Applying Gamification and Virtual Reality to an MRSA Infection Control Guideline

Bråten, Oskar Elias
Igesund, Trygve Eide

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 2020

Western Norway
University of
Applied Sciences

UNIVERSITAS BERGENSIS

Abstract

Methicillin-resistant Staphylococcus aureus (MRSA) is a group of harmful bacteria with resistance to many important antibiotics. Infections are hard to treat and often result in prolonged hospital stays, leading to increased costs and mortality. Like with many other bacteria and diseases, healthcare practitioners follow carefully developed clinical practice guidelines on how to test for and handle occurrences.

In this thesis, we present an experimental study on applying gamification principles to transform an MRSA infection control guideline into a virtual reality game. Our approach consists of designing and developing a fully functional game, describing our process and outlining the challenges associated with this. We also set out to find out how VR and gamification affect practitioner motivation through user testing. Throughout development, we discovered challenges with transforming the guideline into a game that can be used nationally while fully retaining its content. We also found that applying gamification while still staying true to the clinical practice guideline was a difficult balancing act.

Acknowledgements

First and foremost, we would like to express our gratitude to our supervisors at Western Norway University of Applied Sciences (HVL), Harald Soleim and Atle Geitung for their guidance, support and valuable insights throughout our work on this thesis. Their frequent feedback, as well as their eagerness to help us make useful connections, has helped us tremendously and has made this work possible.

We would also like to thank our external supervisors at Helse Vest IKT, Eva C. Backer, Håkon Garfors and Thomas F. Larsen, for providing us with the opportunity of exploring the use of gamification and virtual reality. They have contributed significantly to this thesis with their support and feedback throughout the development process, and it has been a joy working with them.

Furthermore, we would like to thank Grete O. Hole and Daniel Patel for their invaluable input on the thesis, and for writing a research paper about our project, co-authored by Harald Soleim, Atle Geitung, Eva C. Backer and us. The paper has been accepted to the EDEN Annual Conference of 2020. We greatly appreciate your involvement with this thesis.

We also thank Bodil Moss for giving us advice on academic writing and inviting us to present our master's thesis at ”Forsking i fyr og flamme”. Also, thanks to Marianne Sævik at Haukeland Hospital who showed us around in the infection control ward, as well as arranging user testing with the nurses (even if unforeseen circumstances made them impossible!).

Last but not least, we are truly grateful to our friends and fellow classmates, Bendik, Jonas, Kolbein, Lauritz, Magnus, Per and more, for all the fun we have had throughout this past year.

Division of Work

Our contributions to this thesis mostly overlap. In the overview below, we use our names to indicate the parts where one of us did most of the writing.

Chapter 1 - Introduction - **Both**

Chapter 2 - Background

- 2.1 - Methicillin-resistant Staphylococcus aureus - **Bråten**
- 2.2 - CPGs and simulation - **Igesund**
- 2.3 - Gamification - **Igesund**
- 2.4 - Computer graphics - **Bråten**
- 2.5 - Virtual reality - **Both**
- 2.6 - Game engine - **Bråten**

Chapter 3 - Design and Implementation

- 3.1 - From guideline to game - **Bråten**
- 3.2 - Gameplay - **Both**
- 3.3 - Architecture - **Igesund**
- 3.4 - Game logic - **Both**
- 3.5 - VR interface
    - 3.5.1 - Level design - **Bråten**
    - 3.5.2 - Cross-platform support - **Igesund**
    - 3.5.3 - Input and interactors - **Bråten**
    - 3.5.4 - Interactables - **Igesund**
    - 3.5.5 - Feedback - **Igesund**
    - 3.5.6 - Patients - **Bråten**
- 3.6 - Balancing gamification and accuracy - **Both**

- 3.7 - Assets - **Igesund**
- 3.8 - Internationalisation - **Igesund**
- 3.10 - Performance and optimization - **Bråten**

Chapter 4 - Results - **Both**

Chapter 5 - Conclusion - **Both**

Chapter 6 - Further work - **Igesund**

Contents

List of Figures

Glossary

β-lactam antibiotics The term used for antibiotics that contain a β-lactam ring in their molecular structure. Examples include penicillin derivatives such as methicillin. 1, 7

Anteroom The small room through which one must pass in order to reach the isolation room. Its purpose is to prevent infectious pathogens from being passed into or out of the isolation room. 31, 51, 58, 59, 70

Asset Within game development, assets can refer to a variety of content such as 3D models, animations, audio files, textures and even shaders and scripts. 26, 54, 57

Clinical practice guideline (CPG) Systematically developed recomendations and statements for healthcare practitioners on how to diagnose and/or treat a medical condition [6]. 1–3, 5, 8, 12, 29, 53, 54, 69, 70

Extended Reality (XR) An umbrella term used for technology which extends human experiences, usually referring to augmented reality, virtual reality and mixed reality but also future technology. The technology it refers to is situated somewhere on the reality-virtuality continuum, a concept introduced by Paul Milgram et al. [7], which is a one-dimensional representation consisting of the two extremes: reality and virtuality, and everything between. 19, 23, 40

Frame rate (FPS) The rate at which consecutive images appear on a display. 14

Graphics processing unit (GPU) A specialized hardware component for performing highly parallel workloads. 14, 18, 67

Head-mounted display (HMD) A display device worn on the head, typically consisting of one or two display units (one for both eyes, or one per eye). 19, 20, 22, 27, 40, 56, 62, 71, 74

Heads-up display (HUD) In games, it is a graphical view showing the player some information in the user interface while in-game. 22

JSON JavaScript Object Notation, a programming language independent format that allows storage of data in a serialisable way. 36, 66, 74

Methicillin A narrow-spectrum β-lactamase resistant antibiotic of the penicillin class. 7

Methicillin-resistant Staphylococcus aureus (MRSA) A type of pathogenic Staphylococcus aureus which is resistant to methicillin and other β-lactam antibiotics, and often also multi-resistant, i.e. resistant to other antibiotics. i, 1–3, 7, 8, 12, 29, 37, 61, 66, 69, 72–74

OpenVR An application programming interface and runtime which allows access to virtual reality hardware from multiple vendors [8]. 23, 26, 28, 40

Sepsis A life-threatening condition where the body's response to an infection causes harm to its own tissues and organs [9]. Often called blood infection or blood poisoning. vi, 5, 6

Staphylococcus aureus (S. aureus) A potentially pathogenic bacterium, often causing minor skin and post-operative wound infections. 1, 7

Virtual reality (VR) Generally refers to the use of technology to alter and/or replace real world stimulus in order to create a virtual reality. May also refer directly to the technology being used. 2–4, 9, 12, 19–24, 26–30, 34, 37, 39, 40, 47, 51, 52, 54, 60, 62–65, 68, 69, 71–76

Chapter 1

Introduction

In this chapter, we start by introducing the motivation and origin of this thesis. Next, we formulate a goal with accompanying research questions, followed by our methodology explaining how we intend to answer those questions. Then we introduce the thesis scope and limitations, and finally related work.

1.1 Motivation

Staphylococcus aureus (S. aureus) is a type of bacteria commonly found among humans [10]. For a healthy person, the bacteria rarely does any harm, and around 20 - 40% of the population can be a carrier of the bacteria over long timespans. However, in healthcare institutions where patients undergo medical procedures and have weakened immune systems, the bacteria poses a threat. MRSA refers to S. aureus strains that are resistant to β-lactam antibiotics and that are also potentially multi-resistant (i.e. resistant to other types of antibiotics) [10]. If the bacteria were to become common in hospitals, the treatment of S. aureus could become less effective and more costly. Also, an increase could force a change in the use of antibiotics, which could further increase the occurrence of resistant bacteria. For these reasons, Norwegian hospitals implement special measures against MRSA.

A recent study on the temporal evolution of MRSA in Norway shows that its prevalence, although low compared to the rest of Europe, has increased over the last decade. Additionally, the study shows that a large number of occurrences can be attributed to tourism and in recent years to an increase in immigration [11]. Figure 1.1 shows the proportion of MRSA in human blood isolates in countries in Europe. As we can see, the Nordic countries have a fairly low number of occurrences compared to the rest of Europe.

Healthcare practitioners must be able to detect and handle infections in order to prevent the spread of MRSA. Clinical practice guidelines (CPGs) are meant to assist healthcare practitioners by providing systematically developed

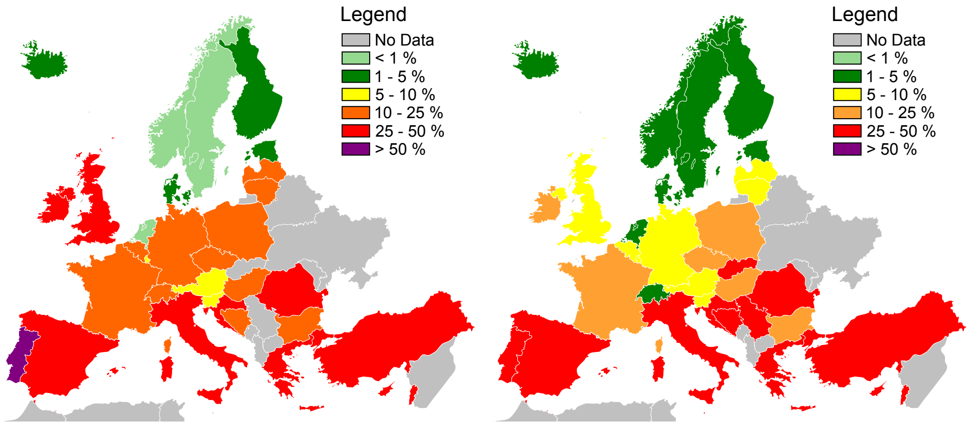


Figure 1.1: Proportion of MRSA human blood isolates from participating countries in 2008 and 2017 [1].

recommendations. One such CPG is Helse Bergen's MRSA infection control guideline, the guideline describes, among other things, which patients to screen for MRSA and how to perform the screening. If a practitioner is not familiar with the recommended practice, it could result in further spreading of the bacteria.

In recent years Virtual reality (VR) technology has seen tremendous improvement, in part thanks to improved tracking, display technology and computational power. At the time of writing, we observe three main categories of application:

- Games, where the goal is for the player to have fun.
- Simulators, where the goal is to mimic real-world scenarios closely.
- Non-interactive experiences, where the player mainly acts as an observer, as if on a visit to a museum.

Gamification can be described as a combination of the two first categories, where one applies game design to a variety of processes with the goal of increasing player motivation, which in turn can help increase engagement and time spent playing [12]. A plethora of such applications can be found on mobile platforms [13] [14] [15]. These are often used for learning things like languages, mathematics and programming. In these applications, you typically complete tasks for points. The common factor for these learning activities is that they lack physical interaction. Such abstract learning activities are, therefore, unlikely to benefit as much from applying VR compared to more physical activities. Within healthcare, a large number of physically involved activities are performed as part of clinical practice. These interactions can be more accurately simulated with VR technology than, for example, on a smartphone.

VR technology, as an educational tool, has been the topic of several studies. A meta-analysis from 2013 examined a number of studies within the categories of games and virtual worlds. It suggests that VR-based instruction is an effective means of improving learning outcomes [16].

Helse Vest is one of the four regional healthcare agencies in Norway. As part of the agency's innovation plan, it asserts the goal of increasing knowledge and use of innovative game technology, as well as a need for closer cooperation with universities and university colleges [17]. In addition, the agency's strategic plan states that new digital solutions play an essential role in future patient treatment and that practitioners, therefore, must have fundamental digital competence. Furthermore, they want to increase the number of employees with high digital competence in order to contribute to innovation, service development, and training of colleagues, patients and relatives [18]. These aims show that there is an apparent demand for innovative technology in Norwegian healthcare institutions. The use of games for learning purposes is one such potential innovation. In this thesis, we seek to apply gamification and virtual reality to the problem of MRSA to meet this demand.

1.2 Origin of the thesis

The idea of creating a learning-game about antibiotics was first discussed in Helse Vest IKT's work with the game "Stopp Sepsis" (see Section 1.6). In discussion with one of the partners of the working group, head of the regional patient security programme Anne G. Kvalvik mentioned that "screening for MRSA" would be a very relevant subject for a learning-game.

Helse Vest IKT's first game, "Stopp Sepsis", was a great success among healthcare practitioners and other stakeholders within the health services in Norway. When the opportunity of developing a new learning-game together with the authors of this thesis arose, the innovation working group put forth the idea of developing a game involving "screening for MRSA". Innovation advisor and nurse Eva C. Backer further developed the idea, and together with technical advisors Håkon Garfors and Thomas F. Larsen, they later presented it to the authors of this thesis.

1.3 Goal and research questions

Together with our consultants at Helse Vest IKT, we formed the goal of developing a VR game that expresses the national guidelines that healthcare workers must follow to detect occurrences of MRSA in patients. We chose to base the game on the CPG from Helse Bergen (Appendix A), which is a well-established guideline used by Haukeland hospital. The goal of this thesis is for the game to function as a light-hearted entry-level learning tool for practitioners, which can be published and used at numerous healthcare institutions in Norway and in turn bring widespread attention to the problem.

For this reason, we base the game on guidelines that apply to as many practitioners as possible. In doing so, we intend to answer the following research questions:

- What are the challenges with developing a virtual reality game based on the clinical practice guideline?
- How does the use of virtual reality, and gamification affect practitioner motivation?

By answering these questions, we hope to understand better how to develop usable and useful VR games based on healthcare guidelines.

1.4 Scope and limitations

As mentioned, the goal is for the game to be relevant for a wide array of healthcare practitioners. Because of this, the game can not contain recommendations that are local to an area or institution, which could otherwise prevent the adoption of the game. Here we can draw a parallel to an earlier game published by Helse Vest IKT called Stopp Sepsis (see Section 1.6). Although the game was successful and had a broad reach, concept developer Eva C. Backer, mentioned that the game was not adopted in a particular area because of local adaptations. Furthermore, it is of interest that the end product is as complete as possible seeing as Helse Vest IKT intends to publish the game. Reducing the scope of the thesis is necessary to make this possible within our time and resource constraints. As a result, we are restricting the game to follow a single flow, designed to capture the essence of the material, without going into details that are specific to a scenario or a particular healthcare institution. By a single flow, we mean that the game consists of a single general chain of tasks, as opposed to multiple specific chains of tasks for all possible scenarios. At the same time, we also employ gamification to make the game more enjoyable, which has the potential of increasing the learning effect of the game and its popularity. The details of how the scope affects the design and implementation become apparent throughout the development process, and we describe it in more detail at various places in Chapter 3, 4 and 5.

Unfortunately, due to the outbreak of COVID-19 scheduled user testing with 10 nurses from the infection control ward at Haukeland Hospital had to be cancelled. Additionally, planned tests with nursing students at Western Norway University of Applied Sciences (HVL) were cancelled due to the university campus being shut down. As an alternative, we instead opted for more in-depth testing with 2 domain experts and also used data gathered from domain expert tests conducted throughout the development process. We describe these limitations and the alternatives in detail in Chapter 4.

1.5 Methodology

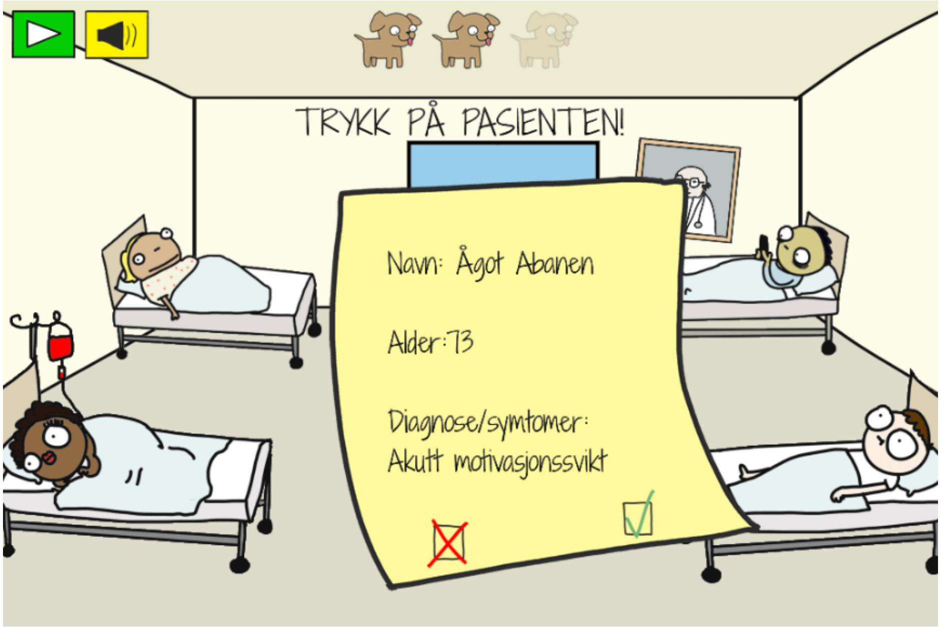
We intend to answer the research questions by developing a fully functioning VR game based on Helse Bergen's CPG for MRSA infection control. Throughout the development process, various challenges arise in areas such as cross-platform compatibility; identifying and implementing gamified aspects of the guideline; user interaction design; performance and optimisation; and more. When faced with such challenges, we consult existing literature and research within the respective fields as well as consulting and performing tests with our external domain experts. Finally, we perform user testing, as described in Chapter 4, to evaluate the resulting game.

In Chapter 3, we describe the design and solution of the game. In Chapter 4 we look at the results from user testing and how they correspond to the different challenges we have tackled and the possible trade-offs between different solutions. Where possible, we present measurements of relevant metrics. Finally, in Chapter 5 we discuss and conclude on our results.

1.6 Related work

The IT department of the Western Norway Regional Health Authority, Helse Vest IKT, has previously published a game revolving around medical procedures called Stopp Sepsis [19]. The player is presented with short information snippets on Sepsis symptoms and treatments, after which they are tasked with following the procedures. The game utilizes gamification principles heavily. The player is punished or rewarded with every decision, depending on whether they correctly follow the procedures or not. The rewards take form as health points illustrated as cute dogs, stars popping up on the screen and triumphant sounds. When the player makes a mistake, a gunshot is heard and one of the health points, in the form of a dog, dissipates into red flowers. The player must keep at least one dog to win the game.

In a blog post from February of 2019, concept developer and scriptwriter Eva C. Backer wrote a blog post on how Stopp Sepsis came to be [20]. In the post, she writes that they have received 70 000 site hits since April of 2018 and that the game has been used as a course in medical learning resources throughout Norway.



Navn: Ågot Abanen

Alder:73

Diagnose/symtomer:
Akutt motivasjonssvikt

☒ ☑

Figure 1.2: A screenshot from Stopp sepsis. The player needs to choose whether the patient should be screened for Sepsis or not.

Chapter 2

Background

2.1 Methicillin-resistant Staphylococcus aureus

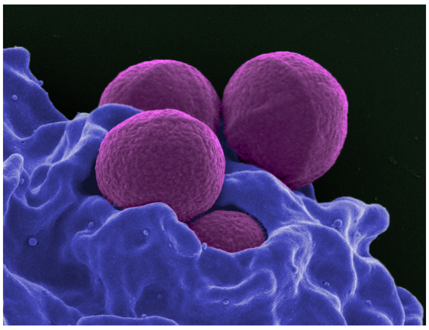


Figure 2.1: MRSA,
National Institute of Allergy and Infectious Diseases, NIH. CC BY NC 2.0

S. aureus is a potentially harmful bacterium, often causing minor skin and post-operative wound infections [21]. In the early 1940s, before penicillin had gained widespread use, the mortality rate of infected individuals was around 80%. Two years later, in 1942, the first penicillin-resistant S. aureus was observed in a hospital. In 1960, around the time when methicillin (a penicillin derivative) was introduced, approximately 80% of S. aureus strains were resistant to penicillin. Only two years later, S. aureus developed methicillin-resistance. Since then, MRSA has spread throughout the world.

The antibiotic methicillin is seldom used these days; instead, MRSA as a term now refers to S. aureus strains that are resistant to β-lactam antibiotics in general while also potentially being multi-resistant (i.e. resistant to other types of antibiotics) [10]. The majority of infections occur in hospitals. Infections are costly, result in extended hospital stays, and more importantly, increased mortality [21]. Additionally, delayed initiation of care and treatment further increases mortality, meaning general awareness and well-trained practitioners is of utmost importance.

In 2008 the Norwegian Institute for Public Health published several recommendations on how to combat the spread of MRSA [10]. The stated goal of the publication is to prevent MRSA from establishing and becoming a significant part of the bacterial flora in Norwegian hospitals and nursing homes. The publication contains guidelines for all Norwegian healthcare practitioners to follow. The guidelines range from standard precautions like hand hygiene to specific procedures for sampling bacteria and what to do with unexpected discoveries in patients. Helse Bergens MRSA infection control guideline is based on the publication.

A possible connection with COVID-19 mortality

Some sources have noted that countries with a high rate of MRSA also tend to have a high mortality rate from COVID-19 [22] [23]. While other factors such as population density and the testing capacity have a significant effect on outcomes, there is also speculation that the secondary infections that the coronavirus paves the way for, when caused by antibiotic-resistant bacteria, can not be adequately treated. Which suggests that MRSA may play a role in the COVID-19 mortality rate.

2.2 CPGs and simulation

A clinical practice guideline (CPG) is a collection of systematically developed recommendations to assist practitioner and patient make decisions about the appropriate health care for a particular circumstance [24]. They can be designed as a series of steps to follow, either for patients or healthcare professionals. They are often searchable in medical databases for quick access when needed.

When developing a CPG, a committee should be formed and cover all the essential aspects of the medical condition in question [6]. Ideally, the CPG should be based on up-to-date scientific knowledge, and it should be possible to follow the recommendations in daily healthcare practice. While some CPGs are made for the patient to read, most are meant for healthcare professionals. A healthcare professional's knowledge of a CPG is crucial for the patient's health.

In a report published by the Norwegian Institute of Public Health (NIPH), Fretheim et al. note that results from systematic overviews of CPGs are not consistently effective [25]. They write that the Norwegian healthcare personnel do not necessarily change practice as soon as the Norwegian Directorate of Health publishes a CPG. In fact, they find many examples of deviations from clinical practice and the official CPGs. Fretheim et al. do not go into detail about what causes these deviations. One factor could be that the health professionals are unwilling to follow all of the advice in the CPG based on local deviations or own experience. Another factor could be that the CPG is not adequately conveyed to the practitioners. In the latter example, it is an issue of learning. One of the ways they learn CPGs in healthcare is through *simulation.*

Simulation

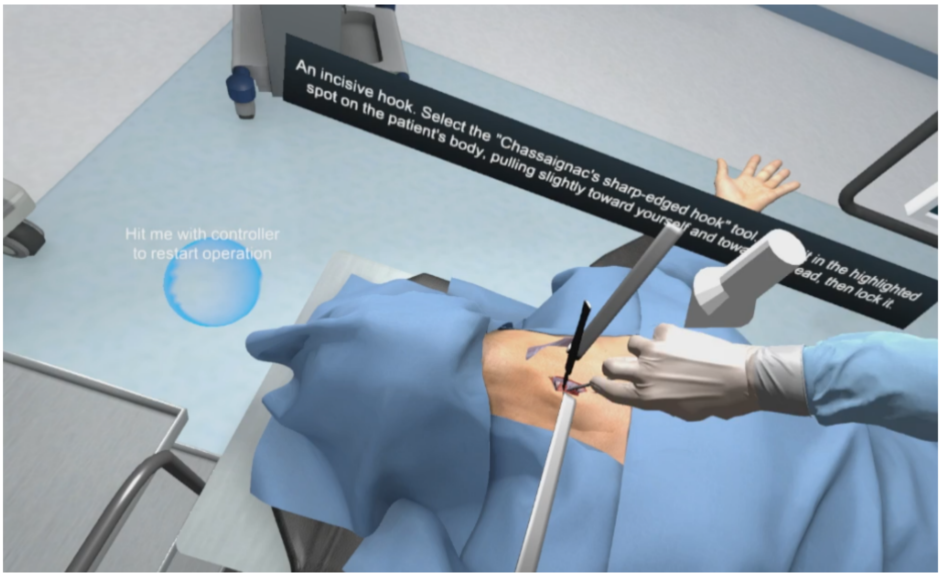
Simulation in healthcare is a way of practising a scenario in a controlled environment. Procedures ranging from trivial tasks such as hand hygiene to highly critical procedures such as responding to heart failure can be trained in simulations. The use of simulation in medical education has been traced as far back as early in the 18th-century [26]. Simulation can be thought of as a technique rather than a technology, although newer technology has opened up different ways to do it.

Norwegian medical students use simulation extensively in their education. The simulations are often performed with real tools and props such as mannequins, and sometimes in VR. Typically, current simulations try to mimic reality as closely as possible, leaving no room for gamification.

VR simulation

Healthcare simulations in VR is still a young topic, although some applications have been made for different procedures. Surgera VR is a free to play simulator of a surgical operation available for computer-powered VR headsets [2]. As the patient is lying on the hospital bed, you as the player must pick the correct tools from the table on the side and cut the throat open and perform the surgery correctly. The simulation is not *gamified* like the examples in Section 2.3 are, but more like a guided simulation. The closest thing to a gamification element in the application is a type of ”Game over” event if you perform too poorly. In that case, you are informed that the operation failed and you can choose to exit or restart the game. As seen in Figure 2.2, there are indicators and text to help you along the way in every step of the surgery. The indicators show you what tools to pick up and where to use them. Text popups always give an exact description of what you need to do.

A different type of healthcare simulation in VR is the RCSI Medical Training Sim [3]. Developed on behalf of the Royal College of Surgeons in Ireland (RCSI), it is a surgeon simulator available for the Oculus Go and phone powered Samsung VR headsets. You take the role as an emergency physician that has to treat a trauma patient. In stark contrast to Surgera VR, this is not a guided simulation; instead, you have the freedom to make independent decisions. As time is ticking, you have to figure out what is wrong with the patient while you are quizzed every step of the way to test your medical knowledge. As seen in the first picture of Figure 2.3, there are many options from which you can collect information about the patient’s health. You are presented with choices on how to do collect information and also how to assess that information. The second picture of Figure 2.3 is an example of such a choice. There is some instant feedback in that there is an indication of whether your decision was correct or not. Different sounds will play depending on the result, and the colour of the correct answer will turn green, while for the incorrect will turn yellow. If you make a fatal mistake, you will be shown that the patient



An incisive hook. Select the "Chassaignac's sharp-edged hook" tool, ... in the highlighted
spot on the patient's body, pulling slightly toward yourself and tow... ead, then lock it.

Figure 2.2: In-game screenshot from Surgera VR [2]. The white downwards facing arrow above the hand is an indicator that shows up at every step of the surgery, showing where to focus next.

has died. The game does, however, continue in the same place, now with the wrong option in red text, indicating that it can not be chosen again. If you pick the wrong answer that is not fatal, the game moves on, meaning you do not get to see the correct answer. At the end of the game, you will see a summary of the simulation, showing how many correct and incorrect decisions you made, how many times the patient ”died”, as well as time spent. This simulation is definitely more gamified than Surgera VR, in that you have to make decisions that are wrong or right, and you get some instant feedback on your choices.

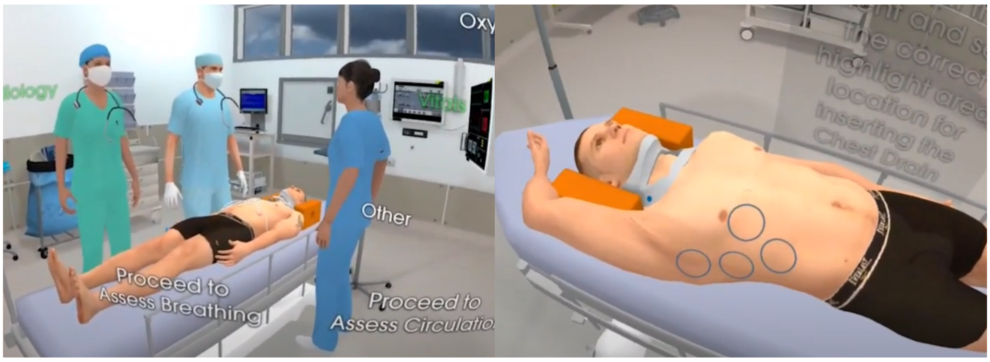


Figure 2.3: Screenshots from RCSI Medical Training Sim [3]. The first picture shows options of different factors you can assess. In the second picture, the player has chosen a chest drain insertion and must place it correctly by pressing the correct circle.

2.3 Gamification

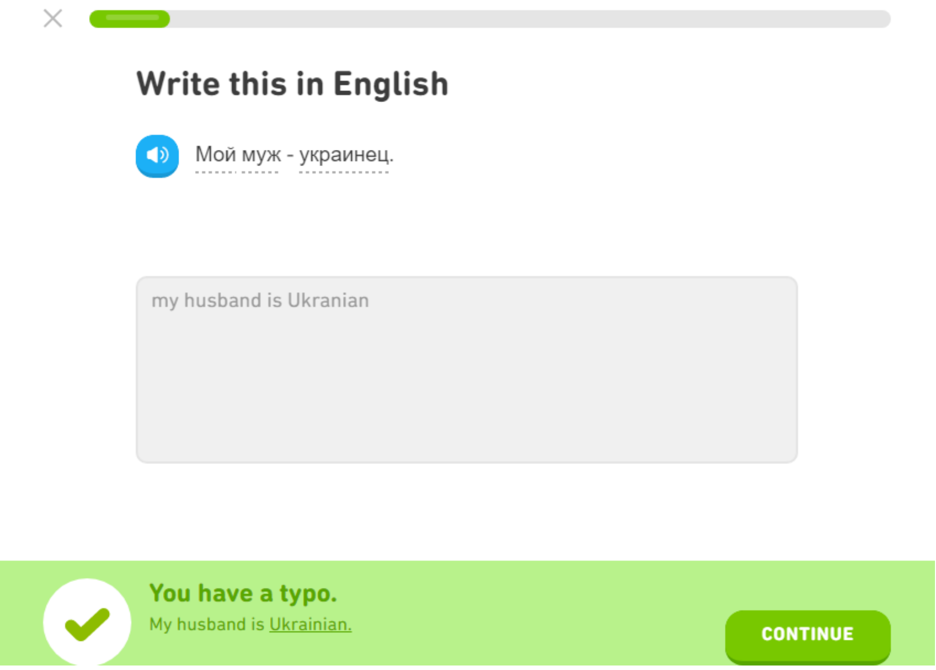
The term gamification has seen a surge in use since about 2010, but has been around for a long time. It is the application of game design principles in other areas than entertainment. It can be used in education or by people learning for leisure, often on mobile or web apps. The idea is that through the extra added motivation to complete tasks, the users will spend more time learning or simply be more engaged while doing it.

In 2014, three Finnish researchers conducted a meta-study on the effectiveness of gamification titled ”Does Gamification work?” [12]. The paper looked at 24 empirical studies on the topic, and found that most papers noted positive effects when it came to behavioural outcomes. Mentioned effects in the education context were increased motivation and engagement. The topic of gamification is young, academically and otherwise, and the knowledge collected on the topic is still evolving.

Common for gamified apps are tasks that you have to complete for some reward. There are gamified apps for learning things like languages [13], programming [14] and mathematics [15]. On the language learning platform Duolingo and while learning mathematics on Khan Academy, you typically complete a series of tasks in a row about a specific subject. In these examples, the tasks typically consist of translating a sentence or solving math problems. When you complete all the tasks, you are rewarded with points and can move on to a new set of tasks on a different subject or keep practising the same subject. On HackerRank, you can choose between many tasks. Unlike Duolingo and Khan Academy, solving a task can be time-consuming, even for more straightforward tasks, so you are only asked to complete a single task before returning to the menu. You can complete a task by submitting a written piece of code that must pass several tests.

On these platforms, there is often a view to show your progress. As you practice different subjects, the ”levels” will be marked as finished. Marking larger portions of the level menu gives the satisfaction of visual progress. On Duolingo, there is a graphical view that is particularly similar to a level menu known from regular games. When beginning to learn a new language, most levels are locked. Completing some subjects will unlock others, and in this way, you can progress through the level tree similarly to a regular game. Getting to the end of a language tree is comparable to beating a game.

On Duolingo you can follow other users, they can be someone you know from real life or someone who has interacted on their forums. You can see a weekly, monthly and all-time leaderboard listing the points scored by all users you are following. This social aspect is designed to incentivise competition and can lead to more learning. On HackerRank, there is a global leaderboard instead, showing all users.



Write this in English

Мой муж - украинец.

my husband is Ukranian

You have a typo.

My husband is Ukrainian.

Figure 2.4: An example of a completed task on Duolingo. Here the user has translated a sentence from Russian to English and can continue with the series of tasks

2.3.1 Gamification in Healthcare

Gamification and healthcare is not a new combination. Nyameino et al. [27] have outlined a model for making games for teaching CPGs. Their method involves a knowledge engineer who creates clinical models which are parsed and fed to a game engine which presents the players with questions. To prove the usefulness of this approach, the researchers developed a prototype e-learning system for mobile platforms. The game takes form as a quiz which gives the player instant feedback on every answer submitted as well as a summary and score in the end. This approach is useful in that it can cover a vast range of CPGs.

The limitation of the model-driven approach becomes apparent when one wants to introduce procedure-specific interfaces. An example from the MRSA screening procedure is practising bacteria screening with the help of a physical tool. In that case, a VR interface could be helpful, but such interfaces are difficult to combine with a model-driven approach. The reason for this is that the development of these types of game interfaces is expensive. Considering the cost of development, each interface should ideally be reused for different CPGs. If a procedure in a CPG is too specific, an advanced interface will have to be developed specifically for that procedure. As a result, many healthcare procedures can not easily be gamified in VR and integrated into a model-driven system. We discuss the possibility further in Section 6.1.

A videogame meant to teach medical students how to perform home visits was developed at the McGill Molson Medical Informatics Project. The game uses simulation to render a house of an older adult where the player must identify risk factors for harmful events. Clicking on objects either gives or removes points and at the end of the game the player receives a summary of correctly and incorrectly identified risk factors. The game was made with Flash and can be played in the browser using a computer mouse. Researchers from the same project conducted a peer-reviewed study on the motivation of medical students while playing the game [28]. When asked if the use of ”edutainment” had improved their learning, the majority of the 56 participants answered positively. 78% of them would recommend the use of edutainment in other ways.

2.3.2 Technology and the learning effect

A typical argument for technology in learning is the added interactivity it allows. This is not necessarily the case in healthcare simulations, as the users would be interacting anyway. On the other hand it does facilitate things like continuous feedback and quick change of environment in the simulation, while providing unstaffed measurement of how the user is doing. This could free up time for the supervisors of the simulation, if the application itself provides feedback to the user.

In a meta-study from 2018, Kang et al. investigated whether using web-based nursing educational programs increases a participant's basic knowledge and clinical performance [29]. Their results confirm the effectiveness of web-based nursing education programs. They mention that advantages include accelerated feedback to the learner and that such programs are convenient for those who cannot enrol in a traditional education environment.

2.4 Computer graphics

Computer graphics is the study of synthesizing, representing and manipulating visual content, and is considered a sub-field of computer science. The end product usually comes in the form of two- or three-dimensional images and is mostly viewed on a planar surface such as a computer monitor or television screen. However, other display methods exist as well, such as volumetric displays (e.g. holograms) [30]. Computer graphics as a subject covers a wide variety of topics including, visualizing geometry, user interfaces, computer vision and image processing.

2.4.1 Geometric primitives

Geometric primitives, within computer graphics, is a term loosely defined as the irreducible geometric entities that a system can represent and visualize.

Common primitives are points, lines, triangles, planes and spheres. In real-time rendering, the most common primitive is the triangle. Triangles are useful because they are guaranteed to be planar, unlike other polygons, and can be used to approximate any shape, while also being efficient to render. Figure 2.5 shows the head of a monkey represented with triangles.

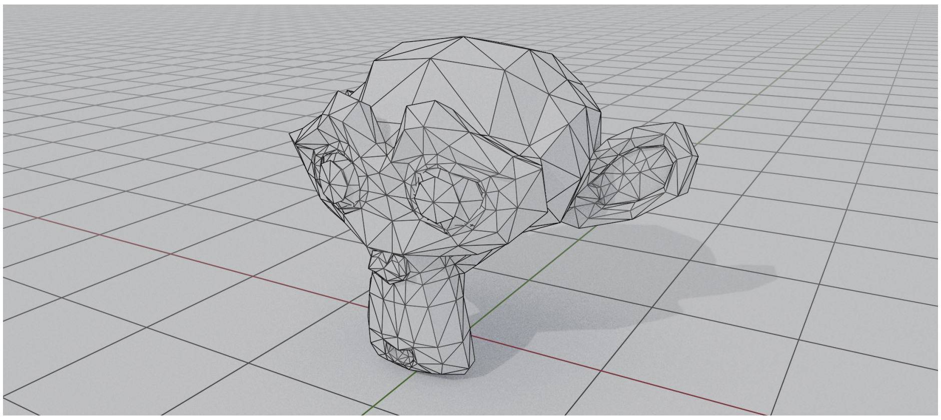


Figure 2.5: A model of a monkey head triangulated and rendered in Blender

2.4.2 Real-time rendering

Real-time rendering as the name suggests is concerned with producing graphics at an interactive rate [4]. We define the rate as the number of images rendered per second, i.e. frames per second (FPS). The number of frames that a program produces is not synonymous with the refresh rate (Hertz) of the display medium. The refresh rate, often expressed in Hertz, of a display is the rate at which it can refresh the currently viewed image. Under optimal conditions, the program produces frames at the same rate as the monitor refreshes the image, which results in the least amount of latency, i.e. the least amount of delay between the program rendering the image and the viewer seeing the results.

Rendering 3-dimensional geometry is a highly parallelizable task. A Graphics processing unit (GPU) is a specialized hardware component that is exceptionally well suited for performing massive amounts of computations in parallel. Using a GPU allows us to achieve interactive frame rates.

Numerous techniques are deployed within real-time rendering to achieve good looking results while remaining interactive. The predominant form of rendering is at the time of writing through a rasterization based pipeline. Figure 2.6 shows a simplified example of such a pipeline. The pipeline is based on the concept of rendering each object in the scene independently one after another. The application usually submits a draw-command to the GPU per object in the scene. In the geometry processing stage, the vertices that form the geometry can be transformed, usually based on the object's transform and the camera viewing it. After this stage, the resulting geometry is passed on to the rasterization stage. In this stage, the pipeline determines which pixels in the viewport the geometry covers. Lastly, the pixel processing stage calculates the final colour of the pixels in the image. The geometry and pixel processing stages are typically programmable via graphics APIs.

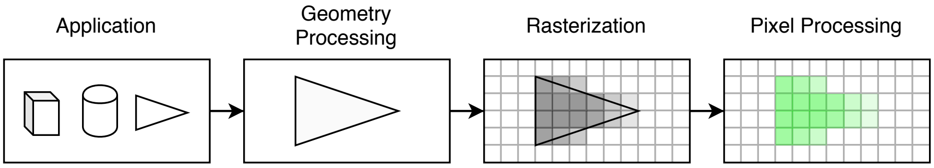


Figure 2.6: Simplified overview of a rasterization based pipeline (based on Figure 2.2 from [4])

While the rasterization pipeline excels at performing computations locally to each object in the scene, global effects such as transparency, shadows and reflections can be hard to implement and often result in physically inaccurate approximations. We refer to these effects as global illumination, and to compute them accurately requires information about other objects in the scene.

An alternative to rasterization is ray tracing. The concept of ray tracing is not new; the painter Albrecht Dürer (1471 - 1528) described the technique as early as 1525 [31], nearly 500 years ago. The concept is widely used within offline-rendering, such as for animated movies and visual effects, because it can achieve highly realistic results and because slow rendering times are not as much of an issue. Ray tracing follows an entirely different paradigm than rasterization. Instead of considering one individual object in the scene at a time and deciding which pixels of the screen to colour, ray tracing starts by deriving a ray based on the position of the pixel on the screen and the properties of the camera, and then asks the question: *which object, if any, does this ray intersect with and where?*.

One significant benefit of ray tracing is that it drastically simplifies the computation of global effects. For example, let us consider a simplified case of computing shadows. As shown in Figure 2.7, once the ray hits a surface we derive a new ray from the intersected position and the direction towards the light, we then check if the ray intersects with any objects before reaching the light and if it does then it must be in shadow. We can improve this further by taking the normal of the surface into account, which allows us to trivially reject any surface that is facing away from the light. Note, however, that this form of ray tracing does not achieve realistic-looking results. To achieve a realistic result, one must also consider the material properties of the surface and various types of light sources, as well as allowing the ray to bounce several times.

Ray tracing has not seen widespread use within real-time rendering because of its heavy computational demands. The computational demands are primarily

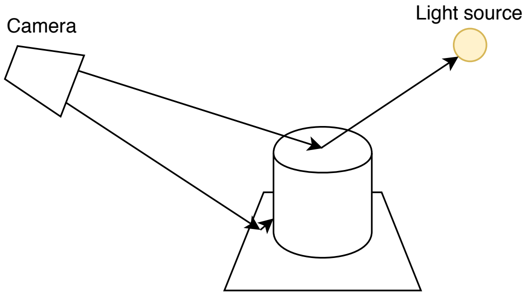
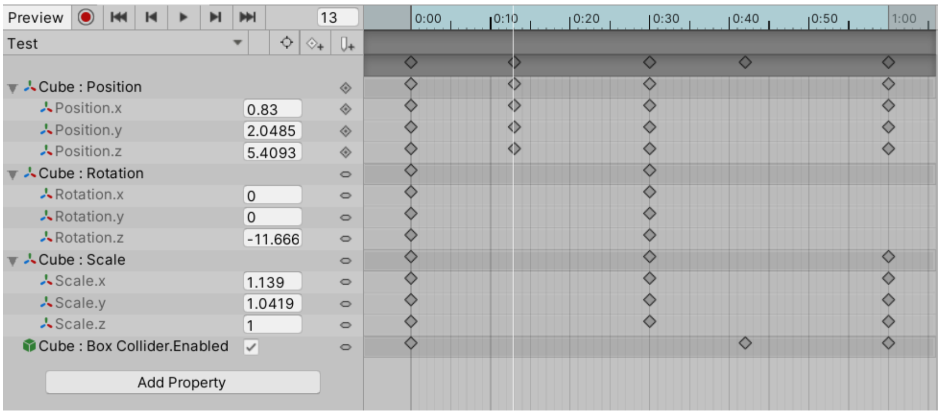


Figure 2.7: Ray tracing: Simple shadows example

due to ray-primitive intersection and can be improved significantly through the use of a bounding-volume hierarchy or a similar space partitioning scheme. In recent years however hardware acceleration has arrived on the consumer market, and the interest for real-time ray tracing seems to be growing. However, for now, the use of ray tracing in real-time applications is mainly limited to a technique called lightmapping. The idea is pre-computing in-direct lighting and storing the results in textures called lightmaps, which can be read and used to efficiently shade objects at run-time [4].

2.4.3 Animations



Cube : Position
Position.x 0.83
Position.y 2.0485
Position.z 5.4093
Cube : Rotation
Rotation.x 0
Rotation.y 0
Rotation.z -11.666
Cube : Scale
Scale.x 1.139
Scale.y 1.0419
Scale.z 1
Cube : Box Collider.Enabled ✓

Figure 2.8: Animation editor (screenshot from the Unity 2019.3 editor)

Animations are an essential part of creating an interactive and visually pleasing game. Multiple techniques are used to create animations. The simplest way of animating something, at least for a software engineer, is through code. One typical example is to interpolate between two values, over a certain amount of time or a certain number of frames, in a function that is called every frame. This technique is useful when the animations are straightforward, for example, when opening and closing a sliding door. However, for more advanced animations, a more comprehensive and designer-friendly system is useful. Figure 2.8 shows the animation editor in the Unity game engine. Through this interface, multiple properties of the selected object can be animated. A keyframe consists of the value of a property at a given point in time. Diamond symbols indicate keyframes on the timeline. Properties such as position, rotation, and scale, are smoothly interpolated between keyframes, while others, such as whether the object can collide, are not.

Animations are not limited to the properties of objects. Often we also want to animate the geometry of an object. One naïve implementation is storing all the vertices for each frame of our animation. This technique is analogous to drawing each frame of a 2D animated film and imposes virtually no restrictions on the artist. However, while this works great for meshes with few vertices, the memory bandwidth requirement can be prohibitively high for detailed meshes [4]. Instead, two other techniques are commonly used: vertex blending and morphing.

Vertex blending

Vertex blending is a common technique used to animate and pose models [4]. The technique consists of two parts: the skin (mesh) and the bones which form a skeleton. The application of this technique is also often referred to as skinning because the geometry forms something akin to a skin around the virtual bones. As shown in Figure 2.9, a series of virtual bones are placed in the mesh. The virtual bones are represented by transforms at each joint, and form a hierarchy. The skin consists of vertices along with weights defining how much each vertex is affected by the virtual bones.

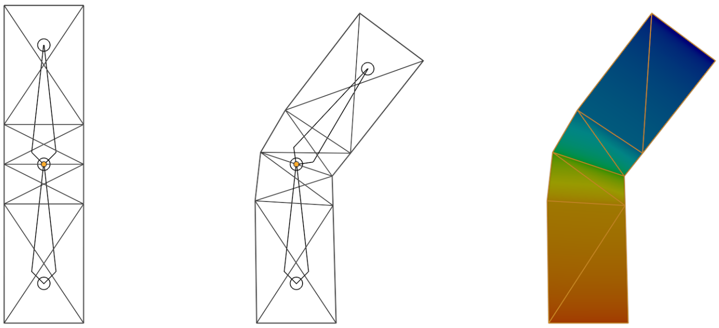


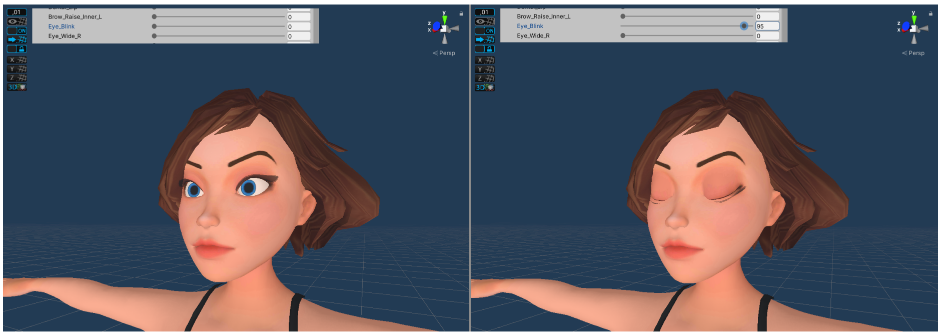
Figure 2.9: Vertex blending: On the left the bone has not been rotated, in the middle the bone has been rotated, on the right vertex weights are visualized with a gradient

As we can see in the right part of Figure 2.9, rotating the joint where the two bones connect moves the surrounding vertices accordingly. Notice the smooth transition to the lower bone. The uppermost vertices are affected entirely by the transform of the upper bone, but as we move down towards the lower bone, we see the vertices being gradually affected more by the lower bone's transform. The hierarchical nature of this setup lends itself well to model vertebrates. For example, rotating the spine of a humanoid will not only result in the surrounding geometry moving but also the entire upper body.

The process of adding virtual bones to a mesh is often referred to as rigging, while the process of assigning weights to the vertices is called skinning. A mesh is rigged when virtual bones have been added. Skinning can be done through manual assignment but can also be done more efficiently through a process similar to painting [32].

Morphing

Morphing is another common way of animating models [4]. The technique works by interpolating between multiple sets of vertices. One problem that must be solved to apply this technique is figuring out which vertices in one set correspond to which vertices in another set. One common way to solve, or rather avoid, this problem is to require a one-to-one relationship between the sets of vertices. In other words, the number of vertices must be equivalent, and each vertex must be at the same relative location in memory. With this setup, vertices can be passed to the GPU and interpolated on a per-vertex basis. One common variant of morphing is *blend shapes* or *morph targets*. This technique works by choosing a set of base vertices and, rather than storing the other sets of vertices as is, storing the displacement or difference between the base vertices and the other vertices. A weight is assigned to each set of vertices representing how much impact those vertices have on the final result.



Brow_Raise_Inner_L 0
Eye_Blink 0
Eye_Wide_R 0

Brow_Raise_Inner_L 0
Eye_Blink 95
Eye_Wide_R 0

Figure 2.10: Example of using blend shapes to close the eyelids of a humanoid model (composited screenshots from Unity 2019.3 editor)

In Figure 2.10 we see the humanoid blinking as a consequence of increasing the weight of the blend shape. A model can have many blend shapes. Blending between an arbitrary number of blend shapes is not always possible due to performance limitations. Hence, the implementation typically blends between a limited number of the blend shapes based on which are weighed most heavily at that moment. The weights can be modified programmatically. Blend shapes can be used, among other things, to impart a more lifelike quality to characters, for example, by making them react to the player's actions.

2.5 Virtual reality

VR technology has seen tremendous improvement over the last few years. VR allows the user to experience and interact with computer generated simulations in a highly immersive manner. It primarily does so by providing alternative visual stimuli while reducing external visual stimulation, usually through the use of a head-mounted display. Immersion can be increased further by providing other types of sensory stimuli, such as auditory and haptic. VR is covered by the umbrella term Extended Reality or XR for short, which includes the entire spectrum of the reality-virtuality continuum [7].

Figure 2.11: HTC Vive VR headset, photo by Jesper Aggergaard at Unsplash

2.5.1 Technology

Modern VR is usually experienced through a Head-mounted display (HMD). The HMD is worn on the head similarly to how a helmet is worn. The entire field of view is engrossed by the inside of the HMD, which has one or more screens on the inside, with one lens per eye. A variety of HMDs are available on the market, some are powered by a computer or a smartphone while others are entirely standalone.

Smartphone powered VR systems consists of an enclosure for the smartphone which can be mounted to the head. Some of these smartphone powered HMDs have additional input devices, such as Samsung Gear VR [33], which allows you to use peripherals such as controllers. Others are even simpler, where there are no controllers and the only way to provide input is through head movement. These kinds systems are usually limited to three degrees of freedom in the form of rotational tracking. This essentially means that translational movement is not tracked, so applications are usually enjoyed while stationary. Perhaps the largest drawback of the smartphone powered approach is the limited graphical processing power. In a field where performance is very important (see Section 2.5.3), users may opt to invest more to get a more powerful VR experience.

The Oculus Quest (shown in Figure 2.12) is an example of an entirely standalone VR system. The hardware in the device is similar to a smartphone in capability, and even runs the same operating system (Android), but is adapted and tailored specifically for VR. The device provides tracking with six degrees of freedom, which means that both rotational and translational movement can be accounted for. This in turn means that experiences are not limited to a stationary setup, but can also be room-scale. Room-scale experiences allow the player to move and play more freely, albeit restricted to a predefined boundary. The obvious benefit of this kind of device is the ease-of-use. No external trackers are necessary meaning the headset can be brought around and used at a variety of locations. Similarly to smartphone powered headsets this kind of headset comes with the drawback of having fairly limited graphical processing power.

Finally, there are the computer powered VR systems. These systems rely on a connection to a computer and sometimes additional external hardware in the form of trackers. One example of this is the HTC Vive (Figure 2.11) which requires the Lighthouse Trackers [34] from Valve in order to operate. Certain headsets do not require external trackers and instead use a form of inside-out tracking. The Oculus Rift S, for example, uses a combination of 5 cameras and an accelerometer, on the HMD, and AI to produce tracking data [35]. The VR system is connected to a computer and relies on the computer to do the processing. One can therefore use high end CPU and GPU hardware to render more detailed and higher quality scenes at an acceptable frame rate. The drawback however is that the player must be tethered to the computer, making the setup less portable and the player potentially less mobile. This can be alleviated somewhat with wireless technology, but still requires a computer for its processing power.

2.5.2 Applications

There are a lot of commercially available VR titles currently on the market. A filtered search on Steam, which at the time of writing is one of the most popular video game marketplaces, shows that there are over five thousand applications with VR support [36]. The vast majority of these support the HTC Vive, with Valve Index and Oculus Rift trailing behind while still supported by the majority of applications. Windows Mixed Reality is by far the least supported out of the four filter options.

Due to the standalone nature of the Oculus Quest a regular user is typically

Figure 2.12: Oculus Quest, courtesy of Oculus

restricted to the Oculus Store, which is a platform-specific marketplace. In an article published by UploadVR, tech journalist Harry Baker could find 170 apps on the Oculus Quest Store in April of 2020 [37]. However, a marketplace called SideQuest is available, which contains a large number of applications and games. SideQuest provides a less restrictive platform than the Oculus Store for developers, but it requires the end user to enable developer-mode on the device to install applications [38]. Additionally, as of November 2019, the headset also supports tethering with Oculus Link, meaning certain PC-based marketplaces are available as well [39].

2.5.3 Concerns and challenges

In Real-time Rendering [4, pp. 915], Akenine-Möller et al. sums up the challenges of VR as the difficulty of balancing performance, comfort, freedom of movement, price and other factors. Many of these are hardware related challenges, like creating comfortable head-gear with good tracking and making effective input devices that are intuitive. However, a software developer developing for VR must also be aware of the limitations and effects the software can have on the people using it. Since one is so immersed in the scene when using a VR headset, it is important to keep in mind that when the world does not act as the user expects it to, the senses will be confused.

One particularly large concern is that latency has a much stronger adverse effect on a user in an immersive virtual reality than it would have on a standard display monitor. The effects are often called *simulation sickness* and can involve dizziness and nausea. The problem arises when the display does not match the expectation caused by the other senses, such as the inner ear's vestibular system of balance and motion.

Michael Abrash at Valve Software, in a much-cited blog post, anecdotally stated that latency of 20 milliseconds is too much and that the threshold for ”too much latency for VR” lies between 7 and 20 milliseconds [40]. Latency can arise from several sources. Getting the tracking data from the HMD and controllers is one such source. Another source is the refresh rate of the display. The HTC Vive Pro has a hardware-locked display refresh rate of 90 Hz, which equates to a minimum latency of approximately 11 milliseconds [41]. Developers of VR applications must be mindful of performance as not to cause latency beyond these thresholds.

2.5.4 Best practices

Designing an interface for VR is different from designing it for a standard display monitor. Visual elements like a Heads-up display (HUD) in a game may have to be displayed differently. A HUD can show information such as a map, player resources, equipment and a list of current objectives. On a standard display monitor, the HUD is usually fixed in place no matter which angle the player is looking. Figure 2.13 shows what a game with and without a HUD looks like. For games played on traditional screens, this is a very effective way of showing information to the player at all times. However, the fact that HUDs follow the camera angle all times turns out to be annoying in VR [42]. Since the screens are very close to the eyes, players get a stronger feeling of the HUD being ”in the way”. Additionally, players who want to read parts of the HUD that are placed on the side of the view will often instinctively turn their heads to look at it. Since the HUD follows the camera angle, it will be pushed away from the view of the player.

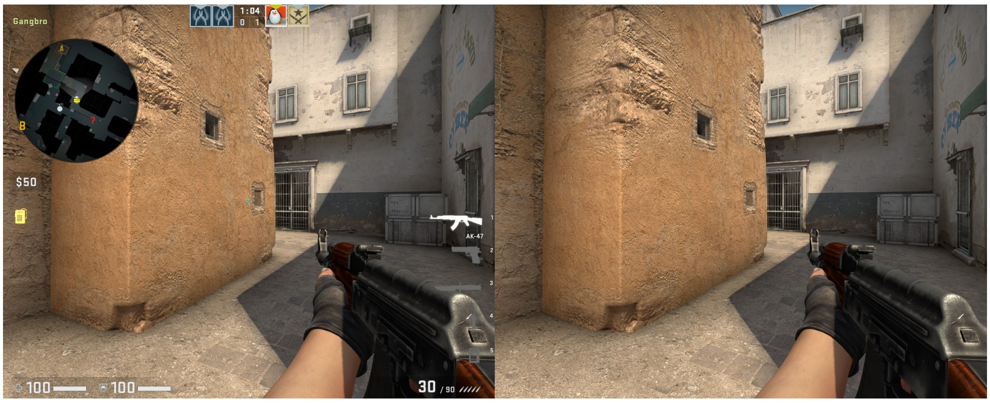


Figure 2.13: Side by side comparison of how a game looks like with and without a HUD. Screenshots from shooter game Counter-Strike: Global Offensive.

Developers should instead place important information around in the scene, for example, on walls or following the player character at a certain distance. If it does follow the player, it should not follow the view angle vertically. This is only one example of a design choice that have been established by experienced VR developers; advice on these issues have been put together by companies with a large investment in VR development, such as Oculus and Unity [43] [44].

2.5.5 Standardization efforts

As software developers, we would often like to target multiple platforms and do so without having to increase the amount of work. The same goes for development for VR. At the time of writing, the VR ecosystem is in the midst of a standardization effort. The ecosystem consists of multiple hardware vendors, such as HTC, Valve, and Oculus. These vendors provide drivers for their hardware along with various APIs, such as SteamVR (OpenVR) and Oculus VR PC. Game engines often provide abstractions over these APIs and offer a single consolidated API, which means that the effort of targeting multiple platforms is left to the engine developers. However, projects that do not utilize a pre-existing game engine must potentially implement such an abstraction from scratch.

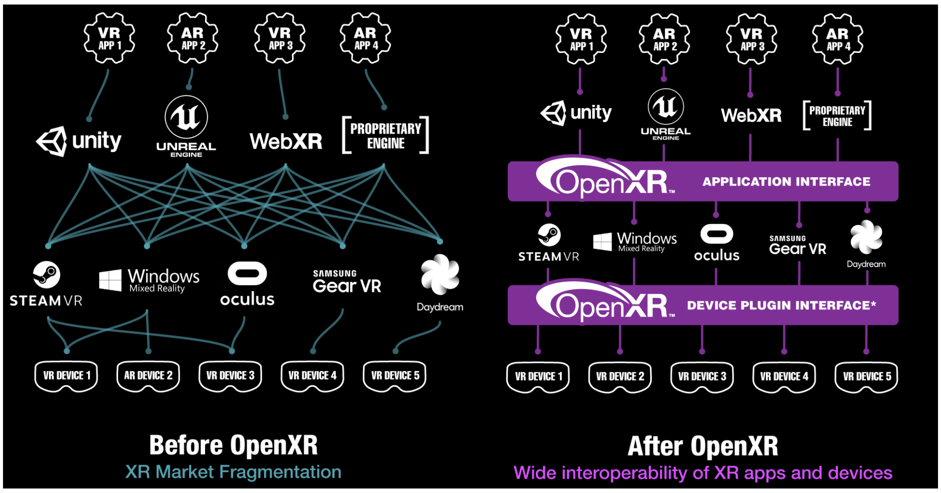


Figure 2.14: OpenXR standardization effort before and after (figure from Khronos Group)

In July 29th 2019 the Khronos Group released an open standard for access to XR platforms and devices [45]. The standard seeks to reduce fragmentation, simplify development and enable applications to target a wide array of platforms without having to be rewritten. Figure 2.14 shows the intention of the standard. As we can see on the left, without OpenXR, applications and game engines must interact with a large number of interfaces. In contrast, with OpenXR, as shown on the left, this work is reduced to programming against a single interface. An additional possible benefit of the standard is that it makes it easier for hardware developers to enter the market because software developers will not have the burden of writing code specifically for that hardware. At the time of writing, all major VR hardware vendors have given their support to the standard.

2.6 Game engine

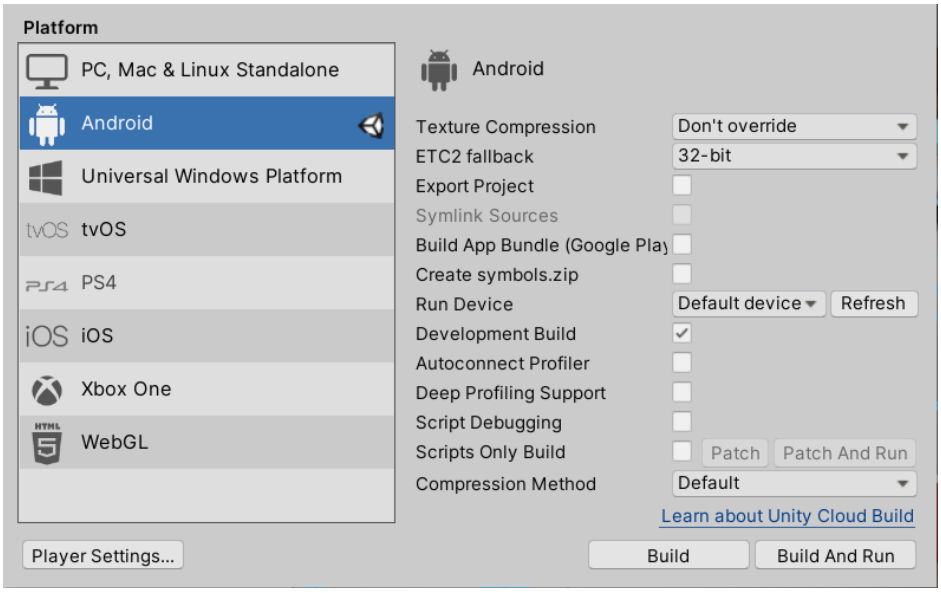
The process of developing a graphics engine from scratch is a highly demanding and costly endeavour. Numerous techniques, such as shadow mapping and global illumination that require much work to implement, are often taken for granted. Taking it a step further, developing a game engine requires even more resources because of the many complicated systems that need to fit together. With a growing interest in serious games, where development teams often have somewhat limited resources, existing game engines can offer the ground-work. Modern game engines exist to make the development of games faster and more cost-effective. They do so by providing implementations for techniques that are ubiquitous across a large number of genres. Few games are made entirely from scratch anymore, as game studios have realized that reuse of base engines are often more cost-effective. This statement is undoubtedly true for many small game studios, but many large game studios also reuse either in-house game engines or utilise commercial ones. The engine reduces the complexity for the developer, providing an uncomplicated API that supports standard features, such as rendering, physics simulation, audio, and animation. Implementing all of these lower level features is left for the game engine developers.

Additionally, popular game engines often come with a built-in editor with an interface where developers can quickly create scenes with game objects and add their components to it. These components can be several different things; one example is geometry, which can be rendered by the engine; another is a script for controlling behaviour in the scene. Using a game engine is a natural choice for this thesis as it provides many features that would otherwise have to be implemented from the ground up, which is neither feasible within the given time and resource constraints nor relevant to our goal.

2.6.1 Why Unity?

At the start of the project, we had to choose a game engine. Initially, we considered two popular engines: Unity and Unreal Engine. Both engines sport all necessary features required for this project and have large developer communities. We chose Unity for two reasons. Firstly, both authors have prior experience with using the engine, which meant that we spent less time getting a working prototype up and running. Secondly, using the same engine as the other VR projects that were working alongside us made it possible to share tips and tricks throughout the development process.

At the time of writing Unity[46] is one of the most popular game engines on the market. It is especially popular among small game studios and amateurs, as well as developers targeting mobile platforms. Being able to deploy to multiple platforms is an integral part of many game engines, and Unity is no exception. As shown in Figure 2.15, we can target the three major desktop operating systems, Android (which includes Android-based VR headsets like the Oculus Quest), videogame consoles, web browsers with WebGL, and more.



Android

Texture Compression	Don't override
ETC2 fallback	32-bit
Export Project	☐
Symlink Sources	☐
Build App Bundle (Google Play	☐
Create symbols.zip	☐
Run Device	Default device
Development Build	☑
Autoconnect Profiler	☐
Deep Profiling Support	☐
Script Debugging	☐
Scripts Only Build	☐
Compression Method	Default

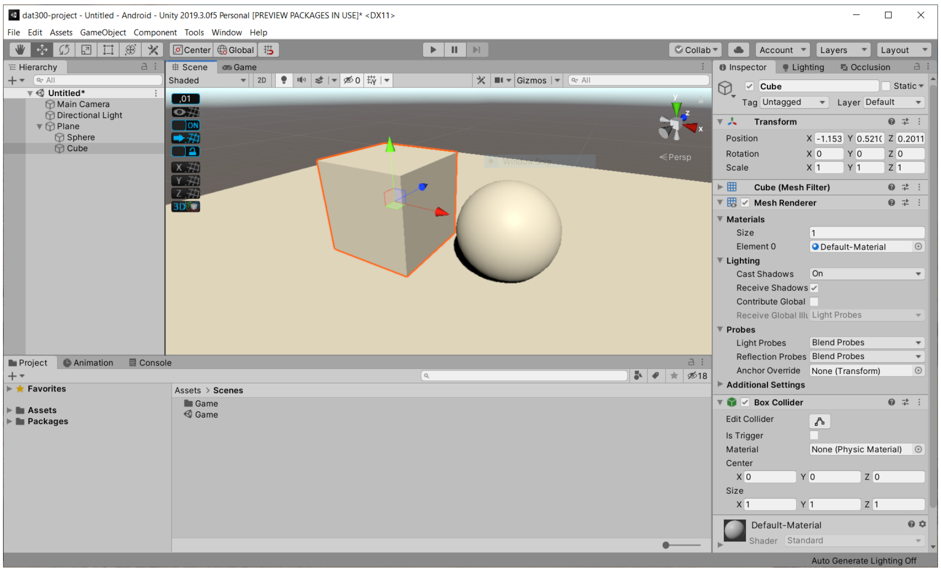
Figure 2.15: Unity build settings showing targetable platforms (screenshot from the Unity 2019.3 editor)

Depending on the application, building for various platforms will require few changes to the code.

2.6.2 Editor

An integral part of Unity is the editor. The game engine consists of several systems (such as physics, rendering and audio) that work together. The editor provides a graphical user interface for working with all these systems and using them to create a game. In Figure 2.16, we see that the editor is divided into multiple panels. The panels can be added, adjusted and closed by the user. A tab in one of the panels can host a window that exposes some functionality to the user. In the centre, we see the *Scene*-view. This window is arguably the most important one and allows the user to move around, view and edit a scene. On the left, we see the *Hierarchy*, which shows the entire scene as a hierarchical list and allows the user to modify and organize the scene. The right panel hosts the *Inspector*. The user can use the *Inspector* to modify objects in the scene. For instance, the user can modify the position, rotation and scale of the object, as well as add and remove various components.

Components allow us to add functionality to our game. All objects have a *Transform*-component which represents the position, rotation and scale. Unity comes with numerous built-in components which expose the features of the game engine. On example is the rendering related components *MeshFilter* and *MeshRenderer*, which makes it possible to add and render geometry. Another example is various physics components such as *BoxCollider* and *RigidBody*. In addition to the built-in components, the developer can program custom ones in C# by extending the *MonoBehaviour*-class. The class exposes several methods that can be overridden. One commonly used method is *Update*, which is called once every frame by the engine. Custom components can be thought of as the glue that binds together a game with its gameplay systems and mechanics.


dat300-project - Untitled - Android - Unity 2019.3.0f5 Personal [PREVIEW PACKAGES IN USE]* <DX11>

Cube ☑ Static □
Tag Untagged Layer Default

Transform
Position X -1.153 Y 0.521(Z 0.2011
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)

Mesh Renderer ☑
Materials
Size 1
Element 0 Default-Material
Lighting
Cast Shadows On
Receive Shadows ☑
Contribute Global □
Receive Global Illu Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings

Box Collider ☑
Edit Collider
Is Trigger □
Material None (Physic Material)
Center
X 0 Y 0 Z 0
Size
X 1 Y 1 Z 1

Default-Material
Shader Standard

Auto Generate Lighting Off

Figure 2.16: Unity 2019.3 editor: Overview

Another part of the Unity editor is an online marketplace called the Asset Store[47]. The Asset Store an integrated part of the editor. Here developers can browse for assets to use in their projects. Unity Technologies themselves publish some assets, but third-party developers publish the majority of the assets. Publishers can put a price on their asset or give them away for free. assets vary widely in form, with the largest category being 3D models, and often come in collections. As a small development team, having an easy way to acquire relevant assets can be invaluable. In our case, with our programming-heavy skill set and a lack of modelling and content creating skills, most assets we have acquired are in the form of 3D models and audio.

2.6.3 Virtual Reality support

Support for VR in Unity has gone through some changes throughout our project. Most notably, when we upgraded the project from Unity version 2019.2 to 2019.3. Version 2019.2 had built-in native support for various VR devices, such as OpenVR and Oculus. In 2019.3 the move was made to a plugin-based architecture, an overview is shown in Figure 2.17.

As we can see on the bottom part of the figure, with this architecture, hardware vendors must implement a plugin that interfaces with Unity's XR SDK. At the time of writing, OpenVR does not provide such a plugin and instead

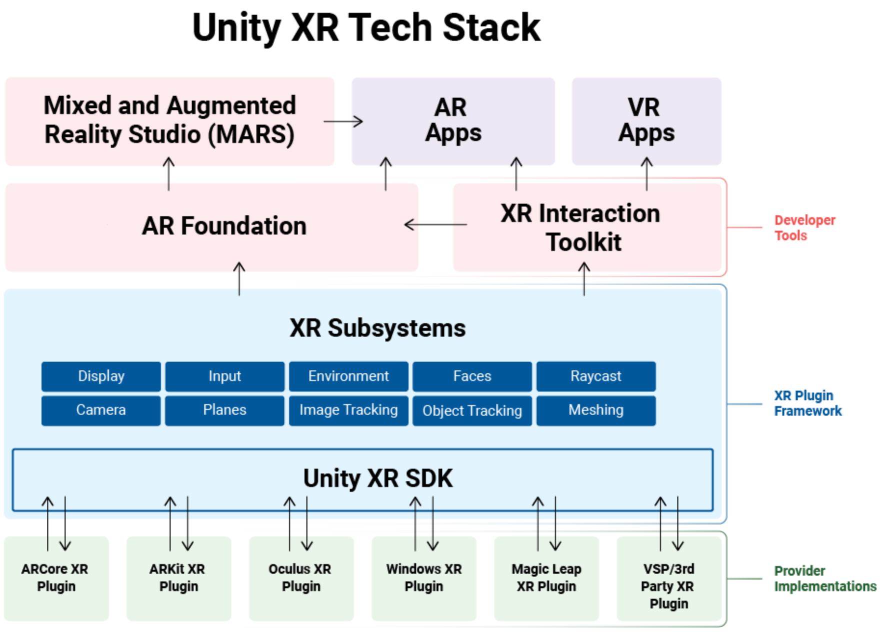


Figure 2.17: Unity XR Tech Stack (from Unity 2019.3 documentation, accessed: 19/05/2020)

relies on the deprecated built-in implementation from 2019.2. Therefore our project must use the deprecated implementation. Over time, if the standardization efforts mentioned in Section 2.5.5 are successful, this architecture can be simplified because multiple plugins will no longer be necessary to interface with the hardware.

Unity exposes the API for accessing the capabilities of a VR headset in the *UnityEngine.XR*-namespace. We can access the various classes and enumerations in this namespace through scripts (written in C#). One notable example is the *XRSettings*-class. The class exposes global properties (with the *static* keyword in C#) such as *supportedDevices*, *loadedDeviceName*, *isDeviceActive*, as well as a global function called *LoadDeviceByName* which can be used to load a specific device. With a device loaded, global functions in the *InputDevices*-class can be used to acquire an input device, such as an HMD or controller. Given an input device that supports tracking, a script can access values such as position, rotation and acceleration and use it, for example, to move the camera or a virtual controller. Using this API directly works fine; however, much of the functionality needed for an application in VR is common to many applications. Instead, we can use a framework which provides common VR functionality.

The SteamVR Unity Plugin [48] is one such framework which provides common functionality such as various forms of hand interaction and player movement. Also, the framework provides a more advanced feature called *skeleton input*, which enables a detailed representation of the player's hands based on controller input. How detailed the player's hands are represented depends of course on the capability of the controller hardware. Although the plugin works with a wide array of VR headsets, it does require the SteamVR runtime, provided by Valve, to be installed and running, which means that standalone headsets such as the Oculus Quest are not supported. Oculus Integration [49] is another similar framework. The framework initially only targeted Oculus devices, but later extended support to OpenVR. As shown in Figure 2.17, Unity also provides a framework called the XR Interaction Toolkit [50]. Similarly to the previously mentioned frameworks, this framework also provides commonly-used functionality but is entirely platform-independent.

Chapter 3

Design and Solution

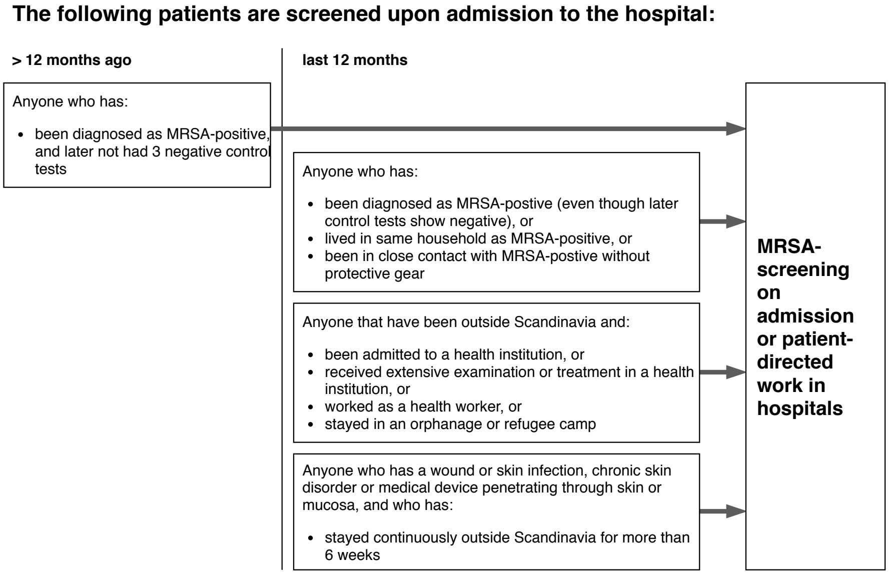
In this chapter we describe the steps required to design and implement a VR game based on Helse Bergen's CPG for MRSA infection control (Appendix A). We begin by providing a high-level overview of how and why we transform various parts of the guideline into the game. Next, we present the resulting gameplay. After which, we propose a top-level architecture for the game. Finally, we go into detail about how we implemented the components of this architecture and how we solved the various challenges that arose.

3.1 From guideline to game

A clinical practice guideline (CPG) is a collection of systematically developed recommendations to assist practitioner and patient make decisions about the appropriate health care for a particular circumstance [24]. As mentioned in Section 1.3 and 1.4, we base the game on recommendations that apply to as many practitioners as possible. Naturally, Helse Bergen's CPG (Appendix A, [51]) consists of some recommendations that are particular to Bergen. Parts of the guideline that contain these local recommendations are left out. Together with our domain expert, Eva C. Backer, we identified two parts of the guideline as suitable, and they became core components of the game. This decision was made, firstly, because they are crucial for preventing the spread of MRSA, and secondly, because there is little variation in how practitioners apply the recommendations. The first part we identified is the criteria for screening a patient (Figure 3.1).

Next is the list of the locations from where the practitioner must collect samples, which holds the following entries:

- Both nostrils (same swab)
- Throat including tonsils
- Perineum (the space between the anus and scrotum or vulva)



The following patients are screened upon admission to the hospital:

> 12 months ago

last 12 months

Anyone who has:

- been diagnosed as MRSA-positive, and later not had 3 negative control tests

Anyone who has:

- been diagnosed as MRSA-postive (even though later control tests show negative), or
- lived in same household as MRSA-positive, or
- been in close contact with MRSA-postive without protective gear

Anyone that have been outside Scandinavia and:

- been admitted to a health institution, or
- received extensive examination or treatment in a health institution, or
- worked as a health worker, or
- stayed in an orphanage or refugee camp

Anyone who has a wound or skin infection, chronic skin disorder or medical device penetrating through skin or mucosa, and who has:

- stayed continuously outside Scandinavia for more than 6 weeks

MRSA-screening on admission or patient-directed work in hospitals

Figure 3.1: Screening criteria diagram from Helse Bergen's guideline (Appendix A, translated to English).

- Wounds, eczema, puss, scars from infection or active skin diseases
- Around insertion site for foreign matter (such as a catheter, drainage and tracheostomy)
- Urine from catheter

In addition to the reasons mentioned above, the latter part, i.e. collecting samples from the patient, appeared to fit well with the physical aspect of VR. Having the player physically carry out the task of collecting samples allows us to make the most of the properties of VR.

As well as consulting our domain expert, we also visited a nurse at the infection control ward at Haukeland hospital. Through this visit, we identified another part of the guideline as relevant to the game, namely droplet-transmission precautions. Droplet-transmission precautions are precautions a practitioner must take when faced with diseases and germs that can be spread through small droplets in the air as a result of coughing and sneezing. When interacting with and screening a patient, the precautions typically involve isolation of the patient, proper hand hygiene and the use of protective equipment such as a coat, mask and gloves.

The next step is turning the three identified parts into gameplay elements. As mentioned in Section 1.4, in the interest of developing a publishable end product, the game should consist of a single flow. The flow for the game was devised based on the most natural order in which a practitioner applies each part. First, the practitioner uses the screening criteria to decide whether or not to screen a patient. Next, they must follow the proper droplet-transmission precautions. Finally, they collect the appropriate samples from the patient.

3.2 Gameplay

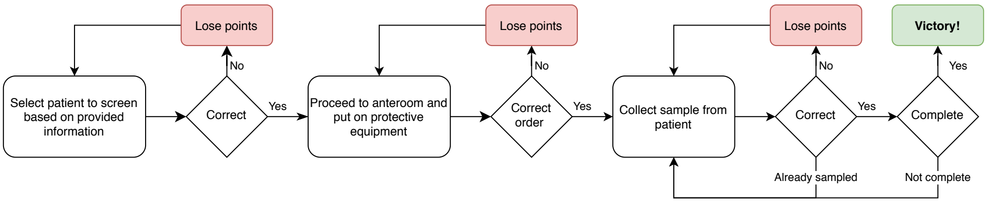
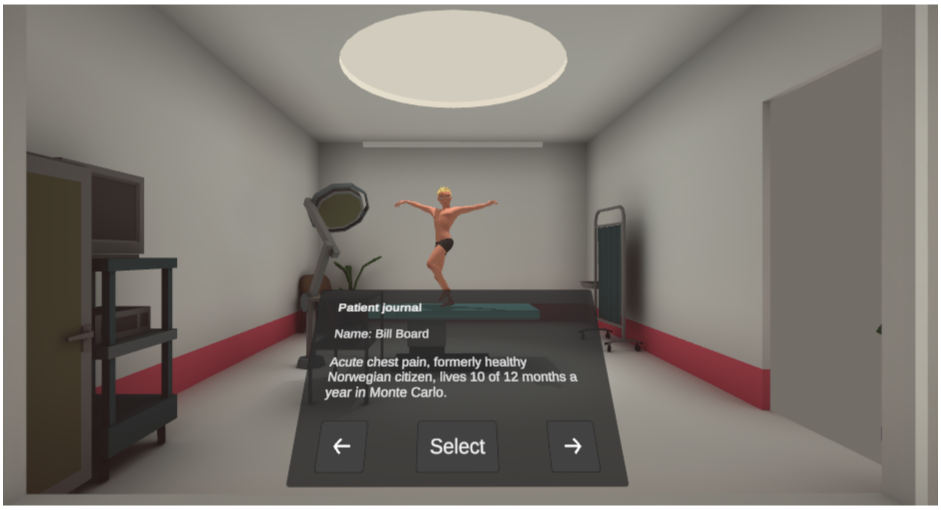

Lose points
Lose points
Lose points
Victory!
No
No
No
Yes
Select patient to screen based on provided information
Correct
Yes
Proceed to anteroom and put on protective equipment
Correct order
Yes
Collect sample from patient
Correct
Yes
Complete
Already sampled
Not complete

Figure 3.2: Gameplay flow diagram

Figure 3.2 shows the flow of the gameplay. The first task is to select the patient that needs to be screened using the screening criteria (see Figure 3.1). The player is presented with a set of patients with an accompanying short description. The player's task is to select the patient that fulfils the criteria based on the patient's description. Upon choosing the correct patient, the player can proceed to the anteroom. Choosing the wrong patient results in a loss of points, but the player can attempt several times until the correct patient is selected.



Patient journal

Name: Bill Board

Acute chest pain, formerly healthy Norwegian citizen, lives 10 of 12 months a year in Monte Carlo.

Figure 3.3: Patient selection (in-game screenshot)

As shown in Figure 3.3, while browsing the patients, the currently viewed one can be seen in the other room through a window. To reach the patient, once the correct one has been selected, the player must first pass through the anteroom. Upon entering the anteroom, the door behind is closed. The player must now perform necessary droplet-transmission precautions, which consists of applying hand sanitizer, equipping a coat and mask, and finally putting on gloves. The player must do so in the correct order; not doing so results in a loss of points and a prompt appearing, which reminds the player that it needs to be done correctly. Once the player has successfully performed the necessary droplet-transmission precautions, the last door opens, leading the player to the patient.

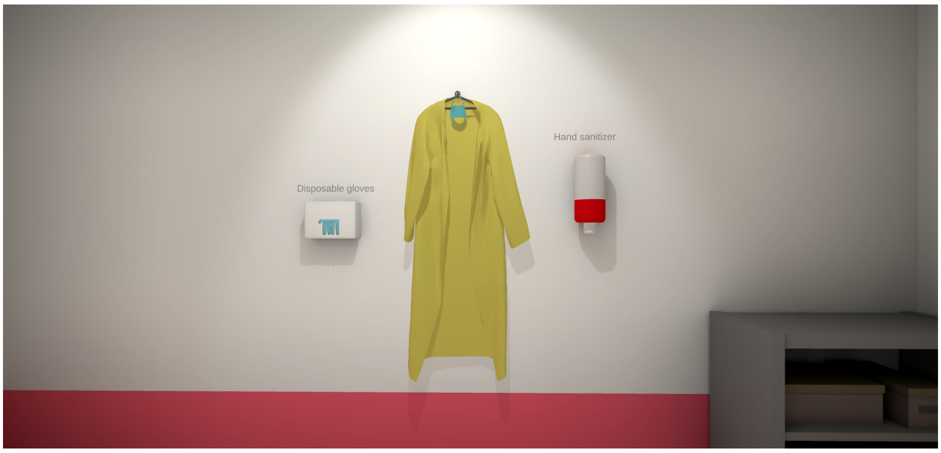


Figure 3.4: Anteroom with hand sanitizer and protective equipment (in-game screenshot)

The final task is to collect the appropriate samples from the patient. The player must collect samples from the nostrils, throat and perineum. Additionally, if the patient has a wound, it must also be sampled, and if the patient has a urinary catheter, then urine must be collected. For a collected sample to be correct, certain conditions must be met. The correct tool must be used at the correct location, for example, a swab for the nostrils or a syringe for the urinary catheter. The tool can not have been contaminated from touching a different surface beforehand. Lastly, the location can not already have been sampled. However, there is no punishment if you do sample the same spot twice. Instead, you are just given feedback that it is already done.

This gameplay element is implemented with physical interaction with a patient. The patient is lying on the hospital bed, as seen in Figure 3.6. Near the bed, there is a table with some tools the player can grab. Only some of the tools are necessary; others are not suitable and only lead to a loss of points if used. Figure 3.5 shows an overview of the tools. The irrelevant tools are meant to make the player think through their choice of tools and can also work as a humoristic element. The locations for where to collect samples are represented by a set of targets on the patient's body. To collect a sample, the player must grab a tool and use it to interact with a target physically. A successfully collected sample rewards the player with points. Also, the location shows up on a visible list so that the player can track their progress. An unsuccessful sampling attempt results in a loss of points, the playback of a negative sound effect, as well as removal of the tool from the player's hand.

Once all samples have been collected, a prompt appears. It informs the player

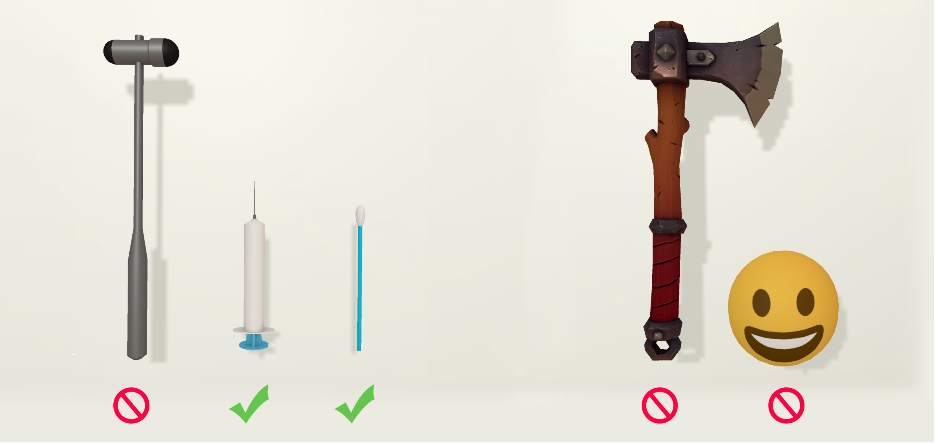


Figure 3.5: Overview of available tools

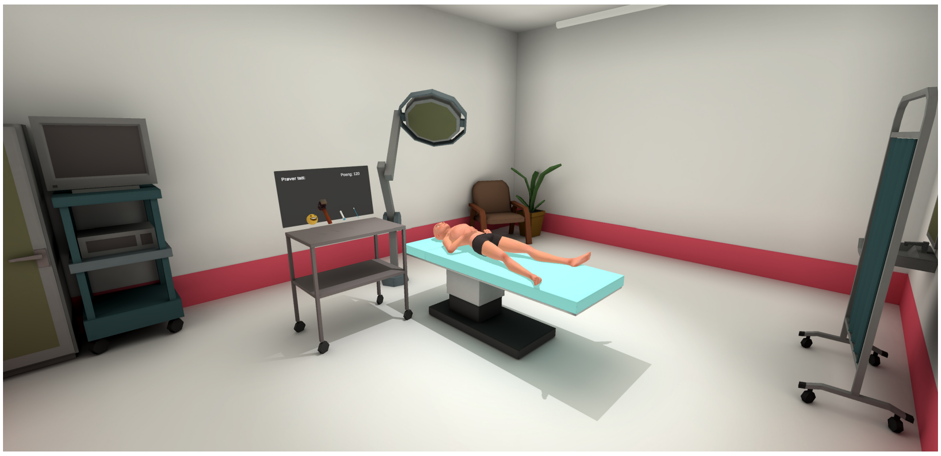
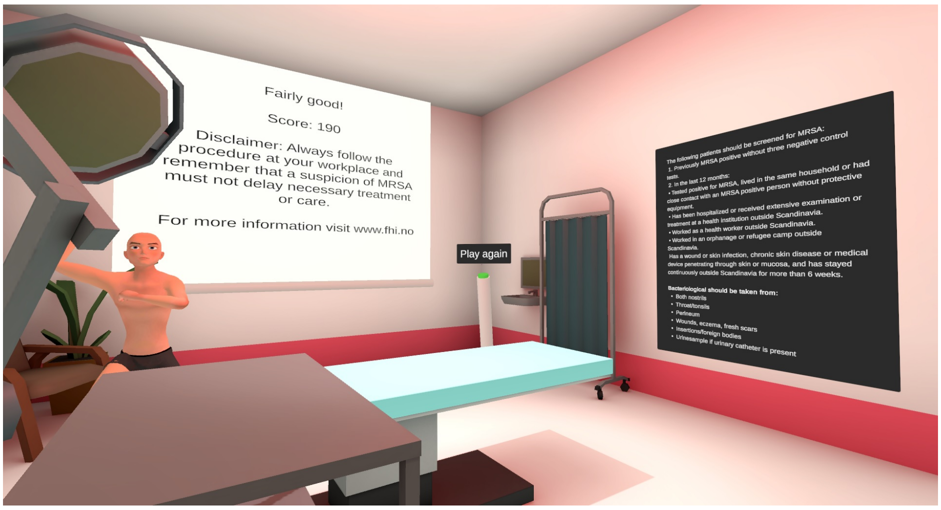


Figure 3.6: Screening (in-game screenshot)

how samples should be labelled before they are dispatched for testing. Upon closing the prompt, the room transforms into an informational discotheque, as seen in Figure 3.7. The tools disappear, the patient dances on the floor instead of lying on the bed, disco music starts playing, and the ceiling light starts looping colourfully. The party atmosphere is intended to give the player a sense of achievement for being victorious, as well as to help shift focus to other regions of the room. A rolling projection screen rolls down, presenting the player with their score, a rough assessment of how they did in plain words, and a disclaimer that they should follow the guidelines of their workplace. Additionally, a summary of the guideline appears on the wall, consisting of the screening criteria and locations. The purpose of this is to make sure the player understands why what they did was correct or incorrect. The only interaction left in the room is the ”Play again” button, which allows the players to play through the game again with a new set of patients.



Fairly good!

Score: 190

Disclaimer: Always follow the procedure at your workplace and remember that a suspicion of MRSA must not delay necessary treatment or care.

For more information visit www.fhi.no

The following patients should be screened for MRSA:
1. Previously MRSA positive without three negative control tests.
2. In the last 12 months:
- Tested positive for MRSA, lived in the same household or had close contact with an MRSA positive person without protective equipment.
- Has been hospitalized or received extensive examination or treatment at a health institution outside Scandinavia.
- Worked as a health worker outside Scandinavia.
- Worked in an orphanage or refugee camp outside Scandinavia.

Has a wound or skin infection, chronic skin disease or medical device penetrating through skin or mucosa, and has stayed continuously outside Scandinavia for more than 6 weeks.

Bacteriological should be taken from:
- Both nostrils
- Throat/tonsils
- Perineum
- Wounds, eczema, fresh scars
- Insertions/foreign bodies
- Urinesample if urinary catheter is present

Figure 3.7: The last room after having completed the game as seen from the corner

3.3 Top-level architecture

The top-level architecture of the game consists of two parts, the game logic and the VR interface. The game logic is an implementation of the gameplay functionality, as shown in Figure 3.2. It exposes the functions and variables that are necessary to play through the game and keeps track of the progress and score of the player. The VR interface is the implementation of everything necessary to play the game through VR. It is responsible for converting input from the player to the relevant interactions with the game logic and showing the resulting feedback. Keeping the game logic separated from the VR interface, makes it possible to target other platforms by adding new interfaces. While the VR interface of the game is tightly coupled to the game logic, the game logic has no coupling to the VR interface. Therefore the game logic can be reused for very different interfaces such as a website or a mobile application.

In the Unity editor, it is easy to place objects around in the scene that can call functions to a singular object like the game logic. To do the opposite, when you want the objects in the scene to react to events that happen in the game logic, it is more complicated. We set up an event system using UnityEvents [52] for the game logic. The game logic triggers these events based on player progression. Every object in the scene can listen and react to these events. Figure 3.8 shows the communication both ways with the full list of functions and events.

The function calls to the game logic already have return values, but that is often not enough. If we consider the example of selecting a patient to be screened, that call is made by a UI board. That board needs to know when the player has chosen the correct patient, but so does many other objects in

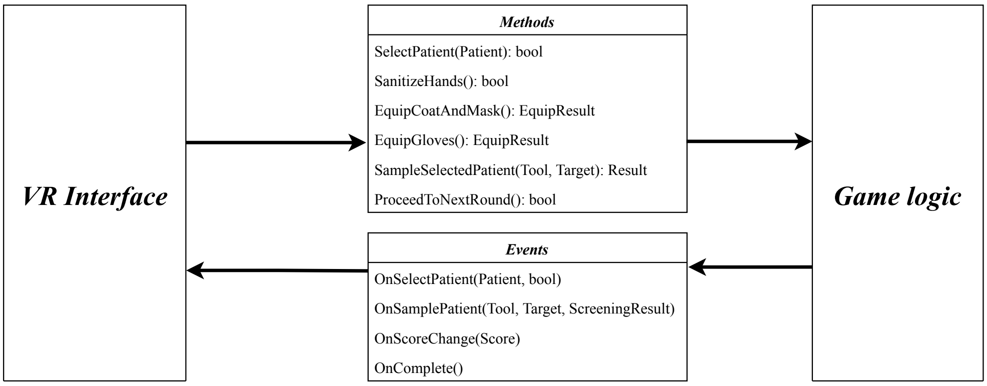

VR Interface
Methods
SelectPatient(Patient): bool
SanitizeHands(): bool
EquipCoatAndMask(): EquipResult
EquipGloves(): EquipResult
SampleSelectedPatient(Tool, Target): Result
ProceedToNextRound(): bool
Events
OnSelectPatient(Patient, bool)
OnSamplePatient(Tool, Target, ScreeningResult)
OnScoreChange(Score)
OnComplete()
Game logic

Figure 3.8: Top-level architecture diagram

the scene. For instance, the patient reacts with a celebratory animation, the door to the next room opens, and the score on the scoreboard changes. If it were the UI board's task to bring the information to all these objects, we would have a highly coupled interface. If every component that triggers such events had such a high coupling, the code would end up unnecessarily intricate and hard to manage. By using events, we avoid this problem.

3.4 Game logic

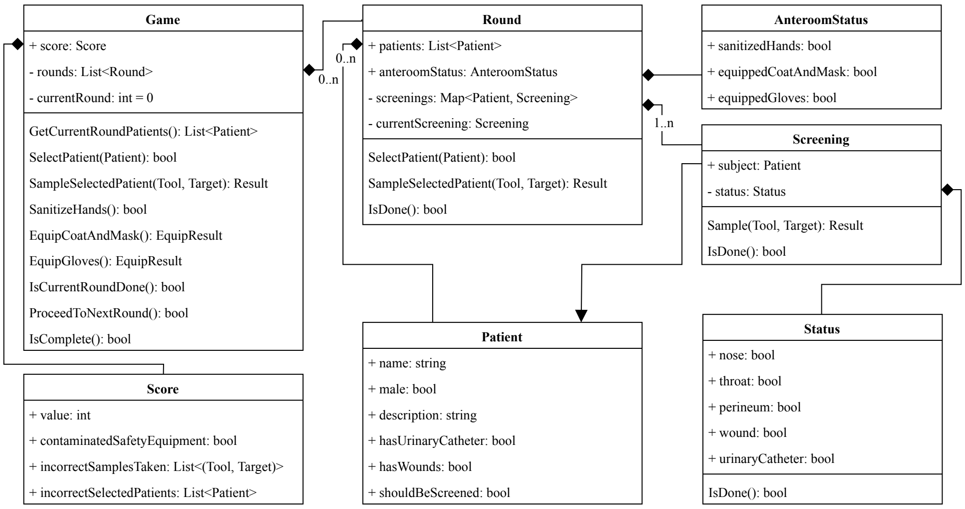

Game
+ score: Score
- rounds: List<Round>
- currentRound: int = 0
GetCurrentRoundPatients(): List<Patient>
SelectPatient(Patient): bool
SampleSelectedPatient(Tool, Target): Result
SanitizeHands(): bool
EquipCoatAndMask(): EquipResult
EquipGloves(): EquipResult
IsCurrentRoundDone(): bool
ProceedToNextRound(): bool
IsComplete(): bool
0..n
0..n
Round
+ patients: List<Patient>
+ anteroomStatus: AnteroomStatus
- screenings: Map<Patient, Screening>
- currentScreening: Screening
SelectPatient(Patient): bool
SampleSelectedPatient(Tool, Target): Result
IsDone(): bool
1..n
AnteroomStatus
+ sanitizedHands: bool
+ equippedCoatAndMask: bool
+ equippedGloves: bool
Screening
+ subject: Patient
- status: Status
Sample(Tool, Target): Result
IsDone(): bool
Score
+ value: int
+ contaminatedSafetyEquipment: bool
+ incorrectSamplesTaken: List<(Tool, Target)>
+ incorrectSelectedPatients: List<Patient>
Patient
+ name: string
+ male: bool
+ description: string
+ hasUrinaryCatheter: bool
+ hasWounds: bool
+ shouldBeScreened: bool
Status
+ nose: bool
+ throat: bool
+ perineum: bool
+ wound: bool
+ urinaryCatheter: bool
IsDone(): bool

Figure 3.9: Game logic class diagram

The game logic is the implementation of the gameplay functionality, as shown in Figure 3.2. It exposes functions to play the game and keeps track of the progress and score of the player. At the core of the game logic is the *Game*-class, as shown in Figure 3.9. This class has several methods that correspond to the actions that the player can perform. Performing an action can result in the player making progress, as well as an increase or decrease in the score.

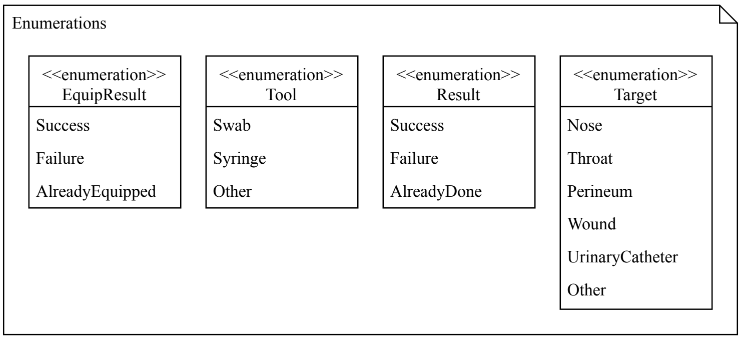


Figure 3.10: Game logic class diagram: Enumerations

A *Game*-instance consists of several rounds, each *Round* having its own set of patients that the player should correctly screen. When all the rounds are completed, the player has completed the game. The final version of the game consists of only one round. One round was found as the most suitable number of rounds through testing with our advisors at Helse Vest IKT. The reason was that we found that the amount of time spent to complete the game was adequate. Note, however, that the functionality for having multiple rounds is still present, which allows for having multiple rounds in future versions.

For every round, a set of four patients are picked randomly from a predefined collection of patients. The idea of having four patients came from the earlier work by Helse Vest IKT, Stopp Sepsis (see Section 1.6), and was found, through testing, to work well. The predefined patients are loaded from a JSON file. Predefined patients can be added, removed and modified by maintainers of the game. As shown in Figure 3.9, a patient has a description containing essential information needed to determine whether they should be screened or not. Additionally, they have information that affects the screening itself, such as whether they have wounds or a urinary catheter. Lastly, they have some characteristics like a name and sex to make them uniquely identifiable.

Every patient that should be screened is referenced by a *Screening*-object, as shown in figure 3.9. This object keeps track of what locations remain to be sampled at any given time. Due to the nature of the procedure, this can differ between patients based on their medical information. For example, only patients with wounds must have their wounds sampled. When a patient is sampled, the screening considers the tool and target and returns one of three results, as shown in the *Result*-enumeration in Figure 3.10. The combination of tool and target can be incorrect, which results in the *Failure*-variant. If the target has already been sampled, it will result in the *AlreadyDone*-variant. Otherwise, the sampling is correct, and the result is the *Success*-variant. In addition to returning the result to the caller, the player loses or gains points, and relevant events, such as *OnScoreChange*, are invoked.

A *Score*-instance represents how the player is performing in the game. The *value*-field is the number of points. Whether or not the player made a mistake with regards to droplet-transmission precautions is stored in the *contaminatedSafetyEquipment*-field. Incorrectly selected patients, as well as sampling attempts where the combination of tool and location was incorrect, are stored in lists. Keeping track of these mistakes makes it possible to adapt the gameplay in various ways. We discuss this possibility further in Section 6.3.

3.5 VR interface

The VR interface is what makes it possible for the player to play the game with a VR headset. It consists of systems to handle tracking and controller input and all the elements that form the virtual scene that the gameplay takes place in, including the code that interacts with the game logic. In this section, we will describe these elements and how they work together to form the virtual experience.

3.5.1 Level design

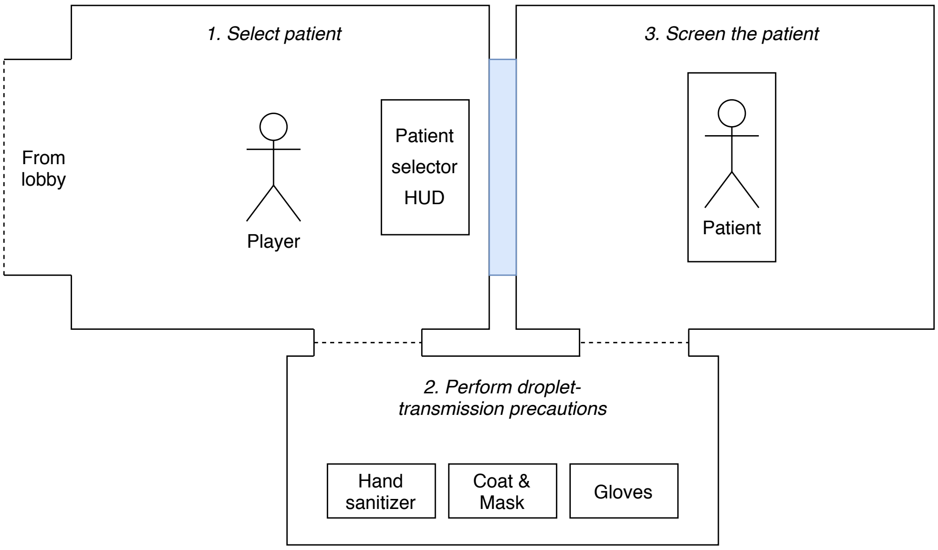


Figure 3.11: Top-down overview of rooms

The level consists of three primary rooms, as illustrated in Figure 3.11. Each room corresponds to a task in the gameplay flow diagram (Figure 3.2). The first room is where the player selects the patient that must be screened for MRSA. The second room is where the player must perform the necessary droplet-transmission precautions, and the third is where samples are collected. The three rooms are physically connected by doors to and from the anteroom as illustrated by the dashed lines. Also, there is a fourth room, the *lobby*, which leads to the other rooms. This design was inspired by our visit to the infection control ward at Haukeland Hospital. It was however heavily simplified and modified to fit better with the gameplay. During the development process we found that we had to adjust the dimensions of the rooms. While testing with one of our advisors at Helse Vest IKT he noted that he felt that the rooms were too big and that it led to excessive amounts of locomotion. We subsequently readjusted the scale of the rooms so that they are closer to the actual dimensions seen in a hospital.

We started the level design process by prototyping the level with ProBuilder [53]. ProBuilder is a package for Unity which adds tools for constructing geometry directly in the Unity editor. The initial design consisted of scaling and placing boxes of various dimensions for the floors, ceilings and walls. This kind of design was great because it allowed for rapid modification while also being relatively efficient from a rendering perspective due to the low polygon count. After reaching a suitable design, we recreated the level in the modelling software Blender, which allowed for greater flexibility and made it easier to optimize the geometry.

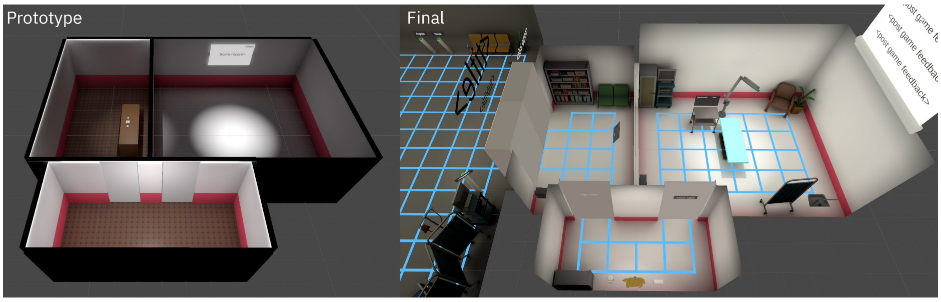


Figure 3.12: Level design: Prototype and final

Figure 3.12 shows the prototype and the final results. In the prototype, the walls, ceilings and floors consisted of boxes, which is evident from the thickness of the walls. Also, it did not have a lobby. In the final version of the level, the structure of each room, i.e. walls, ceilings, and floors, have been combined into a single mesh. A single mesh per room is more efficient than multiple boxes because it reduces the number of draw calls. At some point, however, if the meshes become very complex, it can be more efficient to split it up in order to perform culling. An additional benefit of a combined mesh is that light baking quality is improved because there are fewer places where leaks can occur.

Props

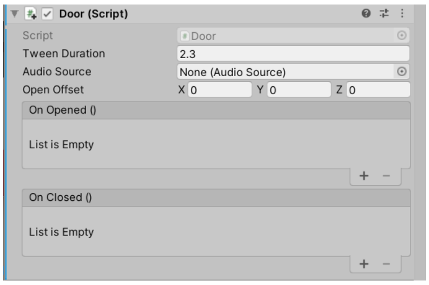
In addition to the structure of the rooms, the level also consists of various props related to healthcare. For instance, the lobby has a bench, a folding screen and a drip stand. We added these props in response to feedback from user testing, and subsequent feedback was positive. By being relatable, the props can add to the player's level of immersion. We acquired the props from the Unity Asset Store, which is described in more detail in Section 3.7.

Lighting strategy

Choosing an appropriate lighting strategy is especially important when working with VR. Our final strategy is a combination of lightmapping, light probes and real-time spotlights, with only a few of the lights in the scene being real-time. The strategy yields excellent performance, which is crucial on mobile hardware, while still providing good looking results. We found our strategy through trial and error and from Unity's VR Best Practices tutorial [44]. This strategy works well because most of the objects in the scene are static. We found that real-time shadows from the spotlights to positively affect the visuals when dealing with dynamic objects such as tools.

Doors

Doors separate the three rooms. To simplify animation work, we opted for sliding doors. The functionality of a door is implemented as a component. Figure 3.13 shows the component in the Unity editor. We achieve a sliding action with a linear interpolation between the original position of the object and an offset over a certain amount of time. An audio clip plays when the door opens and closes. Finally, the component also exposes two events, *OnOpened* and *OnClosed*, which can be used to call a function when a door starts opening or has been fully closed.



Door (Script)

Script: Door

Tween Duration: 2.3

Audio Source: None (Audio Source)

Open Offset: X 0 Y 0 Z 0

On Opened ()

List is Empty

On Closed ()

List is Empty

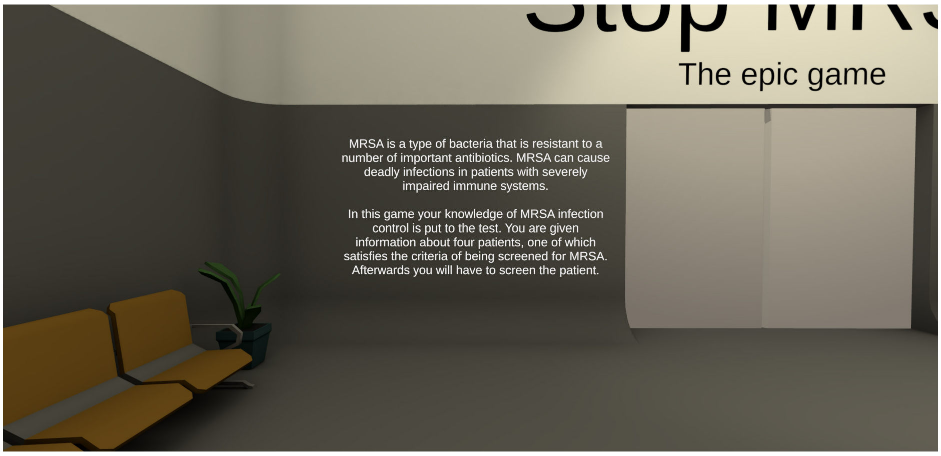
Figure 3.13: Level design: Door component

Static information

It is necessary to provide information to the player at the start and upon completing of the game. Figure 3.14 shows the introductory text in the lobby. We found that placing the text on the wall gave adequate readability, while also avoiding the pitfalls of having visual elements tied directly to the camera, which according to Unity's VR Best Practices [44] tutorial can negatively affect the player's experience.

3.5.2 Cross-platform support

As mentioned in Section 2.6, several frameworks that provide common VR functionality exist for the Unity game engine. Early in the development process, we chose to use the SteamVR framework [48]. The reason for this was that documentation and examples were abundant, which meant that the required effort to get a prototype up and running was fairly low. Also, the framework was and is very feature-complete. However, later in the development process focus shifted to making the game accessible to more platforms.



MRSA is a type of bacteria that is resistant to a number of important antibiotics. MRSA can cause deadly infections in patients with severely impaired immune systems.

In this game your knowledge of MRSA infection control is put to the test. You are given information about four patients, one of which satisfies the criteria of being screened for MRSA. Afterwards you will have to screen the patient.

Figure 3.14: Level design: Introductory text on wall in lobby

The shift in focus was due to the goal of deploying the game at several VR-labs around the country. Which means it must be able to run on all widely available HMDs. Because the SteamVR framework is limited to computer-tethered systems (with support for OpenVR), we made the change to a different framework. We found Unity's XR interaction toolkit [50] to be a suitable replacement. Unity's XR Interaction Toolkit is a component-based interaction system which abstracts away the platform-specific code. It also supports mobile HMDs such as the Oculus Quest. Changing from the SteamVR framework to the XR interaction toolkit was fairly straightforward, due their similarities.

3.5.3 Input and interactors

The XR Interaction Toolkit consists of three primary concepts, *Interactor*-components, *Interactable*-components, and an *Interaction Manager* [50]. An *Interactor*-component is typically attached to and controlled by a VR controller. An object with an *Interactable*-component is something in the world with which the player can interact using an *Interactor*. The *Interaction Manager* keeps track and manages *Interactor*- and *Interactable*-components and allows them to interact in the world. The toolkit also provides a *Controller*-component which handles input data and translates it to interaction states.

The Unity engine provides cross-platform mappings from hardware buttons to virtual inputs on controllers of various XR systems [54]. When configuring a *Controller*-component, the virtual input for selecting and grabbing an *Interactable* can be defined. Through testing, it became apparent that a significant source of confusion came from having multiple ways of interacting. Therefore, we chose a minimalistic approach. In the end, only two physical buttons are ever necessary to interact with the world, while most controllers have more than double that available. One of these buttons is the *teleport*-button, which

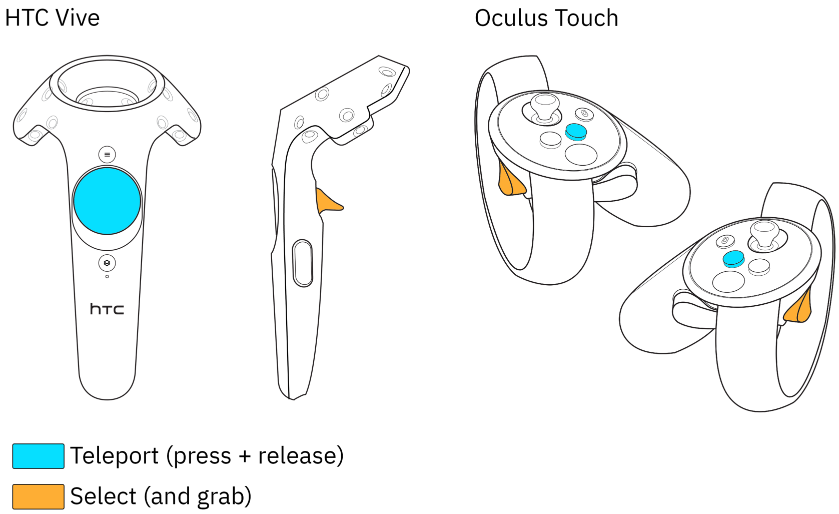


Figure 3.15: How buttons on the HTC Vive and Oculus Touch controllers have been mapped with Unity's XR Interaction Toolkit

is used to move around in the scene. The other is the previously mentioned *select*-button, which is used to interact with things in the scene. Figure 3.15 shows how these interactions are mapped to the physical buttons on the HTC Vive and Oculus Touch controllers.

Locomotion

As shown in Figure 3.17, while pressing the *teleport*-button, a beam is cast from the controller into the environment. On release, the player teleports to the location where the beam hits a surface if it is accessible. Not adequately restricting where the player can teleport can easily result in breakage of immersion. One example we saw early in testing, was that a player would teleport very close to a wall and then clip through that wall, either revealing the adjacent room or the empty void outside the scene. The player rarely did so on purpose, which led to much confusion. In order to restrict the teleport area, we created a mesh that covers just the parts of the floor throughout the game to where the player should be able to teleport (see Figure 3.16). When the beam hits somewhere on the surface of this mesh, the location is considered accessible.

Indicating to the player what areas are accessible is an integral part of making the interaction intuitive. There are multiple ways of doing so. Some notable examples are found in Valve's games The Lab [55] and Half-Life: Alyx [56]. Additionally, the SteamVR Unity framework comes with built-in support for this. Unity's XR Interaction Toolkit, however, only partially supports this with the ability to assign a mesh that will be placed at the surface hit by the beam when it is an accessible area. As seen in Figure 3.17, a cylindrical mesh

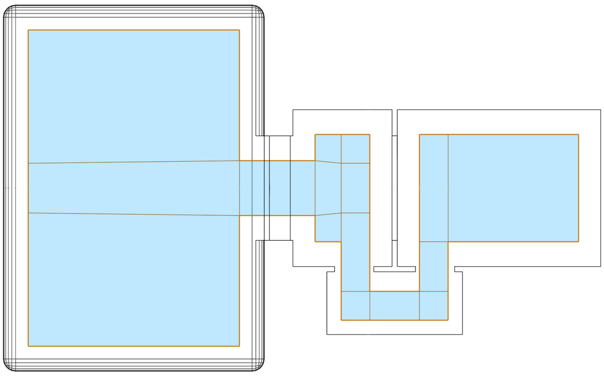


Figure 3.16: Top-down view of the teleport area, which is represented by a mesh (highlighted in blue) tailored to the scene

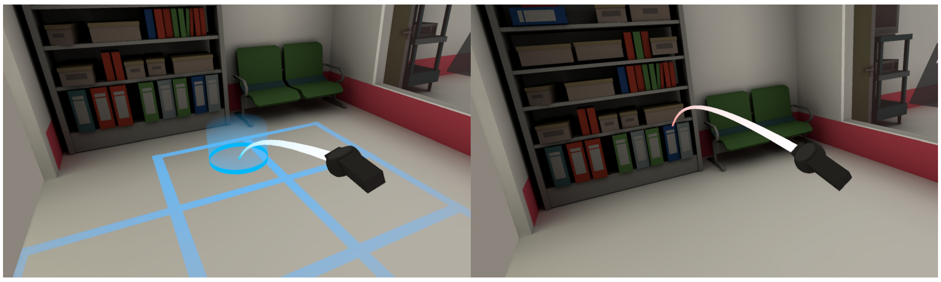


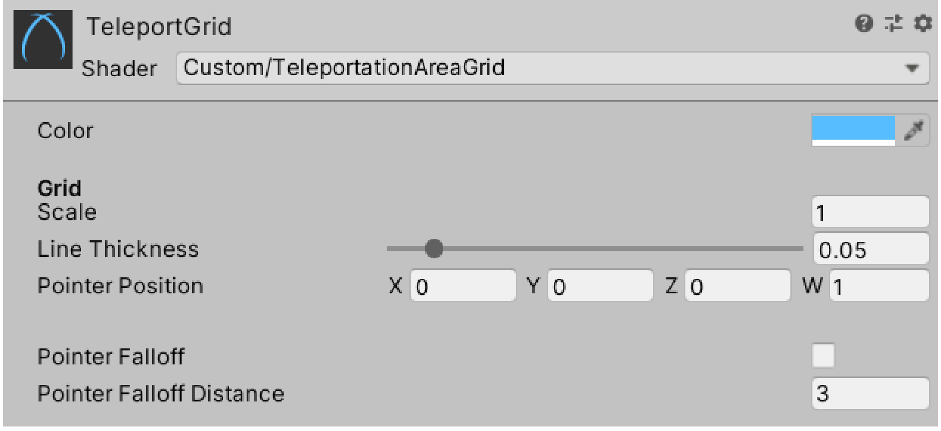
Figure 3.17: Teleport beam while pointing at a valid location and on an invalid location

with a gradient is placed on the location the beam hits. The framework also visualizes the beam itself, which can change colour depending on where it hits. In order to also highlight accessible areas, we implemented a custom shader, as shown in Figure 3.17. Shading the accessible area is an effective means of communicating to the player where they can move and is a technique employed in most games with teleportation (one example being The Lab [55]).

The shader (Listing 1) is written in HLSL and is applied to the teleport area mesh shown in Figure 3.16. The world-space position of the vertices are interpolated via the rasterization pipeline. In the fragment shader, the alpha value is calculated based on the X- and Y-components of the interpolated world space position, such that the final result is a grid (as seen in Figure 3.17). As shown in Figure 3.18, the shader takes five parameters that can be used to adjust how the grid looks. The *Pointer Falloff* parameter enables or disables a radial falloff based on the *Pointer Position* parameter. This parameter is updated per frame based on where the beam hits the mesh. With the falloff enabled, the alpha value is inversely proportional to the distance from that position in the XY-plane. The *Pointer Falloff Distance* parameter is the distance where the alpha value reaches zero, i.e. how far from the position the grid remains visible.

```
float2 distanceToNearestGrid(float2 v) {
    float2 f = frac(v); // Get fractional part
    return float2((f.x < 0.5) ? f.x : (1.0f - f.x), (f.y < 0.5) ? f.y : (1.0f - f.y));
}

fixed4 frag(v2f input) : SV_Target
{
    fixed4 col = _Color;
    float2 dist = distanceToNearestGrid(abs(input.worldSpacePosition.xz) * _Scale);
    float2 delta = fwidth(dist); // anti-aliasing
    float2 factor = 1.0f - smoothstep(_Thickness - delta, _Thickness, dist);

    float distanceFalloff = 1.0f - clamp(distance(_PointerPosition.xz, input.worldSpacePosition.xz) / _PointerFalloffDistance, 0.0, 1.0);

    col.a *= max(factor.x, factor.y); // Grid
    col.a *= (_PointerFalloff == 1.0f) ? distanceFalloff : 1.0f;

    return col;
}
```

Listing 1: Teleport Area Grid: Shader code



TeleportGrid

Shader: Custom/TeleportationAreaGrid

Color

Grid

Scale: 1

Line Thickness: 0.05

Pointer Position: X 0 Y 0 Z 0 W 1

Pointer Falloff: □

Pointer Falloff Distance: 3

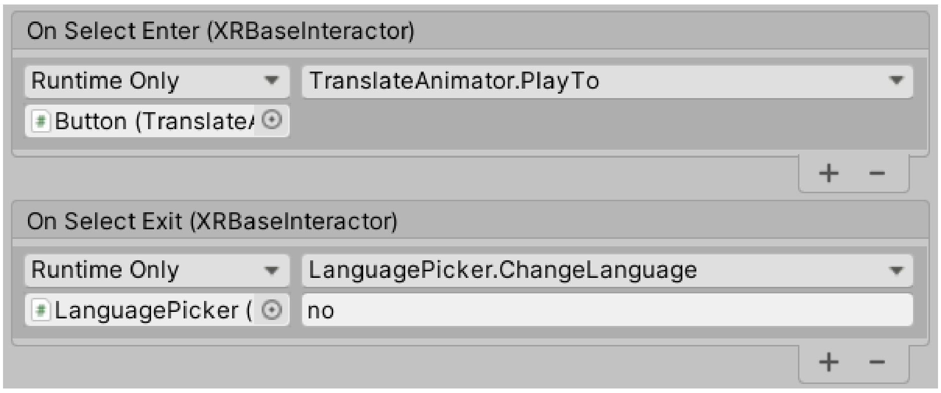
Figure 3.18: Teleport Area Grid: Unity shader parameters

Interactors

Selecting and grabbing are the primary means of interacting in the game. Pressing a button or performing actions in the anteroom are examples of selecting. Picking up an *Interactable*, such as a tool when screening the patient, is an example of grabbing. These interactions are all performed with an *Interactor*. More specifically each of the player's controllers has a *XRDirectInteractor*-component. This component makes it possible to interact with interactable objects in the scene directly. A *XRDirectInteractor* requires that a collision volume, i.e. *Collider*, is specified. We chose to use a sphere which is attached to the player's virtual controller.

3.5.4 Interactables

The player can interact with certain objects in the scene, called "interactables". To make an object interactable, we need to add an *Interactable*-component. The XR Interaction Toolkit provides several types of interactable components, such as the *XRSimpleInteractable* and *XRGrabInteractable*. As shown in Figure 3.19, an interactable exposes an *OnSelectEnter*- and *OnSelectExit*-event. It also exposes other events such as the *OnHoverEnter* and *OnHoverExit* events. We can attach event listeners, i.e. methods, that are invoked when an *Interactor* (such as the player's controller) hovers and selects the *Interactable*. In this example (Figure 3.19), when the *select*-button is pressed the *Interactable* is animated and upon release the language of the game is set to Norwegian.



On Select Enter (XRBaseInteractor)

Runtime Only ▾ | TranslateAnimator.PlayTo ▾

Button (Translate/ ⊙

On Select Exit (XRBaseInteractor)

Runtime Only ▾ | LanguagePicker.ChangeLanguage ▾

LanguagePicker (⊙ | no

Figure 3.19: Interactable select events

Buttons

Throughout the game, the player must use various types of buttons. All buttons use the *XRSimpleInteractable*-component to handle interactions with the controllers. The first button in Figure 3.20 is meant to resemble a physical button.

The second type of buttons is the UI buttons. These are placed in the scene on boards which consist of a black and slightly see-through background with white text. Upon hovering over a UI button, i.e. intersecting it with the controller, it lights up just as UI buttons typically do in response to mouse hovering in regular 2D user interfaces. This effect gives off a responsive quality and communicates that the button is pressable. Some of the boards appear when

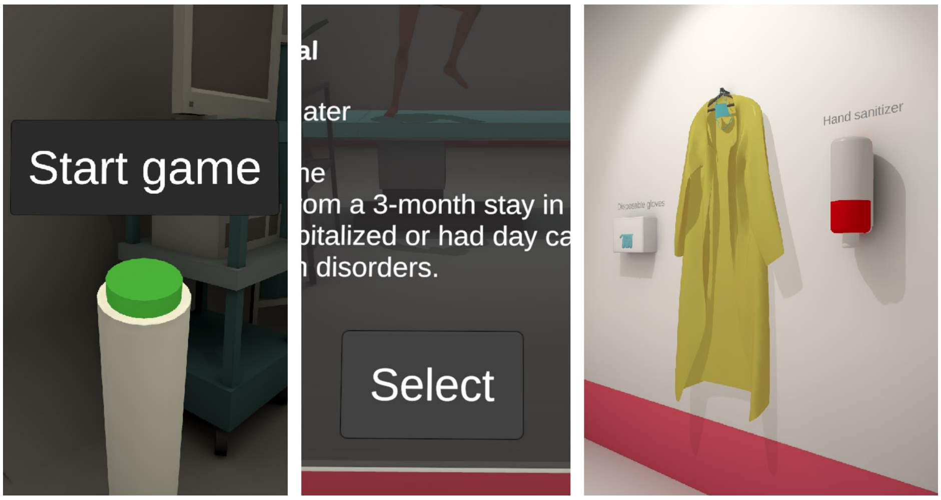


Figure 3.20: Various types of buttons in the game

certain game events are triggered. They often have a short and informative text with a single button to close it. Such boards orient themselves to face the player, making the text readable from every angle.

Other boards are fixed in place and do not rotate to face the player. These can be purely informational, such as the one besides the patient, which displays successfully collected samples. Another example is the one that is used for selecting a patient. It has three buttons, two to browse the set of patients and one to select a patient for screening.

Finally, there are the items on the wall in the anteroom. Although these do not look like buttons, their functionality is similar to the other buttons.

Tools

Tools, unlike buttons, need to be grabbed and held by the player, which is achieved with the *XRGrabInteractable*-component. As the players encounter the tools, they will float suggestively in the air. The player can reach out and grab them by holding in the *select*-button on the controller. As a tool is picked up, the hand controller becomes hidden such that only the tool is in focus, and upon dropping the tool, the hand controller reappears. Additionally, the tool is removed when intersected with a target on the patient, as can be seen in Figure 3.21.

3.5.5 Feedback

When a player is rewarded or loses points, it is indicated by spawning a textual representation which gradually fades away while floating upwards. A reward is indicated by a plus sign followed by the number of points in with green text colour. The punishment has a red colour and uses a minus sign instead.

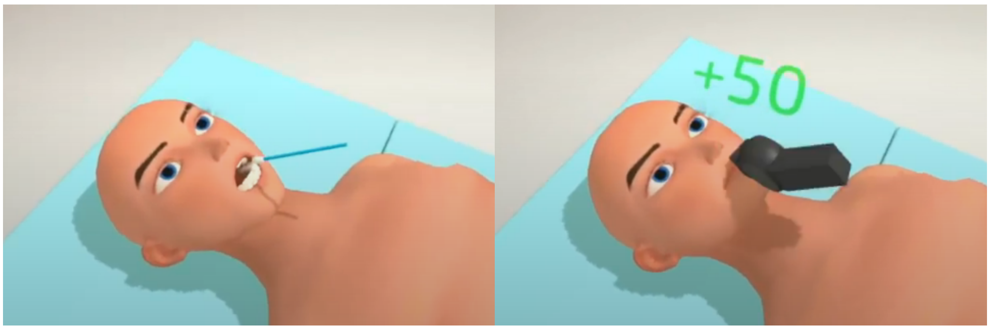
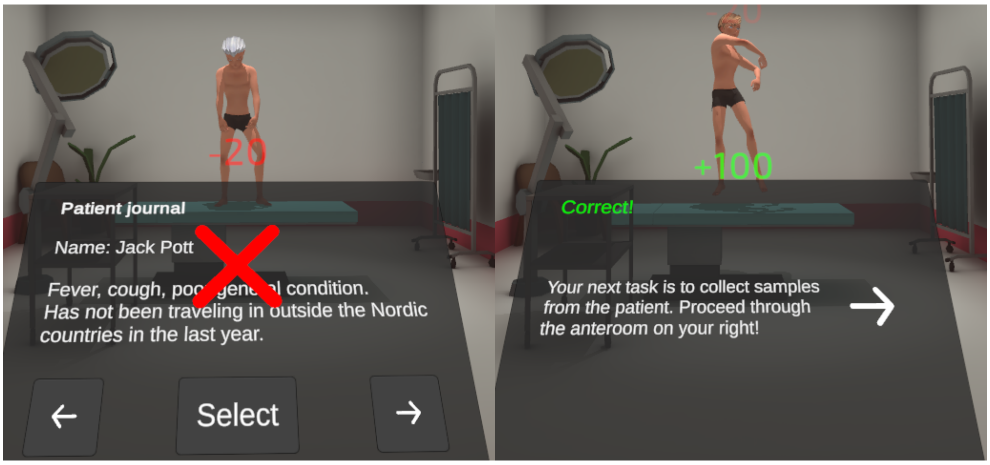


Figure 3.21: Holding a swab in front of the mouth of a patient prompts the patient to open their mouth. After leading the swab down the throat, you see the result of the screening attempt as the patient closes the mouth.



Patient journal

Name: Jack Pott

Fever, cough, poor general condition.
Has not been traveling in outside the Nordic countries in the last year.

Your next task is to collect samples from the patient. Proceed through the anteroom on your right!

Figure 3.22: Side by side comparison of rewarding and punishing points when selecting patients for screening. In both cases the text is floating upwards and slowly fading out.

The text slowly floats up before fading out after a few seconds. The text is spawned by a specific object which is placed close to where the player has acted. For instance, every screening target on the patient has a spawner in its centre. Thus, when the player uses a tool on the target, there will be instant feedback where they are already looking. When the target has already been screened, there is no change in the score, which is indicated with the text ”already screened” and a dark text colour.

Another way to give feedback to the player’s decisions is through audio. The player will hear a positive-sounding ”pling” sound every time the player is rewarded with points. Similarly, a negative ”error” sound is played when they are punished. Using audio like this is common in traditional games, and generally, users can easily recognise some sounds as ”good” and others as ”bad”. The combination of green text, a plus sign, and a recognisable ”positive sound” should leave little to the imagination as to whether you made the right decision or not. This can also strengthen the emotional value of the feedback and can leave a satisfactory impression when hearing positive sounds. In this way, it can feed into the gamification idea of motivation. Many gamification platforms also use audio feedback like this, such as Duolingo [13] and Khan Academy [15].

3.5.6 Patients

In order to be able to collect samples from the selected patient, it must be represented in the scene. The visual representation of a patient in the game consists of a model, i.e. the rendered geometry. We use a custom component, the *Patient*-component, to manage the patient. This component is responsible for initializing the physical representation of the patient based on its information as well as performing certain animations. We chose to use one model for female patients and one for male patients. To give patients a unique look, both models can have one of several hairstyles. Upon initialization, a hairstyle is randomly selected. The hairstyle is an effective way to make each patient unique, even if their bodies are identical. Also, wounds can be at several possible locations on the body. A random one is enabled if the patient has a wound. With the Unity editor, new hairstyles and wounds can easily be added in the *Patient*-component in their respective lists.

Physics

Depending on the application, physical interactions can be an especially important aspect of VR. By physical interaction, we are in this case, referring to a virtual interaction that makes the player feel or at least get the impression that an actual physical interaction has occurred. Tools for collecting samples are represented in the physics engine and respond, for instance, to gravity and collisions with other objects in the scene. However, in early versions of the game, the patient was not physically represented, except for the sample-targets. Therefore, tools would pass through the geometry. In tests, players noted that the lack of a response from the tool touching the patient felt awkward and reduced their feeling of immersion.

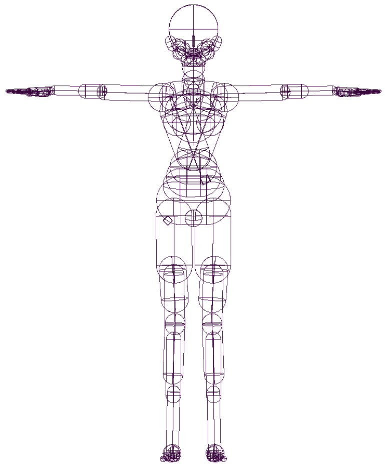


Figure 3.23: Colliders attached to the patient form a compound collider

To make an object in Unity's physics system interact with other physical objects, we add a *Collider*-component. Various types of colliders exist, such as spheres, boxes and capsules. A collider can also consist of geometry if a more accurate representation is required, but this kind of collider is computationally expensive. Instead, we used the primitive colliders, i.e. spheres, capsules and boxes, which are more efficient. As seen in Figure 3.23, by carefully placing multiple such primitive colliders, we can create a compound collider for the patient. How many colliders that are needed depends on the application, and in our case, we needed a moderately accurate representation because the player performs interactions up close. To make the colliders animate with the geometry, we assigned them to the skeletal hierarchy (which was described in Section 2.4.3). That way, the colliders are transformed properly throughout animations.

The resulting compound collider worked well, and although it deviated from the mesh at certain places, it was still accurate enough to give the illusion of proper physical interaction. We did, however, notice, through testing, that players would inadvertently touch colliders in the facial area with the swab, leading to contamination. We adjusted the problematic colliders out of the way, to the detriment of accuracy and betterment of gameplay.

As a side note, we found it necessary to set the physics update rate to the refresh rate of the display (as shown in Listing 2). Without doing so, the tools would visibly stutter while being moved. The physics update rate is the number of times physics calculations are performed per second by the engine.

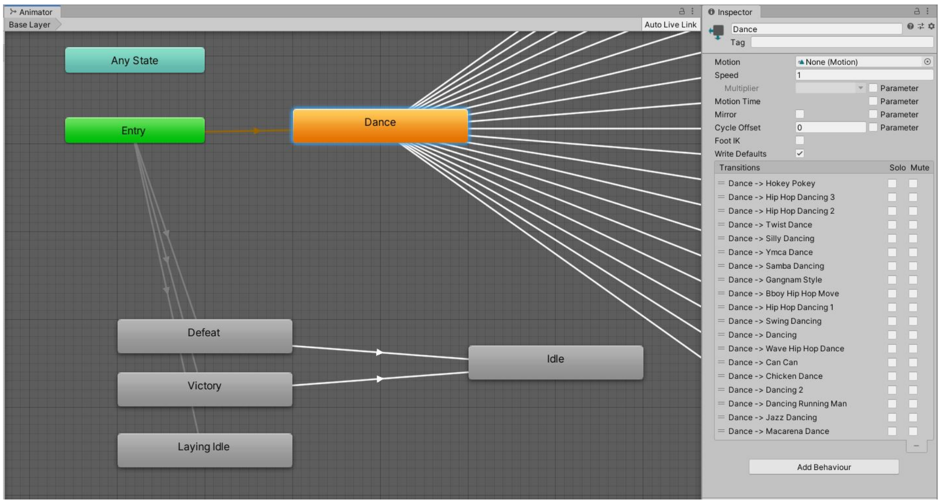
```
if (XRDevice.isPresent)
{
    Time.fixedDeltaTime = 1.0f / XRDevice.refreshRate;
}
```

Listing 2: Setting physics update rate based on device refresh rate

Skeletal animations

In early versions of the game, the patients lacked any form of animation, which was often noted in feedback from user testing. The participants said they felt that there was a lack of response to their actions and that the inanimate patients made for a dull experience. To correct this, we added various forms of animations.

The first animation we added was an idle laying animation for the patient. This animation gives the impression of the patient being alive by adding subtle motions such as breathing. The animation is of the skeletal type (as mentioned in Section 2.4.3), and was added to the patient through Unity's animation system with an *Animation Controller*. An animation controller is essentially a state machine which allows for controlling various transitions between multiple animation clips. An animation controller is represented as an *Animator*, and Figure 3.24 shows it in the Unity editor. The laying idle animation loops seamlessly.



Dance
Tag

Motion	None (Motion)	
Speed	1	
Multiplier		□ Parameter
Motion Time		□ Parameter
Mirror	□	□ Parameter
Cycle Offset	0	□ Parameter
Foot IK	□	
Write Defaults	☑	

Transitions	Solo	Mute
Dance -> Hokey Pokey	□	□
Dance -> Hip Hop Dancing 3	□	□
Dance -> Hip Hop Dancing 2	□	□
Dance -> Twist Dance	□	□
Dance -> Silly Dancing	□	□
Dance -> Ymca Dance	□	□
Dance -> Samba Dancing	□	□
Dance -> Gangnam Style	□	□
Dance -> Bboy Hip Hop Move	□	□
Dance -> Hip Hop Dancing 1	□	□
Dance -> Swing Dancing	□	□
Dance -> Dancing	□	□
Dance -> Wave Hip Hop Dance	□	□
Dance -> Can Can	□	□
Dance -> Chicken Dance	□	□
Dance -> Dancing 2	□	□
Dance -> Dancing Running Man	□	□
Dance -> Jazz Dancing	□	□
Dance -> Macarena Dance	□	□

Figure 3.24: Unity editor: Editing animator

When the player is browsing the patients as part of their first task, they can see the patients through a window. Initially, all patients were lying idle on the hospital bed, which meant that the unique visual characteristics of the patients was their sex, hairstyle and possibly wounds. To add more to their uniqueness, we added dance animations. Each patient is assigned a dance when the game is initialized. When viewing a patient, their dance is played. All dances loop seamlessly. When the player selects a patient if the choice is correct, the patient transitions to a positive reaction animation, and if the choice is incorrect, the patient transitions to a negative reaction animation, this additional animation is meant to supplement the other feedback that the player gets. Through further testing with our advisors, we found that these animations worked well. One of our advisors noted that although the dances are not very representative of a real scenario, they help make the game more enjoyable.

Procedural animations

In the idle pose of the patient mesh, the mouth is closed. In early versions of the game, this meant that the player had to push the swab through the mesh so that it would touch the trigger sphere. Users noted that the interaction felt unresponsive and that it made the experience less immersive. A form of procedural animation using *blend shapes* (as mentioned in Section 2.4.3) greatly improved the interaction. The animation consists of opening the patient's mouth when a swab is in close proximity.

Listing 3 shows the event listeners that linearly interpolates the weight of the blend shape for the mouth. As we can see in the *onTriggerEnter*-listener, the weight of the blend shape is interpolated to 90 over 350 milliseconds. In the *onTriggerExit*-listener the weight is interpolated back to zero in 250 millisec-

```
mouthProximityTrigger.onTriggerEnter.AddListener((Collider other) =>
{
    // ...
    openTween = DOTween.To(() =>
    ↪   this.body.GetBlendShapeWeight(mouthBlendShapeIndex), (x) =>
    ↪   this.body.SetBlendShapeWeight(mouthBlendShapeIndex, x), 90.0f,
    ↪   0.35f);
    // ...
});

mouthProximityTrigger.onTriggerExit.AddListener((Collider other) =>
{
    // ...
    closeTween = DOTween.To(() =>
    ↪   this.body.GetBlendShapeWeight(mouthBlendShapeIndex), (x) =>
    ↪   this.body.SetBlendShapeWeight(mouthBlendShapeIndex, x), 0.0f,
    ↪   0.25f);
    // ...
});
```

Listing 3: Procedural animation: Opening and closing mouth with blend shapes

onds. The interpolation durations were adjusted until they felt adequately smooth while still being responsive, but this is something that can be improved further. Alternative ways of deciding when and how fast the blend shapes are adjusted can be explored further. Additionally, more blend shapes can be utilized to give the patient more expressive power.

Another procedural animation we implemented is blinking with eye rotation. We use a coroutine to perform the animation, as shown in Listing 4 by the signature of the *Blinking*-function. The coroutine starts with a delay by waiting the specified number of seconds. After the delay, the eyelids are closed with an interpolation, followed by a change in eye rotation (omitted to save space), followed by the eyelids opening again. Next, it calculates the time until the next blink based on the *averageBlinksPerMinute*- and *randomBlinkFactor*-variable, followed by a recursive call of the coroutine.

```
private IEnumerator Blinking(float delay = 0.0f)
{
    yield return new WaitForSeconds(delay);

    var close_tween = DOTween.To(() =>
    ↪   this.body.GetBlendShapeWeight(blinkBlendShapeIndex), (x) =>
    ↪   this.body.SetBlendShapeWeight(blinkBlendShapeIndex, x), 90.0f,
    ↪   0.05f);
    yield return close_tween.WaitForCompletion();

    // Change eye rotation slightly

    var open_tween = DOTween.To(() =>
    ↪   this.body.GetBlendShapeWeight(blinkBlendShapeIndex), (x) =>
    ↪   this.body.SetBlendShapeWeight(blinkBlendShapeIndex, x), 0.0f,
    ↪   0.1f);
    yield return open_tween.WaitForCompletion();

    float fixed_delay = (60.0f / averageBlinksPerMinute);
    float offset = fixed_delay * randomBlinkFactor;
    float random_delay = Random.Range(fixed_delay - offset,
    ↪   fixed_delay + offset);

    StartCoroutine(Blinking(random_delay));
}
```

Listing 4: Procedural animation: Blinking with eye rotation

3.6 Balancing gamification and accuracy

The development of a gamified application for VR is, in some sense, a balancing act between accuracy and entertainment. Attempting to recreate every interaction method precisely as it is in reality, might make it difficult for a player to complete the challenges with the restrictions of using VR hand-held controllers. The gameplay might end up being frustrating, as some of the interaction methods would be too tedious to do with the inherently limited precision of the controllers. On the other hand, for every interaction method that is simplified, we stray further from the actual procedure. Additionally, a simplified version could end up rewarding the players for executing the procedure sloppily or even downright incorrectly. The developers are therefore forced to make difficult decisions that can make the game more entertaining or accurate, sometimes to the detriment of the other.

One example is putting on protective equipment in the anteroom. When having close contact with a patient that has been isolated on suspicion of being MRSA positive, nurses are expected to put on a coat, gloves and a face mask [57]. The instructions for putting on the coat, for instance, are well described and goes into details that require finger precision [58]. Mirroring this directly in a VR application is challenging due to the restrictions imposed by hand-held controllers. The same goes for putting on gloves and masks. In the end, for the gameplay's sake, the choice was made to simplify this segment. The act of physically equipping the coat and mask is instead represented by a click followed by the coat and mask being removed from the hanger. A similar simplification is applied to the gloves.

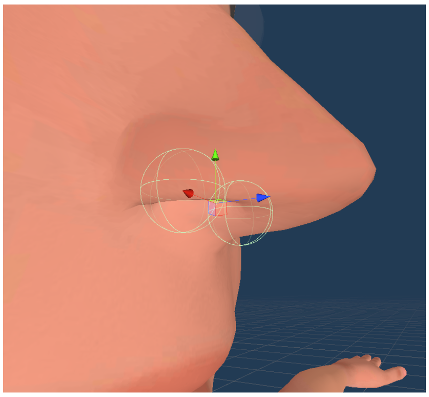


Figure 3.25: Screenshot of trigger spheres in the patients nostrils

Another example is collecting samples from the patient's nose. Perfectly modelling interactions with the patient's nostrils would require accuracy down to a very low scale. Seeing as the scale of the nostril is at around 0.5 - 1cm and with the occasional jittering of the VR headsets tracking occurring, such an implementation would be impractical and possibly tedious for the player. Instead, we opted for a simplified version where interaction with a nostril is summarised as a sphere intersection test. Figure 3.25 shows the positioning of the two trigger spheres that represent the simplified interaction method. In addition to considering the physical interactions, we also considered whether sampling of the nostrils could be combined into a single interaction. The guideline states that when collecting samples from the nose, a single swab needs to be used for both nostrils. However, this could be a little tedious for the player, as we could imagine players putting the swab into one nostril and being confused by the lack of a positive response. On the other hand, if they do receive a visual reward from sampling one nostril, they might think they are done with that swab. We ultimately ended up with the less simplified version, with the player having to sample both nostrils. Haptic feedback is used to indicate that an interaction has occurred, but that the task is not yet completed. It should be noted that the more simplified version was used when undergoing the final tests and that the change was a consequence of the responses. More on this in Chapter 4.

After the player has collected a sample from the patients' body, the question is how to handle that sample. In the case of collecting a sample with a swab, nurses typically put it in a screw cap tube and label it with pertinent information such as the patients name and the location from where the sample was collected, after which the sample is sent to a lab for analysis. This is another example of where the precision requirement is fairly high, both with regards to current tracking hardware and to the physical simulation in the game. Therefore an attempt at perfectly recreating such an interaction would possibly result in a tedious experience for the player. There are multiple ways of implementing this such as to balance accuracy and gamification,

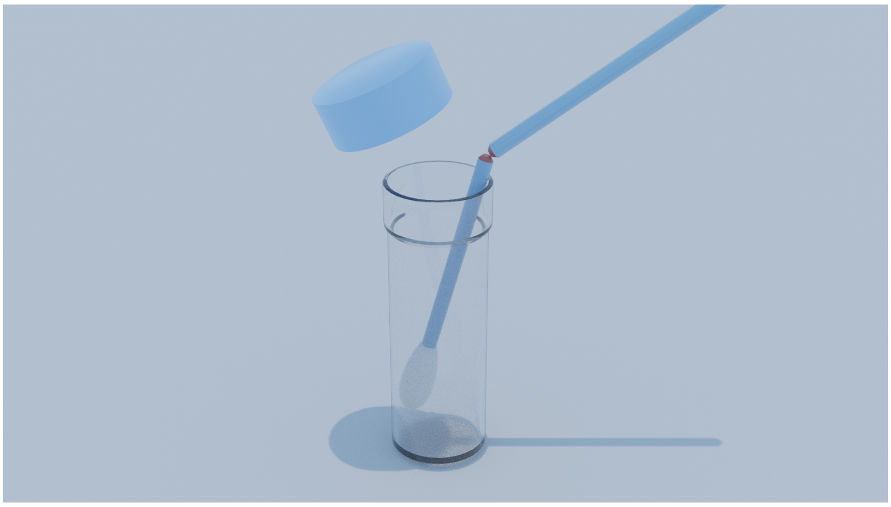


Figure 3.26: Example of collected sample being put in a tube.

and to better account for the restrictions imposed by the hardware. One way would be to include the interaction, but to simplify it to a "snap-in-place" action. Instead of fully simulating the physical interaction between the swab and the tube, simply touching the tube, at the correct spot, with the swab could initiate an automated placement of the sample in the tube. That way the interaction would be restricted to a simple intersection test between the swab and a segment of the tube, and thus the precision requirement lowered. Another alternative would be to not include this interaction at all and instead consider a sample collected once the swab touches the appropriate target on the patients body. Due to resource constraints and prioritization the latter was chosen, and the first option was left for further work. After putting the sample in the tube the nurse needs to, as mentioned, label it with the correct information. A concern was raised by a nurse while we were visiting Haukeland Hospital that incorrect labelling of samples occasionally occurred and was problematic. Therefore as an addition after having collected all samples, the player is informed through a prompt about what information the samples have to be labelled with. This could of course be extended further by requiring the player to perform the labelling. However, as we were later informed by our domain expert consultant, routines for labelling samples can vary somewhat between institutions, and because the game targets a variety of institutions a specific implementation would not be viable.

There are a number of elements in the game that are only there for entertainment purposes. As you choose the patient to screen, the patient can be seen dancing on the hospital bed behind a glass window. The patients have humorous names, often puns. When you are done with the entire screening procedure, the patient will dance again. This time there will also be disco music and colourful flashing lights in the room creating a party atmosphere. Such "silly" elements serve no purpose in mirroring the accuracy of the CPG, but is a part of the gamification of the application. The players are supposed to feel like they are doing something more fun than going through a real life simulation, to keep their attention. There is a fine line, however, as too much silliness can actually be detrimental to the players' attention. We must make sure the silly things do not take all of the focus away from the tasks important to practice the CPG. Selecting the correct patient for screening is an important part of the CPG, and during the ”disco party” there is important text on the walls, like an ”answer sheet” with whom to screen and how to do it. Ideally, the entertaining elements should not distract from any of that, but rather help motivate the players to pay extra attention to the game.

3.7 Assets

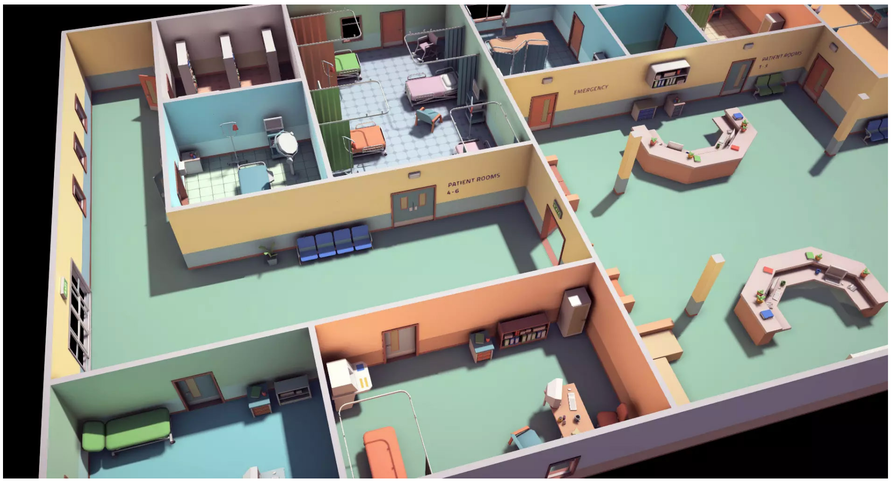


Figure 3.27: Low poly hospital assets

To create the game, we used various assets. We needed 3D models such as patients, hospital beds and various props and more. Certain assets were created from scratch, and others were collected from the internet. As a team of two software developers whose skillsets consist mainly of programming, the availability of assets on the internet proved to be useful. We obtained a collection of low poly hospital models 3.27 from the Unity Asset Store. Low poly refers to an art style where the geometry intentionally consists of a low number of polygons. As a side note, the low number of polygons means that the models are quicker to render, which fits well with one of the targeted VR headsets where the performance requirements are stringent (more on this in section 3.9). We placed the models in the different rooms to make them look better and also make them more relatable to the player.

When the necessary 3D models could not be found online, we instead modelled them in Blender [5]. We created models such as the coat, swab, syringe, patient bed, soap dispenser and urinary catheter. The models were then imported into the game engine. Materials were created and applied in the game engine for compatibility reasons. In some cases, the required models were so simple that we instead combined primitives such as cylinders and cubes directly in Unity. One example of this is the projection screen at the end of the game.

The patients were made with a specialized application called Character Creator 3 [59]. With this application, we made cartoony low poly patients that fit somewhat with the style of the rest of the game. Exported models are already rigged and ready to be animated in Unity. We acquired the character animations from Mixamo [60] and applied them in Unity. Patients have unique dance animations which they perform when the player browses them and at the end of the game. A patient also performs a celebratory animation when correctly selected and a sulking animation when incorrectly selected. Also, they have an idle pose when lying down. The idle pose only consists subtle of breathing, but it gives a much more lifelike feel to it than if the patients had no movement at all.

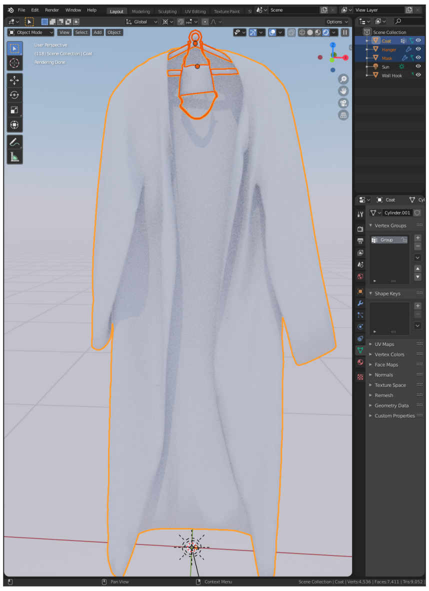


Figure 3.28: Modelling a coat in Blender 2.8 [5]

We acquired audio clips from Freesound [61], a website that lets people upload audio and tag it with any Creative Commons license. For our purpose, we avoided the audio clips that were not allowed to be used commercially, and we only used two clips that had to be credited. The artists of the songs were credited in the "lobby", the first room the players see as they start the game. Two of the audio clips are soundtracks. The first one is some relaxing music which is playing in the background while the player is selecting the patient to screen, and the other is a piece of celebratory disco music which is played upon completing the game.

3.8 Internationalisation

All text in the game is loaded from human-readable files. The files are named with two-letter language codes, like "en" for English or "no" for Norwegian. By default, the system language on the device the application runs on is used to determine which file to load. Additionally, the user can select the language in the game, and the language choice persists. This design is called *internationalisation* and makes it easy to add support for other languages, i.e. to perform *localisation*. Naturally, the patient journal information must also be localised. A similar scheme is applied by dynamically loading the information from the JSON file that ends with the correct language code.

At the time of writing, the game has support for two languages, Norwegian and English. The user can choose their preferred language in a *language picker* located in the first room of the game. As seen in Figure 3.29, the language picker is a collection of buttons, one for each language. Pressing any of these buttons reloads the scene with the chosen language.

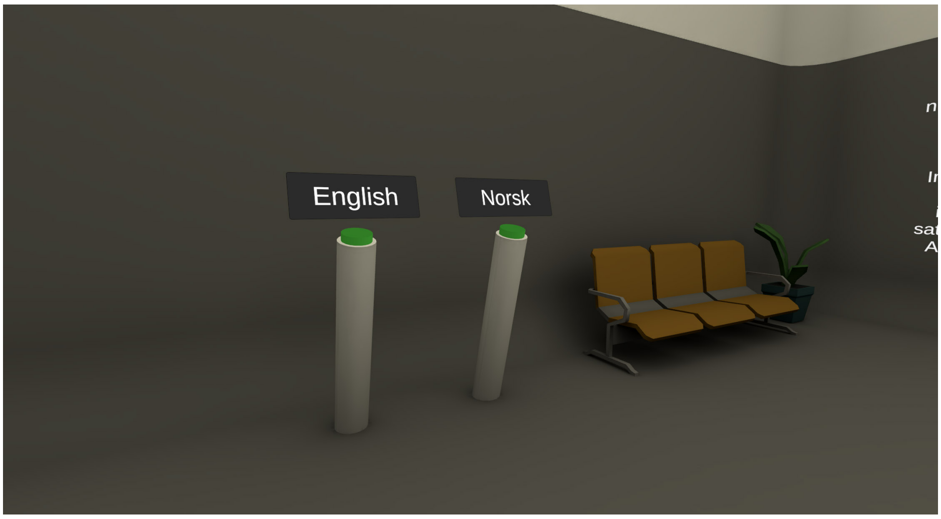


Figure 3.29: The user can choose between English and Norwegian.

If the game needs to be localised to another language, medical experts proficient in that language could help translate the text. When the new files are added, a simple addition of a new button to the language picker will let the user pick that language too. Adding another language picker in the Unity editor will not require changes to the code. These simple buttons work well with few languages, as is our case. However, if the game needed to support a lot more languages, a different language picker design could be better suited.

3.9 Performance and optimization

As mentioned in Section 3.5.2 the game needs to support various hardware and software configurations. For the user experience to remain consistent across these configurations a certain amount of optimization is necessary. We chose to optimize for the least capable HMD, namely the standalone Oculus Quest. This is primarily because it is considered the preferred headset by Helse Vest IKT for this project, but also because it is orders of magnitude less powerful than the desktop powered headsets which would make it less practical to go the other way around.

The Oculus Developer Centre provides recommendations for their HMDs with regards to performance and optimization, and they present the following target metrics for the Oculus Quest [62]:

- 72 frames per second

- 50 - 100 draw calls per frame (for comparison the recommendation for the Oculus Rift is 500 - 1000)
- 50,000 - 100,000 triangles or vertices per frame

The display has a refresh rate of 72 hertz, meaning it is able to update the presented image 72 times per second. Therefore the game should optimally be able to produce enough frames to stay at that limit. In order to achieve this one should aim for 50 - 100 draw calls per frame and 50,000 - 100,000 triangles or vertices per frame. The primary way to achieve these targets is by keeping the number of instantiated assets in the scene low and with a low number of vertices where possible. A draw call is essentially one command from the CPU to the GPU saying that it should render something. In order render one frame the CPU must tell the GPU to render something multiple times. For example, the CPU must the the GPU to render the floor, a table, the patient, etc., until all required objects have been rendered. With a naïve approach the number of draw calls per frame is directly proportional to the number of meshes in the scene. However, we will now look at two common techniques we employed to reduce this number significantly.

Frustum culling

The camera in the scene has a viewing frustum, the volume of space that is visible to the viewer once projected onto the display, as shown in Figure 3.30. Depending on the direction the camera is pointed certain objects will be visible and certain objects will not. Additionally, some objects may be partly visible because they partly overlap with the frustum.

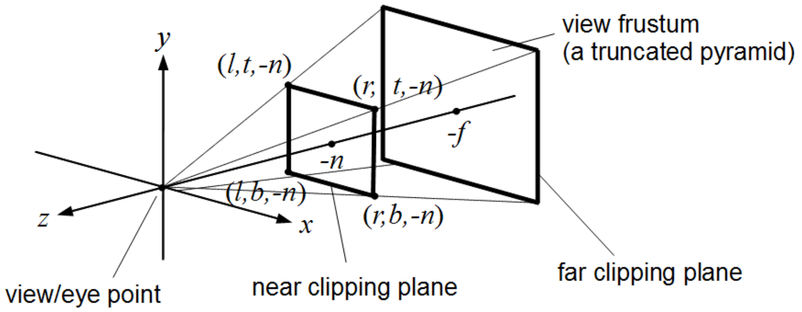


Figure 3.30: Perspective camera viewing frustum, Martin Kraus. CC BY-SA 3.0

Frustum culling is a technique where objects that are not in the frustum are omitted dynamically [4]. It is often done by building a hierarchical spatial structure which allows one to quickly decide which objects to draw. Unity and most other game engines already provides this kind of culling. In Figure 3.31 we can see the camera with its view frustum, and how objects that are not inside it are not being drawn. We only see a few remnants of the objects in the form of baked shadows in the environment. Notice that the environment i.e.

the rooms are still being rendered. This is because the floor, walls and ceiling have been combined into one mesh in order to further reduce the number of draw calls. In this case, having combined the room into one mesh, will not negatively affect the performance because the meshes consist of very few vertices. If on the other hand the meshes were very complex it would probably be better to split them in multiple pieces in order to reduce the number of vertices that would have to be processed in the vertex stage on the GPU.

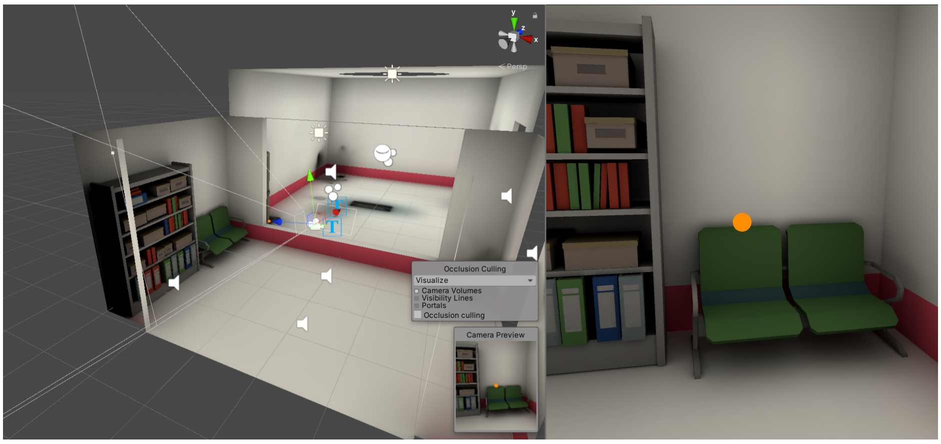


Figure 3.31: Example of frustum culling in the game: On the left we see the camera and its frustum and how other objects that do not intersect with it are not drawn

Portal/room occlusion culling

Since the scene is separated into several rooms, and the player is only located in one room at a time, objects that are in another room where the door has been closed will not be visible to the player. Therefore we can stop rendering those objects. This type of culling is a form of occlusion culling because the objects in the other rooms have be occluded by the door, and it is called portal occlusion culling. Our implementation is based on the fact that as the player progresses throughout the game, actions will trigger doors to open and as the players enter the next room the door closes behind them and that only the room that the player is in, is visible. In Figure 3.32 we can see the technique in effect. As the player moves from the lobby to the office we can see that the office room and the patient room have been enabled and that the lobby room has been disabled. The same applies for when the player moves from the office to the anteroom and from the anteroom to the patient room. Notice however that the office is disabled when the player has entered the patient room, this is because the window used to view the patient room from the office acts as a one way window, which makes the game perform better in the most critical phase of the game.

We implemented this technique by grouping objects in the scene under scene graph nodes representing each room. The nodes are enabled through the use of event-listeners attached to the doors. A door can fire two types of

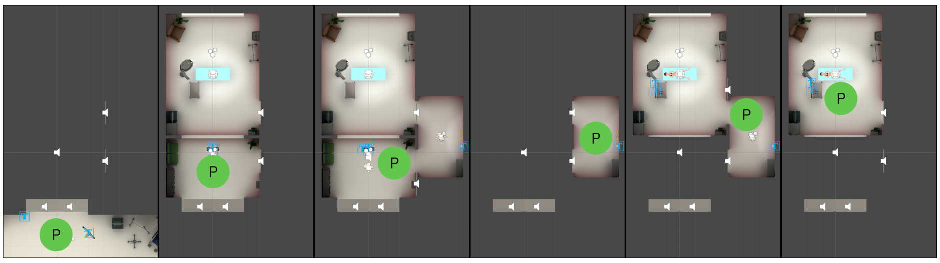


Figure 3.32: Example of portal culling in the game, the green circle with P marks the location of the player.

events OnOpen and OnClose. In the OnOpen-event the next room is enabled and in the OnClose-event, which fires right after the door is fully closed, the previous room is disabled. The opening of a door is in turn triggered by some form of player action. For instance, in the office selecting the correct patient will trigger the next door to open, while in the anteroom washing hands and equipping the protective gear acts as the trigger. Triggers in the form of volumes are placed inside each room so that the players presence can be detected, thus closing the door behind.

Chapter 4

Results and Discussion

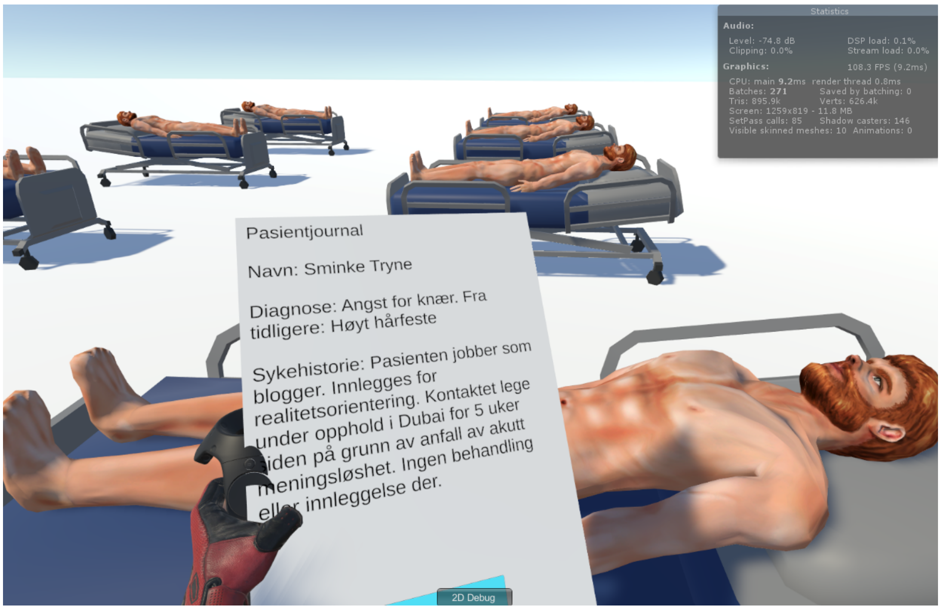
In the previous chapter, we looked at how we went from a clinical practice guideline to a VR game and the various challenges involved with such an endeavour. In this chapter, we look at the results of the tests we have performed. First, we look at the functional aspects of the game, such as how well the game does at reproducing information from the guideline and its perceived usefulness and effect. Afterwards, we look at the measurements of performance and optimization.

How well the game reproduces the information in the guideline and how a practitioner should apply such information, is an indicator of how well our methodology is for creating a VR game from the guideline. We make an assessment based on data recorded from user tests conducted throughout the development process, as well as data from a final round of testing with domain experts. Furthermore, we conducted another test with our external consultants at Helse Vest IKT after having modified the game based on feedback from the final tests. We will now look at the results from these tests.

4.1 Iterative testing

From the beginning of the development, we have had regular meetings with our external advisors at Helse Vest IKT. Two of those consultants have advised us on the technical aspects of the project, with one of them also having prior experience within healthcare. However, our principal advisor has advised us purely on the healthcare and gamification aspect of the game. Throughout these meetings, we continuously received feedback, either based on tests or from video recordings of gameplay. Naturally, the game saw many radical changes in the initial phase as new ideas were brought forth. The development of new features was often planned in the meetings, and features were tweaked according to feedback.

During the first meetings, we identified the selection of the patients and the screening itself as the main components of the game. Initially, we imagined that the game would take place in a single room. All patients lay on hospital beds, and the player could go around and inspect them. The patient journals would rest on a table next to the patient, or hang from the patient's bed. The journals could be picked up with the controller and be read to get the necessary information about each patient. In Figure 4.1, the player is holding a journal.



```
Statistics
Audio:
  Level: -74.8 dB              DSP load: 0.1%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      108.3 FPS (9.2ms)
  CPU: main 9.2ms  render thread 0.8ms
  Batches: 271          Saved by batching: 0
  Tris: 895.9k          Verts: 626.4k
  Screen: 1259x819 - 11.8 MB
  SetPass calls: 85     Shadow casters: 146
  Visible skinned meshes: 10  Animations: 0
```

Pasientjournal

Navn: Sminke Tryne

Diagnose: Angst for knær. Fra tidligere: Høyt hårfeste

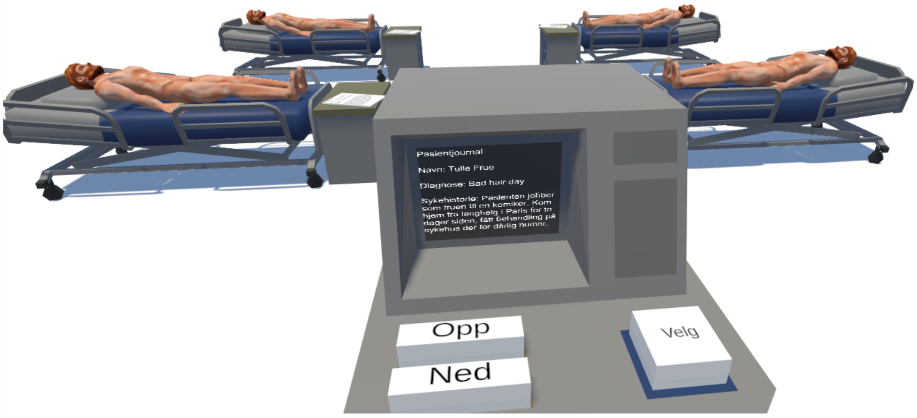
Sykehistorie: Pasienten jobber som blogger. Innlegges for realitetsorientering. Kontaktet lege under opphold i Dubai for 5 uker siden på grunn av anfall av akutt meningsløshet. Ingen behandling eller innleggelse der.

Figure 4.1: Holding a patient journal. In-game screenshot from an early iteration of the game.

We imagined that every patient would have a button close to them that the player could press to select that patient for screening. We experimented with a button at the bottom of the patient journal. However, testing revealed that this was awkward to use, and shortly after we instead tested with a central *patient selector*. An implementation of the latter option can be seen in Figure 4.2. Our advisors tested the implementation. Feedback showed that the concept worked well, and a similar implementation is in the final version of the game.

When all of the text on the patient journals was available on the patient selector, we realised that the player had no real incentive to walk around and inspect the patients. Also, visible cues are, in this game, not at all necessary for the decision to screen for MRSA; the player could instead stand by the patient selector and read the journal information there. This eventually led to the removal of the patient journals next to the patients.

At a visit to the infection control ward at Haukeland hospital, we got the idea of adding an anteroom to the game. In this case, the patient selector would be in a different room than the patients. We were initially a little resistant to



Pasientjournal

Navn: Tulle Frue

Diagnose: Bad hair day

Sykehistorie: Pasienten jobber som fruen til en komiker. Kom hjem fra langhelg i Paris for to dager siden, fått behandling på sykehus der for dårlig humor.

Figure 4.2: The prototype with a patient selector. The player can browse the patients and select one, as well as read the same information that is on the journals spread among the patients. This is an in-game screenshot from an early iteration of the game.

do this idea, as we felt we lost a bit of the VR utilisation if the players did not have to move around and check each patient. One idea was to have patients in individual rooms, each with an anteroom. However, feedback showed that this would lead to too much movement, which was unpractical due to the VR constraints. Besides, it was also unnecessary because the textual descriptions provided all the required information to make a decision. However, allowing the player to see the patient would make possible the use of visual features such as sex/gender, distinct hairstyles and wounds, which could add positively to the experience. In the end, we decided to show the patient through a big window in front of the patient selector.

We established early that the game should be available for all of the most popular HMDs. Some discussion was had if a smartphone version of the game could also be made. Due to the limiting nature of the smartphone platform, the screening of the patient would be very different gameplaywise from the VR game. Considering both this downgrade in gameplay and the time limit of the project, this had a low priority, and the idea was in the end abandoned. It is, however, an appealing thought and is given further consideration in Section 6.2.

4.2 Planned tests

The game in its final form was initially meant to be tested by two groups of participants, both of which we considered to be possible users upon release of the game.

The first group was ten nurses from the infection control ward at Haukeland Hospital; these participants would be proficient in the domain (i.e. infection control). We had scheduled testing for Friday 13/03/20 at 12:00 - 16:00 and Monday 16/03/20 at 12:00 - 16:00, with five participants each day at a designated room at Haukeland Hospital. At that time, the number of COVID-19 cases had started increasing, and for safety reasons, we had to cancel the tests.

The second group was nursing students at Western Norway University of Applied Sciences. We advertised the tests for a class of students at Monday 24/02/20 around 11:30 - 11:45 and asked them to apply on a digital form should they be interested. Seven of the students showed interest and applied. We would likely have performed the tests at 18/03/2020, but these tests also had to be cancelled for safety reasons and due to Kronstad Campus shutting down.

The following is a description of how the tests were to be performed:

1. Participant is given a brief description of the project and the test
2. Pre-testing questions:
   (a) What is your background?
      - Current occupation/profession (only if relevant)
      - Previous occupations/professions (only if relevant)
      - Years of experience within healthcare

   (b) Do you have any experience with VR, if so how much?
3. Testing (the participant is observed playing the game, anything of interest is noted)
4. Post-testing questions:
   - The participant fills out a System-Usability-Scale [63] questionnaire in order to gauge general usability. The questionnaire consists of several statements, and the participant is prompted to assign a numerical value representing their agreement with the statement (ranging from "strongly agree" to "strongly disagree"). The word "system" was replaced by "game" and it was translated to Norwegian (see Appendix B).
   - Did you enjoy playing the game?
     - What made you enjoy/not enjoy playing the game?
   - To what degree would you say that the game portrays relevant information from the guideline?
   - Do you think the game would be suitable as a supplement for education/training?

  - If no, what makes the game unsuitable and what changes would have to be made for it to be suitable?

- Finally, anything you would like to add?

The setup of the tests was similar for both groups but would have had to be evaluated somewhat differently. For instance, the nursing students would not have had the same amount of experience with infection control, and their responses regarding the portrayal of information in the guideline would have to be weighed less so than that of the nurses at Haukeland Hospital.

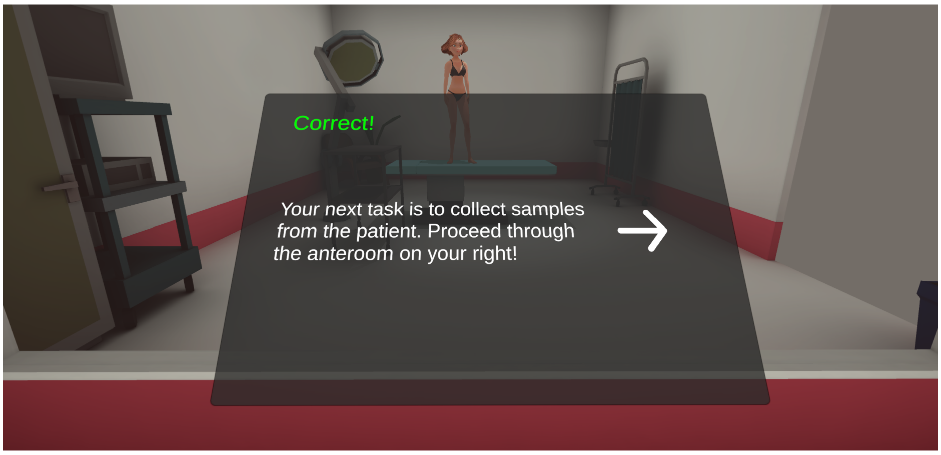
4.3 Final tests

As mentioned, we had to cancel the tests due to the outbreak of COVID-19. At a later time, after the government had loosened some of the restrictions, we gained access to Kronstad Campus, which meant that we could again perform tests, albeit with some restrictions. We ended up with 2 participants and therefore opted for a semi-structured interview form which partly replaced the post-testing questions as described above. This kind of interview allowed for a more in-depth review of the game.

Both participants are active in the healthcare profession, with nearly two decades of experience. They also have relevant experience within teaching and simulation, although minimal experience with VR. They described their experiences with VR as brief and rare, and neither had ever used hand controllers before. After prompting them for their career background, we explained how the controllers work and assisted them in equipping the VR headset.

Initially, there was some confusion with regards to interacting with the VR system for the first time, as is to be expected with someone with little experience with VR and hand controller peripherals. After having pressed the *start game* button, teleported into the next room and started interacting with the UI board there, the participants seemed to loosen up. They could play through the game independently with much more success, although not entirely without error. With regards to gameplay, one of the participants misunderstood the patient selector as a choice of which patient they wanted to screen, rather than the task of choosing the correct patient based on the provided information. It is hard to tell whether this was a one-off mistake by the participant or that the game is not making it apparent enough, and further testing is necessary to pinpoint this.

Both participants were visibly somewhat confused as to what they needed to do after having chosen the correct patient. Figure 4.3 shows the instructions given to the player. The issue we observed was that the arrow, which is animated along the X-axis to indicate the direction the player needs to shift attention to, was mistaken for a button. This mistake may stem from its similarity to buttons in common everyday applications, where its use is associated with a



Correct!

Your next task is to collect samples *from the* patient. Proceed through *the anteroom* on your right!

Figure 4.3: Instructions after having chosen the correct patient

primary action like "ok", "close", "continue" and "next". When the player attempts to press it, attention seems to be drawn to the lack of response rather than their next objective. Of course, more testing is required to be certain that it is an issue, but one possible solution would be to close the interface if the player attempts to press the button. This could force the players attention to the surroundings, possibly yielding the desired result. If this does not work, one should consider adding additional signals such as an attention-grabbing light, additional directional sound and visual cues.

Both participants said they would need technical guidance if they were to use the game. Again, this was to be expected given their lack of experience with VR beforehand. A reservation was given that their age demographic could have something to do with it and that the younger demographic whom would likely have more prior experience with VR systems and hand controllers, could have an easier time using the game without guidance.

As to the question of whether the game would be suitable as a supplement for education/training, both participants expressed an interest in more elaborate scenarios. As mentioned in Chapter 1, this is outside the scope for the thesis. However, it goes without saying that more elaborate scenarios, as described by the participant, would be of interest in the further development of the game. The participants also mentioned some specific modifications or additions that would make the game even more suitable as an educational/training tool. First of all, as mentioned in Section 3.6, before the final tests, we had decided to simplify sampling of the patient's nose such that it only required that the swab touched one of the nostrils. They quickly reacted to this when playing the game, and noted how it was odd that they only had to touch one nostril to collect the sample. In other words, they felt it was an oversimplification. As mentioned in Section 3.6 this was later changed to the player having to sample both nostrils with the same swab.

Another point mentioned by one participant was that in their playthrough, the patient had neither a wound nor a urinary catheter. The reason for this is that patients are picked randomly, and not all patients have a wound or urinary catheter. When informed about this functionality, the participant questioned whether or not it would be beneficial always to include a wound and urinary catheter. As mentioned in Chapter 3, patients are loaded and selected randomly from a JSON-file. Therefore modifying this file such that all patients have a wound and urinary catheter is trivial.

At the time of testing the anteroom consisted of two interactions, hand sanitizing and equipping protective gear. In other words, equipping the coat, mask and gloves was a single action. This design choice was made, as mentioned in Chapter 3, as part of prioritization and in the interest of reducing gameplay complexity. Additionally, a physically accurate cloth simulation for the garments would be complex to implement and potentially require considerable computational power. Therefore an alternative simplified form of interaction had to be implemented. Also, the guideline applies to a variety of scenarios where there may not even be an anteroom, meaning that it was considered less relevant than the other parts of the game. Feedback from the participants did, however, suggest that this part of the game could be expanded and that a less simplified interaction could be beneficial. Therefore somewhat less simplified version was later implemented, where equipping the gloves is a separate interaction from equipping the coat and mask. This change led to positive feedback from the consultants at Helse Vest IKT, who felt that the interaction was more natural than before.

Another feedback regarding the anteroom and protective equipment is the procedure of unequipping the protective gear after having collected samples. Adding this as an additional task is something that can be evaluated in future work. The general nature of such routines, i.e. that they apply not only to MRSA but all airborne pathogens, prompts the question of whether one should consider constructing a game explicitly for exercising these kinds of routines (those mentioned in [57]).

Finally, with regards to the question of having fun, both participants gave the impression that it was fun and that it most certainly was better than just reading the guideline. Both participants mentioned that a more challenging scenario could have been more enjoyable, which makes sense considering their level of expertise.

4.4 Numerical measurements of performance

In the previous chapter, we mentioned a technique called portal occlusion culling and how we utilized it to adapt to the limitations of the Oculus Quest. To measure the performance difference between having portal occlusion culling enabled and disabled, we deployed two versions of the game. We then used a tool called OVR Metrics [64] to record the average number of frames per second and the GPU utilization percentage. Figure 4.4 and Figure 4.5 show these metrics in addition to the display refresh rate of the device as a dashed line, that dashed line is where we would like the average frame rate to be at all times.

As we can see in Figure 4.4 GPU utilization is consistently very high, except for when the game is loading (which completes around the 10-second mark in the graph). High GPU utilization indicates that the amount of work the GPU has to do is too high, which would explain why the average number of frames per second is unstable and often too low. An unstable framerate negatively impacts the user experience and increases the difference between what the user does and subsequently perceives because of the increased latency.

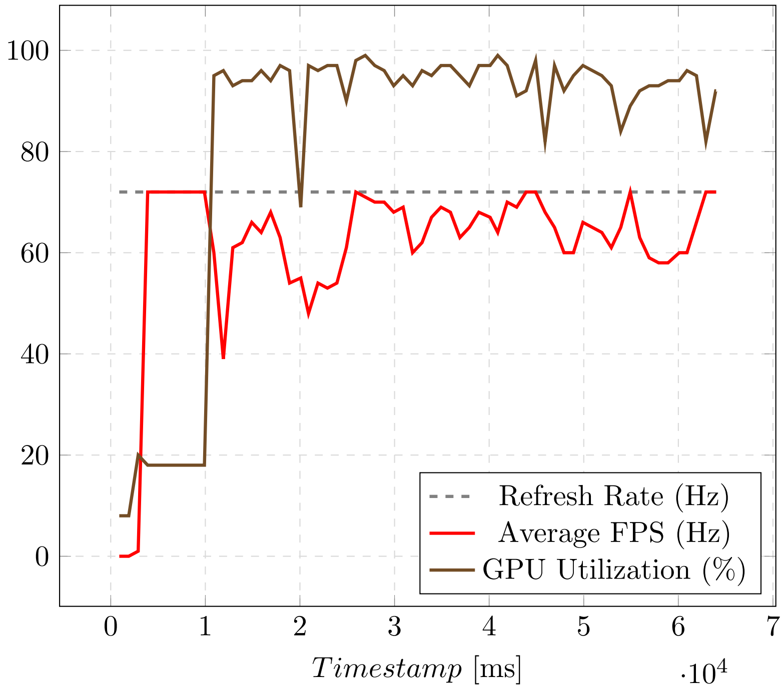


Figure 4.4: Oculus Quest without culling

In Figure 4.5 we can see that the GPU utilization generally is lower with culling enabled. There are still some peaks reaching just below 100%, which suggests that it could be improved further. However, as we can see from the average frame rate, the performance has been dramatically improved. Rarely do we see the frame rate drop below 60 and it seems to be quite stable.

It is important to note that the average frame rate is not the best measurement for the performance. Take, for example, the case where the time spent on one frame alternates between $20ms$ and $7.77ms$ (recurring). The average frame

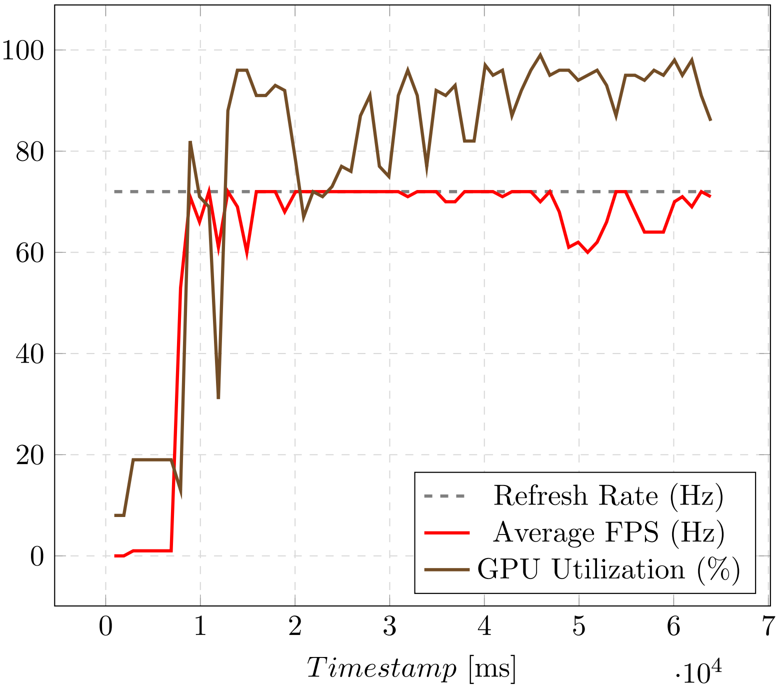


Figure 4.5: Oculus Quest with culling

rate would, in this case, be 72 because:

$$\frac{20ms + 7.77ms}{2} \equiv \frac{27.77ms \times 36}{2 \times 36} \approx \frac{1000ms}{72}$$

, but because the first frame takes $20ms$ the resulting image arrives too late for the display to present it which effectively halves the perceived framerate.

Another factor is that only a single playthrough was measured for each version. Capturing multiple playthroughs with a human would give more data, but the variability in time spent makes it hard to align the data. Additionally, differences in where the player looks will introduce a new unknown variable. Instead, the tests could be improved further by programming an automated playthrough and averaging multiple of those. These tests would also give a better indication of what parts of the game require further optimization.

After having implemented the optimization as mentioned above, one of our external advisors from Helse Vest IKT noted that there had been a noticeable improvement and that it had positively affected the entire experience. This makes sense because a substantial portion of the experience is dependent upon the visual stimuli, which shows the importance of ensuring a sufficient and stable framerate in VR applications.

Chapter 5

Conclusion

In our work, we have developed a VR game based on Helse Bergen's MRSA CPG. Given the results presented in the previous chapter, we attempt to answer to what degree we have reached our initial goals, which includes answering the research questions from Section 1.3.

5.1 What are the challenges with developing a virtual reality game based on the clinical practice guideline?

While developing and testing the game, we identified and dealt with several challenges. Some of the challenges were identified as part of the development process (Chapter 3) and others as part of our tests (as presented in Chapter 4). We will now present those challenges and give our concluding remarks.

From guideline to game

There are local differences in the clinical practice of different healthcare institutions, even within the same country. This is a huge challenge when designing any step-by-step guide on how to follow a CPG, and even more so in a VR setting since it is so detailed. Because the game targets healthcare professionals all over the country, it has to conform to the practices of all of the healthcare institutions in which they are employed.

One concrete example is how to handle samples after collecting them in the screening process. When we considered how to implement this into the game, we were informed by our domain expert advisor that it could vary a lot between institutions. In fact, it varied so much that we decided not to include it in the game in its entirety. Instead, we chose to use a text popup which describes how a sample should be marked because this is the only aspect of it which applies to every institution nationally. If there are too many such examples in a CPG, where parts of it must be reduced to text popups or removed from the game, the game can become less interesting and helpful.

Balancing gamification and accuracy

Creating good gameplay is an especially challenging part of transforming the guideline to a game. The very general nature of the guideline, i.e. that it applies to a large variety of situations, means that the trade-offs between accuracy, generality and gamification can become especially apparent. For instance the guideline does not mention a concrete infectious droplets control regime, but instead assumes that the reader already has general prior knowledge about how their situation is handled. This also applies for example to how collected samples should be labelled, and is reflected in feedback from the final tests where the participants show interest in more elaborate scenarios in general, and more detailed and elaborate interactions. However implementing more detailed and elaborate interactions poses a challenge as mentioned, both because of the possible differences in routines between healthcare institutions and because of possible implications for gamification.

Results from the final tests with two domain experts showed that there was an interest for more elaborate scenarios and more detailed interactions. With regards to more elaborate scenarios this is considered out of scope for this thesis, but is definitely something to consider for future work. With regards to more detailed interactions this shows the challenge of balancing accuracy and gamification. Some of the simplifications that had been made for the gameplays sake and out of the necessity of prioritizing features (as mentioned in section 3.6) were noticeable by experienced simulators. Some of these were rectified after the fact, but others require more testing to be able to make an informed design decision. In this case the originally planned tests would likely have provided a better foundation for making such decisions.

Feedback with regards to the anteroom with droplet-transmission precautions was also mentioned in Chapter 4. As mentioned the anteroom was initially not part of the design of the game and was later considered less important than the other parts. Therefore, the interactions in this section of the game were simplified to a large extent. In the feedback, we saw that these simplifications might have been too drastic, and although this has partly been rectified later, tests still show some difficulties with this part.

Performance and optimization

The measurements of performance with and without optimizations showed that there was a definite improvement. The combined effort of optimizing the assets that were used in the game, as well as employing frustum and portal occlusion culling, lead to a noticeable improvement as noted by one of our technical advisors at Helse Vest IKT. The advisor had previously tested a version without all these optimizations. When testing the game with the optimizations, he noticed a definite difference and said that there was a significant improvement in the experience. This reflects the importance of achieving adequate performance when creating VR games.

Cross-platform support

In the future, targeting several platforms will become more accessible because of standardisation efforts like OpenXR, as mention in Section 2.5.5. However, in this thesis, due to cross-platform support made available by the game engine, as well as a framework for VR interactions, we found that creating a fully cross-platform game was still possible. Still, certain aspects had to be taken into account, such as differences in hand relative tracking data. For example, there was a difference in rotation of about 45 degrees between the HTC Vive and Oculus Touch controllers, which requires calibration to make it consistent.

Another platform inconsistency is the physical to virtual input mapping. This inconsistency is apparent with the *teleport*-button, which is a vital part of the gameplay. On the HTC Vive controllers, the button is large and centred. While on the Oculus Quest controller it is much smaller and right next to another button. Consequently, there is a difference in the level of intuitiveness; more specifically, we saw that users found the *teleport*-button on the HTC Vive controller more intuitive.

Another issue is how it feels to pick up the tools with the different controllers. Most VR-controllers have specific *grip* buttons, often placed on the side of the controller, that are supposed to activate when the hand is squeezing, thus simulating gripping something in real life. We also found that these buttons have varying levels of intuitiveness depending on the controller. On the Oculus Touch controller, it is a large button on the side which you will typically feel with your middle fingers when you are holding the controller. On the HTC Vive controllers, however, they are thin enough to be overlooked by inexperienced VR users. Also, the controllers allow for more flexibility with regards to hand placement. Hence, the hand might not be in the right position to press the buttons properly. This influenced the decision of using the trigger button for grabbing, which is similar on most controllers. All these platform related inconsistencies show that testing with different HMDs throughout development is crucial, even when using a cross-platform framework.

We also found that cross-platform support had implications for the performance of the game. Targeting the mobile-powered Oculus Quest HMD required the game to be significantly less computationally heavy compared to a game that only targets computer-tethered systems. In other words, creating a cross-platform game can mean that more effort must be put into optimisation.

5.2 How does the use of virtual reality, and gamification affect practitioner motivation?

Giving a conclusive answer to this question is hard in and of itself, and was made even harder because of the reduced number of final tests. However, tests performed throughout the development process and as part of the final tests with our advisors at Helse Vest IKT, showed that participants were excited and having fun when playing the game. The level of excitement we saw can indicate that the game has a positive effect on the practitioner's motivation. We could tell that players reacted emotionally to being rewarded or punished with points, which suggests that they were concentrating on getting the results they wanted.

In the final tests (described in Section 4.3), we saw that the testers found the game challenging to use. There is a steep learning curve with all VR applications which could leave the first time VR users demotivated, especially without guidance. On the other hand, in tests with our advisor Eva C. Backer at Helse Vest IKT, we consistently observed a high level of excitement, even though she did not have any prior experience with VR. This suggests that there is a variation in how people respond to the technology, but it is hard to tell which factors are in play. This could, however, suggest that for some players, the technology in itself can provide a baseline level of excitement that could be beneficial for training and learning. Also, this baseline level of excitement can be useful for getting attention to the game and the problem of MRSA. However, it may diminish over time as players get more experienced with the technology.

The planned tests (described in Section 4.2) would have been useful in answering this research question. With the testing of ten healthcare professionals as well as some nursing students, the chances are that at least some of them would have prior experience with VR. This could give especially useful feedback as first time users can be overwhelmed by the experience and controls alone. We could compare the answers of experienced VR users and see if the motivation is as strong without the baseline excitement that comes with the experience of trying out VR for the first time. Also, for those with prior experience with VR, the controls would likely be easier to get into, resulting in less frustration.

5.3 Creating a publishable game

The initial goal of this thesis was to develop a fully functional and publishable game for Helse Vest IKT. Throughout the development process, the game was tested by our advisors at Helse Vest IKT regularly. These tests proved to be tremendously helpful with regards to guiding us towards a finished and publishable game. Through the tests, we identified and implemented several significant changes, and subsequent testing showed that the changes had positively affected the result. The final tests with the participants who had not previously played the game showed that there was a difference in their focus. In retrospect, having a larger pool of testers with more varied backgrounds, within relevant areas of healthcare, would likely have served as an even better guide for the development.

We saw, however, during the final tests with the domain experts, as discussed in Section 4.3, that the test subjects who had no experience with the game itself could successfully utilise their professional knowledge to score points in the game. If we disregard mistakes caused by lack of experience with VR, they selected the right patient to be screened and chose the correct tools to collect samples from the patient in the right places. This shows that the game demanded established knowledge about MRSA screening, and shows that the gameplay reflects the guideline.

Although we consider our goal of creating a publishable game as reached, that does not mean it is perfect. Many exciting ideas were brought forth throughout the development process. Some were unfortunately not possible to add given the scope of the thesis and have instead been described in Chapter 6.

Chapter 6

Further Work

6.1 Generalising for other CPGs

One obvious improvement would be if the game was expanded to cover more clinical practice guidelines than the one for MRSA. One could be inspired by the model-driven approach of Nyameino et al. [27]. Ideally, core aspects of the game could be transformed into a framework where a knowledge engineer with medical knowledge easily can construct a medical procedure with a set of parameters. This could be done in a quiz-like game as in Nyameino et al.'s example, but seems more difficult in a VR application with hands-on interaction. Many medical procedures have vastly different types of interactions. The screening for MRSA is done by picking up tools and using them at various places on the patient's body. Screening procedures that can be described the same way can be generalised with swapping out the items and adding targets to other places on the body. If the procedure has a different way of interaction, like the use of a machine or having to move the patient, it would require additional development which is expensive.

Some parts of the game can more easily be generalised than others. The first part of the game where the player is presented with four patients and have to choose one of them to screen can easily be generalised to other illnesses. As it does not have any specific interactions, changing the theme from MRSA to other illnesses does not take much work. Because of how the patients are loaded in the game, all it takes is swapping a JSON file. One could even create a web interface for knowledge engineers to easily add patients as part of their guidelines, like in Nyameino et al.'s proposed method of their system.

6.2 Expanding to other platforms

As it is now, the game is only available for HMDs that support VR. While these headsets can deliver compelling experiences, they have yet to be accessible to most people. Additionally, they can often be impractical to use, such as when there is limited space available, or you are unsure of your surroundings (like if there are other people nearby). Compare this to how common it is to own computers and the ease-of-use of smartphones, which can be used practically everywhere. It would be cheaper for most institutions to provide games to their employees or students on more traditional platforms, rather than on VR. The advantages of targeting other platforms are clear. Gamification apps are typically made for web and mobile, so they would be natural to consider for this game too. The game logic was purposefully written to be platform-independent so that the game can be ported to other platforms.

The biggest question is how such a porting process would impact the gameplay. Different parts of the game would be affected differently from reducing it to a more traditional gamification platform. Patient selection would probably be the least affected, as you could present information about four patients and let the users choose one of them in a similar manner on traditional platforms. Additionally, different buttons could, for example, represent putting on safety equipment in the anteroom. The player would have to press them in the right order. Alternatively, the player could be given the task of organizing them in the correct order. The actual screening part is the trickiest because it utilizes the spatial aspect of VR, which allows for simulating physical interactions.

If the game were to be expanded to another platform, it would be interesting to see how people react to the different versions. One could imagine testing both versions with different healthcare practitioners and students and seeing what advantages one can offer over the other.

6.3 Adaptive gameplay

As the player plays through the game, indications of how the player is performing are stored in a *Score*-object (see Section 3.4). The data in the score include the number of points and records of mistakes. All incorrectly selected patients are stored, and all screening mistakes are collected as pairs of tool and target. Currently, these lists are not used in any way by the game. Some of the values in the *Score*-object are already shown at the end of the game, such as points and whether you contaminated the safety equipment or not. However, meaningfully presenting the mistakes made when screening and selecting a patient is challenging. Instead, these mistakes can be used to create a profile of the player.

With a player profile with data of past mistakes, the game could pick up trends for which types of mistakes the player makes. Then the game could present tasks that the player seems to struggle with the most. In this way, the gameplay can be tailored to the exact learning needs of the player. However, this has some practical problems with our game. Firstly, for the player profile to be built, the same person must play the game several times. If the game is going to be used in a VR lab, where several co-workers will share the same setup, the player profile will be cluttered by different players' mistakes. Secondly, the game takes longer to play through using a VR interface, so the number of playthroughs by one player will probably be lower, making a smaller data set than if it was on a non-VR platform. This makes this idea work better in combination with the aforementioned idea to expand it to different types of platforms (see Section 6.2).

Appendix A

MRSA Infection Control Guideline

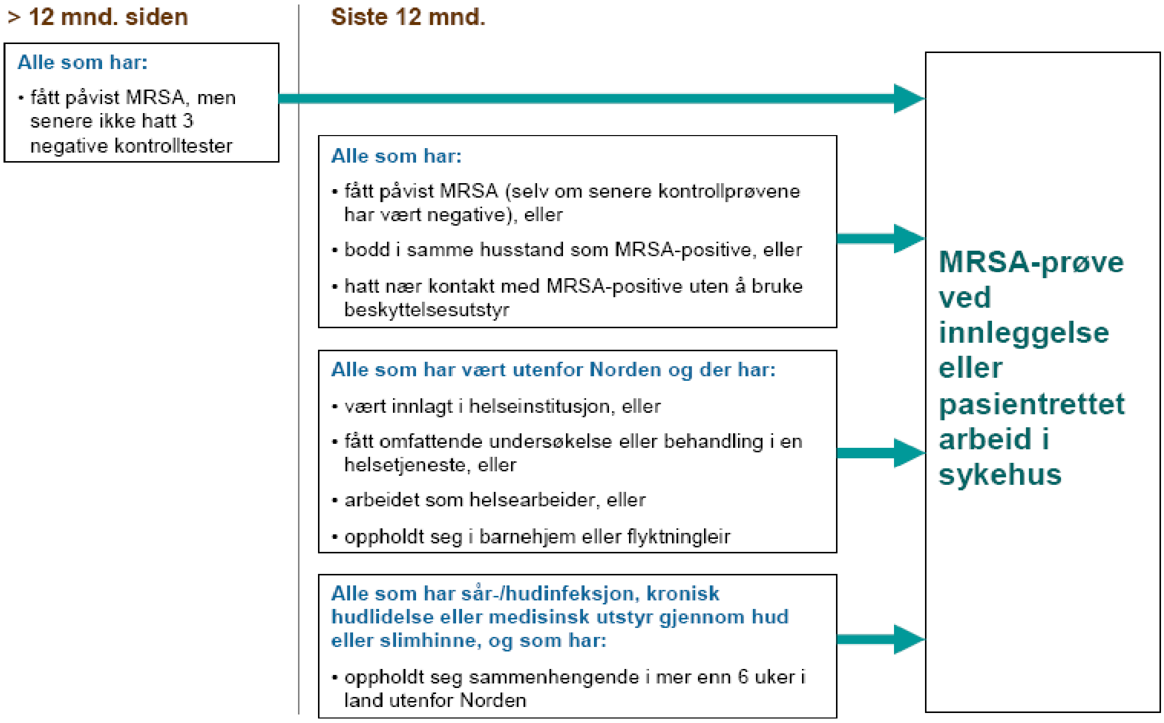
MRSA-informasjon til pasienter på flere ulike språk (Folkehelseinstituttet)

En pasient med MRSA (påvist bærerskap eller tilfeldig funn av MRSA i en klinisk prøve) skal isoleres etter retningslinjene for dråpesmitteregime i den avdelingen pasienten hører hjemme.

Gule stafylokokker (*Staphylokokkus aureus*) er en bakterie som er vanlig å finne hos mennesker. Hos friske personer gir bakterien sjelden sykdom.

- Meticillinresistente gule stafylokokker (MRSA) er gule stafylokokker som er motstandsdyktige mot enkelte typer antibiotika.
- Forekomsten av MRSA er lav i Norge, men betydelig høyere i resten av Europa og i andre verdensdeler.
- Økt reisevirksomhet øker smittepresset mot Norge og norske helseinstitusjoner.
- Nasjonal MRSA-veileder har som mål å forebygge spredning av MRSA i norske helseinstitusjoner.
- MRSA-testing av pasienter og helsepersonell er hjemlet i Smittevernloven §33.

MRSA-undersøkelse av pasient ved innleggelse
Ved innleggelse i sykehus tas MRSA-prøve av følgende pasienter:



> 12 mnd. siden

Alle som har:
- fått påvist MRSA, men senere ikke hatt 3 negative kontrolltester

Siste 12 mnd.

Alle som har:
- fått påvist MRSA (selv om senere kontrollprøvene har vært negative), eller
- bodd i samme husstand som MRSA-positive, eller
- hatt nær kontakt med MRSA-positive uten å bruke beskyttelsesutstyr

Alle som har vært utenfor Norden og der har:
- vært innlagt i helseinstitusjon, eller
- fått omfattende undersøkelse eller behandling i en helsetjeneste, eller
- arbeidet som helsearbeider, eller
- oppholdt seg i barnehjem eller flyktningleir

Alle som har sår-/hudinfeksjon, kronisk hudlidelse eller medisinsk utstyr gjennom hud eller slimhinne, og som har:
- oppholdt seg sammenhengende i mer enn 6 uker i land utenfor Norden

MRSA-prøve ved innleggelse eller pasientrettet arbeid i sykehus

Kilde: Nasjonal MRSA – veileder, Nasjonalt Folkehelseinstitutt og Helsedirektoratet (juni 2009**)**

Hva defineres som omfattende undersøkelse eller behandling

Med omfattende undersøkelse eller behandling menes større prosedyrer der fremmedlegemer føres gjennom hud eller slimhinner (ikke enkel blodprøvetaking). Eksempler er mindre kirurgiske inngrep, dialyse, innlegging av venekateter, urinkateter, dren/tube, stell av større sår (eks suturering)samt tannbehandling av typen implantat, kroner og broer.

Hurtigtest for innlagte pasienter

Mikrobiologisk avdeling tilbyr MRSA-screening med PCR-metodikk for innlagte pasienter(MRSA hurtigtest). Dette er et tilbud for å korte ned tiden pasienten ellers måtte ligget isolert i påvente av dyrkningssvar.

- MRSA-PCR rekvireres i DIPS
- Prøve tas på samme måte som til MRSA-dyrking (penselprøve, eSwab).
- Analyseresultatet vil normalt være klart innen to timer etter at prøven er mottatt på laboratoriet (ring «MRSA-telefonen» 74644).
  Se LAB-info på Innsiden for detaljer.

MRSA-prøver tas fra følgende lokalisasjoner:

1. Begge nesebor (samme pensel)
2. Svelg inklusiv tonsiller.
3. Perineum
4. Sår, eksem, puss, arr etter infeksjon eller aktive hudlidelser
5. Rundt instikksteder for fremmedlegemer (katetre, dren, trakeostoma etc.)
6. Kateterurin

For detaljert veiledning, se MRSA- praktisk fremgangsmåte for prøvetaking

Hva skal skje videre med pasienten?

I påvente av prøvesvar isoleres pasienten på enerom etter retningslinjene for dråpesmitteregime på den sengeposten pasienten tilhører.

Negativt prøvesvar:

- Isolasjonstiltaket opphører når det foreligger negativt svar på MRSA-prøvene fra mikrobiologisk avdeling.

Positivt prøvesvar:

- Dråpesmitteregimet opprettholdes
- Retten til nødvendig undersøkelse og behandling gjelder også for isolerte pasienter, og MRSA-bærerskap må ikke forsinke nødvendig undersøkelse eller behandling

MRSA-positive pasienter i psykiatrisk avdeling bør ha enerom, men behøver ikke isoleres. Prinsippene for håndtering av MRSA i psykiatrien er de samme som for håndtering av MRSA i sykehjem (se nasjonal MRSA-veileder punkt 4.13). Smittevernpersonell ved Seksjon for pasientsikkerhet kan kontaktes for veiledning.

MRSA-undersøkelse av pasient i poliklinikken

Dersom pasienten skal til poliklinisk konsultasjon ber sykehusene i innkallingsbrevet henvisende lege om hjelp til testing slik at MRSA-svaret foreligger før oppmøte til time. Det er likevel ikke alltid dette er gjort på forhånd.

Pasienten skal uansett tas imot i poliklinikken som planlagt og må ikke avvises med begrunnelse i manglende forhåndstesting.
Retningslinjene for isolering av MRSA-bærere i sykehus er i all hovedsak rettet mot inneliggende pasienter. Smitteoverføring skjer først og fremst ved tett kontakt over tid, og oppegående pasienter som kommer til kortvarig poliklinisk undersøkelse og vurdering utgjør en svært liten smitterisiko.

- Pasienten har rett til helsehjelp og et eventuelt MRSA-bærerskap (mistanke om eller påvist) skal ikke forsinke nødvendig undersøkelse eller behandling.
- En pasient med mistenkt eller påvist MRSA kan sitte på venterommet i poliklinikken sammen med de andre pasientene.
- Helsepersonell trenger ikke å bruke eget beskyttelsesutstyr dersom de bare skal samtale med pasienten.
- Helsepersonell bruker hansker, smittefrakk og munnbind (dråpesmitteregime) ved somatisk undersøkelse av pasienten (tett fysisk kontakt mellom helsepersonell og pasient).
- Etter konsultasjonen desinfiseres kontaktpunkt i undersøkelsesrommet, se Desinfeksjon av flater og utstyr

Eksterne referanser

Nasjonal MRSA-veileder, FHI
Lov om vern mot smittsomme sykdommer (smittevernloven)
Analyseoversikten.no

Appendix B

System Usability Scale

Evaluering av MRSA spill

* Required

Brukervennlighet

Nå vil du få 10 påstander om brukervennlighet og rangere om du er enig eller uenig med påstanden fra 1-5. Du trenger ikke bruke lang tid på å tenke deg om, men heller trykk hva du føler med én gang. Er du usikker, svarer du i midten.

Jeg synes spillet var unødvendig komplisert *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg kunne tenke meg å bruke dette spillet ofte *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg synes spillet var lett å bruke *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg tror jeg vil måtte trenge hjelp fra en person med teknisk kunnskap for å kunne bruke dette spillet *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg syntes at de forskjellige delene av spillet hang godt sammen *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg syntes det var for mye inkonsistens i spillet (Det virket “ulogisk”) *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg vil anta at folk flest kan lære seg dette spillet veldig raskt *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg synes spillet var veldig vanskelig å bruke *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg følte meg sikker da jeg brukte spillet *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Jeg trenger å lære meg mye før jeg kan komme i gang med å bruke dette systemet på egen hånd *

	1	2	3	4	5	
Sterkt uenig	○	○	○	○	○	Sterkt enig

Back Submit

Never submit passwords through Google Forms.

Bibliography

[1] Institute for Public Health and the Environment (RIVM), Epidemiology and Surveillance, 3720 BA Bilthoven, The Netherlands, “Proportion of mrsa human blood isolates from participating countries (ears-net and caesar) in 2008 and 2017.,” 2017. Conversion by Trex2001, translation by Phatom87 (Wikimedia Commons CC BY-SA 4.0), `https://commons.wikimedia.org/wiki/File:EARS-Net_CAESAR_MRSA_2017-en.svg` (accessed: 03-02-20).

[2] HealthCare VR Innovations, Inc., “Surgera VR.” `https://store.steampowered.com/app/763860/Surgera_VR/` (accessed: 2020-24-05).

[3] Immersive VR Education Ltd., “RCSI Medical Training Sim.” `https://www.oculus.com/experiences/go/878262692296965/` (accessed: 2020-25-05).

[4] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire, *Real-Time Rendering 4th Edition.* Boca Raton, FL, USA: A K Peters/CRC Press, 2018.

[5] The Blender Foundation, “Blender.” `https://www.blender.org/` (accessed: 2020-07-05).

[6] G. I. f. Q. National Center for Biotechnology Information, Cologne and E. in Health Care (IQWiG), “What are clinical practice guidelines?.” `https://www.ncbi.nlm.nih.gov/books/NBK390308/` (accessed: 2019-09-11).

[7] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[8] Valve Corporation, “Openvr.” `https://partner.steamgames.com/doc/features/steamvr/openvr` (accessed: 2020-21-05).

[9] M. Singer, C. S. Deutschman, C. W. Seymour, M. Shankar-Hari, D. Annane, M. Bauer, R. Bellomo, G. R. Bernard, J.-D. Chiche, C. M. Coopersmith, *et al.*, “The third international consensus definitions for sepsis and septic shock (sepsis-3),” *Jama*, vol. 315, no. 8, pp. 801–810, 2016.

[10] *MRSA-veilederen: Nasjonal veileder for å forebygge spredning av meticillinresistente staphylococcus aureus (MRSA) i helseinstitusjoner*, vol. 16 of *Smittevern (online)*. Oslo: Nasjonalt folkehelseinstitutt, 2009. `https://www.fhi.no/globalassets/dokumenterfiler/rapporter/2009-og-eldre/mrsa-veilederen.pdf`.

[11] F. Ruscio, J. Bjørnholt, T. Leegaard, A. E. F. Moen, and B. De Blasio, "Mrsa infections in norway: A study of the temporal evolution, 2006-2015," *PLOS ONE*, vol. 12, p. e0179771, 06 2017.

[12] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? – a literature review of empirical studies on gamification," in *2014 47th Hawaii International Conference on System Sciences (HICSS)*, (Los Alamitos, CA, USA), pp. 3025–3034, IEEE Computer Society, jan 2014.

[13] Duolingo, "Duolingo." `https://www.duolingo.com/` (accessed: 2020-11-05).

[14] HackerRank, "Hackerrank." `https://www.hackerrank.com/` (accessed: 2020-11-05).

[15] Khan Academy, "Khan Academy." `https://www.khanacademy.org/` (accessed: 2020-11-05).

[16] Z. Merchant, E. T. Goetz, L. Cifuentes, W. Keeney-Kennicutt, and T. J. Davis, "Effectiveness of virtual reality-based instruction on students' learning outcomes in k-12 and higher education: A meta-analysis," *Computers & Education*, vol. 70, pp. 29–40, 2014.

[17] Helse Vest, "Innovasjonsstrategi." `https://helse-vest.no/seksjon/planar-og-rapportar/Documents/Regionale%20planar/2016%20-%20Regional%20innovasjonsstrategi%202016-2020.pdf` (accessed: 2020-21-05).

[18] Helse Vest, "Helse 2035." `https://helse-vest.no/seksjon/styresaker/Documents/2017/02.02.2017/Sak%2001617%20Vedlegg%201%20-%20Ny%20verksemdstrategi%20-%20Helse2035.pdf` (accessed: 2020-21-05).

[19] Helse Vest IKT, "Stopp Sepsis." `https://stoppsepsis.no/` (accessed: 2019-20-03).

[20] E. C. Backer, "Gamification. hvordan det gikk da vi skulle lage spill om blodforgiftning." `https://www.linkedin.com/pulse/gamification-hvordan-det-gikk-da-vi-skulle-lage-spill-backer/` (accessed: 2020-11-05).

[21] M. M. Baddour, *MRSA (methicillin resistant Staphylococcus Aureus) infections and treatment*. Public health in the 21st century series, New York: Nova Science Publishers, 2010.

[22] I. E. Fjeld, “Korona-dødeligheten i italia kan skyldes resistente bakterier,” https://www.nrk.no/urix/korona-dodeligheten-i-italia-kan-skyldes-resistente-bakterier-1.14959498 (accessed: 2020-27-04).

[23] B. Amundsen, “Very low mortality rate from coronavirus in norway compared to other countries,” https://sciencenorway.no/crisis-epidemic-mortality/very-low-mortality-rate-from-coronavirus-in-norway-compared-to-other-countries/1661751 (accessed: 2020-27-04).

[24] K. N. Lohr, M. J. Field, *et al.*, *Guidelines for clinical practice: from development to use.* National Academies Press, 1992.

[25] A. Fretheim, S. Flottorp, and A. Oxman, “Effect of interventions for implementing clinical practice guidelines,” 2015. https://www.fhi.no/en/publ/2015/effect-of-interventions-for-implementing-clinical-practice-guidelines/.

[26] W. C. McGaghie, S. B. Issenberg, E. R. Petrusa, and R. J. Scalese, “A critical review of simulation-based medical education research: 2003–2009,” *Medical Education*, vol. 44, no. 1, pp. 50–63, 2010.

[27] J. Nyameino, F. Rabbi, B.-R. Ebbesvik, M. Were, and Y. Lamo, “A model driven approach to the development of gamified interactive clinical practice guidelines,” 05 2019.

[28] G. Duque, S. Fung, L. Mallet, N. Posel, and D. Fleiszer, “Learning while having fun: The use of video gaming to teach geriatric house calls to medical students,” *Journal of the American Geriatrics Society*, vol. 56, no. 7, pp. 1328–1332, 2008.

[29] J. Kang and G. Seomun, “Evaluating web-based nursing education’s effects: A systematic review and meta-analysis,” *Western journal of nursing research*, vol. 40, no. 11, pp. 1677–1697, 2018.

[30] G. E. Favalora, J. Napoli, D. M. Hall, R. K. Dorval, M. Giovinco, M. J. Richmond, and W. S. Chun, “100-million-voxel volumetric display,” in *Cockpit Displays IX: Displays for Defense Applications*, vol. 4712, pp. 300–312, International Society for Optics and Photonics, 2002.

[31] G. R. Hofmann, “Who invented ray tracing?,” *The Visual Computer*, vol. 6, no. 3, pp. 120–124, 1990.

[32] The Blender Foundation, “Weight Paint - Introduction.” https://docs.blender.org/manual/en/latest/sculpt_paint/weight_paint/introduction.html (accessed: 2020-07-05).

[33] Oculus, “Gear VR.” https://www.oculus.com/gear-vr/ (accessed: 2019-23-10).

[34] Adam Savage's Tested, "SteamVR's "Lighthouse"." https://www.youtube.com/watch?v=xrsUMEbLtOs (accessed: 2020-21-05).

[35] Facebook, "Powered by AI: Oculus Insight." https://ai.facebook.com/blog/powered-by-ai-oculus-insight/, (accessed: 2020-18-04).

[36] Valve Corporation, "Steam search." https://store.steampowered.com/search/?vrsupport=402 (accessed: 2020-06-05).

[37] H. Baker, "Oculus Quest Store 2020 Stats: 170+ Apps And Strong Cross-Buy Support," https://uploadvr.com/oculus-quest-store-stats-2020/ (accessed: 2020-06-05).

[38] SideQuest, "Sidequest." https://sidequestvr.com/ (accessed: 2020-02-06).

[39] C. Marshall, "Your quest can now play oculus rift games, with the right cable and gaming pc." https://www.polygon.com/2019/11/18/20966968/oculus-link-open-beta-quest-rift-pc (accessed: 2020-02-06).

[40] M. Abrash, "Latency – the sine qua non of ar and vr," 2012. https://web.archive.org/web/20190822174539/https://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/.

[41] HTC Corporation, "Vive Pro." https://www.vive.com/eu/product/vive-pro/, (accessed: 2019-18-10).

[42] J. Peters, "Designing a HUD for a Third-Person VR Game." https://www.youtube.com/watch?v=f8an45s_-qs (accessed: 2020-21-05).

[43] Oculus, "VR Best Practices." https://developer.oculus.com/design/latest/concepts/book-bp/ (accessed: 2019-13-11).

[44] Unity Technologies, "VR Best Practice." https://learn.unity.com/tutorial/vr-best-practice (accessed: 2019-13-11).

[45] Khronos Group, "The OpenXR standard," Jul 2019. https://www.khronos.org/openxr/ (accessed: 2020-15-05).

[46] Unity Technologies, "Unity." https://unity.com/ (accessed: 2019-23-08).

[47] Unity Technologies, "Unity Asset Store." https://assetstore.unity.com/ (accessed: 2019-23-08).

[48] Valve Corporation, "SteamVR Unity Plugin." https://valvesoftware.github.io/steamvr_unity_plugin/ (accessed: 2020-21-05).

[49] Oculus, "Oculus Integration for Unity." https://developer.oculus.com/downloads/package/unity-integration/ (accessed: 2020-21-05).

[50] Unity Technologies, "Unity XR Interaction Toolkit." https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit\spacefactor\@m{}0.9/manual/index.html (accessed: 2020-26-03).

[51] D. H. Oma, "Mrsa - smitteverntiltak og testing," 2018. https://ek.helse-bergen.no/docs/pub/dok00512.pdf.

[52] Unity Technologies, "UnityEvents." https://docs.unity3d.com/Manual/UnityEvents.html (accessed: 2020-04-06).

[53] Unity Technologies, "ProBuilder." https://unity3d.com/unity/features/worldbuilding/probuilder (accessed: 2020-21-05).

[54] Unity Technologies, "Unity - Manual: Unity XR Input." https://docs.unity3d.com/Manual/xr_input.html#XRInputMappings (accessed: 2020-25-05).

[55] Valve Corporation, "The Lab." https://store.steampowered.com/app/450390/The_Lab/ (accessed: 2020-21-05).

[56] Valve Corporation, "Half-Life: Alyx." https://store.steampowered.com/app/546560/HalfLife_Alyx/ (accessed: 2020-21-05).

[57] M. E. Gjerde, "Dråpesmitteregime," 2020. https://ek.helse-bergen.no/docs/pub/DOK00499.pdf.

[58] P. E. Akselsen, "Beskyttelse av arbeidstøy og hud," 2017. https://ek.helse-bergen.no/docs/pub/dok18850.pdf.

[59] Reallusion, "Character Creator 3." https://www.reallusion.com/character-creator/ (accessed: 2020-22-03).

[60] Adobe Inc., "Mixamo." https://www.mixamo.com/ (accessed: 2020-22-03).

[61] Freesound, "Freesound." https://freesound.org/ (accessed: 2020-22-03).

[62] Oculus, "Testing and performance analysis." https://developer.oculus.com/documentation/unity/unity-perf/ (accessed: 2020-24-03).

[63] J. Brooke, "SUS - A quick and dirty usability scale," 1996.

[64] Oculus, "OVR Metrics Tool." https://developer.oculus.com/downloads/package/ovr-metrics-tool/ (accessed: 2020-26-03).