

Amund Helgestad
Kristian Aakervik Rønning
Thomas Vincent Lien

A Virtual Reality Clinical Education Tool for Medical Students in Norway

Bachelor's thesis in Bachelor of Engineering in Computer Science
Supervisor: Ivar Farup
May 2022

Amund Helgestad
Kristian Aakervik Rønning
Thomas Vincent Lien

A Virtual Reality Clinical Education Tool for Medical Students in Norway

Bachelor's thesis in Bachelor of Engineering in Computer Science
Supervisor: Ivar Farup
May 2022

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



NTNU

Kunnskap for en bedre verden

Abstract

Medical Education in Norway is currently facing a challenge where they need to enroll more students but have limited capacity for practical clinical training.

Zimmer Digital is a start-up company with the vision to create a digital solution for said problem, by incorporating VR simulation in medical education as a supplement or addition to real-life practical training. As the medical field is subject to rapid innovation, a solution needs to be flexible enough to keep up with future advances.

The project investigates possibilities for and partially implements a scalable software framework. It aims to allow users to easily create new playable simulations on their own, which they then can play through using VR. Such a solution is flexible and can adapt to future advances in the medical field and education, as outdated simulations can easily be edited or replaced by entirely new simulations.

The proposed solution gives Zimmer Digital a framework, which they can further build upon to reach their goal.

Sammendrag

Medisinsk utdanning i Norge står for tiden ovenfor en utfordring; det er et ønske å kunne øke antallet studentplasser, men muligheten for dette er begrenset av tilbudet til å gi praktisk klinikkundervisning.

Zimmer Digital er en start-up bedrift med en visjon om å lage en digital løsning på dette problemet, ved å integrere VR simulering i den medisinske utdanningen som erstatning for manglende – eller et tillegg til allerede eksisterende – praktisk opplæring. Siden utvikling og innovasjon innenfor medisin skjer svært raskt, må en løsning være fleksibel nok til at den kan holde følge med fremtidige endringer.

Prosjektet søker etter muligheter for et slikt skalerbart rammeverk og delvis implementerer en slik programvareløsning. Denne har som mål å la brukere enkelt lage nye spillbare simuleringer på egenhånd og som dem deretter kan spille igjennom med VR. Siden løsningen støtter muligheten til å endre allerede eksisterende simuleringer, eller lage nye, vil det være enklere å holde følge med eventuelle fremtidige endringer.

Den vedlagte løsningen gir Zimmer Digital et programvare-rammeverk som forhåpentligvis kan hjelpe dem med å nå deres fremtidige mål.

Preface

The project report is written by Amund Helgestad, Kristian Aakervik Rønning and Thomas Vincent Lien from the institution of information technology at NTNU in Gjøvik.

We would like to thank those who have supported us throughout the project and with our thesis. Firstly, we would like to thank our supervisor, Ivar Farup, for his excellent guidance and support during this process. Through weekly meetings and through other forms of online communication, it was always reassuring and motivating to chat with Ivar. We would also like to thank the people at Zimmer Digital, Mari Zimmermann, Mathilde Volle Fiborg and Jakob Cyvin, who gave us this project opportunity and who supported us during this process through both weekly meetings and online chat. Additionally, we would like to thank the academic community at NTNU in Gjøvik for the 3 years we have spent here and whom we have learned valuable information from.

At the end, we would like to thank ourselves for good collaboration throughout the project period, despite the fact that this is the first time we have worked together. It has all been a challenging yet also exciting and fulfilling process.

Contents

Abstract	iii
Sammendrag	iv
Preface	v
Contents	vi
Figures	x
Tables	xii
Code Listings	xiii
Acronyms	xiv
Glossary	xv
1 Introduction	1
1.1 Scope	1
1.1.1 Subject Area	2
1.1.2 Delimitations	2
1.1.3 Project Description	2
1.2 Repository	3
1.3 Target Users	3
1.3.1 Application Users	3
1.3.2 Report Readers	3
1.3.3 Future Developers	3
1.4 Academic Background	4
1.5 Project Goals and Objectives	4
1.6 Learning Objectives	4
1.7 Constraints	5
1.7.1 Legal Constraints	5
1.7.2 Technological Constraints	5
1.7.3 Time Constraints	5
1.8 Project Roles	5
1.8.1 Group	5
1.8.2 Other	6
1.9 Report Structure	6
2 Background Theory and Technology	8
2.1 Education of Medical Students	9
2.2 Technologies	9

- 2.2.1 Unity Game Engine 9
- 2.2.2 JSON 9
- 2.2.3 Virtual Reality – Oculus Quest 9
- 2.2.4 MyBox 10
- 3 Requirements 11**
 - 3.1 Functional Requirements 11
 - 3.1.1 Use Cases 12
 - 3.1.2 High level use cases 13
 - 3.1.3 Low level use case 14
 - 3.2 Technical Requirements 16
 - 3.2.1 Complexity of the application 16
 - 3.2.2 User Interface 16
 - 3.2.3 VR compatibility 16
 - 3.2.4 Game Engine 16
 - 3.3 Software Requirements 16
 - 3.3.1 Case Data 16
 - 3.3.2 From Case Designer to Gameplay 16
 - 3.3.3 C# Coding Standards 17
 - 3.3.4 Language 17
 - 3.3.5 Documentation 17
- 4 Design 18**
 - 4.1 Graphical Design 18
 - 4.1.1 Wireframes / UI 18
 - 4.2 Technical Design 20
 - 4.2.1 Various Game Mechanics 20
 - 4.2.2 System Architecture 22
 - 4.2.3 Unity Project File Structure 23
 - 4.2.4 JSON Structure 23
- 5 Development Process 24**
 - 5.1 Choice of Development Process 24
 - 5.1.1 Considerations 24
 - 5.1.2 Waterfall Method 25
 - 5.1.3 Scrum and Kanban 25
 - 5.2 Use of Development Process 25
 - 5.2.1 Kanban Board 25
 - 5.2.2 Iterations 26
 - 5.2.3 Internal Meeting Structure 26
 - 5.2.4 Client Meetings 27
 - 5.2.5 Planning Meetings 27
 - 5.2.6 Review Meetings 27
 - 5.2.7 Supervisor Meetings 27
 - 5.2.8 Gantt Chart 28
 - 5.2.9 Time Tracking 28
 - 5.2.10 Milestones 29

5.2.11 Milestone completion chart	31
6 Implementation	32
6.1 Tools	32
6.1.1 Unity Game Engine	32
6.1.2 Visual Studio	32
6.2 Title Screen	33
6.3 Case Editor	33
6.3.1 Changes during development	36
6.4 Gameplay	37
6.4.1 Controls and Movement	37
6.4.2 Environment	37
6.4.3 Interactions	38
6.4.4 GameManager	44
6.5 JSON Parsing	46
7 Testing and Quality Assurance	49
7.1 Code Quality	49
7.1.1 Conventions	49
7.1.2 Documentation	50
7.2 Testing	50
7.2.1 Unit & Integration Testing	50
7.2.2 Manual Testing	53
7.2.3 User Testing	53
7.3 Collaboration	54
7.3.1 Version Control	54
7.3.2 Protected branches	54
7.3.3 Reviews	54
7.4 Client Feedback	55
8 Deployment	56
8.1 Unity Project Setup	56
8.2 Oculus Development Setup	57
8.3 Standalone VR Game for Oculus	57
8.4 Case Editor for Web	57
8.5 User Manual	57
8.5.1 Further Development	58
9 Discussion	63
9.1 Evaluation of Objectives	63
9.1.1 Learning Objectives	63
9.1.2 Impact Objectives	63
9.1.3 Outcome Objectives	64
9.2 Requirements	64
9.3 Development Method and Process	65
9.3.1 Time Use	65
9.3.2 Internal Meetings	66
9.3.3 Testing	66

- 9.4 Application Design 66
 - 9.4.1 Minimum Viable Product 66
- 9.5 Further Development 67
 - 9.5.1 Databases and File Sharing 67
 - 9.5.2 Order of Actions 67
- 10 Conclusion 70**
- Bibliography 71**
- A Project Description 74**
- B Project Agreement 76**
- C Project Plan 83**
- D User Test 101**
- E The Problem which the Concept Solves 105**
- F Iteration Documentation 108**
- G GitLab Repository Rules 115**
- H Time Tracking 118**

Figures

3.1	Use case diagram.	12
4.1	Case Designer Wireframe of a Phase where actions (which the player can take) and milestones (which must be met) can be set.	19
4.2	Case Designer illustration that consists of 4 wireframes. Chronological order is illustrated by the red arrows: left to right, starting at the top.	19
4.3	A high-order flowchart illustrating the general flows of the system and what actions the user can take.	20
4.4	Case Editor Object Diagram.	22
4.5	Case Editor Class Diagram.	22
4.6	File structure	23
5.1	An example of the Kanban board.	26
5.2	Gantt chart.	28
6.1	The Main Menu.	33
6.2	The clipboard is where information the player receives during play is automatically registered and displayed.	34
6.3	The <i>informasjon</i> tab (meaning <i>information</i> in Norwegian).	34
6.4	The <i>undersøkelser</i> tab (meaning <i>examinations</i> or <i>actions</i> in Norwegian).	35
6.5	Categories of the Case Editor – Figure showing three images side-by-side of parts of the final design. Categories are color-coded to clearer convey parent-child relations.	35
6.6	A screenshot from a development video of a tree-graph system. This solution was later replaced with a list-category system instead. . . .	36
6.7	XR Grab Interactable tool setup.	39
6.8	Screenshot from gameplay showing the progress bar as an action is being performed using a stethoscope.	41
6.9	The anamnesis menu.	42
6.10	The anamnesis category question list.	43
6.11	The referrals UI.	43
6.12	The referrals UI.	44

6.13	A class diagram illustrating the data structure used for creating the JSON files containing the data used in both gameplay and Case Designer.	47
7.1	The Test Runner window showing a number of passed tests.	50
7.2	The tests are located in the <i>Tests</i> -folder under <i>Assets</i>	51
8.1	Tool component configuration.	59
9.1	Illustration of how one may connect the application to a storage or database instead of using the File Explorer as currently done by using an application view.	67
9.2	The <i>Timeline</i> -approach to sort actions in the desired order. As illustrated by the diagram, actions are first displayed in a view or scrollbar represented by the <i>unordered actions</i> window, then they can be dragged into a group. The order of each group matter, however, the order of each action inside a group may not.	68
F.1	Iteration 1	109
F.2	Iteration 2	109
F.3	Iteration 3	110
F.4	Iteration 4	110
F.5	Iteration 5	111
F.6	Iteration 6	111
F.7	Iteration 7	112
F.8	Iteration 8	112
F.9	Iteration 9	113
F.10	Iteration 10	113
F.11	Iteration 11	114
F.12	Iteration 12	114

Tables

3.1	High level use cases.	13
3.2	Low level use case: create case.	14
3.3	Low level use case: play case.	15
4.1	Initial result ideas	21
5.1	Milestones and completion iterations.	31
8.1	New result type examples.	61

Code Listings

6.1	The Action class data fields.	38
6.2	The Result class.	38
6.3	The OnSelect method in Tool.	40
6.4	The OnTriggerStay method in Tool.	40
6.5	The AnamneseAction class.	41
6.6	Example of screen fade using coroutines.	45
6.7	An example of a JSON file containing the case data.	46
6.8	Shows the variable <i>FullAppNameVersion</i> . Together with <i>AppName</i> and <i>AppVersion</i> , it creates the field <i>appVersion</i> in the output JSON (see 6.7.	48
7.1	Example of an Edit Mode test.	51
7.2	Example of a Play Mode test.	52
7.3	Example of a UnitySetUp. This is executed before every test in the same script.	52
7.4	Example of a OneTimeSetup. This is only executed once for the given test script, before any of its tests.	53
8.1	Tool script showing ToolID enum.	60
8.2	The Result class.	61
8.3	Node selection in ListNode.cs.	62

Acronyms

3D Three-dimensional. 2, 3, 5, 39, 58, 60

AR Augmented Reality. 9

GDPR General Data Protection Regulation. 5

GP General Practitioner. 29

IDE Integrated development environment. 32

JSON JavaScript Object Notation. xi, xiii, 9, 13, 16, 23, 30, 32, 33, 42, 44, 46–48, 56, 63

LTS Long-term support. 32

NOK Norwegian Kroner (Currency). 8

NPE Norsk Pasientskade Erstatning. 8

NTNU Norwegian University of Science and Technology. 4, 54

OS Operating System. xv

UI User Interface. 2–4, 40, 42

UX User Experience. 2, 4

VR Virtual Reality. iii, iv, 2, 4, 5, 9, 11, 13, 15, 16, 20, 32, 37, 53, 54, 57, 63, 64

WIP Work-in-Progress. 26

XML Extensible Markup Language. 17

XR Extended Reality. 9

Glossary

- accessor method** A method for getting a variable. Also known as a *'getter'*. 45
- anamnesis** In a medical consultation, anamnesis is the process of asking the patient questions to retrieve information about the patient's medical condition. xvi, 14–16
- C++** C++ is a popular high-level, general-purpose programming language created by Danish computer scientist Bjarne Stroustrup [1]. xv, 4
- C#** C# is a programming language developed by Microsoft. It is based upon C++ and Java [2]. xv, 4, 9, 17, 49, 50
- camel case** Camel case is a type of naming convention where variables that contains multiple words have no spaces but instead have uppercase letters on each subsequent word's initial letter. Camel case variables can start with both lower and uppercase letters; the latter is often called Pascal case. xvi, 50
- case** A case or *kasus*, in the context of this project, refers to a playable game level. xv, 11, 16, 20, 21
- Case Designer** Also known as Case Editor, is the part of this project that takes on the design and creation of Cases. xi, 2, 12, 16, 18, 21–23, 47, 61, 68, 70
- Discord** Discord is a communication platform that allows for both voice, video and text-based communication. 66
- enum** Enums are a datatype that consists of a set of elements or enumerators that behave as constants. xiii, 58–60
- File Explorer** The OS' default program for viewing and selecting files from the computer. xi, 56, 67
- GameObject** In the Unity Engine, a GameObject is the base class for entities in the game world [3]. All classes that is attached to such GameObjects inherit from MonoBehaviour. xv, xvi, 29, 30, 39, 51, 58, 60, 61

Java Java is a programming language developed by James Gosling and other developers at Microsystems [4]. xv, xvi, 4

Least Privilege Least privilege is a common security principle that aims to restrict access right for users and processes in order to protect the system or parts of the system against both intentional and unintentional harm [5]. 54

MonoBehaviour A class in the Unity Engine. Scripts that can be attached to GameObjects must inherit from MonoBehaviour. xv, 51

MVP A *Minimum Viable Product* is an early version with only the bare-minimum of features and functionality; it serves mostly as a proof-of-concept for a future product. Despite its limited features, it should be of a high quality without bugs and errors. 23, 29, 64, 66, 70

pascal case Pascal case is always consistent with Camel case, but Camel case is not always consistent with pascal case. If Camel case starts with a lowercase letter, it is not pascal case. Pascal case can be considered a variation of Camel case. xv, 50

Patient Journal Inside the game, the patient journal looks like a blue clipboard, which is used to display prior known information about the patient. Information acquired through examinations and anamnesis is also noted here. 15

phase In the context of this project, it was a word to describe certain time intervals during play. x, 18, 19, 21, 68

scalable If a system is scalable, it can be added upon without the need of a total restructuring of the whole system. Scalable solutions are often desired. 21, 66

Separation of Duties Separation of Duties is a common security principle that aims to prevent potential harm a single user can do to the system by restricting users of enough privileges so that they cannot misuse the system on their own [6]. According to this principle, one should divide critical functions among different users. For example: one user should not have the privilege to both *create* a merge request and *approve* it, as a single user with malicious intent can then easily destroy or audit the repository. 54

singleton Design pattern used to ensure that only one instance of the object is created. 58

spaghetti-code Spaghetti-code is a slang-term that refers to a tangled web of unstructured code where control of the code jumps all over the place and is generally hard to follow [7]. It is something that should always be avoided. 49

- string** A string is a data type used in programming languages to represent text. In C#, it is represented by the build-in type 'string'. 23
- Toggl** A tool the group used for time-tracking of work-hours. 65
- Unity** An all purpose game engine that support 2D and 3D graphics, and scripting through C#. 2, 55, 66
- Unity Editor** The part of the Unity Game Engine that is outside of Play Mode. 56, 64
- Universal Render Pipeline** A scriptable render pipeline optimized for VR. 57
- wireframe** From Balsamiq Wireframing Academy: a wireframe is 'a schematic or blueprint that is useful for helping you, your programmers and designers think and communicate about the structures of the software or website you're building.' ([8]). x, 18, 19

Chapter 1

Introduction

Medical education requires a large amount of practical training and is currently almost exclusively in-person. This is particularly resource demanding. Whilst more students are admitted to universities, the hospitals still have a limited capacity for practical training. This causes group sizes to increase and amount of training per student to decrease [9].

Zimmer Digital is a start-up company working on creating a product as a substitute to in-person practical training. The product is a virtual simulation game, where medical students can practice in virtual training scenarios.

They are currently in the early phases of development and have made a click-and-point prototype. The patient cases in the prototype are hardcoded, making it very demanding to implement additional cases. Having a large set of both similar and diverse cases is vital to provide a good representation for all the different scenarios medicine students will encounter in their future work as professionals.

The client has tasked us to take their current point-and-click model into a virtual reality environment, as well as increasing the number of playable cases.

1.1 Scope

The task given by the client were originally quite open as their plan was to allow us to choose and focus on a smaller part of their larger goal. Some parts of this larger goal were to implement a separate webpage to run the application in, a database and the software itself. Additionally, they wanted the software to use said database, and allow users to be ranked on their performance, get access to in-game medical documentation and educational information, allow users to connect to a network, implement in-application communication between users and more.

As such, it was paramount for the integrity of the project to be able to find reasonable delimitations in order to make the project plausible and achievable within

the given timeframe. However, these limitations were, according to the client, entirely up to the group to decide. The exact choices the group made when defining the scope is described in this section.

1.1.1 Subject Area

The end-product is meant to serve as a supplementary tool to be used in the practical training of medicine students. The use of virtual training scenarios should help alleviate the limited capacity of real practical training, whilst giving students more training time overall.

1.1.2 Delimitations

The application developed will be limited to two primary components; creation and editing of training scenarios, and a playable VR game.

This project will not include database integration nor will it have handling of user data or user authentication. Files will be saved locally, but in a JSON format to make future implementation easier.

Development will not be focused on the 3D graphics beyond using assets provided by the client. Placeholder graphics will therefore be used where none are provided.

UI is made to provide the best user experience for the target audience. UX and functionality are prioritized above visual style and design.

1.1.3 Project Description

The product will be a VR simulation game for medical training. Despite the client having an already existing point-and-click model, the application is to be developed as a new Unity Project. The product is split into two main components, a *case editor* and a *VR game*.

Case Editor

The *Case Editor*, also known as the *Case Designer*, will allow users to create and edit medical training cases that can be played in the game. It will allow users to define questions and answers to ask the patient. All actions, tools and tests that can be performed has results that can be changed here. The editor will create cases from a template. Initially the application will consist of one environment, being general practitioner, but will be expandable to cover several environments and scenarios using the framework developed.

Virtual Reality Game

The game will consist of a 3D environment imitating a doctor's office. The player will take on the role of a general practitioner treating one patient, also present in

the game. The player will be able to ask questions and perform actions and tests to determine what is affecting the patient. Asking questions will be done through UI elements. Tools will be 3D objects that the player can pick up and use to examine the patient.

At the end of the game the player is given an overview and feedback of how they performed. This will be used to help in the training of medical students.

1.2 Repository

The repository can be found at:

<https://git.gvk.idi.ntnu.no/ThomasVL/bachelorzimmerdigital>.

1.3 Target Users

1.3.1 Application Users

The application is designed to have two separate user groups; medical students, using it for practical clinical training; and medical professionals, for creating new training scenarios.

Medical Students

This is the main end-user of the application. They will use the application in an educational environment and are not expected to have any specific technical experience. It should be as easy as equipping an Oculus Quest and start 'playing'.

Medical Professionals

This user group will consist of lecturers, professors and other medical professionals. They will be using the application for the creation of new training scenarios and editing of existing ones. This group is expected to be non-technical end-users with high medical knowledge.

1.3.2 Report Readers

The report readers are assumed to be third year computer science students, or of equivalent or higher knowledge in software engineering. Primarily it is aimed at the project supervisor and those grading the thesis.

1.3.3 Future Developers

This is the developers that will continue to work on, and improve the product after the product is given to the client.

1.4 Academic Background

All three group members are in the final year of a bachelor in engineering and computer science at NTNU. The group possesses a similar background in topics covered in mandatory courses. This covers a wide range of skills including programming, software security, academic writing, databases, cloud technologies, development processes, agile development and more. Due to differing courses of education some minor differences appear where, for example, some group members have more experience in C++ instead of Java or vice versa.

Elective courses have given group members different experiences in certain areas. Two group members have taken a game development course, giving experience in both Unity and C#. Two group members have taken a course in User Centered Design covering UI/UX, design and development methods.

1.5 Project Goals and Objectives

The goal of the project is to create an application that will make it easier and less time-consuming to create and implement new training scenarios by removing the need for programming knowledge when creating or editing existing cases.

1.6 Learning Objectives

- Planning and executing an engineering project.
- Documenting and reporting an engineering project.
- Work methodically by applying engineering methods.
- Agile development with the Scrumban development methodology.
- Learn in-depth Unity and C# game development.
- Learn VR development within Unity.

Impact Objectives

- Users will be able to easily create and implement new cases, removing the need for developers to hard-code each new case.
- Reduce complexity for making cases.
- Drastically reduce time spent per case.
- Make case creation accessible to those with little to no programming experience.
- Make saving and exporting cases easy.

Outcome Objectives

- Deliver a product which the client can easily expand or implement into their future product.
- Create a framework and an easy-to-use editor to edit and create new cases.

- Create a playable Virtual Reality experience.

1.7 Constraints

1.7.1 Legal Constraints

- The product must be GDPR compliant.
- The product must abide by Norwegian law.
- Copyright is owned by the project group.
- Ownership will belong to the client, Zimmer Digital.
- The project group are given non-commercial right of use.

1.7.2 Technological Constraints

- The product must be developed in the Unity game engine.
- The product must be a 3D VR game.
- Support for Oculus Quest 2.

1.7.3 Time Constraints

- The Project Plan must be finalized and delivered by January 31st.
- Standard agreement for academic collaboration must be signed and delivered by February 1st.
- The Product and Report must be finalized and delivered by May 20th.

1.8 Project Roles

1.8.1 Group

Group Leader – Amund Helgestad

The group leader should always have a general overview of all ongoing activities. They are responsible for leading the project in the correct direction according to what has been agreed to. To achieve this, the project leader may need to delegate tasks between the group members, though everyone should be active in this process and is therefore strongly encouraged to come up with ideas of their own.

The project leader should lead meetings and ensure all items on the meeting agenda have been addressed and discussed.

Chief of Documentation – Kristian Aakervik Rønning

This member is responsible for making sure that documentation is organized and stored in appropriate locations. They are expected to oversee code documentation, pre-development, and post-development documentation. This also includes

organizing and overseeing the project report, along with ensuring everything is documented and included in the report.

This member is also responsible for creating meeting minutes after each meeting, as well as conveying information afterwards and following up on it.

Chief of Communications – Thomas Vincent Lien

This member is our team representative that will be responsible for communication *from* and *to* the group. Specifically, they are responsible for the communication between the client and the group, as well as the supervisor and the group. They are also responsible for sending meeting agenda for meetings hosted by the group.

This member is also responsible for checking through documents, ensuring that everything is working before delivery. This member is responsible for sending the deliveries.

1.8.2 Other

Client – Zimmer Digital

Zimmer Digital is the client behind the project. They will provide project requirements and specify what the end product should be, as well as guiding the development through weekly meetings and feedback. They will also be consulted for medical knowledge and specificities outside the groups field of study.

Supervisor – Ivar Farup

The supervisor will serve as a supportive role; providing feedback, recommendations and assistance for the project, specifically the report and the development process.

1.9 Report Structure

The report is structured to be read sequentially from the first to the last chapter. This is an overview of each chapter content.

Introduction

Introduces the project, its scope and purpose, as well as the group members and background.

Background Theory and Technology

Explains background theory and technology that is used throughout the report.

Requirements

Defines the functional, technical and software requirements of the project. Further expands on the goals and objectives in Introduction.

Design

Shows the design of the components throughout the project, including early designs up to the final design. This chapter gives an overview of the design and its evolution over time, as well as the basis which the implementation builds upon.

Development Process

Describes the development process in its entirety, from choosing a development methodology to each aspect of the execution.

Implementation

Gives a detailed and technical description of how components were implemented, what tools was used and how they work.

Testing and Quality Assurance

Details the testing and quality assurance efforts made, how they were performed, as well as methods and tool that were used.

Deployment

Instructions for setting up the project, building the project and further development on the codebase.

Discussion

Analysis of the project workflow and results.

Conclusion

Final remarks on the project.

Chapter 2

Background Theory and Technology

Virtual reality is redefining what is possible in the healthcare industry. Patients are already benefiting from advanced and more precise surgeries, simulations for mental illness treatments and many other improvements thanks to VR solutions. VR helps make the work of healthcare professionals more effective and the lives of patients easier. (Vardomatski [10])

Medical treatment errors have great consequences for both patient and society. Factors like lack of medical professionals and resources have been blamed, as well as poor communication. Norwegian System of Patient Injury Compensation (NPE) had a record-high payment in 2020 of 1.1 billion NOK to patients that had suffered treatment errors [11]. The consequences of poor choices done by unprepared personnel can end in personal ruin for those affected, great social-economical losses for society and strengthen the public fear for said errors.

The fact that medical innovation is developing exponentially, while the education system is stuck in outdated pedagogical methods, is only short of paradoxical (Zimmer Digital, 2022, Appendix E) [12][13]. Furthermore, patient contact during medical training have been reduced as a consequence of each patient being fewer days at the hospital, more critically ill patients and more students per patient that needs training [14]. According to the client, substitute simulation tools is either expensive, unavailable or both, and there is a lack of teachers who can use said tools and guide the learning. Over the past 2 years especially, the whole world has realized the importance and necessity of good clinical services and educational methods that are both resource efficient and effective (Zimmer Digital, 2022, Appendix E).

2.1 Education of Medical Students

Virtual reality can be used as a form of healthcare simulation, which defined by the Agency for Healthcare Research and Quality, is ‘a technique that creates a situation or environment to allow persons to experience a representation of the real health care event for the purpose of practice, learning, evaluation, testing, or to gain understanding of systems and human actions.’ (Lopreiato and Lioce [15]) Generally, healthcare simulations are used to ‘improve the safety, effectiveness and efficiency of healthcare services.’ (Vardomatski [10])

Furthermore, according to the Agency for Healthcare Research and Quality, healthcare simulation is an effective instructional tool and supplement for working directly with patients. In this regard, VR simulations is particular good at promoting ‘the acquisition of a full range of cognitive, technical, and behavioral skills.’ (Lopreiato [16])

2.2 Technologies

2.2.1 Unity Game Engine

Unity Game Engine, or simply Unity, is a cross-platform game engine developed by Unity Technologies. This engine can be used to create both 2-dimensional and 3-dimensional games as well as interactive simulations and other experiences. The engine has been adopted by other industries outside of video gaming, such as engineering, architecture, construction, film [17] and, in our case, medical simulation training.

2.2.2 JSON

JSON, or JavaScript Object Notation, is in short ‘a text format for storing and transporting data’ [18]. JSON is in reality just simple plain-text which is written in JavaScript object notation. It is programming language independent, meaning that it is not restricted by any specific programming language like C# or Java – it may even be used to transfer data between two otherwise separate languages.

2.2.3 Virtual Reality – Oculus Quest

Extended Realities (XR) is an umbrella term for all the immersive technologies [19]. This includes technologies like Virtual Reality (VR), Augmented Reality (AR), Mixed Reality (MR) and those immersive technologies that are yet to be developed [19].

This project is developed for VR in Oculus Quest 2; by using VR, we can fully immerse the player in a simulated digital environment [19]. Special headsets, commonly known as VR Headsets, must be put on by the player before they can

play. Such a headset has a screen that covers the user's eyes that allows them to see the digital world as if they were themselves standing in it.

2.2.4 MyBox

'It is MyBox. Now it's yours too' (Deadcows [20], owner of MyBox)

MyBox is a set of tools, features and extensions for Unity [20], which comes with a permissive MIT License [21]. In this project, it has only been used for its 'ConditionalField' attribute, which allows for further customization of the open fields in Unity's inspector; this is purely for aesthetics and only visible for the developers, though it may improve clarity and help prevent cluttering during development. However, this is an optional feature and can be removed without extensive refactoring of code.

Chapter 3

Requirements

The client's requirements focused solely on the concept of their final product, hence most of the design and implementation was left for the team to decide. Due to this, there were several meetings with the client, in order to prioritize which aspects should be focused on. The requirements were used to narrow down the scope, and to define specific expectations for the product.

3.1 Functional Requirements

Initially, there was no functional requirements given by the client beyond a playable game. The functional requirements were formed by reviewing the clients demo application and case drawing, and then presenting different solutions early in the process to the client for validation and feedback. Through this process, the project group came up with the following functional requirements:

1. User should be able to create new custom game levels, also known as cases, by editing templates.
2. User should be able to play said cases.
3. Gameplay must support VR.
4. Gameplay must imitate real-life scenarios of a medical cases.
5. Cases will be saved locally, however, it should not require much refactoring for it to work with an online database in the future.

All features of the application can be available for all users, both medical and IT professionals, as well as students. However, it is not required that students nor IT should have the medical knowledge required to create a realistic medical case, but the features can be open for them to try out if they want to. Furthermore, they should all be able to import a Case from file to edit or play and save a Case locally.

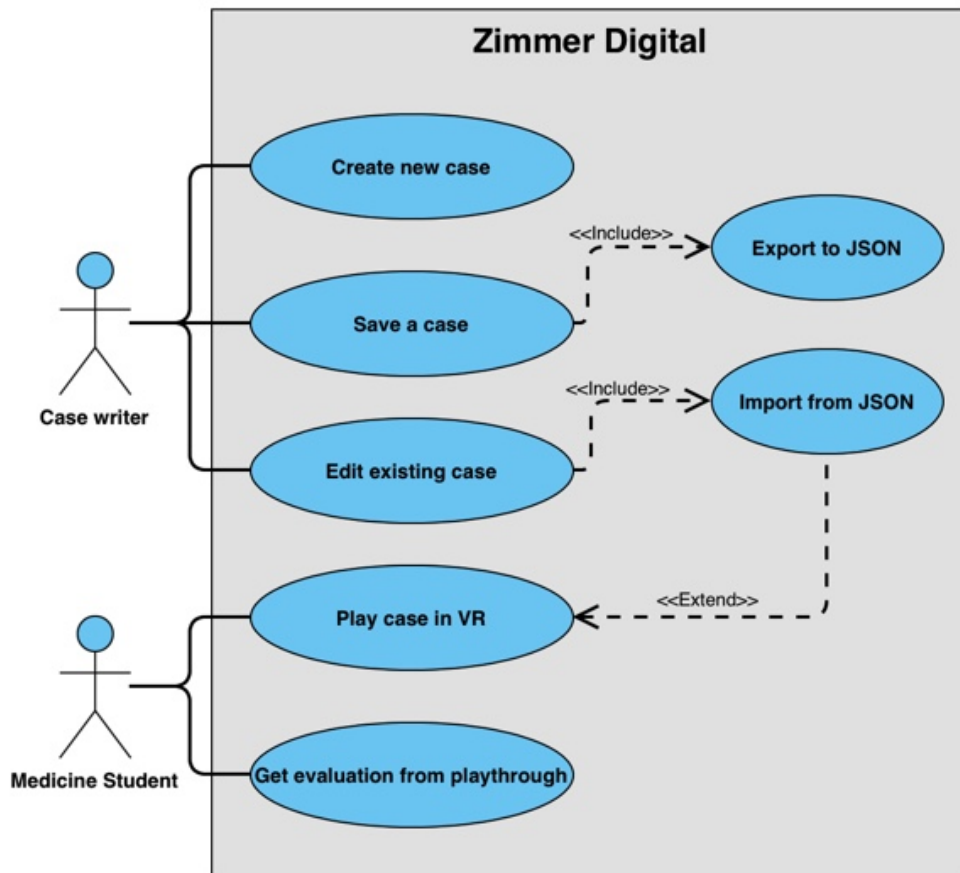


Figure 3.1: Use case diagram.

3.1.1 Use Cases

After gaining an understanding of the client's expectations, the group created an use case diagram to define what each individual party requires from the product.

The actors defined in Figure 3.1 have following descriptions:

- **Case Writer:** User that writes a case using the Case Designer. This can be Medical Professionals writing cases for their students or the client's developers.
- **Medicine Student:** Students enrolled in a university level medicine course. Further described in Section 1.3.1.

3.1.2 High level use cases

Table 3.1 shows use case descriptions for the use case diagram in Figure 3.1.

Use case	Create a new case
Actor	Case writer
Purpose	Create a learning experience for the student
Description	Create a new playable case inside the editor application
Use case	Save a case
Actor	Case writer
Purpose	Store the case in a distributive form
Description	Export the defined case from the editor into a JSON file.
Use case	Edit existing case
Actor	Case writer
Purpose	Make improvements or corrections to an existing case
Description	Fill in the application with data from the old case and add changes as if it was a new case.
Use case	Play case in VR
Actor	Medicine Student
Purpose	Practice medical skills in a safe environment
Description	Play through a case in a virtual environment.
Use case	Get evaluation from playthrough
Actor	Medicine Student
Purpose	Receive feedback on how the patient was treated
Description	Student receives a comparison of what has been done and what should have been done

Table 3.1: High level use cases.

3.1.3 Low level use case

Table 3.2 and Table 3.3 shows detailed flows for case creation and gameplay respectively.

Use case	Create a new case
Actor	Case writer
Purpose	Create a learning experience for the student
Preconditions	Actor has loaded the application
Post conditions	Actor saves the case
Main flow	<ol style="list-style-type: none"> 1. Actor enters the case designer from the main menu 2. Actor navigates to the information window 3. Actor writes the information about the patient that will be shown at the start of the game 4. Actor navigates to the examination tab. 5. Actor clicks on the anamnesis section. 6. Actor creates categories of questions that will be available in the game. 7. Actor clicks on each categories, opening the editor and creates questions that will appear when the player clicks the categories in the game. 8. Actor clicks on the clinical examination section. 9. Actor creates categories of examinations, organizing which examinations are linked to each purpose. 10. Actor clicks on the categories, and loads up the examination editor. 11. Actor defines examinations, which tool is used, where the tool is used, examination time, and the results of the examination.
Alternate flow	<ol style="list-style-type: none"> 1. Actor enters the case designer from the main menu. 2. Actor loads one of the template cases available. 3. Actor resumes from step 2 in the main flow.

Table 3.2: Low level use case: create case.

Use case	Play case in VR
Actor	Medicine student
Purpose	Practice medical skills in a safe environment
Preconditions	Actor is equipped with working Oculus Quest 2 VR head-set and is connected to the application
Main flow	<ol style="list-style-type: none"> 1. Actor enters the game from the main menu. 2. Actor chooses a case, either available in the game or from a locally stored file. 3. Game is created from the file and the application loads up the game. 4. Actor picks up the Patient Journal and reads information known about the patient prior to the visit. 5. Actor puts down the Patient Journal 6. Actor moves to the patient and opens up the Anamnesis interface. 7. Actor asks questions to the patient through clicking the question text boxes and receives answers from the patient. 8. Actor picks up a tool from the table. 9. Actor moves the tool to highlighted areas on the patient. A progress bar appears on the screen, and the actor waits until the bar completes. 10. A notification appears on the actors screen, notifying the user that an examination has been completed. 11. Actor puts down the tool, picks up the Patient Journal and checks if the examination gave any important information. 12. Actor continues to conduct examinations until they are confident on an diagnosis. 13. Actor moves to the PC available in the game and creates a diagnosis for the patient. 14. Actor opens the main menu and clicks the option to end the consultation.

Table 3.3: Low level use case: play case.

3.2 Technical Requirements

3.2.1 Complexity of the application

It must be possible to create a case without any prior programming knowledge or experience with the software. This means that the user should not be exposed to low level behaviours, and that this should be displayed in a way that is intuitive and easy to understand for the user. On the other hand, the Case Designer should still make it possible to create a realistic playthrough. This means that the Case Designer must find intelligent ways of displaying complex structures to the user.

Interactions within the game should be similar to what the user would expect in a real consultation. Tools should be used by moving them to areas in which the examinations are taking place. Interactions with the patient during the anamnesis should take place over the head of the patient to indicate that the patient is speaking.

3.2.2 User Interface

Interfaces must be labeled with correct and accurate terminology. They should be easy to understand, and easy to use. For instance, it should be clear if a text box can be written in, and what should be written inside it.

3.2.3 VR compatibility

The game must be developed to support the VR headset: Oculus Quest 2. This is a requirement set by the client.

3.2.4 Game Engine

The product must be developed in the Unity Game Engine on the version: Unity3D version 2020.3.26f. This is a requirement set by the client.

3.3 Software Requirements

3.3.1 Case Data

Cases must be stored in a sensible format that is readable and store-able. JSON file format will be used to save case data.

3.3.2 From Case Designer to Gameplay

A Case created in the Case Designer must be able to be exported into JSON file format. Gameplay, with the same defined case data must then be possible to load using the same JSON file.

3.3.3 C# Coding Standards

All code other than case data will be written in C#, this code should follow the C# Coding Conventions [22].

3.3.4 Language

Although client wanted the application to be developed in Norwegian, all the logic and code is written in English. The reason for this division is that the project group preferred to write code and documentation in English; the reason being that the core system itself, being the Unity Engine or the C# programming language, is in English, and the group thought it to be confusing to combine both English and Norwegian in the same parts of the system.

3.3.5 Documentation

Code should be well documented with a focus on readability and clarity so that it is easily understood, not only for the current project members, but also for other developers that will carry on developing the product after the group is done.

This means that all classes and most methods will require documentation. Examples of methods that would not require documentation are mutator and accessor methods. Documentation should follow the recommended documentation standards and Microsoft's recommendations for C# using XML elements.

As the product will be composed of many different components, it is also required that a written manual is created. This manual will explain how each different component are linked together and what their responsibilities are.

Chapter 4

Design

4.1 Graphical Design

It was highly important for the client that the application was as intuitive and simple to use as possible, and that there should be no learning curve required before using it. However, the client did not specify a style beyond this. This favoured a style with high visibility, clarity, and simplicity, which most of the complexity should be hidden behind. With the client's requirements in mind, the project group chose to create all of the user interface using mostly basic shapes and light, *easy-on-the-eyes*, faded colours with few large buttons to prevent clutter.

4.1.1 Wireframes / UI

During the first weeks of the planning period, the group developed a wireframe to show the first pitch on the design of the Case Designer. The wireframes can be seen in Figure 4.1 and Figure 4.2. The wireframe was made after reviewing the clients demo video and case drawings.

In the video of the demo application[23], examinations were performed by clicking through a menu. This gave the group the inspiration to define all examinations using the same component, which was named action. Actions would be a simple class that had an activation function which would trigger when the object it is attached to, collided with a section of the patient. To control what would happen after the examination, actions activate all the results it is holding, further described in the section: Various Game Mechanics. In the Case Designer, actions would be displayed on the right side. Adding the action would bring it to the current tab where the user could set the results

The case drawings gave a recommended sequence for the categories. The wireframe presented an implementation where the sequence of actions could be evaluated by phases, further described in: The Phases & Milestones Mechanic.

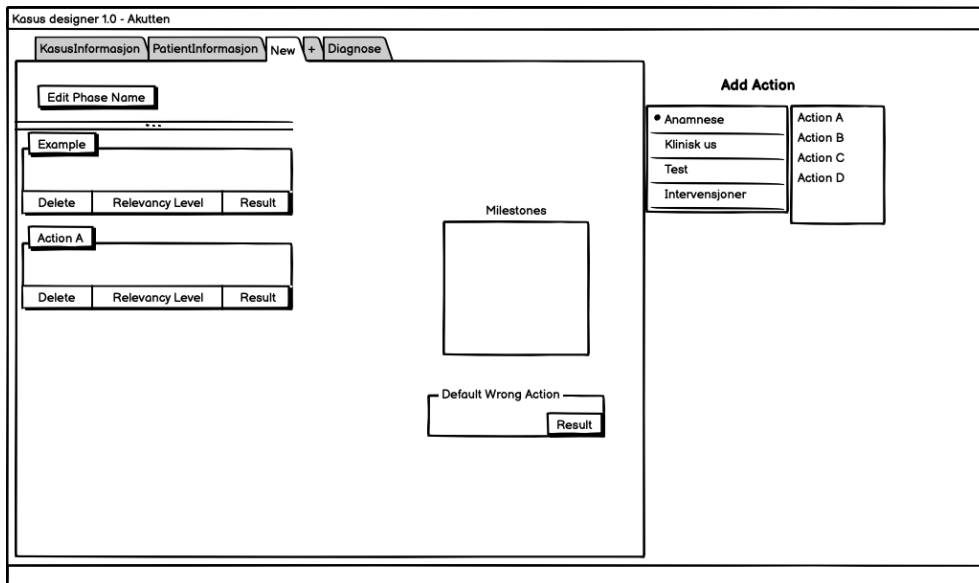


Figure 4.1: Case Designer Wireframe of a Phase where actions (which the player can take) and milestones (which must be met) can be set.

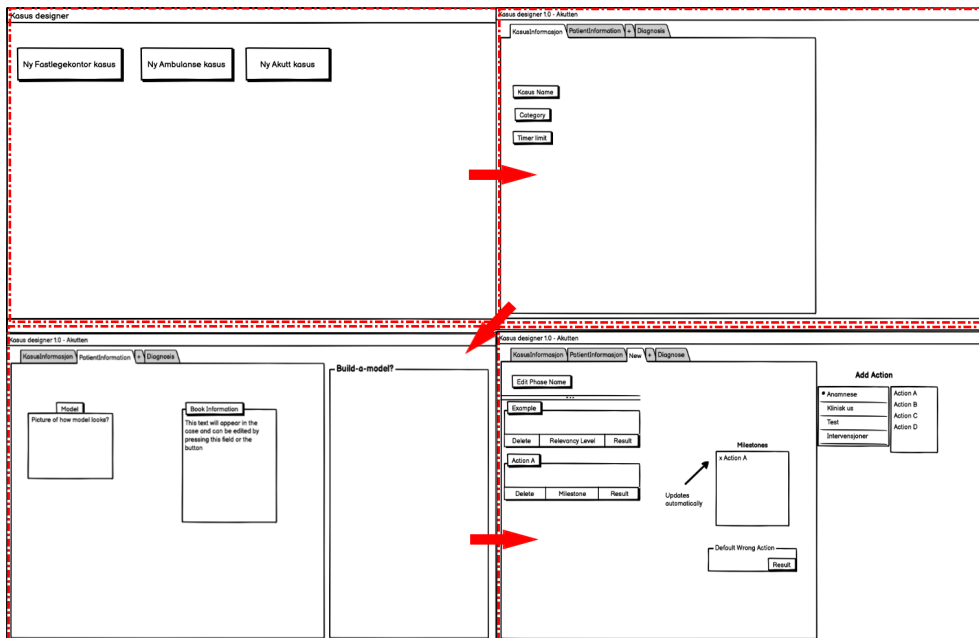


Figure 4.2: Case Designer illustration that consists of 4 wireframes. Chronological order is illustrated by the red arrows: left to right, starting at the top.

Virtual Paradigm - Project Ixora/Arbiter (© Norwegian University of Science and Technology)

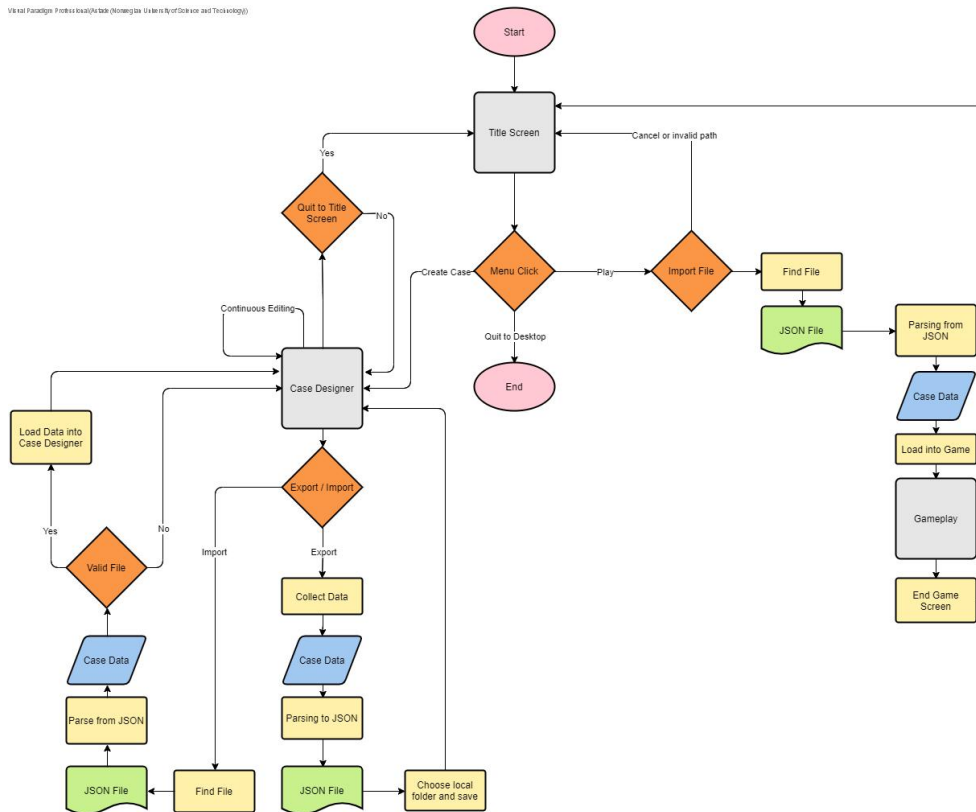


Figure 4.3: A high-order flowchart illustrating the general flows of the system and what actions the user can take.

4.2 Technical Design

As stated in the project description, the project was mainly separated into two parts: the gameplay part (which would be in VR) and a case editor that could be used to create an almost unlimited number of new game levels, commonly referred to as Cases. The relation between the different parts of the system is illustrated by Figure 4.3.

4.2.1 Various Game Mechanics

Actions & Results

As a game mechanic, an *Action* represents something the player can do. The class can be attached to wield-able tools which in turn will activate the action when used on a section of the patient. It can also be attached to the computer, triggering when the user requests a test.

Actions define where they are used, how long it takes to complete, the relevancy of the action and what the results of performing the action. Results are handled

by storing result objects, which are activated when the action is activated.

A result is a change in the game that would happen after the action is performed. The initial ideas for results that could be implemented can be seen in Table 4.1.

Result	Description
Change Time	Add or deduct time remaining for the case.
Bullet Point	Add text to the patient journal
Text Display	Displays text on the tool equipped.

Table 4.1: Initial result ideas

Correctness / Relevancy Level

Each action would have a *correctness* or *relevancy* level, which included the levels: wrong, neutral and correct. It is important to point out that the client prioritized the theoretically correct *process* and not necessarily what would be correct or wrong in the specific case; if the player somehow just *knew* what would be correct – when they have no fair way of actually be sure of it – or they just *guessed* correctly, then this would not necessarily correlate to the action’s correctness level being correct or wrong unless it was relevant in the current context.

The Phases & Milestones Mechanic

As illustrated by Figure 4.1 and Figure 4.2, there was early an idea to set the relevancy level on actions based on the sequence they were completed in. This resulted in the concept: *phases & Milestones*.

The idea was that actions could change their results depending on which phase was currently active, and each phase would have one or more *milestones*. To progress, the player would therefore have had to do all the actions registered as milestones for the current phase in order to progress to the next phase. This would in turn simulate the importance of what order things were done in, without over-complicating the logic.

From a gameplay perspective, this idea seemed to work fine – we even managed to partially implement such a feature into the gameplay – with the caveat that an action that is a milestone in a future phase can be completed in earlier phases. The issue was found in the Case Designer. How would it be to visualize the relations between actions and milestones, and how they change from phase to phase be done in a scalable, yet simple, and intuitive way? All actions would have to be defined in all phases, creating large clutter on each of the tabs. This proved to be a much greater challenge than we first had anticipated, and consequently, this feature was not implemented in the time we had at our disposal.

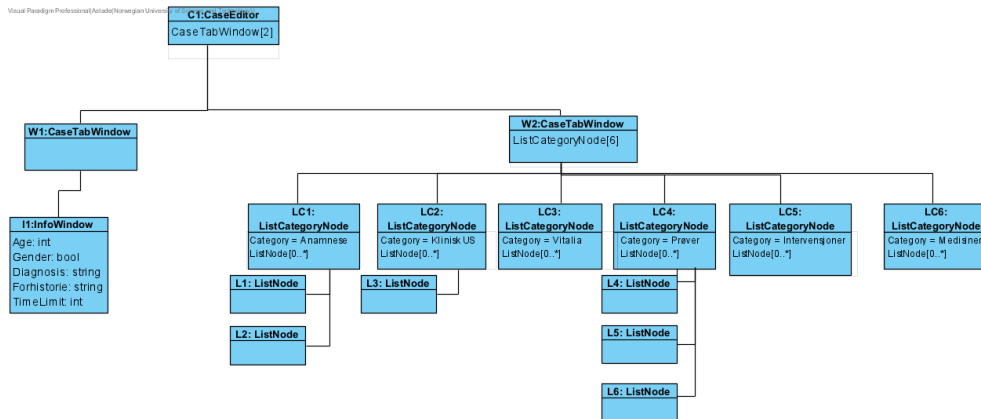


Figure 4.4: Case Editor Object Diagram.

4.2.2 System Architecture

The architecture of the Case Designer – as illustrated by the object diagram in Figure 4.4 and the class diagram in Figure 4.5 – is based around the CaseEditor; it has a collection of CaseTabWindows (the tabs described in Section 6.3), which holds all the ListCategoryNodes and thus also the ListNodes. Clicking on a List-Node will open the relevant NodeEditor, being it – in this case – a ToolEditor or AnamneseEditor. Additional editors can be added in the future if custom features are required for that particular ListNode’s parent category.

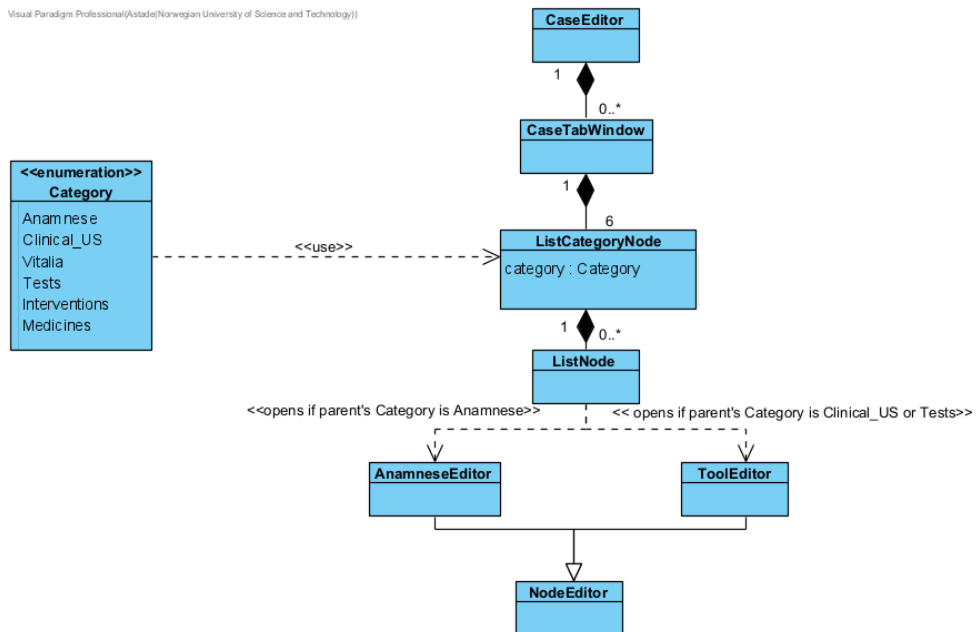


Figure 4.5: Case Editor Class Diagram.

4.2.3 Unity Project File Structure

As seen in Figure 4.6, all scripts, scenes and prefabs are stored inside this folder. The scripts used in the final MVP is located in the 'Scripts/Core' folder.

The scenes used in the application can be found in the 'Scenes/Main' folder. This contains the main menu, gameplay and Case Designer scenes.

Prefabs are found in the '_Prefabs' folder. Prefabs are further divided into prefabs for tools, editors, and gameplay UI.

Tests are found in the 'Tests' folder. Folders for playmode tests, editmode tests and scenes used by the tests are also found in this folder.

Lighting and render pipeline settings are found in the 'Settings' folder.

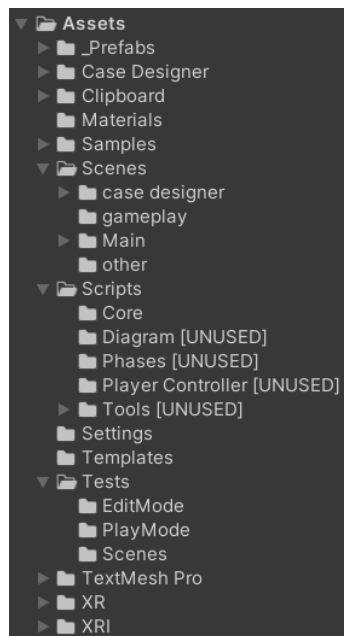


Figure 4.6: File structure

4.2.4 JSON Structure

To store a 'kasmus', or case we used JSON files. These are text files with a certain structure that allows their data to be read. In **Unity**, one can use the **JsonUtility** [24] class to parse an object to a string and vice versa. This string can then be written to a text file with the file-ending '.json'. The group created the class '**CaseData**' to represent said object, illustrated in Figure 6.13. An example JSON file can be seen in Code listing 6.7.

Chapter 5

Development Process

This chapter covers the development process, from choosing the development methodology to planning and executing.

5.1 Choice of Development Process

The development process chosen for the project would play a large role in how the development was conducted. To ensure a good workflow, it is important that the methodology fits the nature of the project. Therefore, a lot of consideration was put into its selection.

5.1.1 Considerations

The given product specifications and design, whilst giving a clear general outline of the product, left out a lot of details and requirements. From the start the group was given a lot of freedom in the design and development process. Due to the open-ended nature of the project, we will constantly have to pitch our ideas and have them approved by the client. The group were also not familiar with the field of medicine. This meant that plans would be highly subject to change as the groups understanding would need to be corrected by the client. The group also had varying experience with the technologies used in the project. This meant that the group would have to invest time in learning them as needed. This would further increase the probability of changes, as new solutions and limitations are found while learning the technologies.

There were many different methodologies that the group could choose from, examples were Rational Unified Process, Extreme Programming and Project Management Body of Knowledge. However, it was preferred that the group was familiar with the chosen methodology. The group has previously worked with the following methodologies in past courses: Waterfall Method, Scrum and Kanban.

5.1.2 Waterfall Method

A sequential process methodology like the waterfall method would require a lot of time being spent on designing the product. There was a high probability that these designs would change as the product materialized. If this were to happen during development, changes to design would not only have to be made on the component, but also on the other components that had its designed derived from the old design.

5.1.3 Scrum and Kanban

The group agreed that an iterative development process would best suit this project. Working in iterations would allow the group to make changes quickly when new requirements came from the client. Using iterations would also allow the group to focus on one component at the time. This was favourable as the system would consist of many decently sized components. The group considered both Scrum and Kanban, mostly favouring Scrum. One of the aspects of scrum that was not favoured was standup meetings and having a scrum master. We believed that it was not necessary to hold daily update meetings as the group only consisted of three members. Due to the group being small, we also did not believe that resources should be spent in assigning someone the role of scrum master. Instead, this could be tracked using cards, following the Kanban methodology. The group decided that the development process should be a mix of Scrum and Kanban, as this would allow the group to choose the favourable aspects from both methodologies. We therefore chose to use the *Scrumban* methodology. [25]

5.2 Use of Development Process

This section details how the group used the Scrumban methodology to guide the development process.

5.2.1 Kanban Board

A Kanban Board was used for task organization and planning. The board gives each group member an overview of what is being done, what tasks needs to be done and what needs approval. Tasks were ordered based on priority, where the highest priority tasks were placed on top of the board. This did not set limitations when choosing tasks, as group members were free to choose their next task after completing their current task.

The GitLab issue board was configured to simulate a Kanban board. Tasks were represented with GitLab issues. An issue contains the task name, state, and description. Comments were also added where they could provide useful details, or as documentation, like screenshots or videos.

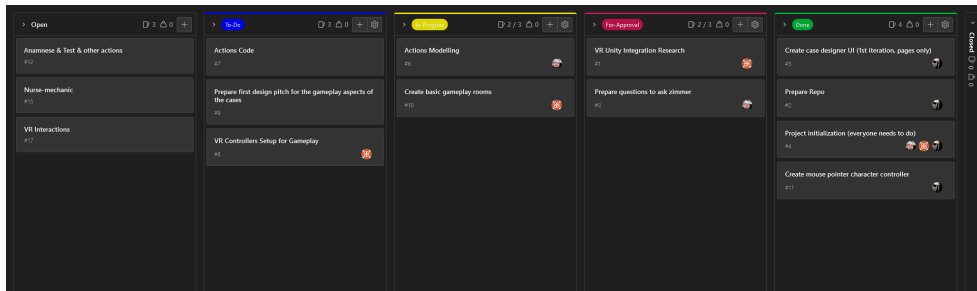


Figure 5.1: An example of the Kanban board.

The board, shown in Figure 5.1, was divided into the following categories: *Open*, *To-Do*, *In-Progress*, *For-Approval* and *Done*. The *Open* category serves as the backlog, containing tasks for future iterations. The *To-Do* category contains tasks that are planned to be completed in the current iteration. The *In-Progress* category contains tasks that are currently being worked on. The *For-Approval* category consists of tasks that has been completed, but await approval from group before it can be merged into the main branch. After having been approved the task is moved to the *Done* category. The *Done* category contains tasks completed in the current iteration, and is emptied by closing the issues after each iteration.

Scrumban methodology defines limits to ensure efficiency and that team members are not overburdened. Scrumban therefore has a WIP limit which states that each group member should only work on one task at a time. This is also limited on the Kanban board, where the number of tasks in the *In-Progress* column is equal to the amount of group members.

5.2.2 Iterations

Following Scrumban methodology the process was divided into iterations. Each iteration lasted for one week, starting on Mondays. Iterations started with a planning meeting and concluded with a review meeting. The short iterations were chosen as the product had many smaller components, and it was preferred to keep the focus of each iteration on few or a singular component.

5.2.3 Internal Meeting Structure

Meetings were predominantly based on demand where any group member could call a meeting when necessary. These meetings were held for planning, decision-making or approval.

Every iteration had one or more planning meetings. A new iteration started with a planning meeting where tasks were selected and added to the Kanban board. Additional planning meetings were called when iterations either needed more tasks or needed to be reviewed.

Review meetings were held to approve and review completed tasks. A review meeting was done at the end of each iteration. Tasks that were waiting for approval was approved and marked as done, allowing them to be merged into the main branch. Not completed tasks were moved to the backlog and could later be added to the next iteration. Further review meetings were called on demand, often when changes needed to be approved.

5.2.4 Client Meetings

Meetings with the client were scheduled every Monday at 17:00. The meeting was used for iteration review and planning. Progress from the previous iteration was presented and feedback was received. The client could also take part in planning tasks for the next iteration, as well as specify wanted features and changes.

5.2.5 Planning Meetings

Planning meetings were initially scheduled on Tuesdays after meeting the client. The planning meeting was used to discuss the feedback given by the client, and set plans for what needs to be done, and which tasks to prioritize.

The group encountered a problem where the client meetings did not result into clear, small achievable tasks for the iteration. We therefore switched to a new approach. Planning meetings were held on Mondays before the client meeting. During the planning meeting we created our own pitch of what should be done during the iteration. This pitch was then shown to the client during the client meeting. This proved to work well, as the client did a great job at setting requirements for the proposed tasks. A short internal meeting was then hosted straight after the client meeting to confirm if the iteration was properly defined.

5.2.6 Review Meetings

Review meetings were held first thing on Mondays. The review meetings were used to go over tasks done, final approvals were given if work had been done during the weekend. After this, the group would have a discussion on how the iteration went and give updates on how the tasks were implemented, this was sometimes done with live demonstrations.

5.2.7 Supervisor Meetings

Weekly meetings with the project supervisor were scheduled every Friday at 12:15. These meetings were used to discuss the project report and the development process, where the supervisor would provide feedback and suggest improvements.

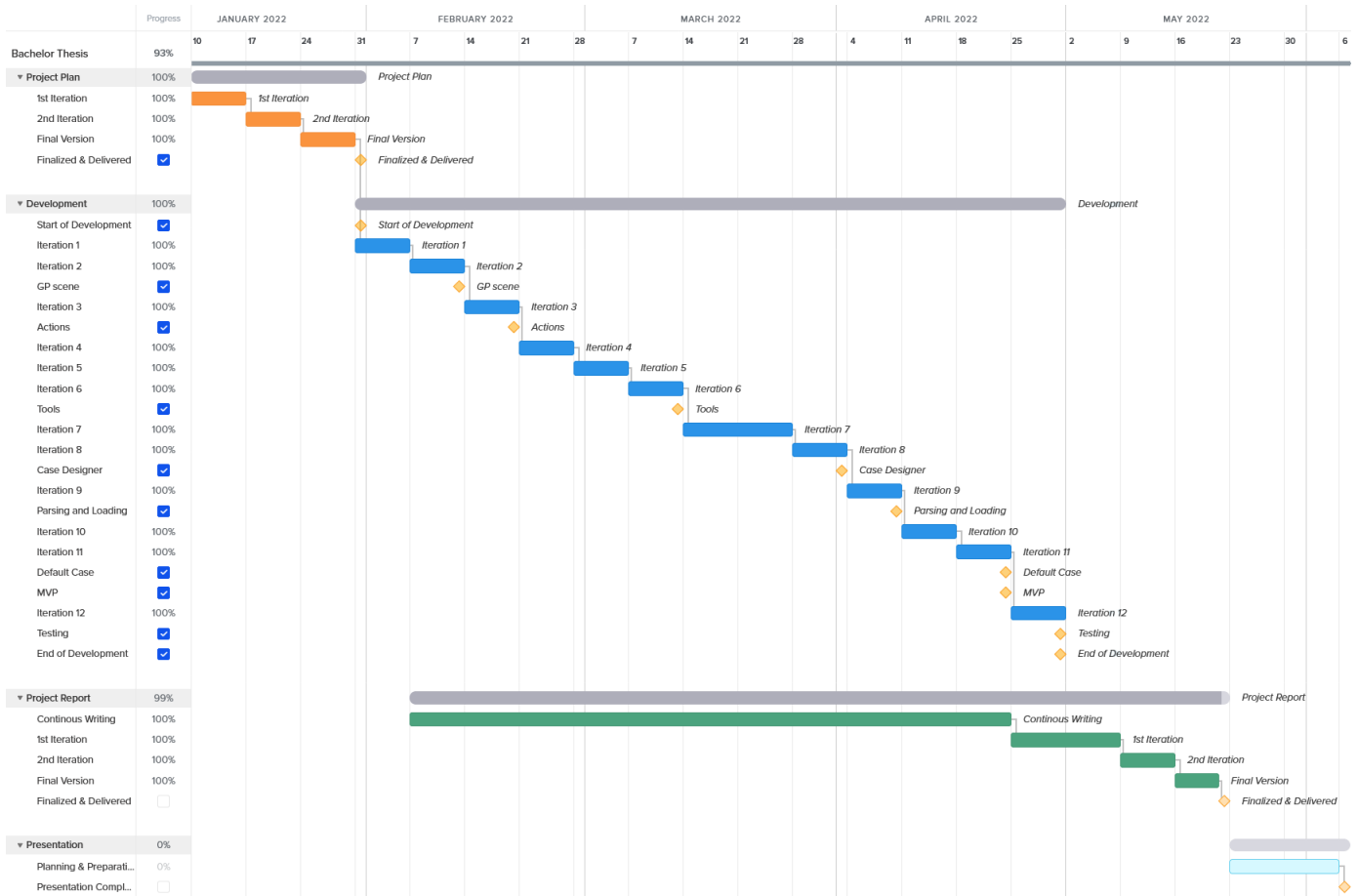


Figure 5.2: Gantt chart.

5.2.8 Gantt Chart

A Gantt chart was created in Trello with the TeamGantt power-up. Figure 5.2 shows the full project timeline.

5.2.9 Time Tracking

Each group member has kept track of their working hours in Toggl during the development. Toggl is a time tracking application that allows the user to title their working session and add labels for further categorising. The main categories used in labeling were:

- **Meetings** – Work related to participating or planning for meetings
- **Product development** – Work related to designing, expanding, improving and documenting the code base
- **Report writing** – Work related to writing the project report
- **Technological Research** – Work related to learning new technologies

- **Iteration Planning** – A subcategory of Meetings that only includes internal meetings planning the iterations.

5.2.10 Milestones

Initially, the only milestone was creating the first MVP. The MVP was meant to be as simple as possible and would be expanded based on the wishes of the client. This was chosen because of the initial lack of requirements, which made it hard to set concrete goals that could be met. At the start of the development period, the group realised that a lot of time would be spent redesigning and re-implementing the case designer as more components were added. It was also not clear when this milestone would be complete as it was ambiguous what a simple solution was. The group therefore spent more time defining the requirements of the product. After doing so, a final list of milestones was made:

1 – General Practitioner Scene

The General Practitioner (GP) scene is the scene in which playthrough is ran on. The goal of this milestone is to design the scene in which the gameplay will take place. This milestone focuses entirely on design and does not include any functionality requirements. To achieve this milestone, these requirements must be met:

- Modelling the room with assets given by the client
- Placing the patient in the room
- Placing tools in the room
- Implementing movement and interactions in gameplay

2 – Actions

The player must be able to do examinations. The examinations will have varying different results based on the examination and context. The goal of an action is to be able to link all different changes in the game on this component. The changes will be its own component, called result. When an action is completed, all results linked with the action will activate and create changes in the game. To achieve this milestone, these requirements must be met:

- Create the classes: **Action** and **Result**
- Actions must have completion time and a field deciding which GameObject it is used on.
- Actions must be able to hold any number of results.
- Action must have an activation function, the activation function must activate all activation functions on the result that it holds.
- Results must be an abstract class that forces the inheritors to include an activation function.

3 – Tools

Examinations will be conducted using tools. This milestone consists of creating a Tool class that will hold actions. To complete this milestone these requirements must be met:

- Create a tool script that holds actions and that can be attached to GameObjects within the game scene.
- Attach the tool script to the tools made in milestone 1.

4 – Case Designer

When reaching this milestone, all components needed to make a playable game should have been implemented. To reach this milestone the case designer must:

- Meet the complexity and user interface requirement declared in Section 3.2.
- Implement all functionalities needed to realize Table 3.2.
- Must organize data in a way that makes parsing in milestone 5 simple.

5 – Exporting and Importing

Milestone 5 consists of parsing the data created in the case designer and transforming it into a JSON file. To complete this milestone these requirements must be met:

- Case data is exported into JSON file format.
- Case designer content can be exported.
- Case designer can load case data.
- Importing case data into the case designer will load the same case data that was exported.
- Game can be loaded using case data.

6 – Default case

Create the first case that will be used to prove the concept of the product. This case must define enough of interactions to realize the use case found in Table 3.3.

7 – Testing

Create a strong foundation for future development. Create test cases to cover all functionalities offered by the product. To complete this milestone these requirements must be met:

- Unit tests on components covering actions, case designer and game manager
- Integration tests that for functionality requirements defined in Section 3.1

MVP

Finalize the product into an MVP. Case designer and the game should be accessible through a main menu. Additionally, the code base must be checked that it meets the requirements set in Section 3.3.

5.2.11 Milestone completion chart

Table 5.1 gives an overview of when each milestone was reached.

Milestone	Completed
GP scene	Iteration 2
Actions	Iteration 3
Tools	Iteration 6
Case Designer	Iteration 8
Parsing and Loading	Iteration 9
Default Case	Iteration 11
Testing	Iteration 12
MVP	Iteration 11

Table 5.1: Milestones and completion iterations.

Chapter 6

Implementation

The implementation was primarily divided into the two main components described in chapter 3: gameplay and case editor. JSON parsing was also added as a component that would bind the previous together, through reading from and writing to file. A title screen, or main menu, was added to navigate between these, and to start or exit the application.

6.1 Tools

6.1.1 Unity Game Engine

Development was done within the Unity game engine (see 2.2.1) as per project requirements. The version used was 2020.3.26f and was chosen based on it being the latest LTS version at the time of the project starting, and therefore considered stable and proven.

Assets

The client provided us with an asset pack, named *Hospital - modular building, props and characters* [26], that was used for most of the gameplay environment.

Packages

Unity comes with the `XRInteractionToolkit` for VR game development. This toolkit was used to handle VR input events.

6.1.2 Visual Studio

Visual Studio was the IDE used for script editing. Unity uses Visual Studio as its default IDE and comes with the *Code Editor Package for Visual Studio* to offer seamless integration.

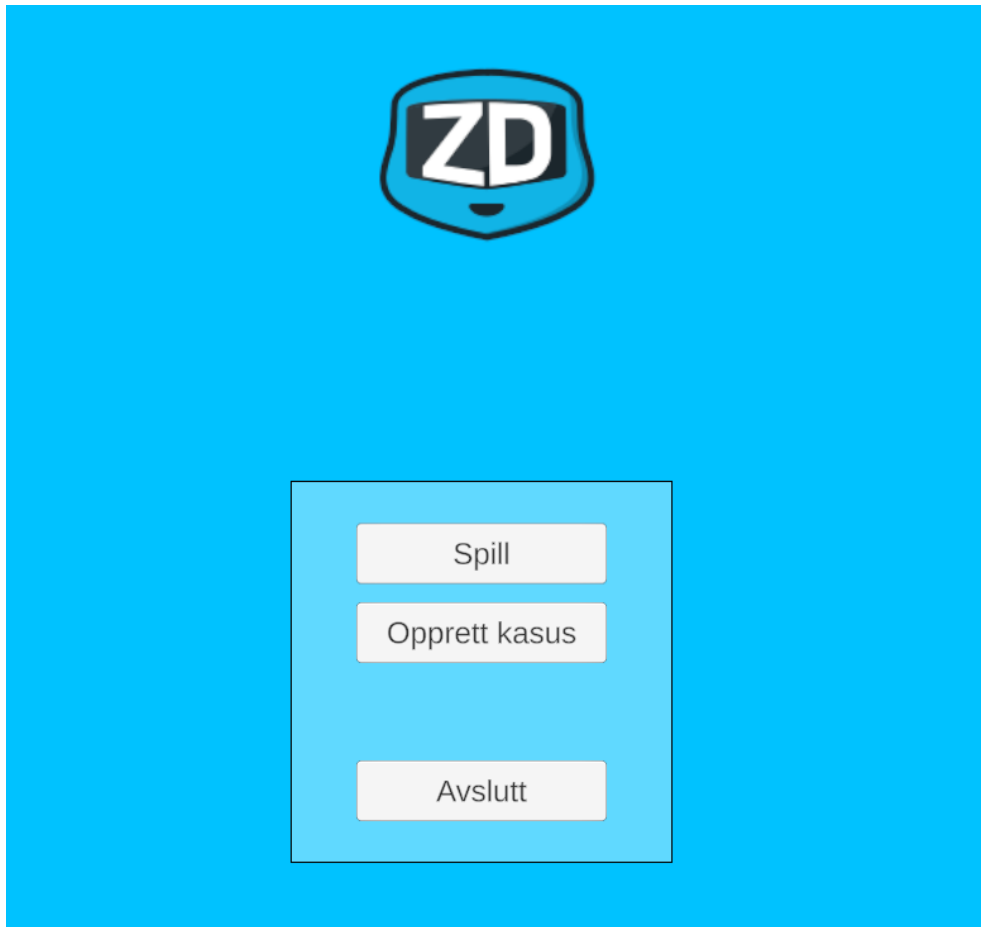


Figure 6.1: The Main Menu.

6.2 Title Screen

The title screen serves as a main menu to navigate between the case editor and gameplay, or to exit the application. Figure 6.1 shows a cropped version of the title screen.

The first button, *Spill*, is used to start a new game. When clicked it will open the file explorer to select the case JSON file to be played. The file selector panel is provided by the `UnityEditor.EditorUtility` class. The second button, *Opprett kasus*, opens the case editor scene. The last button, *Avslutt*, exits the application.

6.3 Case Editor

The case editor consists of two main tabs named *informasjon* and *undersøkelser*, which are the Norwegian words for *information* and *examinations* or *actions* respectively.

The first tab (*informasjon*), shown in Figure 6.3, consists mostly of information concerning the patient and the case, which the player would automatically get to know at the very beginning of play. Currently, this information is loaded directly into the player's *clipboard* (see Figure 6.2) at the start of the game.

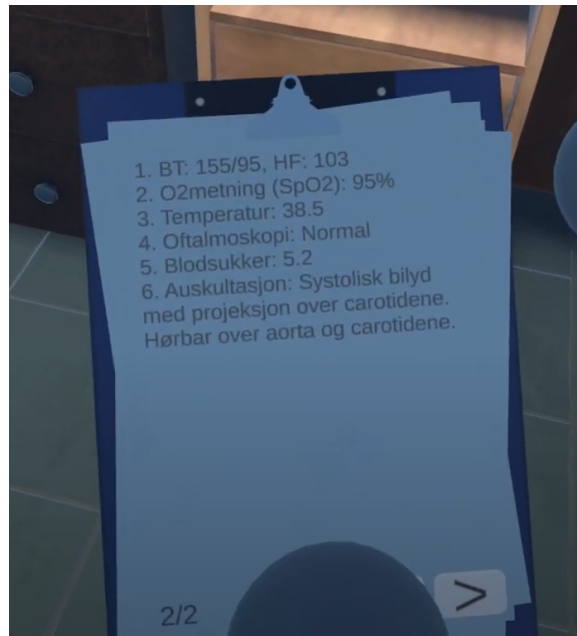


Figure 6.2: The clipboard is where information the player receives during play is automatically registered and displayed.

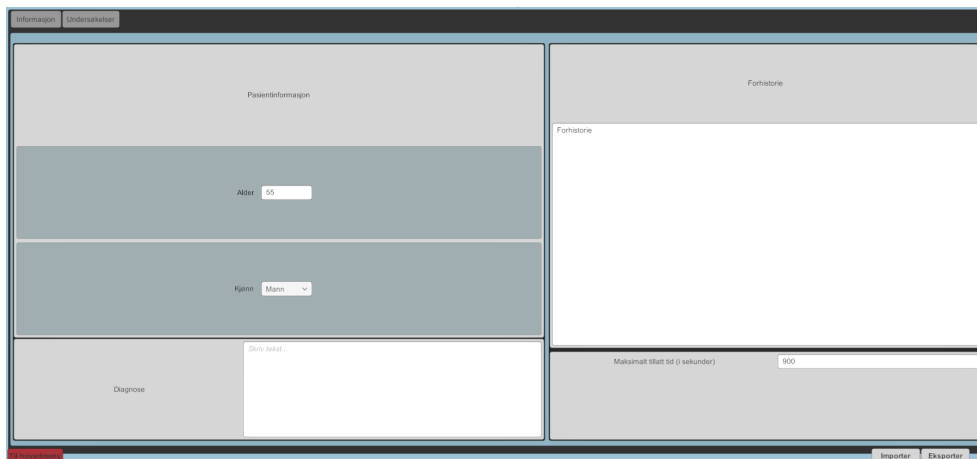


Figure 6.3: The *informasjon* tab (meaning *information* in Norwegian).

The second tab (*undersøkelser*), shown in Figure 6.4 and 6.5, holds most of the Case Editor logic. It is divided into six main categories: *Anamnese*, *Klinisk US*, *Vitalia*, *Prøver*, *Intervensjoner* and *Medisiner*. The decision to divide them was based

on client preference. Only *Anamnese*, *Klinisk US*, and *Prøver* were implemented, however, the framework of this system was made with scalability in mind, which supports the necessary further development which would be needed.

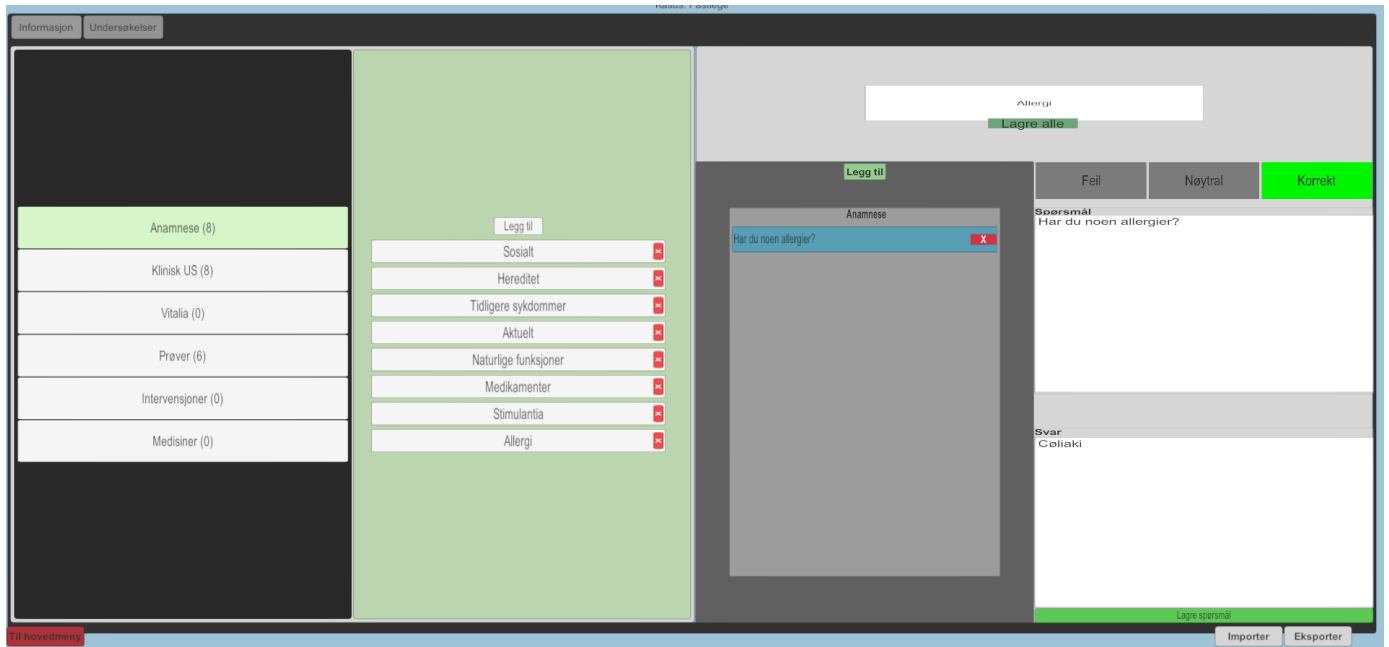


Figure 6.4: The *undersøkelser* tab (meaning *examinations* or *actions* in Norwegian).

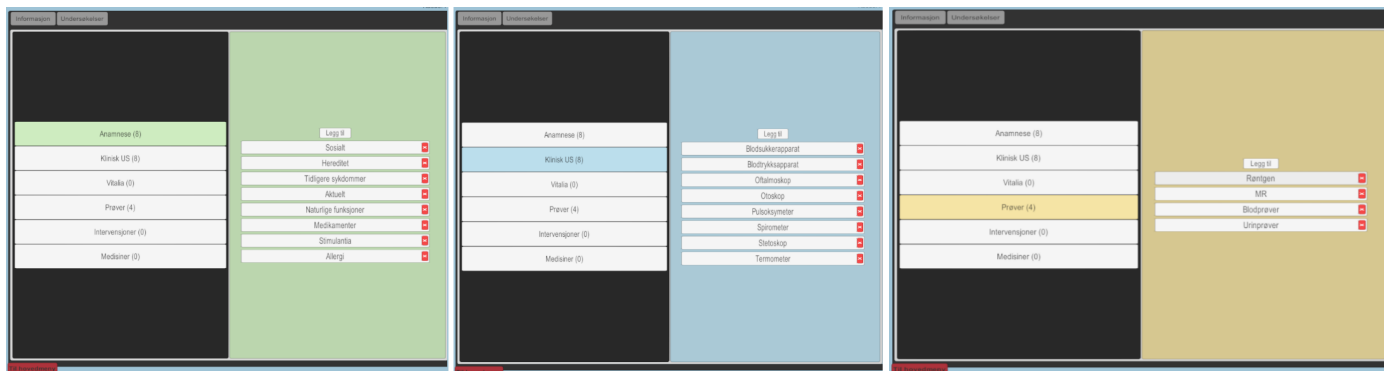


Figure 6.5: Categories of the Case Editor – Figure showing three images side-by-side of parts of the final design. Categories are color-coded to clearer convey parent-child relations.

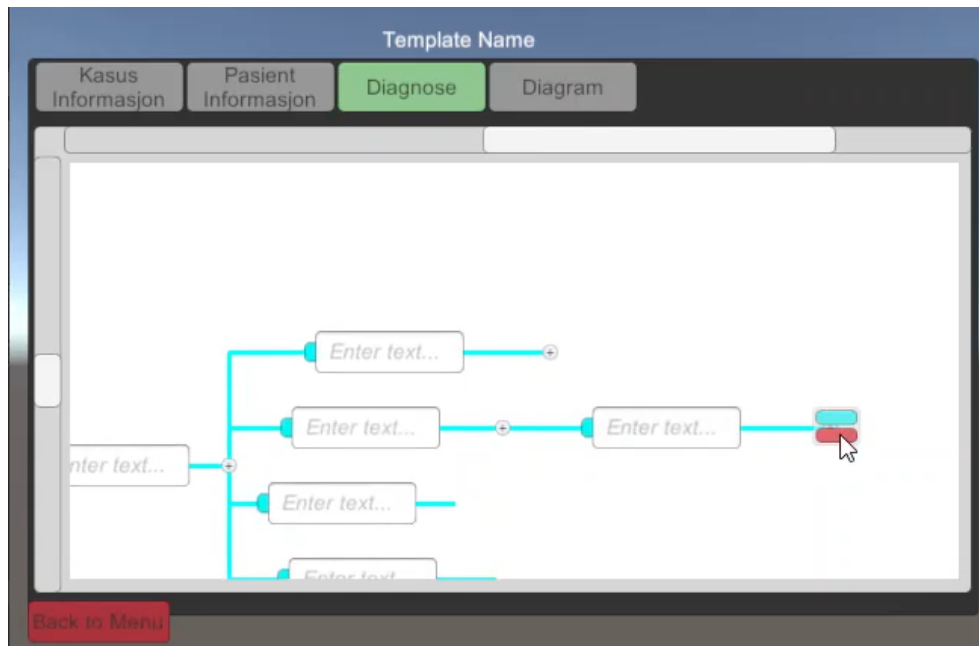


Figure 6.6: A screenshot from a development video of a tree-graph system. This solution was later replaced with a list-category system instead.

6.3.1 Changes during development

A consequence of little client involvement, a relatively open task, and a subject area the team has little knowledge and experience in, was that the product was highly likely to change quite significantly during development; and as such, it was difficult to design a system architecture we could use and follow early. The project team would often need to learn what worked and what did not as the development went on, and so the software went through several radical changes.

The Tree-Graph Solution

Perhaps one of the biggest changes from the original plans was the implementation of a tree-graph, seen in Figure 6.6; this was inspired by several medical diagrams sent by the client, which illustrated the different actions a player could potentially take as a tree where choices could branch out into new choices. This *solution* is further described in issue 19 in the project repository, which can be found in Section 1.2. The main problem and difficulty with such a solution was that the tree graph would have to be dynamic as to account for the changes the user would do when creating a case. This was significantly harder to implement than expected, as it was always an issue with trying to keep the tree tight, but not so tight that any nodes would overlap each other. Calculating the exact distance and space each node would need to not overlap was fairly easy for one and maybe two levels deep in the tree, but further than that the tree began to branch

in different directions and would need different amounts of space depending on which side it would branch out to. In the end, we decided to abandon this idea for a simpler solution (see Section 6.3.1).

Current Solution

The current solution is a radical simplification to the tree-graph solution; it basically works like the two first levels of the tree-tree graph, represented by lists instead. Having lists instead, meant that it was very easy to dynamically add or remove elements from the list. This list is the collection of buttons inside the green box as shown in Figure 6.4.

6.4 Gameplay

The current gameplay implementation consists of one scene, *GameplayGP*, representing a general practitioner environment.

6.4.1 Controls and Movement

Controls and movement are implemented with the *XR Interaction Toolkit* package.

Adhering to best practices for VR movement and to minimize motion sickness, movement is performed by pointing the controller and teleporting. The player can teleport onto any surface that is a *TeleportationArea*. Currently the teleportation area covers the entire floor. The area can be customized by placing and resizing teleportation area prefabs.

A screen fade effect was added when teleporting, mainly to minimize motion sickness. This gives a slight fade to black when teleporting around.

Interactable objects, such as tools and the clipboard, utilizes the *XR Grab Interactable* class provided by the *XR Interaction Toolkit*. This makes objects grabbable by the VR controllers and was used as a foundation to expand upon by defining the behaviour on interaction.

6.4.2 Environment

The gameplay environment was built mostly with the assets specified in Section 6.1.1. The room is built to best simulate a physical doctor's office. Due to limited access to assets, some visual elements are either not present or represented by simple placeholders.

Most noticeably there was no assets for the tools, therefore these were given placeholder visuals, mostly made up of simple cubes. They were given slightly differing shapes to help differentiate between them. The current implementation also has the meshes and script components as separate child object. This was done to allow the meshes to be easily changed later on.

Code listing 6.1: The Action class data fields.

```

1  public readonly string name;
2  public readonly Tool.ToolID parentID;
3  public readonly float timeToComplete;
4  public readonly bool background;
5  public readonly RequiredTag requiredTag;
6  public readonly List<Result> results;
7  public readonly Correctness.Level correctness;
8  public Action(string name, Tool.ToolID parentID, float timeToComplete, List<
    Result> results, bool background, RequiredTag requiredTag, Correctness.
    Level correctness)

```

Code listing 6.2: The Result class.

```

1  public abstract class Result
2  {
3      /// <summary>
4      /// <see cref="GiveResults()"/> is the method that gives feedback in the game
5      /// from doing actions.
6      /// i.e. Adds a bullet point to journal, change patient properties and turn on
7      /// machines
8      /// </summary>
9      public abstract void GiveResults();
10 }

```

6.4.3 Interactions

Actions

The Action class represents an intractable event that can be triggered by the player. It will hold all necessary data to perform the action, as well as the result triggered upon completion. Code listing 6.1 shows all the data an Action object holds.

Actions are completed by calling the PerformAction method, which will then trigger all the results of the Action object.

Results

The current implementation contains three result types; notebook, time and tool display. All result types are derived from the abstract class Result in Code listing 6.2. This abstraction layer should make it easier to expand functionality by adding new result types.

Results are triggered by calling the GiveResult method. Each result type has their unique data and GiveResult method. The NotebookResult class will add a defined result string to the players in-game *clipboard* (see Section 6.4.3). The TimeResult class changes the gameplay timer, to either give the player more or

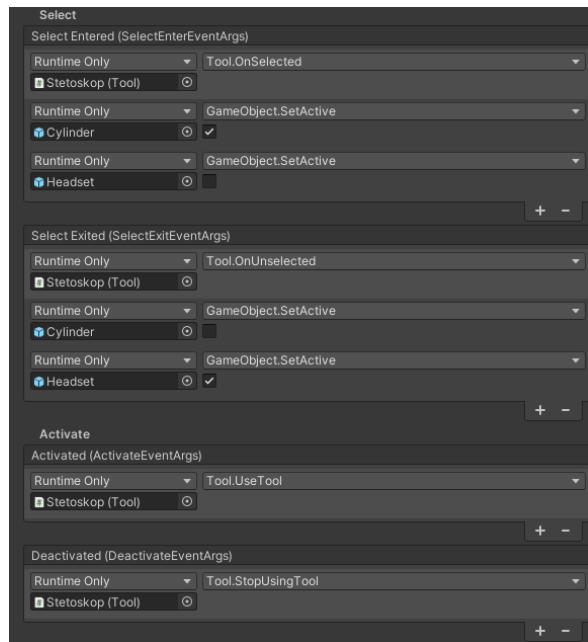


Figure 6.7: XR Grab Interactable tool setup.

less time, by interacting with the GameManager (see Section 6.4.4). The `Tool-TextDisplayResult` class is used to update the visual display of tools when they are used, currently only used for the blood sugar monitor.

Tools

Tools are interactable `GameObjects` in the game that the user interact with to perform actions. A tool object has a `Tool` script and an `XRGrabInteractable` component attached to it.

The `XRGrabInteractable` script triggers events when the object is picked up, activated, deactivated and put down. The events are added as shown in Figure 6.7. Every tool is configured with the `OnSelected`, `OnUnselected`, `UseTool` and `StopUsingTool` methods provided by the `Tool` component.

Additional events are also added to some tools where more functionality is wanted. An example of this is the stethoscope, where the 3D model is changed when picking it up, such that the player is holding the chest-piece instead of the whole device. This is done by the `GameObject.SetActive` events also seen in Figure 6.7.

The `Tool` script acts as a controller for a tool, that handles a tools actions and interaction events.

When a tool is picked up or put down the `OnSelected` and `OnUnselected` methods are called. This will enable or disable the `ActionTrigger` objects that the tool can be used on by calling the `TriggerController`, as shown in Code listing 6.3.

Code listing 6.3: The OnSelect method in Tool.

```

1  /// <summary>
2  /// Show where the <see cref="Tool"/> can be used visually by enabling <see
   cref="ActionTrigger"/>s for all <see cref="Action.requiredTag"/> in <see
   cref="actions"/>.
3  /// </summary>
4  public void OnSelected()
5  {
6      TriggerController.Instance.ShowTriggers(actions.Select(action => action.
   requiredTag).Distinct().ToList());
7  }

```

Code listing 6.4: The OnTriggerStay method in Tool.

```

1  /// <summary>
2  /// Checks if tag of <paramref name="other"/> equals <see cref="requiredTag"/>,
   if true <see cref="inTrigger"/> is true.
3  /// </summary>
4  /// <param name="other">Collider of other object</param>
5  private void OnTriggerStay(Collider other)
6  {
7      if (other.CompareTag("ActionTrigger") && inTrigger == Action.RequiredTag.
   Usatt)
8      {
9          inTrigger = other.GetComponent<ActionTrigger>().TriggerTag;
10         other.GetComponent<ActionTrigger>().SetColorGreen();
11     }
12 }

```

A tool can only be used or activated when it is in an `ActionTrigger`. This is handled by the `OnTriggerStay` and `OnTriggerExit` methods of the `Tool` class. When a trigger is entered, i.e. collision is detected, the tool will set the `inTrigger` property to the entered trigger and the tool can be used. Code listing 6.4 shows the `OnTriggerStay` method.

Pressing or releasing the controller activate button will trigger the `UseTool` and `StopUsingTool` methods respectively. This will call the `ProgressBar.Instance` to either `ProgressAction` or `CancelProgression`.

The progress bar is a UI element controlled by the `ProgressBar` script. When using a tool, the progress bar seen in Figure 6.8 will progress until the `timeToComplete` of the action is reached, before being completed. Releasing the activate button will cancel said progression.

Anamnesis

An anamnesis consists of categorized questions the player can ask the patient. These are represented by the `AnamneseAction` class in Code listing 6.5. Such an action consists of a question, answer, category, and correctness.

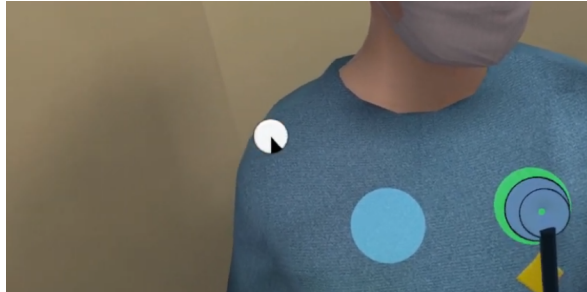


Figure 6.8: Screenshot from gameplay showing the *progress bar* as an action is being performed using a stethoscope.

Code listing 6.5: The AnamneseAction class.

```

1  /// <summary>
2  /// An action of type anamnesis, holding a question with an answer and its category
   and correctness.
3  /// </summary>
4  public class AnamneseAction
5  {
6      public readonly string question;
7      public readonly string answer;
8      public readonly string category;
9      public readonly Correctness.Level correctness;
10
11     /// <summary>
12     /// Initializes a new instance of <see cref="AnamneseAction"/>.
13     /// </summary>
14     /// <param name="question">The question</param>
15     /// <param name="answer">The answer</param>
16     /// <param name="category">The category</param>
17     /// <param name="correctness">The correctness</param>
18     public AnamneseAction(string question, string answer, string category,
19         Correctness.Level correctness)
20     {
21         this.question = question;
22         this.answer = answer;
23         this.category = category;
24         this.correctness = correctness;
25     }
26 }

```

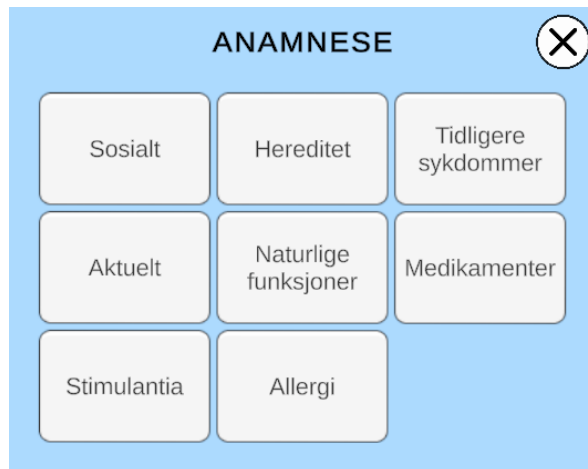


Figure 6.9: The anamnesis menu.

In the game the anamnesis is performed by interacting with a UI above the patient. When the anamnesis icon is pressed, a menu will appear containing all the categories. It is shown in Figure 6.9. The menu is automatically generated from the `CaseData`. Every category can be clicked to open a list of all questions in the respective category. This list is shown in Figure 6.10.

The creation and handling of an anamnesis is done by the `AnamnesisController` script. The controller starts by importing all `AnamnesisActions` from the `GameManager`. It will then add all the UI for each Action with the help of prefabs and programmatically added button `onClick` listeners.

The goals of this implementation were to ensure scalability, allowing new categories to be added directly in the editor or the JavaScript Object Notation (JSON) file. It should be able to handle the addition of large amounts of both new categories and questions.

Referrals

Referrals are stored and treated as Action objects belonging to the tool: `pc`. They differ from normal actions in that they are activated through a UI on the PC within the game.

The referral UI is created similarly to the anamnesis, by the `ReferralController`. The controller is attached to the PC game object along with a `Tool` component. The `Tool` component will fetch all actions with the `pc toolId` at the start of a game. The controller will import the list of actions from the `Tool` component to create the menus and buttons.

First a menu of all the categories is created for each unique `requiredTag` in the list of Actions. Figure 6.11 shows the categories on the left. Each category is a button



Figure 6.10: The anamnesis category question list.



Figure 6.11: The referrals UI.

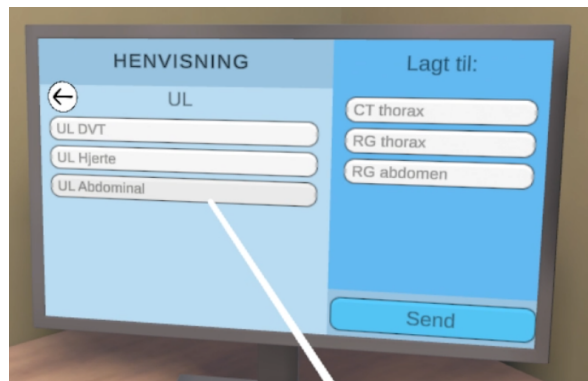


Figure 6.12: The referrals UI.

that can be clicked to open a list of its respective actions. From there an action can be added to the *Added* or *Lagt til* list seen on the right side of Figure 6.12.

After actions have been added the player can click the *Send* button to simulate sending referrals. All Actions in the list will then be performed simultaneously. A screen fade was added as a way to simulate the passing of time; i.e. the appointment ending and the patient returning, starting a new appointment. The screen fade effect in Code listing 6.6 is implemented using coroutines, such that the events trigger after the screen has faded to black.

Clipboard

The clipboard is used to display `NotebookResults` after the player has performed an action. It consists of the `Notebook` and `Page` classes. `Notebook` controls the behaviour of the clipboard, adding new entries and pages, as well as navigating through the pages.

The first page contains the patient information and initial background of the task. The following pages has the results. Adding a new point can be done using `Notebook.Instance.AddPoint`. When an addition is made it will check if there is space on the current page, or if a new page needs to be added.

A `Page` object holds all entries on a page. All pages are stored as a linked list in the `Notebook` class. Navigation between pages is performed by clicking the arrow buttons on the bottom of the clipboard.

6.4.4 GameManager

The game manager handles several key gameplay elements, including loading, time limits and game ending.

At the start of a game, the `GameManager` will be the first component to initialize. It will load `CaseData` from the selected JSON file and parse the data to `Action`

Code listing 6.6: Example of screen fade using coroutines.

```

1  /// <summary>
2  /// Starts corutine to fade out and send referrals, fades back in to resume the
   game.
3  /// </summary>
4  void OnSendButtonClick()
5  {
6      if (addedReferrals.Count != 0)
7          StartCoroutine(FadeSequence());
8  }
9
10 /// <summary>
11 /// Fades to black, waits for a second, then executes <see cref="SendReferrals"/>
   and fades back to game.
12 /// </summary>
13 IEnumerator FadeSequence()
14 {
15     fadeScreen.DefaultFadeIn();
16
17     yield return fadeScreen.CurrentRoutine;
18     yield return new WaitForSeconds(1.0f);
19
20     SendReferrals();
21
22     fadeScreen.DefaultFadeOut();
23 }

```

and AnamneseAction objects. Tools and AnamneseControllers can then get these from the GameManager.Instance when they are initialized later, using accessor methods.

The clipboard front page is also updated with relevant information from the Case-Data. This is done by finding the Notebook.Instance and using its AddPage method.

During gameplay the GameManager will count and keep track of the time limit, updating the in-game timer, and ending the game when time is out.

When a game is ended, either by time or player input, the GameManager will end the case playthrough and send the player to the end game screen.

The end game screen is a separate room where the player is given an overview of the playthrough. A screen will show which actions have been performed as well as their correctness. The player can also toggle a view of all available actions, to see what could or should have also been performed.

6.5 JSON Parsing

Code listing 6.7: An example of a JSON file containing the case data.

```

1  {
2    "appVersion": "ZimmerDigitalSim_v0.1.2",
3    "patientInfo": {
4      "age": 55,
5      "isMale": true,
6      "diagnosis": "",
7      "forhistorie": "Over 4 uker med feber, nattesvette, nedsatt appetitt og
8        utmattelse.",
9      "timeLimit": 900
10   },
11   "nodeList": [
12     {
13       "name": "Tidligere sykdommer",
14       "parentCategory": 0,
15       "anamneseDataList": [
16         {
17           "question": "Hypertensjon",
18           "answer": "2005: hypertensjon, kalsiumantagonist/amlodipin",
19           "level": 1
20         }
21       ],
22       "actionDataList": []
23     },
24     {
25       "name": "Blodtrykksapparat",
26       "parentCategory": 1,
27       "anamneseDataList": [],
28       "actionDataList": [
29         {
30           "name": "Blodtrykk",
31           "toolID": 1,
32           "timeToComplete": 3.0,
33           "background": false,
34           "requiredTag": 6,
35           "results": [
36             {
37               "resultType": 0,
38               "notebookPoint": "BT: 155/95, HF: 103",
39               "timeAmount": 0,
40               "statName": "",
41               "displayText": ""
42             }
43           ],
44           "level": 1
45         }
46       ]
47     }
48   ]
49 }

```

The final implementation of JSON file structure, as can be seen in Code listing 6.7, is reflected by the structure shown in the CaseData class diagram Figure 6.13. Without going into exact details, what is important for the file to be accepted and read is that the field `appVersion` exists and that its content matches the value

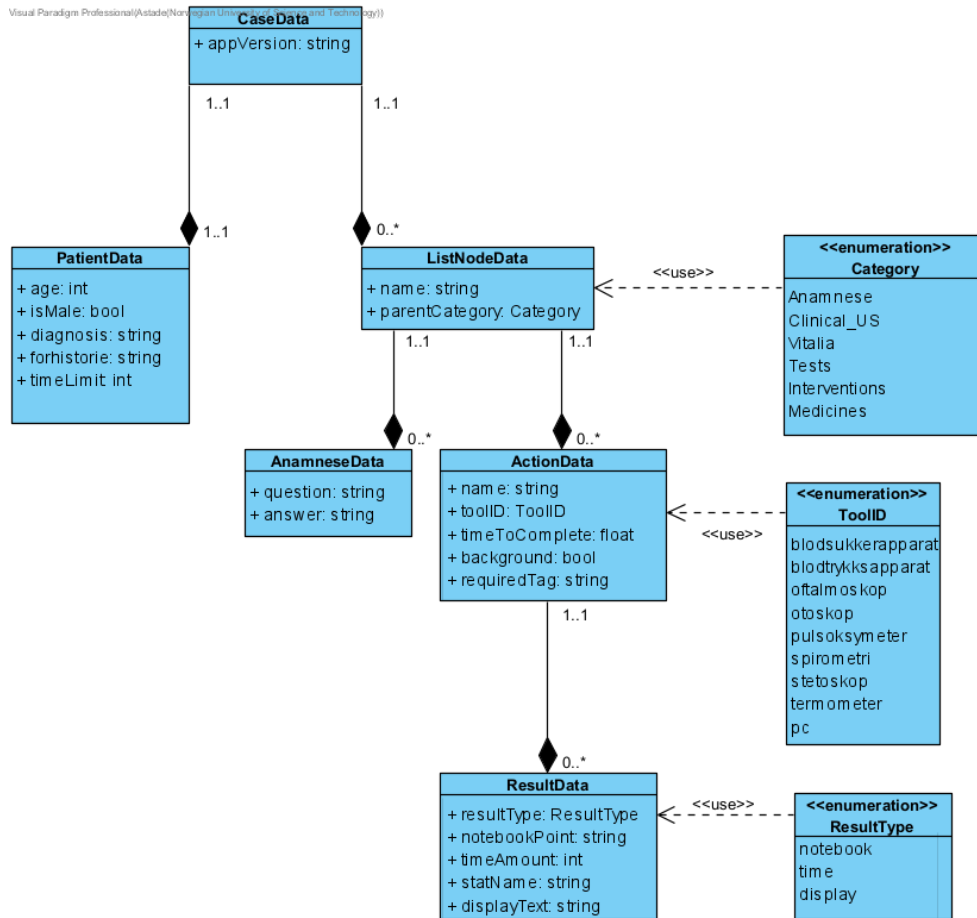


Figure 6.13: A class diagram illustrating the data structure used for creating the JSON files containing the data used in both gameplay and Case Designer.

Code listing 6.8: Shows the variable *FullAppNameVersion*. Together with *AppName* and *AppVersion*, it creates the field *appVersion* in the output JSON (see 6.7).

```
1  /// <summary>
2  /// This is a collection of all hard-coded static and global constants, all in one
   place.
3  /// </summary>
4  public static class StaticVariables
5  {
6      // Software name and version
7      public static readonly string AppName = "ZimmerDigitalSim";
8      public static readonly string AppVersion = "0.1.2";
9      public static readonly string FullAppNameVersion = AppName + "_v" + AppVersion;
10
11     ...
12 }
```

of the variable *FullAppNameVersion* in the *StaticVariables*-class, as shown in Code listing 6.8. Furthermore, fields and most values must not be tampered with as it may lead to the fields not being read correctly.

Chapter 7

Testing and Quality Assurance

A diverse set of tools and systems were used from the very start to ensure high quality of the project work. As a whole, this process can be divided into 4 parts: code quality, testing, collaboration and client feedback. The first is to ensure that all project members follow the same high standards for code quality and style, to ensure high code quality, consistency and readability. The second is testing, to ensure the code behaves as expected, even under uncommon situations. The third is the review and approval by other group members before changes are merged into the project's main branch. The last is to ensure the project work is always moving in a direction that the client approves of to ensure client satisfaction.

7.1 Code Quality

Consistency in code quality and style is important to keep a structured code base. If each individual programmer uses their own unique style, even if it results in high quality code on its own, the lack of consistency between members would hurt the overall structure and quality of the project as a whole; it may even lead to misunderstandings and higher chances of creating *Spaghetti-code*. This section will describe the various tools and conventions which the group used to ensure high quality and a common style.

7.1.1 Conventions

Coding conventions create a consistent look and make it easier for readers to focus on the content instead of layout; it enables the readers to understand the code more quickly by allowing them to make assumptions based on previous experience [22].

The project group decided early to use Microsoft's coding convention for C# [22] as it demonstrates common best practices, specifically for C#. Furthermore, this is probably the most commonly used convention for all C# projects, which in turn

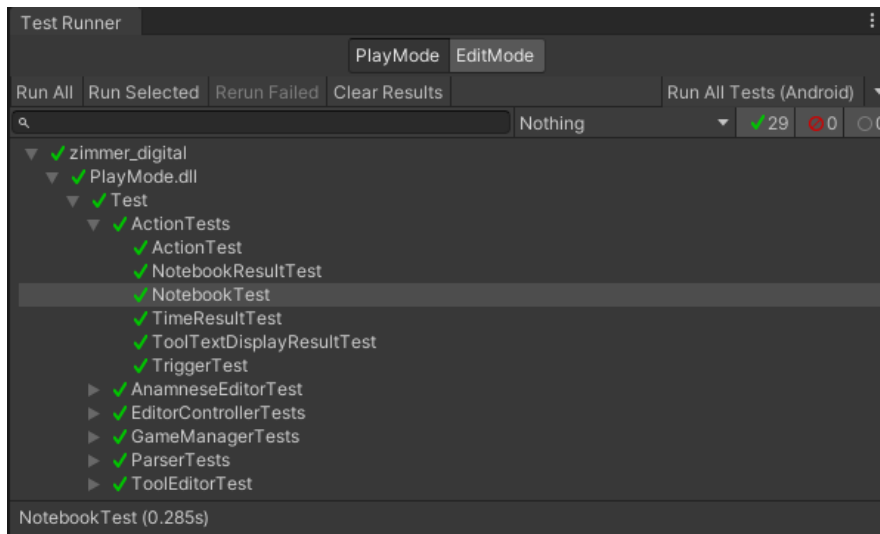


Figure 7.1: The Test Runner window showing a number of passed tests.

includes Unity-projects. In short it includes naming conventions – such as the use of Camel case or Pascal case – as well as language guidelines.

7.1.2 Documentation

After having decided to use Microsoft’s coding convention for C# [22] as described in section Section 7.1.1, the group chose to stay consistent and followed Microsoft’s documentation standard too, [27] by using XML tags as per Microsoft standards.

7.2 Testing

7.2.1 Unit & Integration Testing

Automated testing in the Unity Game Engine is done through the Unity Test Framework [28]. The Unity Game Engine runs in two main modes, Play-Mode and Edit-Mode, and as such has two distinctly – though vaguely-similar – types of tests [29]. They are both run through the Test Runner window, which is opened by selecting *Window* → *General* → *Test Runner*, as shown in Figure 7.1.

All tests in the product is saved under the *Tests* folder, as shown in Figure 7.2.

Edit Mode Tests

Edit mode tests (also known as Editor tests) are run in the Unity Editor outside of Play Mode. They are the closest resemblance of a traditional Unit Test out of the two types of tests in Unity [30]. An example can be seen in Code listing 7.1.

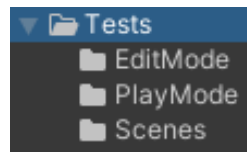


Figure 7.2: The tests are located in the *Tests*-folder under *Assets*.

Code listing 7.1: Example of an Edit Mode test.

```

1 namespace Test
2 {
3     public class ActionTests
4     {
5         // A Test behaves as an ordinary method
6         [Test]
7         public void ConstructorTest()
8         {
9             Action action = new Action("name", Tool.ToolID.otoskop, 10, null, false
10                , Action.RequiredTag.ArmHøyre, EditorDataParser.Correctness.Level.
11                Wrong);
12            // Using the Assert class to test conditions
13            Assert.AreEqual("name", action.name);
14            Assert.AreEqual(Tool.ToolID.otoskop, action.parentID);
15            Assert.AreEqual(10, action.timeToComplete);
16            Assert.AreEqual(null, action.results);
17            Assert.AreEqual(false, action.background);
18            Assert.AreEqual(false, action.background);
19            Assert.AreEqual(Action.RequiredTag.ArmHøyre, action.requiredTag);
20            Assert.AreEqual(EditorDataParser.Correctness.Level.Wrong, action.
21                correctness);
22        }
23    }
24 }

```

Edit Mode tests are marked with the `[Test]` attribute and are always of return type `void`. Classes that inherit from `MonoBehaviour` are usually not suited for Editor tests, as they are attached to `GameObjects`, which rarely exists outside of Play Mode. Such scripts usually requires a Play Mode test instead.

Play Mode Tests

In Unity, Play Mode tests resembles integration tests and run in Play Mode (see Code listing 7.2). They are marked with the `[UnityTest]` attribute and defined as a *coroutine* [30]. They are useful for testing that objects act as expected in the given scene and to exercise game code. Almost all of non-manual software tests are Play Mode tests; additionally, most of our game logic is subsequently tested using Play Mode tests.

There are certain aspects of gameplay that requires real-time to pass before we

Code listing 7.2: Example of a Play Mode test.

```

1  /// <summary>
2  /// Tests if GameManager was setup correctly after loading the scene,
3  /// and that it has the expected behavior.
4  /// </summary>
5  /// <returns></returns>
6  [UnityTest]
7  public IEnumerator InitTest()
8  {
9      Assert.IsNotNull(GameManager.Instance);
10     Assert.IsNotNull(GameManager.Instance.gameData);
11     Assert.IsFalse(GameManager.Instance.gameData.gameOver);
12     Assert.AreApproximatelyEqual(maxTimeToComplete, GameManager.Instance.
13         gameData.timeRemaining, 0.2f);
14     yield return new WaitForSeconds(2);
15     Assert.AreApproximatelyEqual(maxTimeToComplete-2, GameManager.Instance.
16         gameData.timeRemaining, 0.2f);
17
18     // We have waited more than the max allowed time. Should be game over
19     yield return new WaitForSeconds(maxTimeToComplete-2);
20     Assert.IsTrue(GameManager.Instance.gameData.gameOver);
21 }

```

Code listing 7.3: Example of a UnitySetUp. This is executed before every test in the same script.

```

[UnitySetUp]
public IEnumerator Setup()
{
    yield return SceneManager.LoadSceneAsyncInPlayMode("Assets/Tests/
        Scenes/MainEditorSceneTest.unity", new LoadSceneParameters(
        LoadSceneMode.Single));
}

```

can test for the expected behavior, however, the code is read at a per-frame basis. This is why a *UnityTest* is defined as a *coroutine*, as that allows us to delay parts of the test until the game code has exercised completely. This delay can be everything from a few frames (or many times simply until the end of the current frame) to several minutes.

Play Mode tests usually requires some sort of preparation before it can be run, like loading the correct scene into the Play Mode environment, before testing. This can be done in several ways, one of which is using the *[UnitySetUp]* attribute, like in Code listing 7.3. A method marked with the *[UnitySetUp]* attribute is automatically called before every test in the script, however, this may be unnecessarily demanding and resource-heavy especially if there are many tests in the script.

If the setup is only necessary to do once, then one may consider using a *[One-TimeSetUp]* instead, as shown in Code listing 7.4. The only drawback to this, compared to a *[UnitySetUp]* is that this type of setup is only done once, therefore one

Code listing 7.4: Example of a OneTimeSetup. This is only executed once for the given test script, before any of its tests.

```
[UnityEngine.TestTools.OneTimeSetUp]
public void Setup()
{
    PrepareCaseData();
    SceneManager.LoadSceneAsyncInPlayMode("Assets/Tests/Scenes/
        ActionTestScene.unity", new LoadSceneParameters(LoadSceneMode.Single));
}
```

test may accidentally affect the results of another.

7.2.2 Manual Testing

Manual testing is almost always useful, though it may be more time-consuming to always be in need of manual tests to ensure that a small change one place doesn't break something another place during development, especially on larger projects. However, a manual test is usually more flexible, simpler and easier to do, and it may better reflect actual practical use of the software.

The group used manual tests during development, especially before and after a merge. Furthermore, the group used manual tests on everything that was unsuited for other forms of tests, like testing out that the VR tools worked with the application and how the VR gameplay felt.

Apart from what each individual programmer would do on their own, manual testing was also done systematically through reviews (see Section 7.3.3) as part of the Scrum development process. As seen in Figure 5.1, tasks are moved from *In-Progress* to *For-Approval* when it is considered finished by the individual member working on it. After it has been manually reviewed and approved by the group, it is moved to done and the change is merged into the main branch.

7.2.3 User Testing

User testing is an important part of the development process in order to test the usability and design of the product. The tests were conducted to give us an insight into the target users, how they use the application, potential issues with the design and suggested improvements.

Method

The user research technique chosen for the testing was evaluation. This technique is more contextual and requires fewer users, allowing us to test the product in a more relevant context whilst not being too resource demanding.

The evaluation method used was a usability test, chosen because it is very good at

evaluating the design's ease of use and potential design flaws. The user was asked to perform a series of tasks, targeted at testing specific use cases. A Think-Aloud Protocol was used, where the user conveys what they are thinking as they are performing the tasks, to provide valuable insight into a user's thought process.

A test would conclude with the user giving general feedback and answering questions to expand on observations made during the test.

Due to limitations the testing was conducted over Teams using screen sharing, and the VR gameplay testing had to be replaced with a walkthrough of the demo video [23]. Despite this, the testing provided valuable results that should be considered for future improvements.

Results

Results are found in Appendix D.

7.3 Collaboration

7.3.1 Version Control

The group used the NTNU GitLab for version control. Rules for use of the repository was defined in Appendix G. This was particularly useful as it set a high standard for the workflow and also set measures to ensure that the repository was handled in a secure way. Furthermore, risk mitigation and general security and repository Section 1.2 health was ensured by following common security principles, such as *Separation of Duties* and *Least Privilege*.

7.3.2 Protected branches

As described in the Project Plan (Appendix C), the main branch was always protected; this was done to be in accord with the security principle of Separation of Duties. However, since we were a relatively small group of only three students, it meant that there were always only two people with approval rights. Therefore, we felt that a single non-author approval was enough and a healthy balance between security and practicality.

7.3.3 Reviews

At the start of each iteration during the iteration planning meeting, the group would review changes from the previous iteration. If said changes met the required standards and quality that the group expected, then the changes would be approved. Approval of merge requests was also done during this meeting, but it could also be done earlier *during* an iteration if it included changes necessary for the iteration, but same rules would apply.

7.4 Client Feedback

To ensure the project work always were heading in the right direction, the group had weekly meetings with the client where we could discuss problems and solutions, gain feedback on the previous iteration and learn more about the client's needs and visions.

The group had a meeting with the client at the end of the development period where the final results of the Unity project were presented, alongside two videos that was prepared beforehand, available here: [31] [23]. During this meeting, the client affirmed that they were satisfied with the results, and positively surprised to what we managed to do in such a relatively short amount of time given the scope of the task.

Chapter 8

Deployment

As the Unity Project currently stands, it cannot be built into an executable as it makes use of the *EditorUtility* package, which is only available in the Unity Editor. This package is what allows us to open the File Explorer to select a JSON file. To see the current software, one will instead need to open the project as a Unity Project, which is the development environment.

In the future, one must remove all usage of the *EditorUtility* package. Since the software itself only need to know the path to the file to be able to read it, the *EditorUtility* package is not needed for the software to work, but it is an easy and quick way to select path to said files during development. This should not be an issue for the client, however, as the client is interested in connecting the application to a database either way (see Section 9.5.1).

8.1 Unity Project Setup

To try out this project, you will need to open the project in the Unity Engine. Steps:

1. Download Unity Hub.¹
2. Install the correct version of unity from the Unity Hub window by clicking *Installs*, then, *Add*, then select Unity 2020.3.26f1.²
3. Download the project from our repository Section 1.2 and open it from Unity Hub.
4. Download the hospital asset pack [26] from the Unity Asset store and import it into the project (*Licence required*). You can also manually add the asset pack by dropping it under the *Assets*-folder if you already have it downloaded.

¹<https://unity.com/download>

²We cannot guarantee that the project works as intended in other versions of Unity.

5. Materials need to be upgraded to use Universal Render Pipeline.³

Unity should now be setup to test and work on the project. If an Oculus headset is setup and connected (see 8.2), the *Play* button can now be used to play test the gameplay.

8.2 Oculus Development Setup

In order to play and test the project in Unity with VR, you need a connected Oculus headset configured for development. A Oculus developer account is needed. The headset must be connected to your PC either through a USB cable or over WiFi. A step-by-step guide for this setup is provided by the Oculus Developer documentation. [32]

8.3 Standalone VR Game for Oculus

The project is currently configured with the optimal build settings for Android, which is the platform Oculus runs on. To get an executable file that can be ran on an Oculus headset, you will need to build the project. Open the File menu, go to Build Settings and click Build.

If a build excluding the editor is wanted, you will need to change two things. Remove the *MainEditorScene* from the build settings. The *Opprett kasus* button in the *MainMenu* scene also needs to be disabled by selecting it in the Hierarchy and disabling it in the top-left corner of the Inspector.

8.4 Case Editor for Web

In order to build a runnable case editor for web, the project build settings needs to be changed to web. Similarly to the previous Oculus build, two changes must be made. Remove the *GameplayGP* scene from the build settings. The *Spill* button in the *MainMenu* scene also needs to be disabled by selecting it in the Hierarchy and disabling it in the top-left corner of the Inspector.

8.5 User Manual

The three main scenes – main menu, case designer and gameplay – are located in *Assets/Scenes/Main* with the file extension *.unity*.

All scripts that are used in the current implementation are located in *Assets/Scripts/Core*.

³Edit → Render Pipeline → Universal Render Pipeline → Upgrade Project Materials to UniversalRP Materials

8.5.1 Further Development

Adding more tabs in Case Editor

If you open the main editor scene, and find the `GameObject` named *DiagramTabs*, here all the tab buttons should be added. You can simply duplicate one of those that are already there and edit it slightly. Then you can add the window panel, which should be added as a child to the Diagram Window parent. You see that each tab window has the component `CaseTabWindow` class. Make sure to drag the correct tab button you just duplicated into the inspector field of *Button* in `CaseTabWindow`. Your window should now be set up and be opened when the tab button is pressed.

Additional Game Editors

To add a new editor, create a new class and make it inherit from the abstract class: `GameEditor`. This will enforce that you define the required methods for the editor to work within the system. If the editor will be changing new datatypes, these will have to be added on the `ListNodeData` class in `EditorDataParser`. Navigate to the `ListNodeData` class and add a new list of the datatype into the class.

If the editor needs a new category that is not previously defined, this must be added in the enum: `Category` in `ListCategoryNode`.

Both of the current editors are written as singletons, it is therefore recommended that new `GameEditors` also are singletons. Once an editor script has been made, create a user interface that can display the data, and that allows the user to edit them. It is important to remember that behaviours from the script like `add`, `set` and `finalize` must be manually attached to user interface elements on the scene view. Once the user interface is completed, it can be put on the scene as a child of `GameObject` named `EditorController`.

The next step is to go in the `EditorController` script and add the name of the new editor into the enum: `Editors`. Next, add the editor into the load function, simply define the switch case for the new editor enum-entry and set the `GameObject` linked to the script to active.

If the editor is changing new datatypes, create a new overload for the `SetNodeData` method. The final step is to change the `SelectNode` method, simply add the switch case for the wanted `ListCategory`, and tell the `EditorController` to load the node with the correct editor type as seen in Code listing 8.3.

Adding tools

Adding a new tool can be done by adding a 3D model of the tool and adding the required components. The object needs to have a collider, an `XRGrabInteractable` and a `Tool` component.

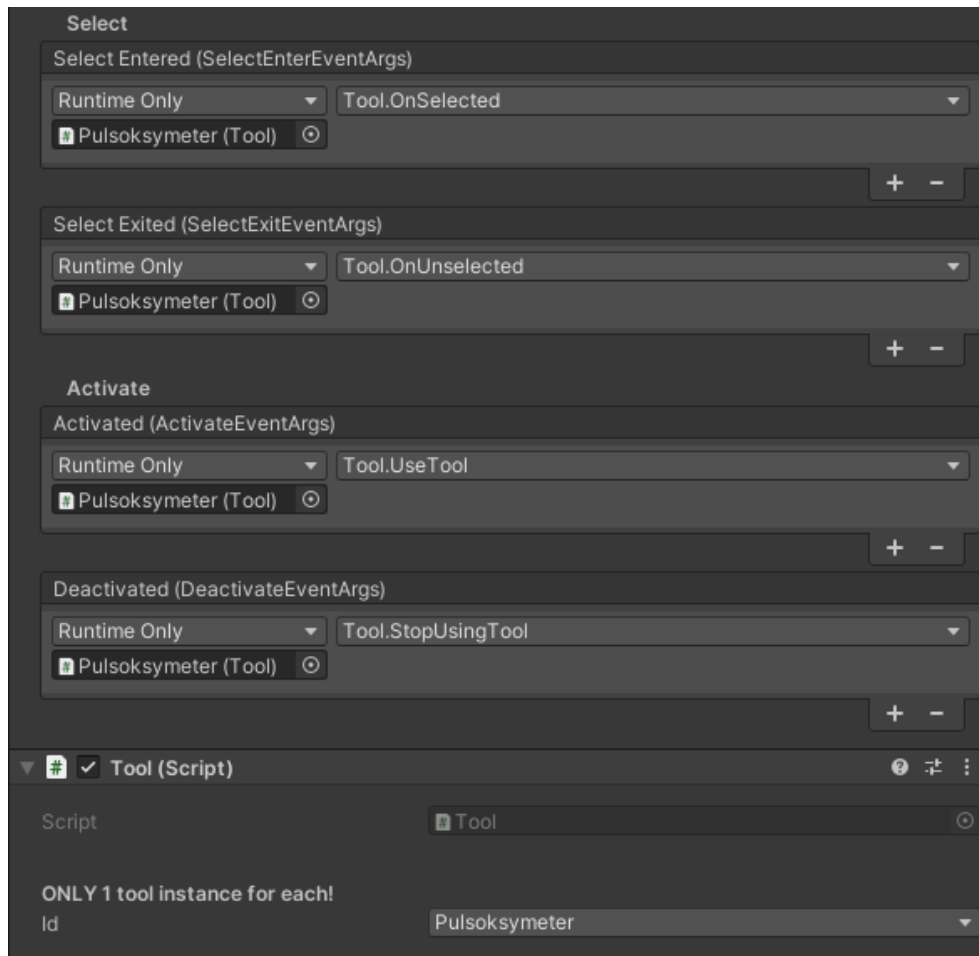


Figure 8.1: Tool component configuration.

The `XRGrabInteractable` needs to be correctly configured with the correct events. Figure 8.1 shows how this should be setup. Events need to be added as shown to the `Select` and `Activate` categories of the component. `Movement Type` should be set to `Kinematic`. Enabling `Smooth Position` and `Smooth Rotation` is recommended.

The collider can be either in the parent object or a child object, and will be fetched automatically by the `XRGrabInteractable` component. The tools `Rigidbody Collision Detection` needs to be set to `Continuous Speculative`.

A new enum value needs to be added to the `Tool.ToolID` enum (Code listing 8.1), every tool required a unique id. Lastly set the id of the `Tool` component to the newly created `ToolID`.

Code listing 8.1: Tool script showing ToolID enum.

```

1 public class Tool : MonoBehaviour
2 {
3     /// <summary>
4     /// This is to tell which <see cref="Tool"/> this is.
5     /// There should always be exactly one <see cref="Tool"/> <see cref="GameObject
6     /// </summary>
7     public enum ToolID
8     {
9         blodsukkerapparat,
10        blodtrykksapparat,
11        oftalmoskop,
12        otoskop,
13        pulsoksymeter,
14        spirometri,
15        stetoskop,
16        termometer,
17        pc
18    }
19    ...
20 }

```

Alternatively, a new tool can be created easily by making a copy of an existing tool, unpacking the prefab, renaming the `GameObject` and changing the `Id` to a new `Tool.ToolID`. Then you only need to change the mesh and collider to have it look how you want. Existing tools makes this easy by having them as a separate child object, named *Mesh*, that can be freely replaced.

Adding triggers

Additional triggers can be added to allow tools to be used where a trigger is missing. This required adding a new enum value and placing the 3D object in the scene.

To add the new trigger enum value, open the *Action.cs* script and find the `RequiredTag` enum. Add a new value to the enum. Changing the integer values of existing values will break existing json case files. All values are hard set to a number to avoid it changing accidentally when adding new values.

To add the 3D trigger, go to the *GameplayGP* scene and find the Triggers object. This object has the `TriggerController` script and controls all `ActionTriggers`. An `ActionTrigger` must be a child object of Triggers.

Create a new child object of Triggers. Set the gameobject to have the tag `ActionTrigger` in the inspector. Set the object material to `HitboxGlow`. Add a collider if none is present, set the collider's *Is Trigger* value to checked. Add a `ActionTrigger` component and set its `Trigger Tag` to the new enum value. Add all three `HitboxGlow` materials to the `ActionTrigger` component.

Result	Activation Function Implementation
Orientate patient	Change the orientation on the GameObject holding the patient.
Patient talk	Play a premade sound file, or implement a text to voice system that can read out text.
Change View	Changes the view of the user to somewhere else. This could be changing the view to look directly into the ear, allowing the user to get information based on what they see inside the view. Alternatively, it could load a picture of an ear.

Table 8.1: New result type examples.

Adding result types

Additional results can be implemented by creating a new class that inherits from the `Result` class. The result class is an abstract, simple class that forces the inheritors to define an activation function, seen in Code listing 8.2.

Code listing 8.2: The `Result` class.

```

1 public abstract class Result
2 {
3     /// <summary>
4     /// <see cref="GiveResults()"/> is the method that gives feedback in the game
5     /// from doing actions.
6     /// i.e. Adds a bullet point to journal, change patient properties and turn on
7     /// machines
8     /// </summary>
9     public abstract void GiveResults();
10 }

```

There are no restrictions on what the activation function can do, it is this component that will create new changes in the game environment. Examples of new results that can be implemented in the future can be seen in Table 8.1.

It is important to note that new results will also have to be implemented in the Case Designer. New case data types must be made to hold the data, and user interfaces for inputting values must be made in the Tool Editor. The value fields in the user interface must additionally be added to the *Result Parser* that resides within the Tool Editor `GameObject`.

Implementing remaining categories

Implementation on remaining categories is different if they need a new editor. If this is the case, follow the steps outlined in Section 8.5.1.

If the category does not use a new editor, it needs to define the switch case, saying which editor to load in the *ListNode*'s function: *Select node*. The switch case can be seen in Code listing 8.3. Furthermore, it needs to be added in the *Set Node* function found in *ListNode*, as this function controls which categories of nodes can be created.

Code listing 8.3: Node selection in *ListNode.cs*.

```
1  /// <summary>
2  /// Selects the pressed <see cref="ListNode"/> to open the relevant Editor
   based on the <see cref="Parent"/>'s <see cref="ListCategoryNode"/>
3  /// </summary>
4  public void SelectNode()
5  {
6      switch (Parent.GetCategory())
7      {
8          // TODO: Implement this... Open the correct editor
9          case Category.Anamnese:
10             EditorController.Instance.Load(this, EditorController.Editors.
               Anamnese);
11             break;
12         case Category.Clinical_US:
13             EditorController.Instance.Load(this, EditorController.Editors.
               Action);
14             break;
15         case Category.Tests:
16             EditorController.Instance.Load(this, EditorController.Editors.
               Action);
17             break;
18         default:
19             break;
20     }
21 }
```

Importing actions to tool

The *Tool.cs* script automatically imports the actions with the same *ToolID* to the tool from the *GameManager.Instance*.

Assembly Definitions

The code in this project uses custom assembly definitions. It is not paramount that you know everything about how these work in Unity, but important to note a few key aspects; Assembly definitions are essentially used by the compiler to compile the scripts. If you are a future developer who is going to work with this project later and you are experiencing import warnings and missing assembly exceptions, it may be because you haven't set up the assembly definitions or its references correctly – as this is normally done automatically by Unity, but with custom assemblies they are not. Read more about assemblies in the Unity manual. [33]

Chapter 9

Discussion

9.1 Evaluation of Objectives

As described in Section 1.5, the group mapped out a series of goals and objectives for this project. This was divided into three groups of objectives: learning, impact and outcome.

9.1.1 Learning Objectives

The group was happy with how it used the Scrumban methodology during development. We were also happy with how we adapted the methodology to our needs. An example of this can be seen in Planning Meetings, where the planning meetings were moved to Mondays to adapt to.

The group is also happy with how much they have learned about Unity and game development. It was an exciting journey, being able to use such Unity, that offered so many tools and using the tools to realize our product. The group also got their first-hand experience on creating applications for Virtual Reality (VR) and the group was pleased with how the functionalities, only available with VR was used to create a VR experience.

9.1.2 Impact Objectives

The group succeeded in creating a data form that would make saving and exporting cases easy. Using JSON, the case data is now stored in a readable form which also can be easily stored in a future database. The group is happy with the complexity of creating a case.

The user is given the opportunity to define where examination takes place, what tool is used for the examination and what will happen after the examination. These options are shown displayed in a way that only allows the user to choose existing locations, tools, and results. This allows the case creators to use their

medical knowledge to create realistic examinations while not exposing them to the complex behaviours happening in the logic of the software.

The new method (see Section 6.3.1) for making cases is a drastic improvement to producing more cases. The functionality to load cases into the editor allows the use of templates, which can define the neutral examinations, allowing the case developer to only define the important examinations that leads to a diagnosis. There is no limit to how many templates can be created as any case can be loaded to the case creator. It is therefore simple to create similar cases with small variations, further reducing time spent per case.

9.1.3 Outcome Objectives

Considering the requirements outlined in Chapter 3, the group has succeeded in creating a product that the client can expand or implement into their future product. The group succeeded in creating a MVP that met the objective of creating a playable Virtual Reality (VR) experience. It also succeeded in creating a framework that will make expansion of the product easier.

Unfortunately, the project was not concluded with a build of the project. However, this is not something that the group should be unsatisfied with as it only would have been a bonus for showcasing the product. In its current state, it is not ready to be released for commercial use anyways, but is still fully playable within the Unity Editor. The changes mentioned in Deployment to make the build possible will also be simple when the client develops a database for case selection.

9.2 Requirements

All the functional and technological requirements described in section Section 3.1 and Section 3.2 were implemented, although, perhaps not to the full extent that we initially envisioned; it is currently possible to create and edit cases and play the application using VR, but it needs further development before it is ready for release. The exact details is described in section Section 9.4.1 and Section 9.5.

The application itself worked very well in Norwegian, despite all behind-the-scenes logic and code being written in English. Considering the requirements, goals, scope, all the challenges with a project like this and all the time and hard work the group put into this project, it would be unreasonable for the group to be unsatisfied with our project results. The goal was never to make a fully-fledged application, but rather a scalable framework for further development, which is something the group delivered on.

9.3 Development Method and Process

The implementation of the development method worked well overall; a combination of weekly meetings, sometimes several, and individual work encouraged high effectivity and involvement as long as each individual member stayed motivated and did what was agreed upon, especially when it relates to daily work-ours.

The choice to use an agile development method was probably the correct choice for this project; as the project group had little to no prior medical knowledge or experience, we therefore had to learn and plan the exact details during the development process, and an agile development method supported such work.

Nevertheless, it should come to no surprise that the project task was challenging, given our limited knowledge on the subjects, and the sheer size of the scope – which we already had shaved down quite a bit from the client’s original dreams. The fact that the client only shared the visions of a larger dream/goal and no specific requirements and allowed the group the amount of freedom as they did may have been more of a challenge than a benefit, especially considering the group’s limited medical knowledge, as it was difficult to develop something we know so little about.

9.3.1 Time Use

According to plans, each project member where expected to work 30 hours a week. However, these plans where adjusted to around 18-20 hours a week, to give more time for hour second course/subject we had this semester, which went on for full until its exam 23.3.2022. Additionally, the thesis was completely put on pause during the week of the exam in this other course.

This meant that the Project Plan Period and until iteration 8 of the Development Period was at reduced work-hours. With this in mind, the group – according to the initial plan – was expected to work up to around 900 hours, not accounting for work with the project report itself; the group was around 200 hours off from this plan. On the other hand, the group was careful with only registering time when we were actually *working*, and working from home using Toggl made that fairly easy, even if we wanted to take breaks or divide the daily work-hours up throughout the day.

The report was planned to be written in the last 3 weeks of the project period. The group was expected to work 270 hours in total on this. The group spent 187 hours in total on the report. This was the considered fine as the group managed to conduct peer reviews and reviews from the supervisor while also making the proposed changes.

9.3.2 Internal Meetings

The organization of internal meetings could have been better; we had loosely planned weekly meetings, where the time varied a lot, and where the time where often not set before late the day before, or sometimes even a few hours before the meeting on the same day. We went with a very *on-demand* method for additional meetings too, but it all worked fairly well, since the group members had quite similar circadian rhythm and where often online on Discord during this time. However, if it hadn't worked so well, we probably would have adapted accordingly and found a better way to organize meetings.

9.3.3 Testing

The software itself have been tested extensively through both manual and unit-like tests, and so the group is confident in the software's robustness; if the project where to be changed as a consequence of further development, the code and scripts that are already written should still work as intended unless they are modified. The fact that we sat up testing environments and files separate from the actual project files, means that even if the project were to change, the tests should still pass as before, and if not, the issue lies probably with changes other potential programmers outside the group has done, but the tests should still give some tips to where so they can locate their mistakes.

9.4 Application Design

The client wanted an application with a simple design that was easy-to-use by practically anyone. Since the task itself was relatively complex, it was indeed challenging to come up with a way to convey it to the user in a way that it didn't feel like it was. Despite the challenge, we the group believe we struck a fine balance between the necessary complexity and simplicity, though the final graphical design probably needs some more final tweaks before a full release.

On the other hand, the project group believes that the current application's design is a good concept and starting point for further development.

9.4.1 Minimum Viable Product

The project group's final product, the Unity project, should be considered a MVP and a starting point for further development as some features are yet to be implemented. However, its logic is of relatively good quality, fairly Scalable and its bugs should be fairly limited.

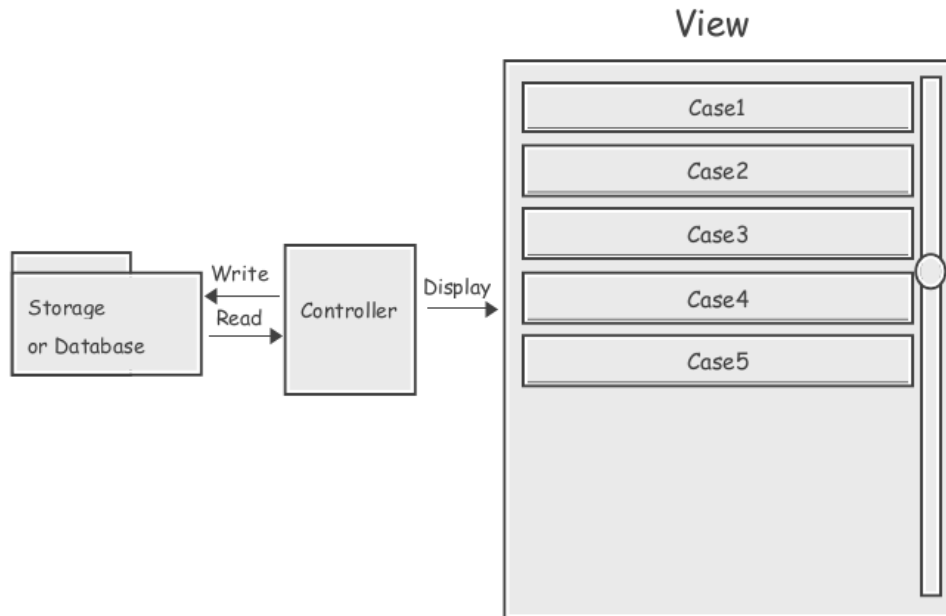


Figure 9.1: Illustration of how one may connect the application to a storage or database instead of using the File Explorer as currently done by using an application view.

9.5 Further Development

9.5.1 Databases and File Sharing

To build the Unity Project into an executable, the `EditorUtility` package cannot be used to find files using the File Explorer. However, this is of no issue if the client intend to implement a database features later anyway.

A possible solution could then be to remove any usage of `EditorUtility`, and instead let the application read from a database and create a separate in-application view that visualize the files in said database as illustrated in Figure 9.1. The minimum data the view needs to store, is the path to each available file in the database, so that when the button is pressed, it can read the file by going to said path.

9.5.2 Order of Actions

The client stated that in most cases the order of which the player performed the available actions would matter, sometimes even quite significantly. And doing the same action multiple times may give different results; like giving the patient medicine several times would probably not have the same effect every time – they may even die if given an overdose. Although the project group did not get the time to implement such a feature, we had nonetheless worked out some plans for possible

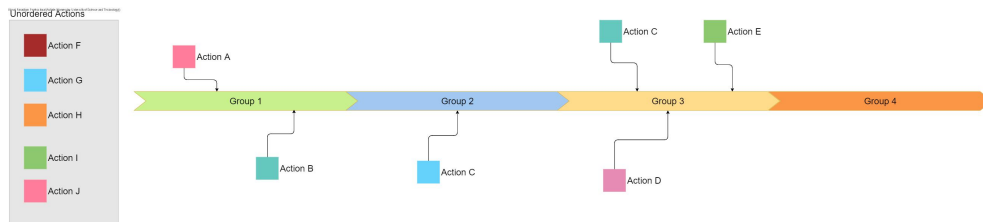


Figure 9.2: The *Timeline*-approach to sort actions in the desired order. As illustrated by the diagram, actions are first displayed in a view or scrollbar represented by the *unordered actions* window, then they can be dragged into a group. The order of each group matter, however, the order of each action inside a group may not.

solutions for this very problem.

Phases and Milestones

Our first idea to solve this problem is to divide the game loop into smaller sections known by the team as *phases*. Each phase would then have a list of milestones that needed to be performed before the player could progress to the next phase (see Figure 4.1). However, this would prove to be in conflict with our design goal of simplicity, as the way we thought to implement it required a new tab for each phase where actions would be duplicated between each, cluttering the screen and likely confusing the user.

Timeline

In the *Timeline*-approach (see Figure 9.2), the Case Designer only needs one additional tab, and the ones that do already exist can stay mostly as they already are. It is logically quite similar to the Phases and Milestones approach, as described in Section 9.5.2, only that it is visualized for the user in a simpler, more compact way.

As illustrated by Figure 9.2, all actions that are added in the *undersøkelser* tab described in section Section 6.3 can simply be loaded into a side-menu (represented by the *Unordered Actions*), where the user can select the action and drag it into the timeline-window to the desired group. In this approach, only the order of the groups matter, but not the children of the groups; therefore, if there is a collection of actions where the children of a group between themselves don't matter, but it matters that all of them is done before some other group, then simply drag them into the same group.

When it comes to the relevancy or correctness level of each action, there are probably several possible solutions when using this approach, here are some ideas:

- Allow the creation of multiple timelines, and mark the timeline with a rel-

evancy level. Everything that happens outside of the timeline may be considered wrong.

- Allow more levels of groups so that every situation is accounted for in its own level if necessary.
- Unordered actions – that is, actions that is listed in the *unordered actions* view – have no particular order and thus can be done whenever the player feels like it. They may have relevancy level *neutral*.

Chapter 10

Conclusion

The benefit of training simulations is not just limited to the medical field, they can be a great tool for many fields and professions that require practical training. They offer a new, exciting, and practical arena where students can practice in a safe environment. By using simulations, the learner can make mistakes without causing real damages to people or their environment. It also increases the availability to practice, as the game can be played at home, at school and wherever the user can connect to the game.

In this project we were assigned to improve the speed at which cases can be created. Starting from an empty Unity project, the group has developed a product that offers an environment where medical students can practice their practical skills by conducting examinations, tests, and setting a diagnosis on a patient. The product also offers an easy-to-use case designing tool where medical professionals can make cases for their students to play through. The group refined the requirements to ensure that the final MVP would meet the expectations of the client. This worked out well, resulting in a satisfied client as seen in: Client Feedback.

The Case Designer is a good example that additional tools for creating scenarios is a great way to increase simulation variety while simultaneously decrease the time spent setting up each scenario. It also increases the accessibility, allowing contributors to create scenarios without the need of any prior programming experience.

The product still has features that is needed before the product is commercially ready, seen in Further Development. The product was developed with this in mind, and the accommodations made should hopefully make the implementation simple.

Bibliography

- [1] *C++*, 2022. [Online]. Available: <https://en.wikipedia.org/wiki/C%5C%2B%5C%2B> (visited on 12/05/2022).
- [2] *C sharp*, 2019. [Online]. Available: https://no.wikipedia.org/wiki/C_Sharp (visited on 12/05/2022).
- [3] *Unity - scripting api: Gameobject*. [Online]. Available: <https://docs.unity3d.com/2020.3/Documentation/ScriptReference/GameObject.html> (visited on 05/05/2022).
- [4] *Java (programmeringsspråk)*, 2021. [Online]. Available: [https://no.wikipedia.org/wiki/Java_\(programmeringsspr%C3%5C%A5k\)](https://no.wikipedia.org/wiki/Java_(programmeringsspr%C3%5C%A5k)) (visited on 12/05/2022).
- [5] BeyondTrust, *Least privilege*. [Online]. Available: <https://www.beyondtrust.com/resources/glossary/least-privilege> (visited on 04/05/2022).
- [6] Information Technology Laboratory, *Seperation of duty (sod)*. [Online]. Available: https://csrc.nist.gov/glossary/term/Seperation_of_Duty (visited on 04/05/2022).
- [7] techopedia, *Spaghetti code*. [Online]. Available: <https://www.techopedia.com/definition/9476/spaghetti-code> (visited on 04/05/2022).
- [8] P.Guilizzoni, *What are wireframes?* [Online]. Available: <https://balsamiq.com/learn/articles/what-are-wireframes/> (visited on 29/04/2022).
- [9] M. Zimmermann, 'Vr som pedagogisk prinsipp på medisinstudier – en visjon,' M.S. thesis, UiO, 2021.
- [10] S. Vardomatski. 'Virtual reality simulations in healthcare.' (2022), [Online]. Available: <https://www.forbes.com/sites/forbestechcouncil/2022/01/24/virtual-reality-simulations-in-healthcare/?sh=2943c9b6382a> (visited on 05/05/2022).
- [11] Norsk Pasientskade-Erstatning (NPE). 'Årsrapport 2020 - norsk pasientskadeerstatning.' (2020), [Online]. Available: <https://www.npe.no/globalassets/dokumenter-pdf-og-presentasjoner/arsmelding/2020/arsrapport-2020.pdf> (visited on 06/05/2022).

- [12] K. Jubbal. 'Medicine in old days vs now.' (2019), [Online]. Available: <https://medschoolinsiders.com/medical-student/medicine-in-old-days-vs-now/> (visited on 09/05/2022).
- [13] E. Sweeney. 'Harvard researchers: 'absurdly outdated' medical education needs more emphasis on analytics.' (2017), [Online]. Available: <https://www.fiercehealthcare.com/analytics/medical-education-nejm-technology-analytics-artificial-intelligence-machine-learning-data> (visited on 09/05/2022).
- [14] Grimstadutvalget, 'Studieplasser i medisin i norge,' p. 68, Sep. 2019.
- [15] J. Lopreiato and L. Lioce. 'Virtual reality simulations in healthcare.' (2020), [Online]. Available: <https://www.ahrq.gov/patient-safety/resources/simulation/terms.html> (visited on 05/05/2022).
- [16] J. Lopreiato. 'How does health care simulation affect patient care?' (2017), [Online]. Available: <https://psnet.ahrq.gov/perspective/how-does-health-care-simulation-affect-patient-care#ref1> (visited on 05/05/2022).
- [17] *Unity (game engine)*. [Online]. Available: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) (visited on 05/05/2022).
- [18] W3schools.com, *Json - introduction*. [Online]. Available: https://www.w3schools.com/js/js_json_intro.asp (visited on 03/05/2022).
- [19] B. Marr. 'What is extended reality technology? a simple explanation for anyone.' (2019), [Online]. Available: <https://www.forbes.com/sites/bernardmarr/2019/08/12/what-is-extended-reality-technology-a-simple-explanation-for-anyone/?sh=5f27201c7249> (visited on 03/05/2022).
- [20] Deadcows, tonygiang, wqyeo, wilsnat, Martian-Child, TheWalruzz, karsion, schodemeiss, favoyang, AdrienVR, j-jorge, CrizGames, derfium, rfadeev and AminSojoudi, *Mybox*. [Online]. Available: <https://github.com/Deadcows/MyBox> (visited on 28/04/2022).
- [21] A. Rumak, *Mybox license*. [Online]. Available: <https://github.com/Deadcows/MyBox/blob/master/LICENSE.md> (visited on 28/04/2022).
- [22] B. Wagner, M. Williamson, N. P. N and P. Kulikov, *C# coding conventions*. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions> (visited on 20/01/2022).
- [23] A. Helgestad, K. A. Rønning and T. V. Lien. 'Zimmer demo vr.' (2022), [Online]. Available: <https://youtu.be/AoA1iXPT16I> (visited on 02/05/2022).
- [24] Unity Technologies, *Json serialization*. [Online]. Available: <https://docs.unity3d.com/2020.3/Documentation/Manual/JSONSerialization.html> (visited on 28/04/2022).
- [25] C. Ladas, *Scrumban*. [Online]. Available: <https://www.agilealliance.org/scrumban/> (visited on 09/05/2022).

- [26] Mixall, *Hospital - modular building, props and characters*. [Online]. Available: <https://assetstore.unity.com/packages/3d/environments/industrial/hospital-modular-building-props-and-characters-131680> (visited on 02/05/2022).
- [27] B. Wagner, M. Arndt, C. Yeleighton and A. Edelen, *Recommended xml tags for c# documentation comments*. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xml/doc/recommended-tags> (visited on 20/01/2022).
- [28] Unity Technologies, *About unity test framework*. [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html> (visited on 27/04/2022).
- [29] Unity Technologies, *Unit testing*. [Online]. Available: <https://docs.unity3d.com/2020.3/Documentation/Manual/testing-editor-test-runner.html> (visited on 27/04/2022).
- [30] Unity Technologies, *Edit mode vs. play mode tests*. [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/edit-mode-vs-play-mode-tests.html> (visited on 27/04/2022).
- [31] A. Helgestad, K. A. Rønning and T. V. Lien. 'Zimmer demo editor.' (2022), [Online]. Available: <https://youtu.be/7nF4eD8C09E> (visited on 02/05/2022).
- [32] Oculus, *Oculus developer hub: Unity*. [Online]. Available: <https://developer.oculus.com/documentation/unity/ts-odh/#set-up> (visited on 02/05/2022).
- [33] Unity Technologies, *Unity - manual: Assembly definitions*. [Online]. Available: <https://docs.unity3d.com/2020.3/Documentation/Manual/ScriptCompilationAssemblyDefinitionFiles.html> (visited on 02/05/2022).

Appendix A

Project Description

Praktisk-teoretisk virtuell klinikktraining for medisinstudenter i Norge

Det er et politisk mål å øke antall studieplasser på medisin i Norge. Dette er ikke mulig slik forholdene på helseforetakene er i dag, da spesielt med tanke på den kliniske delen av studiet. Antall praksisplasser, samt størrelse på kliniske smågrupper som undervises på avdelinger ved dagens universitetssykehus, har allerede nådd et øvre tak. Antall timer med praksis begrenser seg dermed opp mot de praktiske utfordringene.

Firmaet Zimmer Digital AS er et startup-firma etablert av tre studenter og/eller nylig utdannede innen medisin og IKT. Firmaet er i gang med utviklingen av et simuleringsprogram i en virtuell medisinsk verden hvor medisinstudenter må jobbe med virtuelle pasientkasuser. Kasusene går ut på å hente ut informasjon om pasientens tilstand, undersøke pasienten, samt diagnostisere og ta en beslutning på videre behandling. Etter endt kasus vil de få tilbakemelding på gjennomførelsen som også blir lagret på deres bruker for at brukeren kan følge sin egen utvikling.

Vi holder hovedsaklig til i Trondheim/Oslo og ble nylig inkludert i UiO; livsvitenskap sitt inkubatorprogram; Growth House med støtte både økonomisk og praktisk.

Vi har så langt en pek-og-klikk modell klar, men mangler VR-integrasjon, samt flere medisinske kasuser for å kunne nærme oss markedet.

Oppgavens mål

Oppgaven vil bestå i å ta dagens pek-og-klikk modell inn i det virtuelle miljø slik at studenter kan teste programvaren ved bruk av VR-briller. Det er også ønskelig at det programmeres flere medisinske kasuser. Dette være seg medisinske kasuser fra operasjonssalen, i helikopteret etter å ha plukket opp et offer for snøskred, på fastlegekontoret eller på barselavdelingen.

Det benyttes Unity for utvikling av programvaren i 3D og i VR.

Oppgavens krav

Oppbygging av en browserbasert plattform for bruk av den virtuelle virkeligheten med mulighet for å ta i bruk VR-briller dersom ønskelig. Dersom ikke – bruk av mus for panorering.

Programmering av minimum et medisinsk kasus, med dertil visuelt uttrykk og integrasjon for utforsking med VR-briller. Dette vil skje i samarbeid med medisinsk personell på UiO slik at kasusene er realistiske og faglig korrekt. Pedagogisk sett etterstrebes en nettverksstruktur på forholdet mellom presentert symptom og dertil handling som gir feedback fra pasienten. Dette da medisinske kasuser sjeldent utarter seg etter formen symptom-handling-løsning, men er komplekse og ofte uten et riktig svar frem mot kurert pasient eller i ytterste konsekvens død.

Analyse av datasikkerhet i lagring av både brukerdata og selve programvaren, og i valg av støttetjenester. Programvaren skal være i henhold til GDPR.

Vi er åpne for spissing av oppgaven etter studentenes preferanser. Dette kan innebære at omfanget i «oppgavens krav» økes eller reduseres.

Kontaktperson:
Mari Zimmermann
mari@zimmerdigital.no

Appendix B

Project Agreement

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

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

Forklaring av begrep

Opphavsrett

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

Eiendomsrett til resultater

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

Bruksrett til resultater

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

Prosjektbakgrunn

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

Utsatt offentliggjøring

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

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Ivar Farup e-post og tlf.: ivar.farup@ntnu.no , 91695718
Ekstern virksomhet: Zimmer Digital AS Ekstern virksomhet sin kontaktperson, e-post og tlf.: Mari Zimmermann, mari@zimmerdigital.no , 46447033
Student: Thomas Vincent Lien Fødselsdato: 10/10/2000
Student: Kristian Aakervik Rønning Fødselsdato: 28/06/1997
Student: Amund Helgestad Fødselsdato: 10/08/1999

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

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	×
Prosjektoppgave	
Annen oppgave	

Startdato: 10/01/2022
Sluttdato: 20/05/2022

Oppgavens arbeidstittel er:

A Virtual Reality Clinical Education Tool for Medicine Students in Norway

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

3. Ekstern virksomhet sine plikter

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

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

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:
--

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

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven¹. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

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

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

5. Den eksterne virksomheten sine rettigheter

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

Alternativ a) (sett kryss) Hovedregel

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

¹ Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

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

Alternativ b) (sett kryss) Unntak

×	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
---	---

Studentene vil få bruksrett i ikke-kommersielle sammenhenger.

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

Sluttproduktet vil være et kommersielt produkt som er fundamentalt i Zimmer Digital sin videre utvikling.

6. Godtgjøring ved patenterbare oppfinnelser

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

7. NTNU sine rettigheter

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

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

×	Oppgaven skal være offentlig
---	------------------------------

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

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

Sett kryss	Sett dato
<input type="checkbox"/>	ett år
<input checked="" type="checkbox"/>	to år 01.06.2024
<input type="checkbox"/>	tre år

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:
Zimmer Digital har planlagt lansering innen 2024 og ønsker derfor å beskytte produktet mot mulige konkurrenter.

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

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

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

9. Generelt





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

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

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

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

Signaturer:

Instituttleder: _____	dato: _____
Veileder ved NTNU: _____	dato: _____
Ekstern virksomhet:  Mari Zimmermann (Jan 31, 2022 10:47 GMT+1)	dato: <u>Jan 31, 2022</u>
Student:  Amund Helgestad (Jan 30, 2022 14:22 GMT+1)	dato: <u>Jan 30, 2022</u>
Student: 	dato: <u>Jan 30, 2022</u>
Student:  Thomas Vincent Lien (Jan 30, 2022 13:08 GMT)	dato: <u>Jan 30, 2022</u>

Appendix C

Project Plan



Norwegian University of
Science and Technology

DEPARTMENT OF COMPUTER SCIENCE

IDATG2900 - BACHELOR THESIS

**A Virtual Reality Clinical Education Tool
for Medicine Students in Norway**

Project Plan

Authors:

Amund Helgestad
Kristian Aakervik Rønning
Thomas Vincent Lien

January, 2022

Table of Contents

List of Figures	ii
List of Tables	ii
Glossary	iii
Acronyms	iii
1 Project Goals and Constraints	1
1.1 Background	1
1.2 Project Goals and Objectives	1
1.2.1 Process Objectives	1
1.2.2 Impact Objectives	1
1.2.3 Outcome Objectives	1
1.3 Constraints	2
1.3.1 Legal Constraints	2
1.3.2 Technological Constraints	2
1.3.3 Time Constraints	2
2 Scope	2
2.1 Subject Area	2
2.2 Delimitations	2
2.3 Task Description	3
3 Project Organization	3
3.1 Responsibilities and Roles	3
3.1.1 Project Leader - Amund Helgestad	3
3.1.2 Chief of communications - Thomas Vincent Lien	3
3.1.3 Chief of documentation - Kristian Aakervik Rønning	3
3.1.4 Recorder - Kristian Aakervik Rønning	4
3.1.5 Delivery - Thomas Vincent Lien	4
3.2 Routines and Group Rules	4
3.2.1 Working Hours	4
3.2.2 GitLab	4
3.2.3 Meeting Agenda	4
3.2.4 Attendance & Absence	4

3.2.5	Sanctions	4
3.2.6	Decisions	5
4	Planning, Follow-up and Reporting	5
4.1	Development Process	5
4.1.1	Choice of Development Process	5
4.1.2	Use of Development Process	5
4.2	Plan for Status Meetings and Decision-Points in the Period	6
5	Organization of Quality Assurance	7
5.1	Documentation, Standards, Configurations, Tools and Source Code	7
5.1.1	Documentation	7
5.1.2	Standards	7
5.1.3	Tools	7
5.2	Plan for Inspection and Testing	8
5.3	Risk Analysis	8
5.3.1	Risk mitigation and contingency plan	8
6	Plan for Implementation	9
6.1	Timeline	9
6.2	Milestones	10
	Bibliography	11
	Appendix	12
	A Git Repository - Rules and Practices	12
List of Figures		
1	Gantt chart	10
List of Tables		
1	Risk indices	8
2	Risk analysis overview	9

Glossary

MVP A Minimum Viable Product is an early version with only the bare-minimum of features and functionality; it serves mostly as a proof-of-concept for a future product. Despite its limited features, it should be of a high quality without bugs and errors.. 2

Acronyms

3D Three-dimensional. 2, 3, 7

FHI Norwegian Institute of Public Health. 9

GDPR General Data Protection Regulation. 2

IDE Integrated development environment. 7

NTNU Norwegian University of Science and Technology. 7, 9

VR Virtual Reality. 1–3, 7, 9

WIP Work-in-Progress. 6

1 Project Goals and Constraints

1.1 Background

Medical education requires a large amount of practical training and is currently almost exclusively in-person. This is particularly resource demanding. Whilst more students are admitted to universities, the hospitals still have a limited capacity for practical training. This causes group sizes to increase and training amount per student to decrease [1].

Zimmer Digital is a start-up company working on creating a product as a substitute to in-person practical training. The product is a virtual simulation game, where medical students get to practice on virtual training scenarios.

They are currently in the early phases of development, and have made a click-and-point prototype. The patient cases in the prototype are hardcoded, making it very demanding to implement additional cases. Having a large set of both similar and diverse cases is vital to provide a good representation for all the different scenarios they will encounter in their future work as professionals.

1.2 Project Goals and Objectives

The goal of the project is to create an interface that will make it easier and less time-consuming for the client to create and implement new cases, removing the need for programming knowledge for every new case.

1.2.1 Process Objectives

- Conduct weekly meeting with client and supervisor, including planning, follow-up and reporting.
- Become familiar with the Scrumban Framework.
- Learn in-depth Unity and C#.
- Learn about implementation of Virtual Reality, specifically in Unity.

1.2.2 Impact Objectives

- Users will be able to easily create and implement new cases, removing the need for developers to hard-code each new case.
- Reduce complexity for making cases.
- Drastically reduce time spent per case.
- Make case creation accessible to those with little to no programming experience.
- Make saving and exporting cases easy.

1.2.3 Outcome Objectives

- Deliver a product which the client can easily expand or implement into their future product.
- Create a framework and an easy-to-use tool to manage and create new cases.
- Create a playable Virtual Reality experience, or - if not playing in VR - use mouse and keyboard controls.

1.3 Constraints

1.3.1 Legal Constraints

- The product must be GDPR compliant.
- The product must abide by Norwegian law.
- Copyright is owned by the project group.
- Ownership will belong to Zimmer Digital (the client).
- We, the project group, will be given non-commercial right of use.

1.3.2 Technological Constraints

- The product must be developed in the game engine Unity.
- The product must be a 3D game.
- Support for Oculus Quest 2.

1.3.3 Time Constraints

- The Project Plan must be finalized and delivered by January 31st.
- Standard agreement for academic collaboration must be signed and delivered by February 1st.
- The Product and Report must be finalized and delivered by May 20th.

2 Scope

2.1 Subject Area

The client aims to create a virtual simulation environment where medical students can practice a wide variety of patient cases. This will serve as a substitute to in-person clinic training, and also be available as an additional learning resource for the universities to use in their courses.

The final product is meant to serve as a MVP which the client can use in their future development and as a proof of concept. The product will display the current point-and-click model taken into a 3D and VR environment. We also aim to make a scalable framework for handling and implementing new cases, which would make it easier and less resource intensive to create new cases as opposed to hard-coding.

2.2 Delimitations

The development of the game and case designer will be done in a new Unity project. Development of the game will be done at a minimum. The purpose of the game is only to import game-actions available in the case-designer, and to be able to test and play through produced cases.

2.3 Task Description

The final product should be a platform to allow for creation of cases and a VR environment in Unity to play through them. The platform will have two main components, a case creation tool and the VR game. Case representation will be object-oriented. Cases will be saved as JSON files and will be stored locally. The exact structure of which will be subject to change over the course of the development process.

The case creation tool should be easy to use for all users with various levels of technical competence.

When creating a new case the user will choose from 1 of 3 templates, each representing a different environment; general practitioner, emergency room or hospital. Each case will have a set of tests and treatments that are available in the selected template. Every test needs to be categorized as correct, neutral and wrong, which is used to make the result at the end of the case. The correct order of actions is also specified when creating each task.

The VR game component will be able to fetch cases previously created. The player will then be able to play through these cases and end up with a final result.

The 3D environment consists of 1 doctor (the player), 1 nurse and 1 patient. The player is given an initial case background on the patient, which is the starting point. The player will act on information given and gained from tests performed to progress through the case until finally choosing treatment for the patient. Tests will take different amounts of time to complete, certain tests can be performed simultaneously whilst others can not.

The result will be text based and give an indication of what was done correctly, things that were unnecessary or wrong. The result is based on time used, actions taken and what order they were taken in and the final diagnosis/treatment given.

3 Project Organization

3.1 Responsibilities and Roles

3.1.1 Project Leader - Amund Helgestad

The project leader always have a general overview of all ongoing activities within the project group. They expected to lead the project work in the correct direction according to what has been agreed to. To achieve this, the project leader may need to delegate tasks between the group members, though everyone should be active in this process and is therefore strongly encouraged to come up with ideas of their own.

The project leader should lead meetings and ensure all items on the meeting agenda have been addressed and discussed.

3.1.2 Chief of communications - Thomas Vincent Lien

This member is our team representative that will be responsible for communication *from* and *to* the group. Specifically they are responsible for the communication between the client and the group, as well as the supervisor and the group. They are also responsible for sending meeting agenda for meetings hosted by the group.

3.1.3 Chief of documentation - Kristian Aakervik Rønning

This member is responsible for making sure that documentation is organized and stored in appropriate locations. They are expected to oversee code documentation, pre-development and post-

development documentation. This also includes organizing and overseeing the project report, along with ensuring everything is documented and included in the report.

3.1.4 Recorder - Kristian Aakervik Rønning

This member is responsible for creating meeting minutes after each meeting, as well as conveying information afterwards and following up on it.

3.1.5 Delivery - Thomas Vincent Lien

This member is responsible for checking through documents, ensuring that everything is working before delivery. This member is also responsible for sending the deliveries.

3.2 Routines and Group Rules

3.2.1 Working Hours

Work hours are flexible to promote a healthy work-life balance. Each member is expected to commit at least 20 hours each week, and each member should try to work between 08:00 and 20:00 on workdays, unless something else is agreed upon. Workdays are Monday to Friday, excluding holidays.

It is also possible to work outside of workdays, but it is encouraged to use these days to rest. The purpose of this model is to ensure each working hour is as efficient as possible and that each member can work when they are at their best. Excess hours are considered voluntary, and will not deduct from the expected 6 daily hours on workdays.

The nature of the work requires regular communication and it is recommended to co-ordinate online hours for effective collaboration. Members are still expected to attend meetings, workshops and other team ceremonies.

3.2.2 GitLab

Rules and routines for GitLab is defined in Git Repository - Rules and Practices.

3.2.3 Meeting Agenda

Meeting Agenda will be sent out before each meeting by Chief of communications - Thomas Vincent Lien. Agenda should be sent at least 24 hours beforehand. These should include the agenda for the meeting, as well as documents that should be looked at or read beforehand.

3.2.4 Attendance & Absence

All meetings have mandatory attendance. Absence must be notified no later than 24 hours before a meeting. In cases where the meeting chair or recorder is absent these roles are to be reassigned.

3.2.5 Sanctions

If a team member does not follow these routines, i.e. does not notify absence in time or lacking noticeably in work load, an internal meeting will be called to resolve it.

Repeated occurrences will result in a written warning being sent. If this proves ineffective the project supervisor will be notified and assist in further actions.

3.2.6 Decisions

Decisions and disagreement are to be solved by a majority voting. Every member have the right to argue for their decision or solution before voting. In cases were a solution cannot be found the leader will have the final and deciding vote. If a disagreement continues to cause problems that prove detrimental to the project, the supervisor can be called in to advice in the matter.

4 Planning, Follow-up and Reporting

4.1 Development Process

4.1.1 Choice of Development Process

Before choosing a framework for the development process, it is crucial to take the requirements and expectations of the client into consideration. Zimmer Digital has a goal of creating a larger software product and want us to develop a solution for a part of the larger goal, which they can use or even develop further in the future on their own. However, they do not have a precise plan for the implementation of the said solution. This means the project group will have to plan the implementation mostly on our own, with only feedback and support gained from our weekly meetings with the client to guide our progress. Such a working environment consequently favours an agile or Scrum-based approach, as the development plans would need to be flexible enough to accommodate feedback we get during development.

On the other hand, Scrum usually requires at least a daily meeting to discuss the progress and development process of that day, which the client or stakeholders should ideally attend; however, this is not practically feasible for this project. The client is available for one weekly meeting only, as well as on-demand communication on slack.

Kanban is another very flexible framework, as there are many benefits with it being relatively easy to use, and it has a good overview over available tasks which can be claimed and worked on. Additionally, GitLab has a good Kanban board feature - the so-called Issue Board - which several of our members have positive previous experience with; consequently, we would like to use this feature again. On the other hand, Kanban is mostly not time-bound and there are little to no rules for when a task needs to be done to. Such a framework as Kanban would therefore be hard to manage in terms of time on such a large project as this.

Taking the pros and cons of both Kanban and Scrum into account, we think a combination of the two would serve our interests the best. Such an approach is often referred to as *Scrumban*.

4.1.2 Use of Development Process

Scrumban is an agile system development process and is a hybrid between Scrum and Kanban. Scrumban allows us to work in small iterations, not too different from sprints in Scrum. Short iterations allows us to quickly and easily adapt our course throughout the project if need be. Every iteration starts with a planning meeting where tasks to be done are selected and prioritized on the board. Scrumban focuses on task prioritization over task assignment, meaning each member can prioritize and choose tasks themselves from the board.

Unlike Scrum, Scrumban does not require specific roles. Roles are rather defined by the tasks the team member chooses themselves, which tends to best suit their speciality and skill set.

Scrumban also has limits to ensure efficiency and that team members are not overburdened. Scrum-

ban methodology therefore has a WIP limit, which says that each team member should only work on one task at a time. This is also limited on the board, where the number of tasks in the *In-Progress* column is equal to the amount of team members.

Scrumban Board

We will use the Issue Board in GitLab and configure it as a Kanban board. It will include the following categories:

- Backlog
- To-Do
- In-Progress
- For-Approval
- Done
- Archive

The *To-Do* category will mostly consist of highly prioritized tasks to do in the relevant iteration, and must be completed by the end of the iteration. These tasks may be claimed by a member so that they can do them later when they get time, though this should be approved in group-meetings prior to claiming. The *In-Progress* category aims to communicate which tasks are currently being worked on, and by whom. Each user are limited to only work on one task at any time. The *For-Approval* category consists of tasks that has been completed, but await approval from group before it can be merged into the main branch. After having been approved as described in 5.2, the task is further moved to the *Done* category. The *Done* category should be emptied after each iteration and the issues moved to the *Archive*. Issues that are not completed during an iteration are moved to the *Backlog* of the next.

Iterations

Iterations will start on Tuesdays and will last 5 working days. The meeting with the client on Mondays will represent the end of the current iteration as well as the start on a rough planning phase for the next iteration. Each member are free to manage their time outside work-days as long as they show the expected progress to the expected time.

4.2 Plan for Status Meetings and Decision-Points in the Period

For each iteration we will conduct weekly meeting with both the client and the supervisor.

Client meetings are scheduled every Monday at 17:00. In the meeting the client will give feedback on the previous iteration, as well as planning the next iteration. Most product specific decision-points will be done here.

Meetings with the project supervisor is scheduled every Friday at 12:15. These meeting are used to review project progress and status, specifically related to the final report and decisions related to it.

Planning meeting will be held at the start of each iteration after the meeting with the client every Monday.

Daily meetings are held every weekday at 09:00. This will last about 15 minutes. The meeting is used to walk through yesterdays progress, and to plan the current day.

5 Organization of Quality Assurance

5.1 Documentation, Standards, Configurations, Tools and Source Code

The client has indicated that the final product could be part of their future development. Documentation will provide the client and other developers with knowledge required to do so. Adhering to set standards and conventions will ensure a codebase that is understandable and consistent.

5.1.1 Documentation

Pre-development documentation will help create a picture of what is to be developed. This will include requirements for the product, technologies used and design documentation. This is produced with active feedback from the client, and is critical in making sure we best understand the clients vision.

Code should be well documented with a focus on readability and clarity so that it is easily understandable for not only other current project members, but for other programmers in the future as well. This means that simple code that are self-explanatory may not need as much commenting, however, larger code chunks probably need it. **Most functions** (except mutators and accessor methods) and **all classes** should be documented; when doing so, project members are expected to follow the recommended documentation standards and Microsoft's recommendations for C# using XML elements. [2].

Post-development documentation will consist of user manuals and installation guides. These are created to aid users using the product after the project is concluded.

5.1.2 Standards

ISO-8601 ISO-8601 is an international standard for exchange of date and time [3]. Following this standard, dates are written as YYYY-MM-DD and time as hh:mm.

C# Coding Standards When we are developing the product, all code should follow the C# Coding Conventions. [4].

5.1.3 Tools

Unity Unity is a cross-platform game engine widely used for game development. It also offers built in functionality for VR. Unity will be used for developing the 3D game [5]. Unity runs on C#.

Visual Studio Visual Studio will be the IDE used for code editing. There exists other alternatives, but Visual Studio is perhaps the most common for programming in Unity.

Git & GitLab Git is a version control system to track version and changes to the source code and files. For the project we have chosen the internal NTNU Gjøvik GitLab.

Toggl Toggl Track is an application used for tracking working hours in a team. Tracking is done per user and each entry contains time, description and tags. This allows us to track how much time is spent and where it is spent.

Severity / Probability	5 - Very High	4 - High	3 - Medium	2 - Low	1 - Very Low
5 - Catastrophic	25	20	15	10	5
4 - Critical	20	16	12	8	4
3 - Moderate	15	12	9	6	3
2 - Marginal	10	8	6	4	2
1 - Negligible	5	4	3	2	1

Table 1: Risk indices

LaTeX LaTeX is an advanced and powerful tool for writing and preparing technical and scientific documentation. LaTeX will be used for writing of documentation and the final report.

Discord, Teams, Zoom & Slack Discord is used for internal communication. Teams will be used for weekly supervisor meetings. Zoom is used for weekly meetings with the client. Slack is used to communicate with the client outside of scheduled meetings, sending documents and planning meetings.

5.2 Plan for Inspection and Testing

Following Scrumban methodology review meetings will be called on-demand. This will be conducted throughout the development process to inspect and test the software.

All tasks that are completed will have to go through approval before being marked as done on the Scrumban board. This is to ensure that every addition or change to the code is working, well documented and adhering to coding conventions. Every task will require approval from at least one additional team member, not being the one responsible for the task. Tasks requiring approval are always listed on the Scrumban board. Approvals can also be requested in daily meetings or on Discord.

The client will have access to working iterations throughout development. This allows the client to inspect quality and direction of the product, and provide essential feedback guiding the development.

5.3 Risk Analysis

A risk analysis has been completed to assess potential threats to the project. These are chosen for their relevance to the project or the current world situation.

Risk Index is derived from Table 1, and is calculated as $Severity * Probability$. Table 2 shows an overview of risks and their respective probability, severity and overall risk index rating.

5.3.1 Risk mitigation and contingency plan

PROG-1: To ensure we are able to represent real scenarios we will regularly take feedback from the client and show demos of the current development progress.

PROG-2: We may need to create a rough prototype fast and test that, and iterate on it from what we learn and client feedback.

PROG-3: We will need to get feedback during development and adjust our course accordingly if necessary.

ID	Description	Probability	Severity	Risk Index
PROG-1	Product not being able to produce cases that represents real scenarios	3 - Medium	4 - Critical	12
PROG-2	Evaluation from cases being too vague	3 - Medium	4 - Critical	12
PROG-3	Product unable to be integrated into clients future product	3 - Medium	4 - Critical	12
TOOL-1	Gitlab crashing	1 - Very Low	4 - Critical	4
TOOL-2	Unity crashing	1 - Very Low	4 - Critical	4
TOOL-3	VR tools unavailable during development	3 - Medium	2 - Marginal	6
PERS-1	Project member unable to work due to sickness	1 - Very Low	4 - Critical	4
PERS-2	Project member infected by Corona	4 - High	2 - Marginal	8
PERS-3	Lack of technical knowledge	3 - Medium	3 - Medium	9

Table 2: Risk analysis overview

TOOL-1: Frequently updated back-ups will be stored in GitHub repositories. In the event that both GitLab and Github are down, the project members will still have local copies on their computer.

TOOL-2: Only the playthroughs of cases are required to be made in Unity. If Unity becomes unavailable for a long period, the project team should evaluate if the case designer should be developed using other, available tools.

TOOL-3: VR is only required for testing the cases in the virtual environment. If the VR tools offered by NTNU become unavailable, the project team still has one set of VR tools at their disposal. Since the project revolves around making a case creation application, most testing will not require VR tools.

PERS-1: If any project member becomes too ill to work, we will have to temporary or permanent changes of the roles within the team.

PERS-2: FHI is expecting a surge in cases of COVID-19, Omicron variant from January to March [6]. Furthermore, they state that the probability of getting severely ill as a vaccinated, young person is very low.

Most work is done remotely over digital collaboration tools and from home, allowing for work during a potential quarantine without transmission risk.

PERS-3: Selecting tasks in accordance to each member's strengths. Performing technical research where necessary.

6 Plan for Implementation

6.1 Timeline

The Gantt chart in Figure 1 shows the currently planned timeline of the project.

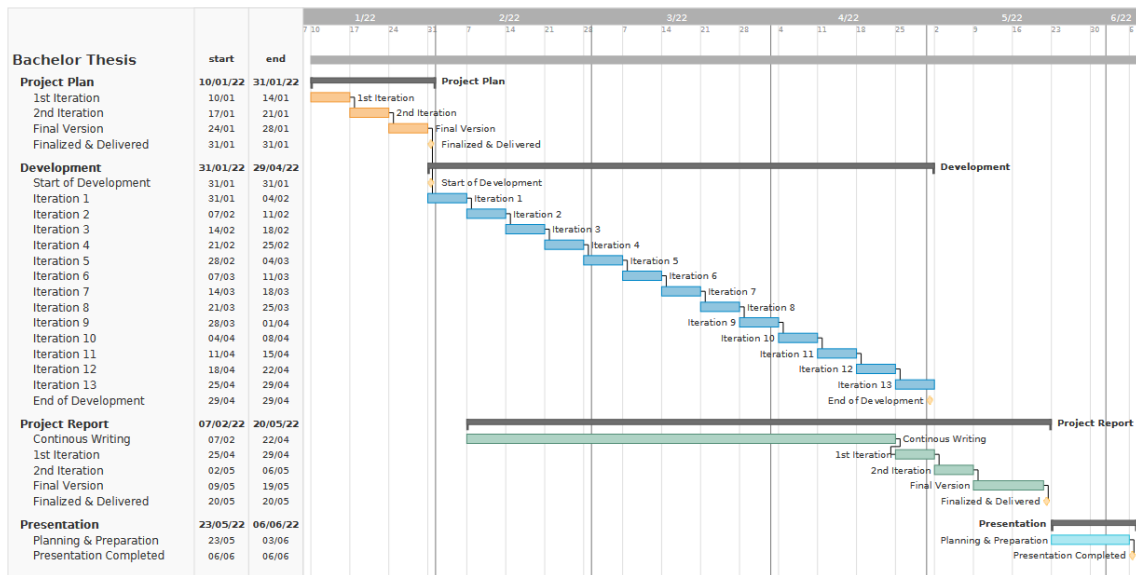


Figure 1: Gantt chart

6.2 Milestones

The following milestones have been selected and will indicate key points in the project progress.

2022-01-31 Project plan finalized and delivered.

2022-01-31 Start of development.

2022-04-29 End of development.

2022-05-20 Project report finalized and delivered.

2022-06-06 Project presentation completed.

Bibliography

1. Zimmermann M. VR som pedagogisk prinsipp på medisinstudier – en visjon. MA thesis. UiO, 2021
2. Wagner B, Arndt M, Yeleighton C and Edelen A. Recommended XML tags for C# documentation comments. Available from: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xml/doc/recommended-tags> [Accessed on: 2022 Jan 20]
3. Wikipedia.org. ISO 8601. Available from: https://no.wikipedia.org/wiki/ISO_8601 [Accessed on: 2022 Jan 20]
4. Wagner B, Williamson M, N NP and Kulikov P. C# Coding Conventions. Available from: <https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions> [Accessed on: 2022 Jan 20]
5. Unity Technologies. Unity. Available from: <https://unity.com/> [Accessed on: 2022 Jan 20]
6. Public Health NI of. Oppdatert risikovurdering og modelleringsrapport om omikron-varianten. Available from: <https://www.fhi.no/nyheter/2022/oppdatert-risikovurdering-og-modelleringsrapport-om-omikron-varianten/> [Accessed on: 2022 Jan 27]

Appendix

A Git Repository - Rules and Practices

Protected Branches

1. It is important to state that the repository is the property of the *whole group*, and not that of an *individual group member*. This means that changes to shared resources, such as the main branch, must be approved by the group.
2. The **main** branch is always marked as protected and cannot be changed without the approval of **at least one other member** and cannot be approved by the person who created or initiated the change. This is in compliance with point 1 above and the security principle commonly known as **Separation of Duties**.
3. The main branch *cannot be pushed to directly*, as that would undermine point 1 and 2. To apply a change or commit, one would instead need to create a **merge request**.

Workflow, Issues and Issue Board

1. The common workflow to synchronise work done by multiple members is to have a separate branch for each system. If multiple members are working on the same system, then a separate branch for each member is needed. These branches should then be merged together in a timely manner depending on the size and complexity of the branch and change.
2. Members who are working on the same system are required to communicate and collaborate more closely together to ensure their branches are up-to-date with each other.
3. Members should **try to push their work to the external Git repository** instead of leaving it to only be stored locally. This is to further encourage collaboration within the group, and to allow other members - especially the project leader - to see the progress of the project.
4. The type of work or tasks that are being worked on are usually described in a corresponding *issue*. Such issues that describes the tasks that needs to be done in the project and their current status, is usually shown on the **Issue Board**. The issue board should be updated and new tasks/issues should be added during group meetings according to what is stated in the development process.

Commits and Commit messages

1. Write **short and precise**, yet **meaningful** commit messages that fully describes the updates this commit brings since last commit. Simply writing "Updated [insert thing here]" is in most cases *not ok*, while writing *too long* is not good either. If the commit is large and need a longer message to be described fully, simply **create a new issue** instead and refer to it in the commit message (see point 2).
2. **Try to tie a commit to an issue** by writing '#'+issue number in the commit message. (F.ex "#1 Updated README.md"). Even if there are currently no active issues on the commit, you can create a new issue right before you make the commit. This is also a way to describe the commit in more detail without it making the commit message itself too long.

Merging and Merge Requests

1. To merge two branches, one would need to create a *merge request*. A merge request may require approval before it can be executed.

-
2. **Squashing Commits** is an option when merging; This will remove each individual commit from the commit history and only show the merge commit. This is meant to simplify the history, and may help to make it look cleaner, however, I would **recommend to not use** it in most cases.

Merge Conflicts:

A merge may in some cases be unable to be performed due to a *merge conflict*. In this case, the merge must be done locally and each conflict must be solved manually. A useful tool to do this with is **GitHub Desktop**. Then, after all merge conflicts seems to have been resolved and the merge have successfully been performed, one would need to do a manual check of the system in question to make sure everything is working correctly. The merged branch you now have locally must then be uploaded to a new branch and a new merge request must be created, and in some cases, approved.

Common Git Commands

1. **git branch** : Will show all the current branches
2. **git checkout branch_name** : Will switch to the specified branch, if it exists. A variation of this command is **git checkout -b new_branch_name**, which create a new branch and switch to it.
3. **git checkout -b new_branch existing_branch** : Will create a new branch with name "new_branch" that is identical to the "existing_branch", and switch to it.
4. **git status** : Lists which files are staged, unstaged and untracked.
5. **git add .** : Will add all files from the current directory to be staged for commit.
6. **git commit -m "Commit Message"** : Will commit all files that are staged for commit and save them under the commit message "Commit Message".
7. **git pull branch_name** : Will pull changes from the specified branch and immediately merge it into the local copy.
8. **git push origin branch_name** : Will push changes from current branch to the specified remote branch.
9. **git clone link_to_repo local_directory_name** : Will clone the repository from the specified link into the specified local folder. If the specified local folder do not exist, it is automatically created.

For more information, you can check out the **Git Cheat Sheet** available at <https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>.

Appendix D

User Test

User test

Experience

3rd year medicine student that has previously tested the client's implementation. Does not have much experience using simulation apart from using it in skydiving. Competent with digital systems to a moderate level.

Game

The user was shown the entire demo video of the product once before questions were asked.

Q: Does this look familiar?

A: Yes, this looks similar to the client's product

The user was then asked questions regarding the objects found within the game

Q: Do you recognise this? (The patient journal)

A: Yes, this is a patient journal

Q: Do you think it is appropriate that most results are noted down in the journal?

A: This is fine, normally this is read from a screen, or something related to the tool. But some of it ultimately ends up in the patient journal anyways.

The video was then progressed to the part where it conducts examinations

.

Q: What do you think this blue area is? (The visible hitbox of the patient when equipped with a tool)

A: This shows the area in which the tool can be used.

The user then gave some feedback on what they would like to see

User: It would be nice if information was not only given by the journal. An example is for heart examinations, it is common to listen to the sounds and make out information from reading charts.

Case Designer

The user was initially given an empty case designer and asked to design their own case.

Information tab – Easily understood what to fill in on each section and quickly filled them all in. (Needed explanation on the game duration section)

Listnodes and category

Navigated easily to the anamnese category and added a category within it.

Anamnese

After setting the name of the category the user was shortly stuck on how to add a question to the category. (Needed explanation using the video to show the difference between the category and a question)

After this, the user quickly grasped the concept of creating questions, setting the relevancy, and filling out the answers. (User did not give any wrong inputs)

Klinisk US

Quickly navigated to this section and created categories within "Klinisk US". Quickly adds an action, setting all the input fields and adds a journal point result. (Explanation on time to complete was given using the demo video)

Note:

- Was confused that they could choose the computer as a tool, as "Klinisk US" is the category for examinations using tools.
- Thought that only one result could be bound to one action.

Exporting and Importing

The user was asked to save their case, before resetting the application, the group subtly asked her to remove a category from the "Klinisk US".

The user understood the functionalities of importing and exporting. After the case editor was restarted, the user imported the case data using the import button.

The user was then quizzed on what happened to the removed category, as it was now inside the case. The user then explained that this happened as it was deleted after saving the case, which is correct.

Second attempt on creating a case

To see how quickly the user learns how to use the case editor, the user was given a second attempt at creating a case. The user was given a template case with prefilled information.

The user quickly made changes in the information tab.

The user navigated smoothly throughout the "undersøkelser" section, creating questions quickly in the anamnese section.

The user then moved to “Klinisk US” section, creating new actions, and adding results with ease. The user also figured out that they could edit actions/questions by clicking existing ones.

Notes

The user would sometimes forget, or almost forget to set relevancy levels before setting questions/answers.

The user did not encounter any visual input feedback, as they did not make any mistakes using the input fields. They were later shown this, and was able to explain what was wrong when wrong inputs were given.

Appendix E

The Problem which the Concept Solves

Problemet ideen løser

Hvilket problem løser ideen?

Behandlingsfeil innen helsesektoren har store konsekvenser for pasient og samfunn. Miljøer har pekt på faktorer som ekspertise- og ressursmangel, samt dårlig kommunikasjon. I 2020, hadde NPE rekordutbetaling på 1,1 milliarder NOK til pasienter som ble feilbehandlet. Konsekvensene av dårlige valg, besluttet av uforberedt personell kan ende i personlig ruin hos den ansvarlige, sosioøkonomiske tap for samfunnet og redselkultur. Det er et paradoks at medisinsk innovasjon har eksponentiell utvikling, mens utdanningen av personell står fast i utdaterte pedagogiske og ressursløsende metoder. Attpåtil har pasientkontakt under opplæring minsket da hver pasient har færre sykehusdøgn, dårligere pasienter og flere studenter per pasient som trenger å øve. Det er dyrt og lite tilgjengelig simuleringsutstyr, samt mangel på gode veiledere. Helse har de siste to år lammet en hel verden, derfor er det helt nødvendig å ha utdanningsmetoder som er ressurseffektive og holder gullstandard innen opplæring.

Hvem løser ideen problemet for?

Medisinske eksperter er ikke ferdig utdannet ved studieslutt, men må utvikle og utdanne seg gjennom hele deres profesjonelle karriere. De trenger alle å være kontinuerlig oppdatert innen medisin og teknologi som utvikler seg med en eksponentiell fart. Det samme gjør kompleksiteten i vurderingene. Helsesektoren er ofte preget av ufullstendig situasjonsbilder i miljøer med høyt langvarig stress. Feil vurdering eller dårlig kommunikasjon kan føre til død eller livsvarige mén for enkeltindivider med dertil byrde også for samfunnet. Vårt fokus er helsesektoren, og vi starter på grasrotnivå. Gode beslutninger må læres tidlig, samt internaliseres gjennom langsiktig repetisjon. Medisinstudenter, vil som fremtidige leger, ta valg hvor konsekvensene kan gi død og erstatningskrav Mental forberedelse er bokstavelig talt livsviktig. Det er forøvrig i dag, hverken menneskelige eller økonomiske ressurser til å takle oppgaven om langsiktig praktisk-teoretisk medisinsk trening for studenten.

Hva er deres løsning på problemet?

Zimmer Digital introduserer et nytt skifte som har som mål å utvikle menneskelig nær medisin med teknologiske løsninger. Det medisinske virtuelle miljøet gir studenten et veiledende verktøy som følger sluttbruker igjennom hele studiet. De får personlig tilpasset arbeidsprogram, veiledning og tilbakemelding, samt oppdatert pasientkasuser for mengdetrening. De får tilgang til et hovedrom hvor oversikt over arbeidsplan, mål og delmål, samt tilbakemeldinger, er lett tilgjengelige. Programmet er laget for flere plattformer som senker terskelen for bruk. Selve kasusene blir gjennomført i en realistisk setting med virtuelle pasienter. Studenten må selv undersøke, vurdere, diagnostisere og sette i gang behandling. Etter endt kasus får man relevant tilbakemelding i skriftlig format hvor styrker og potensiale blir tatt opp. Simulering er den optimale treningsmetoden for helsefagarbeidere og er overlegen tradisjonell klinisk trening. Programmet fremmer pasientsikkerhet og

reduserer kostnader og ressursbruk. Vi har 5 verdier rettet mot både studenten og samfunnet. 1. Direkte – Økt motivasjon og effektivisering av læring for studenten 2. Sosialt – Bedre pasientbehandling, færre uønskede hendelser 3. Operasjonelt – Effektivisering av behandling. Lavere behandlingsnivå 4. Strategisk – Positiv arbeidskultur, fokus på ønskete verdier, bærekraft 5. Finansielt – Økt inntjening og færre kostnader Det ble av professor og overlege innen gastrokirurgi ved Rikshospitalet, Oslo uttalt følgende om Zimmer Digital sine løsninger etter å ha deltatt på en demonstrasjon: «Zimmer Digital lager et program som gjenspeiler måten vi i framtiden vil tenke om utdanning» (vinter 2022).

Det unike ved løsningen

Selskapets løsning skal gi studenten tilgang til produktet gjennom noen enkle tastetrykk, på hvilken som helst mobil enhet/plattform og på det viset gjøre studenthverdagen enklere og mer oversiktlig. De har tilgang til simulering, flervalgsspørsmål på mobil relatert til gjennomgått kasus og annet læremateriell. Innovative løsninger innen VR og 3D-simulering, pedagogikk og IT vil bli utviklet i tett samarbeid med sluttbruker. Flere av eierne har vært en del av grasrota og sluttbruker, og ser dermed hvordan smarte og tekniske innovasjoner, sent eller aldri når endebrukeren. Vi har vært kontakt med 414 brukere, helseinstitusjoner, fagfellerapporter og media som har vist klart og tydelig hvor problemene ligger. Vi vil vha maskinlæring utvikle et vurderings og tilbakemeldingssystem som på lang sikt kan utvikles inn mot andre sektorer. Som feks klinisk beslutningsstøtte. Vi skal med brukerrettet- og teknologisk innovasjon lage et generisk produkt for alle, skreddersydd for den enkelte.

Appendix F

Iteration Documentation

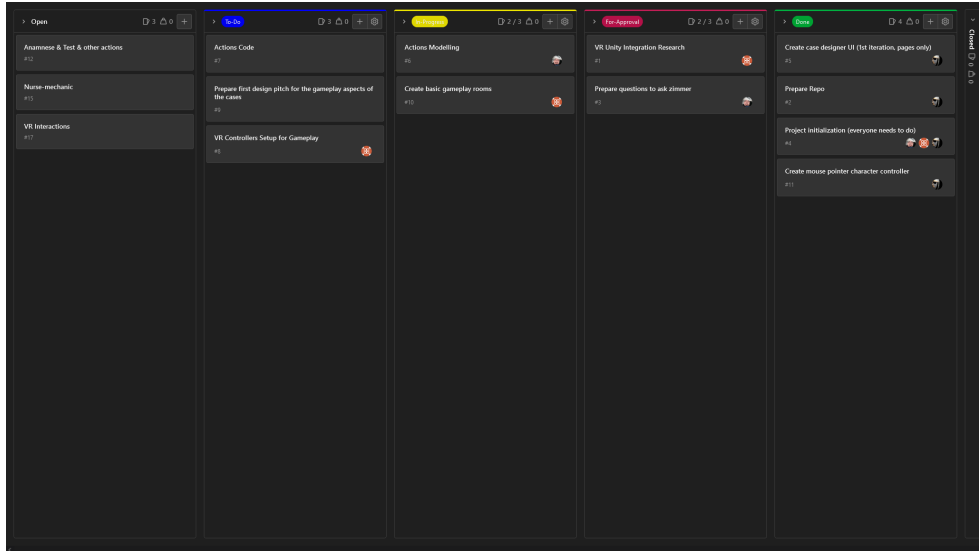


Figure F.1: Iteration 1

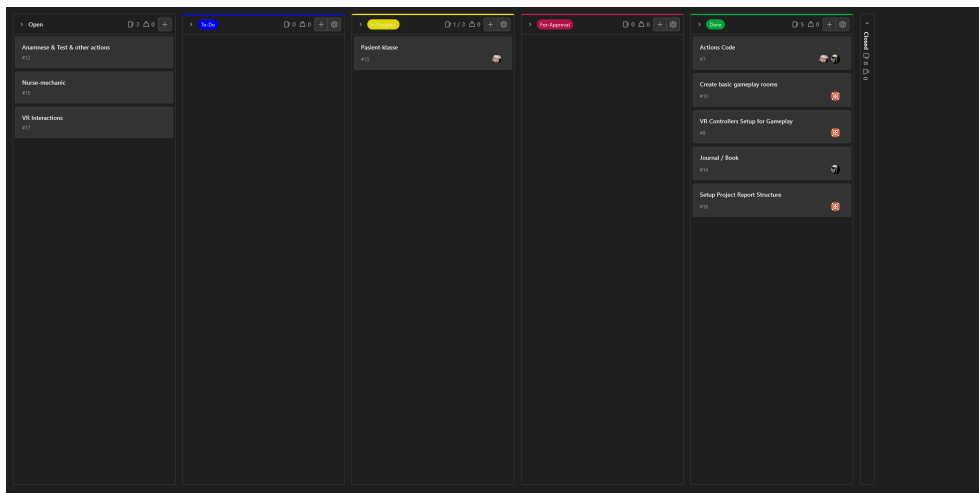


Figure F.2: Iteration 2

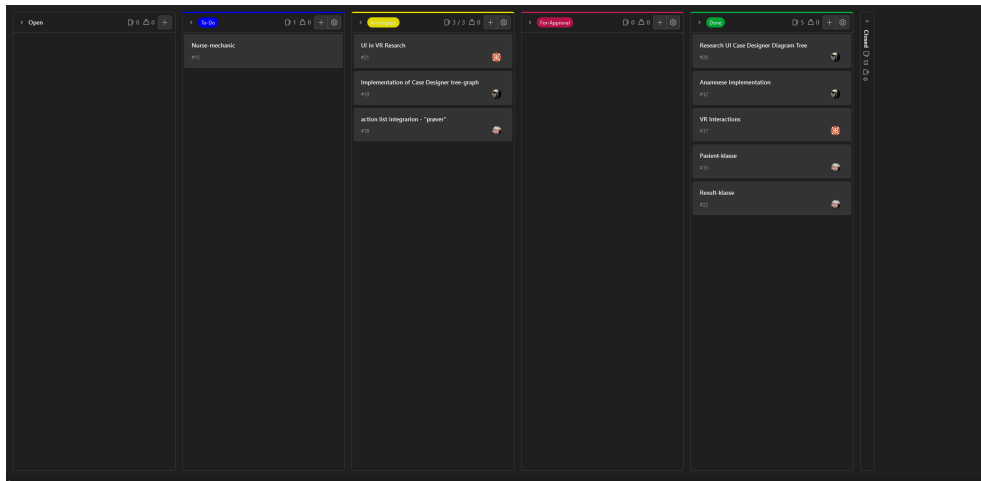


Figure E.3: Iteration 3

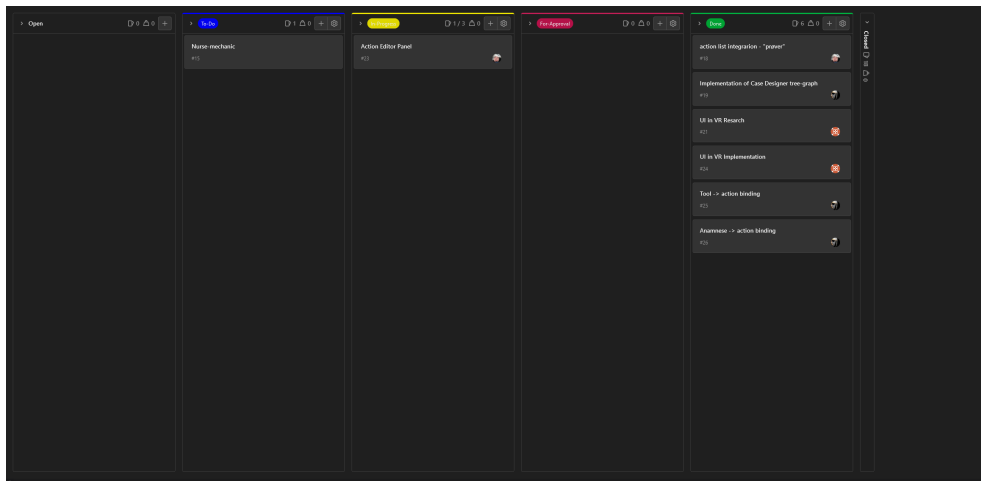


Figure E.4: Iteration 4

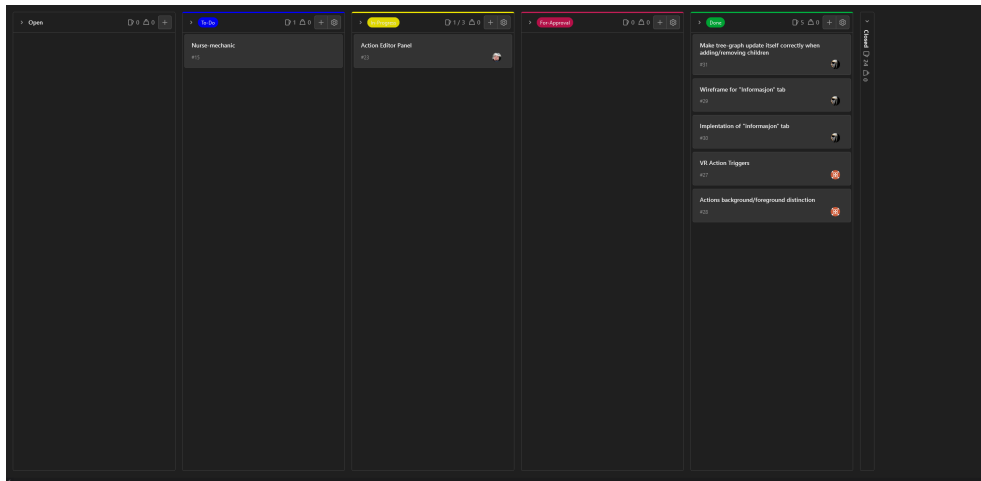


Figure F.5: Iteration 5

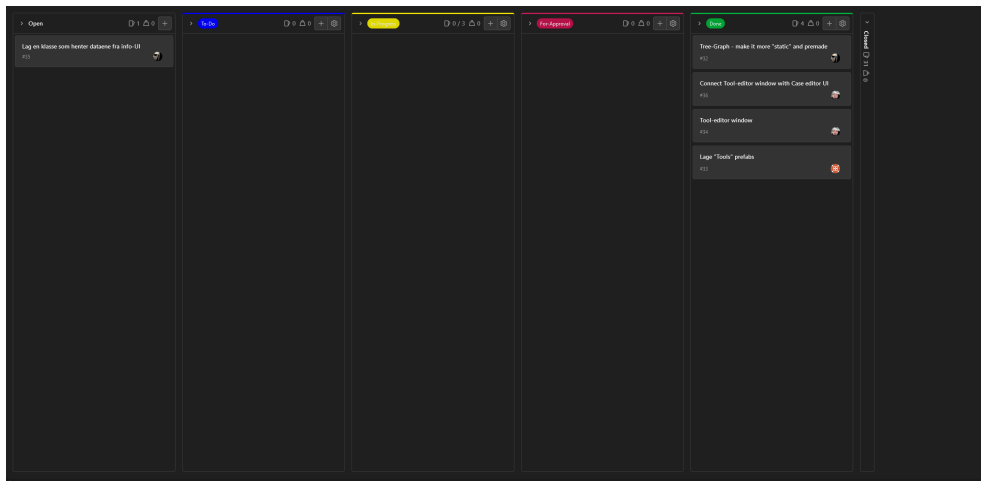


Figure F.6: Iteration 6

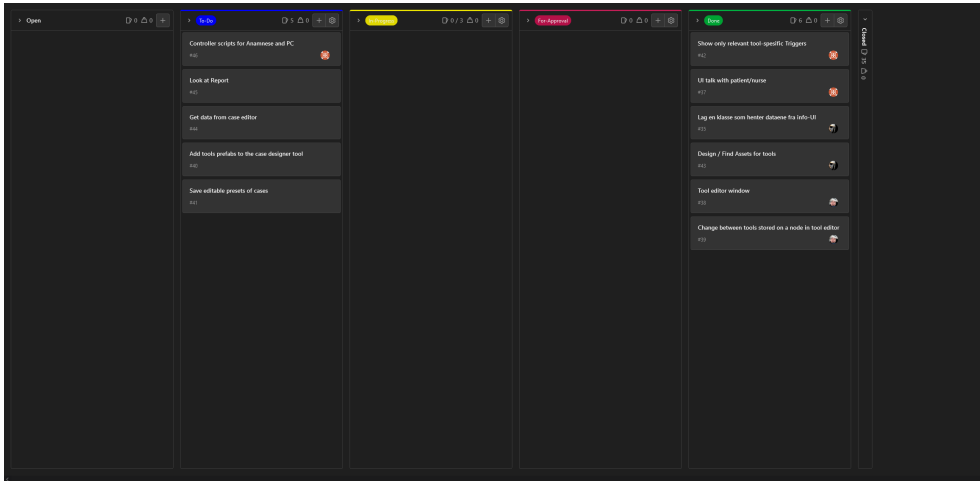


Figure F.7: Iteration 7

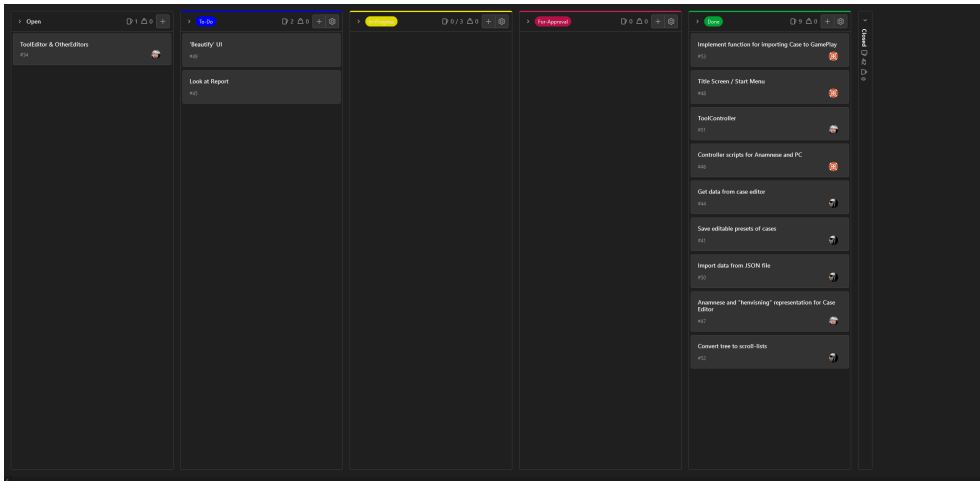


Figure F.8: Iteration 8

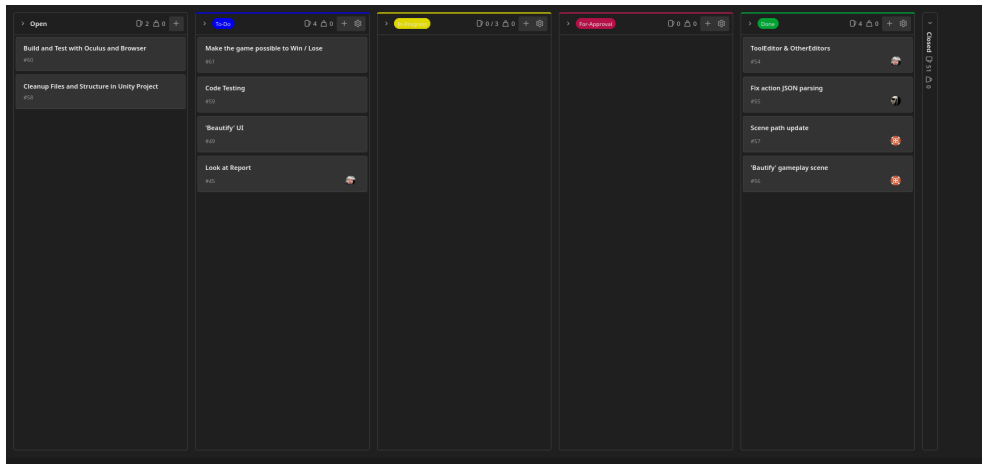


Figure F.9: Iteration 9

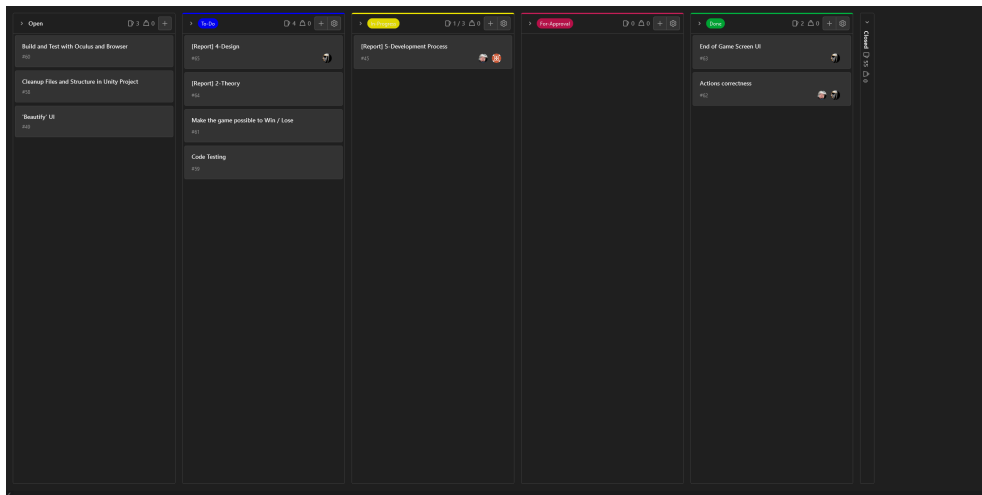


Figure F.10: Iteration 10

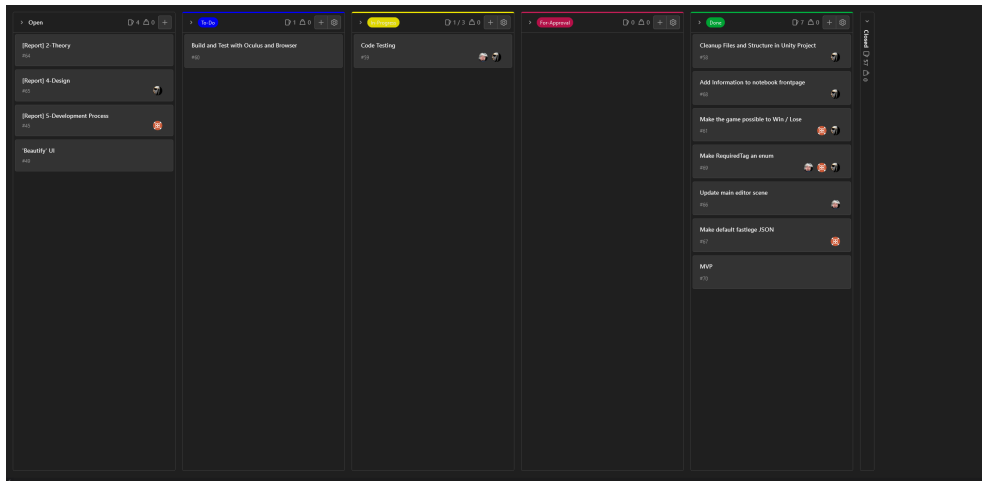


Figure F.11: Iteration 11

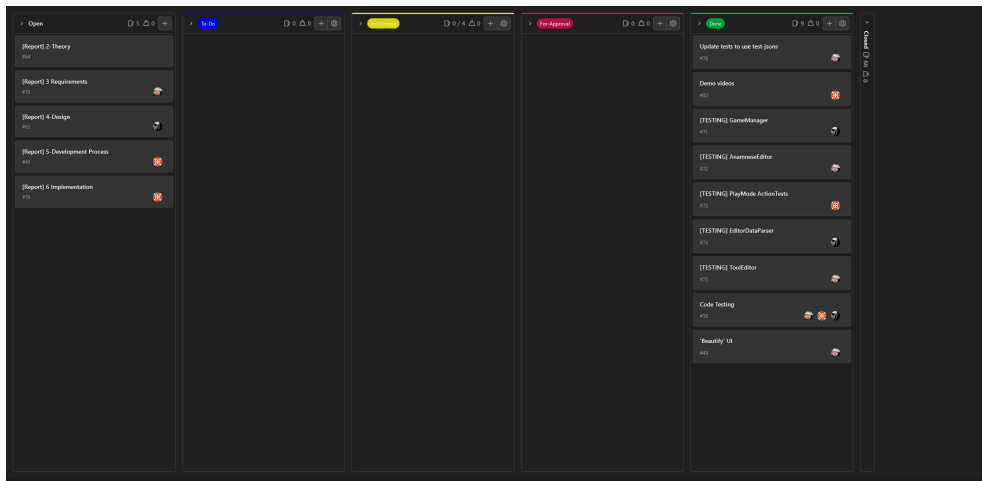


Figure F.12: Iteration 12

Appendix G

GitLab Repository Rules

1 Git Repository - Rules and Practices

Protected Branches

1. It is important to state that the repository is the property of the *whole group*, and not that of an *individual group member*. This means that changes to shared resources, such as the main branch, must be approved by the group.
2. The **main** branch is always marked as protected and cannot be changed without the approval of **at least one other member** and cannot be approved by the person who created or initiated the change. This is in compliance with point 1 above and the security principle commonly known as **Separation of Duties**.
3. The main branch *cannot be pushed to directly*, as that would undermine point 1 and 2. To apply a change or commit, one would instead need to create a **merge request**.

Workflow, Issues and Issue Board

1. The common workflow to synchronise work done by multiple members is to have a separate branch for each system. If multiple members are working on the same system, then a separate branch for each member is needed. These branches should then be merged together in a timely manner depending on the size and complexity of the branch and change.
2. Members who are working on the same system are required to communicate and collaborate more closely together to ensure their branches are up-to-date with each other.
3. Members should **try to push their work to the external Git repository** instead of leaving it to only be stored locally. This is to further encourage collaboration within the group, and to allow other members - especially the project leader - to see the progress of the project.
4. The type of work or tasks that are being worked on are usually described in a corresponding **issue**. Such issues that describes the tasks that needs to be done in the project and their current status, is usually shown on the **Issue Board**. The issue board should be updated and new tasks/issues should be added during group meetings according to what is stated in the development process.

Commits and Commit messages

1. Write **short and precise**, yet **meaningful** commit messages that fully describes the updates this commit brings since last commit. Simply writing "Updated [insert thing here]" is in most cases *not ok*, while writing *too long* is not good either. If the commit is large and need a longer message to be described fully, simply **create a new issue** instead and refer to it in the commit message (see point 2).
2. **Try to tie a commit to an issue** by writing '#'+issue number in the commit message. (F.ex "#1 Updated README.md"). Even if there are currently no active issues on the commit, you can create a new issue right before you make the commit. This is also a way to describe the commit in more detail without it making the commit message itself too long.

Merging and Merge Requests

1. To merge two branches, one would need to create a **merge request**. A merge request may require approval before it can be executed.
2. **Squashing Commits** is an option when merging; This will remove each individual commit from the commit history and only show the merge commit. This is meant to simplify the history, and may help to make it look cleaner, however, I would **recommend to not use it** in most cases.

Merge Conflicts:

A merge may in some cases be unable to be performed due to a *merge conflict*. In this case, the merge must be done locally and each conflict must be solved manually. A useful tool to do this with is **GitHub Desktop**. Then, after all merge conflicts seems to have been resolved and the merge have successfully been performed, one would need to do a manual check of the system in question to make sure everything is working correctly. The merged branch you now have locally must then be uploaded to a new branch and a new merge request must be created, and in some cases, approved.

Common Git Commands

1. **git branch** : Will show all the current branches
2. **git checkout branch_name** : Will switch to the specified branch, if it exists. A variation of this command is **git checkout -b new_branch_name**, which create a new branch and switch to it.
3. **git checkout -b new_branch existing_branch** : Will create a new branch with name "new_branch" that is identical to the "existing_branch", and switch to it.
4. **git status** : Lists which files are staged, unstaged and untracked.
5. **git add .** : Will add all files from the current directory to be staged for commit.
6. **git commit -m "Commit Message"** : Will commit all files that are staged for commit and save them under the commit message "Commit Message".
7. **git pull branch_name** : Will pull changes from the specified branch and immediately merge it into the local copy.
8. **git push origin branch_name** : Will push changes from current branch to the specified remote branch.
9. **git clone link_to_repo local_directory_name** : Will clone the repository from the specified link into the specified local folder. If the specified local folder do not exist, it is automatically created.

For more information, you can check out the **Git Cheat Sheet** available at <https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>.

Appendix H

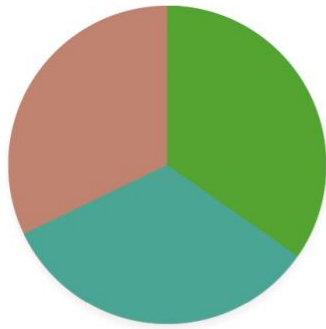
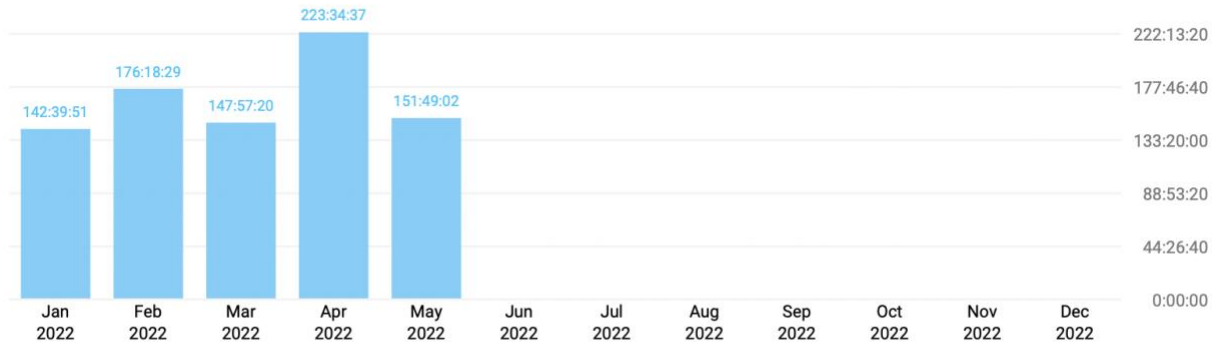
Time Tracking

Summary Report



01/01/2022 – 12/31/2022

TOTAL HOURS: 842:19:19



USER	DURATION
AH Amund Helgestad	292:55:45
KR Kristian	278:54:30
TH Thomasvl	270:29:04

