::ISFloatVector, and Json::ISDoubleVector.

Definition at line 113 of file serializer.cpp.

References GetProperty().

11.28.2.3 GetName()

```
virtual rapidjson::Value Json::ISValue::GetName (
            rapidjson::Document & d )  [inline], [virtual]
```

For future expansion.

Typeid is mostly implemented for future expansion, but it helps with making the JSON file more readable for humans.

Reimplemented in Json::ISObject< T >, and Json::ISMember< T >.

Definition at line 60 of file serializer.h.

Referenced by Json::serialize(), and ToDom().

11.28.2.4 GetProperty()

```
virtual ISProperties Json::ISValue::GetProperty ( )  [inline], [virtual]
```

GetProperty enables the serializer to deal with composite type like objects and members.

Definition at line 48 of file serializer.h.

Referenced by Json::deserialize(), FromDom(), Json::serialize(), and ToDom().

11.28.2.5 ToDom()

```
rapidjson::Value Json::ISValue::ToDom (
            rapidjson::Document & d )  [virtual]
```

ToDom is the function that enables the serializer to take data from the application to the JSON file.

Reimplemented in Json::ISInt, Json::ISFloat, Json::ISDouble, Json::ISBool, Json::ISString, Json::ISObject< T >, Json::ISObjectVector< T >, Json::ISObjVecVec< T >, Json::ISMemVecVec< T >, Json::ISMember< T >, Json::ISMemberVector< T >, Json::ISIntVector, Json::ISFloatVector, and Json::ISDoubleVector.

Definition at line 95 of file serializer.cpp.

References GetName(), and GetProperty().

The documentation for this class was generated from the following files:

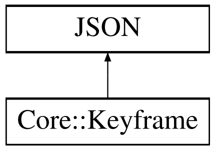
- include/core/serializer.h
- src/core/serializer.cpp

11.29 Core::Keyframe Struct Reference

A structure representing an agent's position in cartesian space at a given point in time.

```
#include <types.h>
```

Inheritance diagram for Core::Keyframe:



Public Member Functions

- Keyframe ()
- Keyframe (int agentId, float timeStamp, CartesianCoordinate position)
- JSONSTART JSONINT (AgentId)
- JSONSTART JSONFLOAT (TimeStamp)

Public Attributes

- int AgentId
- float TimeStamp
- CartesianCoordinate Position

11.29.1 Detailed Description

A structure representing an agent's position in cartesian space at a given point in time.

Definition at line 68 of file types.h.

11.29.2 Constructor & Destructor Documentation

11.29.2.1 Keyframe() [1/2]

```
Core::Keyframe::Keyframe ( )   [inline]
```

Definition at line 70 of file types.h.

11.29.2.2 Keyframe() [2/2]

```
Core::Keyframe::Keyframe (
            int agentId,
            float timeStamp,
            CartesianCoordinate position ) [inline]
```

Definition at line 72 of file types.h.

11.29.3 Member Function Documentation

11.29.3.1 JSONFLOAT()

```
JSONSTART Core::Keyframe::JSONFLOAT (
            TimeStamp )
```

11.29.3.2 JSONINT()

```
JSONSTART Core::Keyframe::JSONINT (
            AgentId )
```

11.29.4 Member Data Documentation

11.29.4.1 AgentId

```
int Core::Keyframe::AgentId
```

Definition at line 76 of file types.h.

Referenced by KeyframeManagement::KeyframeManager::AddKeyframe(), Routemaker::Routemaker::MakeRoute(), and KeyframeManagement::KeyframeManager::RemoveKeyframe().

11.29.4.2 Position

```
CartesianCoordinate Core::Keyframe::Position
```

Definition at line 78 of file types.h.

Referenced by KeyframeManagement::KeyframeManager::AddKeyframe(), Routemaker::Routemaker::MakeRoute(), and KeyframeManagement::KeyframeManager::RemoveKeyframe().

11.29.4.3 TimeStamp

```
float Core::Keyframe::TimeStamp
```

Definition at line 77 of file types.h.

Referenced by KeyframeManagement::KeyframeManager::AddKeyframe(), CompileScenario::Scenario::Compile(), Routemaker::Routemaker::MakeRoute(), Gui::Timeline::mouseReleaseEvent(), and KeyframeManagement::KeyframeManager::Rem

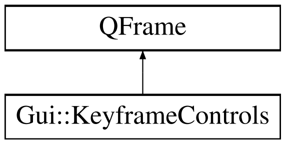
The documentation for this struct was generated from the following file:

- include/core/types.h

11.30 Gui::KeyframeControls Class Reference

```
#include <keyframe_controls.h>
```

Inheritance diagram for Gui::KeyframeControls:



Signals

- void DeleteSelectedKeyframes ()

Public Member Functions

- KeyframeControls (QWidget ∗parent=nullptr)

Private Attributes

- KeyframeList ∗ m_KeyframeList
- QPushButton ∗ m_DeleteKeyframesButton
- QGridLayout ∗ m_Layout

11.30.1 Detailed Description

Definition at line 11 of file keyframe_controls.h.

11.30.2 Constructor & Destructor Documentation

11.30.2.1 KeyframeControls()

```
Gui::KeyframeControls::KeyframeControls (
            QWidget * parent = nullptr ) [explicit]
```

Definition at line 8 of file keyframe_controls.cpp.

References DeleteSelectedKeyframes(), m_DeleteKeyframesButton, m_KeyframeList, and m_Layout.

11.30.3 Member Function Documentation

11.30.3.1 DeleteSelectedKeyframes

```
void Gui::KeyframeControls::DeleteSelectedKeyframes ( ) [signal]
```

Referenced by KeyframeControls().

11.30.4 Member Data Documentation

11.30.4.1 m_DeleteKeyframesButton

```
QPushButton* Gui::KeyframeControls::m_DeleteKeyframesButton [private]
```

Definition at line 22 of file keyframe_controls.h.

Referenced by KeyframeControls().

11.30.4.2 m_KeyframeList

```
KeyframeList* Gui::KeyframeControls::m_KeyframeList [private]
```

Definition at line 21 of file keyframe_controls.h.

Referenced by KeyframeControls().

11.30.4.3 m_Layout

```
QGridLayout* Gui::KeyframeControls::m_Layout  [private]
```

Definition at line 24 of file keyframe_controls.h.

Referenced by KeyframeControls().

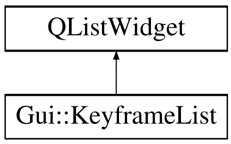
The documentation for this class was generated from the following files:

- include/gui/keyframe_controls.h
- src/gui/keyframe_controls.cpp

11.31 Gui::KeyframeList Class Reference

```
#include <keyframe_list.h>
```

Inheritance diagram for Gui::KeyframeList:



Public Slots

- void Update ()
- void DeleteSelected ()

Public Member Functions

- KeyframeList (QWidget ∗parent=nullptr)

Private Attributes

- QVBoxLayout ∗ m_Layout

11.31.1 Detailed Description

Definition at line 9 of file keyframe_list.h.

11.31.2 Constructor & Destructor Documentation

11.31.2.1 KeyframeList()

```
Gui::KeyframeList::KeyframeList (
            QWidget * parent = nullptr )
```

Definition at line 11 of file keyframe_list.cpp.

References Update().

11.31.3 Member Function Documentation

11.31.3.1 DeleteSelected

```
void Gui::KeyframeList::DeleteSelected ( )  [slot]
```

Definition at line 40 of file keyframe_list.cpp.

References KeyframeManagement::KeyframeManager::Instance(), KeyframeManagement::KeyframeManager::RemoveKeyframe(), and Update().

11.31.3.2 Update

```
void Gui::KeyframeList::Update ( )  [slot]
```

Definition at line 19 of file keyframe_list.cpp.

References KeyframeManagement::KeyframeManager::GetKeyframes(), and KeyframeManagement::KeyframeManager::Instance().

Referenced by DeleteSelected(), and KeyframeList().

11.31.4 Member Data Documentation

11.31.4.1 m_Layout

```
QVBoxLayout* Gui::KeyframeList::m_Layout  [private]
```

Definition at line 21 of file keyframe_list.h.

The documentation for this class was generated from the following files:

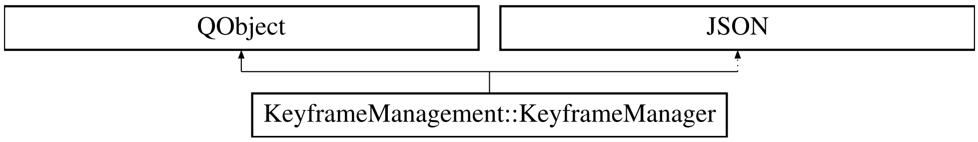
- include/gui/keyframe_list.h
- src/gui/keyframe_list.cpp

11.32 KeyframeManagement::KeyframeManager Class Reference

This is the class that manages keyframes.

```
#include <keyframe_manager.h>
```

Inheritance diagram for KeyframeManagement::KeyframeManager:



Signals

- void KeyframeAdded ()

Public Member Functions

- void AddKeyframe (int agentId, float timeStamp, float x, float y, float z)

  Adds a keyframe to the keyframe list using x, y, and z coordinates.
- void AddKeyframe (int agentId, float timeStamp, Core::CartesianCoordinate position)

  Adds a keyframe to the keyframe list using a CartesianCoordinate.
- void AddKeyframe (Core::Keyframe &keyframe)

  Adds a keyframe object to the keyframe list.
- void RemoveKeyframe (const Core::Keyframe &keyframe)

  Removes a keyframe from the keyframe list.
- void DebugDump (void) const

  Dumps keyframe information to the console for debugging purposes.
- std::vector< Core::Keyframe > & GetKeyframes ()

  Returns a reference to the list of keyframes.

Static Public Member Functions

- static KeyframeManager & Instance ()

  Returns the singleton instance of the KeyframeManager.

Private Member Functions

- KeyframeManager ()

  Private constructor for singleton pattern.
- ∼KeyframeManager ()

  Private destructor for singleton pattern.
- KeyframeManager (const KeyframeManager &)=delete
- KeyframeManager & operator= (const KeyframeManager &)=delete

Private Attributes

- std::vector< Core::Keyframe > m_Keyframes

11.32.1 Detailed Description

This is the class that manages keyframes.

Definition at line 14 of file keyframe_manager.h.

11.32.2 Constructor & Destructor Documentation

11.32.2.1 KeyframeManager() [1/2]

```
KeyframeManagement::KeyframeManager::KeyframeManager ( )  [inline], [private]
```

Private constructor for singleton pattern.

Definition at line 84 of file keyframe_manager.h.

11.32.2.2 ∼KeyframeManager()

```
KeyframeManagement::KeyframeManager::∼KeyframeManager ( )  [inline], [private]
```

Private destructor for singleton pattern.

Definition at line 86 of file keyframe_manager.h.

11.32.2.3 KeyframeManager() [2/2]

```
KeyframeManagement::KeyframeManager::KeyframeManager (
            const KeyframeManager &  )  [private], [delete]
```

11.32.3 Member Function Documentation

11.32.3.1 AddKeyframe() [1/3]

```
void KeyframeManagement::KeyframeManager::AddKeyframe (
            Core::Keyframe & keyframe )
```

Adds a keyframe object to the keyframe list.

Parameters

keyframe	The keyframe object to add

Definition at line 30 of file keyframe_manager.cpp.

References Core::Keyframe::AgentId, KeyframeAdded(), m_Keyframes, Core::Keyframe::Position, and Core::Keyframe::TimeStamp.

11.32.3.2 AddKeyframe() [2/3]

```
void KeyframeManagement::KeyframeManager::AddKeyframe (
            int agentId,
            float timeStamp,
            Core::CartesianCoordinate position )
```

Adds a keyframe to the keyframe list using a CartesianCoordinate.

Parameters

agentId	The ID of the agent
timeStamp	The timestamp of the keyframe
position	The CartesianCoordinate representing the position

Definition at line 18 of file keyframe_manager.cpp.

References AddKeyframe().

11.32.3.3 AddKeyframe() [3/3]

```
void KeyframeManagement::KeyframeManager::AddKeyframe (
            int agentId,
            float timeStamp,
            float x,
            float y,
            float z )
```

Adds a keyframe to the keyframe list using x, y, and z coordinates.

Parameters

agentId	The ID of the agent
timeStamp	The timestamp of the keyframe
x	The x coordinate
y	The y coordinate
z	The z coordinate

Definition at line 9 of file keyframe_manager.cpp.

References AddKeyframe().

Referenced by AddKeyframe(), and Gui::MapViewer::mousePressEvent().

11.32.3.4 DebugDump()

```
void KeyframeManagement::KeyframeManager::DebugDump (
            void ) const
```

Dumps keyframe information to the console for debugging purposes.

Definition at line 65 of file keyframe_manager.cpp.

References m_Keyframes.

11.32.3.5 GetKeyframes()

```
std::vector< Core::Keyframe > & KeyframeManagement::KeyframeManager::GetKeyframes ( ) [inline]
```

Returns a reference to the list of keyframes.

Returns

A reference to the list of keyframes

Definition at line 75 of file keyframe_manager.h.

References m_Keyframes.

Referenced by CompileScenario::Scenario::Compile(), Gui::MapViewer::DrawKeyframes(), Gui::Timeline::mouseReleaseEvent(), Gui::Timeline::paintEvent(), and Gui::KeyframeList::Update().

11.32.3.6 Instance()

```
static KeyframeManager & KeyframeManagement::KeyframeManager::Instance ( ) [inline], [static]
```

Returns the singleton instance of the KeyframeManager.

Returns

A reference to the singleton instance of the KeyframeManager

Definition at line 25 of file keyframe_manager.h.

Referenced by Gui::MainWindow::ConnectSlotsAndSignals(), Gui::KeyframeList::DeleteSelected(), Gui::MapViewer::DrawKeyframes(), Gui::MapViewer::mousePressEvent(), Gui::Timeline::mouseReleaseEvent(), Gui::Timeline::paintEvent(), Gui::Timeline::Timeline(), and Gui::KeyframeList::Update().

11.32.3.7 KeyframeAdded

```
void KeyframeManagement::KeyframeManager::KeyframeAdded ( ) [signal]
```

Referenced by AddKeyframe().

11.32.3.8 operator=()

```
KeyframeManager & KeyframeManagement::KeyframeManager::operator= (
            const KeyframeManager & ) [private], [delete]
```

11.32.3.9 RemoveKeyframe()

```
void KeyframeManagement::KeyframeManager::RemoveKeyframe (
            const Core::Keyframe & keyframe )
```

Removes a keyframe from the keyframe list.

Parameters

keyframe	The keyframe to remove

Definition at line 50 of file keyframe_manager.cpp.

References Core::Keyframe::AgentId, m_Keyframes, Core::Keyframe::Position, Core::Keyframe::TimeStamp, Core::CartesianCoordinate::X, Core::CartesianCoordinate::Y, and Core::CartesianCoordinate::Z.

Referenced by Gui::KeyframeList::DeleteSelected(), and Gui::Timeline::mouseReleaseEvent().

11.32.4 Member Data Documentation

11.32.4.1 m_Keyframes

```
std::vector<Core::Keyframe> KeyframeManagement::KeyframeManager::m_Keyframes [private]
```

Definition at line 91 of file keyframe_manager.h.

Referenced by AddKeyframe(), DebugDump(), GetKeyframes(), and RemoveKeyframe().

The documentation for this class was generated from the following files:

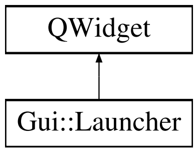
- include/keyframe_management/keyframe_manager.h
- src/keyframe_management/keyframe_manager.cpp

11.33 Gui::Launcher Class Reference

The launcher widget used to launch scenarios.

```
#include <launcher.h>
```

Inheritance diagram for Gui::Launcher:



Public Member Functions

- Launcher (QWidget ∗parent=nullptr)

    Constructs the launcher widget.
- ∼Launcher ()

    Destructs the launcher widget.

Private Attributes

- QVBoxLayout ∗ m_Layout

    The layout of the launcher widget.

11.33.1 Detailed Description

The launcher widget used to launch scenarios.

Contains the graphical functionality to launch scenarios.

Definition at line 11 of file launcher.h.

11.33.2 Constructor & Destructor Documentation

11.33.2.1 Launcher()

```
Gui::Launcher::Launcher (
            QWidget * parent = nullptr )
```

Constructs the launcher widget.

Parameters

parent	The parent of the launcher widget.

Definition at line 7 of file launcher.cpp.

References m_Layout.

11.33.2.2 ∼Launcher()

```
Gui::Launcher::∼Launcher ( )
```

Destructs the launcher widget.

Definition at line 17 of file launcher.cpp.

11.33.3 Member Data Documentation

11.33.3.1 m_Layout

```
QVBoxLayout∗ Gui::Launcher::m_Layout [private]
```

The layout of the launcher widget.

Definition at line 23 of file launcher.h.

Referenced by Launcher().

The documentation for this class was generated from the following files:

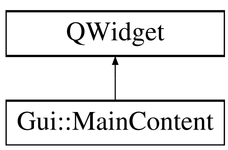
- include/gui/launcher.h
- src/gui/launcher.cpp

11.34 Gui::MainContent Class Reference

The main content of the main window.

```
#include <main_content.h>
```

Inheritance diagram for Gui::MainContent:



Public Member Functions

- MainContent (QWidget ∗parent=nullptr)

    Constructs the main content widget.

Private Attributes

- QGridLayout ∗ m_Layout

    The layout of the main content.
- Sidebar ∗ m_Sidebar

    The sidebar of the main content.
- TabWidget ∗ m_TabWidget

    The tab widget of the main content.

11.34.1 Detailed Description

The main content of the main window.

The main content of the main window essentially contains everything except the menu bar. It exists as a separate class to make the main window class more concise.

Definition at line 16 of file main_content.h.

11.34.2 Constructor & Destructor Documentation

11.34.2.1 MainContent()

```
Gui::MainContent::MainContent (
            QWidget * parent = nullptr )
```

Constructs the main content widget.

Parameters

parent	The parent widget of the main content.

Definition at line 10 of file main_content.cpp.

References m_Layout, m_Sidebar, and m_TabWidget.

11.34.3 Member Data Documentation

11.34.3.1 m_Layout

```
QGridLayout* Gui::MainContent::m_Layout [private]
```

The layout of the main content.

The main content uses a grid layout to easily be able to cover the available space in the window.

Definition at line 29 of file main_content.h.

Referenced by MainContent().

11.34.3.2 m_Sidebar

```
Sidebar* Gui::MainContent::m_Sidebar [private]
```

The sidebar of the main content.

The sidebar of the main content exists to provide the user access to tools related to the active tab in the tab widget.

Definition at line 35 of file main_content.h.

Referenced by MainContent().

11.34.3.3 m_TabWidget

```
TabWidget* Gui::MainContent::m_TabWidget [private]
```

The tab widget of the main content.

This widget is responsible for containing the core functionality of Hivemind; planning, simulating and launching. They are separated in their own tabs as a user should only need to access one of these at any point in time.

Definition at line 43 of file main_content.h.

Referenced by MainContent().

The documentation for this class was generated from the following files:

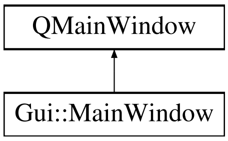
- include/gui/main_content.h
- src/gui/main_content.cpp

11.35 Gui::MainWindow Class Reference

Handles the main window of Hivemind.

```
#include <main_window.h>
```

Inheritance diagram for Gui::MainWindow:



Signals

- void ScenarioCompiled (std::pair< CompileScenario::Scenario::RouteMap::iterator, CompileScenario::Scenario::RouteMap::iterator >)
- void ScenarioLoaded ()
- void AgentAdded (std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator >)
- void SyncAgentColor ()

Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)

  Constructs the main window.
- ∼MainWindow ()

  Descructs the main window.

Private Slots

- void SaveScenario (const std::string &filepath)
- void LoadScenario (const std::string &filepath)
- void UpdateScenario (float, float, float)
- void CompileScenario ()
- void CreateNewAgent ()

Private Member Functions

- void ConnectSlotsAndSignals ()

Private Attributes

- MenuBar ∗ m_MenuBar

  The menu bar of the main window.
- MainContent ∗ m_MainContent

  The main content of the main window.
- std::shared_ptr< CompileScenario::Scenario > m_Scenario
- MapDialog ∗ m_ScenarioSettingsDialog

11.35.1 Detailed Description

Handles the main window of Hivemind.

This class is responsible for handling the main window of Hivemind, which contains the core functionality such as scenario editing, simulation and launching.

Definition at line 17 of file main_window.h.

11.35.2 Constructor & Destructor Documentation

11.35.2.1 MainWindow()

```
Gui::MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

Constructs the main window.

Parameters

parent	The parent widget of main window

Definition at line 18 of file main_window.cpp.

References ConnectSlotsAndSignals(), CreateNewAgent(), m_MainContent, and m_MenuBar.

11.35.2.2 ∼MainWindow()

```
Gui::MainWindow::∼MainWindow ( )
```

Descructs the main window.

Definition at line 37 of file main_window.cpp.

11.35.3 Member Function Documentation

11.35.3.1 AgentAdded

```
void Gui::MainWindow::AgentAdded (
            std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >↩
::iterator > ) [signal]
```

Referenced by ConnectSlotsAndSignals(), CreateNewAgent(), and LoadScenario().

11.35.3.2 CompileScenario

```
void Gui::MainWindow::CompileScenario ( )  [private], [slot]
```

Definition at line 161 of file main_window.cpp.

References m_Scenario, and ScenarioCompiled().

Referenced by ConnectSlotsAndSignals().

11.35.3.3 ConnectSlotsAndSignals()

```
void Gui::MainWindow::ConnectSlotsAndSignals ( )  [private]
```

Definition at line 40 of file main_window.cpp.

References AgentAdded(), CompileScenario(), CreateNewAgent(), Gui::MapViewer::DataReceived(), MapManagement::MapManage KeyframeManagement::KeyframeManager::Instance(), MapManagement::MapManager::Instance(), LoadScenario(), m_MenuBar, m_ScenarioSettingsDialog, MapManagement::MapManager::RequestImage(), SaveScenario(), ScenarioCompiled(), ScenarioLoaded(), SyncAgentColor(), UpdateScenario(), and Gui::MapViewer::WaitForData().

Referenced by MainWindow().

11.35.3.4 CreateNewAgent

```
void Gui::MainWindow::CreateNewAgent ( )  [private], [slot]
```

Definition at line 169 of file main_window.cpp.

References AgentAdded(), getRandomColor(), m_Scenario, and SyncAgentColor().

Referenced by ConnectSlotsAndSignals(), and MainWindow().

11.35.3.5 LoadScenario

```
void Gui::MainWindow::LoadScenario (
             const std::string & filepath )  [private], [slot]
```

Definition at line 145 of file main_window.cpp.

References AgentAdded(), m_Scenario, and ScenarioLoaded().

Referenced by ConnectSlotsAndSignals().

11.35.3.6 SaveScenario

```
void Gui::MainWindow::SaveScenario (
            const std::string & filepath )  [private], [slot]
```

Definition at line 139 of file main_window.cpp.

References m_Scenario.

Referenced by ConnectSlotsAndSignals().

11.35.3.7 ScenarioCompiled

```
void Gui::MainWindow::ScenarioCompiled (
            std::pair< CompileScenario::Scenario::RouteMap::iterator, CompileScenario::↩
Scenario::RouteMap::iterator > )  [signal]
```

Referenced by CompileScenario(), and ConnectSlotsAndSignals().

11.35.3.8 ScenarioLoaded

```
void Gui::MainWindow::ScenarioLoaded ( )  [signal]
```

Referenced by ConnectSlotsAndSignals(), and LoadScenario().

11.35.3.9 SyncAgentColor

```
void Gui::MainWindow::SyncAgentColor ( )  [signal]
```

Referenced by ConnectSlotsAndSignals(), and CreateNewAgent().

11.35.3.10 UpdateScenario

```
void Gui::MainWindow::UpdateScenario (
            float latitude,
            float longitude,
            float size )  [private], [slot]
```

Definition at line 154 of file main_window.cpp.

References m_Scenario.

Referenced by ConnectSlotsAndSignals().

11.35.4 Member Data Documentation

11.35.4.1 m_MainContent

`MainContent* Gui::MainWindow::m_MainContent [private]`

The main content of the main window.

Basically all content other than the menubar.

Definition at line 55 of file main_window.h.

Referenced by MainWindow().

11.35.4.2 m_MenuBar

`MenuBar* Gui::MainWindow::m_MenuBar [private]`

The menu bar of the main window.

Definition at line 51 of file main_window.h.

Referenced by ConnectSlotsAndSignals(), and MainWindow().

11.35.4.3 m_Scenario

`std::shared_ptr<CompileScenario::Scenario> Gui::MainWindow::m_Scenario [private]`

Definition at line 57 of file main_window.h.

Referenced by CompileScenario(), CreateNewAgent(), LoadScenario(), SaveScenario(), and UpdateScenario().

11.35.4.4 m_ScenarioSettingsDialog

`MapDialog* Gui::MainWindow::m_ScenarioSettingsDialog [private]`

Definition at line 58 of file main_window.h.

Referenced by ConnectSlotsAndSignals().

The documentation for this class was generated from the following files:

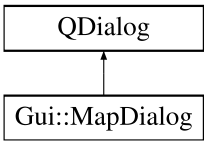
- include/gui/main_window.h
- src/gui/main_window.cpp

11.36 Gui::MapDialog Class Reference

The MapDialog class represents a dialog window for inputting map data.

```
#include <map_dialog.h>
```

Inheritance diagram for Gui::MapDialog:



Public Slots

- void Finish ()

  Slot called when the user finishes input and submits the data.

Signals

- void SendData (const QString &data)

  Signal emitted when data is ready to be sent.
- void Finished ()

  Signal emitted when the dialog has finished.
- void MapDataReady (float latitude, float longitude, float size)

  Signal emitted when map data is ready to be processed.

Public Member Functions

- MapDialog (QWidget *parent=nullptr)

  Constructs a new MapDialog object.

Private Attributes

- QLineEdit * m_LatitudeCoordInput
- QLineEdit * m_LongitudeCoordInput
- QLineEdit * m_SizeInput

11.36.1 Detailed Description

The MapDialog class represents a dialog window for inputting map data.

Definition at line 15 of file map_dialog.h.

11.36.2 Constructor & Destructor Documentation

11.36.2.1 MapDialog()

```
Gui::MapDialog::MapDialog (
            QWidget * parent = nullptr )
```

Constructs a new MapDialog object.

Parameters

parent	The parent widget of the dialog.

Definition at line 10 of file map_dialog.cpp.

References Finish(), m_LatitudeCoordInput, m_LongitudeCoordInput, and m_SizeInput.

11.36.3 Member Function Documentation

11.36.3.1 Finish

```
void Gui::MapDialog::Finish ( )  [slot]
```

Slot called when the user finishes input and submits the data.

Definition at line 41 of file map_dialog.cpp.

References Finished(), m_LatitudeCoordInput, m_LongitudeCoordInput, m_SizeInput, and MapDataReady().

Referenced by MapDialog().

11.36.3.2 Finished

```
void Gui::MapDialog::Finished ( )  [signal]
```

Signal emitted when the dialog has finished.

Referenced by Finish().

11.36.3.3 MapDataReady

```
void Gui::MapDialog::MapDataReady (
             float latitude,
             float longitude,
             float size )  [signal]
```

Signal emitted when map data is ready to be processed.

Parameters

latitude	The latitude coordinate of the map data.
longitude	The longitude coordinate of the map data.
size	The size of the map data.

Referenced by Finish().

11.36.3.4 SendData

```
void Gui::MapDialog::SendData (
            const QString & data )  [signal]
```

Signal emitted when data is ready to be sent.

Parameters

data	The data to be sent.

11.36.4 Member Data Documentation

11.36.4.1 m_LatitudeCoordInput

```
QLineEdit* Gui::MapDialog::m_LatitudeCoordInput  [private]
```

Definition at line 54 of file map_dialog.h.

Referenced by Finish(), and MapDialog().

11.36.4.2 m_LongitudeCoordInput

```
QLineEdit* Gui::MapDialog::m_LongitudeCoordInput  [private]
```

Definition at line 55 of file map_dialog.h.

Referenced by Finish(), and MapDialog().

11.36.4.3 m_SizeInput

```
QLineEdit* Gui::MapDialog::m_SizeInput  [private]
```

Definition at line 56 of file map_dialog.h.

Referenced by Finish(), and MapDialog().

The documentation for this class was generated from the following files:

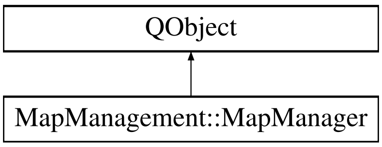
- include/gui/map_dialog.h
- src/gui/map_dialog.cpp

11.37 MapManagement::MapManager Class Reference

This is the class responsible for retrieving maps from Kartverket.

```
#include <map_manager.h>
```

Inheritance diagram for MapManagement::MapManager:



Signals

- void GotImage ()
  Signal emitted when the map image data has been retrieved.
- void RequestImage ()

Static Public Member Functions

- static MapManager & Instance ()
  Returns the singleton instance of the class.
- static void GetMap (Core::UTMCoordinate coord, int size)
  Retrieves the map from Kartverket for the specified UTM coordinate and size.
- static void CalculateCornerCoordinates (Core::UTMCoordinate coord, int size)
  Calculates the UTM corner coordinates for the specified UTM coordinate and size.
- static QByteArray & GetData ()
  Returns the map data as a byte array.
- static int GetImageResolution ()

Private Member Functions

- MapManager ()
  Constructor.
- ∼MapManager ()=default
  Destructor.

Private Attributes

- QByteArray m_Data
- QString m_Area
- int m_ImageResolution

11.37.1 Detailed Description

This is the class responsible for retrieving maps from Kartverket.

Definition at line 14 of file map_manager.h.

11.37.2 Constructor & Destructor Documentation

11.37.2.1 MapManager()

```
MapManagement::MapManager::MapManager ( )  [inline], [private]
```

Constructor.

Definition at line 67 of file map_manager.h.

11.37.2.2 ∼MapManager()

```
MapManagement::MapManager::~MapManager ( )  [private], [default]
```

Destructor.

11.37.3 Member Function Documentation

11.37.3.1 CalculateCornerCoordinates()

```
void MapManagement::MapManager::CalculateCornerCoordinates (
            Core::UTMCoordinate coord,
            int size )  [static]
```

Calculates the UTM corner coordinates for the specified UTM coordinate and size.

This function calculates the UTM corner coordinates for the specified UTM coordinate and size, and stores them in the CornerCoordinates variable.

Parameters

coord	The UTM coordinate for the center of the map.
size	The size of the map in meters.

Definition at line 77 of file map_manager.cpp.

References Core::UTMCoordinate::Easting, Instance(), m_Area, and Core::UTMCoordinate::Northing.

Referenced by GetMap().

11.37.3.2 GetData()

```
static QByteArray & MapManagement::MapManager::GetData ( )  [inline], [static]
```

Returns the map data as a byte array.

Definition at line 49 of file map_manager.h.

References Instance(), and m_Data.

Referenced by Gui::MapViewer::paintEvent().

11.37.3.3 GetImageResolution()

```
static int MapManagement::MapManager::GetImageResolution ( )  [inline], [static]
```

Definition at line 55 of file map_manager.h.

References Instance(), and m_ImageResolution.

Referenced by Gui::MapViewer::paintEvent().

11.37.3.4 GetMap()

```
void MapManagement::MapManager::GetMap (
            Core::UTMCoordinate coord,
            int size )  [static]
```

Retrieves the map from Kartverket for the specified UTM coordinate and size.

This function retrieves the satellite map data from Kartverket with a HTTP request for the specified UTM coordinate and size.

Parameters

coord	The UTM coordinate for the center of the map.
size	The size of the map in meters.

Definition at line 17 of file map_manager.cpp.

References CalculateCornerCoordinates(), Instance(), m_Area, m_ImageResolution, and RequestImage().

Referenced by CompileScenario::Scenario::Scenario(), and CompileScenario::Scenario::SetOrigin().

11.37.3.5 GotImage

```
void MapManagement::MapManager::GotImage ( )  [signal]
```

Signal emitted when the map image data has been retrieved.

Referenced by Gui::MainWindow::ConnectSlotsAndSignals().

11.37.3.6 Instance()

```
static MapManager & MapManagement::MapManager::Instance ( )  [inline], [static]
```

Returns the singleton instance of the class.

Definition at line 20 of file map_manager.h.

Referenced by CalculateCornerCoordinates(), Gui::MainWindow::ConnectSlotsAndSignals(), GetData(), GetImageResolution(), and GetMap().

11.37.3.7 RequestImage

```
void MapManagement::MapManager::RequestImage ( )  [signal]
```

Referenced by Gui::MainWindow::ConnectSlotsAndSignals(), and GetMap().

11.37.4 Member Data Documentation

11.37.4.1 m_Area

```
QString MapManagement::MapManager::m_Area  [private]
```

Definition at line 73 of file map_manager.h.

Referenced by CalculateCornerCoordinates(), and GetMap().

11.37.4.2 m_Data

```
QByteArray MapManagement::MapManager::m_Data [private]
```

Definition at line 72 of file map_manager.h.

Referenced by GetData().

11.37.4.3 m_ImageResolution

```
int MapManagement::MapManager::m_ImageResolution [private]
```

Definition at line 74 of file map_manager.h.

Referenced by GetImageResolution(), and GetMap().

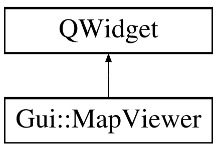
The documentation for this class was generated from the following files:

- include/map_management/map_manager.h
- src/map_management/map_manager.cpp

11.38 Gui::MapViewer Class Reference

```
#include <map_viewer.h>
```

Inheritance diagram for Gui::MapViewer:



Public Slots

- void DataReceived ()
- void WaitForData ()
- void UpdateRoutes (std::pair< CompileScenario::Scenario::RouteMap::iterator, CompileScenario::↩ Scenario::RouteMap::iterator > routes)
- void UpdateAgents (std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator > agents)
- void UpdateActiveAgent (int id)
- void UpdateTimeStamp (float timeStamp)

Public Member Functions

- MapViewer (QWidget *parent=nullptr)

Protected Member Functions

- void paintEvent (QPaintEvent *event) override
- void resizeEvent (QResizeEvent *event) override
- void mousePressEvent (QMouseEvent *event) override

Private Member Functions

- void UpdateRenderingArea ()
- void DrawKeyframes (QPainter &painter)
- void DrawRoutes (QPainter &painter)
- void DrawLoader (QPainter &painter) const

Private Attributes

- int m_StartX
- int m_StartY
- int m_Size
- bool m_WaitingForData
- QTimer * m_WaitingForDataTimer
- QElapsedTimer m_WaitingForDataElapsedTimer
- float m_LoaderAngle
- int m_LoaderSize
- float m_LoaderSpeed
- float m_LoaderSpan
- int m_LoaderThickness
- std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator > m_Agents
- std::pair< CompileScenario::Scenario::RouteMap::iterator, CompileScenario::Scenario::RouteMap::iterator > m_Routes
- int m_ActiveAgentId
- float m_TimeStamp

11.38.1 Detailed Description

Definition at line 16 of file map_viewer.h.

11.38.2 Constructor & Destructor Documentation

11.38.2.1 MapViewer()

```
Gui::MapViewer::MapViewer (
            QWidget * parent = nullptr )   [explicit]
```

Definition at line 12 of file map_viewer.cpp.

References m_LoaderAngle, m_LoaderSpeed, m_WaitingForDataElapsedTimer, m_WaitingForDataTimer, UpdateRenderingArea(), and WaitForData().

11.38.3 Member Function Documentation

11.38.3.1 DataReceived

```
void Gui::MapViewer::DataReceived ( ) [slot]
```

Definition at line 132 of file map_viewer.cpp.

References m_WaitingForData, and m_WaitingForDataTimer.

Referenced by Gui::MainWindow::ConnectSlotsAndSignals().

11.38.3.2 DrawKeyframes()

```
void Gui::MapViewer::DrawKeyframes (
            QPainter & painter ) [private]
```

Definition at line 141 of file map_viewer.cpp.

References Core::Agent::Color, KeyframeManagement::KeyframeManager::GetKeyframes(), CoordinateConverter::CoordConv::GetS
KeyframeManagement::KeyframeManager::Instance(), m_Agents, m_Size, m_StartX, m_StartY, and CoordinateConverter::CoordCo

Referenced by paintEvent().

11.38.3.3 DrawLoader()

```
void Gui::MapViewer::DrawLoader (
            QPainter & painter ) const [private]
```

Definition at line 227 of file map_viewer.cpp.

References m_LoaderAngle, m_LoaderSize, m_LoaderSpan, m_LoaderThickness, m_Size, m_StartX, and m_StartY.

Referenced by paintEvent().

11.38.3.4 DrawRoutes()

```
void Gui::MapViewer::DrawRoutes (
            QPainter & painter ) [private]
```

Definition at line 175 of file map_viewer.cpp.

References Core::Agent::Color, CoordinateConverter::CoordConv::GetSize(), m_Agents, m_Routes, m_Size, m_StartX, m_StartY, CoordinateConverter::CoordConv::SymmetricToAsymmetric(), Core::CartesianCoordinate::X, and Core::CartesianCoordinate::Y.

Referenced by paintEvent().

11.38.3.5 mousePressEvent()

```
void Gui::MapViewer::mousePressEvent (
            QMouseEvent * event )  [override], [protected]
```

Definition at line 69 of file map_viewer.cpp.

References KeyframeManagement::KeyframeManager::AddKeyframe(), CoordinateConverter::CoordConv::AsymmetricToSymmetric(CoordinateConverter::CoordConv::GetSize(), KeyframeManagement::KeyframeManager::Instance(), m_ActiveAgentId, m_Size, m_StartX, m_StartY, and m_TimeStamp.

11.38.3.6 paintEvent()

```
void Gui::MapViewer::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

Definition at line 38 of file map_viewer.cpp.

References DrawKeyframes(), DrawLoader(), DrawRoutes(), MapManagement::MapManager::GetData(), MapManagement::MapManager::GetImageResolution(), m_Size, m_StartX, m_StartY, and m_WaitingForData.

11.38.3.7 resizeEvent()

```
void Gui::MapViewer::resizeEvent (
            QResizeEvent * event )  [override], [protected]
```

Definition at line 63 of file map_viewer.cpp.

References UpdateRenderingArea().

11.38.3.8 UpdateActiveAgent

```
void Gui::MapViewer::UpdateActiveAgent (
            int id )  [inline], [slot]
```

Definition at line 36 of file map_viewer.h.

References m_ActiveAgentId.

11.38.3.9 UpdateAgents

```
void Gui::MapViewer::UpdateAgents (
            std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >↩
::iterator > agents )  [slot]
```

Definition at line 258 of file map_viewer.cpp.

References m_Agents.

11.38.3.10 UpdateRenderingArea()

```
void Gui::MapViewer::UpdateRenderingArea ( )  [private]
```

Definition at line 109 of file map_viewer.cpp.

References m_Size, m_StartX, and m_StartY.

Referenced by MapViewer(), and resizeEvent().

11.38.3.11 UpdateRoutes

```
void Gui::MapViewer::UpdateRoutes (
            std::pair< CompileScenario::Scenario::RouteMap::iterator, CompileScenario::↩
Scenario::RouteMap::iterator > routes )  [slot]
```

Definition at line 248 of file map_viewer.cpp.

References m_Routes.

11.38.3.12 UpdateTimeStamp

```
void Gui::MapViewer::UpdateTimeStamp (
            float timeStamp )  [inline], [slot]
```

Definition at line 42 of file map_viewer.h.

References m_TimeStamp.

11.38.3.13 WaitForData

```
void Gui::MapViewer::WaitForData ( )  [slot]
```

Definition at line 123 of file map_viewer.cpp.

References m_WaitingForData, and m_WaitingForDataTimer.

Referenced by Gui::MainWindow::ConnectSlotsAndSignals(), and MapViewer().

11.38.4 Member Data Documentation

11.38.4.1 m_ActiveAgentId

```
int Gui::MapViewer::m_ActiveAgentId  [private]
```

Definition at line 79 of file map_viewer.h.

Referenced by mousePressEvent(), and UpdateActiveAgent().

11.38.4.2 m_Agents

```
std::pair<std::vector<Core::Agent>::iterator, std::vector<Core::Agent>::iterator> Gui::Map↩
Viewer::m_Agents  [private]
```

Definition at line 74 of file map_viewer.h.

Referenced by DrawKeyframes(), DrawRoutes(), and UpdateAgents().

11.38.4.3 m_LoaderAngle

```
float Gui::MapViewer::m_LoaderAngle  [private]
```

Definition at line 66 of file map_viewer.h.

Referenced by DrawLoader(), and MapViewer().

11.38.4.4 m_LoaderSize

`int Gui::MapViewer::m_LoaderSize [private]`

Definition at line 67 of file map_viewer.h.

Referenced by DrawLoader().

11.38.4.5 m_LoaderSpan

`float Gui::MapViewer::m_LoaderSpan [private]`

Definition at line 69 of file map_viewer.h.

Referenced by DrawLoader().

11.38.4.6 m_LoaderSpeed

`float Gui::MapViewer::m_LoaderSpeed [private]`

Definition at line 68 of file map_viewer.h.

Referenced by MapViewer().

11.38.4.7 m_LoaderThickness

`int Gui::MapViewer::m_LoaderThickness [private]`

Definition at line 70 of file map_viewer.h.

Referenced by DrawLoader().

11.38.4.8 m_Routes

```
std::pair<CompileScenario::Scenario::RouteMap::iterator, CompileScenario::Scenario::RouteMap↩
::iterator> Gui::MapViewer::m_Routes [private]
```

Definition at line 77 of file map_viewer.h.

Referenced by DrawRoutes(), and UpdateRoutes().

11.38.4.9 m_Size

```
int Gui::MapViewer::m_Size  [private]
```

Definition at line 61 of file map_viewer.h.

Referenced by DrawKeyframes(), DrawLoader(), DrawRoutes(), mousePressEvent(), paintEvent(), and UpdateRenderingArea().

11.38.4.10 m_StartX

```
int Gui::MapViewer::m_StartX  [private]
```

Definition at line 60 of file map_viewer.h.

Referenced by DrawKeyframes(), DrawLoader(), DrawRoutes(), mousePressEvent(), paintEvent(), and UpdateRenderingArea().

11.38.4.11 m_StartY

```
int Gui::MapViewer::m_StartY  [private]
```

Definition at line 60 of file map_viewer.h.

Referenced by DrawKeyframes(), DrawLoader(), DrawRoutes(), mousePressEvent(), paintEvent(), and UpdateRenderingArea().

11.38.4.12 m_TimeStamp

```
float Gui::MapViewer::m_TimeStamp  [private]
```

Definition at line 80 of file map_viewer.h.

Referenced by mousePressEvent(), and UpdateTimeStamp().

11.38.4.13 m_WaitingForData

```
bool Gui::MapViewer::m_WaitingForData  [private]
```

Definition at line 63 of file map_viewer.h.

Referenced by DataReceived(), paintEvent(), and WaitForData().

11.38.4.14 m_WaitingForDataElapsedTimer

`QElapsedTimer Gui::MapViewer::m_WaitingForDataElapsedTimer [private]`

Definition at line 65 of file map_viewer.h.

Referenced by MapViewer().

11.38.4.15 m_WaitingForDataTimer

`QTimer* Gui::MapViewer::m_WaitingForDataTimer [private]`

Definition at line 64 of file map_viewer.h.

Referenced by DataReceived(), MapViewer(), and WaitForData().

The documentation for this class was generated from the following files:

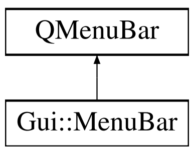
- include/gui/map_viewer.h
- src/gui/map_viewer.cpp

11.39 Gui::MenuBar Class Reference

The main menubar of the user interface.

`#include <menu_bar.h>`

Inheritance diagram for Gui::MenuBar:



Signals

- void SaveScenario (const std::string &filename)
- void LoadScenario (const std::string &filename)

Public Member Functions

- MenuBar (QWidget *parent=nullptr)

    Constructs the menu bar.

11.39.1 Detailed Description

The main menubar of the user interface.

The main menubar exists to provide the user with easy access to functionality such as creating new scenarios, opening existing scenarios etc.

Definition at line 13 of file menu_bar.h.

11.39.2 Constructor & Destructor Documentation

11.39.2.1 MenuBar()

```
Gui::MenuBar::MenuBar (
            QWidget * parent = nullptr )
```

Constructs the menu bar.

Parameters

parent	The parent widget of the menu bar

Definition at line 13 of file menu_bar.cpp.

References LoadScenario(), and SaveScenario().

11.39.3 Member Function Documentation

11.39.3.1 LoadScenario

```
void Gui::MenuBar::LoadScenario (
            const std::string & filename )  [signal]
```

Referenced by MenuBar().

11.39.3.2 SaveScenario

```
void Gui::MenuBar::SaveScenario (
            const std::string & filename )  [signal]
```

Referenced by MenuBar().

The documentation for this class was generated from the following files:

- include/gui/menu_bar.h
- src/gui/menu_bar.cpp

11.40 Routemaker::Node< T > Struct Template Reference

Represents a node in a Graph data structured made for path-finding.

```
#include <graph.h>
```

Public Attributes

- T Data

  Data stored in the the node.
- std::weak_ptr< Node< T > > Parent

  A non-owner pointer to the parent of the node.
- bool Visited

  Specifies if a given node has been visited during path-finding.
- double GlobalGoal

  Represents the assumed cost from the start to the goal node through this node.
- double LocalGoal

  Represents the cost from the start node to this node.

11.40.1 Detailed Description

template<typename T>
struct Routemaker::Node< T >

Represents a node in a Graph data structured made for path-finding.

Template Parameters

T	Type of data to store inside the node

Definition at line 17 of file graph.h.

11.40.2 Member Data Documentation

11.40.2.1 Data

```
template<typename T >
T Routemaker::Node< T >::Data
```

Data stored in the the node.

Stores data not needed by the A$*$ path-finding algorithm. This is what the user actually wants to store in the Graph.

Definition at line 23 of file graph.h.

Referenced by Routemaker::Routemaker::UpdateOrigin().

11.40.2.2 GlobalGoal

```
template<typename T >
double Routemaker::Node< T >::GlobalGoal
```

Represents the assumed cost from the start to the goal node through this node.

Should not be set by the user. The A∗ path-finding algorithm uses cost to find the shortest path in a reasonable amount of time. This member contains the sum of the cost to get to this node from the start node, represented in LocalGoal, plus the assumed cost to get from this node to the goal node. The A∗ path-finding algorithm uses this value during Graph traversal to sort a priority queue in order to explore the assumed shortest paths first.

Definition at line 52 of file graph.h.

11.40.2.3 LocalGoal

```
template<typename T >
double Routemaker::Node< T >::LocalGoal
```

Represents the cost from the start node to this node.

Should not be set by the user. The A∗ path-finding algorithm uses cost to find the shortest path in a reasonable amount of time. This member contains the sum of the cost to get to this node from the start node. While traversing the Graph, the A∗ path-finding algorithm updates and uses this member to check for shorter paths.

Definition at line 62 of file graph.h.

11.40.2.4 Parent

```
template<typename T >
std::weak_ptr<Node<T> > Routemaker::Node< T >::Parent
```

A non-owner pointer to the parent of the node.

Should not be set by user. The A∗ path-finding algorithm sets the value for this member when traversing the Graph. It used to find the way back to the start after the goal is found.

Definition at line 30 of file graph.h.

11.40.2.5 Visited

```
template<typename T >
bool Routemaker::Node< T >::Visited
```

Specifies if a given node has been visited during path-finding.

Should not be set by user. Is generally only used internally by the A∗ path-finding algorithm when traversing the Graph. May be used in debug views to visualize which nodes are visited during path-finding.

Definition at line 39 of file graph.h.

The documentation for this struct was generated from the following file:

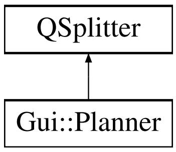
- include/routemaker/graph.h

11.41 Gui::Planner Class Reference

The planner widget used for planning scenarios.

```
#include <planner.h>
```

Inheritance diagram for Gui::Planner:



Public Member Functions

- Planner (QWidget ∗parent=nullptr)

  Constructs the planner widget.
- ∼Planner ()

  Destructs the planner widget.

Private Attributes

- MapViewer ∗ m_MapViewer

  The layout of the planner widget.
- Timeline ∗ m_Timeline

11.41.1 Detailed Description

The planner widget used for planning scenarios.

Contains the graphical functionality to plan scenarios.

Definition at line 13 of file planner.h.

11.41.2 Constructor & Destructor Documentation

11.41.2.1 Planner()

```
Gui::Planner::Planner (
            QWidget * parent = nullptr )
```

Constructs the planner widget.

Parameters

parent	The parent of the planner widget.

Definition at line 5 of file planner.cpp.

References m_MapViewer, and m_Timeline.

11.41.2.2 ∼Planner()

```
Gui::Planner::∼Planner ( )
```

Destructs the planner widget.

Definition at line 19 of file planner.cpp.

11.41.3 Member Data Documentation

11.41.3.1 m_MapViewer

```
MapViewer* Gui::Planner::m_MapViewer [private]
```

The layout of the planner widget.

Definition at line 26 of file planner.h.

Referenced by Planner().

11.41.3.2 m_Timeline

```
Timeline* Gui::Planner::m_Timeline [private]
```

Definition at line 27 of file planner.h.

Referenced by Planner().

The documentation for this class was generated from the following files:

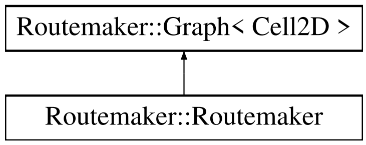
- include/gui/planner.h
- src/gui/planner.cpp

11.42 Routemaker::Routemaker Class Reference

Main class responsible for handling creation of routes between keyframes.

```
#include <routemaker.h>
```

Inheritance diagram for Routemaker::Routemaker:



Public Member Functions

- Routemaker (const Core::UTMCoordinate &origin, int size)
  Instatiates a routemaker object, along with it's Heightmap member.
- std::vector< Core::CartesianCoordinate > MakeRoute (const Core::Keyframe &a, const Core::Keyframe &b)
  Creates a a vector of coordinates defining a path between two keyframes.
- NodePtr GetNode (uint32_t x, uint32_t y) const
  Get a node at a position.
- void UpdateOrigin (Core::UTMCoordinate UTMOrigin, int size)
  Updates the origin coordinate and the size of the map.
- void UpdateResolution ()

Public Member Functions inherited from Routemaker::Graph< Cell2D >

- virtual std::vector< NodePtr > GetNeighbors (NodePtr node)=0
  Collects all neighbor nodes of `node`.
- virtual double GetCost (NodePtr a, NodePtr b)=0
  Returns the cost between `a` and `b`.
- virtual bool HasLineOfSight (NodePtr a, NodePtr b)=0
  Determines if there is a direct line of sight between node `a` and node `b`.
- virtual void ResetNodes (void)=0
  Resets all local and global goals and parent relationships of all nodes.
- void SolveAStar (NodePtr start, NodePtr goal)
  Finds cheapest path from `start` to `goal`.
- void PostSmooth (NodePtr start, NodePtr goal)
  Simplifies the path from `start` to `goal`.

Private Member Functions

- std::vector< NodePtr > GetNeighbors (NodePtr node) override
  Collects all neighbor nodes of `node`.
- double GetCost (NodePtr a, NodePtr b) override
  Returns the cost between `a` and `b`.
- bool HasLineOfSight (NodePtr a, NodePtr b) override
  Determines if there is a direct line of sight between node `a` and node `b`.
- void ResetNodes () override
  Resets all local and global goals and parent relationships of all nodes.
- std::list< NodePtr > BresenhamLine (const NodePtr &a, const NodePtr &b) const
  Calculates the `Bresenham Line` between two nodes.

Private Attributes

- std::vector< NodePtr > m_Nodes

  All the nodes that make up the graph.
- std::unique_ptr< HeightManagement::HeightManager > m_HeightMap

  HeightManager instance owned by Routemaker.
- int m_MapWidth

  Width (and height) of the active scenario.
- int m_RoutemakerRes

  Resolution of the routemaker in meters.
- int m_RoutemakerWidth

  Width (and height) of the routemaker.

Additional Inherited Members

Public Types inherited from Routemaker::Graph< Cell2D >

- using NodePtr = std::shared_ptr< Node< Cell2D > >

  Helper alias to make code more readable.

11.42.1 Detailed Description

Main class responsible for handling creation of routes between keyframes.

Definition at line 22 of file routemaker.h.

11.42.2 Constructor & Destructor Documentation

11.42.2.1 Routemaker()

```
Routemaker::Routemaker::Routemaker (
            const Core::UTMCoordinate & origin,
            int size )  [explicit]
```

Instatiates a routemaker object, along with it's Heightmap member.

The `origin` and `size` of the scenario are simply passed to the HeightMap member. In the case that the Height←Map class is converted to a singleton or the scenario class gains ownership over the Heightmap, they should not be necessary.

Parameters

origin	The origin of the scenario in UTM coordinate space.
size	The size of the scenario in meters

Definition at line 15 of file routemaker.cpp.

11.42.3 Member Function Documentation

11.42.3.1 BresenhamLine()

```
std::list< Routemaker::NodePtr > Routemaker::Routemaker::BresenhamLine (
            const NodePtr & a,
            const NodePtr & b ) const  [private]
```

Calculates the `Bresenham Line` between two nodes.

Parameters

a	Pointer to first node
b	Pointer to seconds node

Returns

A list of pointers to the nodes that make up the `Bresenham Line` between `a` and `b`.

Definition at line 203 of file routemaker.cpp.

11.42.3.2 GetCost()

```
double Routemaker::Routemaker::GetCost (
            NodePtr a,
            NodePtr b )  [override], [private], [virtual]
```

Returns the cost between `a` and `b`.

Implemented by sub-classes of Graph. The `a*` path-finding algorithm uses cost to efficiently find the best path between two nodes. In order to do this, it requires some method of calculating the cost of moving between any two nodes. It is up to the sub-class to define how this is calulated. An example of this cost may be the euclidean distance between two nodes.

Parameters

a	Pointer to the first Node
b	Pointer to the second Node

Returns

Cost between node `a` and node `b`.

Implements Routemaker::Graph< Cell2D >.

Definition at line 168 of file routemaker.cpp.

11.42.3.3 GetNeighbors()

```
std::vector< Routemaker::NodePtr > Routemaker::Routemaker::GetNeighbors (
            NodePtr node ) [override], [private], [virtual]
```

Collects all neighbor nodes of `node`.

Implemented by sub-classes of Graph. The neighbor relationship between nodes define the edges of the graph. It is up to the subclass to define these relationships. For a 2D grid, the neighbors would simply be the nodes directly to the north, south, east and west, in addition to the corners between them. For a road network, the relationships may be more complex.

Parameters

node	A pointer to the node from which to collect all neighbors

Returns

A vector of pointers to all the neighbors of `node`

Implements Routemaker::Graph< Cell2D >.

Definition at line 103 of file routemaker.cpp.

11.42.3.4 GetNode()

```
Routemaker::NodePtr Routemaker::Routemaker::GetNode (
            uint32_t x,
            uint32_t y ) const
```

Get a node at a position.

Parameters

x	x-coordinate of position
y	y-coordinate of position

Returns

A shared pointer to the node at the specified location

Definition at line 250 of file routemaker.cpp.

11.42.3.5 HasLineOfSight()

```
bool Routemaker::Routemaker::HasLineOfSight (
            NodePtr a,
            NodePtr b ) [override], [private], [virtual]
```

Determines if there is a direct line of sight between node a and node b.

Implemented by sub-classes of Graph. The Graph::PostSmooth method traverses the already found path through the A∗ path-finding algorithm and simplifies it by using this method. In a graph representing a 2D grid, a Bresenham implementation or ray-casting can be used to determine line of sight.

Parameters

a	Pointer to the first Node
b	Pointer to the second Node

Returns

bool specifying whether or not there is a direct line of sight

Implements Routemaker::Graph< Cell2D >.

Definition at line 184 of file routemaker.cpp.

11.42.3.6 MakeRoute()

```
std::vector< Core::CartesianCoordinate > Routemaker::Routemaker::MakeRoute (
            const Core::Keyframe & a,
            const Core::Keyframe & b )
```

Creates a a vector of coordinates defining a path between two keyframes.

Utilizes methods from the Graph interface, namely GetNeighbors, GetCost, HasLineOfSight and BresenhamLine, to generate a path between a and b.

Parameters

a	First keyframe to create to create path from
b	Second keyframe to create path from

returns A vector of coordinates in symmetrical cartesian coordinate system space, which together forms a path.

Definition at line 257 of file routemaker.cpp.

References Core::Keyframe::AgentId, CoordinateConverter::CoordConv::AsymmetricToSymmetric(), DRONE_FLIGHT_HEIGHT, Core::Keyframe::Position, CoordinateConverter::CoordConv::SymmetricToAsymmetric(), Core::Keyframe::TimeStamp, and Core::CartesianCoordinate::X.

11.42.3.7 ResetNodes()

```
void Routemaker::Routemaker::ResetNodes (
            void ) [override], [private], [virtual]
```

Resets all local and global goals and parent relationships of all nodes.

Implemented by sub-classes of Graph. In order to be able to re-use the same graph for several A∗ searches, the Graph::SolveAStar method needs to be able to reset all the nodes. As this interface does not contain the actual collection of nodes, this needs to be implemented in the sub-classes.

Implements Routemaker::Graph< Cell2D >.

Definition at line 26 of file routemaker.cpp.

11.42.3.8 UpdateOrigin()

```
void Routemaker::Routemaker::UpdateOrigin (
            Core::UTMCoordinate UTMOrigin,
            int size )
```

Updates the origin coordinate and the size of the map.

Parameters

UTMOrigin	The new origin coordinate for the map
size	The new size of the map in meters

Definition at line 64 of file routemaker.cpp.

References Routemaker::Node< T >::Data, and DRONE_FLIGHT_HEIGHT.

11.42.3.9 UpdateResolution()

```
void Routemaker::Routemaker::UpdateResolution ( )
```

Definition at line 44 of file routemaker.cpp.

11.42.4 Member Data Documentation

11.42.4.1 m_HeightMap

```
std::unique_ptr<HeightManagement::HeightManager> Routemaker::Routemaker::m_HeightMap [private]
```

HeightManager instance owned by Routemaker.

Definition at line 93 of file routemaker.h.

11.42.4.2 m_MapWidth

```
int Routemaker::Routemaker::m_MapWidth [private]
```

Width (and height) of the active scenario.

Definition at line 96 of file routemaker.h.

11.42.4.3 m_Nodes

```
std::vector<NodePtr> Routemaker::Routemaker::m_Nodes [private]
```

All the nodes that make up the graph.

Definition at line 90 of file routemaker.h.

11.42.4.4 m_RoutemakerRes

```
int Routemaker::Routemaker::m_RoutemakerRes [private]
```

Resolution of the routemaker in meters.

A resolution of 3 meters would mean that any one move in vertical or horizontal direction would correspond to a 3 meter movement. A higher value increases performance of the routemaker, but decreases route fidelity.

Definition at line 104 of file routemaker.h.

11.42.4.5 m_RoutemakerWidth

```
int Routemaker::Routemaker::m_RoutemakerWidth [private]
```

Width (and height) of the routemaker.

Will always equal *m_MapWidth* divided by *m_RoutemakerRes*

Definition at line 109 of file routemaker.h.

The documentation for this class was generated from the following files:

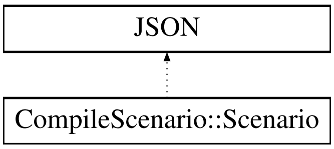
- include/routemaker/routemaker.h
- src/routemaker/routemaker.cpp

11.43 CompileScenario::Scenario Class Reference

The Scenario class represents a scenario with keyframes and routes.

```
#include <scenario.h>
```

Inheritance diagram for CompileScenario::Scenario:



Public Types

- using RouteMap = std::map< int, std::vector< std::vector< Core::CartesianCoordinate > > >

Public Member Functions

- Scenario (std::string name, Core::GeographicalCoordinate origin, int size)

    Constructs a new Scenario object with the given name, origin, and size.
- RouteMap & Compile ()

    Compiles the scenario into a map of routes.
- void save (std::string filename)

    Saves the scenario to a file with the given filename.
- void load (std::string filename)

    Loads a scenario from a file with the given filename.
- std::pair< RouteMap::iterator, RouteMap::iterator > GetRoutes ()
- std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator > GetAgents ()
- void AddAgent (Core::Agent newAgent)
- void SetOrigin (Core::GeographicalCoordinate GeoCoord, int size)

    Sets the origin of the scenario to the given geographical coordinates and size.

Private Member Functions

- JSONSTART JSONSTRING (m_Name)
- JSONSTART JSONMEMBER (Core::GeographicalCoordinate, m_Origin)
- JSONSTART JSONINT (m_Size)
- JSONSTART JSONMEMBERVECTOR (Core::Agent, m_Agents)

Private Attributes

- KeyframeManagement::KeyframeManager & m_KeyframeManager
- std::vector< Core::Agent > m_Agents
- RouteMap m_Routes
- std::unique_ptr< Routemaker::Routemaker > m_Routemaker
- std::string m_Name
- Core::GeographicalCoordinate m_Origin
- int m_Size

11.43.1 Detailed Description

The Scenario class represents a scenario with keyframes and routes.

The Scenario class provides functionality for creating a scenario with keyframes and routes, as well as saving and loading the scenario to and from file.

Definition at line 21 of file scenario.h.

11.43.2 Member Typedef Documentation

11.43.2.1 RouteMap

```
using CompileScenario::Scenario::RouteMap = std::map<int, std::vector<std::vector<Core::CartesianCoordinate>
>>
```

Definition at line 24 of file scenario.h.

11.43.3 Constructor & Destructor Documentation

11.43.3.1 Scenario()

```
CompileScenario::Scenario::Scenario (
            std::string name,
            Core::GeographicalCoordinate origin,
            int size )
```

Constructs a new Scenario object with the given name, origin, and size.

Parameters

name	The name of the scenario.
origin	The geographical coordinates of the origin.
size	The size of the scenario.

Definition at line 11 of file scenario.cpp.

References CoordinateConverter::CoordConv::GeographicToUTM(), MapManagement::MapManager::GetMap(), m_Routemaker, and CoordinateConverter::CoordConv::ResetOrigin().

11.43.4 Member Function Documentation

11.43.4.1 AddAgent()

```
void CompileScenario::Scenario::AddAgent (
            Core::Agent newAgent )
```

Definition at line 84 of file scenario.cpp.

11.43.4.2 Compile()

```
Scenario::RouteMap & CompileScenario::Scenario::Compile ( )
```

Compiles the scenario into a map of routes.

Returns

A map of routes.

Definition at line 39 of file scenario.cpp.

References KeyframeManagement::KeyframeManager::GetKeyframes(), m_KeyframeManager, m_Routes, and Core::Keyframe::TimeStamp.

11.43.4.3 GetAgents()

```
std::pair< std::vector< Core::Agent >::iterator, std::vector< Core::Agent >::iterator >
CompileScenario::Scenario::GetAgents ( ) [inline]
```

Definition at line 57 of file scenario.h.

References m_Agents.

11.43.4.4 GetRoutes()

```
std::pair< RouteMap::iterator, RouteMap::iterator > CompileScenario::Scenario::GetRoutes ( )
[inline]
```

Definition at line 49 of file scenario.h.

References m_Routes.

11.43.4.5 JSONINT()

```
JSONSTART CompileScenario::Scenario::JSONINT (
            m_Size ) [private]
```

11.43.4.6 JSONMEMBER()

```
JSONSTART CompileScenario::Scenario::JSONMEMBER (
            Core::GeographicalCoordinate ,
            m_Origin ) [private]
```

11.43.4.7 JSONMEMBERVECTOR()

```
JSONSTART CompileScenario::Scenario::JSONMEMBERVECTOR (
            Core::Agent ,
            m_Agents ) [private]
```

11.43.4.8 JSONSTRING()

```
JSONSTART CompileScenario::Scenario::JSONSTRING (
            m_Name ) [private]
```

11.43.4.9 load()

```
void CompileScenario::Scenario::load (
            std::string filename )
```

Loads a scenario from a file with the given filename.

Parameters

filename	The name of the file to load from.

Definition at line 96 of file scenario.cpp.

References Json::deserialize().

11.43.4.10 save()

```
void CompileScenario::Scenario::save (
            std::string filename )
```

Saves the scenario to a file with the given filename.

Parameters

filename	The name of the file to save to.

Definition at line 90 of file scenario.cpp.

References Json::serialize().

11.43.4.11 SetOrigin()

```
void CompileScenario::Scenario::SetOrigin (
            Core::GeographicalCoordinate GeoCoord,
            int size )
```

Sets the origin of the scenario to the given geographical coordinates and size.

Parameters

GeoCoord	The geographical coordinates of the origin.
size	The size of the scenario.

Definition at line 27 of file scenario.cpp.

References CoordinateConverter::CoordConv::GeographicToUTM(), MapManagement::MapManager::GetMap(), m_Origin, m_Routemaker, m_Size, and CoordinateConverter::CoordConv::ResetOrigin().

11.43.5 Member Data Documentation

11.43.5.1 m_Agents

```
std::vector<Core::Agent> CompileScenario::Scenario::m_Agents  [private]
```

Definition at line 74 of file scenario.h.

Referenced by GetAgents().

11.43.5.2 m_KeyframeManager

`KeyframeManagement::KeyframeManager& CompileScenario::Scenario::m_KeyframeManager [private]`

Definition at line 73 of file scenario.h.

Referenced by Compile().

11.43.5.3 m_Name

`std::string CompileScenario::Scenario::m_Name [private]`

Definition at line 77 of file scenario.h.

11.43.5.4 m_Origin

`Core::GeographicalCoordinate CompileScenario::Scenario::m_Origin [private]`

Definition at line 78 of file scenario.h.

Referenced by SetOrigin().

11.43.5.5 m_Routemaker

`std::unique_ptr<Routemaker::Routemaker> CompileScenario::Scenario::m_Routemaker [private]`

Definition at line 76 of file scenario.h.

Referenced by Scenario(), and SetOrigin().

11.43.5.6 m_Routes

`RouteMap CompileScenario::Scenario::m_Routes [private]`

Definition at line 75 of file scenario.h.

Referenced by Compile(), and GetRoutes().

11.43.5.7 m_Size

```
int CompileScenario::Scenario::m_Size [private]
```

Definition at line 79 of file scenario.h.

Referenced by SetOrigin().

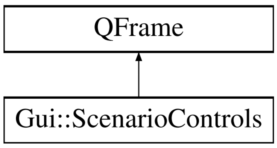
The documentation for this class was generated from the following files:

- include/compile_scenario/scenario.h
- src/compile_scenario/scenario.cpp

11.44 Gui::ScenarioControls Class Reference

```
#include <scenario_controls.h>
```

Inheritance diagram for Gui::ScenarioControls:



Signals

- void OpenSettingsDialog ()
- void CompileScenario ()

Public Member Functions

- ScenarioControls (QWidget ∗parent=nullptr)

Private Attributes

- QPushButton ∗ m_SettingsButton
- QPushButton ∗ m_CompileButton
- QGridLayout ∗ m_Layout

11.44.1 Detailed Description

Definition at line 10 of file scenario_controls.h.

11.44.2 Constructor & Destructor Documentation

11.44.2.1 ScenarioControls()

```
Gui::ScenarioControls::ScenarioControls (
            QWidget * parent = nullptr ) [explicit]
```

Definition at line 8 of file scenario_controls.cpp.

References CompileScenario(), m_CompileButton, m_Layout, m_SettingsButton, and OpenSettingsDialog().

11.44.3 Member Function Documentation

11.44.3.1 CompileScenario

```
void Gui::ScenarioControls::CompileScenario ( ) [signal]
```

Referenced by ScenarioControls().

11.44.3.2 OpenSettingsDialog

```
void Gui::ScenarioControls::OpenSettingsDialog ( ) [signal]
```

Referenced by ScenarioControls().

11.44.4 Member Data Documentation

11.44.4.1 m_CompileButton

```
QPushButton* Gui::ScenarioControls::m_CompileButton [private]
```

Definition at line 22 of file scenario_controls.h.

Referenced by ScenarioControls().

11.44.4.2 m_Layout

```
QGridLayout* Gui::ScenarioControls::m_Layout [private]
```

Definition at line 23 of file scenario_controls.h.

Referenced by ScenarioControls().

11.44.4.3 m_SettingsButton

```
QPushButton* Gui::ScenarioControls::m_SettingsButton [private]
```

Definition at line 21 of file scenario_controls.h.

Referenced by ScenarioControls().

The documentation for this class was generated from the following files:

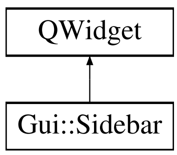
- include/gui/scenario_controls.h
- src/gui/scenario_controls.cpp

11.45 Gui::Sidebar Class Reference

The sidebar of the main window.

```
#include <sidebar.h>
```

Inheritance diagram for Gui::Sidebar:



Signals

- void scenarioDataReady (Core::UTMCoordinate coord, int size)

    Signal emitted when scenario data is ready to be processed.

Public Member Functions

- Sidebar (QWidget ∗parent=nullptr)

    Construct the sidebar.

Private Attributes

- QVBoxLayout ∗ m_Layout

    The layout of the sidebar.
- ScenarioControls ∗ m_ScenarioControls
- AgentControls ∗ m_AgentControls
- KeyframeControls ∗ m_KeyframeControls

11.45.1 Detailed Description

The sidebar of the main window.

The sidebar of the main content exists to provide the user access to tools related to the active tab in the tab widget.

Definition at line 22 of file sidebar.h.

11.45.2 Constructor & Destructor Documentation

11.45.2.1 Sidebar()

```
Gui::Sidebar::Sidebar (
            QWidget * parent = nullptr )
```

Construct the sidebar.

Parameters

parent	The parent of the sidebar.

Definition at line 17 of file sidebar.cpp.

References m_AgentControls, m_KeyframeControls, m_Layout, and m_ScenarioControls.

11.45.3 Member Function Documentation

11.45.3.1 scenarioDataReady

```
void Gui::Sidebar::scenarioDataReady (
            Core::UTMCoordinate coord,
            int size ) [signal]
```

Signal emitted when scenario data is ready to be processed.

Parameters

coord	The UTM coordinate of the center of the scenario.
size	The size of the scenario in meters.

11.45.4 Member Data Documentation

11.45.4.1 m_AgentControls

`AgentControls* Gui::Sidebar::m_AgentControls [private]`

Definition at line 66 of file sidebar.h.

Referenced by Sidebar().

11.45.4.2 m_KeyframeControls

`KeyframeControls* Gui::Sidebar::m_KeyframeControls [private]`

Definition at line 67 of file sidebar.h.

Referenced by Sidebar().

11.45.4.3 m_Layout

`QVBoxLayout* Gui::Sidebar::m_Layout [private]`

The layout of the sidebar.

Definition at line 63 of file sidebar.h.

Referenced by Sidebar().

11.45.4.4 m_ScenarioControls

`ScenarioControls* Gui::Sidebar::m_ScenarioControls [private]`

Definition at line 65 of file sidebar.h.

Referenced by Sidebar().

The documentation for this class was generated from the following files:

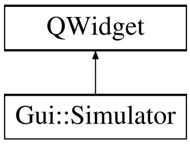
- include/gui/sidebar.h
- src/gui/sidebar.cpp

11.46 Gui::Simulator Class Reference

The simulator widget used to simulate scenarios.

```
#include <simulator.h>
```

Inheritance diagram for Gui::Simulator:



Public Member Functions

- Simulator (QWidget ∗parent=nullptr)

  Constructs the simulator widget.
- ∼Simulator ()

  Destructs the simulator widget.
- QSize sizeHint () const override

Private Attributes

- QGridLayout ∗ m_Layout

  The layout of the simulator widget.

11.46.1 Detailed Description

The simulator widget used to simulate scenarios.

Contains the graphical functionality to simulate scenarios.

Definition at line 13 of file simulator.h.

11.46.2 Constructor & Destructor Documentation

11.46.2.1 Simulator()

```
Gui::Simulator::Simulator (
            QWidget * parent = nullptr )
```

Constructs the simulator widget.

Parameters

parent	The parent of the simulator widget.

Definition at line 7 of file simulator.cpp.

References m_Layout.

11.46.2.2 ∼Simulator()

```
Gui::Simulator::∼Simulator ( )
```

Destructs the simulator widget.

Definition at line 19 of file simulator.cpp.

11.46.3 Member Function Documentation

11.46.3.1 sizeHint()

```
QSize Gui::Simulator::sizeHint ( ) const [inline], [override]
```

Definition at line 24 of file simulator.h.

11.46.4 Member Data Documentation

11.46.4.1 m_Layout

```
QGridLayout* Gui::Simulator::m_Layout [private]
```

The layout of the simulator widget.

Definition at line 31 of file simulator.h.

Referenced by Simulator().

The documentation for this class was generated from the following files:

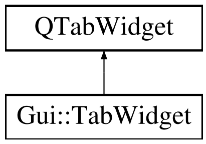
- include/gui/simulator.h
- src/gui/simulator.cpp

11.47 Gui::TabWidget Class Reference

The tab widget of the main window.

```
#include <tab_widget.h>
```

Inheritance diagram for Gui::TabWidget:



Public Member Functions

- TabWidget (QWidget ∗parent=nullptr)

    Constructs the tab widget.
- ∼TabWidget ()

    Destructs the tab widget.

Private Attributes

- Planner ∗ m_Planner

    The planner widget.
- Simulator ∗ m_Simulator

    The simulator widget.
- Launcher ∗ m_Launcher

    The launcher widget.

11.47.1 Detailed Description

The tab widget of the main window.

Hivemind; planning, simulating and launching. They are separated in their own tabs as a user should only need to access one of these at any point in time.

Definition at line 18 of file tab_widget.h.

11.47.2 Constructor & Destructor Documentation

11.47.2.1 TabWidget()

```
Gui::TabWidget::TabWidget (
            QWidget * parent = nullptr )
```

Constructs the tab widget.

Parameters

parent	The parent of the tab widget.

Definition at line 7 of file tab_widget.cpp.

References m_Launcher, m_Planner, and m_Simulator.

11.47.2.2 ∼TabWidget()

```
Gui::TabWidget::∼TabWidget ( )
```

Destructs the tab widget.

Definition at line 20 of file tab_widget.cpp.

11.47.3 Member Data Documentation

11.47.3.1 m_Launcher

```
Launcher* Gui::TabWidget::m_Launcher [private]
```

The launcher widget.

Contains the graphical functionality to launch a scenario.

Definition at line 42 of file tab_widget.h.

Referenced by TabWidget().

11.47.3.2 m_Planner

```
Planner* Gui::TabWidget::m_Planner [private]
```

The planner widget.

Contains the graphical functionality to plan scenarios.

Definition at line 32 of file tab_widget.h.

Referenced by TabWidget().

11.47.3.3 m_Simulator

```
Simulator* Gui::TabWidget::m_Simulator [private]
```

The simulator widget.

Contains the graphical functionality to simulate scenarios.

Definition at line 37 of file tab_widget.h.

Referenced by TabWidget().

The documentation for this class was generated from the following files:

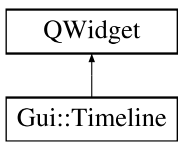
- include/gui/tab_widget.h
- src/gui/tab_widget.cpp

11.48 Gui::Timeline Class Reference

A custom QWidget to represent a timeline with keyframes.

```
#include <timeline.h>
```

Inheritance diagram for Gui::Timeline:



Signals

- void timeStampSelected (float timeStamp)

    Signal that is emitted when a timestamp is selected.

Public Member Functions

- Timeline (QWidget *parent=nullptr)

    Constructor for the Timeline class.
- int GetActiveAgent ()

    Get the active agent ID.
- float GetTimeStamp ()

    Get the current timestamp.

Protected Member Functions

- void paintEvent (QPaintEvent ∗event) override

  Paint event handler.
- void mouseReleaseEvent (QMouseEvent ∗event) override

  Mouse release event handler.
- void resizeEvent (QResizeEvent ∗event) override

  Resize event handler.

Private Attributes

- float m_timeStamp

  The current timestamp.
- int m_activeAgentId

  ID of the active agent.
- float m_pixelsPerSecond

  Pixels per second on the timeline.

11.48.1 Detailed Description

A custom QWidget to represent a timeline with keyframes.

Definition at line 12 of file timeline.h.

11.48.2 Constructor & Destructor Documentation

11.48.2.1 Timeline()

```
Gui::Timeline::Timeline (
            QWidget * parent = nullptr )  [explicit]
```

Constructor for the Timeline class.

Parameters

parent	The parent QWidget

Definition at line 13 of file timeline.cpp.

References KeyframeManagement::KeyframeManager::Instance().

11.48.3 Member Function Documentation

11.48.3.1 GetActiveAgent()

```
int Gui::Timeline::GetActiveAgent ( ) [inline]
```

Get the active agent ID.

Returns

The ID of the active agent

Definition at line 27 of file timeline.h.

References m_activeAgentId.

11.48.3.2 GetTimeStamp()

```
float Gui::Timeline::GetTimeStamp ( ) [inline]
```

Get the current timestamp.

Returns

The current timestamp

Definition at line 37 of file timeline.h.

References m_timeStamp.

11.48.3.3 mouseReleaseEvent()

```
void Gui::Timeline::mouseReleaseEvent (
            QMouseEvent * event ) [override], [protected]
```

Mouse release event handler.

Parameters

event	The mouse release event

Definition at line 66 of file timeline.cpp.

References KeyframeManagement::KeyframeManager::GetKeyframes(), KeyframeManagement::KeyframeManager::Instance(), m_pixelsPerSecond, m_timeStamp, KeyframeManagement::KeyframeManager::RemoveKeyframe(), Core::Keyframe::TimeStamp, and timeStampSelected().

11.48.3.4 paintEvent()

```
void Gui::Timeline::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

Paint event handler.

Parameters

event	The paint event

Definition at line 25 of file timeline.cpp.

References KeyframeManagement::KeyframeManager::GetKeyframes(), KeyframeManagement::KeyframeManager::Instance(), m_pixelsPerSecond, and m_timeStamp.

11.48.3.5 resizeEvent()

```
void Gui::Timeline::resizeEvent (
            QResizeEvent * event )  [override], [protected]
```

Resize event handler.

Parameters

event	The resize event

Definition at line 116 of file timeline.cpp.

11.48.3.6 timeStampSelected

```
void Gui::Timeline::timeStampSelected (
            float timeStamp )  [signal]
```

Signal that is emitted when a timestamp is selected.

Parameters

timeStamp	The selected timestamp

Referenced by mouseReleaseEvent().

11.48.4 Member Data Documentation

11.48.4.1 m_activeAgentId

`int Gui::Timeline::m_activeAgentId [private]`

ID of the active agent.

Definition at line 69 of file timeline.h.

Referenced by GetActiveAgent().

11.48.4.2 m_pixelsPerSecond

`float Gui::Timeline::m_pixelsPerSecond [private]`

Pixels per second on the timeline.

Definition at line 70 of file timeline.h.

Referenced by mouseReleaseEvent(), and paintEvent().

11.48.4.3 m_timeStamp

`float Gui::Timeline::m_timeStamp [private]`

The current timestamp.

Definition at line 68 of file timeline.h.

Referenced by GetTimeStamp(), mouseReleaseEvent(), and paintEvent().

The documentation for this class was generated from the following files:

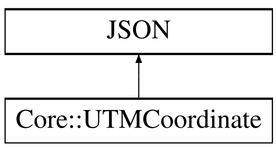
- include/gui/timeline.h
- src/gui/timeline.cpp

11.49 Core::UTMCoordinate Struct Reference

\ A structure that represents a coordinate in the Universal Transverse Mercator coordinate system

`#include <types.h>`

Inheritance diagram for Core::UTMCoordinate:



Public Member Functions

- UTMCoordinate (double northing=0.0, double easting=0.0, int zone=33, bool isNorthHemisphere=true, double meridian=1)
- JSONSTART JSONDOUBLE (Northing)
- JSONSTART JSONDOUBLE (Easting)
- JSONSTART JSONINT (Zone)
- JSONSTART JSONBOOL (IsNorthHemisphere)

Public Attributes

- double Northing
- double Easting
- int Zone
- bool IsNorthHemisphere
- double Meridian

11.49.1 Detailed Description

\ A structure that represents a coordinate in the Universal Transverse Mercator coordinate system

Definition at line 45 of file types.h.

11.49.2 Constructor & Destructor Documentation

11.49.2.1 UTMCoordinate()

```
Core::UTMCoordinate::UTMCoordinate (
            double northing = 0.0,
            double easting = 0.0,
            int zone = 33,
            bool isNorthHemisphere = true,
            double meridian = 1 )  [inline]
```

Definition at line 47 of file types.h.

11.49.3 Member Function Documentation

11.49.3.1 JSONBOOL()

```
JSONSTART Core::UTMCoordinate::JSONBOOL (
            IsNorthHemisphere )
```

11.49.3.2 JSONDOUBLE() [1/2]

```
JSONSTART Core::UTMCoordinate::JSONDOUBLE (
            Easting  )
```

11.49.3.3 JSONDOUBLE() [2/2]

```
JSONSTART Core::UTMCoordinate::JSONDOUBLE (
            Northing  )
```

11.49.3.4 JSONINT()

```
JSONSTART Core::UTMCoordinate::JSONINT (
            Zone  )
```

11.49.4 Member Data Documentation

11.49.4.1 Easting

```
double Core::UTMCoordinate::Easting
```

Definition at line 54 of file types.h.

Referenced by MapManagement::MapManager::CalculateCornerCoordinates(), CoordinateConverter::CoordConv::GeographicToUTM
HeightManagement::HeightManager::UpdateOrigin(), and CoordinateConverter::CoordConv::UTMToGeographic().

11.49.4.2 IsNorthHemisphere

```
bool Core::UTMCoordinate::IsNorthHemisphere
```

Definition at line 56 of file types.h.

Referenced by CoordinateConverter::CoordConv::GeographicToUTM(), and CoordinateConverter::CoordConv::UTMToGeographic().

11.49.4.3 Meridian

```
double Core::UTMCoordinate::Meridian
```

Definition at line 57 of file types.h.

11.49.4.4 Northing

```
double Core::UTMCoordinate::Northing
```

Definition at line 54 of file types.h.

Referenced by MapManagement::MapManager::CalculateCornerCoordinates(), CoordinateConverter::CoordConv::GeographicToUTM HeightManagement::HeightManager::UpdateOrigin(), and CoordinateConverter::CoordConv::UTMToGeographic().

11.49.4.5 Zone

```
int Core::UTMCoordinate::Zone
```

Definition at line 55 of file types.h.

Referenced by CoordinateConverter::CoordConv::GeographicToUTM(), and CoordinateConverter::CoordConv::UTMToGeographic().

The documentation for this struct was generated from the following file:

- include/core/types.h

Chapter 12

File Documentation

12.1 docs/coding_standards.md File Reference

12.2 docs/get_started.md File Reference

12.3 docs/testing_standard.md File Reference

12.4 docs/user_guide.md File Reference

12.5 include/compile_scenario/scenario.h File Reference

```
#include "core/serializer.h"
#include "keyframe_management/keyframe_manager.h"
#include "routemaker/routemaker.h"
#include <algorithm>
#include <memory>
#include <string>
```

Classes

- class CompileScenario::Scenario

  The Scenario class represents a scenario with keyframes and routes.

Namespaces

- namespace CompileScenario

12.6 scenario.h

Go to the documentation of this file.

```
#pragma once

#include "core/serializer.h"
#include "keyframe_management/keyframe_manager.h"
#include "routemaker/routemaker.h"

#include <algorithm>
#include <memory>
#include <string>

namespace CompileScenario
{

    ///
    /// \brief The Scenario class represents a scenario with keyframes and
    /// routes.
    ///
    /// The Scenario class provides functionality for creating a scenario with
    /// keyframes and routes, as well as saving and loading the scenario to and
    /// from file.
    class Scenario : JSON
    {
      public:
        using RouteMap =
            std::map<int, std::vector<std::vector<Core::CartesianCoordinate»>;
        ///
        /// \brief Constructs a new Scenario object with the given name, origin,
        /// and size. \param name The name of the scenario. \param origin The
        /// geographical coordinates of the origin. \param size The size of the
        /// scenario.
        Scenario(std::string name, Core::GeographicalCoordinate origin, int size);

        ///
        /// \brief Compiles the scenario into a map of routes.
        /// \return A map of routes.
        RouteMap& Compile();

        ///
        /// \brief Saves the scenario to a file with the given filename.
        /// \param filename The name of the file to save to.
        void save(std::string filename);

        ///
        /// \brief Loads a scenario from a file with the given filename.
        /// \param filename The name of the file to load from.
        void load(std::string filename);

        inline std::pair<RouteMap::iterator, RouteMap::iterator>
        GetRoutes()
        {
            return std::make_pair<RouteMap::iterator, RouteMap::iterator>(
                m_Routes.begin(), m_Routes.end());
        }

        inline std::pair<std::vector<Core::Agent>::iterator,
                         std::vector<Core::Agent>::iterator>
        GetAgents()
        {
            return std::make_pair<std::vector<Core::Agent>::iterator,
                                  std::vector<Core::Agent>::iterator>(
                m_Agents.begin(), m_Agents.end());
        }

        void AddAgent(Core::Agent newAgent);

        ///
        /// \brief Sets the origin of the scenario to the given geographical
        /// coordinates and size. \param GeoCoord The geographical coordinates
        /// of the origin. \param size The size of the scenario.
        void SetOrigin(Core::GeographicalCoordinate GeoCoord, int size);

      private:
        KeyframeManagement::KeyframeManager& m_KeyframeManager;
        std::vector<Core::Agent> m_Agents;
        RouteMap m_Routes;
        std::unique_ptr<Routemaker::Routemaker> m_Routemaker;
        std::string m_Name;
        Core::GeographicalCoordinate m_Origin;
        int m_Size;

        JSONSTART
        JSONSTRING(m_Name), JSONMEMBER(Core::GeographicalCoordinate, m_Origin),
```

```
            JSONINT(m_Size), JSONMEMBERVECTOR(Core::Agent, m_Agents),
            JSONMEMBER(KeyframeManagement::KeyframeManager,
                       m_KeyframeManager) JSONEND
    };

} // namespace CompileScenario
```

12.7 include/coordinate_converter/coordinate_converter.h File Reference

```
#include "core/types.h"
#include <GeographicLib/Geodesic.hpp>
#include <GeographicLib/LocalCartesian.hpp>
#include <GeographicLib/UTMUPS.hpp>
```

Classes

- class CoordinateConverter::CoordConv

  This is the class that performs coordinate conversions.

Namespaces

- namespace CoordinateConverter

12.8 coordinate_converter.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"
#include <GeographicLib/Geodesic.hpp>
#include <GeographicLib/LocalCartesian.hpp>
#include <GeographicLib/UTMUPS.hpp>

namespace CoordinateConverter
{

    ///
    /// \brief This is the class that performs coordinate conversions
    class CoordConv
    {
      public:
        ///
        /// \brief Sets the origin coordinate to use with relative coordinates
        ///
        /// \param geoCoord Geographical coordinate to be used as the
        /// origin of relative coordinates
        ///
        static void ResetOrigin(Core::GeographicalCoordinate geoCoord,
                                int size);

        ///
        /// \brief Function used to convert a geographical coordinate to a
        /// cartesian coordinate
        ///
        /// \param geoCoord Geograhical coordinate to convert
        /// \return return a cartesian point relative to origin
        static Core::CartesianCoordinate
        GeographicalToCartesian(Core::GeographicalCoordinate geoCoord);

        ///
        /// \biref Function used to convert a cartesian coordinate to a
```

```
        /// geograpical coordinate
        ///
        /// \param cartCoord Cartesian coordinate to convert
        /// \return return a geographical point relative to origin and the
        /// cartesian coordinates.
        static Core::GeographicalCoordinate
        CartesianToGeographical(Core::CartesianCoordinate cartCoord);

        ///
        /// \return The geographical coordinates to origin.
        static Core::GeographicalCoordinate GetOrigin();

        ///
        /// \brief Function used to convert a coordinate in a symmetric
        /// coordinate system to a coordinate in an asymmetric coordinate system
        ///
        /// \param symmetric Cartesian coordinate in a symmetric coordinate
        /// system \return The asymmetric coordinate corresponds to the
        /// symmetric coordinate
        static Core::CartesianCoordinate
        SymmetricToAsymmetric(Core::CartesianCoordinate symmetric);

        ///
        /// \brief Function used to convert a coordinate in an asymmetric
        /// cooridnate system to a coordinate in a symmetric coordinate system
        /// \param asymmetric Cartesian coordinate in an asymmetric coordinate
        /// system \return The symmetric coordinate corresponds to the
        /// asymmetric coordinate
        static Core::CartesianCoordinate
        AsymmetricToSymmetric(Core::CartesianCoordinate asymmetric);

        ///
        /// \brief Function used to convert a geographical coordinate to a UTM
        /// coordinate \param GeoCoord Geographical coordinate \return UTM
        /// coordinate corresponds to the geographical coordinate
        static Core::UTMCoordinate
        GeographicToUTM(Core::GeographicalCoordinate GeoCoord);

        ///
        /// \brief Function used to convert a UTM coordinate to a geographical
        /// coordinate \param UTMCoord UTM coordinate \return Geographical
        /// coordinate corresponds to the UTM coordinate
        static Core::GeographicalCoordinate
        UTMToGeographic(Core::UTMCoordinate UTMCoord);

        static inline int
        GetSize()
        {
            return GetInstance().m_Size;
        }

      private:
        ///
        /// \brief The constructor is made private to adhere to the singleton
        /// pattern
        CoordConv() : m_OriginGeographical(0, 0) {}

        /// \brief Get the single instance of CoordConv.
        ///
        /// \return The single instance of CoordConv.
        static CoordConv&
        GetInstance()
        {
            static CoordConv instance;
            return instance;
        }

      private:
        Core::GeographicalCoordinate m_OriginGeographical;
        GeographicLib::LocalCartesian m_Origin;
        int m_Size;
    };

} // namespace CoordinateConverter
```

12.9 include/core/serializer.h File Reference

```
#include "rapidjson/document.h"
#include <iostream>
#include <map>
```

```
#include <memory>
#include <string>
#include <type_traits>
#include <vector>
```

Classes

- struct Json::ISProperty

  Serializing and deserializing (persistent values) requires recflection which is a way for the programmer to ensure that the data you serialize will get back to the place you want it to be when you deserialize it later.
- class Json::ISValue

  Rflection is made possible by the help of the ISValue class and the type classes.
- class Json::ISInt

  Implementation for integers.
- class Json::ISFloat

  Implementation for floats.
- class Json::ISDouble

  Implementation for doubles.
- class Json::ISBool

  Implementation for bools.
- class Json::ISString

  Implementation for strings.
- class Json::ISObject< T >

  Implementation for objects.
- class Json::ISObjectVector< T >

  Implementation for a vector with objects.
- class Json::ISObjVecVec< T >

  Implementation for a vector with vectors with objects.
- class Json::ISMemVecVec< T >

  Implementation for a vector with vectors with members.
- class Json::ISMember< T >

  Implementation for Members.
- class Json::ISMemberVector< T >

  Implementation for a vector with members.
- class Json::ISIntVector

  Implementation for a vector with integers.
- class Json::ISFloatVector

  Implementation for a vector with floats.
- class Json::ISDoubleVector

  Implementation for a vector with doubles.
- class Json::ISConstructors

  Implemented for future expansion.

Namespaces

- namespace Json

Macros

- #define JSON

    Macros To serialize an object you need to have the GetProperty() function in the object.
- #define JSONSTART
- #define JSONINT(m)
- #define JSONINTVECTOR(m)
- #define JSONFLOAT(m)
- #define JSONFLOATVECTOR(m)
- #define JSONDOUBLE(m)
- #define JSONDOUBLEVECTOR(m)
- #define JSONBOOL(m)
- #define JSONSTRING(m)
- #define JSONOBJECT(T, m)
- #define JSONOBJECTVECTOR(T, m)
- #define JSONOBJVECVEC(T, m)
- #define JSONMEMBER(T, m)
- #define JSONMEMBERVECTOR(T, m)
- #define JSONMEMVECVEC(T, m)
- #define JSONEND

Typedefs

- using Json::ISValuePtr = std::shared_ptr< ISValue >
- using Json::ISValues = std::vector< ISValuePtr >
- using Json::ISProperties = std::vector< ISProperty >

    ISProperties is a vector with ISProperty.
- using Json::ISIV = std::vector< int >
- using Json::ISFV = std::vector< float >
- using Json::ISDV = std::vector< double >

Functions

- void Json::serialize (std::string filename, ISValue *p)

    Function to start serializing an onbject.
- void Json::deserialize (std::string filename, ISValue *p)

    Function to start deserializing a file.

Variables

- bool debug

12.9.1 Macro Definition Documentation

12.9.1.1 JSON

```
#define JSON
```

Value:
```
public      \
  Json::ISValue
```

Macros To serialize an object you need to have the GetProperty() function in the object.

This is complex for each application programmer to execute so therefore these macros have benn implemented. Each macro start with JSONSTART then each of the types you want to serialize and to end the macro you write JSONEND.

Definition at line 525 of file serializer.h.

12.9.1.2 JSONBOOL

```
#define JSONBOOL(
                m )
```

Value:
```
    {                                             \
#       m, std::make_shared < Json::ISBool>(m) \
    }
```

Definition at line 557 of file serializer.h.

12.9.1.3 JSONDOUBLE

```
#define JSONDOUBLE(
                m )
```

Value:
```
    {                                               \
#       m, std::make_shared < Json::ISDouble>(m)  \
    }
```

Definition at line 549 of file serializer.h.

12.9.1.4 JSONDOUBLEVECTOR

```
#define JSONDOUBLEVECTOR(
                m )
```

Value:
```
    {                                                     \
#       m, std::make_shared < Json::ISDoubleVector>(m)  \
    }
```

Definition at line 553 of file serializer.h.

12.9.1.5 JSONEND

```
#define JSONEND
```

Value:

```
    }            \
    ;            \
    return prop; \
    }            \
    ;
```

Definition at line 590 of file serializer.h.

12.9.1.6 JSONFLOAT

```
#define JSONFLOAT(
                m )
```

Value:

```
    {                                              \
#       m, std::make_shared < Json::ISFloat>(m)  \
    }
```

Definition at line 541 of file serializer.h.

12.9.1.7 JSONFLOATVECTOR

```
#define JSONFLOATVECTOR(
                m )
```

Value:

```
    {                                                    \
#       m, std::make_shared < Json::ISFloatVector>(m)  \
    }
```

Definition at line 545 of file serializer.h.

12.9.1.8 JSONINT

```
#define JSONINT(
                m )
```

Value:

```
    {                                            \
#       m, std::make_shared < Json::ISInt>(m)  \
    }
```

Definition at line 533 of file serializer.h.

12.9.1.9 JSONINTVECTOR

```
#define JSONINTVECTOR(
                m )
```

Value:

```
    {                                                \
#       m, std::make_shared < Json::ISIntVector>(m)  \
    }
```

Definition at line 537 of file serializer.h.

12.9.1.10 JSONMEMBER

```
#define JSONMEMBER(
                T,
                m )
```

Value:

```
    {                                                  \
#       m, std::make_shared < Json::ISMember < T»(m)  \
    }
```

Definition at line 577 of file serializer.h.

12.9.1.11 JSONMEMBERVECTOR

```
#define JSONMEMBERVECTOR(
                T,
                m )
```

Value:

```
    {                                                        \
#       m, std::make_shared < Json::ISMemberVector < T»(m)  \
    }
```

Definition at line 581 of file serializer.h.

12.9.1.12 JSONMEMVECVEC

```
#define JSONMEMVECVEC(
                T,
                m )
```

Value:

```
    {                                                     \
#       m, std::make_shared < Json::ISMemVecVec < T»(m)  \
    }
```

Definition at line 585 of file serializer.h.

12.9.1.13 JSONOBJECT

```
#define JSONOBJECT(
            T,
            m )
```

Value:

```
    {                                                    \
#       m, std::make_shared < Json::ISObject < T»(m) \
    }
```

Definition at line 565 of file serializer.h.

12.9.1.14 JSONOBJECTVECTOR

```
#define JSONOBJECTVECTOR(
            T,
            m )
```

Value:

```
    {                                                          \
#       m, std::make_shared < Json::ISObjectVector < T»(m) \
    }
```

Definition at line 569 of file serializer.h.

12.9.1.15 JSONOBJVECVEC

```
#define JSONOBJVECVEC(
            T,
            m )
```

Value:

```
    {                                                       \
#       m, std::make_shared < Json::ISObjVecVec < T»(m) \
    }
```

Definition at line 573 of file serializer.h.

12.9.1.16 JSONSTART

```
#define JSONSTART
```

Value:

```
    virtual Json::ISProperties GetProperty() \
    {                                        \
        Json::ISProperties prop = {
```

Definition at line 529 of file serializer.h.

12.9.1.17 JSONSTRING

```
#define JSONSTRING(
              m )
```

Value:

```
    {                                             \
#       m, std::make_shared < Json::ISString>(m) \
    }
```

Definition at line 561 of file serializer.h.

12.9.2 Variable Documentation

12.9.2.1 debug

```
bool debug  [extern]
```

Definition at line 12 of file serializer.cpp.

Referenced by Json::ISDouble::ToDom(), and Json::ISMemberVector< T >::ToDom().

12.10 serializer.h

Go to the documentation of this file.

```
#pragma once

#include "rapidjson/document.h"
#include <iostream>
#include <map>
#include <memory>
#include <string>
#include <type_traits>
#include <vector>

extern bool debug;

namespace Json
{
    // IS... IntroSpection
    class ISValue;
    using ISValuePtr = std::shared_ptr<ISValue>;
    using ISValues = std::vector<ISValuePtr>;

///
///\brief Serializing and deserializing (persistent values) requires recflection which is a way for the programmer to ensure that
///the data you serialize will get back to the place you want it to be when you deserialize it later.
///As this is not supported by C++ this is implemented by the ISProperty structure with the ISValue helper classes. The ISValue keeps the references
///to the actual values in the application. The ISProperty is the collection of all the application data.
///
    struct ISProperty
    {
        std::string name;
        ISValuePtr value;
    };
///
///\brief ISProperties is a vector with ISProperty.
///
    using ISProperties = std::vector<ISProperty>;
///
```

```
///\brief Rflection is made possible by the help of the ISValue class and the type classes. Each type needs their own implementation for
///reflection to work. At the moment only JSON is supported by this library.
///Making the library work for other format than JSON would require implementing each type again for the new format by in theory would not
///impact the application programmers at all
///
    class ISValue
    {
      public:
///
///\brief GetProperty enables the serializer to deal with composite type like objects and members.
///
        virtual ISProperties
        GetProperty()
        {
            return ISProperties{};
        };
///
///\brief For future expansion.
///
        virtual void CreateObject(){};
///
///\brief For future expansion
///
        virtual rapidjson::Value
        GetName(rapidjson::Document& d)
        {
///
///\brief Typeid is mostly implemented for future expansion, but it helps with making the JSON file more readable for humans.
///
            rapidjson::Value tid;
            tid.SetString(typeid(*this).name(), d.GetAllocator());
            return tid;
        };
///
///\brief ToDom is the function that enables the serializer to take data from the application to the JSON file.
///
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
///
///\brief FromDom is the function that enables the serializer to get data out of the JSON file and put it in the application.
///
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

///
///\brief Implementation for integers
///
    class ISInt : public ISValue
    {
        int& value;

      public:
        ISInt(int& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

///
///\brief Implementation for floats
///
    class ISFloat : public ISValue
    {
        float& value;

      public:
        ISFloat(float& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

///
///\brief Implementation for doubles
///
    class ISDouble : public ISValue
    {
        double& value;

      public:
        ISDouble(double& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

```

```
///
///\brief Implementation for bools
///
    class ISBool : public ISValue
    {
        bool& value;

      public:
        ISBool(bool& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

///
///\brief Implementation for strings
///
    class ISString : public ISValue
    {
        std::string& value;

      public:
        ISString(std::string& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

///
///\brief Implementation for objects
///
    template<typename T>
    class ISObject : public ISValue
    {
        std::shared_ptr<T>& value;

      public:
        ISObject(std::shared_ptr<T>& v) : value(v){};
        virtual rapidjson::Value GetName(rapidjson::Document& d);
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
        void CreateObject();
    };

    template<typename T>
    rapidjson::Value
    ISObject<T>::ToDom(rapidjson::Document& d)
    {
        if (value != nullptr)
            return value->ToDom(d);
        else
            return rapidjson::Value("");
    };

    template<typename T>
    void
    ISObject<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        if (v.IsObject()) {
            CreateObject();
            value->FromDom(v, d);
        }
    };

    template<typename T>
    rapidjson::Value
    ISObject<T>::GetName(rapidjson::Document& d)
    {
        rapidjson::Value tid;
        tid.SetString(typeid(T).name(), d.GetAllocator());
        return tid;
    }

    template<typename T>
    void
    ISObject<T>::CreateObject()
    {
        value = std::make_shared<T>();
    }

///
///\brief Implementation for a vector with objects
///
    template<typename T>
    class ISObjectVector : public ISValue
    {
        std::vector<std::shared_ptr<T>>& value;

      public:
```

```
        ISObjectVector(std::vector<std::shared_ptr<T»& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    template<typename T>
    rapidjson::Value
    ISObjectVector<T>::ToDom(rapidjson::Document& d)
    {
        rapidjson::Value a;
        a.SetArray();
        for (auto& element : value) {
            rapidjson::Value v = element->ToDom(d);
            a.PushBack(v, d.GetAllocator());
        }
        return a;
    }

    template<typename T>
    void
    ISObjectVector<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        for (rapidjson::SizeType i = 0; i < v.Size();
             i++) { // rapidjson uses SizeType instead of size_t.
            std::shared_ptr<T> cv = std::make_shared<T>();
            cv->FromDom(v[i], d);
            value.push_back(cv);
        }
    }

///
///\brief Implementation for a vector with vectors with objects
///
    template<typename T>
    class ISObjVecVec : public ISValue
    {
        std::vector<std::vector<std::shared_ptr<T>»& value;

      public:
        ISObjVecVec(std::vector<std::vector<std::shared_ptr<T>»& v)
            : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    template<typename T>
    rapidjson::Value
    ISObjVecVec<T>::ToDom(rapidjson::Document& d)
    {
        rapidjson::Value outer;
        outer.SetArray();
        for (auto& outer_element : value) {
            rapidjson::Value inner;
            inner.SetArray();
            for (auto& inner_element : outer_element) {
                rapidjson::Value v = inner_element->ToDom(d);
                inner.PushBack(v, d.GetAllocator());
            }
            outer.PushBack(inner, d.GetAllocator());
        }
        return outer;
    }

    template<typename T>
    void
    ISObjVecVec<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        for (rapidjson::SizeType i = 0; i < v.Size(); i++) {
            std::vector<std::shared_ptr<T» line;
            for (rapidjson::SizeType j = 0; j < v[i].Size(); j++) {
                std::shared_ptr<T> cv = std::make_shared<T>();
                cv->FromDom(v[i][j], d);
                line.push_back(cv);
            }
            value.push_back(line);
        }
    }

///
///\brief Implementation for a vector with vectors with members
///
    template<typename T>
    class ISMemVecVec : public ISValue
    {
        std::vector<std::vector<T»& value;

      public:
```

```
        ISMemVecVec(std::vector<std::vector<T»& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    template<typename T>
    rapidjson::Value
    ISMemVecVec<T>::ToDom(rapidjson::Document& d)
    {
        rapidjson::Value outer;
        outer.SetArray();
        for (auto& outer_element : value) {
            rapidjson::Value inner;
            inner.SetArray();
            for (auto& inner_element : outer_element) {
                rapidjson::Value v = inner_element.ToDom(d);
                inner.PushBack(v, d.GetAllocator());
            }
            outer.PushBack(inner, d.GetAllocator());
        }
        return outer;
    }

    template<typename T>
    void
    ISMemVecVec<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        for (rapidjson::SizeType i = 0; i < v.Size(); i++) {
            std::vector<T> line;
            for (rapidjson::SizeType j = 0; j < v[i].Size(); j++) {
                T cv;
                cv.FromDom(v[i][j], d);
                line.push_back(cv);
            }
            value.push_back(line);
        }
    }

///
///\brief Implementation for Members
///
    template<typename T>
    class ISMember : public ISValue
    {
        T& value;

      public:
        ISMember(T& v) : value(v){};
        virtual rapidjson::Value GetName(rapidjson::Document& d);
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
        void CreateObject();
    };

    template<typename T>
    rapidjson::Value
    ISMember<T>::ToDom(rapidjson::Document& d)
    {
        return value.ToDom(d);
    };

    template<typename T>
    void
    ISMember<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        value.FromDom(v, d);
    };

    template<typename T>
    rapidjson::Value
    ISMember<T>::GetName(rapidjson::Document& d)
    {
        rapidjson::Value tid;
        tid.SetString(typeid(T).name(), d.GetAllocator());
        return tid;
    }

    template<typename T>
    void
    ISMember<T>::CreateObject()
    {
        assert(false);
    }

///
///\brief Implementation for a vector with members
///
```

```
    template<typename T>
    class ISMemberVector : public ISValue
    {
        std::vector<T>& value;

      public:
        ISMemberVector(const ISMemberVector<T>&) { assert(false); };

        ISMemberVector(std::vector<T>& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    template<typename T>
    rapidjson::Value
    ISMemberVector<T>::ToDom(rapidjson::Document& d)
    {
        rapidjson::Value a;
        a.SetArray();
        debug = true;
        for (auto& element : value) {
            rapidjson::Value v = element.ToDom(d);
            a.PushBack(v, d.GetAllocator());
        }
        debug = false;
        return a;
    }

    template<typename T>
    void
    ISMemberVector<T>::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        for (rapidjson::SizeType i = 0; i < v.Size();
             i++) { // rapidjson uses SizeType instead of size_t.
            T cv;
            cv.FromDom(v[i], d);
            value.push_back(cv);
        }
    }

    using ISIV = std::vector<int>;

///
///\brief Implementation for a vector with integers
///
    class ISIntVector : public ISValue
    {
        std::vector<int>& value;

      public:
        ISIntVector(ISIV& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    using ISFV = std::vector<float>;

///
///\brief Implementation for a vector with floats
///
    class ISFloatVector : public ISValue
    {
        std::vector<float>& value;

      public:
        ISFloatVector(ISFV& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    using ISDV = std::vector<double>;

///
///\brief Implementation for a vector with doubles
///
    class ISDoubleVector : public ISValue
    {
        std::vector<double>& value;

      public:
        ISDoubleVector(ISDV& v) : value(v){};
        virtual rapidjson::Value ToDom(rapidjson::Document& d);
        virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    };

    // using ISBV = std::vector<bool>;
    // class ISBoolVector: public ISValue {
```

```
    // std::vector<bool> &value;
    // public:
    // ISBoolVector (ISBV &v) : value(v) {};
    // virtual rapidjson::Value ToDom(rapidjson::Document& d);
    // virtual void FromDom(rapidjson::Value& v, rapidjson::Document& d);
    //};

///
///\brief Implemented for future expansion
///
    class ISConstructors
    { // OBS OBS this is an implementation of the Singleton design pattern.
      public:
        static ISConstructors&
        GetInstance()
        {
            static ISConstructors instance; // Guaranteed to be destroyed.
                                            // Instantiated on first use.
            return instance;
        }

      private:
        ISConstructors(){}; // Constructor? (the {} brackets) are needed here.

        std::map<std::string, Json::ISValuePtr (*)()> m_TheRegistry;

      public:
        ISConstructors(const ISConstructors&) = delete;
        void operator=(const ISConstructors&) = delete;

        int AddConstructor(std::string name, ISValuePtr (*creator)());
        ISValuePtr GetObject(std::string name);
    };
///
///\brief Function to start serializing an onbject.
///
///\param std::string filename
///Name of the file you want to store the application data in.
///
///\param ISValue* p
///A pointer to the object you want to serialize.
///
    void serialize(std::string filename, ISValue* p);
///
///\brief Function to start deserializing a file
///
///\param std::string filename
///Name of the file you want to extract data from.
///
///\param ISValue* p
///A pointer to the top object so it know where to start.
///
    void deserialize(std::string filename, ISValue* p);
///
///\brief Macros
///To serialize an object you need to have the GetProperty() function in the object.
///This is complex for each application programmer to execute so therefore these macros have benn implemented.
///Each macro start with JSONSTART then each of the types you want to serialize and to end the macro you write JSONEND.
///
#define JSON \
  public     \
    Json::ISValue

#define JSONSTART                             \
    virtual Json::ISProperties GetProperty() \
    {                                         \
        Json::ISProperties prop = {
#define JSONINT(m)                              \
    {                                           \
#        m, std::make_shared < Json::ISInt>(m) \
    }
#define JSONINTVECTOR(m)                              \
    {                                                 \
#        m, std::make_shared < Json::ISIntVector>(m) \
    }
#define JSONFLOAT(m)                              \
    {                                             \
#        m, std::make_shared < Json::ISFloat>(m) \
    }
#define JSONFLOATVECTOR(m)                              \
    {                                                   \
#        m, std::make_shared < Json::ISFloatVector>(m) \
    }
#define JSONDOUBLE(m)                              \
    {                                              \
```

```
#         m, std::make_shared < Json::ISDouble>(m) \
    }
#define JSONDOUBLEVECTOR(m)                              \
    {                                                    \
#         m, std::make_shared < Json::ISDoubleVector>(m) \
    }
#define JSONBOOL(m)                              \
    {                                            \
#         m, std::make_shared < Json::ISBool>(m) \
    }
#define JSONSTRING(m)                              \
    {                                              \
#         m, std::make_shared < Json::ISString>(m) \
    }
#define JSONOBJECT(T, m)                                \
    {                                                   \
#         m, std::make_shared < Json::ISObject < T»(m) \
    }
#define JSONOBJECTVECTOR(T, m)                                \
    {                                                         \
#         m, std::make_shared < Json::ISObjectVector < T»(m) \
    }
#define JSONOBJVECVEC(T, m)                                \
    {                                                      \
#         m, std::make_shared < Json::ISObjVecVec < T»(m) \
    }
#define JSONMEMBER(T, m)                                \
    {                                                   \
#         m, std::make_shared < Json::ISMember < T»(m) \
    }
#define JSONMEMBERVECTOR(T, m)                                \
    {                                                         \
#         m, std::make_shared < Json::ISMemberVector < T»(m) \
    }
#define JSONMEMVECVEC(T, m)                                \
    {                                                      \
#         m, std::make_shared < Json::ISMemVecVec < T»(m) \
    }

#define JSONEND  \
    }            \
    ;            \
    return prop; \
    }            \
    ;

} // namespace Json
```

12.11 include/core/types.h File Reference

```
#include "core/serializer.h"
```

Classes

- struct Core::CartesianCoordinate

  A structure that represents a cartesian coordinate.
- struct Core::GeographicalCoordinate

  A structure that represents a geographic coordinate.
- struct Core::UTMCoordinate

  \ A structure that represents a coordinate in the Universal Transverse Mercator coordinate system
- struct Core::Keyframe

  A structure representing an agent's position in cartesian space at a given point in time.
- struct Core::Agent

Namespaces

- namespace Core

12.12 types.h

Go to the documentation of this file.

```
#pragma once

#include "core/serializer.h"

namespace Core
{

    ///
    /// \brief A structure that represents a cartesian coordinate
    ///
    struct CartesianCoordinate : JSON
    {
        CartesianCoordinate(double x = 0.0, double y = 0.0, double z = 0.0)
            : X(x), Y(y), Z(z)
        {}

        double X;
        double Y;
        double Z;

        JSONSTART
        JSONDOUBLE(X), JSONDOUBLE(Y), JSONDOUBLE(Z) JSONEND
    };

    ///
    /// \brief A structure that represents a geographic coordinate
    ///
    struct GeographicalCoordinate : JSON
    {
        GeographicalCoordinate(double lat, double lon)
            : Latitude(lat), Longitude(lon)
        {}

        double Latitude;
        double Longitude;

        JSONSTART
        JSONDOUBLE(Latitude), JSONDOUBLE(Longitude) JSONEND
    };

    ///
    /// \ A structure that represents a coordinate in the Universal Transverse
    /// Mercator coordinate system
    ///
    struct UTMCoordinate : JSON
    {
        UTMCoordinate(double northing = 0.0, double easting = 0.0,
                      int zone = 33, bool isNorthHemisphere = true,
                      double meridian = 1)
            : Northing(northing), Easting(easting), Zone(zone),
              IsNorthHemisphere(isNorthHemisphere), Meridian(meridian)
        {}

        double Northing, Easting;
        int Zone;
        bool IsNorthHemisphere;
        double Meridian;

        JSONSTART
        JSONDOUBLE(Northing), JSONDOUBLE(Easting), JSONINT(Zone),
            JSONBOOL(IsNorthHemisphere), JSONDOUBLE(Meridian) JSONEND
    };

    ///
    /// \brief A structure representing an agent's position in cartesian space
    /// at a given point in time
    ///
    struct Keyframe : JSON
    {
        Keyframe() : AgentId(0), TimeStamp(0), Position(0, 0, 0) {}

        Keyframe(int agentId, float timeStamp, CartesianCoordinate position)
            : AgentId(agentId), TimeStamp(timeStamp), Position(position)
        {}

        int AgentId;
        float TimeStamp;
        CartesianCoordinate Position;

        JSONSTART
        JSONINT(AgentId), JSONFLOAT(TimeStamp),
            JSONMEMBER(CartesianCoordinate, Position) JSONEND
```

```
    };

    struct Agent : JSON
    {
        Agent(int id = 0, std::string name = "Untitled Agent",
              std::string color = "#FFFFFF")
            : Id(id), Name(name), Color(color)
        {}

        int Id;
        std::string Name;
        std::string Color;

        JSONSTART
        JSONINT(Id), JSONSTRING(Name), JSONSTRING(Color) JSONEND
    };

} // namespace Core
```

12.13 include/gui/action.h File Reference

```
#include <QAction>
```

Classes

- class Gui::Action

  Small wrapper around QAction.

Namespaces

- namespace Gui

12.14 action.h

Go to the documentation of this file.

```
#pragma once

#include <QAction>

namespace Gui
{

    /// \brief Small wrapper around QAction
    ///
    /// A tiny wrapper class around QAction that simply provides constructor
    /// arguments to add on-click functionality and keyboard shortcuts.
    class Action : public QAction
    {
      public:
        /// \brief Constructs the Action widget.
        /// \param parent The parent of the Action widget.
        /// \param label The label to be displayed in the action.
        /// \param onClick A function to call when the action is clicked.
        /// \param shortcut A keyboard shortcut to activate the action.
        ///
        /// Typical usage:
        /// \code{.cpp}
        /// Action* openAction = new Action(
        ///     parent, QString::fromUtf8("Open..."),
        ///     []() {
        ///         QString fileName = QFileDialog::getOpenFileName(
        ///             nullptr, QString::fromUtf8("Open Image"),
        ///             QDir::currentPath(),
        ///             QString::fromUtf8("Image Files (*.png *.jpg *.bmp)"));
        ///         qInfo() « "File: " « fileName;
```

```
        ///     },
        ///     QKeySequence::Open);
        /// \endcode
        Action(QWidget* parent, const QString& label, void (*onClick)(void),
               const QKeySequence& shortcut = QKeySequence::UnknownKey);
    };

} // namespace Gui
```

12.15 include/gui/agent_controls.h File Reference

```
#include "core/types.h"
#include "gui/color_box.h"
#include <QComboBox>
#include <QFrame>
#include <QGridLayout>
#include <QPushButton>
```

Classes

- class Gui::AgentControls

Namespaces

- namespace Gui

12.16 agent_controls.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"
#include "gui/color_box.h"

#include <QComboBox>
#include <QFrame>
#include <QGridLayout>
#include <QPushButton>

namespace Gui
{

    class AgentControls : public QFrame
    {
        Q_OBJECT
      public:
        explicit AgentControls(QWidget* parent = nullptr);

      signals:
        void AddAgent();
        void AgentChanged(std::pair<std::vector<Core::Agent>::iterator,
                                    std::vector<Core::Agent>::iterator>);
        void ActiveAgentChanged(int);

      public slots:
        void UpdateAgents(std::pair<std::vector<Core::Agent>::iterator,
                                    std::vector<Core::Agent>::iterator>);
        void SetActiveAgentIndex(int index);
        void SyncColor();

      private slots:
        void SetAgentColor(QColor color);

```

```
    private:
      QGridLayout* m_Layout;

      QComboBox* m_ActiveAgentComboBox;
      ColorBox* m_ActiveAgentColorBox;
      QPushButton* m_NewAgentButton;

      int m_ActiveAgentIndex;

      std::pair<std::vector<Core::Agent>::iterator,
                std::vector<Core::Agent>::iterator>
          m_Agents;
  };

} // namespace Gui
```

12.17 include/gui/color_box.h File Reference

```
#include <QColorDialog>
#include <QPushButton>
```

Classes

- class Gui::ColorBox

Namespaces

- namespace Gui

12.18 color_box.h

Go to the documentation of this file.

```
#pragma once

#include <QColorDialog>
#include <QPushButton>

namespace Gui
{

  class ColorBox : public QPushButton
  {
      Q_OBJECT
    public:
      explicit ColorBox(QWidget* parent = nullptr);

    signals:
      void ColorUpdated(QColor color);

    protected:
      void paintEvent(QPaintEvent* event) override;
      void mousePressEvent(QMouseEvent* event) override;

    public slots:
      void UpdateColor(QColor color);

    private slots:
      void SelectColor();

    private:
      QColor m_Color;
      QColorDialog* m_ColorDialog;
  };

} // namespace Gui
```

12.19 include/gui/keyframe_controls.h File Reference

```
#include "gui/keyframe_list.h"
#include <QFrame>
#include <QPushButton>
```

Classes

- class Gui::KeyframeControls

Namespaces

- namespace Gui

12.20 keyframe_controls.h

Go to the documentation of this file.

```
#pragma once

#include "gui/keyframe_list.h"

#include <QFrame>
#include <QPushButton>

namespace Gui
{

    class KeyframeControls : public QFrame
    {
        Q_OBJECT
      public:
        explicit KeyframeControls(QWidget* parent = nullptr);

      signals:
        void DeleteSelectedKeyframes();

      private:
        KeyframeList* m_KeyframeList;
        QPushButton* m_DeleteKeyframesButton;

        QGridLayout* m_Layout;
    };

} // namespace Gui
```

12.21 include/gui/keyframe_list.h File Reference

```
#include <QListWidget>
#include <QVBoxLayout>
```

Classes

- class Gui::KeyframeList

Namespaces

- namespace Gui

12.22 keyframe_list.h

Go to the documentation of this file.

```
#pragma once

#include <QListWidget>
#include <QVBoxLayout>

namespace Gui
{

    class KeyframeList : public QListWidget
    {
        Q_OBJECT

      public:
        KeyframeList(QWidget* parent = nullptr);

      public slots:
        void Update();
        void DeleteSelected();

      private:
        QVBoxLayout* m_Layout;
    };

} // namespace Gui
```

12.23 include/gui/launcher.h File Reference

```
#include <QVBoxLayout>
#include <QWidget>
```

Classes

- class Gui::Launcher

  The launcher widget used to launch scenarios.

Namespaces

- namespace Gui

12.24 launcher.h

Go to the documentation of this file.

```
#pragma once

#include <QVBoxLayout>
#include <QWidget>

namespace Gui
{
    /// \brief The launcher widget used to launch scenarios.
    ///
    /// Contains the graphical functionality to launch scenarios.
    class Launcher : public QWidget
    {
      public:
        /// \brief Constructs the launcher widget.
        /// \param parent The parent of the launcher widget.
        Launcher(QWidget* parent = nullptr);

        /// \brief Destructs the launcher widget.
        ~Launcher();

      private:
        /// \brief The layout of the launcher widget.
        QVBoxLayout* m_Layout;
    };
} // namespace Gui
```

12.25 include/gui/main_content.h File Reference

```
#include "compile_scenario/scenario.h"
#include "gui/sidebar.h"
#include "gui/tab_widget.h"
#include <QGridLayout>
#include <QWidget>
```

Classes

- class Gui::MainContent

  The main content of the main window.

Namespaces

- namespace Gui

12.26 main_content.h

Go to the documentation of this file.

```
#pragma once

#include "compile_scenario/scenario.h"
#include "gui/sidebar.h"
#include "gui/tab_widget.h"
#include <QGridLayout>
#include <QWidget>

namespace Gui
{
    /// \brief The main content of the main window
```

```
    ///
    /// The main content of the main window essentially contains everything
    /// except the menu bar. It exists as a separate class to make the main
    /// window class more concise.
    class MainContent : public QWidget
    {
        Q_OBJECT
      public:
        /// \brief Constructs the main content widget.
        /// \param parent The parent widget of the main content.
        MainContent(QWidget* parent = nullptr);

      private:
        /// \brief The layout of the main content
        ///
        /// The main content uses a grid layout to easily be able to cover the
        /// available space in the window.
        QGridLayout* m_Layout;

        /// \brief The sidebar of the main content.
        ///
        /// The sidebar of the main content exists to provide the user access to
        /// tools related to the active tab in the tab widget.
        Sidebar* m_Sidebar;

        /// \brief The tab widget of the main content.
        ///
        /// This widget is responsible for containing the core functionality of
        /// Hivemind; planning, simulating and launching. They are separated in
        /// their own tabs as a user should only need to access one of these at
        /// any point in time.
        TabWidget* m_TabWidget;
    };
} // namespace Gui
```

12.27 include/gui/main_window.h File Reference

```
#include "gui/main_content.h"
#include "gui/map_dialog.h"
#include "gui/menu_bar.h"
#include <QMainWindow>
```

Classes

- class Gui::MainWindow

  Handles the main window of Hivemind.

Namespaces

- namespace Gui

12.28 main_window.h

Go to the documentation of this file.

```
#pragma once

#include "gui/main_content.h"
#include "gui/map_dialog.h"
#include "gui/menu_bar.h"

#include <QMainWindow>

namespace Gui
```

```
{

    /// \brief Handles the main window of Hivemind
    ///
    /// This class is responsible for handling the main window of Hivemind,
    /// which contains the core functionality such as scenario editing,
    /// simulation and launching.
    class MainWindow : public QMainWindow
    {
        Q_OBJECT

      public:
        /// \brief Constructs the main window
        ///
        /// \param parent The parent widget of main window
        MainWindow(QWidget* parent = nullptr);

        /// \brief Descructs the main window
        ~MainWindow();

      signals:
        void ScenarioCompiled(
            std::pair<CompileScenario::Scenario::RouteMap::iterator,
                      CompileScenario::Scenario::RouteMap::iterator>);
        void ScenarioLoaded();
        void AgentAdded(std::pair<std::vector<Core::Agent>::iterator,
                                  std::vector<Core::Agent>::iterator>);
        void SyncAgentColor();

      private:
        void ConnectSlotsAndSignals();

      private slots:
        void SaveScenario(const std::string& filepath);
        void LoadScenario(const std::string& filepath);
        void UpdateScenario(float, float, float);
        void CompileScenario();
        void CreateNewAgent();

      private:
        /// \brief The menu bar of the main window.
        MenuBar* m_MenuBar;

        /// \brief The main content of the main window. Basically all content
        /// other than the menubar.
        MainContent* m_MainContent;

        std::shared_ptr<CompileScenario::Scenario> m_Scenario;
        MapDialog* m_ScenarioSettingsDialog;
    };

} // namespace Gui
```

12.29 include/gui/map_dialog.h File Reference

```
#include <QDialog>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>
```

Classes

- class Gui::MapDialog

  The MapDialog class represents a dialog window for inputting map data.

Namespaces

- namespace Gui

12.30 map_dialog.h

Go to the documentation of this file.

```
#pragma once

#include <QDialog>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>

namespace Gui
{
    ///
    /// \brief The MapDialog class represents a dialog window for inputting map
    /// data.
    ///
    class MapDialog : public QDialog
    {
        Q_OBJECT
      public:
        ///
        /// \brief Constructs a new MapDialog object.
        /// \param parent The parent widget of the dialog.
        ///
        MapDialog(QWidget* parent = nullptr);

      signals:
        ///
        /// \brief Signal emitted when data is ready to be sent.
        /// \param data The data to be sent.
        ///
        void SendData(const QString& data);

        ///
        /// \brief Signal emitted when the dialog has finished.
        ///
        void Finished();

        ///
        /// \brief Signal emitted when map data is ready to be processed.
        /// \param latitude The latitude coordinate of the map data.
        /// \param longitude The longitude coordinate of the map data.
        /// \param size The size of the map data.
        ///
        void MapDataReady(float latitude, float longitude, float size);

      public slots:

        ///
        /// \brief Slot called when the user finishes input and submits the
        /// data.
        ///
        void Finish();

      private:
        QLineEdit* m_LatitudeCoordInput;
        QLineEdit* m_LongitudeCoordInput;
        QLineEdit* m_SizeInput;
    };
} // namespace Gui
```

12.31 include/gui/map_viewer.h File Reference

```
#include "core/types.h"
#include "compile_scenario/scenario.h"
#include <QElapsedTimer>
#include <QGridLayout>
#include <QLabel>
#include <QMovie>
#include <QWidget>
```

Classes

- class Gui::MapViewer

Namespaces

- namespace Gui

12.32 map_viewer.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"

#include "compile_scenario/scenario.h"

#include <QElapsedTimer>
#include <QGridLayout>
#include <QLabel>
#include <QMovie>
#include <QWidget>

namespace Gui
{

    class MapViewer : public QWidget
    {
        Q_OBJECT

      public:
        explicit MapViewer(QWidget* parent = nullptr);

      public slots:
        void DataReceived();
        void WaitForData();

        void
        UpdateRoutes(std::pair<CompileScenario::Scenario::RouteMap::iterator,
                               CompileScenario::Scenario::RouteMap::iterator>
                         routes);
        void UpdateAgents(std::pair<std::vector<Core::Agent>::iterator,
                                    std::vector<Core::Agent>::iterator>
                              agents);

        inline void
        UpdateActiveAgent(int id)
        {
            m_ActiveAgentId = id;
        }

        inline void
        UpdateTimeStamp(float timeStamp)
        {
            m_TimeStamp = timeStamp;
        }

      protected:
        void paintEvent(QPaintEvent* event) override;
        void resizeEvent(QResizeEvent* event) override;
        void mousePressEvent(QMouseEvent* event) override;

      private:
        void UpdateRenderingArea();

        void DrawKeyframes(QPainter& painter);
        void DrawRoutes(QPainter& painter);
        void DrawLoader(QPainter& painter) const;

      private:
        int m_StartX, m_StartY;
        int m_Size;

        bool m_WaitingForData;
        QTimer* m_WaitingForDataTimer;
        QElapsedTimer m_WaitingForDataElapsedTimer;
```

```
        float m_LoaderAngle;
        int m_LoaderSize;
        float m_LoaderSpeed;
        float m_LoaderSpan;
        int m_LoaderThickness;

        std::pair<std::vector<Core::Agent>::iterator,
                  std::vector<Core::Agent>::iterator>
            m_Agents;
        std::pair<CompileScenario::Scenario::RouteMap::iterator,
                  CompileScenario::Scenario::RouteMap::iterator>
            m_Routes;

        int m_ActiveAgentId;
        float m_TimeStamp;
    };

} // namespace Gui
```

12.33 include/gui/menu_bar.h File Reference

```
#include <QMenuBar>
```

Classes

- class Gui::MenuBar

  The main menubar of the user interface.

Namespaces

- namespace Gui

12.34 menu_bar.h

Go to the documentation of this file.

```
#pragma once

#include <QMenuBar>

namespace Gui
{

    /// \brief The main menubar of the user interface.
    ///
    /// The main menubar exists to provide the user with easy access to
    /// functionality such as creating new scenarios, opening existing scenarios
    /// etc.
    class MenuBar : public QMenuBar
    {
        Q_OBJECT
      public:
        /// \brief Constructs the menu bar
        /// \param parent The parent widget of the menu bar
        MenuBar(QWidget* parent = nullptr);

      signals:
        void SaveScenario(const std::string& filename);
        void LoadScenario(const std::string& filename);

      private:
    };
} // namespace Gui
```

12.35 include/gui/planner.h File Reference

```
#include "gui/map_viewer.h"
#include "gui/timeline.h"
#include <QSplitter>
```

Classes

- class Gui::Planner

  The planner widget used for planning scenarios.

Namespaces

- namespace Gui

12.36 planner.h

Go to the documentation of this file.

```
#pragma once

#include "gui/map_viewer.h"
#include "gui/timeline.h"

#include <QSplitter>

namespace Gui
{
    /// \brief The planner widget used for planning scenarios
    ///
    /// Contains the graphical functionality to plan scenarios.
    class Planner : public QSplitter
    {
        Q_OBJECT
      public:
        /// \brief Constructs the planner widget.
        /// \param parent The parent of the planner widget.
        Planner(QWidget* parent = nullptr);

        /// \brief Destructs the planner widget.
        ~Planner();

      private:
        /// \brief The layout of the planner widget.
        MapViewer* m_MapViewer;
        Timeline* m_Timeline;
    };
} // namespace Gui
```

12.37 include/gui/scenario_controls.h File Reference

```
#include <QFrame>
#include <QGridLayout>
#include <QPushButton>
```

Classes

- class Gui::ScenarioControls

Namespaces

- namespace Gui

12.38 scenario_controls.h

Go to the documentation of this file.

```
#pragma once

#include <QFrame>
#include <QGridLayout>
#include <QPushButton>

namespace Gui
{

    class ScenarioControls : public QFrame
    {
        Q_OBJECT
      public:
        explicit ScenarioControls(QWidget* parent = nullptr);

      signals:
        void OpenSettingsDialog();
        void CompileScenario();

      private:
        QPushButton* m_SettingsButton;
        QPushButton* m_CompileButton;
        QGridLayout* m_Layout;
    };

} // namespace Gui
```

12.39 include/gui/sidebar.h File Reference

```
#include "coordinate_converter/coordinate_converter.h"
#include "core/types.h"
#include "gui/agent_controls.h"
#include "gui/keyframe_controls.h"
#include "gui/scenario_controls.h"
#include "gui/tab_widget.h"
#include "keyframe_management/keyframe_manager.h"
#include "map_management/map_manager.h"
#include <QVBoxLayout>
#include <QWidget>
```

Classes

- class Gui::Sidebar
  The sidebar of the main window.

Namespaces

- namespace Gui

12.40 sidebar.h

Go to the documentation of this file.

```cpp
#pragma once

#include "coordinate_converter/coordinate_converter.h"
#include "core/types.h"
#include "gui/agent_controls.h"
#include "gui/keyframe_controls.h"
#include "gui/scenario_controls.h"
#include "gui/tab_widget.h"
#include "keyframe_management/keyframe_manager.h"
#include "map_management/map_manager.h"

#include <QVBoxLayout>
#include <QWidget>

namespace Gui
{

    /// \brief The sidebar of the main window
    ///
    /// The sidebar of the main content exists to provide the user access to
    /// tools related to the active tab in the tab widget.
    class Sidebar : public QWidget
    {
        Q_OBJECT

      public:
        /// \brief Construct the sidebar.
        /// \param parent The parent of the sidebar.
        Sidebar(QWidget* parent = nullptr);

      signals:
        /// \brief Signal emitted when scenario data is ready to be processed.
        /// \param coord The UTM coordinate of the center of the scenario.
        /// \param size The size of the scenario in meters.
        void scenarioDataReady(Core::UTMCoordinate coord, int size);

      private slots:

        //        ///
        //        /// \brief Handle the keyframe data received from the
        //        AddKeyFrameDialog.
        //        /// \param agentId The ID of the agent associated with the
        //        keyframe.
        //        /// \param timeStamp The timestamp of the keyframe.
        //        /// \param x The x-coordinate of the keyframe.
        //        /// \param y The y-coordinate of the keyframe.
        //        /// \param z The z-coordinate of the keyframe.
        //        void OnAddKeyframeDialogDataReady(int agentId, float
        //        timeStamp, float x, float y, float z);

        //        ///
        //        /// \brief Handle the map data received from the MapDialog.
        //        /// \param latitude The latitude-coordinate of the center of
        //        the map.
        //        /// \param longitude The longitude-coordinate of the center of
        //        the map.
        //        /// \param size The size of the map in meters.
        //        void OnMapDataReady(float latitude, float longitude, float
        //        size);

      private:
        /// \brief The layout of the sidebar.
        QVBoxLayout* m_Layout;

        ScenarioControls* m_ScenarioControls;
        AgentControls* m_AgentControls;
        KeyframeControls* m_KeyframeControls;
    };

} // namespace Gui
```

12.41 include/gui/simulator.h File Reference

```cpp
#include "gui/map_viewer.h"
#include <QGridLayout>
#include <QWidget>
```

Classes

- class Gui::Simulator

  The simulator widget used to simulate scenarios.

Namespaces

- namespace Gui

12.42 simulator.h

Go to the documentation of this file.

```
#pragma once

#include "gui/map_viewer.h"

#include <QGridLayout>
#include <QWidget>

namespace Gui
{
    /// \brief The simulator widget used to simulate scenarios.
    ///
    /// Contains the graphical functionality to simulate scenarios.
    class Simulator : public QWidget
    {
      public:
        /// \brief Constructs the simulator widget.
        /// \param parent The parent of the simulator widget.
        Simulator(QWidget* parent = nullptr);

        /// \brief Destructs the simulator widget.
        ~Simulator();

        QSize
        sizeHint() const override
        {
            return { parentWidget()->width(), parentWidget()->height() };
        }

      private:
        /// \brief The layout of the simulator widget.
        QGridLayout* m_Layout;
    };
} // namespace Gui
```

12.43 include/gui/tab_widget.h File Reference

```
#include "gui/launcher.h"
#include "gui/planner.h"
#include "gui/simulator.h"
#include <QTabWidget>
```

Classes

- class Gui::TabWidget

  The tab widget of the main window.

Namespaces

- namespace Gui

12.44 tab_widget.h

Go to the documentation of this file.

```
#pragma once

#include "gui/launcher.h"
#include "gui/planner.h"
#include "gui/simulator.h"

#include <QTabWidget>

namespace Gui
{

    /// \brief The tab widget of the main window.
    ///
    // This widget is responsible for containing the core functionality of
    /// Hivemind; planning, simulating and launching. They are separated in
    /// their own tabs as a user should only need to access one of these at
    /// any point in time.
    class TabWidget : public QTabWidget
    {
      public:
        /// \brief Constructs the tab widget.
        /// \param parent The parent of the tab widget.
        TabWidget(QWidget* parent = nullptr);

        /// \brief Destructs the tab widget.
        ~TabWidget();

      private:
        /// \brief The planner widget.
        ///
        /// Contains the graphical functionality to plan scenarios.
        Planner* m_Planner;

        /// \brief The simulator widget.
        ///
        /// Contains the graphical functionality to simulate scenarios.
        Simulator* m_Simulator;

        /// \brief The launcher widget.
        ///
        /// Contains the graphical functionality to launch a scenario.
        Launcher* m_Launcher;
    };
} // namespace Gui
```

12.45 include/gui/timeline.h File Reference

```
#include <QComboBox>
#include <QResizeEvent>
#include <QWidget>
```

Classes

- class Gui::Timeline

  A custom QWidget to represent a timeline with keyframes.

Namespaces

- namespace Gui

12.46 timeline.h

Go to the documentation of this file.

```
#pragma once

#include <QComboBox>
#include <QResizeEvent>
#include <QWidget>

namespace Gui
{

    /// \class Timeline
    /// \brief A custom QWidget to represent a timeline with keyframes.
    class Timeline : public QWidget
    {
        Q_OBJECT
      public:
        ///
        /// \brief Constructor for the Timeline class.
        /// \param parent The parent QWidget
        ///
        explicit Timeline(QWidget* parent = nullptr);

        ///
        /// \brief Get the active agent ID
        /// \return The ID of the active agent
        ///
        inline int
        GetActiveAgent()
        {
            return m_activeAgentId;
        }

        ///
        /// \brief Get the current timestamp
        /// \return The current timestamp
        ///
        inline float
        GetTimeStamp()
        {
            return m_timeStamp;
        }

      protected:
        ///
        /// \brief Paint event handler
        /// \param event The paint event
        ///
        void paintEvent(QPaintEvent* event) override;

        ///
        /// \brief Mouse release event handler
        /// \param event The mouse release event
        ///
        void mouseReleaseEvent(QMouseEvent* event) override;

        ///
        /// \brief Resize event handler
        /// \param event The resize event
        ///
        void resizeEvent(QResizeEvent* event) override;
      signals:
        ///
        /// \brief Signal that is emitted when a timestamp is selected
        /// \param timeStamp The selected timestamp
        ///
        void timeStampSelected(float timeStamp);

      private:
        float m_timeStamp;        ///< The current timestamp
        int m_activeAgentId;      ///< ID of the active agent
        float m_pixelsPerSecond; ///< Pixels per second on the timeline
    };

} // namespace Gui
```

12.47 include/height_management/height_manager.h File Reference

```
#include "core/types.h"
#include <array>
#include <iostream>
#include <vector>
```

Classes

- class HeightManagement::HeightManager
- struct HeightManagement::HeightManager::heightdata

Namespaces

- namespace HeightManagement

12.48 height_manager.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"
#include <array>
#include <iostream>
#include <vector>

namespace HeightManagement
{

    class HeightManager
    {
      public:
        struct heightdata
        {
            double x;
            double y;
            double z;
        };

        /// \brief Constructor of HeightManager class.
        ///
        /// \returns No object.
        HeightManager();

        /// \brief Function to update the origin point. Running this will also
        /// trigger the population of height data for the chosen subset of the
        /// GeoTiff file.
        ///
        /// \param x X coordinate used for GeoTiff subset origin.
        /// \param y Y coordinate used for GeoTiff subset origin.
        /// \returns No object, but will update the origin for this instance of
        /// HeightManager and will populate the instance with height data.
        void UpdateOrigin(Core::UTMCoordinate UTMCoord, int size);

        /// \brief Function to return the whole "height_management" for a given
        /// point.
        ///
        /// \param inputRelativeX The X coordinate in the relative system (where
        /// 0,0 is the top left corner of the system). \param inputRelativeY The
        /// Y coordinate in the relative system. \returns A height_management,
        /// containing the geographic (absolute) x, y and z coordinates.
        bool GetVertex(int inputRelativeX, int inputRelativeY,
                       heightdata& vertex);

        /// \brief Function to return height, given relative coordinates (from a
        /// system where 0, 0 is in the upper left corner)
        ///
```

```
    /// \param inputRelativeX The relative X value of the point.
    /// \param inputRelativeY The relative Y value of the point.
    /// \returns A float containing the height value of the point in metres.
    bool GetHeight(int inputRelativeX, int inputRelativeY, float& height);

    /// \brief Function to get the height_management of an absolute
    /// (geographic) coordinate, using the same coordinate system of the
    /// dataset.
    ///
    /// \param inputX The absolute X value of the point.
    /// \param inputY The absolute Y value of the point.
    /// \returns A float containing the height of the point in metres.
    bool GetVertexAbsolute(double inputX, double inputY,
                           heightdata& vertex);

    /// \brief Function to get the height of an absolute (geographic)
    /// coordinate, using the same coordinate system of the dataset.
    ///
    /// \param inputX The absolute X value of the point.
    /// \param inputY The absolute Y value of the point.
    /// \returns A float containing the height of the point in metres.
    float GetHeightAbsolute(double inputX, double inputY);

    /// \brief Function to allow user to change GeoTiff file used in
    /// planning. If this function is not run, the user can still update the
    /// origin and Hivemind will run using the cached GeoTiff file.
    ///
    /// \param filePath Complete file path of the file to be used.
    /// \param x X coordinate used for GeoTiff subset origin. Height data
    /// will be populated in a 500x500 pixel centered on the origin point.
    /// This is hard coded into the class. \param y Y coordinate used for
    /// GeoTiff subset origin. \returns No object, but will update the path
    /// for the cached tif.
    void LoadTif(const char* filePath, double x, double y);

  private:
    /// \brief Function that will open the GeoTiff file and extract all
    /// heights for the given subset of the dataset used.
    ///
    /// \returns No object, but after this has run, all heights will have
    /// been imported into the instance of the class and the various
    /// GetHeight methods can be run.
    void PopulateVertices();

    /// \brief Function to test whether a point exists within the scope of
    /// the selected data subset.
    ///
    /// \param x the X value of the coordinate to be tested.
    /// \param y the Y value of the coordinate to be tested.
    /// \returns A bool indicating whether or not the input exists in the
    /// subset and is valid.
    bool ValidInput(int x, int y);

    /// \brief Function to test whether a point exists within the scope of
    /// the elected data subset. Overloaded version of ValidInput() that
    /// takes doubles.
    ///
    /// \param x The X value of the coordinate to be tested.
    /// \param y The Y value of the coordinate to be tested.
    /// \returns A bool indicating whether or not the input exists in the
    /// subset and is valid.
    bool ValidInput(double x, double y);

    /// \brief Function that tests whether the selected origin point is
    /// within the bounds of the currently active data set, given the buffer
    /// size required to extract the subset.
    ///
    /// \param x The X value of the origin point.
    /// \param y The Y value of the origin point.
    /// \returns A bool indicating whether or not the origin point is within
    /// bounds.
    bool OrigoWithinBounds(double x, double y);

    /// \brief Function to update the corner coordinates saved within the
    /// member instance of the chosen dataset.
    ///
    /// \returns No object, but the corner coordinates will be updated,
    /// given there were no problems opening the GeoTiff file.
    void UpdateCornerCoords();

  private:
    const char* m_CachedTifName = "../res/Kongsberg.tif";
    const char* m_CoordinateSystem{ "UTM33" };
    int m_Resolution{ 1 };
    int m_Size;
    long m_UpperLeftX;
    long m_UpperLeftY;
```

```
        long m_LowerRightX;
        long m_LowerRightY;
        heightdata* m_Vertices;
        heightdata m_Origo{ 0, 0, 0 };
        heightdata m_SelectionCorner;
    };

} // namespace HeightManagement
```

12.49 include/keyframe_management/keyframe_manager.h File Reference

```
#include "core/serializer.h"
#include "core/types.h"
#include <QObject>
#include <vector>
```

Classes

- class KeyframeManagement::KeyframeManager

  This is the class that manages keyframes.

Namespaces

- namespace KeyframeManagement

12.50 keyframe_manager.h

Go to the documentation of this file.

```
#pragma once

#include "core/serializer.h"
#include "core/types.h"

#include <QObject>
#include <vector>

namespace KeyframeManagement
{

    ///
    /// \brief This is the class that manages keyframes
    class KeyframeManager : public QObject,
                            JSON
    {
        Q_OBJECT

      public:
        ///
        /// \brief Returns the singleton instance of the KeyframeManager
        /// \return A reference to the singleton instance of the KeyframeManager
        ///
        static KeyframeManager&
        Instance()
        {
            static KeyframeManager instance;
            return instance;
        }

        ///
        /// \brief Adds a keyframe to the keyframe list using x, y, and z
        /// coordinates
```

```
        /// \param agentId The ID of the agent
        /// \param timeStamp The timestamp of the keyframe
        /// \param x The x coordinate
        /// \param y The y coordinate
        /// \param z The z coordinate
        ///
        void AddKeyframe(int agentId, float timeStamp, float x, float y,
                         float z);

        ///
        /// \brief Adds a keyframe to the keyframe list using a
        /// CartesianCoordinate \param agentId The ID of the agent \param
        /// timeStamp The timestamp of the keyframe \param position The
        /// CartesianCoordinate representing the position
        ///
        void AddKeyframe(int agentId, float timeStamp,
                         Core::CartesianCoordinate position);

        ///
        /// \brief Adds a keyframe object to the keyframe list
        /// \param keyframe The keyframe object to add
        ///
        void AddKeyframe(Core::Keyframe& keyframe);

        ///
        /// \brief Removes a keyframe from the keyframe list
        /// \param keyframe The keyframe to remove
        ///
        void RemoveKeyframe(const Core::Keyframe& keyframe);

        ///
        /// \brief Dumps keyframe information to the console for debugging
        /// purposes
        ///
        void DebugDump(void) const;

        ///
        /// \brief Returns a reference to the list of keyframes
        /// \return A reference to the list of keyframes
        ///
        inline std::vector<Core::Keyframe>&
        GetKeyframes()
        {
            return m_Keyframes;
        }

      signals:
        void KeyframeAdded();

      private:
        KeyframeManager() {}  ///< Private constructor for singleton pattern

        ~KeyframeManager() {} ///< Private destructor for singleton pattern

        KeyframeManager(const KeyframeManager&) = delete;
        KeyframeManager& operator=(const KeyframeManager&) = delete;

        std::vector<Core::Keyframe> m_Keyframes;

        JSONSTART
        JSONMEMBERVECTOR(Core::Keyframe, m_Keyframes)
        JSONEND
    };

} // namespace KeyframeManagement
```

12.51 include/map_management/map_manager.h File Reference

```
#include "core/types.h"
#include <QObject>
```

Classes

- class MapManagement::MapManager

  This is the class responsible for retrieving maps from Kartverket.

Namespaces

- namespace MapManagement

12.52 map_manager.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"

#include <QObject>

namespace MapManagement
{

    ///
    /// \brief This is the class responsible for retrieving maps from
    /// Kartverket.
    ///
    class MapManager : public QObject
    {
        Q_OBJECT
      public:
        /// \brief Returns the singleton instance of the class.
        static MapManager&
        Instance()
        {
            static MapManager instance;
            return instance;
        }

        /// \brief Retrieves the map from Kartverket for the specified UTM
        /// coordinate and size.
        ///
        /// This function retrieves the satellite map data from Kartverket with
        /// a HTTP request for the specified UTM coordinate and size.
        ///
        /// \param coord The UTM coordinate for the center of the map.
        /// \param size The size of the map in meters.
        static void GetMap(Core::UTMCoordinate coord, int size);

        /// \brief Calculates the UTM corner coordinates for the specified UTM
        /// coordinate and size.
        ///
        /// This function calculates the UTM corner coordinates for the
        /// specified UTM coordinate and size, and stores them in the
        /// CornerCoordinates variable.
        ///
        /// \param coord The UTM coordinate for the center of the map.
        /// \param size The size of the map in meters.
        static void CalculateCornerCoordinates(Core::UTMCoordinate coord, int size);

        /// \brief Returns the map data as a byte array.
        static inline QByteArray&
        GetData()
        {
            return Instance().m_Data;
        }

        static inline int
        GetImageResolution()
        {
            return Instance().m_ImageResolution;
        }

      signals:
        /// \brief Signal emitted when the map image data has been retrieved.
        void GotImage();
        void RequestImage();

      private:
        /// \brief Constructor.
        MapManager() : m_ImageResolution{ 1024 } {};

        /// \brief Destructor.
        ~MapManager() = default;

        QByteArray m_Data;
        QString m_Area;
```

```
        int m_ImageResolution;
    };

} // namespace MapManagement
```

12.53 include/routemaker/graph.h File Reference

```
#include <functional>
#include <iostream>
#include <memory>
#include <queue>
#include <vector>
```

Classes

- struct Routemaker::Node< T >

    Represents a node in a Graph data structured made for path-finding.
- class Routemaker::Graph< T >

    Abstract graph interface optimized for path-finding.

Namespaces

- namespace Routemaker

12.54 graph.h

Go to the documentation of this file.

```
#pragma once

#include <functional>
#include <iostream>
#include <memory>
#include <queue>
#include <vector>

namespace Routemaker
{

    /// \brief Represents a node in a Graph data structured made for
    /// path-finding
    ///
    /// \tparam T Type of data to store inside the node
    template<typename T>
    struct Node
    {
        /// \brief Data stored in the the node.
        ///
        /// Stores data not needed by the A\* path-finding algorithm. This is
        /// what the user actually wants to store in the \ref Graph.
        T Data;

        /// \brief A non-owner pointer to the parent of the node.
        ///
        /// Should not be set by user. The A\* path-finding algorithm sets the
        /// value for this member when traversing the \ref Graph. It used to
        /// find the way back to the start after the goal is found.
        std::weak_ptr<Node<T» Parent;

        /// \brief Specifies if a given node has been visited during
        /// path-finding.
        ///
```

```
        /// Should not be set by user. Is generally only used internally by the
        /// A\* path-finding algorithm when traversing the \ref Graph. May be
        /// used in debug views to visualize which nodes are visited during
        /// path-finding.
        bool Visited;

        /// \brief Represents the assumed cost from the start to the goal node
        /// through this node.
        ///
        /// Should not be set by the user.
        /// The A\* path-finding algorithm uses cost to find the shortest path
        /// in a reasonable amount of time. This member contains the sum of the
        /// cost to get to this node from the start node, represented in \ref
        /// LocalGoal, plus the assumed cost to get from this node to the goal
        /// node. The A\* path-finding algorithm uses this value during \ref
        /// Graph traversal to sort a priority queue in order to explore the
        /// assumed shortest paths first.
        double GlobalGoal;

        /// \brief Represents the cost from the start node to this node.
        ///
        /// Should not be set by the user.
        /// The A\* path-finding algorithm uses cost to find the shortest path
        /// in a reasonable amount of time. This member contains the sum of the
        /// cost to get to this node from the start node. While traversing the
        /// \ref Graph, the A\* path-finding algorithm updates and uses this
        /// member to check for shorter paths.
        double LocalGoal;
    };

    /// \brief Abstract graph interface optimized for path-finding.
    ///
    /// \tparam T Type of user data to store in each node
    ///
    /// This interface is designed to be flexible and scalable. The sub-classes
    /// are required to implement a few methods, such as \ref Graph.GetNeighbors
    /// and \ref Graph.GetCost for the A\* path-finding algorithm to work.
    template<typename T>
    class Graph
    {
      public:
        using NodePtr = std::shared_ptr<Node<T»; ///< Helper alias to make code
                                                 ///< more readable.

      public:
        /// \brief Collects all neighbor nodes of \p node.
        ///
        /// Implemented by sub-classes of Graph.
        /// The neighbor relationship between nodes define the edges of the
        /// graph. It is up to the subclass to define these relationships. For a
        /// 2D grid, the neighbors would simply be the nodes directly to the
        /// north, south, east and west, in addition to the corners between
        /// them. For a road network, the relationships may be more complex.
        ///
        /// \param node A pointer to the node from which to collect all
        /// neighbors \return A vector of pointers to all the neighbors of \p
        /// node
        virtual std::vector<NodePtr> GetNeighbors(NodePtr node) = 0;

        /// \brief Returns the cost between \p a and \p b.
        ///
        /// Implemented by sub-classes of Graph.
        /// The a\* path-finding algorithm uses cost to efficiently find the
        /// best path between two nodes. In order to do this, it requires some
        /// method of calculating the cost of moving between any two nodes. It
        /// is up to the sub-class to define how this is calulated. An example
        /// of this cost may be the euclidean distance between two nodes.
        ///
        /// \param a Pointer to the first \ref Node
        /// \param b Pointer to the second \ref Node
        /// \return Cost between node \p a and node \p b.
        virtual double GetCost(NodePtr a, NodePtr b) = 0;

        /// \brief Determines if there is a direct line of sight between node \p
        /// a and node \p b.
        ///
        /// Implemented by sub-classes of Graph.
        /// The \ref Graph.PostSmooth method traverses the already found path
        /// through the A\* path-finding algorithm and simplifies it by using
        /// this method. In a graph representing a 2D grid, a Bresenham
        /// implementation or ray-casting can be used to determine line of
        /// sight.
        ///
        /// \param a Pointer to the first \ref Node
        /// \param b Pointer to the second \ref Node
        /// \return bool specifying whether or not there is a direct line of
        /// sight
```

```
    virtual bool HasLineOfSight(NodePtr a, NodePtr b) = 0;

    /// \brief Resets all local and global goals and parent relationships of
    /// all nodes.
    ///
    /// Implemented by sub-classes of Graph.
    /// In order to be able to re-use the same graph for several A\*
    /// searches, the \ref Graph.SolveAStar method needs to be able to reset
    /// all the nodes. As this interface does not contain the actual
    /// collection of nodes, this needs to be implemented in the
    /// sub-classes.
    virtual void ResetNodes(void) = 0;

    /// \brief Finds *cheapest* path from \p start to \p goal.
    ///
    /// \param start Pointer to the node to start the path from
    /// \param goal  Pointer to the node to find the path to
    ///
    /// Using the A\* algorithm, this method explores the graph's nodes and
    /// updates their local and global goals, their visited flags, as well
    /// as their parent relationships.
    ///
    /// When the algorithm finishes, given that a path exists between the
    /// nodes, the cheapest path between them is defined by the parent
    /// relationships. The path can be *extracted* by starting at the \p
    /// goal and following the \ref Node.Parent pointers until \p start is
    /// reached, saving each node in a list and reversing it at the end.
    void SolveAStar(NodePtr start, NodePtr goal);

    /// \brief Simplifies the path from \p start to \p goal.
    ///
    /// \param start Pointer to the start node of the path
    /// \param goal  Pointer to the end node of the path
    ///
    /// Should be run on the same nodes as \ref Graph.SolveAStar, and should
    /// only be called after \ref Graph.SolveAStar has finished.
    ///
    ///
    void PostSmooth(NodePtr start, NodePtr goal);
};

template<typename T>
void
Graph<T>::SolveAStar(NodePtr start, NodePtr goal)
{
    ResetNodes(); // Make sure all relational values are reset

    NodePtr current = start;
    current->LocalGoal = 0.0;
    current->GlobalGoal = GetCost(current, goal);

    // Create a priority queue that compares nodes' global goal value
    std::priority_queue<NodePtr, std::vector<NodePtr>,
                        std::function<bool(NodePtr, NodePtr)>>
        notTested([](NodePtr a, NodePtr b) {
            return a->GlobalGoal > b->GlobalGoal;
        });

    notTested.push(start);

    // Let's go for all long as we have untested discovered nodes
    while (!notTested.empty()) {
        // But in case we have already discovered nodes in our list, let's
        // remove them
        while (!notTested.empty() && notTested.top()->Visited) {
            notTested.pop();
        }

        // If we ended up removing some nodes, and we are now out of
        // untested nodes, let's break from the loop
        if (notTested.empty()) {
            break;
        }

        current = notTested.top();
        current->Visited = true;

        for (auto neighbor : GetNeighbors(current)) {
            // We only want to explore unoccupied cells.
            if (!neighbor->Visited && !neighbor->Data.Occupied) {
                notTested.push(neighbor);
            }

            // Let's calculate the cost of the travel to this node so far +
            // the cost to get from here to the neighbor, and update the
            // neighbors relational values if it is a new record for the
            // neighbor.
```

```
                double candidateGoal =
                    current->LocalGoal + GetCost(current, neighbor);
                if (candidateGoal < neighbor->LocalGoal) {
                    neighbor->Parent = current;
                    neighbor->LocalGoal = candidateGoal;
                    neighbor->GlobalGoal =
                        neighbor->LocalGoal + GetCost(neighbor, goal);
                }
            }
        }
    }

    // Quite a simple little algorithm to simplify and smooth out a path found
    // through A*: We just start at the goal, and check if we have a direct line
    // of sight to our grandparent. If we do, then we can remove the middle man,
    // our parent, from the equation and make our grandparent our parent
    // instead. Then check again for our new grandparent. If we do not have a
    // direct line of sight to our grandparent, we move on to our parent and
    // check its grandparent. We do this recursively until we reach the start
    // node.
    template<typename T>
    void
    Graph<T>::PostSmooth(NodePtr start, NodePtr goal)
    {
        NodePtr current = goal;
        NodePtr parent = current->Parent.lock();
        while (current && parent && (current != start)) {
            NodePtr grandParent = parent->Parent.lock();
            if (!grandParent) {
                break;
            }
            if (HasLineOfSight(current, grandParent)) {
                current->Parent = grandParent;
                parent = grandParent;
                continue;
            }
            current = parent;
            parent = current->Parent.lock();
        }
    }

} // namespace Routemaker
```

12.55 include/routemaker/routemaker.h File Reference

```
#include "core/types.h"
#include "height_management/height_manager.h"
#include "routemaker/graph.h"
#include <cstdint>
#include <list>
#include <vector>
```

Classes

- struct Routemaker::Cell2D
- class Routemaker::Routemaker

  Main class responsible for handling creation of routes between keyframes.

Namespaces

- namespace Routemaker

12.56 routemaker.h

Go to the documentation of this file.

```
#pragma once

#include "core/types.h"
#include "height_management/height_manager.h"
#include "routemaker/graph.h"

#include <cstdint>
#include <list>
#include <vector>

namespace Routemaker
{

    struct Cell2D
    {
        uint32_t X, Y;
        bool Occupied;
    };

    /// \brief Main class responsible for handling creation of routes between
    /// keyframes.
    class Routemaker : public Graph<Cell2D>
    {
      public:
        /// \brief Instatiates a routemaker object, along with it's Heightmap
        /// member.
        ///
        /// The \p origin and \p size of the scenario are simply passed to the
        /// HeightMap member. In the case that the HeightMap class is converted
        /// to a singleton or the scenario class gains ownership over the
        /// Heightmap, they should not be necessary.
        ///
        /// \param origin The origin of the scenario in UTM coordinate space.
        /// \param size The size of the scenario in meters
        explicit Routemaker(const Core::UTMCoordinate& origin, int size);

        /// \brief Creates a a vector of coordinates defining a path between two
        /// keyframes.
        ///
        /// Utilizes methods from the Graph interface, namely GetNeighbors,
        /// GetCost, HasLineOfSight and BresenhamLine, to generate a path
        /// between \p a and \p b.
        ///
        /// \param a First keyframe to create to create path from
        /// \param b Second keyframe to create path from
        ///
        /// returns A vector of coordinates in symmetrical cartesian coordinate
        /// system space, which together forms a path.
        std::vector<Core::CartesianCoordinate>
        MakeRoute(const Core::Keyframe& a, const Core::Keyframe& b);

        /// \brief Get a node at a position
        ///
        /// \param x x-coordinate of position
        /// \param y y-coordinate of position
        /// \returns A shared pointer to the node at the specified location
        NodePtr GetNode(uint32_t x, uint32_t y) const;

        /// \brief Updates the origin coordinate and the size of the map
        ///
        /// \param UTMOrigin The new origin coordinate for the map
        /// \param size The new size of the map in meters
        void UpdateOrigin(Core::UTMCoordinate UTMOrigin, int size);

        void UpdateResolution();

      private:
        std::vector<NodePtr> GetNeighbors(NodePtr node) override;

        double GetCost(NodePtr a, NodePtr b) override;

        bool HasLineOfSight(NodePtr a, NodePtr b) override;

        void ResetNodes() override;

        /// \brief Calculates the <a
        /// href="https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html">Bresenham
        /// Line</a> between two nodes
        ///
        /// \param a Pointer to first node
        /// \param b Pointer to seconds node
        /// \returns A list of pointers to the nodes that make up the <a
```

```
        /// href="https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html">Bresenham
        /// Line</a>  between \p a and \p b.
        std::list<NodePtr> BresenhamLine(const NodePtr& a,
                                         const NodePtr& b) const;

      private:
        /// \brief All the nodes that make up the graph
        std::vector<NodePtr> m_Nodes;

        /// \brief HeightManager instance owned by Routemaker
        std::unique_ptr<HeightManagement::HeightManager> m_HeightMap;

        /// \brief Width (and height) of the active scenario
        int m_MapWidth;

        /// \brief Resolution of the routemaker in meters.
        ///
        /// A resolution of 3 meters would mean that any one move in vertical or
        /// horizontal direction would correspond to a 3 meter movement. A
        /// higher value increases performance of the routemaker, but decreases
        /// route fidelity.
        int m_RoutemakerRes;

        /// \brief Width (and height) of the routemaker
        ///
        /// Will always equal \a m_MapWidth divided by \a m_RoutemakerRes
        int m_RoutemakerWidth;
    };

} // namespace Routemaker
```

12.57 README.md File Reference

12.58 src/compile_scenario/scenario.cpp File Reference

```
#include "compile_scenario/scenario.h"
#include "coordinate_converter/coordinate_converter.h"
#include "map_management/map_manager.h"
```

Namespaces

- namespace CompileScenario

12.59 scenario.cpp

Go to the documentation of this file.

```
#include "compile_scenario/scenario.h"
#include "coordinate_converter/coordinate_converter.h"
#include "map_management/map_manager.h"

namespace CompileScenario
{

    // The constructor to the scenario class update the size and origin to
    // coordinate converter, map manager and routemaker so the whole system uses
    // the same values.
    Scenario::Scenario(std::string name, Core::GeographicalCoordinate origin,
                       int size)
        : m_Name(name), m_Size(size), m_Origin(origin),
          m_KeyframeManager(KeyframeManagement::KeyframeManager::Instance())
    {
        CoordinateConverter::CoordConv::ResetOrigin(origin, size);
        MapManagement::MapManager::GetMap(
            CoordinateConverter::CoordConv::GeographicToUTM(origin), size);
        m_Routemaker = std::make_unique<Routemaker::Routemaker>(
            CoordinateConverter::CoordConv::GeographicToUTM(origin), size);
```

```
    }

    // SetOrigin to the scenario class update the size and origin to
    // coordinate converter, map manager and routemaker so the whole system uses
    // the same values.
    void
    Scenario::SetOrigin(Core::GeographicalCoordinate GeoCoord, int size)
    {
        m_Origin = GeoCoord;
        m_Size = size;
        CoordinateConverter::CoordConv::ResetOrigin(GeoCoord, size);
        m_Routemaker->UpdateOrigin(
            CoordinateConverter::CoordConv::GeographicToUTM(GeoCoord), size);
        MapManagement::MapManager::GetMap(
            CoordinateConverter::CoordConv::GeographicToUTM(GeoCoord), size);
    }

    Scenario::RouteMap&
    Scenario::Compile()
    {
        if (m_KeyframeManager.GetKeyframes().empty()) {
            return m_Routes;
        }

        m_Routes.clear();
        auto keyframes = m_KeyframeManager.GetKeyframes();
        // Keyframes need to be sorted by agentID and timestamp before planning
        // routes between them.
        std::sort(keyframes.begin(), keyframes.end(),
                  [](Core::Keyframe a, Core::Keyframe b) {
                      if (a.AgentId != b.AgentId) {
                          return a.AgentId < b.AgentId;
                      }
                      return a.TimeStamp < b.TimeStamp;
                  });
        // The routes generated by the routemaker is stored in a map where the
        // key is the agent ID and the value is a vector with the coordinates
        // the routemaker returns.
        for (int i = 0; i < keyframes.size() - 1; i++) {
            // Check if the current and next keyframe belongs to the same agent
            // and search for the agent Id in the map. If this is the first
            // agent with this ID the routes will be inserted as a new element
            // in the map. If the agent ID is already in the map, the returned
            // values from the routemaker will be pushed back into the place for
            // the agent ID to the agent.
            if (keyframes[i].AgentId == keyframes[i + 1].AgentId) {
                auto iter = m_Routes.find(keyframes[i].AgentId);
                if (iter == m_Routes.end()) {
                    std::vector<std::vector<Core::CartesianCoordinate» vec;
                    vec.push_back(m_Routemaker->MakeRoute(keyframes[i],
                                                          keyframes[i + 1]));
                    m_Routes.insert(std::make_pair(keyframes[i].AgentId, vec));
                } else {
                    iter->second.push_back(m_Routemaker->MakeRoute(
                        keyframes[i], keyframes[i + 1]));
                }
            }
        }

        return m_Routes;
    }

    void
    Scenario::AddAgent(Core::Agent newAgent)
    {
        m_Agents.push_back(newAgent);
    }

    void
    Scenario::save(std::string filename)
    {
        Json::serialize(filename, this);
    }

    void
    Scenario::load(std::string filename)
    {
        Json::deserialize(filename, this);
        SetOrigin(m_Origin, m_Size);
    }

} // namespace CompileScenario
```

12.60 src/coordinate_converter/coordinate_converter.cpp File Reference

```
#include "coordinate_converter/coordinate_converter.h"
```

Namespaces

- namespace CoordinateConverter

12.61 coordinate_converter.cpp

Go to the documentation of this file.

```
#include "coordinate_converter/coordinate_converter.h"

namespace CoordinateConverter
{

    // ResetOrigin will manage that rest of the function always work with the
    // same origin. The size parameter represent the size of the coordinate
    // system.
    void
    CoordConv::ResetOrigin(Core::GeographicalCoordinate geoCoord, int size)
    {
        auto& instance = GetInstance();
        instance.m_Size = size;
        instance.m_OriginGeographical = { geoCoord.Latitude,
                                          geoCoord.Longitude };
        instance.m_Origin.Reset(geoCoord.Latitude, geoCoord.Longitude, 0);
    }

    // This function convert a geographical coordinate to a cartesian
    // coordinate.
    Core::CartesianCoordinate
    CoordConv::GeographicalToCartesian(Core::GeographicalCoordinate geoCoord)
    {
        auto& instance = GetInstance();
        double x, y, z;
        instance.m_Origin.Forward(geoCoord.Latitude, geoCoord.Longitude, 0, x,
                                  y, z);
        return { x, y, z };
    }

    // This function convert a cartesian coordinate to a geographical coordinate
    Core::GeographicalCoordinate
    CoordConv::CartesianToGeographical(Core::CartesianCoordinate cartCoord)
    {
        auto& instance = GetInstance();
        double lat, lon, alt;
        instance.m_Origin.Reverse(cartCoord.X, cartCoord.Y, cartCoord.Z, lat,
                                  lon, alt);
        return { lat, lon };
    }

    // This function return the origin to hivemind.
    Core::GeographicalCoordinate
    CoordConv::GetOrigin()
    {
        auto& instance = GetInstance();
        return instance.m_OriginGeographical;
    }

    // This function convert from a symmetric coordinate system to an
    // asynmmetric coordinate system. The size parameter represent the size of
    // the coordinate system.
    Core::CartesianCoordinate
    CoordConv::SymmetricToAsymmetric(Core::CartesianCoordinate symmetric)
    {
        auto& instance = GetInstance();
        symmetric.X = symmetric.X + (instance.m_Size / 2);
        symmetric.Y = -symmetric.Y + (instance.m_Size / 2);
        return symmetric;
    }

    // This function convert from an asymmetric coordinate system to a
```

```
    // synmmetric coordinate system. The size parameter represent the size of
    // the coordinate system.
    Core::CartesianCoordinate
    CoordConv::AsymmetricToSymmetric(Core::CartesianCoordinate asymmetric)
    {
        auto& instance = GetInstance();
        asymmetric.X = asymmetric.X - (instance.m_Size / 2);
        asymmetric.Y = -asymmetric.Y + (instance.m_Size / 2);
        return asymmetric;
    }

    // Hivemind are using UTM33N and therefore are this hardcoded in the call to
    // geographiclib. For scalability and easier maintenance this should be able
    // to configured.
    Core::UTMCoordinate
    CoordConv::GeographicToUTM(Core::GeographicalCoordinate GeoCoord)
    {
        Core::UTMCoordinate utmCoord;
        GeographicLib::UTMUPS::Forward(GeoCoord.Latitude, GeoCoord.Longitude,
                                       utmCoord.Zone,
                                       utmCoord.IsNorthHemisphere,
                                       utmCoord.Easting, utmCoord.Northing, 33);
        return utmCoord;
    }

    // This function convert from UTM coordinates to geographical coordinates.
    Core::GeographicalCoordinate
    CoordConv::UTMToGeographic(Core::UTMCoordinate UTMCoord)
    {
        Core::GeographicalCoordinate geoCoord(0, 0);
        GeographicLib::UTMUPS::Reverse(
            UTMCoord.Zone, UTMCoord.IsNorthHemisphere, UTMCoord.Easting,
            UTMCoord.Northing, geoCoord.Latitude, geoCoord.Longitude);
        return geoCoord;
    }

} // namespace CoordinateConverter
```

12.62 src/core/serializer.cpp File Reference

```
#include "core/serializer.h"
#include "rapidjson/document.h"
#include "rapidjson/istreamwrapper.h"
#include "rapidjson/prettywriter.h"
#include <fstream>
#include <iostream>
#include <memory>
#include <string>
```

Namespaces

- namespace Json

Macros

- #define RAPIDJSON_HAS_STDSTRING 1

Functions

- void Json::serialize (std::string filename, ISValue ∗p)

  Function to start serializing an onbject.
- void Json::deserialize (std::string filename, ISValue ∗p)

  Function to start deserializing a file.

Variables

- bool debug = false

12.62.1 Macro Definition Documentation

12.62.1.1 RAPIDJSON_HAS_STDSTRING

```
#define RAPIDJSON_HAS_STDSTRING 1
```

Definition at line 1 of file serializer.cpp.

12.62.2 Variable Documentation

12.62.2.1 debug

```
bool debug = false
```

Definition at line 12 of file serializer.cpp.

Referenced by Json::ISDouble::ToDom(), and Json::ISMemberVector< T >::ToDom().

12.63 serializer.cpp

Go to the documentation of this file.

```
00001 #define RAPIDJSON_HAS_STDSTRING 1
00002
00003 #include "core/serializer.h"
00004 #include "rapidjson/document.h"
00005 #include "rapidjson/istreamwrapper.h"
00006 #include "rapidjson/prettywriter.h" // for stringify JSON
00007 #include <fstream>
00008 #include <iostream>
00009 #include <memory>
00010 #include <string>
00011
00012 bool debug = false;
00013
00014 namespace Json
00015 {
00016
00017     rapidjson::Value
00018     ISInt::ToDom(rapidjson::Document&)
00019     {
00020         rapidjson::Value v;
00021         v.SetInt(value);
00022         return v;
00023     }
00024
00025     void
00026     ISInt::FromDom(rapidjson::Value& v, rapidjson::Document&)
00027     {
00028         assert(v.IsInt());
00029         value = v.GetInt();
```

```
    }

    rapidjson::Value
    ISFloat::ToDom(rapidjson::Document&)
    {
        rapidjson::Value v;
        v.SetFloat(value);
        return v;
    }

    void
    ISFloat::FromDom(rapidjson::Value& v, rapidjson::Document&)
    {
        assert(v.IsFloat());
        value = v.GetFloat();
    }

    rapidjson::Value
    ISDouble::ToDom(rapidjson::Document&)
    {
        rapidjson::Value v;
        if (debug)
            std::cout « value « std::endl;
        v.SetDouble(value);
        return v;
    }

    void
    ISDouble::FromDom(rapidjson::Value& v, rapidjson::Document&)
    {
        assert(v.IsDouble());
        value = v.GetFloat();
    }

    rapidjson::Value
    ISBool::ToDom(rapidjson::Document&)
    {
        rapidjson::Value v;
        v.SetBool(value);
        return v;
    }

    void
    ISBool::FromDom(rapidjson::Value& v, rapidjson::Document&)
    {
        assert(v.IsBool());
        value = v.GetBool();
    }

    rapidjson::Value
    ISString::ToDom(rapidjson::Document& d)
    {
        rapidjson::Value v;
        v.SetString(value.c_str(), d.GetAllocator());
        return v;
    }

    void
    ISString::FromDom(rapidjson::Value& v, rapidjson::Document&)
    {
        assert(v.IsString());
        value = v.GetString();
    }

    rapidjson::Value
    ISValue::ToDom(rapidjson::Document& d)
    {
        ISProperties p = GetProperty();

        rapidjson::Value v;
        v.SetObject();
        v.AddMember("TypeId", GetName(d), d.GetAllocator());

        for (auto& element : p) {
            rapidjson::Value n;
            n.SetString(element.name, d.GetAllocator());
            if (element.value != nullptr)
                v.AddMember(n, element.value->ToDom(d), d.GetAllocator());
        }
        return v;
    }

    void
    ISValue::FromDom(rapidjson::Value& v, rapidjson::Document& d)
    {
        ISProperties p = GetProperty();
        if (v.IsObject()) {
```

```
        for (auto& element : p) {
            element.value->FromDom(v[element.name], d);
        }
    }
}

rapidjson::Value
ISIntVector::ToDom(rapidjson::Document& d)
{
    rapidjson::Value a;
    a.SetArray();
    for (auto& element : value) {
        rapidjson::Value v;
        v.SetInt(element);
        a.PushBack(v, d.GetAllocator());
    }
    return a;
}

void
ISIntVector::FromDom(rapidjson::Value& v, rapidjson::Document&)
{
    for (rapidjson::SizeType i = 0; i < v.Size();
         i++) { // rapidjson uses SizeType instead of size_t.
        value.push_back(v[i].GetInt());
    }
}

rapidjson::Value
ISFloatVector::ToDom(rapidjson::Document& d)
{
    rapidjson::Value a;
    a.SetArray();
    for (auto& element : value) {
        rapidjson::Value v;
        v.SetFloat(element);
        a.PushBack(v, d.GetAllocator());
    }
    return a;
}

void
ISFloatVector::FromDom(rapidjson::Value& v, rapidjson::Document&)
{
    for (rapidjson::SizeType i = 0; i < v.Size();
         i++) { // rapidjson uses SizeType instead of size_t.
        value.push_back(v[i].GetFloat());
    }
}

rapidjson::Value
ISDoubleVector::ToDom(rapidjson::Document& d)
{
    rapidjson::Value a;
    a.SetArray();
    for (auto& element : value) {
        rapidjson::Value v;
        v.SetDouble(element);
        a.PushBack(v, d.GetAllocator());
    }
    return a;
}

void
ISDoubleVector::FromDom(rapidjson::Value& v, rapidjson::Document&)
{
    for (rapidjson::SizeType i = 0; i < v.Size();
         i++) { // rapidjson uses SizeType instead of size_t.
        value.push_back(v[i].GetDouble());
    }
}

int
ISConstructors::AddConstructor(std::string name, ISValuePtr (*creator)())
{
    m_TheRegistry[name] = creator;

    return 0;
}

ISValuePtr
ISConstructors::GetObject(std::string name)
{
    Json::ISValuePtr (*cnsctr)() = m_TheRegistry[name];
    Json::ISValuePtr no = cnsctr();
    return no;
}
```

```
    void
    serialize(std::string filename, ISValue* top)
    {
        rapidjson::Document document;
        document.SetObject();

        ISProperties p = top->GetProperty();

        document.AddMember("TypeId", top->GetName(document),
                           document.GetAllocator());

        for (auto& element : p) {
            rapidjson::Value n;
            n.SetString(element.name, document.GetAllocator());
            if (element.value != nullptr)
                document.AddMember(n, element.value->ToDom(document),
                                   document.GetAllocator());
        }

        rapidjson::StringBuffer sb;

        rapidjson::PrettyWriter<rapidjson::StringBuffer> writer(sb);
        document.Accept(
            writer); // Accept() traverses the DOM and generates Handler events.
        std::fstream jsonout(filename, std::ios_base::out);
        jsonout « sb.GetString() « std::endl;
        jsonout.close();
    }

    void
    deserialize(std::string filename, ISValue* top)
    {
        rapidjson::Document document;

        std::ifstream ifs(filename);
        rapidjson::IStreamWrapper isw(ifs);

        document.ParseStream(isw);
        ISProperties p = top->GetProperty();
        if (document.IsObject()) {
            for (auto& element : p) {
                element.value->FromDom(document[element.name], document);
            }
        }
    }

} // namespace Json
```

12.64 src/gui/action.cpp File Reference

```
#include "gui/action.h"
#include <QAction>
#include <QWidget>
```

Namespaces

- namespace Gui

12.65 action.cpp

Go to the documentation of this file.

```
#include "gui/action.h"

#include <QAction>
#include <QWidget>

namespace Gui
{
```

```
    Action::Action(QWidget* parent, const QString& label, void (*onClick)(void),
                   const QKeySequence& shortcut)
        : QAction(parent)
    {
        setText(label);
        setShortcut(shortcut);
        QObject::connect(this, &QAction::triggered, onClick);
    }

} // namespace Gui
```

12.66 src/gui/agent_controls.cpp File Reference

```
#include "gui/agent_controls.h"
#include <QColorDialog>
#include <QLabel>
```

Namespaces

- namespace Gui

12.67 agent_controls.cpp

Go to the documentation of this file.

```
#include "gui/agent_controls.h"

#include <QColorDialog>
#include <QLabel>

namespace Gui
{
    AgentControls::AgentControls(QWidget* parent)
        : QFrame(parent), m_Layout{ new QGridLayout(this) },
          m_ActiveAgentComboBox{ new QComboBox(this) },
          m_ActiveAgentColorBox{ new ColorBox(this) },
          m_NewAgentButton{ new QPushButton(this) }, m_ActiveAgentIndex{}
    {
        setObjectName("AgentControls");

        setFrameStyle(QFrame::Panel | QFrame::Raised);

        QLabel* heading{ new QLabel(this) };
        heading->setText("Agent Controls");
        m_Layout->addWidget(heading, 0, 0, 1, 3, Qt::AlignHCenter);

        QFrame* hLine{ new QFrame(this) };
        hLine->setFrameStyle(QFrame::HLine | QFrame::Sunken);
        m_Layout->addWidget(hLine, 1, 0, 1, 3);

        QFrame* activeAgentFrame{ new QFrame(this) };
        activeAgentFrame->setFrameStyle(QFrame::Panel | QFrame::Raised);
        QGridLayout* activeAgentFrameLayout{ new QGridLayout(
          activeAgentFrame) };
        QLabel* activeAgentHeading{ new QLabel(activeAgentFrame) };
        activeAgentHeading->setText("Active agent");
        activeAgentFrameLayout->addWidget(activeAgentHeading, 0, 0, 1, 3,
                                          Qt::AlignLeft);
        m_ActiveAgentComboBox->setCursor(Qt::PointingHandCursor);
        activeAgentFrameLayout->addWidget(m_ActiveAgentComboBox, 1, 0, 1, 2);
        activeAgentFrameLayout->addWidget(m_ActiveAgentColorBox, 1, 2, 1, 1);
        m_Layout->addWidget(activeAgentFrame, 2, 0, 1, 3);

        m_NewAgentButton->setText("New agent");
        m_NewAgentButton->setCursor(Qt::PointingHandCursor);
        m_Layout->addWidget(m_NewAgentButton, 3, 0, 1, 3);

        connect(m_ActiveAgentComboBox, SIGNAL(currentIndexChanged(int)), this,
                SLOT(SetActiveAgentIndex(int)));
```

```
    connect(m_NewAgentButton, &QPushButton::clicked,
            [this]() { emit AddAgent(); });

    connect(m_ActiveAgentColorBox, SIGNAL(ColorUpdated(QColor)), this,
            SLOT(SetAgentColor(QColor)));
}

void
AgentControls::SetAgentColor(QColor color)
{
    auto agent =
        std::find_if(m_Agents.first, m_Agents.second,
                     [&](const Core::Agent& agent) { return agent.Id == m_ActiveAgentIndex; });
    if (agent != m_Agents.second) {
        agent->Color = color.name().toStdString();
    }

    emit AgentChanged(m_Agents);
}

void
AgentControls::UpdateAgents(
    std::pair<std::vector<Core::Agent>::iterator, std::vector<Core::Agent>::iterator>
        agents)
{
    m_Agents = agents;
    m_ActiveAgentComboBox->clear();
    for (auto iter{ agents.first }; iter != agents.second; ++iter) {
        QString newAgentText = "Agent " + QString::number(iter->Id);
        m_ActiveAgentComboBox->blockSignals(true);
        m_ActiveAgentComboBox->insertItem(iter->Id, newAgentText);
        m_ActiveAgentComboBox->setCurrentIndex(iter->Id);
        m_ActiveAgentComboBox->blockSignals(false);
    }
    m_ActiveAgentIndex = m_ActiveAgentComboBox->currentIndex();

    m_ActiveAgentColorBox->update();
    emit AgentChanged(m_Agents);
    emit ActiveAgentChanged(m_ActiveAgentIndex);
}

void
AgentControls::SetActiveAgentIndex(int index)
{
    if (index == -1) {
        return;
    }

    m_ActiveAgentIndex = index;
    auto agent =
        std::find_if(m_Agents.first, m_Agents.second,
                     [&](const Core::Agent& agent) { return agent.Id == index; });
    if (agent != m_Agents.second) {
        m_ActiveAgentColorBox->UpdateColor(
            QColor(QString::fromStdString(agent->Color)));
    }

    m_ActiveAgentColorBox->update();
    emit ActiveAgentChanged(m_ActiveAgentIndex);
}

void
AgentControls::SyncColor()
{
    auto agent =
        std::find_if(m_Agents.first, m_Agents.second,
                     [&](const Core::Agent& agent) { return agent.Id == m_ActiveAgentIndex; });
    if (agent != m_Agents.second) {
        m_ActiveAgentColorBox->UpdateColor(
            QColor(QString::fromStdString(agent->Color)));
    }

    m_ActiveAgentColorBox->update();
}

} // namespace Gui
```

12.68 src/gui/color_box.cpp File Reference

```
#include "gui/color_box.h"
#include <QPainter>
```

```
#include <QPainterPath>
```

Namespaces

- namespace Gui

12.69 color_box.cpp

Go to the documentation of this file.

```
#include "gui/color_box.h"

#include <QPainter>
#include <QPainterPath>

namespace Gui
{

    ColorBox::ColorBox(QWidget* parent)
        : QPushButton(parent), m_Color{ Qt::gray },
          m_ColorDialog{ new QColorDialog(this) }
    {
        setObjectName("ColorBox");
        setFixedSize(50, 50);
        setCursor(Qt::PointingHandCursor);

        m_ColorDialog->setModal(true);
    }

    void
    ColorBox::paintEvent(QPaintEvent* event)
    {
        if (m_Color.isValid()) {
            QPainter painter(this);
            painter.setRenderHint(QPainter::Antialiasing);

            int radius = qRound(width() * 0.2); // Adjust the factor as needed

            QPainterPath path;
            path.addRoundedRect(rect(), radius, radius);
            painter.fillPath(path, m_Color);

            painter.setPen({ Qt::black, 2 });
            painter.drawPath(path);
        }
    }

    void
    ColorBox::mousePressEvent(QMouseEvent* event)
    {
        SelectColor();
    }

    void
    ColorBox::UpdateColor(QColor color)
    {
        m_Color = color;
        m_ColorDialog->setCurrentColor(color);
    }

    void
    ColorBox::SelectColor()
    {
        m_ColorDialog->open();
        m_ColorDialog->raise();
        m_ColorDialog->exec();

        QColor color = m_ColorDialog->selectedColor();
        if (color.isValid()) {
            m_Color = color;
            update();
            emit ColorUpdated(m_Color);
        }
    }

} // namespace Gui
```

12.70 src/gui/keyframe_controls.cpp File Reference

```
#include "gui/keyframe_controls.h"
#include <QLabel>
```

Namespaces

- namespace Gui

12.71 keyframe_controls.cpp

Go to the documentation of this file.

```
#include "gui/keyframe_controls.h"

#include <QLabel>

namespace Gui
{

    KeyframeControls::KeyframeControls(QWidget* parent)
        : QFrame(parent), m_Layout{ new QGridLayout(this) },
          m_KeyframeList{ new KeyframeList(this) },
          m_DeleteKeyframesButton{ new QPushButton(this) }
    {
        setObjectName("KeyframeControls");

        setFrameStyle(QFrame::Panel | QFrame::Raised);

        QLabel* heading{ new QLabel(this) };
        heading->setText("Keyframe Controls");
        m_Layout->addWidget(heading, 0, 0, 1, 3, Qt::AlignHCenter);

        QFrame* hLine{new QFrame(this)};
        hLine->setFrameStyle(QFrame::HLine | QFrame::Sunken);
        m_Layout->addWidget(hLine, 1, 0, 1, 3);

        QFrame* keyframeListFrame{new QFrame(this)};
        QVBoxLayout* keyframeListFrameLayout{new QVBoxLayout(keyframeListFrame)};
        QLabel* keyframeListFrameHeading{new QLabel(keyframeListFrame)};
        keyframeListFrameHeading->setText("Keyframes");
        keyframeListFrameHeading->setAlignment(Qt::AlignLeft);
        keyframeListFrameLayout->addWidget(keyframeListFrameHeading);
        keyframeListFrameLayout->addWidget(m_KeyframeList);
        m_Layout->addWidget(keyframeListFrame, 2, 0, 5, 3);

        m_DeleteKeyframesButton->setText("Delete keyframe(s)");
        m_DeleteKeyframesButton->setCursor(Qt::PointingHandCursor);
        m_Layout->addWidget(m_DeleteKeyframesButton, 7, 0, 1, 3);

        connect(m_DeleteKeyframesButton, &QPushButton::clicked, [this]() {
            emit DeleteSelectedKeyframes();
        });
    }

} // namespace Gui
```

12.72 src/gui/keyframe_list.cpp File Reference

```
#include "gui/keyframe_list.h"
#include "keyframe_management/keyframe_manager.h"
#include <QListWidgetItem>
#include <QVariant>
```

Namespaces

- namespace Gui

12.73 keyframe_list.cpp

Go to the documentation of this file.

```
#include "gui/keyframe_list.h"

#include "keyframe_management/keyframe_manager.h"

#include <QListWidgetItem>
#include <QVariant>

namespace Gui
{

    KeyframeList::KeyframeList(QWidget* parent)
        : QListWidget(parent), m_Layout(new QVBoxLayout(this))
    {
        setObjectName("KeyframeList");
        Update();
    }

    void
    KeyframeList::Update()
    {
        clear();
        auto& keyframes =
            KeyframeManagement::KeyframeManager::Instance().GetKeyframes();
        for (auto& keyframe : keyframes) {
            QString itemText =
                QString("AgentId: %1, TimeStamp: %2, X: %3, Y: %4, Z: %5")
                    .arg(keyframe.AgentId)
                    .arg(keyframe.TimeStamp)
                    .arg(keyframe.Position.X)
                    .arg(keyframe.Position.Y)
                    .arg(keyframe.Position.Z);
            QListWidgetItem* item = new QListWidgetItem(itemText, this);
            item->setFlags(item->flags() | Qt::ItemIsUserCheckable);
            item->setCheckState(Qt::Unchecked);
            item->setData(Qt::UserRole, QVariant::fromValue(keyframe));
        }
    }

    void
    KeyframeList::DeleteSelected()
    {
        for (int i = count() - 1; i >= 0; --i) {
            QListWidgetItem* itemToCheck = item(i);
            if (itemToCheck->checkState() == Qt::Checked) {
                KeyframeManagement::KeyframeManager::Instance().RemoveKeyframe(
                    itemToCheck->data(Qt::UserRole).value<Core::Keyframe>());
                delete takeItem(i);
            }
        }
        Update();
    }

} // namespace Gui
```

12.74 src/gui/launcher.cpp File Reference

```
#include "gui/launcher.h"
#include <QLabel>
```

Namespaces

- namespace Gui

12.75 launcher.cpp

Go to the documentation of this file.

```
#include "gui/launcher.h"

#include <QLabel>

namespace Gui
{
    Launcher::Launcher(QWidget* parent)
        : QWidget(parent), m_Layout(new QVBoxLayout(this))
    {
        setObjectName("Launcher");
        QLabel* title = new QLabel(this);
        title->setText("Launcher");
        title->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
        m_Layout->addWidget(title);
    }

    Launcher::~Launcher() {}
} // namespace Gui
```

12.76 src/gui/main_content.cpp File Reference

```
#include "gui/main_content.h"
#include "coordinate_converter/coordinate_converter.h"
#include <QPushButton>
#include <QSplitter>
```

Namespaces

- namespace Gui

12.77 main_content.cpp

Go to the documentation of this file.

```
#include "gui/main_content.h"

#include "coordinate_converter/coordinate_converter.h"

#include <QPushButton>
#include <QSplitter>

namespace Gui
{
    MainContent::MainContent(QWidget* parent)
        : QWidget(parent), m_Layout(new QGridLayout(this)),
//          m_Scenario(std::make_shared<Scenario>(
//              "Test scenario", GeographicalCoordinate(59.66584230, 9.65059460),
//              2700)),
          m_Sidebar(new Sidebar), m_TabWidget(new TabWidget(this))
    {
        setObjectName("MainContent");
        QSplitter* splitter = new QSplitter(Qt::Horizontal, this);
        splitter->addWidget(m_Sidebar);
        splitter->addWidget(m_TabWidget);
        splitter->setStretchFactor(0, 1);
        splitter->setStretchFactor(1, 1000);

        splitter->setChildrenCollapsible(false);

        m_Layout->addWidget(splitter, 0, 0);
        m_Layout->setColumnStretch(0, 1);
        m_Layout->setRowStretch(0, 1);
        }

} // namespace Gui
```

12.78 src/gui/main_window.cpp File Reference

```
#include "gui/main_window.h"
#include <QDebug>
#include <QRandomGenerator>
```

Namespaces

- namespace Gui

Functions

- static QColor getRandomColor ()

12.78.1 Function Documentation

12.78.1.1 getRandomColor()

```
static QColor getRandomColor ( )  [static]
```

Definition at line 7 of file main_window.cpp.

Referenced by Gui::MainWindow::CreateNewAgent().

12.79 main_window.cpp

Go to the documentation of this file.

```
00001 #include "gui/main_window.h"
00002
00003 #include <QDebug>
00004 #include <QRandomGenerator>
00005
00006 static QColor
00007 getRandomColor()
00008 {
00009     QRandomGenerator* generator{ QRandomGenerator::global() };
00010     auto r = static_cast<float>(generator->generateDouble());
00011     auto g = static_cast<float>(generator->generateDouble());
00012     auto b = static_cast<float>(generator->generateDouble());
00013     return QColor::fromRgbF(r, g, b);
00014 }
00015
00016 namespace Gui
00017 {
00018     MainWindow::MainWindow(QWidget* parent)
00019         : QMainWindow(parent), m_MenuBar{ new MenuBar(this) },
00020           m_MainContent{ new MainContent(this) },
00021           m_Scenario{ std::make_shared<CompileScenario::Scenario>(
00022               "Untitled Scenario",
00023               Core::GeographicalCoordinate(59.66584230, 9.65059460), 2700) },
00024           m_ScenarioSettingsDialog{ new MapDialog(this) }
00025     {
00026         setObjectName("MainWindow");
00027         setWindowTitle("Hivemind");
00028         setWindowIcon(QIcon(":/icons/logo_transparent_512.png"));
```

```
    setMenuBar(m_MenuBar);
    setCentralWidget(m_MainContent);
    resize(1280, 720);

    ConnectSlotsAndSignals();
    CreateNewAgent();
}

MainWindow::~MainWindow() {}

void
MainWindow::ConnectSlotsAndSignals()
{
    // Menu bar signals
    connect(m_MenuBar, SIGNAL(SaveScenario(const std::string&)), this,
            SLOT(SaveScenario(const std::string&)));
    connect(m_MenuBar, SIGNAL(LoadScenario(const std::string&)), this,
            SLOT(LoadScenario(const std::string&)));

    // Connect keyframe list and keyframe manager
    auto keyframeList{ findChild<KeyframeList*>("KeyframeList") };
    if (keyframeList) {
        connect(&KeyframeManagement::KeyframeManager::Instance(),
                SIGNAL(KeyframeAdded()), keyframeList, SLOT(Update()));
        connect(this, SIGNAL(ScenarioLoaded()), keyframeList,
                SLOT(Update()));
    }

    auto mapViewer{ findChild<MapViewer*>("MapViewer") };
    if (mapViewer) {
        // Connect map viewer and keyframe manager
        connect(&KeyframeManagement::KeyframeManager::Instance(),
                SIGNAL(KeyframeAdded()), mapViewer, SLOT(update()));

        // Connect satellite image request and map loading
        connect(&MapManagement::MapManager::Instance(),
                &MapManagement::MapManager::RequestImage, mapViewer,
                &MapViewer::WaitForData);
        connect(&MapManagement::MapManager::Instance(),
                &MapManagement::MapManager::GotImage, mapViewer,
                &MapViewer::DataReceived);

        connect(
            this,
            SIGNAL(ScenarioCompiled(
                std::pair<CompileScenario::Scenario::RouteMap::iterator,
                          CompileScenario::Scenario::RouteMap::iterator>)),
            mapViewer,
            SLOT(UpdateRoutes(
                std::pair<CompileScenario::Scenario::RouteMap::iterator,
                          CompileScenario::Scenario::RouteMap::iterator>)));

        auto timeline{ findChild<Timeline*>("Timeline") };
        if (timeline) {
            connect(timeline, SIGNAL(timeStampSelected(float)), mapViewer,
                    SLOT(UpdateTimeStamp(float)));
        }
    }

    // Connect deletion of keyframes in GUI
    auto keyframeControls{ findChild<KeyframeControls*>(
        "KeyframeControls") };
    if (keyframeControls && keyframeList) {
        connect(keyframeControls, SIGNAL(DeleteSelectedKeyframes()),
                keyframeList, SLOT(DeleteSelected()));
    }

    auto scenarioControls{ findChild<ScenarioControls*>(
        "ScenarioControls") };
    if (scenarioControls) {
        connect(scenarioControls, SIGNAL(OpenSettingsDialog()),
                m_ScenarioSettingsDialog, SLOT(exec()));
        connect(m_ScenarioSettingsDialog,
                SIGNAL(MapDataReady(float, float, float)), this,
                SLOT(UpdateScenario(float, float, float)));
        connect(scenarioControls, SIGNAL(CompileScenario()), this,
                SLOT(CompileScenario()));
    }

    auto agentControls{ findChild<AgentControls*>("AgentControls") };
    if (agentControls) {
        connect(this, SIGNAL(SyncAgentColor()), agentControls,
                SLOT(SyncColor()));
        if (mapViewer) {
            connect(agentControls,
                    SIGNAL(AgentChanged(
                        std::pair<std::vector<Core::Agent>::iterator,
```

```
                              std::vector<Core::Agent>::iterator>)),
                   mapViewer,
                   SLOT(UpdateAgents(
                       std::pair<std::vector<Core::Agent>::iterator,
                              std::vector<Core::Agent>::iterator>)));
               connect(agentControls, SIGNAL(ActiveAgentChanged(int)),
                   mapViewer, SLOT(UpdateActiveAgent(int)));
           }

           connect(
               this,
               SIGNAL(AgentAdded(std::pair<std::vector<Core::Agent>::iterator,
                                  std::vector<Core::Agent>::iterator>)),
               agentControls,
               SLOT(UpdateAgents(std::pair<std::vector<Core::Agent>::iterator,
                                  std::vector<Core::Agent>::iterator>)));

           connect(agentControls, SIGNAL(AddAgent()), this,
                 SLOT(CreateNewAgent()));
       }
   }

   void
   MainWindow::SaveScenario(const std::string& filepath)
   {
       m_Scenario->save(filepath);
   }

   void
   MainWindow::LoadScenario(const std::string& filepath)
   {
       m_Scenario->load(filepath);
       emit AgentAdded(m_Scenario->GetAgents());
       emit ScenarioLoaded();
       update();
   }

   void
   MainWindow::UpdateScenario(float latitude, float longitude, float size)
   {
       Core::GeographicalCoordinate coord{ latitude, longitude };
       m_Scenario->SetOrigin(coord, static_cast<int>(size));
   }

   void
   MainWindow::CompileScenario()
   {
       m_Scenario->Compile();
       auto routes = m_Scenario->GetRoutes();
       emit ScenarioCompiled(m_Scenario->GetRoutes());
   }

   void
   MainWindow::CreateNewAgent()
   {
       int maxId{ -1 };
       auto agents = m_Scenario->GetAgents();
       for (auto iter{ agents.first }; iter != agents.second; ++iter) {
           maxId = std::max(maxId, iter->Id);
       }
       int id{ maxId == -1 ? 0 : maxId + 1 };

       std::string color = getRandomColor().name().toStdString();
       m_Scenario->AddAgent({ id, "Untitled agent", color });
       emit AgentAdded(m_Scenario->GetAgents());
       emit SyncAgentColor();
   }

} // namespace Gui
```

12.80 src/gui/map_dialog.cpp File Reference

```
#include "gui/map_dialog.h"
#include "coordinate_converter/coordinate_converter.h"
#include <QLabel>
#include <QtGui>
#include <QtWidgets>
```

Namespaces

- namespace Gui

12.81 map_dialog.cpp

Go to the documentation of this file.

```
#include "gui/map_dialog.h"
//#include "keyframe_management/keyframe_management.h"
#include "coordinate_converter/coordinate_converter.h"
#include <QLabel>
#include <QtGui>
#include <QtWidgets>

namespace Gui
{
    MapDialog::MapDialog(QWidget* parent) : QDialog(parent)
    {
        QVBoxLayout* layout = new QVBoxLayout(this);

        QLabel* xCoordinate = new QLabel("Enter latitude:", this);
        layout->addWidget(xCoordinate);
        m_LatitudeCoordInput = new QLineEdit(this);
        layout->addWidget(m_LatitudeCoordInput);

        QLabel* yCoordinate = new QLabel("Enter longitude:", this);
        layout->addWidget(yCoordinate);
        m_LongitudeCoordInput = new QLineEdit(this);
        layout->addWidget(m_LongitudeCoordInput);

        QLabel* size = new QLabel("Size:", this);
        layout->addWidget(size);
        m_SizeInput = new QLineEdit(this);
        layout->addWidget(m_SizeInput);


        QPushButton* finishButton = new QPushButton("Set location", this);
        layout->addWidget(finishButton);

        QObject::connect(finishButton, SIGNAL(clicked()), this, SLOT(Finish()));

        QObject::connect(finishButton, SIGNAL(clicked()), this, SLOT(accept()));

        layout->addStretch(1);
    }

    void
    MapDialog::Finish()
    {
        bool conversionOk;

        float x = m_LatitudeCoordInput->text().toFloat(&conversionOk);
        if (!conversionOk)
            return;

        float y = m_LongitudeCoordInput->text().toFloat(&conversionOk);
        if (!conversionOk)
            return;
        float size = m_SizeInput->text().toFloat(&conversionOk);
        if (!conversionOk)
            return;

        emit MapDataReady(x, y, size);
        emit Finished();

        m_LatitudeCoordInput->clear();
        m_LongitudeCoordInput->clear();
        m_SizeInput->clear();

    }
} // namespace Gui
```

12.82 src/gui/map_viewer.cpp File Reference

```
#include "gui/map_viewer.h"
#include "gui/main_window.h"
```

```
#include <QPainter>
#include <QRandomGenerator>
#include <QTimer>
```

Namespaces

- namespace Gui

12.83 map_viewer.cpp

Go to the documentation of this file.

```
#include "gui/map_viewer.h"

#include "gui/main_window.h"

#include <QPainter>
#include <QRandomGenerator>
#include <QTimer>

namespace Gui
{

    MapViewer::MapViewer(QWidget* parent)
        : QWidget(parent), m_WaitingForData(true),
          m_WaitingForDataTimer(new QTimer(this)), m_LoaderAngle(0),
          m_LoaderSize(100), m_LoaderSpeed(180.0f), m_LoaderSpan(270.0f),
          m_LoaderThickness(8), m_StartX{}, m_StartY{}, m_Size{},
          m_ActiveAgentId{}, m_TimeStamp{}
    {
        setObjectName("MapViewer");

        connect(m_WaitingForDataTimer, &QTimer::timeout, this, [this]() {
            qint64 elapsedMilliseconds = m_WaitingForDataElapsedTimer.elapsed();
            float deltaTimeSeconds =
                static_cast<float>(elapsedMilliseconds) / 1000.0f;
            m_LoaderAngle -= m_LoaderSpeed * deltaTimeSeconds;
            m_WaitingForDataElapsedTimer.restart();
            update();
        });

        setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
        UpdateRenderingArea();

        m_WaitingForDataElapsedTimer.start();
        WaitForData();
    }

    void
    MapViewer::paintEvent(QPaintEvent* event)
    {
        QPainter painter(this);

        if (m_WaitingForData) {
            DrawLoader(painter);
            return;
        }

        QByteArray mapData = MapManagement::MapManager::GetData();
        if (!mapData.isEmpty()) {
            int pixmapResolution{
                MapManagement::MapManager::GetImageResolution()
            };
            QPixmap pixmap(pixmapResolution, pixmapResolution);
            pixmap.loadFromData(mapData);
            painter.drawPixmap(m_StartX, m_StartY,
                               pixmap.scaled(m_Size, m_Size));
        }

        DrawRoutes(painter);
        DrawKeyframes(painter);
    }

    void
    MapViewer::resizeEvent(QResizeEvent* event)
    {
```

```
    UpdateRenderingArea();
}

void
MapViewer::mousePressEvent(QMouseEvent* event)
{
    event->ignore();

    // Only respond to left mouse button clicks
    if (event->button() != Qt::LeftButton) {
        return;
    }

    // Ignore clicks if they are outside the rendering area
    int x = static_cast<int>(event->position().x());
    int y = static_cast<int>(event->position().y());
    bool contained = (x >= m_StartX && x < (m_StartX + m_Size) &&
                      y >= m_StartY && y < (m_StartY + m_Size));
    if (!contained) {
        return;
    }

    // Relative coordinates of mouse click within the rendering area
    float xRel{ static_cast<float>(x - m_StartX) };
    float yRel{ static_cast<float>(y - m_StartY) };

    float size{ static_cast<float>(
        CoordinateConverter::CoordConv::GetSize()) };

    // Find relative coordinate within scenario space
    xRel = xRel * size / static_cast<float>(m_Size);
    yRel = yRel * size / static_cast<float>(m_Size);
    Core::CartesianCoordinate symmetricPosition{
        CoordinateConverter::CoordConv::AsymmetricToSymmetric(
            { xRel, yRel, 0 })
    };

    Core::Keyframe newKeyframe(m_ActiveAgentId, m_TimeStamp,
                               symmetricPosition);
    KeyframeManagement::KeyframeManager::Instance().AddKeyframe(
        newKeyframe);
}

void
MapViewer::UpdateRenderingArea()
{
    int maxWidth = width();
    int maxHeight = height();

    m_Size = std::min(maxWidth, maxHeight);
    int marginX = maxWidth - m_Size;
    int marginY = maxHeight - m_Size;

    m_StartX = marginX / 2;
    m_StartY = marginY / 2;
}

void
MapViewer::WaitForData()
{
    m_WaitingForData = true;
    m_WaitingForDataTimer->start(16);
    setCursor(Qt::WaitCursor);
    update();
}

void
MapViewer::DataReceived()
{
    m_WaitingForData = false;
    m_WaitingForDataTimer->stop();
    setCursor(Qt::ArrowCursor);
    update();
}

void
MapViewer::DrawKeyframes(QPainter& painter)
{
    int radius = 8;
    int scenarioSize{ CoordinateConverter::CoordConv::GetSize() };
    auto keyframes =
        KeyframeManagement::KeyframeManager::Instance().GetKeyframes();
    for (const Core::Keyframe& keyframe : keyframes) {
        Core::CartesianCoordinate keyframePositionAsymmetric{
            CoordinateConverter::CoordConv::SymmetricToAsymmetric(
                keyframe.Position)
        };
```

```
        int x{ static_cast<int>(
            keyframePositionAsymmetric.X / scenarioSize * m_Size +
            m_StartX - static_cast<float>(radius) / 2.0f) };
        int y{ static_cast<int>(
            keyframePositionAsymmetric.Y / scenarioSize * m_Size +
            m_StartY - static_cast<float>(radius) / 2.0f) };

        auto agent = std::find_if(m_Agents.first, m_Agents.second,
                                  [&](const Core::Agent& agent) {
                                      return agent.Id == keyframe.AgentId;
                                  });
        QColor color(Qt::magenta);
        if (agent != m_Agents.second) {
            color = QColor(QString::fromStdString(agent->Color));
        }

        painter.setPen(Qt::black);
        painter.setBrush({ color });
        painter.drawEllipse(x, y, radius, radius);
    }
}

void
MapViewer::DrawRoutes(QPainter& painter)
{
    QPen pen(Qt::red, 2);
    painter.setPen(pen);
    painter.setRenderHint(QPainter::Antialiasing);

    for (auto iter = m_Routes.first; iter != m_Routes.second; ++iter) {
        int agentId = iter->first;
        auto route = iter->second;
        for (int j = 0; j < route.size(); j++) {
            for (int k = 0; k < route[j].size() - 1; k++) {
                Core::CartesianCoordinate asymmetricA =
                    CoordinateConverter::CoordConv::SymmetricToAsymmetric(
                        route[j][k]);
                Core::CartesianCoordinate asymmetricB =
                    CoordinateConverter::CoordConv::SymmetricToAsymmetric(
                        route[j][k + 1]);

                int x1{ static_cast<int>(
                    asymmetricA.X /
                        CoordinateConverter::CoordConv::GetSize() * m_Size +
                    m_StartX) };
                int y1{ static_cast<int>(
                    asymmetricA.Y /
                        CoordinateConverter::CoordConv::GetSize() * m_Size +
                    m_StartY) };
                int x2{ static_cast<int>(
                    asymmetricB.X /
                        CoordinateConverter::CoordConv::GetSize() * m_Size +
                    m_StartX) };
                int y2{ static_cast<int>(
                    asymmetricB.Y /
                        CoordinateConverter::CoordConv::GetSize() * m_Size +
                    m_StartY) };

                auto agent = std::find_if(m_Agents.first, m_Agents.second,
                                          [&](const Core::Agent& agent) {
                                              return agent.Id == agentId;
                                          });
                QColor color(Qt::magenta);
                if (agent != m_Agents.second) {
                    color = QColor(QString::fromStdString(agent->Color));
                }

                painter.setPen({ color, 2 });
                painter.drawLine(x1, y1, x2, y2);
            }
        }
    }
}

void
MapViewer::DrawLoader(QPainter& painter) const
{
    QColor hivemindOrange(227, 118, 39);
    QPen pen(hivemindOrange, m_LoaderThickness, Qt::SolidLine,
             Qt::RoundCap);
    painter.setPen(pen);
    painter.setRenderHint(QPainter::Antialiasing);

    int x{ m_StartX + (m_Size - m_LoaderSize) / 2 };
    int y{ m_StartY + (m_Size - m_LoaderSize) / 2 };

    QRectF rectangle(x, y, m_LoaderSize, m_LoaderSize);
```

```

        // Multiply angles by 16 because Qt's angles are specified 1/16th of a
        // degree
        int spanAngle = static_cast<int>(m_LoaderSpan) * 16;
        int startAngle = static_cast<int>(m_LoaderAngle) * 16;
        painter.drawArc(rectangle, startAngle, spanAngle);
    }

    void
    MapViewer::UpdateRoutes(
        std::pair<CompileScenario::Scenario::RouteMap::iterator,
                  CompileScenario::Scenario::RouteMap::iterator>
            routes)
    {
        m_Routes = routes;
        update();
    }

    void
    MapViewer::UpdateAgents(std::pair<std::vector<Core::Agent>::iterator,
                                      std::vector<Core::Agent>::iterator>
                                agents)
    {
        m_Agents = agents;
        update();
    }

} // namespace Gui
```

12.84 src/gui/menu_bar.cpp File Reference

```
#include "gui/menu_bar.h"
#include "gui/action.h"
#include "gui/main_content.h"
#include <QFileDialog>
```

Namespaces

- namespace Gui

Functions

- void quitApp (void)

12.84.1 Function Documentation

12.84.1.1 quitApp()

```
void quitApp (
        void )
```

12.85 menu_bar.cpp

Go to the documentation of this file.

```
#include "gui/menu_bar.h"

#include "gui/action.h"
#include "gui/main_content.h"

#include <QFileDialog>

extern void quitApp(void);

namespace Gui
{

    MenuBar::MenuBar(QWidget* parent) : QMenuBar(parent)
    {
        QMenu* menu = new QMenu(this);
        menu->setTitle("File");

        Action* newFile = new Action(
            this, QString::fromUtf8("New"), [] {}, QKeySequence::New);
        menu->addAction(newFile);

        auto open = new QAction(this);
        open->setText("Open...");
        open->setShortcut(QKeySequence::Open);
        connect(open, &QAction::triggered, [this]() {
            QString filename = QFileDialog::getOpenFileName(
                this->window(), "Open scenario", QDir::currentPath(),
                "Hivemind Scenario Files (*.hmsc)");
            if (filename != "") {
                emit this->LoadScenario(filename.toStdString());
            }
        });
        menu->addAction(open);

        auto saveAs = new QAction(this);
        saveAs->setText("Save as...");
        saveAs->setShortcut(QKeySequence::SaveAs);
        QObject::connect(saveAs, &QAction::triggered, [this]() {
            QString fileName = QFileDialog::getSaveFileName(
                this->window(), QString::fromUtf8("Save scenario"),
                QDir::currentPath(),
                QString::fromUtf8("Hivemind Scenario Files (*.hmsc)"));
            if (fileName != "") {
                emit this->SaveScenario(fileName.toStdString());
            }
        });
        menu->addAction(saveAs);

        Action* save = new Action(
            this, QString::fromUtf8("Save"), [] {}, QKeySequence::Save);
        menu->addAction(save);

        addAction(menu->menuAction());
    }

} // namespace Gui
```

12.86 src/gui/planner.cpp File Reference

```
#include "gui/planner.h"
```

Namespaces

- namespace Gui

12.87 planner.cpp

Go to the documentation of this file.

```
#include "gui/planner.h"

namespace Gui
{
    Planner::Planner(QWidget* parent)
        : QSplitter(Qt::Vertical, parent),
          m_MapViewer(new MapViewer(this)),
          m_Timeline(new Timeline(this))
    {
        addWidget(m_MapViewer);
        addWidget(m_Timeline);

        setStretchFactor(0, 1000);
        setStretchFactor(1, 1);

        setChildrenCollapsible(false);
    }

    Planner::~Planner() {}
} // namespace Gui
```

12.88 src/gui/scenario_controls.cpp File Reference

```
#include "gui/scenario_controls.h"
#include <QLabel>
```

Namespaces

- namespace Gui

12.89 scenario_controls.cpp

Go to the documentation of this file.

```
#include "gui/scenario_controls.h"

#include <QLabel>

namespace Gui
{

    ScenarioControls::ScenarioControls(QWidget* parent)
        : QFrame(parent), m_Layout{ new QGridLayout(this) },
          m_SettingsButton{ new QPushButton(this) },
          m_CompileButton{ new QPushButton(this) }
    {
        setObjectName("ScenarioControls");

        setFrameStyle(QFrame::Panel | QFrame::Raised);

        QLabel* heading{ new QLabel(this) };
        heading->setText("Scenario settings");
        m_Layout->addWidget(heading, 0, 0, 1, 3, Qt::AlignHCenter);

        QFrame* hLine{ new QFrame(this) };
        hLine->setFrameStyle(QFrame::HLine | QFrame::Sunken);
        m_Layout->addWidget(hLine, 1, 0, 1, 3);

        m_CompileButton->setText("Compile scenario");
        m_CompileButton->setCursor(Qt::PointingHandCursor);
        m_Layout->addWidget(m_CompileButton, 2, 0, 1, 3);

        m_SettingsButton->setText("Scenario settings");
        m_SettingsButton->setCursor(Qt::PointingHandCursor);
        m_Layout->addWidget(m_SettingsButton, 3, 0, 1, 3);
```

```
        connect(m_SettingsButton, &QPushButton::clicked,
                [this]() { emit OpenSettingsDialog(); });

        connect(m_CompileButton, &QPushButton::clicked,
                [this]() { emit CompileScenario(); });
    }

} // namespace Gui
```

12.90 src/gui/sidebar.cpp File Reference

```
#include "gui/sidebar.h"
#include "compile_scenario/scenario.h"
#include "gui/keyframe_list.h"
#include "gui/map_dialog.h"
#include "gui/tab_widget.h"
#include "keyframe_management/keyframe_manager.h"
#include <QDialog>
#include <QLabel>
#include <QPushButton>
#include <iostream>
```

Namespaces

- namespace Gui

12.91 sidebar.cpp

Go to the documentation of this file.

```
#include "gui/sidebar.h"

#include "compile_scenario/scenario.h"
#include "gui/keyframe_list.h"
#include "gui/map_dialog.h"
#include "gui/tab_widget.h"
#include "keyframe_management/keyframe_manager.h"

#include <QDialog>
#include <QLabel>
#include <QPushButton>

#include <iostream>

namespace Gui
{
    Sidebar::Sidebar(QWidget* parent)
        : QWidget(parent), m_Layout(new QVBoxLayout(this)),
          m_ScenarioControls{ new ScenarioControls(this) },
          m_AgentControls{ new AgentControls(this) },
          m_KeyframeControls{ new KeyframeControls(this) }
    {
        setObjectName("Sidebar");

        m_Layout->addStretch(2);

        QLabel* logoLabel = new QLabel(this);
        QPixmap logoPixmap(":/icons/logo_transparent_512.png");
        logoLabel->setPixmap(logoPixmap.scaled(QSize(100, 100)));
        logoLabel->setAlignment(Qt::AlignTop | Qt::AlignHCenter);
        m_Layout->addWidget(logoLabel);

        m_Layout->addStretch(1);
        m_Layout->addWidget(m_ScenarioControls);
        m_Layout->addStretch(1);
        m_Layout->addWidget(m_AgentControls);
        m_Layout->addStretch(1);
        m_Layout->addWidget(m_KeyframeControls);
        m_Layout->addStretch(2);
    }
} // namespace Gui
```

12.92 src/gui/simulator.cpp File Reference

```
#include "gui/simulator.h"
#include <QLabel>
```

Namespaces

- namespace Gui

12.93 simulator.cpp

Go to the documentation of this file.

```
#include "gui/simulator.h"

#include <QLabel>

namespace Gui
{
    Simulator::Simulator(QWidget* parent)
        : QWidget(parent),
          m_Layout(new QGridLayout(this))
    {
        setObjectName("Simulator");
                QLabel* title = new QLabel(this);
                title->setText("Simulator");
                title->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
                m_Layout->addWidget(title);

    }

    Simulator::~Simulator() {}
} // namespace Gui
```

12.94 src/gui/tab_widget.cpp File Reference

```
#include "gui/tab_widget.h"
#include "gui/map_viewer.h"
```

Namespaces

- namespace Gui

12.95 tab_widget.cpp

Go to the documentation of this file.

```
#include "gui/tab_widget.h"

#include "gui/map_viewer.h"

namespace Gui
{
    TabWidget::TabWidget(QWidget* parent)
        : QTabWidget(parent),
          m_Simulator(new Simulator(this)), m_Launcher(new Launcher(this)),
           m_Planner(new Planner(this))
```

```
    {
        setObjectName("TabWidget");

        addTab(m_Planner, "Planner");
        addTab(m_Simulator, "Simulator");
        addTab(m_Launcher, "Launcher");

    }

    TabWidget::~TabWidget() {}
} // namespace Gui
```

12.96 src/gui/timeline.cpp File Reference

```
#include "gui/timeline.h"
#include "keyframe_management/keyframe_manager.h"
#include <QComboBox>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QMouseEvent>
#include <QPaintEvent>
#include <QPainter>
```

Namespaces

- namespace Gui

12.97 timeline.cpp

Go to the documentation of this file.

```
#include "gui/timeline.h"
#include "keyframe_management/keyframe_manager.h"
#include <QComboBox>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QMouseEvent>
#include <QPaintEvent>
#include <QPainter>

namespace Gui
{

    Timeline::Timeline(QWidget* parent)
        : QWidget(parent), m_timeStamp(0.0f), m_activeAgentId(1),
          m_pixelsPerSecond(11.75)
    {
        setObjectName("Timeline");
        setMinimumHeight(100);

        QObject::connect(&KeyframeManagement::KeyframeManager::Instance(),
                         SIGNAL(KeyframeAdded()), this, SLOT(update()));
    }

    void
    Timeline::paintEvent(QPaintEvent* event)
    {
        QPainter painter(this);
        painter.setBrush(Qt::black);
        painter.setPen(Qt::black);
        painter.drawRect(0, 0, width(), height());
        int numDivisions = 20;
        float increment = width() / (float)numDivisions;
        for (int i = 0; i <= numDivisions; ++i) {
            float xPos = i * increment;
            painter.setPen(Qt::lightGray);
            painter.drawLine(xPos, 0, xPos, height());
```

```
            QString timestampText = QString::number(i * 5);
            painter.setPen(Qt::lightGray);
            painter.drawText(QPointF(xPos + 2, height() - 5), timestampText);
        }
        float xPos = (m_timeStamp / 100.0f) * width();
        m_pixelsPerSecond = width() / 100.0f;
        painter.setBrush(Qt::red);
        painter.setPen(Qt::red);
        painter.drawLine(xPos, 0, xPos, height());

        int squareSize = 10;

        const std::vector<Core::Keyframe>& keyframes =
            KeyframeManagement::KeyframeManager::Instance().GetKeyframes();
        float secondsPerPixel = 1.0f / m_pixelsPerSecond;
        for (const Core::Keyframe& kf : keyframes) {
            float timeStamp = kf.TimeStamp;

            int x = static_cast<int>((timeStamp / secondsPerPixel) -
                                     (squareSize / 2));
            int y = height() / 2 - squareSize / 2;

            painter.setPen(Qt::NoPen);
            painter.setBrush(Qt::red);
            painter.drawRect(x, y, squareSize, squareSize);
        }
    }

    void
    Timeline::mouseReleaseEvent(QMouseEvent* event)
    {
        float xPosition = event->position().x();
        float timeStamp = (xPosition / width()) * 100.0f;

        bool keyframeClicked = false;

        if (event->button() == Qt::RightButton) {
            int squareSize = 10;
            float secondsPerPixel = 1.0f / m_pixelsPerSecond;

            const std::vector<Core::Keyframe>& keyframes =
                KeyframeManagement::KeyframeManager::Instance().GetKeyframes();

            for (size_t i = 0; i < keyframes.size(); ++i) {
                const Core::Keyframe& kf = keyframes[i];
                float kfTimeStamp = kf.TimeStamp;

                int x = static_cast<int>((kfTimeStamp / secondsPerPixel) -
                                         (squareSize / 2));
                int y = height() / 2 - squareSize / 2;

                QRect keyframeRect(x, y, squareSize, squareSize);

                if (keyframeRect.contains(event->pos())) {
                    QMessageBox::StandardButton reply = QMessageBox::question(
                        this, "Delete keyframe",
                        "Do you want to delete this keyframe?",
                        QMessageBox::Yes | QMessageBox::No);

                    if (reply == QMessageBox::Yes) {
                        KeyframeManagement::KeyframeManager::Instance()
                            .RemoveKeyframe(kf);
                        update();
                    }

                    keyframeClicked = true;
                    break;
                }
            }
        }

        if (!keyframeClicked) {
            m_timeStamp = timeStamp;
            update();
            emit timeStampSelected(m_timeStamp);
        }
    }

    void
    Timeline::resizeEvent(QResizeEvent* event)
    {
        update();
        QWidget::resizeEvent(event);
    }

} // namespace Gui
```

12.98 src/height_management/height_manager.cpp File Reference

```
#include "height_management/height_manager.h"
#include <gdal.h>
#include <gdal_priv.h>
```

Namespaces

- namespace HeightManagement

12.99 height_manager.cpp

Go to the documentation of this file.

```
#include "height_management/height_manager.h"

#include <gdal.h>
#include <gdal_priv.h>

namespace HeightManagement
{

    HeightManager::HeightManager() {}

    void
    HeightManager::LoadTif(const char* filePath, double x, double y)
    {
        m_CachedTifName = filePath;
        Core::UTMCoordinate UTMCoord{ x, y };
        UpdateOrigin(UTMCoord, m_Size);
        return;
    }

    void
    HeightManager::UpdateOrigin(Core::UTMCoordinate UTMCoord, int size)
    {
        m_Origo = { UTMCoord.Easting, UTMCoord.Northing, 0 };
        m_Size = size;
        m_Vertices = new heightdata[m_Size * m_Size];

        UpdateCornerCoords();

        if (!OrigoWithinBounds(UTMCoord.Easting, UTMCoord.Northing)) {
            // If I get the WCS request to work, that will be initialized here!
            std::cerr
                « "Selected origin not within bounds! Please ensure x is "
                  "within the range "
                « m_UpperLeftX « " - " « m_LowerRightX
                « " and Y is within the range " « long(m_LowerRightY) « " - "
                « long(m_UpperLeftY) « " (" « m_CoordinateSystem « ")"
                « "." « std::endl;
            return;
        }

        PopulateVertices();
        m_Origo.z = GetHeightAbsolute(UTMCoord.Easting, UTMCoord.Northing);
    }

    void
    HeightManager::PopulateVertices()
    {
        // Opening the dataset
        GDALAllRegister();

        GDALDataset* dataset =
            (GDALDataset*)GDALOpen(m_CachedTifName, GA_ReadOnly);
        if (dataset == NULL) {
            std::cerr « "Failed to open file" « std::endl;
        }

        // Extracting raster band data. Elevation data is located on band 1 as a
        // rule for GeoTiff files.
        GDALRasterBand* band = dataset->GetRasterBand(1);
```

```
    if (band == NULL) {
        std::cerr « "Failed to get raster band" « std::endl;
    }

    // Converting and extracting data from pixels/lines to coordinates
    double geoTransform[6];
    dataset->GetGeoTransform(geoTransform);

    // The corners for the entire dataset can be found on [0] and [3] of the
    // geotransformed array.
    double upperLeftX = geoTransform[0];
    double upperLeftY = geoTransform[3];

    // Defining the upper right corner of our selection. Because the origin
    // point is in the center of the dataset, the distance from the origin
    // x, y to each corner is half of the total size of the subset.
    double selectionCornerX = (m_Origo.x - m_Size / 2);
    double selectionCornerY = (m_Origo.y + m_Size / 2);

    // Updating the member variable for the selected subset's top left
    // corner coordinate for use in other methods.
    m_SelectionCorner = { selectionCornerX, selectionCornerY, 0 };

    // Extracting height values from the band containing height data
    // (elevationData). This is placed in a one-dimensional array
    // elevationData.
    int xOffset = (selectionCornerX - upperLeftX);
    int yOffset = (upperLeftY - selectionCornerY);

    float* elevationData = new float[m_Size * m_Size];

    CPLErr result =
        band->RasterIO(GF_Read, xOffset, yOffset, m_Size, m_Size,
                       elevationData, m_Size, m_Size, GDT_Float32, 0, 0);

    // Placing height data into member variable for use in other methods.
    // The method to find any given point in a coordinate system with 0, 0
    // in the top left corner is (y coordinate * size of one dimension of
    // the imagined two-dimensional array) + x coordinate. If the array is
    // 500*500 in size and you want the height for the (5, 10) coordinate,
    // the calculation will be (10 * 500) + 5.
    for (int yDex = 0; yDex < m_Size; yDex++) {
        for (int xDex = 0; xDex < m_Size; xDex++) {
            m_Vertices[yDex * m_Size + xDex].x = selectionCornerX + xDex;
            m_Vertices[yDex * m_Size + xDex].y = selectionCornerY - yDex;
            m_Vertices[yDex * m_Size + xDex].z =
                elevationData[yDex * m_Size + xDex];
        }
    }

    // Cleaning up and closing dataset.
    delete[] elevationData;
    GDALClose(dataset);
}

bool
HeightManager::GetVertex(int inputRelativeX, int inputRelativeY,
                         HeightManager::heightdata& vertex)
{
    if (ValidInput(inputRelativeX, inputRelativeY)) {
        vertex.z = m_Vertices[inputRelativeY * m_Size + inputRelativeX].z;
        return true;
    }

    else {
        std::cerr « "Request out of bounds" « std::endl;
        return false;
    }
}

bool
HeightManager::GetVertexAbsolute(double inputX, double inputY,
                                 HeightManager::heightdata& vertex)
{
    if (ValidInput(inputX, inputY)) {
        double inputOffsetX = inputX - m_SelectionCorner.x;
        double inputOffsetY = m_SelectionCorner.y - inputY;
        vertex.z = m_Vertices[int(inputOffsetY * m_Size + inputOffsetX)].z;
        return true;
    }

    else {
        std::cerr « "Request out of bounds" « std::endl;
        return false;
    }
}

```

```
float
HeightManager::GetHeightAbsolute(double inputX, double inputY)
{
    if (ValidInput(inputX, inputY)) {
        double inputOffsetX = inputX - m_SelectionCorner.x;
        double inputOffsetY = m_SelectionCorner.y - inputY;
        return m_Vertices[int(inputOffsetY * m_Size + inputOffsetX)].z;
    }

    else {
        std::cerr « "Request out of bounds" « std::endl;
        return 0;
    }
}

bool
HeightManager::GetHeight(int inputRelativeX, int inputRelativeY,
                         float& height)
{
    if (ValidInput(inputRelativeX, inputRelativeY)) {
        height = m_Vertices[inputRelativeY * m_Size + inputRelativeX].z;
        return true;
    }

    else {
        std::cerr « "Request out of bounds" « std::endl;
        return false;
    }
}

bool
HeightManager::ValidInput(int x, int y)
{
    bool validInput = (0 <= y) && (y < m_Size) && (0 <= x) && (x < m_Size);
    return validInput;
}

bool
HeightManager::ValidInput(double x, double y)
{
    bool validInput = (m_SelectionCorner.x <= x) &&
                      (x <= (m_SelectionCorner.x + m_Size)) &&
                      (m_SelectionCorner.y >= y) &&
                      (y >= m_SelectionCorner.y - m_Size);
    return validInput;
}

bool
HeightManager::OrigoWithinBounds(double x, double y)
{
    double min_x = m_UpperLeftX + m_Size / 2;
    double max_x = m_LowerRightX - m_Size / 2;
    double min_y = m_LowerRightY + m_Size / 2;
    double max_y = m_UpperLeftY - m_Size / 2;

    return ((x <= max_x) && (x >= min_x) && (y <= max_y) && (y >= min_y));
}

void
HeightManager::UpdateCornerCoords()
{
    GDALAllRegister();

    GDALDataset* dataset =
        (GDALDataset*)GDALOpen(m_CachedTifName, GA_ReadOnly);
    if (dataset == NULL) {
        std::cerr « "Failed to open file" « std::endl;
    }

    GDALRasterBand* band = dataset->GetRasterBand(1);
    if (band == NULL) {
        std::cerr « "Failed to get raster band" « std::endl;
    }

    double geoTransform[6];
    dataset->GetGeoTransform(geoTransform);

    m_UpperLeftX = geoTransform[0];
    m_UpperLeftY = geoTransform[3];
    m_LowerRightX =
        m_UpperLeftX + geoTransform[1] * dataset->GetRasterXSize();
    m_LowerRightY =
        m_UpperLeftY + geoTransform[5] * dataset->GetRasterXSize();

    GDALClose(dataset);
}
```

```
} // namespace HeightManagement
```

12.100 src/keyframe_management/keyframe_manager.cpp File Reference

```
#include "keyframe_management/keyframe_manager.h"
#include <iostream>
```

Namespaces

- namespace KeyframeManagement

12.101 keyframe_manager.cpp

Go to the documentation of this file.

```
#include "keyframe_management/keyframe_manager.h"

#include <iostream>

namespace KeyframeManagement
{

    void
    KeyframeManager::AddKeyframe(int agentId, float timeStamp, float x, float y,
                                 float z)
    {
        Core::CartesianCoordinate position = { x, y, z };
        Core::Keyframe newKeyframe = { agentId, timeStamp, position };
        AddKeyframe(newKeyframe);
    }

    void
    KeyframeManager::AddKeyframe(int agentId, float timeStamp,
                                 Core::CartesianCoordinate position)
    {
        Core::Keyframe newKeyframe = { agentId, timeStamp, position };
        AddKeyframe(newKeyframe);
    }

    // This function iterate over each keyframe in the m_Keyframes vector and
    // check if the timestamp and agent ID of the keyframe match the provided
    // keyframe. If a match is found, update the position of the existing
    // keyframe
    void
    KeyframeManager::AddKeyframe(Core::Keyframe& keyframe)
    {
        bool exists = false;
        for (Core::Keyframe& kf : m_Keyframes) {
            if (keyframe.TimeStamp == kf.TimeStamp &&
                keyframe.AgentId == kf.AgentId) {
                kf.Position = keyframe.Position;
                exists = true;
            }
        }
        // If no existing keyframe with the same timestamp and agent ID is
        // found, add the new keyframe
        if (!exists) {
            m_Keyframes.push_back(keyframe);
        }

        emit KeyframeAdded();
    }

    void
    KeyframeManager::RemoveKeyframe(const Core::Keyframe& keyframe)
    {
        for (auto it = m_Keyframes.begin(); it != m_Keyframes.end(); ++it) {
            if (it->AgentId == keyframe.AgentId &&
```

```
                it->TimeStamp == keyframe.TimeStamp &&
                it->Position.X == keyframe.Position.X &&
                it->Position.Y == keyframe.Position.Y &&
                it->Position.Z == keyframe.Position.Z) {
                m_Keyframes.erase(it);
                break;
            }
        }
    }

    void
    KeyframeManager::DebugDump(void) const
    {
        std::cout « "DebugDump called. Number of keyframes: "
                  « m_Keyframes.size() « std::endl;
        std::cout « "Keyframes:" « std::endl;
        for (const Core::Keyframe& kf : m_Keyframes) {
            std::cout « "AgentId: " « kf.AgentId
                      « " TimeStamp: " « kf.TimeStamp
                      « " X: " « kf.Position.X « " Y: " « kf.Position.Y
                      « " Z: " « kf.Position.Z « std::endl;
        }
    }

} // namespace KeyframeManagement
```

12.102 src/main.cpp File Reference

```
#include "compile_scenario/scenario.h"
#include "gui/main_window.h"
#include <QApplication>
#include <QFile>
#include <QFont>
#include <QFontDatabase>
#include <iostream>
```

Functions

- int main (int argc, char *argv[ ])

12.102.1 Function Documentation

12.102.1.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

Definition at line 12 of file main.cpp.

12.103 main.cpp

Go to the documentation of this file.

```
#include "compile_scenario/scenario.h"
#include "gui/main_window.h"

#include <QApplication>
#include <QFile>
#include <QFont>
#include <QFontDatabase>

#include <iostream>

int
main(int argc, char* argv[])
{
    QApplication* app = new QApplication(argc, argv);

//    QFile file(":style/darkstyle.qss");
//    file.open(QFile::ReadOnly);
//    QString styleSheet = QLatin1String(file.readAll());
//    app->setStyleSheet(styleSheet);

//    int id = QFontDatabase::addApplicationFont(":/fonts/Poppins-Medium.ttf");
//    QString family = QFontDatabase::applicationFontFamilies(id).at(0);
//    QFont poppins(family);
//    poppins.setStyleHint(QFont::Monospace);
//    poppins.setPointSize(12);
//    app->setFont(poppins);

    Gui::MainWindow* mainWindow = new Gui::MainWindow;
    mainWindow->showMaximized();

    int ret = app->exec();
    delete mainWindow;
    delete app;

    return 0;
}
```

12.104 src/map_management/map_manager.cpp File Reference

```
#include "map_management/map_manager.h"
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QNetworkRequest>
#include <QtNetwork>
#include <vector>
```

Namespaces

- namespace MapManagement

12.105 map_manager.cpp

Go to the documentation of this file.

```
#include "map_management/map_manager.h"

#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QNetworkRequest>
#include <QtNetwork>
#include <vector>

namespace MapManagement
```

```
{

    //"GetMap" retrieves map data from geonorges WMS service. It sets up a
    // network
    // request to fetch the map image and processes the response to store the
    // data.
    void
    MapManager::GetMap(Core::UTMCoordinate coord, int size)
    {
        // Create a QNetworkAccessManager object for making network requests
        QNetworkAccessManager* manager = new QNetworkAccessManager(nullptr);

        // Set the URL endpoint for the map service
        QString endpoint =
            "https://openwms.statkart.no/skwms1/wms.norges_grunnkart";
        QUrl url(endpoint);
        QUrlQuery query;

        // Calculate the corner coordinates of the map based on the UTM
        // coordinate and size
        MapManager::CalculateCornerCoordinates(coord, size);

        // Add query parameters
        query.addQueryItem("service", "WMS");
        query.addQueryItem("version", "1.3.0");
        query.addQueryItem("request", "GetMap");
        query.addQueryItem("layers",
                           "hoyde,Arealtyper,fjellskygge,helning,Vann_og_"
                           "vassdrag,Samferdsel,Bygninger");
        query.addQueryItem("styles", "default");
        query.addQueryItem("format", "image/png");
        query.addQueryItem("crs", "EPSG:25833");
        // Set the bounding box query parameter using the map area that was
        // calulated with the CalculateCornerCoordinates function
        query.addQueryItem("bbox", Instance().m_Area);
        // Set the width and height query parameters based on the image
        // resolution
        query.addQueryItem("width",
                           QString::number(Instance().m_ImageResolution));
        query.addQueryItem("height",
                           QString::number(Instance().m_ImageResolution));
        url.setQuery(query);
        QNetworkRequest request(url);

        // Emit a signal to indicate that an image request is being made
        emit Instance().RequestImage();
        QNetworkReply* reply = manager->get(request);

        // Connect a lambda function to the finished signal of the network reply
        // and check if the reply has no error. If it has no error it read the
        // response data from the reply.
        QObject::connect(reply, &QNetworkReply::finished, [=]() {
            if (reply->error() == QNetworkReply::NoError) {
                QByteArray data = reply->readAll();
                Instance().m_Data = data;
                // Emit a signal to indicate that the image has been received
                emit Instance().GotImage();
            }
        });
    };

    // CalculateCornerCoordinates calculates the corner coordinate of the map,
    // ensuring that the origin is centered in the middle and the sides of the
    // map are equal in length to the specified size. This is added to a QString
    // variable, which will be used in the HTTP request within the "getMap"
    // function.
    void
    MapManager::CalculateCornerCoordinates(Core::UTMCoordinate coord, int size)
    {
        double minX = coord.Easting - (size / 2);
        double minY = coord.Northing - (size / 2);
        double maxX = coord.Easting + (size / 2);
        double maxY = coord.Northing + (size / 2);

        const QStringList wmsRequestCoordsList{
            QString::number(minX),
            QString::number(long(minY)),
            QString::number(maxX),
            QString::number(long(maxY)),
        };
        Instance().m_Area = wmsRequestCoordsList.join(",");
    };

} // namespace MapManagement
```

12.106 src/routemaker/routemaker.cpp File Reference

```
#include "routemaker/routemaker.h"
#include "coordinate_converter/coordinate_converter.h"
#include <algorithm>
#include <array>
#include <cassert>
#include <cmath>
```

Namespaces

- namespace Routemaker

Macros

- #define DRONE_FLIGHT_HEIGHT 175

12.106.1 Macro Definition Documentation

12.106.1.1 DRONE_FLIGHT_HEIGHT

```
#define DRONE_FLIGHT_HEIGHT 175
```

Definition at line 11 of file routemaker.cpp.

12.107 routemaker.cpp

Go to the documentation of this file.

```
00001 #include "routemaker/routemaker.h"
00002
00003 #include "coordinate_converter/coordinate_converter.h"
00004
00005 #include <algorithm>
00006 #include <array>
00007 #include <cassert>
00008 #include <cmath>
00009
00010 // Temporary: When 3D, drone height should vary
00011 #define DRONE_FLIGHT_HEIGHT 175
00012
00013 namespace Routemaker
00014 {
00015     Routemaker::Routemaker(const Core::UTMCoordinate& origin, int size)
00016         : m_MapWidth(size),
00017           m_HeightMap(std::make_unique<HeightManagement::HeightManager>())
00018     {
00019         UpdateOrigin(origin, size);
00020     }
00021
00022     // Relational data that forms paths need to be reset before running A*
00023     // again. This method serves as a simple way to make sure all of these
00024     // values are reset.
00025     void
00026     Routemaker::ResetNodes()
```

```
    {
        for (uint32_t y{}; y < m_RoutemakerWidth; ++y) {
            for (uint32_t x{}; x < m_RoutemakerWidth; ++x) {
                NodePtr node{ GetNode(x, y) };
                node->Parent = std::weak_ptr<Node<Cell2D»();
                node->GlobalGoal = std::numeric_limits<double>::infinity();
                node->LocalGoal = std::numeric_limits<double>::infinity();
                node->Visited = false;
            }
        }
    }

    // In order to improve efficiency, the resolution of routemaker is adjusted
    // based on the size of the scenario. If you are working on a 2km scale, you
    // probably don't need 1 meter fidelity. This should maybe be adjustable as
    // part of the scenario settings in the future.
    void
    Routemaker::UpdateResolution()
    {
        if (m_MapWidth < 250) {
            m_RoutemakerRes = 1;
        } else if (m_MapWidth < 500) {
            m_RoutemakerRes = 2;
        } else if (m_MapWidth < 1000) {
            m_RoutemakerRes = 3;
        } else if (m_MapWidth < 2000) {
            m_RoutemakerRes = 4;
        } else {
            m_RoutemakerRes = 5;
        }

        m_RoutemakerWidth = m_MapWidth / m_RoutemakerRes;
    }

    // Whenever the scenario's origin or size is updated, we need to create a
    // new graph with the proper height data
    void
    Routemaker::UpdateOrigin(Core::UTMCoordinate utmOrigin, int size)
    {
        m_HeightMap->UpdateOrigin(utmOrigin, size);
        m_MapWidth = size;
        UpdateResolution();
        m_Nodes = std::vector<NodePtr>(m_RoutemakerWidth * m_RoutemakerWidth);

        for (uint32_t y{ 0 }; y < m_RoutemakerWidth; ++y) {
            for (uint32_t x{ 0 }; x < m_RoutemakerWidth; ++x) {
                uint32_t xRel{ x * m_RoutemakerRes };
                uint32_t yRel{ y * m_RoutemakerRes };
                float height{ 0.0f };
                bool occupied{ false };
                for (int j{ 0 }; j < m_RoutemakerRes; ++j) {
                    for (int i{ 0 }; i < m_RoutemakerRes; ++i) {
                        float heightCandidate;
                        if (m_HeightMap->GetHeight(static_cast<int>(xRel) + i,
                                                   static_cast<int>(yRel) + j,
                                                   heightCandidate)) {
                            height = std::max(height, heightCandidate);
                        }
                        // Set occupied to true if any of the heights are larger
                        // tha DRONE_FLIGHT_HEIGHT
                        occupied = (occupied || (height > DRONE_FLIGHT_HEIGHT));
                    }
                }
                Node<Cell2D> node{};
                node.Data = { x, y, occupied };
                m_Nodes[x + y * m_RoutemakerWidth] =
                    std::make_shared<Node<Cell2D»(node);
            }
        }
    }

    // Not a pretty method, but does the job. Currently, the routemaker is
    // considering a 2D space, keeping the drones at the same altitude. In the
    // future, we need to consider 3D, meaning the GetNeighbors method needs to
    // consider neighbors above and below the current node as well.
    std::vector<Routemaker::NodePtr>
    Routemaker::GetNeighbors(NodePtr node)
    {
        std::vector<NodePtr> neighbors;

        uint32_t x{ node->Data.X };
        uint32_t y{ node->Data.Y };

        if (x > 0) {
            NodePtr neighbor{ GetNode(x - 1, y) };
            if (!neighbor->Data.Occupied) {
                neighbors.push_back(neighbor);
```

```
        }
    }
    if (x < m_RoutemakerWidth - 1) {
        NodePtr neighbor{ GetNode(x + 1, y) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }
    if (y > 0) {
        NodePtr neighbor{ GetNode(x, y - 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    if (y < m_RoutemakerWidth - 1) {
        NodePtr neighbor{ GetNode(x, y + 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    if ((x > 0) && (y > 0)) {
        NodePtr neighbor{ GetNode(x - 1, y - 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    if ((x < (m_RoutemakerWidth - 1)) && (y > 0)) {
        NodePtr neighbor{ GetNode(x + 1, y - 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    if ((x < (m_RoutemakerWidth - 1)) && (y < (m_RoutemakerWidth - 1))) {
        NodePtr neighbor{ GetNode(x + 1, y + 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    if ((x > 0) && (y < (m_RoutemakerWidth - 1))) {
        NodePtr neighbor{ GetNode(x - 1, y + 1) };
        if (!neighbor->Data.Occupied) {
            neighbors.push_back(neighbor);
        }
    }

    return neighbors;
}

double
Routemaker::GetCost(NodePtr a, NodePtr b)
{
    double x1{ static_cast<double>(a->Data.X) * m_RoutemakerRes };
    double y1{ static_cast<double>(a->Data.Y) * m_RoutemakerRes };
    double x2{ static_cast<double>(b->Data.X) * m_RoutemakerRes };
    double y2{ static_cast<double>(b->Data.Y) * m_RoutemakerRes };

    // We are using a standard cartesian grid, with each cell being one
    // node. As such, Euclidean distance is a nice measure of cost.
    return std::sqrt(std::pow(x2 - x1, 2) + std::pow(y2 - y1, 2));
}

// Bresenham's algorithm is a fine starting point for detecting line of
// sight. However, for better accuracy and more scalability for 3D,
// Ray-casting should probably be considered in the future.
bool
Routemaker::HasLineOfSight(NodePtr a, NodePtr b)
{
    // Assume we have a line of sight
    bool hasLineOfSight{ true };

    std::list<NodePtr> nodes{ BresenhamLine(a, b) };
    std::for_each(nodes.begin(), nodes.end(),
                  [&hasLineOfSight](const NodePtr& n) {
                      if (n->Data.Occupied) {
                          // If any nodes are occupied, there is no line of
                          // sight.
                          hasLineOfSight = false;
                      }
                  });

    return hasLineOfSight;
}
```

```
std::list<Routemaker::NodePtr>
Routemaker::BresenhamLine(const NodePtr& a, const NodePtr& b) const
{
    std::list<NodePtr> list;

    auto x1{ static_cast<int32_t>(a->Data.X) };
    auto y1{ static_cast<int32_t>(a->Data.Y) };
    auto x2{ static_cast<int32_t>(b->Data.X) };
    auto y2{ static_cast<int32_t>(b->Data.Y) };

    bool isSteep{ std::abs(y2 - y1) > std::abs(x2 - x1) };
    if (isSteep) {
        std::swap(x1, y1);
        std::swap(x2, y2);
    }
    if (x1 > x2) {
        std::swap(x1, x2);
        std::swap(y1, y2);
    }

    int32_t deltaX{ x2 - x1 };
    int32_t deltaY{ std::abs(y2 - y1) };
    int32_t error{}, yStep, y{ y1 };

    if (y1 < y2) {
        yStep = 1;
    } else {
        yStep = -1;
    }

    for (int32_t x{ x1 }; x <= x2; ++x) {
        if (isSteep) {
            list.push_back(GetNode(y, x));
        } else {
            list.push_back(GetNode(x, y));
        }

        error += deltaY;
        if (2 * error >= deltaX) {
            y += yStep;
            error -= deltaX;
        }
    }

    return list;
}

Routemaker::NodePtr
Routemaker::GetNode(uint32_t x, uint32_t y) const
{
    uint32_t index{ x + y * m_RoutemakerWidth };
    return m_Nodes[index];
}

std::vector<Core::CartesianCoordinate>
Routemaker::MakeRoute(const Core::Keyframe& a, const Core::Keyframe& b)
{
    // Scenario class should have sorted the keyframes already, but just to
    // make sure:
    assert(b.TimeStamp > a.TimeStamp && a.AgentId == b.AgentId);

    // Keyframes store positions in a symmetric space. Let's make them
    // symmetric to fit our grid.
    Core::CartesianCoordinate asymmetricAPosition{
        CoordinateConverter::CoordConv::SymmetricToAsymmetric(a.Position)
    };
    Core::CartesianCoordinate asymmetricBPosition{
        CoordinateConverter::CoordConv::SymmetricToAsymmetric(b.Position)
    };

    // Let's also divide by our resolution to find the cells each position
    // fits in.
    asymmetricAPosition.X /= m_RoutemakerRes;
    asymmetricAPosition.Y /= m_RoutemakerRes;
    asymmetricAPosition.Z /= m_RoutemakerRes;

    asymmetricBPosition.X /= m_RoutemakerRes;
    asymmetricBPosition.Y /= m_RoutemakerRes;
    asymmetricBPosition.Z /= m_RoutemakerRes;

    NodePtr start{ GetNode(static_cast<uint32_t>(asymmetricAPosition.X),
                           static_cast<uint32_t>(asymmetricAPosition.Y)) };
    NodePtr goal{ GetNode(static_cast<uint32_t>(asymmetricBPosition.X),
                          static_cast<uint32_t>(asymmetricBPosition.Y)) };

    SolveAStar(start, goal); // We find the path
```

```
        PostSmooth(start, goal); // We make it smooth(er)

        // After the path has been generated, we start at the goal, and
        // recursively travel to the start, collecting all positions that make
        // up the route
        std::vector<Core::CartesianCoordinate> route;
        NodePtr current{ goal };
        NodePtr parent{ goal->Parent.lock() };
        while (current != start && parent != nullptr) {
            // We scale up by resolution again
            Core::CartesianCoordinate position{
                (float)current->Data.X * m_RoutemakerRes,
                (float)current->Data.Y * m_RoutemakerRes, DRONE_FLIGHT_HEIGHT
            };

            // Scenario likes symmetric positions, so let's transform back
            Core::CartesianCoordinate positionSymmetric{
                CoordinateConverter::CoordConv::AsymmetricToSymmetric(position)
            };
            route.push_back(positionSymmetric);
            current = parent;
            parent = current->Parent.lock();
        }
        Core::CartesianCoordinate positionSymmetric{
            CoordinateConverter::CoordConv::AsymmetricToSymmetric(
                { (float)start->Data.X * m_RoutemakerRes,
                  (float)start->Data.Y * m_RoutemakerRes, DRONE_FLIGHT_HEIGHT })
        };
        route.push_back(positionSymmetric);

        // Finally, let's reverse the path, so it goes from start to goal,
        // rather than from goal to start.
        std::reverse(route.begin(), route.end());

        return route;
    }

} // namespace Routemaker
```

Index

Appendix P

Medforfattererklæring

Universitetet
i Sørøst-Norge

Medforfattererklæring - bacheloroppgave

Dette skjemaet skal fylles ut og signeres av alle studentene i prosjektgruppen. Ferdig utfylt og signert skjema skal ligge som et vedlegg i rapporten.

Tittel på oppgaven	Hivemind
Veileder fra USN	Dag Andreas Hals Samuelsen

Beskriv hva hver student i prosjektgruppen har bidratt med i bacheloroppgaven. *Eksempelvis i forhold til problemformulering, litteratursøk, planlegging av forsøk/valg av metoder, datainnsamling/bygging av prototype, analyse/tolking av data/uttesting, skriving osv.* *Husk at alle studentene er ansvarlige for helheten av den innleverte oppgaven.*
Aurora Moholth har bidratt med: *Fellesansvar:* Problemformulering, valg av metoder, utvikle produkt, skrive rapport, møteleder-rolle, møtereferater *Ansvarsområder:* Arkitektur, Kompetanseflyt, Team building
Harald Moholth har bidratt med: *Fellesansvar:* Problemformulering, valg av metoder, utvikle produkt, skrive rapport, møteleder-rolle, møtereferater *Ansvarsområder:* Krav, testing
Hilde Marie Moholth har bidratt med: *Fellesansvar:* Problemformulering, valg av metoder, utvikle produkt, skrive rapport, møteleder-rolle, møtereferater *Ansvarsområder:* Dokumentasjon, informasjonsflyt, sosiale medier
Nils Herman Lien Håre har bidratt med: *Fellesansvar:* Problemformulering, valg av metoder, utvikle produkt, skrive rapport, møteleder-rolle, møtereferater *Ansvarsområder:* Dokumentmaler, Risikoanalyse
Ruben Sørensen har bidratt med: *Fellesansvar:* Problemformulering, valg av metoder, utvikle produkt, skrive rapport, møteleder-rolle, møtereferater *Ansvarsområder*: Versjonskontroll, programmeringsansvarlig

Dato	**Signatur**
21/05-23	Harald Moholth
21/05-23	Ruben Sørensen
21/05/23	Hilde M. Moholth
21/5	[illegible]
21/05-23	Aurora Moholth