NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

IT2901 - INFORMATICS PROJECT II, SPRING 2023

ID: SintefDigital_boardGame

Report: Sintef Digital Board Game

Martin Valderhaug Larsen Mathias Gunnarshaug
Thomas William Johansen Tim Matras Torbjørn Stakvik Runar Johnsen

September 5, 2023

Contents

1 Introduction

1.1 The Team and Stakeholders

The team is made up of six students who had diverse backgrounds and knowledge before the project started. Some group members have previously worked with creating games and therefore have more skills in developing games and related software. Others have more knowledge about server work, and so on. At the beginning of the project, each team member's motivations and background were discussed so that everyone could display their strengths and areas that required more work, which again would lead to gaining more knowledge. The result of the discussion was a competency matrix, which is available in section 3.2.3.

List of stakeholders that have been identified:

- Sintef
- End users
- Future developers
- Supervisor
- The Team
- Examiner

The most important stakeholder in the project is the customer, Sintef. They have developed the physical board game, which currently is a workshop version, but plans for an extended version are under development. The game versions will also be easier to develop once the board game becomes digital. The workshop version is meant to be used during workshops with other collaborators in a project where orchestrating traffic is the main goal. It is also important that the workshop version does not take too much time to play. The board game illustrates how important it is for different traffic situations to be managed together. The group is responsible for making a digital version of the board game for Sintef so that people can play the game without being there physically. Once the project is over, the developers at Sintef must be able to continue working on the digital version of the game by, for example adding new features and fixing possible problems.

In addition, it is important that the product is usable for the end users. Their needs should therefore be kept in mind when developing the digital version of the game so that it is also usable for them.

During development, future developers must be kept in mind such that they can work on the project after the bachelor project is over. This is important as the game is not fully set in stone and will likely change after implementation.

Writing the report gradually during development is important to be coherent and accurate. This is important because then the examiner can make an accurate assessment.

1.2 Motivation

The increasing popularity of autonomous cars has created the opportunity to establish a system of automatic communication between vehicles. This concept, known as remote traffic management, is a highly complex topic, as explained by Sintef. To make the topic more accessible to a broader audience, Sintef has developed a board game as an introduction. The game

is designed to be user-friendly and provide a basic understanding of the concepts involved in remote traffic management. Sintef has now requested that a digital version of the board game be created to expand its reach and accessibility.

This project is important for several reasons. Firstly, it allows a broader audience to learn about and understand the concept of remote traffic management. Using a board game format makes the topic more approachable and user-friendly, making it accessible to people who may not have a technical background in the field. Secondly, as autonomous cars become more prevalent, the need for effective remote traffic management systems becomes increasingly important. The board game and its digital version can help educate people about the potential benefits of such systems, such as increased road efficiency. They can also help generate interest and support for further research and development. Lastly, the digital version of the board game can also be used as an educational tool for students in related fields such as transportation engineering, computer science, and urban planning. It can provide a hands-on, interactive way for them to learn about the complex topic of remote traffic management and its potential impact on the future of transportation.

1.3 Product overview

As mentioned previously, the product is a digital board game trying to introduce the topic of traffic management to a broader audience. With most people not understanding how this concept works, this game aims to introduce them to it. The board game does this by presenting traffic management in a fun and engaging way.

There are two versions of the physical board game: the workshop and the extended versions. As the name implies, the extended version extends the workshop version, offering more gameplay features, such as players having multiple objectives. The product owner expressed that the extended version could also be implemented if there was extra time. However, it was important to focus on finishing the workshop version first.

The workshop version is played with up to 7 players, where one is the traffic orchestrator, and the others are road users. The state of the game is represented by a board containing a map of the city where traffic will be managed. The map contains nodes describing stops and edges describing the roads that connect these nodes/stops.

Each road belongs to a zone and is coloured accordingly. Each zone has a state card describing the traffic situation in that zone and what measures the traffic orchestrator has taken to manage the traffic in that zone. Each road user has a unique colour and is represented on the map by a car. The road users must pick up and drop off an object of their colour. The game is played by trying to pick up and drop off their deliveries with limited moves. Normally, one move along the edge is counted as one move, except for when there is a lot of traffic, then it takes more moves, or when some measures are taken by the orchestrator that could give more moves to the player or restrict them from going somewhere.

This game currently exists in a physical workshop version, and, as mentioned previously, the project will be porting that physical version over to a digital board game. Since the only foundation to work from is the physical workshop version, the entire game and server will be made from scratch.

As mentioned in the previous paragraph, this project aims to develop the workshop version, a running prototype. After development, adding elements from the extended version should be possible.

1.3.1 Extended version

As described in the previous section, there will also be an extended version of the board game. Finalising the workshop version of the board game has been the top priority. After that, there was a heavy focus on making it easy for future developers to continue an extended version. This is because it is important for the customer that the workshop version can be expanded later – they would like to add the extended version's features easily.

In the extended version of the board game, there will be several new features. The customer has already thought through some of these, whereas others might be included later. This is also why it is important that the code is expandable. At the start of the game, the player should have access to a pot of money to pay tolls on some roads so that the orchestrator can limit traffic in certain districts. Another possibility is that the player can earn money while playing the game, for example, if they finish their task or win. For this to make sense, it should be possible to play several rounds in succession, where also the money could come into play. The customer also wants the players to have multiple tasks simultaneously to see and understand the different traffic situations and relate them to one another. These are just some examples of the features that could be a part of the extended version. Overall the customer has not set any limit on the features of the extended version.

1.4 Objectives and Key Results

Our key features are:

- Show the state of the game clearly that communicate all the necessary information.
- Checking that the actions made by a player are legal.
- Multiplayer functionality.

Implementing the key features in this project utilized Rust on the server side and Unity and C# for the client side. Requirements from the product owners were flexible, but both the development team and the product owner were able to reach a common ground that Unity offers the best tools for implementing the key features. Clean architecture is important, as flexibility and re-usability are important to the product owner. This way, future developers can work on expanding the product with minimal difficulty after the development team has finished working on it.

The initial plan for this project was to develop a digital game based on the physical board game, as described in the Product chapter. The workshop version was successfully implemented, which was the project's goal, but there was no time to include features from the extended version. The development process was slowed due to inexperience with Unity, Rust, and, most importantly, multiplayer game development, as it is a very hard task. In addition to this, not everyone within the group managed to work the minimum work hours. Had it not been for these setbacks, it is not unreasonable that some of the extended version's features also could have been implemented. Then again, the main goal has been achieved.

It should be mentioned that since the physical board game is still not fully defined, it is not possible to make a full and clear list of objectives and requirements since the customer has not yet defined that fully. It takes a lot of time to develop a board game that works well, and since this project aims to digitize the board game, that will be the main focus, not the rules and dynamics of the game. There will therefore be created objectives and requirements that can act as a guide in developing a system that is easy to change so that the customer can keep working on the system afterwards.

1.4.1 Level of ambition

This project's ambition level is a fully working prototype based on the workshop version of the game, with the option to extend the game to the extended version if there is enough time. The purpose of the fully working prototype is to have the option to play the game over the internet to get feedback from people who cannot be present in real life to play the physical version. Considering that this is just a prototype, future developers must be able to easily iterate and expand on the prototype for further functionality and other desired changes. The timeline for the prototype is relatively short and strict, but it opens the possibility of adding extended functionality to the project, as that is what the product owner desires.

1.4.2 User stories

To make developing the project easier, a couple of objectives have been defined for each milestone (can be found in 5.2). Functional requirements (described in 2.2.1) were defined based on these objectives, wishes from the product owner, and game rules. User stories were defined before each milestone and are based on the functional requirements. The following user stories were defined in no particular order:

ID	User story	Priority (low, medium, high)	Relevant requirements and/or goals
U1	As a player, I want to see the current state of the game on my screen so that I understand the current situation	high	Fr23
U2	As a road user, I want to be able to see my player on the game board so that I know my position	high	Fr24
U3	As a player, I want to see which player's turn it is, so that I know whose turn it is	medium	Fr25
U4	As a road user, I want to see where I need to pick up the packages and drop them off so that I know where I need to go	high	Fr26
U5	As a road user, I want to see which nodes I can move to so that I know where I am allowed to go	high	Fr27
U6	As a road user, I want to be able to click on a node to move to it so that I can try to pick up and deliver my package	high	Fr26, Fr28
U7	As an orchestrator, I want to be able to create a lobby so that I can play the game with others	high	Fr30
U8	As an orchestrator, I want to choose situation cards before the game so that we can play out the scenarios in the game	high	Fr2
U9	As a road user, I want to be able to get an assignment card that matches the situation card so that I have an objective in the game	high	Fr3
U10	As a player, I would like the board to be updated with any relevant objects based on the situation and assignment card(s) so that the road users know how it will affect them	medium	Fr4

U11	As an orchestrator, I would like to be able to apply measures to the zones so that I can help the road users with getting to their destinations	high	Fr5
U12	As a player, I would like to be able to see all the available lobbies so that I can see which games I can join	high	Fr45
U13	As a player, I would like to join a lobby so that I can play with other people	high	Fr46
U14	As a player, I would like to leave a lobby so that I can join another one if I joined one as a mistake	medium	Fr47
U15	As a player, I would like to leave a game so I can join another one	medium	Fr47
U16	As an orchestrator, I would like to start the game so that we can play the game together	high	Fr48
U17	As a player, I would like to perform an action when it is my turn so the next player can get their turn	high	Fr50
U18	As an orchestrator, I would like to add restrictions to a district so that I can manage the traffic	high	Fr51
U19	As a player, I would like to move no further than what is within the restrictions that apply to me, so I can follow the restrictions laid out by the orchestrator	high	Fr38 Fr41
U20	As a player, I would like to see how many remaining moves I have so that I can plan which way I should go	high	Fr40
U21	As a player, I would like to undo my actions so that I can undo my mistakes	high	Fr53
U22	As a player, I would like to see how much it costs to move to a node so that I know how the movement affects me	medium	
U23	As an orchestrator, I would like to be able to set up park & ride, so that I can reduce traffic in a given district	medium	Fr18
U24	As an orchestrator, I would like to be able to put up restrictions on specific roads so that I can reduce traffic within a district	medium	
U25	As an orchestrator, I would like to see how my traffic restrictions affect the traffic within the districts so that I can find the best restrictions for a given situation	high	
U26	As a player, I would like to see where my and the other players markers are placed	high	Fr24
U27	As a player, I would like to see where my and the other players packages and destinations are	high	Fr25
U28	As a player, when starting the game I would like to see the ”enter username” screen	high	Fr34
U29	As a player, I would like to go to the lobby screen after creating my username	high	Fr35
U30	As a player, I would like to see the restrictions placed on the different zones	high	Fr36
U31	As a user, I would like other players that are not active in the game to be removed after some time	high	Fr55

2 The product

2.1 Prestudy

The starting point of our project is described in 1.3.

2.1.1 Purpose

The point of this prestudy is to help guide the development process by comparing a selection of technologies and libraries, addressing their pros and cons, and eventually choosing the most optimal for our project. This will make it easier for us to define the project scope based on our choices and clarify what resources and knowledge we need to complete it. The prestudy also allows us to think and reflect on the choices before making them.

2.1.2 Tech-stack

2.1.2.1 Evaluation criteria With the many people involved in our project, it becomes necessary to have clear evaluation criteria when evaluating different alternatives. One could argue that evaluation criteria are unnecessary because we must discuss different options against each other given all their strengths and weaknesses. On the other hand, we argue that it has a positive effect by making things clearer.

The criteria will make decisions easier for the group by first limiting potential alternatives in the first place. Selection will also be simplified by evaluating each alternative against our criteria. As for the product owner, having set criteria will clarify why we made the decision. With the reasoning cleared, we can move on to our criteria.

- The main criterion is that the alternative, at a minimum, must cover the demands of the product owner.
- The alternative should also have some flexibility. Both to cope with potential changes the stakeholders make and for future development by other teams.
- The team's experience and ease of learning should weigh in positively when evaluating an alternative.

The criteria will not cover every possible scenario, but as stated earlier, in most cases can simplify the process of making a decision.

2.1.2.2 Game Engine To ensure that we meet the demands of the product owner, have some flexibility and take into account our group's experience and ease of learning, we have evaluated various options for the game development platform.

One such alternative was Tabletop Simulator (also defined in 5.1). One of the main problems with this alternative is that it requires the players and developers to pay for the game if they want to use it to develop and/or play games. In addition, using the Tabletop Simulator ecosystem to develop the game will make it dependent on the ecosystem, making it less flexible. However, making the game in Tabletop Simulator might be easier than other solutions as it comes bundled with its board game creator and has a relatively simple Graphical User Interface (GUI). However, the game/platform costs money, and no one in our group has any experience developing games in Tabletop Simulator. As a result, we decided that Tabletop Simulator was not a suitable option for this project because we also wanted to give the product owner more control of the project after delivery.

Unreal Engine (also defined in 5.1) is another free option if the game makes no income. It has a lot of different tools and resources to facilitate game development, with a built-in user interface system and physics engine. It also supports 2D and 3D game development but is more suited to the latter. Despite its flexibility, it is known for its steep learning curve, which makes it less suitable for our team, as only one person has experience with it. To make the game development process more practical and efficient, we would like to use a game engine that is easier to use and that some of our members are already familiar with, like Unity.

Unity is also a powerful and versatile engine and is free in the same way as Unreal Engine. Unity offers many features and capabilities and is much more user-friendly and easy to learn than the other options. Unity also has many tools and systems to help with the game development process, like Unreal Engine and supports both 2D and 3D game development. This also gives Unity the flexibility we want. In addition, three of our group members have experience with it.

After comparing the game development platforms, we decided that Unity best fits our project. Unity allows us to easily develop the project with the help of its extensive features, tools and systems. It's user-friendly interface and easy-to-learn framework make it readily accessible for our team, even for those with limited to no experience with game development. Unity's flexibility also gives the product owner more control of the project after delivery.

2.1.2.3 Networking solution We initially considered using Mirror (also defined in 5.1), a high-level open-source networking library designed for Unity. However, it is more desirable from the product owner's side that the network is completely detached from Unity and that it is possible to run a dedicated server to avoid any problems regarding firewalls. Due to this preference, we decided to drop Mirror.

Instead, we will implement online multiplayer by having the game communicate with a dedicated server through REST API (also defined in 5.1). This will restrict us to implementing the network solution in a client-server architecture rather than peer-to-peer. This will also lead to the exclusive use of the Transmission Control Protocol (TCP) since REST is HTTP based. The server will keep track of the game state and update the state based on the player's inputs. The consequence of this change is that developing this dedicated server will now be a bigger part of the project, as the server has to be deployed near the end of the development process.

Initially, this was going to be implemented using the .NET framework built on top of the C# language, but it has since been changed to the Actix Web framework, which runs on top of the Rust language. Actix offers many web tools and an introduction to how these tools are used [5]. The main concern with using the .NET framework was the learning curve and how easy it is to write bad code in C#. The more advanced the code got, the more prone it was to errors, especially with multithreading in the picture. Switching to Rust has eliminated all of these problems and also improved performance by a significant amount. The choice of using Rust instead of the .NET framework was made in agreement with the product owner.

Another gain from switching over to Rust was that the code would be more sustainable. A server that is going to provide a service, such as a backend provides, has to keep running for as long as the game service is needed. With a prolonged hosting service, energy efficiency becomes a concern. Higher efficiency means lower hosting costs, so there is an economic gain in preserving energy. In a research article about energy efficiency across programming languages, Rust ranks much higher than C# in every category. There categories are: *Time*, *Energy*, and *Memory*. The run benchmarks test different combinations of these [2]. As shown by the benchmarks, Rust ranks high in overall efficiency.

The product owner also requested that the game be easy to play. For context, they plan to show the game to others at conventions where they will hold presentations. Unity provides

options for downloading the game as a .zip file, hosting the game on a game hosting service like Steam, or building a web application using WebGL. The group concluded that WebGL would best fit the product owner's needs since people only need to enter a link in a browser to start playing. Mirror is incompatible with WebGL builds, contributing to our networking solution choice.

2.1.2.4 User interface (UI) Unity offers two officially supported solutions for customising a visual user interface - uGUI and UI Toolkit (also defined in 5.1). UI Toolkit is the newer alternative and works similarly to HTML and CSS. It is considered the successor to uGUI, but some features found in uGUI are still not implemented in UI Toolkit.

One of our group members has experience with uGUI and UI Toolkit and considers uGUI the more stable option. According to Unity, uGUI is established and production-proven, while UI Toolkit is in active development. This means we are missing out on regular updates, but since we are making a game for an actual organisation, we value stability more.

In addition, three people in the group are familiar with uGUI compared to only one with UI Toolkit. Since we intend to deliver a high-quality prototype and satisfy the customer, we can dedicate more time if people are already familiar with the technologies. We will, for these reasons, use uGUI as our UI solution.

2.2 Requirements

Our guiding (as mentioned in 1.4) requirements are here.

2.2.1 Functional requirements

ID	Requirement	Priority (low, medium, high)	Game-version: (workshop, extended)
Fr1	It should be possible to choose an orchestrator as well as up to 6 players	medium	workshop
Fr2	The orchestrator should be able to choose a situation card	high	workshop
Fr3	The players should randomly be dealt a situation assessment/objective card that matches the situation card	medium	workshop
Fr4	Based on the situation card and assessment/objective cards, the board should be updated with the relevant objects	medium	workshop
Fr5	The orchestrator should be able to apply measurements to districts	high	workshop
Fr6	The traffic markers should be updated based on the measurements that are currently applied	medium	workshop
Fr7.1	The players should be able to move their pieces	high	workshop
Fr7.2	Only legal moves should be allowed, based on the measures	medium	workshop
Fr8	There should be an option to reset the board for a new round	medium	workshop

Fr9	Electrical vehicle access restrictions should block players without the corresponding vehicle type from entering the given district	high	workshop
Fr10	Dangerous vehicle access restrictions should block players without the corresponding vehicle type from entering the given district	high	workshop
Fr11	Emergency vehicle access restrictions should block players without the corresponding vehicle type from entering the given district	high	workshop
Fr12	Bus and shuttle bus should limit movement to the given markers if the player parks at the bus and shuttle spot	high	workshop
Fr13	Vehicles with high load factor vehicles limited the movement of players without the corresponding marker in the given district	high	workshop
Fr14	Destination vehicle access restrictions should block players without an objective within the district from entering the corresponding district	high	workshop
Fr15	Priorities added to a district should only affect said district with the given amount of bonus moves	high	workshop
Fr16	Speed added to a district should only affect said district with the given amount of speed	high	extended
Fr17	Toll added to a district should only affect said district with the given amount of toll amount	low	extended
Fr18	Park and Ride should only be allowed on the specified roads, allowing the orchestrate to create a path. Any negative effects of traffic get ignored	high	workshop
Fr19	Block network segment should block the placed upon segment	high	extended
Fr20	One-way drive should only be allowed on a given road, and apply the given modifier	high	extended
Fr21	A round is finished when all players have completed their objectives within their given time limit	high	extended
Fr22	Without any measures, a player should be able to move up to 8 spaces	medium	workshop
Fr23	The gameboard is visible	high	workshop
Fr24	The players are visible on screen	high	workshop
Fr25	The package and goal is visible on screen	high	workshop
Fr26	The nodes are highlighted when hovered over	medium	workshop
Fr27	The nodes are clickable	high	workshop
Fr28	The player can move from their current position to another node by clicking on the node they want to go to	high	workshop
Fr29	The players should be able to be distinguishable by a unique ID	high	workshop
Fr30	A new game can be created with a specified name and the player who wants to create it	high	workshop

Fr31	The server should be able to receive player inputs and apply them to the game if the inputs are legal	high	workshop
Fr32	The client should send the correct data structures to the server when a player does an input	high	workshop
Fr33	The client should be able to parse the game state returned by the server correctly and update the screen accordingly.	high	workshop
Fr34	The start menu is visible when starting the game	high	workshop
Fr35	The buttons on the start menu go to the scene/screens they are supposed to go to	high	workshop
Fr36	The district cards are visible on the screen	high	workshop
Fr37	The parts of the district cards that can add restrictions should be pressable	high	workshop
Fr38	Player movement is restricted based on how much the player already has moved	high	workshop
Fr39	The pickup and drop-off points are placed correctly based on the drawn objective cards	high	workshop
Fr40	Players can see how many actions they have left	medium	workshop
Fr41	Players can see which legal nodes they can move to	medium	workshop
Fr42	District cards containing information about the districts should be displayed on the game board screen	medium	workshop
Fr43	The orchestrator should be able to interact with the district cards to apply restrictions	high	workshop
Fr44	Players should be able to see how much money they have	low	extended
Fr45	The server should be able to return all games/lobbies that have not started yet.	high	workshop
Fr46	The server should be able to add a player to a game that has not started yet.	high	workshop
Fr47	The server should be able to receive a request from the player to leave a game or lobby and should handle it correctly.	high	workshop
Fr48	The orchestrator can start the game	high	workshop
Fr49	A player that is not an orchestrator cannot start the game	medium	workshop
Fr50	Only one player can do an action at a time	medium	workshop
Fr51	The orchestrator can add restrictions to a district	high	workshop
Fr52	The players are restricted based on the restrictions	high	workshop
Fr53	The players should have the ability to undo their actions	medium	workshop
Fr54	The server should be able to find all the nodes a player can move to and send it to the player with the game state	high	workshop
Fr55	The server removes players after a certain amount of time	medium	workshop
Fr56	Empty (no players) games should be removed automatically	medium	workshop

Fr57	Road users can choose multiple assignment cards at the beginning of the game	medium	
Fr58	The Orchestrator can see how much money they've gotten from tolls	low	extended
Fr59	The Orchestrator can buy upgrades with their money	low	extended
Fr60	Road users can have multiple assignment cards	low	extended
Fr61	Road users can assign assignment cards to themselves	low	extended
Fr62	Road users get money from completing assignments	low	extended
Fr63	Road users lose money when entering a district with toll	low	extended

2.2.2 Non-functional requirements

ID	Requirement	Priority (low, medium, high)
NFrM1	(Modifiability) Every piece of code should be made after the open-closed principle, meaning that the code should be able to be extended without being modified. This means new features should be added through new code instead of changing existing code.	medium
NFrU1	(Usability) What can be done automatically should be done automatically (placing markers, limiting/ increasing movement etc.)	low
NFrU2	(Usability) The system should show which options are available if they don't get in the way of the game	medium
NFrP1	(Performance) From the time a player ends their turn, there should go no more than 1 second until it is visually updated for the other players	high
NFrA1	(Availability) The client should not crash more than 1 time out of 100 games played	high

2.3 Architecture

2.3.1 Why have an architecture?

Our team is interested in good architecture to get a unified understanding of the project and to have a template everyone can follow when executing the project. Sintef Digital is interested in good architecture to end up with a good product and product that is made to their specifications and caters to their needs. The examiner is interested in good architecture documentation and clear code to give a well-founded evaluation. Our supervisor is also interested in good architecture documentation to give us clear feedback and specific points to improve. Given that this project is based on the workshop version of the game and that the game is meant to be expanded upon after we finish working on it, future developers will also be interested in

a well-documented and well-fitting architecture to understand what the code does and break down its core functions.

2.3.2 Tactics and patterns

Unity Engine utilizes the Entity-Component-System pattern (ECS) (also defined in 5.1), where Systems manage different components attached to Entities (in Unity, these are called Game Objects). ECS is also used in engines such as Unreal and Godot (defined in 5.1) and is generally common in game development.

We used the Object Pooling pattern to improve performance on the client side. In our solution, we return unused Game Objects to a pool to be used again. This prevents performance issues related to the Instantiate method in Unity.

Model-View-Controller (MVC) (also defined in 5.1) is an architectural pattern used in many games and works well with ours. Our view is the client, our server is the model, and our controller is the interface between the two. This pattern allows us to work on the model, view and controller individually or exchange each part for different versions, for instance, when we later have multiple game versions (workshop and extended). This increases cohesion by using the tactic of split modules. We have reduced coupling by the encapsulation tactic, as the client can only contact the server through the game controller and vice versa. Another tactic used is the support system initiative using the maintain system model[1], as the system shows which nodes the user can move their piece to by highlighting the appropriate nodes.

The Client-Server pattern is used when the client sends an input to the server, then the server responds with the updated game state, whilst all the other players constantly send requests to the server to get the updated game state.

2.3.3 The architecture

We can explain the architecture using the 4 + 1 architecture view model. This model provides five different perspectives on software architecture. It is like this in our case:

- **Logical view:** The game architecture provides the Network namespace for communication with the server and a game assembly for representing and interacting with the active game state. On the server side of things, there is a game controller responsible for updating the state of the game based on input received from the REST API ”server”, marked as ”src/main.rs” on the figure 1. This will be done by validating that the actions from the players are valid and that the following state of the game is also valid using the Rule Controller. Creating and joining games and other input is also handled by most of the same components/classes. Please see figure 1 for a more detailed view of how the system is connected.

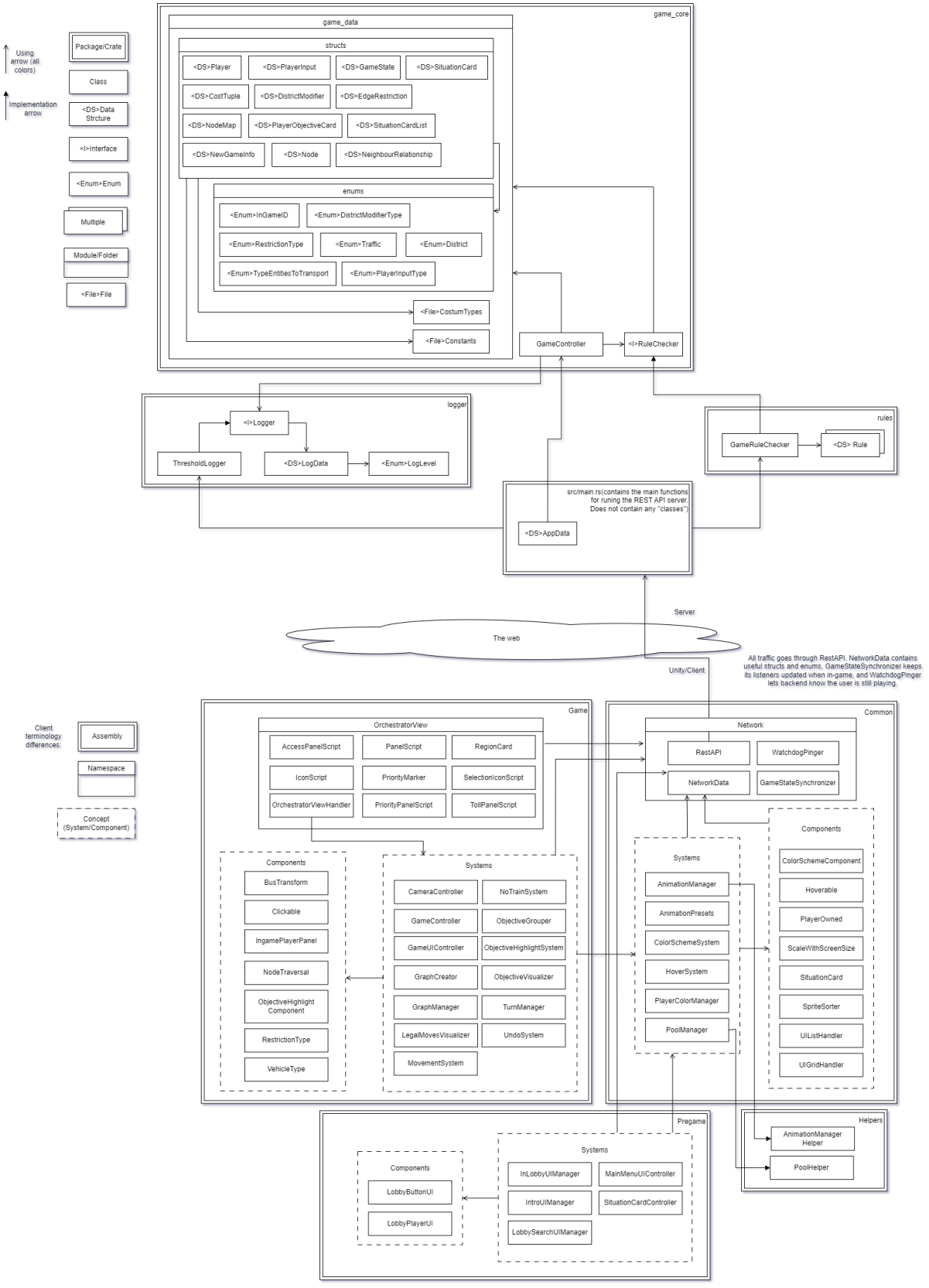


Figure 1: Package and component diagram. This figure is simplified so as not to include hundreds of arrows. It is worth noting that Rust does not have classes, only structs with implementations; therefore, there are no classes on the server side. The frontend's Editor assembly is also hidden for simplicity.

- **Development view:** See figure 2. Blocks indicate assemblies for the client and crates for the server. Arrows between the assemblies/crates indicate dependency from A to B (e.g. Game scripts can access and use Common scripts but not the other way around). On the client side, there are 5 assemblies: Editor, Helpers, Game, Pregame and Common. Editor contains editor scripts (which make it possible to add buttons and more to Unity's inspector). Helpers implement interfaces in other assemblies by dependency injection; hence it is high up in the dependency hierarchy. Game contains all scripts that are only used when playing the game, Pregame are all the scripts used exclusively for everything except the game, and common has scripts that are needed both in-game and outside of a game.

  On the server side: src/main is responsible for hosting the Rest-API, receiving requests and sending back responses. It functions as an interface between the other crates and the client. logger ensures that logging works properly whether deployed on a server or locally. The rules crate checks everything needed to determine if a user action should be accepted or denied. The rules it checks ranges between everything from checking moves left to checking if a player can change to a specified role. All logic and data structures not related to rules are defined in game_core.

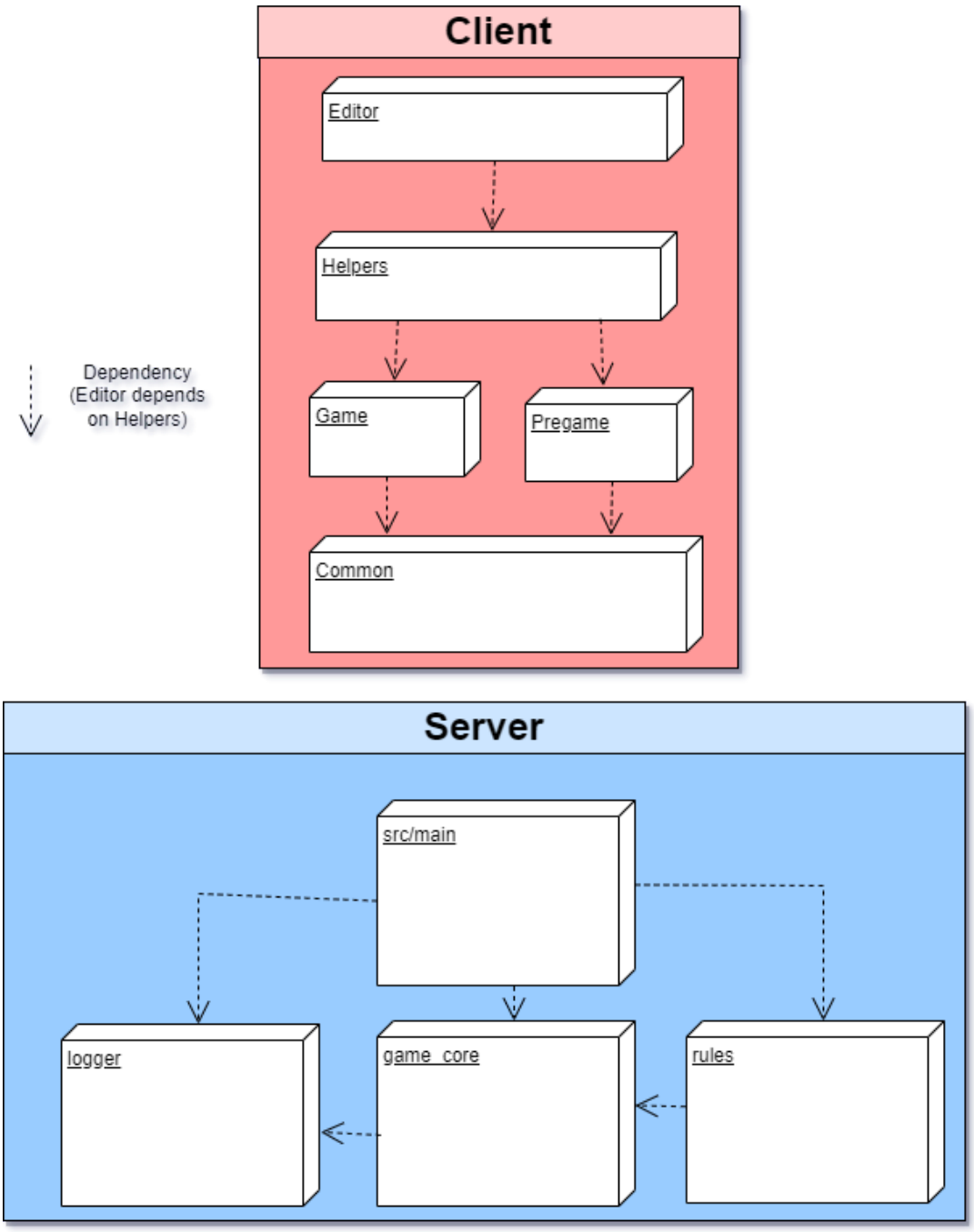


Figure 2: The assembly diagram for our system

- **Physical view:** We want to run the server code on a server separate from the client since our architecture allows us to do that through abstraction. The server waits for a player to do an action, and when they do, they send a request to the server and the server responds based on the action's validity. The other players are constantly asking the server for the game state in the hopes that it's updated. See figure 3.

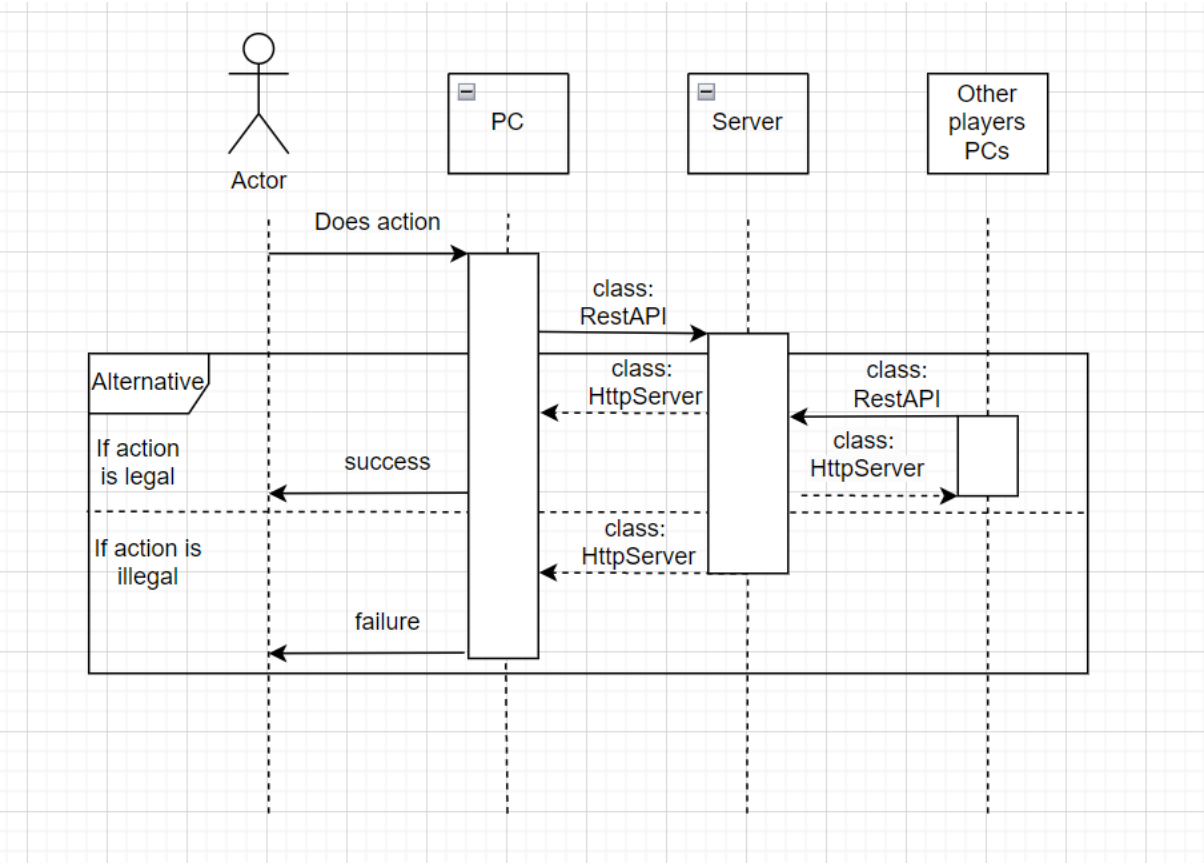


Figure 3: The physical view of our system, using a sequence diagram. The boxes indicate different computers/dedicated servers. The lines indicate direct communication through HTTP requests.

- **Process view:** The user starts out at the username screen. Here the player can input a desired name that will be used to join a lobby or create a new game. The player's username does not need to be unique because every player needs to have a unique id generated by the server. So before joining a lobby or creating a new game, the player will also need to get a unique id from the server, this is done in the background through the RestAPI class, so the player does not need to do anything.

  The user then sees the "Create game" button and presses it. Doing this fills out a NewGameInfo struct that is passed on to the server, and if everything is alright, a new game state will be passed back to the player. If the player instead chooses to join a lobby, then the RestAPI class will send a request to the server about joining the said game and will return back to a game state if everything went alright.

  If the player is the orchestrator of the game, then the player can choose which situation card they want to use in the game. Once the player has selected a situation card, then an update will be sent to the server on what the desired situation card is. All the other players in the lobby are waiting for the orchestrator to start the game. If a situation card is selected, the orchestrator can start the game by pressing the "Start game" button. Doing this will send an input to the server telling it to start the game. If a player who is not an orchestrator tries to start a game, the server won't let them. In the lobby and in the game, the player has the option to disconnect from it and be sent back to the lobby list screen. Disconnecting from a lobby will send a request to the server, and the server will remove

the player from the game. Whilst in the lobby, the player also has the option to change to another role, a player or an orchestrator, based on what they are currently. Pressing the button to change the role will send an input to the server that will change the player's role and return a new game state with the updated role. This back-and-forth between the server and the client/player happens for nearly any input the player does.

When the game has started, everyone waits for the orchestrator to play and end their turn, after which players take turns one after another. When the last player has ended their turn, the situation is finished, and everyone is returned to the lobby. At this point, all users can once again join and leave the lobby as they please, and the orchestrator can start the game (after selecting a situation card).

See figure 4 for more context.

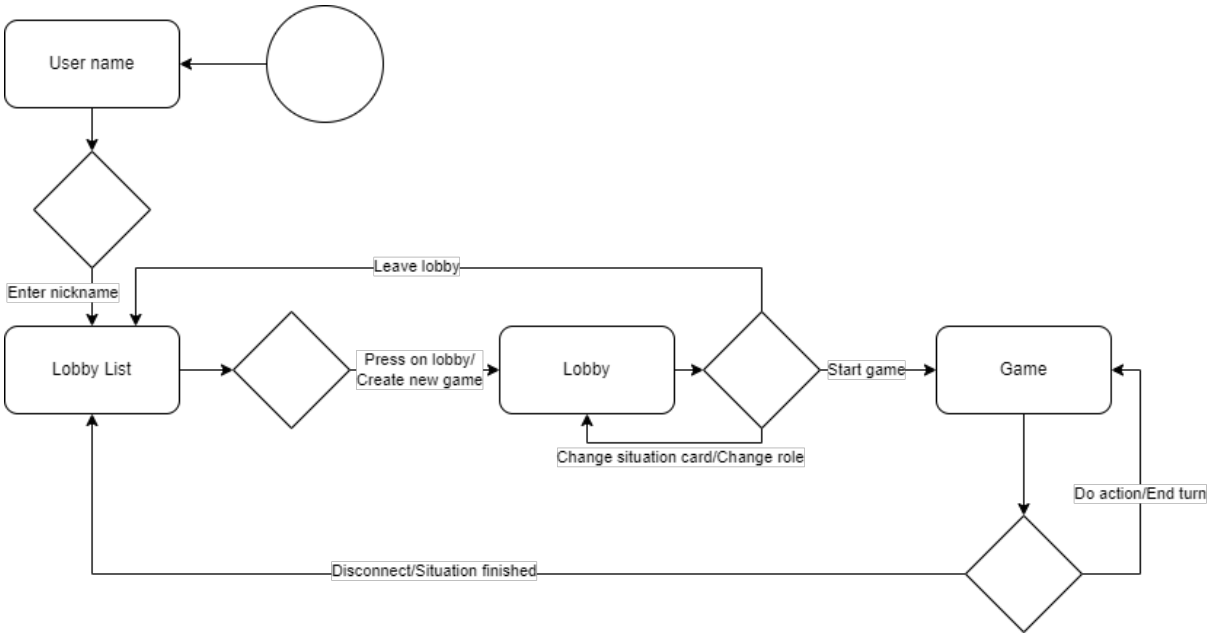


Figure 4: Activity diagram

- **Scenario view:** The Scenario view shows which actions a user can do within the application and gives an overview of the use cases of the system. The Square shows what happens within the system. The bubbles are connected to the actors capable of performing those actions. If an ”extend” arrow points from a bubble connected to an actor towards a standalone bubble, it means that that action will always happen when the action that is in the bubble that is connected to the player happens. If the ”extend” arrow is pointing in the opposite direction, it means that the action in the standalone bubble might happen when the action in the bubble connected to the player happens. See figure 5.

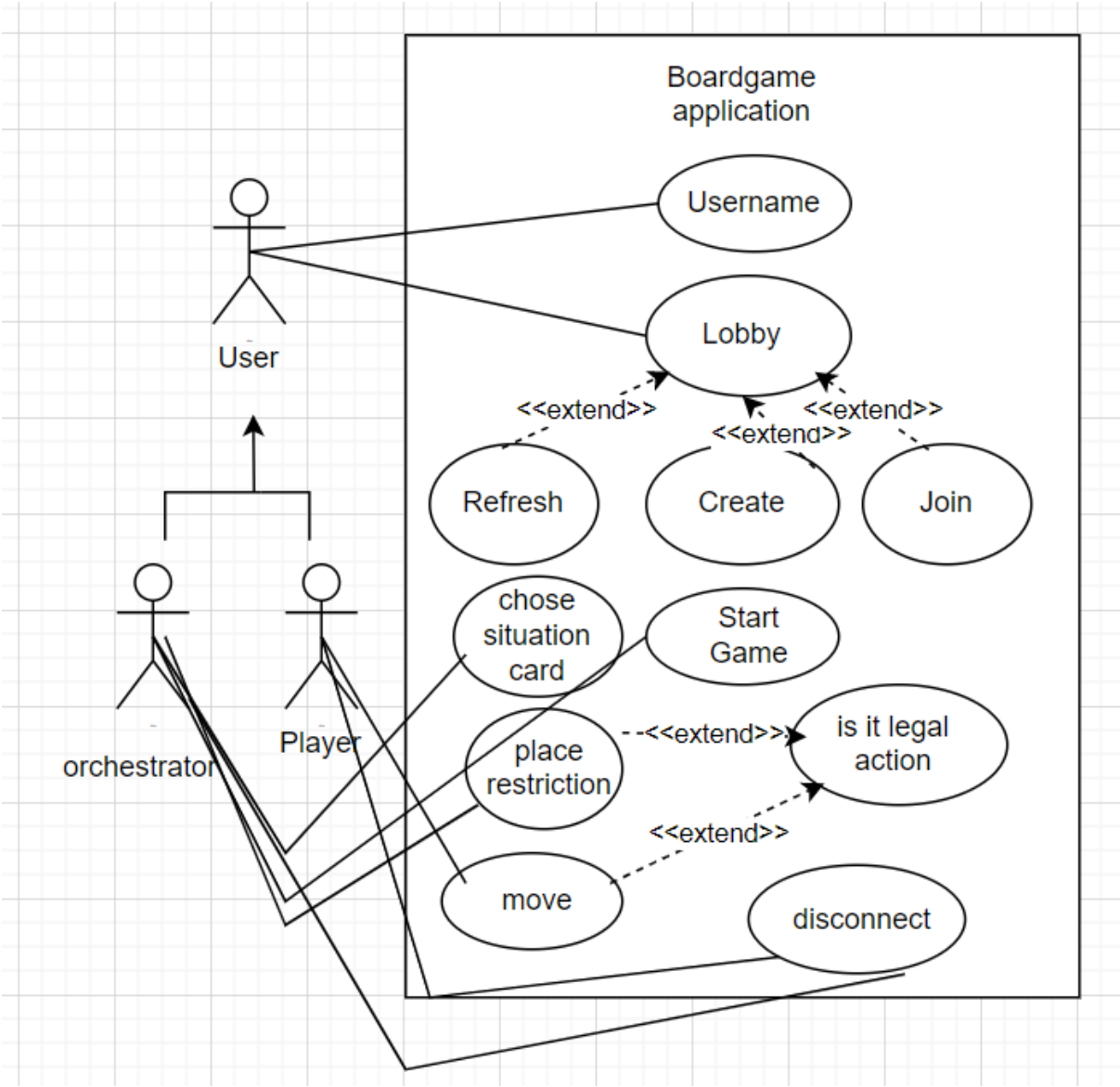


Figure 5: The Scenario view of our system, using a use case diagram

2.3.4 Drivers

The main driver behind our architecture is the project requirements, which the product owner defines. The main requirements are that the game should be playable on a computer and that the game is multiplayer. In our case, this means we have a client which presents the game and a server that handles all the logic for that game.

Other important drivers behind our architecture are high flexibility and maintainability. This means we needed to design a software architecture that allows future developers and us to easily modify the code and architecture, should the need arise. To achieve this, we separated the core logic of the game from any details like a game engine or framework. As a result, switching to another game engine is easy because the core logic of the system is independent of any framework or game engine. This is because as long as the new code using the new game engine implements the interfaces defined by the server, the game should work as expected.

Another important driver in our architecture is testability. Our system must be easy to

test to ensure it works as expected. To achieve this, we have separated the hard-to-test code, like GUI and Input from Unity, from the core code in the system. The result is that all the code that is hard to test is kept to a minimum, decreasing the chances of bugs and unexpected behaviours, whilst all the code that is easy to test is testable through tests like unit tests, which should catch most bugs and unexpected behaviours.

2.4 Implementation

The implementation consists of both a front- and backend. The way both ends communicate with each other is through a REST API. The server keeps track of the entire game state, and each client sends their input to the server, which the server handles and uses to update the game state the input is for, and that the other players can get.

The backend was initially written in the .NET 6 framework, which uses the C# language. However, we encountered some very annoying issues, so we decided to look for other options and made the decision to move the server over to Rust. The server would instead be implemented using the Actix web framework for Rust (defined in 5.1).

The frontend is implemented in Unity which uses the C# programming language. The project builds to WebGL and is thus available through the browser, currently by accessing a website provided by the product owner: `http://boardgame.opendatalab.no/`. This leads to the following screen: 6

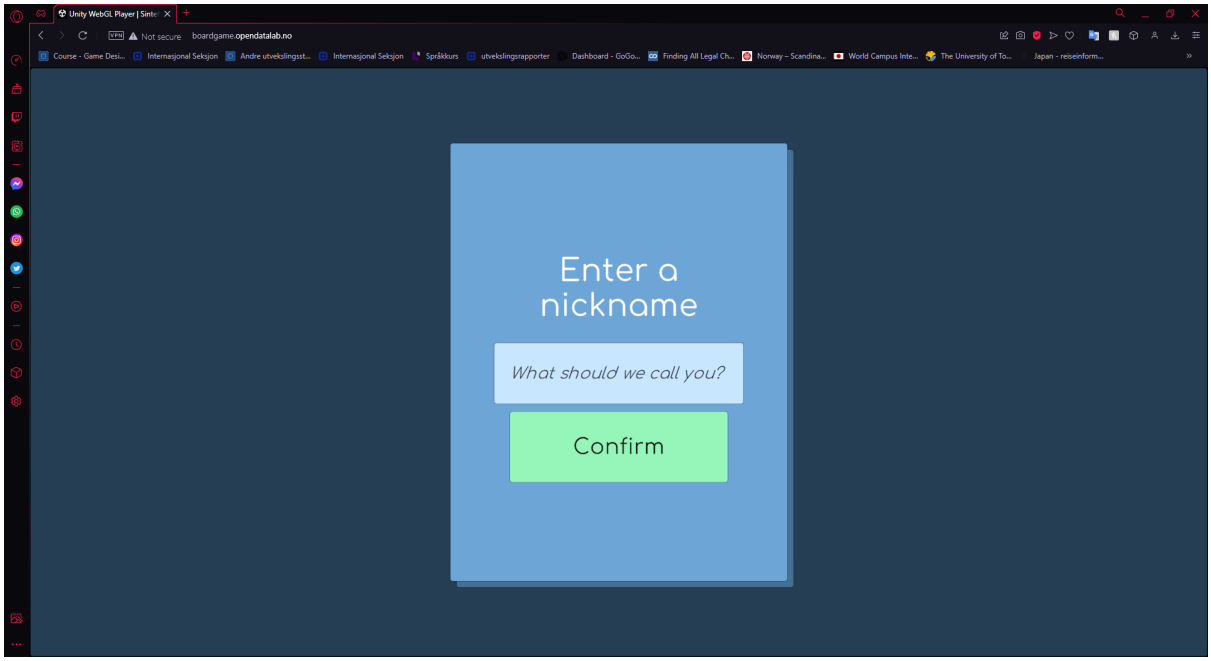


Figure 6: What the user first sees when entering the website

The game window can be expanded to full screen by clicking the blue button. Still, the game works on both resolutions – we have made it a priority to make the UI as responsive to different resolutions as possible. We also implemented a custom UI colour system to let the product owner more quickly choose different colour palettes for the game. The default we chose is a blue-green palette with occasional red.

The first screen lets the user choose a nickname for themselves before establishing a connection with the server. Upon choosing a nickname, they are greeted with the lobby search

screen: 7

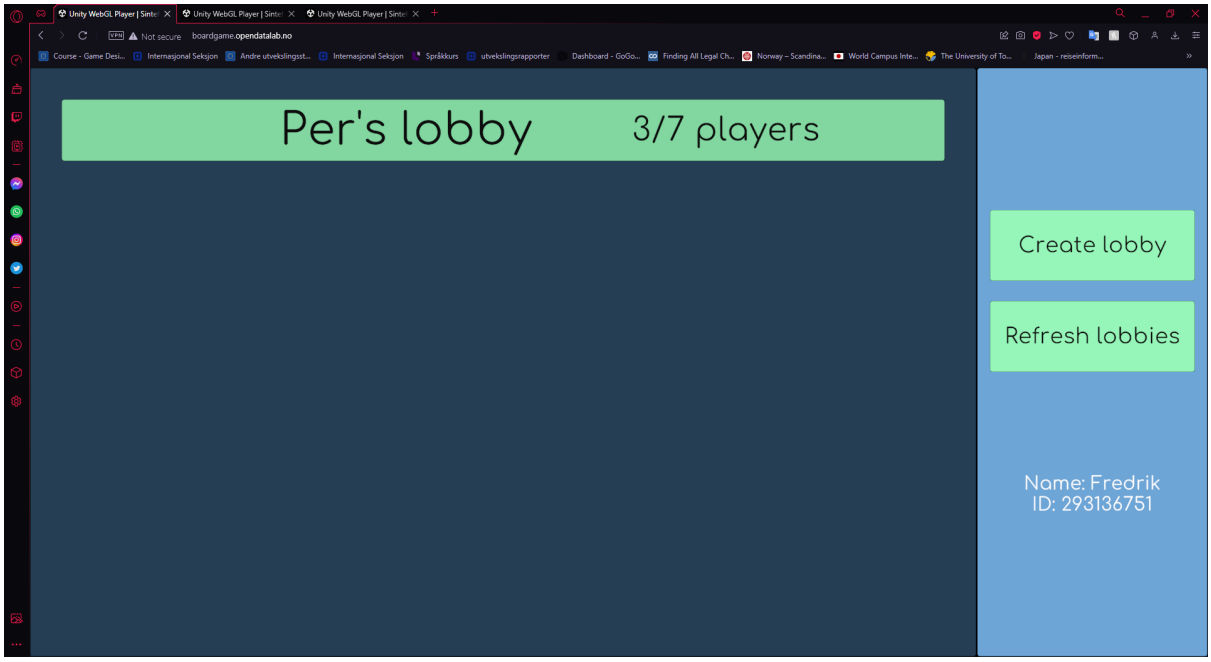


Figure 7: Lobby search screen

Here, the user can create their own lobby or join someone else's lobby. A "lobby" is a virtual room where players who want to play together gather before starting the game. After joining a lobby, players can see each other. This covers Fr1 in 2.2.1, since we can have 5 or more players in the lobby: 8

Figure 8: In a lobby (as a player)

How the same lobby looks like for the orchestrator: 9

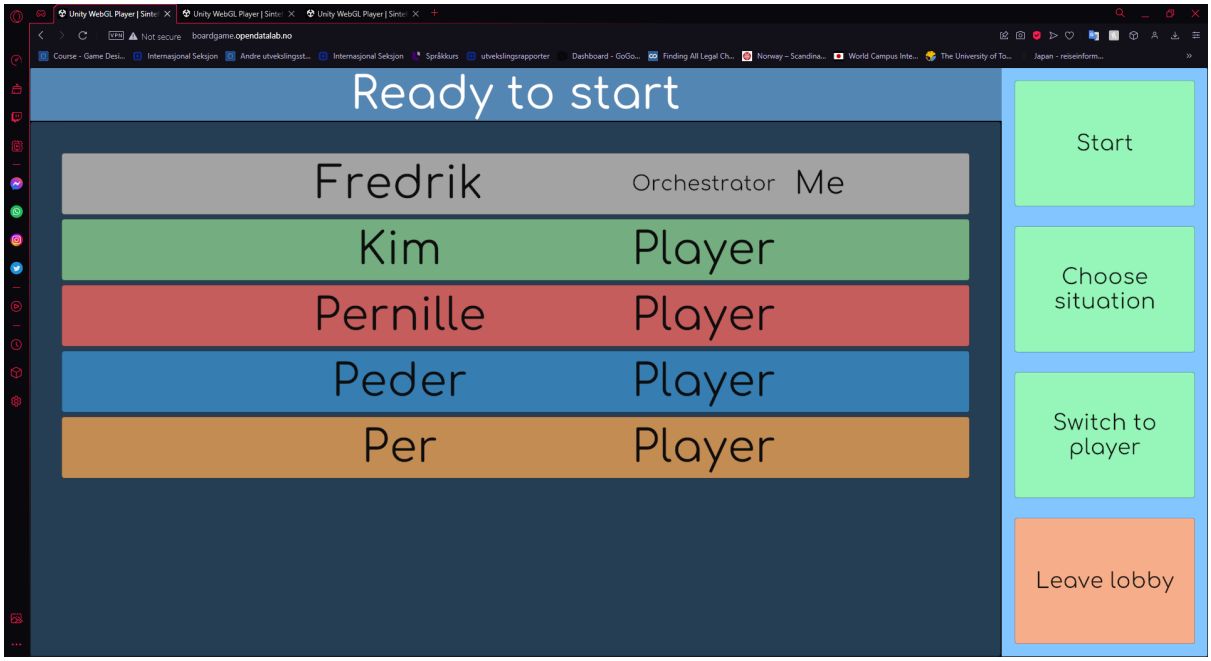


Figure 9: In a lobby (as the orchestrator)

The orchestrator "controls" the lobby in the sense that they can choose the situation and actually perform the action of starting the game. This covers Fr2, that the orchestrator can choose the situationcard. The situation choice screen looks like this: 10

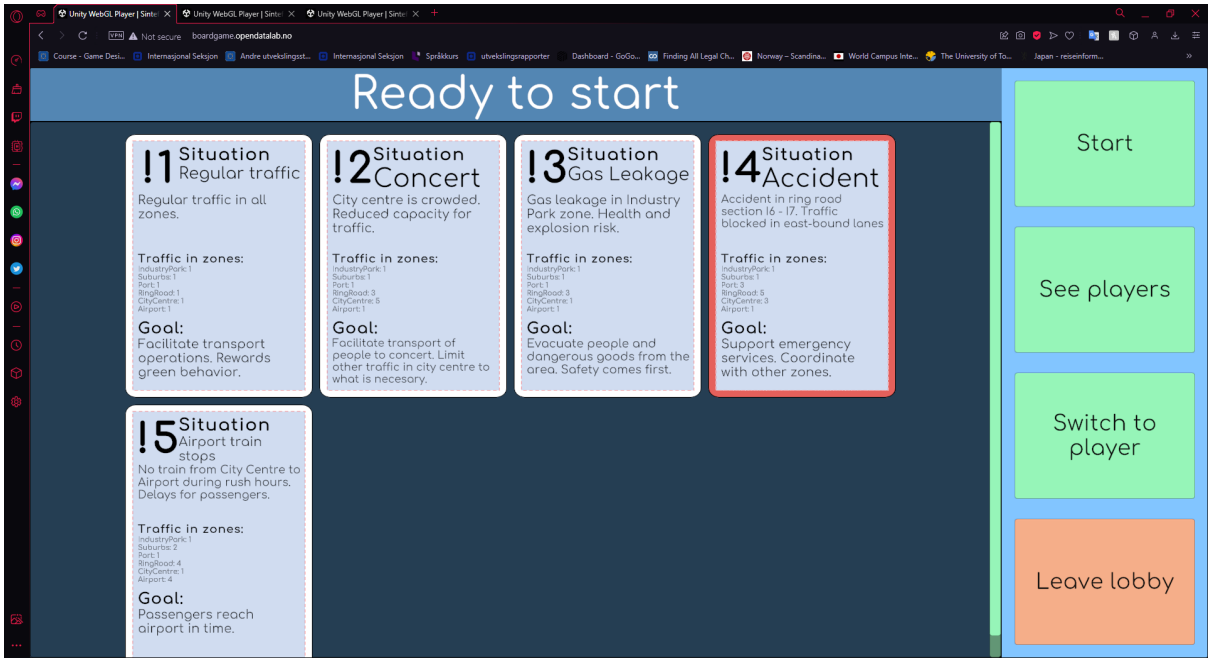


Figure 10: The orchestrator can choose the situation

After starting the game, all users are taken to the restrictions screen. Here, the orchestrator chooses traffic restrictions on different regions depending on what players need for their objectives. For example, the orchestrator can give priority to emergency vehicles in Suburbs, reducing traffic for those vehicles: 11

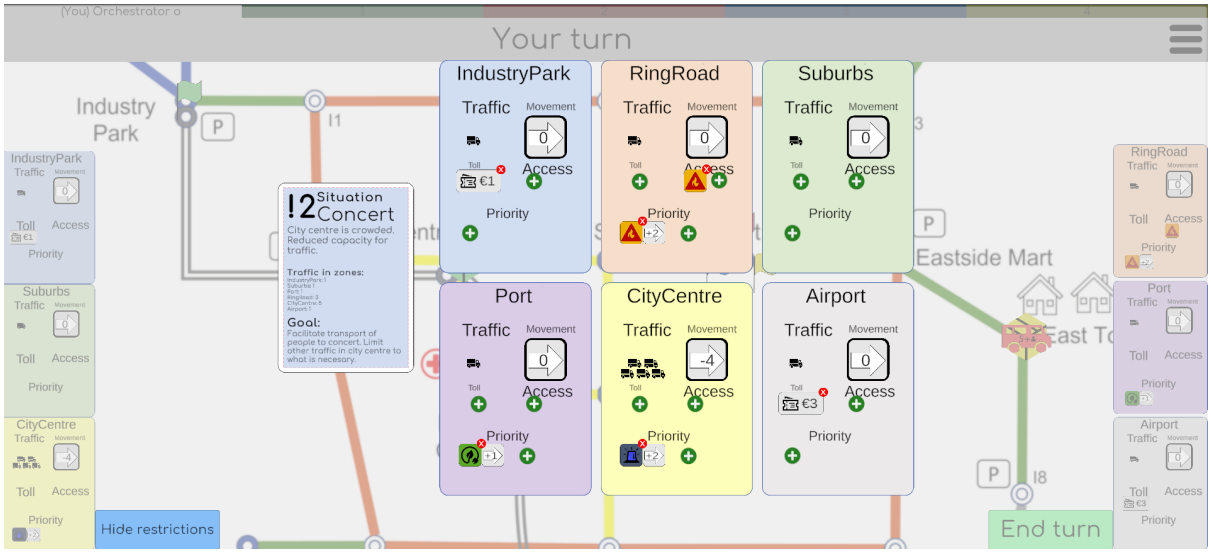


Figure 11: The orchestrator modifies traffic restrictions

Players sit and watch while the orchestrator creates restrictions. This solves the Fr5 by allowing the orchestrator to put measures into place: 12

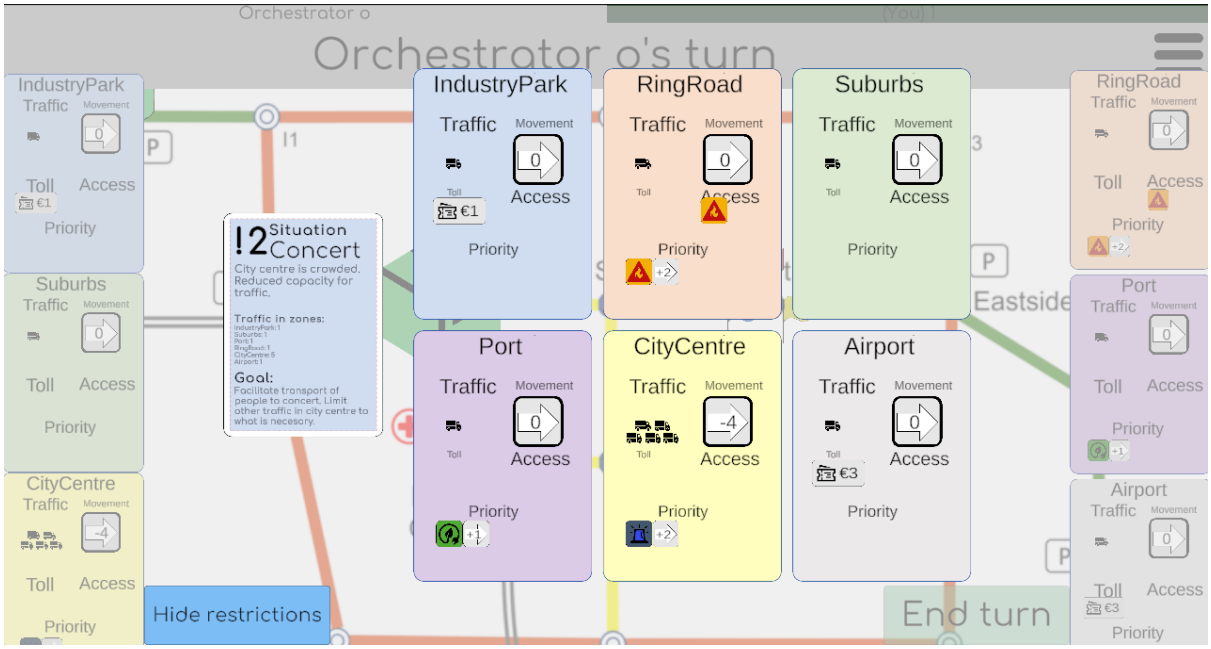


Figure 12: Players see the restrictions made by the orchestrator

After the restrictions have been made, players take turns in driving to their pick-up point, and then driving to their drop-off point to complete their objective. The players' pick-up, drop-off and starting positions are colour-coded markers on the map. Your car is picking up the corresponding package and bringing it to the goal (flag). This part of the game covers most of the functional requirements: 13

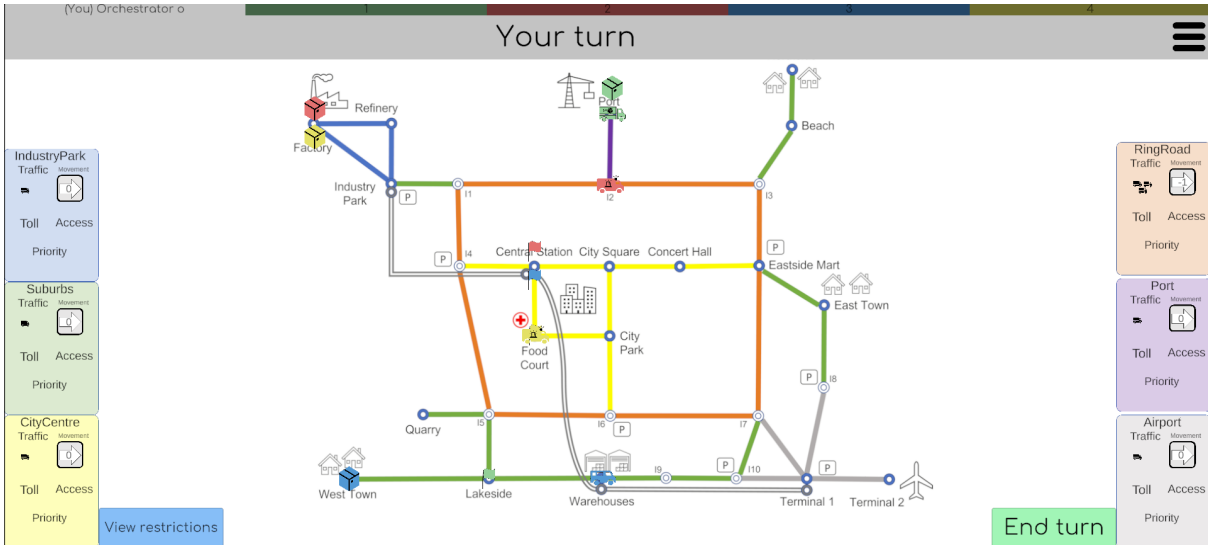


Figure 13: The game board, with visible objectives for each player

The game ends in success if all players manage to complete their objective, so the orchestrator's actions heavily impact whether everyone will complete their objective. After every player have finish their turn, the game loops back to the lobby. (figure 9)

2.5 Testing

2.5.1 Functional testing

The way the game works, the client and server exchange a significant amount of information. Due to the amount of communication between the client and server, either system is prone to errors. To mitigate the risk of errors, a proper test framework was prioritised. The tests would be designed to cover as much ground as possible. These include unit tests, integration testing, and user acceptance testing.

The benefit of unit testing was that the tests were automated and required little to no human interaction. These tests were beneficial for ensuring that the core logic works as intended and would be the primary testing method for the server logic. For the client side, bugs can occur due to a specific interaction that produces an undesired result. Some specific error cases can be clicking a button that does not do anything or that the client produces either incorrect or faulty data forms for the API calls. Some of the bugs for the client cannot always be caught by unit tests alone, which is why it is important to implement additional testing methods such as integration testing and user acceptance testing. It is worth noting that we stopped using and writing unit and integration tests towards the end of the project. Ultimately, we decided to remove the old tests from the server since we felt like they did not do much and cluttered the workspace. This was a combination of the multiplayer logic becoming too complex to be tested, and we had to prioritize other problems because writing the tests would take a lot of time, and given our time frame, we did not have time to do so. So we had to sacrifice quality for the sake of finishing the product on time. Since we had spent so much time on testing, the frontend team didn't have much to do during the first half of the project - it wasn't possible to start the game. We concluded that it was better for the backend to focus on functionality and rather test through the frontend manually, which sped up development for the rest of the project.

Writing software tests does also not seem like a trend in the games industry because, from our own experiences, a lot of games these days are released with lots of bugs and crashes. This is most likely because games are very complex and adding multiplayer functionality to games is even more complex. Writing tests that cover the code well will take a lot of time and since developing the game itself takes a lot of time, it is likely most often deemed unnecessary and playtesting is favoured because development can move faster.

2.5.2 Non-functional testing

Aside from the functional aspects, it was also important to ensure that non-functional requirements were met. These included issues related to response time, security and stability.

Although the product owner had no strict performance requirement, the goal was to keep the latency low by optimising the server. This was achieved by implementing techniques such as multithreading and writing efficient code. For the sake of non-functional testing, stability was the top priority. On runtime, it is desirable that the server can run with a load without eventually crashing, which can happen due to memory leaks. Memory leaks occur when a program does not free the memory it no longer needs. This should not be an issue with most modern programming languages, but it can still be useful to test memory usage to set a benchmark for performance and stability. With multithreading, the risks of race conditions may also occur. A race condition is when multiple threads try to access the same memory simultaneously. If either or both threads write to the same memory, an undefined behaviour may occur. The back end was written using thread-safe code to mitigate these errors. However, it was still important to test how the server handles multithreading. Since we were using Rust, none of these issues occurred due to the memory- and thread safety that Rust provides natively.

Another important requirement from the product owner was user-friendliness. Testing this requirement could not be automated and would be solved with user acceptance testing. One of the ways this is was done was by creating a Figma mock-up for the product owner to receive early feedback on changes made to the UI, but as the UI was beginning to take shape in Unity, we moved away from this. Receiving unbiased feedback on the UI was be essential to achieve the product owner's vision of the game.

2.6 Summary

The product is a digital cooperative board game about traffic management. In each game, the orchestrator changes traffic restrictions to reduce traffic for different vehicles. The players then take turns driving to their pick-up and drop-off points on the map. When all the players have completed their turn, the game is finished, and the players are returned back to the lobby.

The team performed a prestudy to find which technologies and libraries were suitable for the project before eventually agreeing upon which ones to use in the project, both within the team, but also with the customer. It was important for both the team and the customer that the technologies covered the demands of the product, as well as provided flexibility so that the product could be developed further after we had finished it. Also, the team's previous experience with technologies was taken into consideration.

Some of the requirements for the product were set by the product owner, but they were discussed with the team so that any questions or ambiguity could be cleared up. Both functional and non-functional requirements were an important part of the project, as the team had to keep both in mind during the process. The requirements were also the main drivers behind the architecture that we used, with the main requirements being that the game is a multiplayer game that is playable on a computer. In addition to this, the architecture should also have high flexibility and maintainability.

Functional testing for the game included unit tests, user tests and integration tests. However, we decided to drop it during the second half of the project to save time. Regarding non-functional testing, there has been a focus on performance, stability and usability, for which measures have been put into place. If we were to retry this project, we would try to ensure that the front end developers did not have to wait for backend functionality in order to start development on the game itself. In that case, we would have had to make an architecture better suited for development that allows offline play.

3 The process

3.1 Project management

3.1.1 Project plan

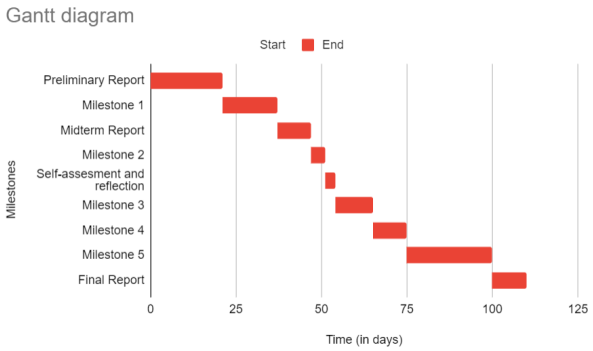


Figure 14: Initial Gantt diagram

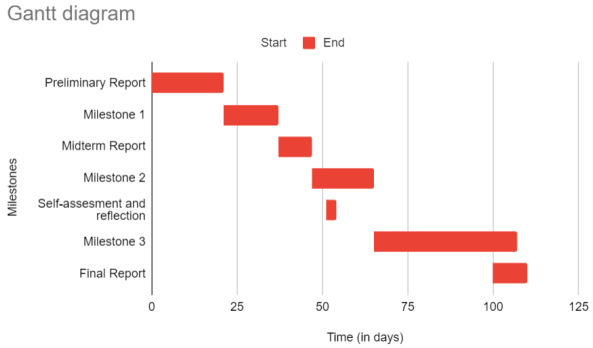


Figure 15: Final Gantt diagram

Week	Description
6	**Feb 10th: Preliminary report**
7	
8	**Feb 26th: Milestone 1**. Simple Unity preview of the board, the player and associated things. It is meant to reassure that the customer and our team have the same idea of how the game should look. Unity functionality is from the player's perspective → the player can move from one node to another. The server is set up to send and receive some information from the client. The code for milestone 1 is fast and temporary so we can benefit from early feedback.
9	
10	**March 8th: Midterm report.**
11	**March 15th: Midterm self-assessment and reflection**
12	**March 26th: Milestone 2.** Unity is set up to transport data between the client and the server. The orchestrator is also implemented in this milestone, both its view and possible actions. Menus, such as the start menu and main menu, are added. The implementation of the server side of the architecture is at full scale, but not all the rules of the game are supposed to be done.
13	
14 15	**April 3rd: Easter break** **April 11th: Back from vacation** **April 16th: Milestone 3.** There is full communication between Unity and the server. The extended version of the game should also be started. All the rules should be implemented on the server. The extended version should also be started. What should be included in the extended version will be discussed and agreed upon with the customer once applicable.
16	**April 23rd: Milestone 4.** The extended version should be finalised; this applies both to the server and Unity.
17	**April 30th: Milestone 5.** The server should be deployed. The milestone should be spent finalising all agreed-upon features and cleaning up code for readability and future development, including testing the system and fixing bugs. This milestone is also relatively open for possible unexpected work.
18	**May 3rd: Internal deadline for our report**
19	**May 10th: Deliver final report & video**

Table 3: Preliminary plan

3.1.2 Events and changes

3.1.2.1 Remote group member Our first problem was that one of our group members lived in Japan for the first half of the semester. This left us with two options: fly to Japan for every meeting or do the meetings digitally. With us students having limited time and not having infinite money or resources, we chose to have digital meetings. Sometimes we forgot to call him up during stand-ups, and he sometimes had trouble with his internet. If we exclude his internet connection and the fact that we sometimes forgot he existed, the solution worked well. But with his return from Japan in the latter half of the project, it is clear that physical collaboration is easier.

Setting up a meeting was also a small problem we encountered. Finding times when all 6 group members can attend meetings is complicated by itself, but combining time zones makes it harder. With Norwegian daytime being Japanese nighttime, our group member in Japan pre-

ferred early meetings at Norwegian time. With the members back in Norway having a tight schedule and not being early birds, we could only settle for one hour a week when every member could attend the meeting. This might sound dramatic, but we had 9 hours a week where 5 of 6 members could meet, thus having multiple hours a week where we could communicate efficiently with each other. It only took a couple of weeks before all group members started attending every meeting regularly.

3.1.2.2 Less works hours Another problem we encountered after the first week of work was that some group members had not worked the expected amount of hours, about 15, in a week. This was noticed and addressed in the group. We decided on a solution where we should measure if the average is about 15 hours per week over a more extended period, like three weeks, so even if someone works less than 15 hours one week, they can catch up in the coming weeks.

The 3-week moving average was not much used in the mid part of the project. This was due to discovering that much of the missing work was a result of people not reporting their work hours in our system. With the problem not being as big as expected, we mostly forgot about the moving average. In retrospect, this may have been a part of why we did not follow up on missing work hours as well as we could.

3.1.2.3 Multiplayer solution Early on, we decided to use a multiplayer solution through peer-to-peer hosting for the game's multiplayer functionality. This meant that the game was hosted through the game host's IP address. However, in the second meeting with the product owner, it was raised as a concern that this solution could cause problems due to most of the game's users playing on work computers with strict firewalls. With a peer-to-peer solution, there is a chance that it could be blocked when playing from different countries. With this in mind, the product owner and we opted for a different solution. This solution is to have a back end handle all the game logic while the client only updates the state of the game in the server, using a REST API. This solution should mitigate any firewall concerns since the host computer is at the product owner's discretion. This solution reduces the complexity of the Unity UI, making it simpler and easier to modify for future developers.

3.1.2.4 Change of backend language The multiplayer implementation was a clear objective when changing to a REST API solution using C# and .NET 6. As stated in the later risk analysis, this decision was based on the practicality of having a universal programming language across the entire project. With this, the solution was good on paper. But as with many things, this does not mean it works as expected. Early on in the development of the server, performance issues and bugs started becoming apparent. Consequently, a large chunk of time has been spent debugging the server. Since the performance would affect the user experience, other frameworks and languages were discussed. One of these options was Rust, and a server replica was created. This server outperformed the .NET 6 server by a significant amount. Moving the server over to Rust became an important talking point, and weighing the pros and cons would be necessary for making an informed decision. Right before deciding, one of the problems on the original server was solved, which made both servers perform at roughly the same speed. With the main problem out of the way, switching to Rust was no longer a performance question. But as previously stated, C# and .NET 6 introduced some bugs with handling API calls. It clarified that writing secure code with Rust rather than C# was easier. Since one of the

developers has much experience with Rust, the decision was made to move the server over to Rust. The risks are addressed in the next section.

3.1.2.5 Less work hours V2 Towards the end of the project, it became quite clear that some people had fallen behind on work hours. This was due to different reasons like travelling, other assignments and some personal matters. With some reasons being totally valid and unexpected, there is not much we can do to control them through the project, other than following up on it and making people aware of the situation.

As previously mentioned, we are not completely satisfied with how fast we followed up on people falling a bit behind. Regardless of the validity of the missing hours, it would help everyone to catch up or readjust the workload if we followed them up quicker. On the other hand, we could argue that this is not a group responsibility, but the personal responsibility of each group member.

For other non-valid reasons like travelling and other assignments, we could again argue that it is the individual group member's responsibility to plan this ahead of time. But if we look at this as a group, we could, for example, have sat down and restructured the work hours some weeks. Some weeks different group members could have had a heavier workload, thus giving them reduced weeks at a later point when they had other things planned. This is a solution which could have worked well since all group members know about their big assignments. It should not come as a surprise to them. Due to this issue coming up at such a late point in the project, we did not have any time to implement such a solution to the problem.

The problem wildly improved towards the end of the project, with people putting in 8+ hours each day, thus making their average hours a week 15+. This was not at all a solution to our problem, this was to make sure that everything was delivered on time. (See appendix 5.5 for the logged hours)

3.1.2.6 Unclear rules When starting this project, the formal task of the project was to digitalize the physical version of Sintef's game. We, therefore, expected to get a set of clear rules that we were supposed to implement, thus making it a straightforward process. But throughout the implementation, we realized that many of the rules were not set, or did not make much sense.

With Sintef's rules not being clear or making sense, we had to spend much of the time discussing the game and the rules, thus not giving time to work on our digital product. It sometimes also resulted in us having to set some parts of the development on hold until we could get a clarification on the specific rules.

We can also argue that discussing and guiding the product owner is a part of the group's responsibility and that we should expect this from product owners. But independent of this, we felt like we used most of the time discussing the rules.

It was not much we could do about this, other than try to discuss the relevant rules for the next week's implementation. But still ran into different "edge cases" with the questions: "What if that, and ... and ... that, what then?". Fortunately, this most of the time did not hold us too much back, since we just worked on some other feature in the meantime. Which was a solution which worked fine.

3.1.3 Risk-analysis

3.1.3.1 Early on The development team has recognised multiple risks that might affect the project negatively and have tried to minimise them by addressing them in the group and defining some solutions to minimise the damage. The full list and analysis of the risks can be found in the risk analysis table in chapter 5.4 in the appendix.

One of the main risks in the project is the volatility of the game rules. The current rules of the game are not yet finished and are very likely to change. However, the higher "logic" of the game is very unlikely to change. To address the volatility of the game rules, the best option is to create a robust system against volatile rules. So, it is imperative that changing and adding new rules is an easy task.

One more big risk is the possibility of disagreements internally in the group. The risk level has been reduced from high to moderate by planning to make changes and solve conflicts and embedding that into the group contract that everyone signed (5.3 in appendix). In addition, it is encouraged that all group members share their ideas and opinions and that a general consensus can be reached.

Another important risk in the project is the uneven workload among the members. This risk will be significantly reduced by describing what tasks need to be done and adjusting the amount of work and tasks based on the situation. It is also important that we help those with less experience and or knowledge so that they feel more comfortable with their tasks.

Since the project may be passed on to other developers, there is a risk of the product not being of a high enough standard. This could be through messy code, unstructured architecture and a buggy game in general. Thus it is extra important to keep everything structured to minimize the risk of future developers not having a pleasant experience. This risk was the main reason the structure is divided into different modules. And also the reason behind the strict use of namespaces makes it clearer for future developers which part of the script's code belongs. Further info can be found in section 3.3.1. Overall it helps future developers and us, thus reducing the risk.

3.1.3.2 Throughout the project The previous solution having the same programming language made it easier to work full stack on the project. This ultimately made it easy to reorganize people if there was any group member who, for some reason, had to stop working on the project. Changing the language the server runs on has made it more difficult to be a full-stack developer on the project if the person is unfamiliar with either C# or Rust. If a group member drops out, it could potentially lead to a knowledge gap which is difficult to fill. This could delay the project and prevent it from being completed in the worst-case scenario.

With this relatively high risk, actions have been taken to reduce the possible consequences. Code reviews can help in reducing the knowledge gap and making more team members familiar with how the code works, whether it is an implementation on the client or server side. In the case where a group member drops out, reducing the knowledge gap should decrease the severity of the consequence.

To quickly summarise, the project has some risks that have been recognised and addressed to minimise the potential damage they may cause. Other smaller risks have been lightly addressed in the appendix.

3.2 Team organisation

3.2.1 Roles and responsibilities

NOTE: The roles below come in addition to working on the project, but some of the time that would have been used on the project will be used on the role's responsibilities instead.

Leader (Mathias Gunnarshaug)

- Responsible for communication with the supervisor and the product owner.
- Responsible for keeping a general overview of the project. Such that they can answer questions from team members. This person also leads the ”agile” meetings.
- Keep up to date on the progression chart and summarise it at the start of every meeting.
- Maintain the time chart.

Technical leader (Torbjørn Stakvik)

- Should have a general overview of the whole system.
- Responsible for GitHub and Unity -> Administrative.
- Responsible for making concrete tasks.

Room and socially responsible (Runar Johnsen)

- Book room for meetings.
- Social happenings (party, dinner, etc.).

Document manager (Tim Matras)

- Prepare documents (everything from meetings to reports).
- Deliver documents on time.
- Is not responsible for writing all the documents but preparing and delivering them.

Remaining group members (Martin Larsen and Thomas Johansen)

- Responsible for filling in where needed.

All group members

- If a member is working on the UI, then that member is responsible for ensuring that a prototype in Figma is made before it is implemented so that we can get early feedback from the customer and team. That member is also responsible for ensuring that all feedback is processed and addressed and that the prototype is upkept. This is to ensure a good-quality UI.
- If a member is working on something that can be Unit tested (except for mock code), then that member is responsible for ensuring that good covering tests are made, such that the code written behaves as expected.

3.2.2 Routines

Common schedule:

Tuesday	Wednesday	Friday
Work: 12:00 to 15:00	Internal meeting: 12:00 to 13:00 Work: 13:00 to 15:00	Work: 12:00 to 15:00

In addition to this, we had frequent meetings with Sintef and with our supervisor. As every member is expected to work 15-20 hours a week we only had group meetings for 9 of those hours, the rest of the hours were up to each individual member to complete at a time convenient to them, working on issues agreed upon during the meetings.

3.2.3 Competency matrix

	Martin	Mathias	Thomas	Tim	Torbjørn	Runar
Unity						
Artwork/Design						
Reports						
Maths						
Software Architecture						

3.2.4 Group and customer collaboration

We made a group contract early on in the project so that everyone agreed on the workload, how to deal with disagreements, etc. The group contract can be found in chapter 5.3. We agreed on having three weekly meetings with the group. Digital meetings with the group were held on Discord. For quick and short messages within the group messenger was used. Google Disk was used to store charts, diagrams, and links to documents. These were used as all members were familiar with the technology. Meetings with the customer when needed where we have asked for a meeting when there, for instance, was a need for clarification of the rules or to show the progress of the game. These meetings with the customer have been about every other week. As well as a handful of meetings with our supervisor. The first part of the project was dedicated to working on the report and agreeing upon which technologies to use.

We first met with the customer in week 4, where we were introduced to the board game. First, the customer had a presentation for us, where we talked about the project that the game was a part of and why the game was made. The customer's motivation for making the board game was to use the game during workshops so that the people involved in the project could understand the difficult topic of traffic management in a fun and engaging way. Early on, they had developed a more complex version of the board game, but they quickly realised that this version was not very suitable for playing during a workshop, at least not as a board game, as it would take too much time to play. The customer did, however, want the digital version of the board game to implement or at least be able to implement more of the complex rules not included in the workshop version.

We quickly selected a group leader, this person was responsible for contacting the customer, both when we had questions regarding the project and arranging more meetings with the customer after our first meeting. Correspondence with the customer was done through email, with the other group members on carbon copy. The group leader was also responsible for checking that the group members had received the same emails and, therefore, the same

information from the customer as the group leader. The group leader also emails the customer every Friday to describe and summarise the group's work that week.

With our first meeting being an introduction to their project and the board game, our next meeting was set up so that the customer could answer questions about the project, such as the board game rules, non-functional requirements and milestones. Having a meeting with the customer, instead of sending an email, made room for discussions related to our questions and gave both our team and the customer a common understanding of the project's progress. Meeting the customer instead of communicating by email reduces the chance of misunderstandings and errors. We got to know each other better and became more comfortable with each other. This made future communications a lot easier and more reliable.

The meetings with the customer are a great way of receiving immediate feedback about design choices or giving a demo of how the game looks as it is at the present time. The feedback was then used to improve the game even more and tailor it to the customer's needs before the next meeting.

Throughout the project, we realised that the meetings were especially great for getting feedback on our design choices and clarifying game rules. With both of these things not having too many restrictions from the start, we had to get feedback from the customers on potential changes in design and rules. Through these discussions, we put forward our thoughts and ended up with the most optimal solution.

3.3 Development process

3.3.1 The development process

We tried to keep in mind the non-functional requirements throughout the development process. We planned to build the project so that every module in the frontend's architecture is written in its own so-called "assembly". This speeds up compile time and forces us to adhere to the architecture (dependency hierarchy). By doing this, it is easier to write clean code. In addition to using assemblies, we also used namespaces. All scripts in an assembly had a corresponding namespace, which immediately tells which assembly a script belongs to, and makes it obvious which scripts can be accessed by others. We further used namespaces to separate unrelated code within the same assembly. The backend uses a similar approach where the different server files are put in separate packages/crates, depending on which purpose they serve for the backend. This in return makes the code look more organized and easier to navigate.

On the server side, we needed to make sure that the code was documented well. This is because we want future developers to have an easier time when picking up the project after us. Rust's Cargo provides a built-in documentation builder[3] which can compile the code documentation into an interactive website. This can make it easier for future developers to work on the server as they can quickly search for the data structures or functions they want to know more about on the interactive website. In addition to this, we decided that it would be a good idea to add a README in the repository which gives an introduction to the server side of the project, including how to set up the server and an overview of how the repository is structured.

To make it easier for the client side of the project to interact with the system, we made a Rest-API document that would be updated along with the server. This document follows the OpenAPI3.0 specification [4], making it possible to render it as interactive documentation in Swagger UI or Swagger Editor, which are free tools made by those who made the OpenAPI specification that represent the Rest-API interface in a clear and easy to understand way. This is great for future developers, as those who want to interact with the server can follow the specifications defined in the Rest-API documentation, which is easier to understand than looking

at the code itself.

We used GitHub (defined in 5.1) for collaboration. As this was the wish of the customer, and all the members knew how to use it. Each GitHub issue was based on a functional requirement and placed on a GitHub project board depending on if they were free to be started on, worked on, or finished. They were also sorted into which part of the project they belonged to (report, client or server). A new branch was created for every new issue that was being worked on. The main branch in GitHub will be protected from direct pushes and will instead require a successful pull request. Pull requests are requests made to the repository that requests that the changes from one branch, be merged into another branch. The way the process around changes is structured is that the changes made will be pushed into an unstable dev branch, and once there's a stable product, it will be pushed into the main branch. When creating a pull request, reviewers can be assigned. What has been practised is that those who primarily work on the server side, review pull requests made to the back end, whereas those primarily working on the client side review pull requests made to the front end. The reason this is practised is to ensure quality in the working product, and so everybody can be on the same page in terms of what is functioning in the code.

In the very early stage, Figma prototypes were used to obtain feedback about client-side design choices. However, as the project progressed, the use of Figma prototypes diminished in favour of game demos, which have proven successful in getting feedback.

Pair programming has been practised to some extent, mostly when it is about troubleshooting code, or solving merge conflicts. Merge conflicts occur when there is a clear mismatch of the files one person has changed, with what is currently in the branch they are trying to merge their code into. Aside from that, we have mostly opted in for working individually on issues, with a standup for each meeting where progress is shared with the rest of the group.

3.3.2 Project history

Date	Event
24.01.2023	First meeting with the product owner. We were given a short introduction to the project and its motivations. The meeting ends with everyone playing a game on the physical workshop version.
03.02.2023	Mailed the product owner to ask if the multiplayer should be implemented locally or over the internet. The response was that multiplayer should be implemented over the internet.
08.02.2023	Meeting with the product owner. In this meeting, a P2P solution was proposed but rejected in favour of a client/server architecture. After some planning, a REST API implementation was agreed upon.
15.02.2023	It was agreed upon that testing should be a priority in the project. A plan for how testing was to be carried out has been made. For the front end, it has been agreed upon that Figma would be used to create a prototype. One reason is early feedback from the product owner. Another one is that it creates a clear goal of what the client is supposed to look like.
16.02.2023	Test tool for the back end has been agreed upon. xUnit is to be used for testing the server. xUnit is a testing tool for the .NET framework.
26.02.2023	Milestone 1 is finished but the project does not satisfy all criteria for the first milestone. Discussion around the question as to why it wasn't finished has taken place.

27.02.2023	The group comes to an agreement that those working on the UI have the responsibility of maintaining a Figma prototype so that early feedback can be given on the design before it is implemented. Those working on features that have to be unit-tested should also write the tests.
03.03.2023	The group starts looking into issues regarding performance and difficulty of writing code in C# for the server. Rust is suggested as a new language to address these issues, given the convenience of the language. Over the weekend, everyone is to be expected to form some opinion on the pros and cons of switching to Rust.
07.03.2023	Pros and cons of migrating the server over to Rust have been discussed. The main arguments for migrating the code to a different framework have been performance, as well as time saved on debugging in the long run. After some discussion, it has been agreed that switching over to Rust is the best course of action. The product owner will be notified of this change, but the change will not go through before the product owner has accepted this.
09.03.2023	We were given a green light to switch the back-end code to Rust by the product owner.
14.03.2023	A new decision was made, to fix a short agenda of what is going to happen in the three weekly meetings, for a more concise meeting plan. A new rule was implemented that makes it so only someone who has not worked on a branch may approve the pull request, which avoids any bias for code review. Milestone 1 was merged into the main branch. A new rule has been added which states that meetings happen at the hour and not 15 minutes afterwards unless a group member has a lecture preventing them from joining the meeting on the clock.
18.04.2023	Meeting with the product owner. Concerns regarding not finishing the extended version have been brought up and it has been decided that we should focus on finishing the workshop version of the game. Questions about hosting and domains have been brought up. The product owner says they are going to fix a domain for the game to be hosted on.
24.04.2023	Deployed server on the domain provided by the product owner.
09.05.2023	Final meeting with the customer and we did a playthrough of the final prototype. The customer seemed satisfied with our version and looked forward to expanding the game towards the extended version.

3.4 Summary

The group encountered several problems during the course of the project. We initially had issues with communication as one group member was in Japan with a relatively poor internet connection, and not everyone has fulfilled their minimum work hours. Deciding on a network solution also proved challenging, first suggesting P2P before eventually going for client-server using a REST API. We then experienced performance issues on the backend, after which we decided to change the backend to use Rust instead. Changing to Rust was a risk in the sense that fewer knew the language, which could have slowed the process heavily if they decided to leave the project. They did not, so the risk paid off. After experiencing zero runtime errors on the backend, which saved a lot of development time for the front end, we can safely say that using Rust for simple multiplayer games is a great choice. We can't recommend poor internet, however.

It was good that we had defined an architecture, although it was not final, at the beginning

of the project. This helped us with understanding how the code worked together whilst we worked on the project. This is something that we would definitely do again, however, next time we could probably use more time on the architecture because it changed a lot during development. Maybe a more suitable architecture could be better defined if we'd use more time on it. At times it was hard to use Git and GitHub since merge conflicts occurred multiple times, especially for those who worked with Unity. Unity has another way of collaborating like with Git, it's called Unity DevOps and is better geared towards game development[6].

4 References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2021.

[2] Marco Couto et al. "Energy Efficiency across Programming Languages". In: (2017). DOI: 10.1145/3136014.3136031.

[3] The Rust Programming Language. *cargo-doc*. URL: https://doc.rust-lang.org/cargo/commands/cargo-doc.html (visited on 05/10/2023).

[4] SmartBear Software. *What Is OpenAPI?* URL: https://swagger.io/docs/specification/about/ (visited on 05/10/2023).

[5] The Actix Team. *Actix Web*. URL: https://actix.rs (visited on 05/01/2023).

[6] Unity Technologies. *Unity DevOps*. URL: https://unity.com/products/unity-devops (visited on 05/10/2023).

5 Appendix

5.1 Definitions

- Unity: Unity is a game engine which drastically simplifies the process of building a game.
- C#: the scripting language used with Unity - which is the language its users code with.
- REST API: Short for Representational state transfer application programming interface. REST API makes it possible to send and receive HTTP requests between networks.
- .NET 6: .NET is a framework for the C# programming language, and is also used when scripting in Unity.
- Rust: Rust is a highly praised programming language that is known for fast runtime, helpful compiler errors, and forcing users to write safe code.
- Actix: a web framework for Rust, which can be used to create a REST API.
- Github: Hosting service which provides centralized storage with which developers can use to collaborate code using Git.
- Git: Distributed version-control software that specializes in merging different code changes fast.
- Tabletop Simulator: Tabletop Simulator is a game available on the Steam platform where users can either create board games using a high-level interface or play these games.
- Mirror: Mirror is a Unity package that provides functionality for networked games.
- uGUI: Unity's legacy UI solution.
- UI Toolkit: Unity's new UI solution.
- Godot: a relatively simple game engine that can either be scripted using Golang or visual nodes.
- Unreal engine: A high-performance game engine with support for advanced graphics. The scripting language is either C++ or node-based.
- WebGL: A web player based on OpenGL. By exporting to WebGL in Unity, the developer gets a webpage where the game can be played.
- MVC: Model-View-Controller is an architectural pattern which separates the program's state (Model) from the visual representation (View) and manages the two using Controllers.
- ECS: Entity-Component-System is a common architectural pattern in game engines that treats every visible object as Entities which have several Components to define behaviour. These are managed by Systems.

5.2 Milestone objectives

5.2.1 Milestone 1

ID	Objective	Measurement	Goal	Result
O1	Unity: The gameboard is visible on screen	Visibility: Yes/No	Yes	Yes
O2	Unity: The players are visible on screen	Visibility: Yes/No	Yes	Yes, one player is visible
O3	Unity: The current player is highlighted on screen	Visibility: Yes/No	Yes	No
O4	Unity: The package pickup and dropoff are visible on screen	Visibility: Yes/No	Yes	Yes
O5	Unity: The nodes are highlighted when hovered over	Visibility: Yes/No	Yes	Yes
O6	Unity: The nodes are clickable	Clickable: Yes/No	Yes	Yes
O7	Unity: The players can move from their current position to another node by clicking on the node they want to go to	The player moves to that node: Yes/No	Yes	Yes
O8	Server: Retrieves REST API call for getting a unique player ID and returns it	Retrieves unique player ID: Yes/No	Yes	Yes
O9	Server: Retrieves REST API call for starting a new game with a specified name and player and creates it	Returned REST API object is a game state with the name specified: Yes/No	Yes	No
O10	Server: The server receives a REST API call for moving the player to another node and returns the game state with that update.	Returned REST API object is a game state with updated player placement: Yes/No	Yes	No

5.2.2 Milestone 2

ID	Objective	Measurement	Goal	Result
O11	Unity: Sends correct REST API objects for input to the server	Input is sent correctly: Yes/No	Yes	
O12	Unity: Receives and handles the game state object from the REST API correctly	Received and handled correctly: Yes/No	Yes	
O13	Unity: The start menu is visible when starting the game	Visibility: Yes/No	Yes	
O14	Unity: The buttons on the start menu go to the scene they are supposed to go to	Visibility: Yes/No	Yes	
O15	Unity: The zone cards are visible on the screen	Visibility: Yes/No	Yes	
O16	Unity: The parts of the zone cards that can add restrictions should be pressable	Visibility: Yes/No	Yes	
O17	Server: Player movement is restricted based on how much the player already has moved	Movement restricted: Yes/No	Yes	
O18	Server: The response time from an input (sent to the server) to a new game state received should not be too much when the client and server are on the same computer	Response time in milliseconds from new input sent to handled game state is received	<=100ms	

5.2.3 Milestone 3

ID	Objective	Measurement	Goal	Result
O19	Unity: The pickup and drop-off points are placed correctly based on the drawn situation cards	Placed correctly: Yes/No	Yes	
O20	Unity: Players can see how many actions they have left	Can see remaining actions: Yes/No	Yes	
O21	Unity: Players can see which legal nodes they can move to	Legal nodes are highlighted: Yes/No	Yes	
O22	Unity: District cards containing information about the districts should be displayed on the game board	Can the players see the district cards: Yes/No	Yes	

O23	Unity: The orchestrator should be able to interact with the district cards to apply restrictions	Can orchestrator interact with the district card: Yes/No	Yes	
O24	Unity: Players should be able to see how much money they have	Can see money on screen: Yes/No	Yes	
O25	Server: Retrieves REST API call for lobby list and returns it	Returned API object is a list of lobbies: Yes/No	Yes	
O26	Server: Retrieves REST API call from a player for joining a lobby and returns the lobby with the player in it	Returned API object is the wanted lobby: Yes/No	Yes	
O27	Server: Retrieves REST API call from a player for leaving a lobby	Player connected to the lobby: Yes/No	No	
O28	Server: Retrieves REST API call from a player for leaving a game	Player connected to the game: Yes/No	No	
O29	Server: The orchestrator can start the game	Game is started: Yes/No	Yes	
O30	Server: A player that is not an orchestrator cannot start the game	Game is started: Yes/No	No	
O31	Server: Only one player can do an action at a time	All other players are restricted: Yes/No	Yes	
O32	Server: The orchestrator can add restrictions to a district	Can add restrictions: Yes/No	Yes	
O33	Server: The players are restricted based on the restrictions	Yes/No	Yes	
O34	Server: Add the ability to undo on your turn	Can undo actions: Yes/No	Yes	
O35	Server: Retrieves a REST API call for getting all legal nodes and returns a list with legal node IDs	Sends correct list: Yes/No	Yes	
O36	Server: Removes unused IDs after a certain amount of time	Removed unused IDs: Yes/No	Yes	

O37	Server: Empty (no players) games should be removed automatically	Empty games removed: Yes/No	Yes	

5.2.4 Milestone 4

O38	Unity: Road users can choose multiple assignment cards at the beginning of the game	Road users can choose multiple assignment cards: Yes/No	Yes	
O39	Unity: The Orchestrator can see how much money they've gotten from tolls	Orchestrator can see how much money they've gotten: Yes/No	Yes	
O40	Unity: The Orchestrator can buy upgrades with their money	Can buy upgrades: Yes/No	Yes	
O41	Server: Road users can have multiple assignment cards	Can have multiple assignment cards: Yes/No	Yes	
O42	Server: Road users can assign assignment cards to themselves	Can assign assignment cards to themselves: Yes/No	Yes	
O43	Server: Road users get money from completing assignments	Get money: Yes/No	Yes	
O44	Server: Road users lose money when entering a district with toll	Lose money: Yes/No	Yes	

5.3 Group contract

Group-contract

Full name	Phonenumber	Email
Martin Valderhaug Larsen	48258766	martinvlarsen01@hotmail.com
Mathias Gunnarshaug	91911137	mathiasgun@icloud.com
Thomas	47844946	thomawj@stud.ntnu.no
Tim Matras	48182233	timma@stud.ntnu.no
Torbjørn	41368522	torbjss@stud.ntnu.no
Runar Johnsen	99597498	runarjo@stud.ntnu.no

Motivation and effort

What is the level of ambition when it comes to the evaluation of this project, in the team?

Ambisjonsnivået vårt er på å nå A-karakteren.

What competency does the team have?

- Unity
- Design
- Board-game knowledge
- A lot of experience
- Software architecture
- Documentation/Writing reports

How do you define if some work is “done”?

To define a piece of work as completed, it must meet all requirements that have been set and pass any tests that have been defined. It is also important that new code that is written is reviewed and approved by at least one other person in the group so that it can be ensured that the code is correct and that there are no errors or omissions.

How much do the group members expect to work on the project?

Average 15 to 20 hours a week. Exceptions can be granted during unforeseen events (such as illness, funerals, etc.).

Routines and communication

When are we going to work together?

Meeting plan:
Tuesdays: Working session: 12:00-15:00
Onsdag: **Internal meeting: 12:00 - 13:00**, Working session: 13:00-15:00
Fredag: Working session: 12:00-15:00

Other meetings are planned as needed, but we cannot expect *attendance* at those meetings.

What routines do we have for work?

- If a team member participates digitally in meetings to work, it must be done through Discord unless otherwise specified.
- We will use the GitHub issue board and issues to break down larger tasks into smaller tasks that are feasible.
- Each issue must have its branch.
- The main branch on GitHub is protected and can only be updated through pull requests.
- For a pull request, another person, who has not worked with the branch, must approve the pull request
- We pair-program as needed.

How do we communicate in the group when we are not gathered?

- We use messenger for important common information.
- We use Discord for collaboration between team members and information that is not critical.
- External meetings and communication with people outside of our group is done through their desired digital platforms or physical meetings.

How are we going to give each other feedback?

The team will implement code reviews to ensure the high quality of the code. Before integrating new code into the main branch, it should be reviewed and approved by at least one other group member.

Feedback on documents and reports must be given using the comment function in the Google Docs documents and the equivalent in Overleaf.

Feedback on the group's dynamics must be given continuously.

All feedback must be constructive, diplomatic and not personal.

How should we make decisions?

To make a choice regarding the group, the majority must be in favour. That is 4/6 with the current members.

If it is split 3 to 3, then talk to the supervisor. If no one has changed their mind after that, Spin The Wheel decides.

What do we do when we don’t have anything to do?

If you have nothing to do, you let the group know that you are free and need a task to work on.

What do we do if we are stuck with a task?

If someone is stuck, they must let the group know, and then the first person who has the opportunity to help will help.

Roles

Role descriptions:

Leader (Mathias Gunnarshaug)

- Responsible for communicating with supervisor and product owner.
- Responsible for keeping a general overview of what is happening in the project. So that we have someone to go to. Also, the one who leads the "agile" meetings.
- Keep up-to-date on the progress chart, and present it immediately before each meeting.
- Maintenance of timesheet form.

Technical resposible (Torbjørn Stakvik)

- Must have a general overview of the entire (project) system.
- Responsible for GitHub and Unity -> Administrative.
- Responsibility for creating specific tasks regarding the technical aspect of the project.

Room and socially responsible (Runar Johnsen)

- Book rooms for meetings and work sessions.
- Social gatherings (paries, dinners, etc.).

Document responsible (Tim Matras)

- Prepare documents (everything from meeting minutes to reports).
- Deliver documents on time.
- IMPORTANT! Not responsible for writing the entire document, just getting ready to write (setting up the document, etc.) and delivering.

Group members (all)

- If you work with UI, you are responsible for making a (Figma) prototype before it is implemented so that you can get early feedback from the customer and the team. You also have the responsibility to ensure that the feedback on the prototype is processed and relayed to the team, and you also have the responsibility to ensure that the prototype is maintained. This is to ensure good quality of the UI.
- If you are working with something that can be Unit tested (without mock code), then you are responsible for making sure that comprehensive Unit tests are written so that you can confirm that the code you write behaves as expected.

What are our expectations of the team members?

1. We expect all roles and members to do their jobs.
2. We also expect those with specified roles to check messenger at least once every day if there are any questions related to them.
3. Everyone works roughly the same number of hours, but those with specified roles spend some time on their particular tasks.

How do we choose a candidate for a role?

Those interested in a specific role, let the team know. If we disagree about who should have the role, you resolve it with Spin The Wheel. If some roles are not filled, we use Spin The Wheel on the remaining members who do not have a role to fill the roles.

Conflict handling

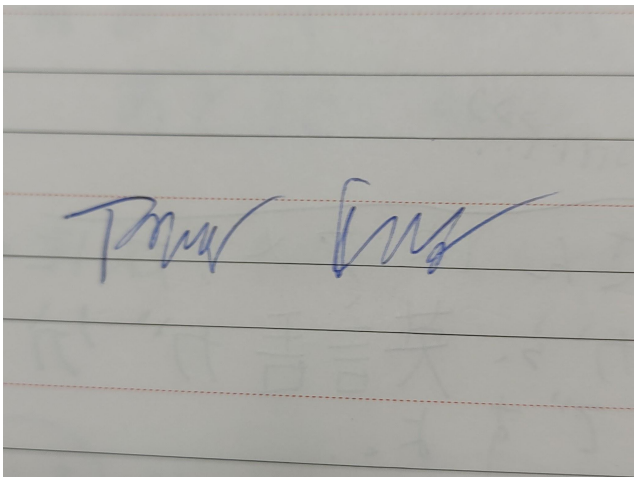
How do we handle disagreements?

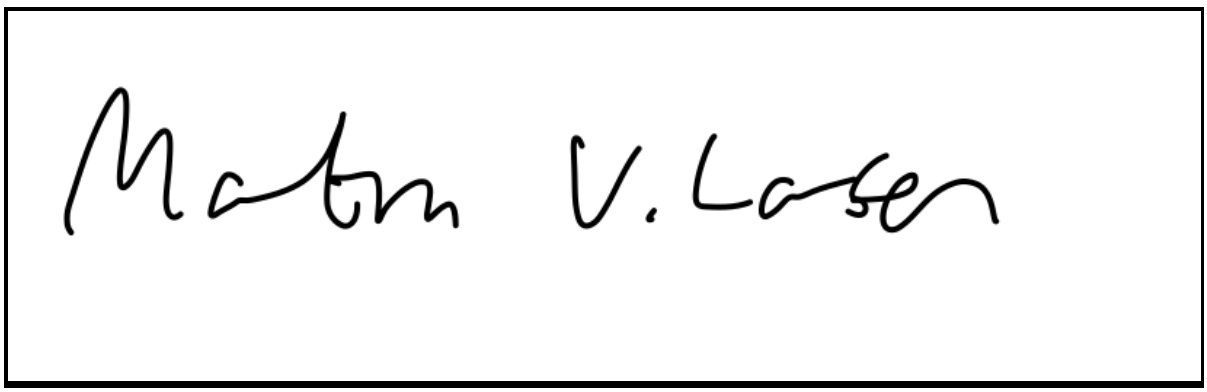
If a disagreement becomes difficult to solve, we try to resolve it in the same way as we make decisions. We, therefore, expect all members to behave respectfully.

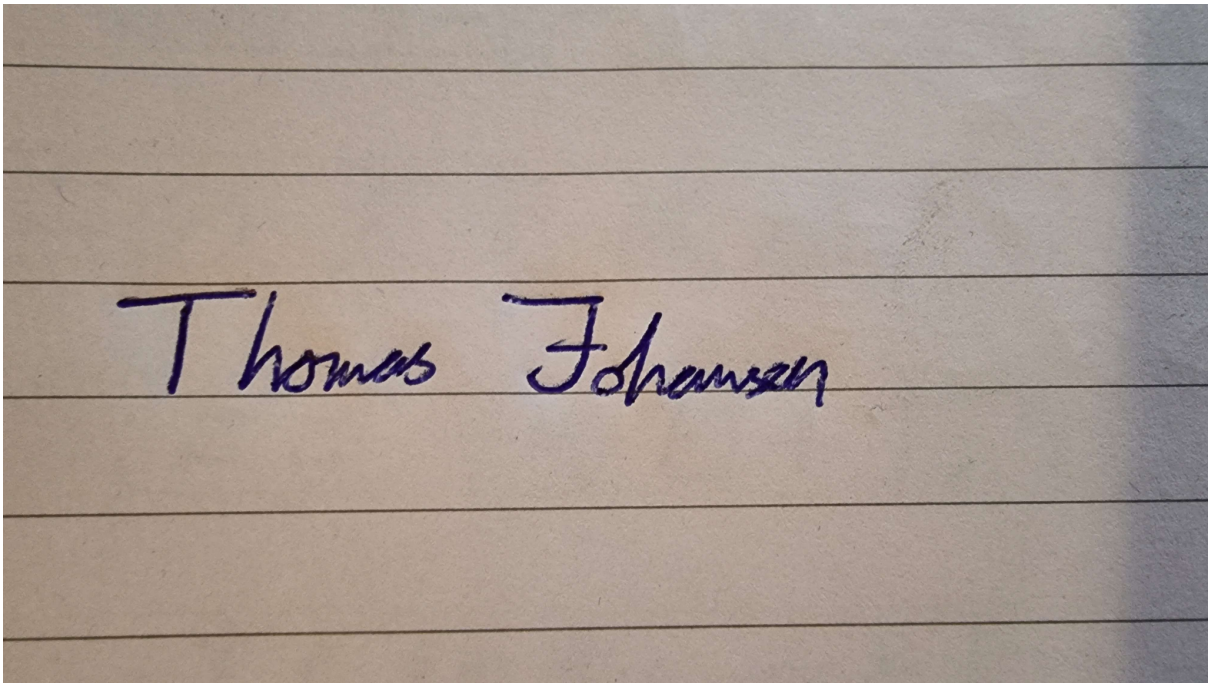
How do we handle breach of contract?

We try to resolve breaches of the contract respectfully. If it doesn't work, we contact the supervisor to find a solution.

Signatures:






Thomas Johansen


Tim Matoras

5.4 Risk analysis table

ID	Risikobeskrivelse/ Risk description levels: lav/low, moderat/moderate, høy/high, svært høy/very high Stikkord om: (1) innledende hendelse(r) (2) informasjonssikkerhetsbruddet (3) de uønskede konsekvensene som kan oppstå	Reason for consequenceassesment	Consequences	corresponding probability	Reason for assesment of corresponding probability	Risk Level	Assesment on handling the risk	Consequences after measures	Probability after measures	Residual risk
R1	Teammates dropping out	If a person drops out, the rest of the group will have to do more work.	Very high	Low	Unforseen events can happen that result in someone having to drop out.	Moderate	We will reduce the scope of the project, so that we do not become overworked. Additionally we will inform the relevant people about this. With all memebrs having to review code, all should have relativly good control on hw to fill any role.	Moderate	Low	Low
R2	Uneven workload	The group members have different levels of experience with the technology we are going to use. Lack of clear directions of what to do when you have nothing to work on. Unclear what work there is left to do.	High	Moderate	The members of our group have different levels of experience with the different parts of the project, which can make some mebers unsure of what they are supposed to do. There has also been situations where it has been unclear who is going to do what.	Moderate	Describe which tasks that have to be done. We should dynamically change the workload and tasks based on the situaution. Additionally, it is important to help the members that have less experience.	Moderate	Low	Low
R3	Availability	We all have differnt responsebilities that require our attention, which makes us less flexible. This can lead to tasks not being done in time, the effeciency in the group will decrease, and the preassure from the workload will increase.	Moderate	High	Our group have different timetables, and not every member is located in norway at the beginning of the project. Additionally, some of our members have other responsebilities such as work etc.	Moderate	Made a plan for when to have meetings, and who can meet. The members also have to send a message up front if they can't make a meeting.	Moderate	Moderate	Moderate
R4	Disagreements	We all have different opinions. Misunderstandings and conflicts will therefore arise. This can lead to low motivaton,low productivity and low moral within the team.	High	High	Because our group has so many menbes and everyone has a different opinion, conflics can easily arise.	High	We have made a plan on how to solve problems and make decisions. We also encourage each other to share our ideas and opinions, and try to solve the problem in a way that makes everyone happy.	Moderate	Moderate	Moderate
R5	Changes from the Product owner	When we are creating a product that is specified by a product owner, big changes in requrements can ruin progression and create uncertainty.	Moderate	Moderate	When creating a board game that has clear rules, big changes are unlikely. There is however a chance that small chnages take place. This will halt the progression, but only moderately.	Moderate	To prevent such changes to create dissatisfaction, from reduced progression, it is important to record what requirements the product owner has set forth. So that we can justify the reduction to progress when the requirements are changed.	Low	Moderate	Low
R6	Misunderstandings with the customer	The product owner can change priorities, not understand the project fully, or constantily ask for improvements or updates.	High	Moderate	Because the product is seen differentily from the product owner and the group point of views, there is a high chance that we have different interpretations and expectations in regards to the project.	Moderate	We send a mail every friday to update the product owner on the satus of the product. We also set up meetings throuout the project period in order to get feedback on the product. We also try to conform to the wishes of the product owner	Moderate	Low	Low
R7	A new round with a pandemic (covid 19)	The pandmic came to norway in february of 2020 and lead to multiple schools and locations closing for a duration.	High	Low	With increased measures against contamination, we are more equiped for a new wave of the pandemic to hit. Even though the probaility of closed campuses is low, it is still not zero.	Moderate	Restructuring of workroutines and meetings with both the group as well as the product owner will be necessary.	Moderate	Low	Low
R8	Changes in rules defined by the customer	The game we are trying to make is a workshop version, which is not the final version of the game and ther rules are at risk of being changed often. Adapting to these changes can cost a lot of time and work.	High	Very high	The customer has said that the rules for the game are not set in stone and are very likely to change during the development process.	Very high	We are going to do our best to create a system that is robust and stable in the light of the volatile rules. This means that if some rules change, we should be able to quickly and easily change the existing rules and or add new ones to address these new or changed rules. However, this will not	Very high	Low	Moderate
R9	Future developers	With this game most likely being handed over to other developers at a later time, it is important for us to write clean and structured code. This makes it much easier for furture developers to enter a project, and reduces the risk of them not being able to continiue on our project.	High	Moderate	If they cant develop and maintain the code, our project is overall a waste	Moderate	We have taken different messures troughout the project. The detailes about it is found in our report.	High	Low	Moderate
R10	Family crisis	When someone in your familiy is doing badly, it is hard for a person to focus on the project while also worring about their family member	Moderate	Moderate	This project takes place over a semester, and we are 6 members. It is therfore moderatily probable that a family member of one of the students gets grievously ill or hurt.	Moderate	This is out of our control and so there is no way of prevernting hurt or illness, but we have multiple students on multiple parts of the projects, so we are not hampered by one student working less for a period.	Moderate	Moderate	Moderate

5.5 Work Hours

Week	Martin	Mathias	Thomas	Tim	Torbjørn	Runar
Average	16,59	15,10	15,56	18,92	17,92	9,73
5	19,42	10,00	6,50	14,17	7,92	6,00
6	19,35	15,98	8,83	17,17	21,75	8,42
7	2,08	13,90	10,00	15,17	15,75	10,27
8	15,00	7,00	19,50	18,42	14,25	8,50
9	16,42	7,00	17,75	22,73	19,00	16,00
10	19,25	21,23	14,50	21,73	15,25	12,00
11	17,08	13,08	9,17	15,75	11,00	3,75
12	14,00	6,00	16,50	18,57	4,00	0
13,14	14,92	0	11,08	8,58	10,25	0
15	15,67	11,00	8,50	10,00	13,75	18,75
16	15,73	3,00	6,33	11,67	18,17	9,00
17	20,58	11,50	9,00	23,08	24,42	10,00
18	25,82	57,65	55,50	41,17	47,83	21,50
19	17,00	33,98	24,67	26,70	27,50	12,00
Total hours	232,32	211,33	217,83	264,90	250,83	136,18